Solving staff rostering problems with column generation

Dominance cost functions, aggregate branching, neighbourhood pricing and other matheuristic improvements.

Isaac Cleland

A thesis submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy in Operations Research,
The University of Auckland, 2022.

Abstract

This thesis presents a methodology for modelling and solving generic staff rostering problems using column generation. We solve two difficult problem sets using column generation and demonstrate some novel exact and heuristic improvements to standard column generation, which are needed to solve these problem sets.

The first problem set we solved was from the International Nurse Rostering Competition (INRC). The INRC was held in 2010 to identify novel approaches to solve staff-rostering problems. Although many of the INRC problems were too tricky to solve to proven optimality, since the competition in 2010, multiple research groups have reported improved solutions to the problems compared to those proposed during the competition. However, up until this thesis, optimal solutions had not been found for 11 out of the 30 hardest INRC problem instances. In this thesis, we report how we obtained optimal solutions for the 30 hardest INRC problem instances within a 4-hour time window. Some of the identified solutions are of a higher quality than those obtained previously in the literature. We identified these solutions by implementing a series of improvements to Genie++, a nested column generation algorithm capable of solving staff-rostering problems. These improvements include a new dominance technique (dominance cost functions), new branching techniques, an objective function perturbation technique, and a shift aggregation relaxation.

The second set of problems were to create rosters for wards serviced by the Waikato District Health Board (DHB). The exact techniques used to solve the INRC problems were ineffective at finding solutions to the large and complex Waikato DHB problems. Thus, we developed some column-generation-based matheuristics to build high-quality rosters. We compare a suite of novel column-generation-based local search matheuristics for improving an incumbent roster solution. Our novel implementations of “fixed days" local search and “maximum shift changes per employee" local search were particularly effective at quickly improving an incumbent roster solution’s quality.

We also compare a suite of novel column-generation subproblem heuristics that we integrated into a branch and price dive to find high-quality roster solutions quickly. Our novel “LP neighbourhood pricing" methodology was especially effective at producing high-quality roster solutions quickly.

Many of the techniques developed in this thesis can be applied to other column generation problems, especially those with column generation subproblems which involve solving complex shortest path problems with resource constraints.

Acknowledgements

I would firstly like to thank my supervisors, Associate Professor Andrew Mason and Dr Michael O'Sullivan. Andrew's lofty vision for this project has always inspired me. And I have appreciated Mike's practical wisdom. I couldn't have asked for a better team guide me through these PhD years. I also want to acknowledge that both Andrew and Mike have consistently put aside time every week to meet with me and keep me on track throughout the entire PhD. I have really appreciated their dedication.

I want to thank my parents, Peter and Adrienne Cleland, for their belief in me. Over the years, I've had many struggles with this PhD. Both of you have always been there to encourage me and help me through any obstacle.

I want to thank my partner, Sunny Feng, for her unwavering support. She has always pushed me to perform my best, and without her, this PhD would never have reached the level that it did.

I want to thank my godmother, Andrea Craven. She always drove me to do a PhD and fulfil my potential, and I'm so thankful that I listened to her.

I want to thank Rachel Clarke from the Waikato DHB for her patience and support in providing data and helping us to model real-world nurse rostering problems.

Contents

Chapter 1

Introduction

1.1 The staff rostering problem

A Staff Rostering Problem is a problem in Operations Research that involves identifying feasible rosters of the highest quality for a group of employees while satisfying a set of constraints and requirements established by management and a set of roster-quality measures agreed upon by the employees. The roster requirements specify the number of employees required during different periods throughout the day. They also may determine the number of employees needed with particular skills or seniority. These periods are covered by a set of shifts that employees can work. A "shift" is defined as the period between an initial date and time and a final date and time. Each employee works a "roster-line", which is defined as a sequence of shifts worked and days off corresponding to that employee's work schedule.

Figure 1.1 presents an illustrative roster solution in a case comprising three employees. A roster-line is included for each employee over a 17-day roster. This example is modelled using three types of shifts (D, A, and N in Figure 1.1). Shifts of the same type are defined as having identical start and end times but are worked on different days.

This thesis's primary objective was to build a state-of-the-art generic system for modelling and solving challenging staff rostering problems automatically.

Even today, many staff rostering problems are solved manually. This is time-consuming and constrains the objective assessment of the quality of each roster. On the other hand, automatic roster generation saves time and usually yields better roster solutions that can be assessed in terms of predefined quality criteria. Therefore, several attempts have been made over the years to solve such problems using both exact and heuristic algorithms. However, despite the advantages, roster problems have proven challenging to solve automatically—they are typically NP-Hard (Karp, 2010), often involve a considerable number of variables, possess non-linear or multi-objective cost functions, and are tightly constrained. For a comprehensive overview of modern approaches to solve Staff Rostering Problems, please consult the extensive reviews by Van Den Bergh et al. (2013) on generic personnel scheduling, Ernst et al. (2004) on generic staff rostering and Burke et al. (2004a) and Cheang et al. (2003) on general nurse rostering.

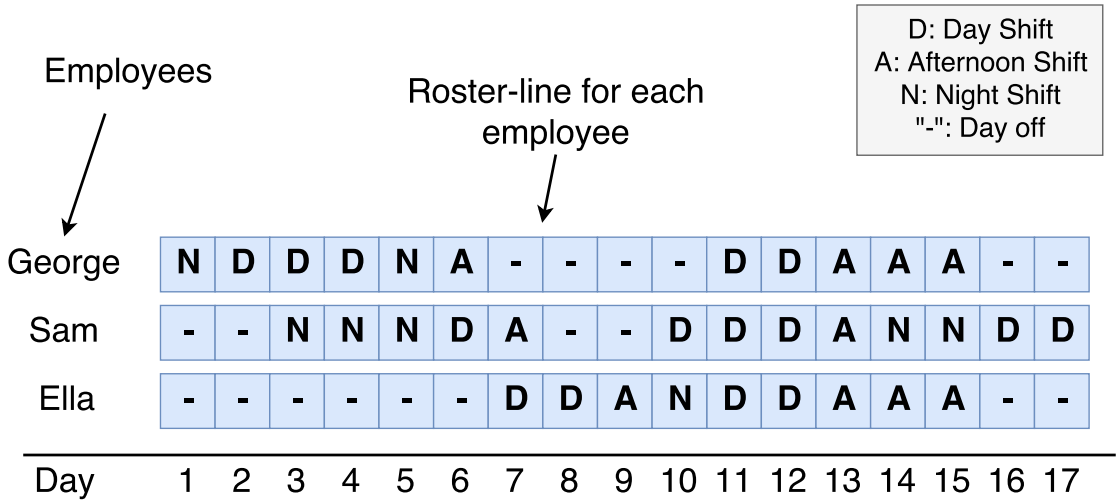


Figure 1.1: An example roster solution over 17 days with three employees.

Automated approaches to Staff Rostering Problems usually employ heuristics, e.g., variable neighbourhood search (Burke et al., 2004b, 2008), greedy neighbourhood search (Bellanti et al., 2004), scatter search (Burke et al., 2010a), ant colony optimization (Gutjahr and Rauner, 2007), bee colony optimization (Todorovic and Petrovic, 2013), and genetic algorithms (Aickelin and Dowsland, 2000). Other techniques have also been used such as constraint programming (Nonobe, 2010) and mixed integer programming (MIP) (Beaumont, 1997, Burke et al., 2010b, Santos et al., 2016, Mischek and Musliu, 2016), the latter often including the use of column generation which is the core algorithm discussed in our research; please see §3.2 for more details on the use of column generation to solve staff rostering problems in literature.

In this thesis, the software package, Genie++ (Mason and Smith, 1998, Dohn and Mason, 2013), is extended to address several difficult roster problems. Genie++ was initially developed to automatically solve a broad range of staff rostering problems using *column generation*. For an overview on column generation, see §2.2.

Dohn and Mason (2013) have already applied Genie++ to three large, complex, real-world staff rostering problems and have proven it effective in solving these problems to optimality in two cases and near optimality in one case within a 10 hour time window.

Our secondary objective for this thesis was to solve two challenging sets of staff rostering problems: the International Nurse Rostering Competition (INRC) problems which we introduce in §1.2 and the problems given to us by the Waikato DHB, which we first introduce in §1.3.

1.2 International Nurse Rostering Competition

In order to compare the performance of Genie++ with other modern staff rostering algorithms, we evaluated Genie++ on the nurse rostering problems developed for the International Nurse Rostering Competition (INRC).

Haspeslagh et al. (2010) developed this competition and the associated test instances. These problems serve as useful benchmarks, and several research groups have published solutions to these problems using a large variety of exact and heuristic methods. Further, before our research, optimal solutions to some of the INRC problems were yet to be identified (Rahimian et al., 2017a).

The quality of any roster solutions to an INRC problem instance is judged in terms of "cost", where smaller costs correspond to rosters of better quality. This metric is defined to be the sum of the penalties incurred by certain undesirable features of the roster. These undesirable features include the assignment of an employee to a shift that they requested to be exempted from; the assignment of an employee to a work-day that they requested to be exempted from; the assignment of a regular nurse to a head-nurse shift; and the assignment of an employee to work in contravention of the rules specified in their employment agreement. The only hard constraints are that an employee cannot be assigned more than one shift per day and that there cannot be more or fewer employees than mandated working any given shift. Each problem instance contains 2 to 4 different *contracts* that specify these rules for different subsets of employees. Employees of the same *contract* had identical rules. The only difference between these employees were preferences to work different shifts or have different days off. For an example of an INRC problem instance, please consult §4.3.1.

Although the INRC problem instances have been extensively investigated in previous studies and are considered to be important benchmarks for the evaluation of staff rostering techniques, they suffer from certain limitations compared to real-world scenarios. As all undesirable features are assigned similar penalties (between 1 and 10) within the INRC scenarios, any of the employee's contractual regulations are allowed to be violated within feasible, optimal roster solutions. This is not representative of real rostering problems, as most contractual regulations are strict and cannot be legally violated. Further, since the INRC problems involve only two hard constraints, they allow for the assignment of an employee to shifts on every successive day over a 28-day roster or to zero shifts over a 28-day roster. In one competition problem, the assignment of a head nurse to shifts on every successive day during the entire roster period produces a solution whose quality is within 20% of that of the best-known roster solution. Thus, these problems are not entirely realistic.

Another unrealistic aspect of these rostering problems is that they ignore all of the previous shifts which the employees have worked before starting the roster. For example, if an employee has just finished working six days consecutively and the new roster begins with the employee working six days consecutively, then the employee has effectively worked 12 days consecutively. However, despite these limitations, the INRC problem instances are still useful as a common benchmark for staff rostering techniques. Among existing techniques, the Genie++ extension presented in Chapter 5 is the first to prove optimality to all of the challenging INRC problems, and it did so in a reasonable amount of time (less than four hours).

The INRC problem instances can be classified into three categories: (i) 'sprints', which require the rostering of 10 nurses to 4 shift types distributed over a period of 4 weeks; (ii) 'mediums', which require the rostering of 30 nurses to 4 different shift types distributed over a period of 4 weeks; and (iii) 'longs', which require the rostering of 50 nurses to 5 different shift types

Instances	sprint	medium	long	total
early	~~10~~	5	5	20
late	~~10~~	5 (1†)	5 (1†)	20
hidden	~~10~~	5 (5†)	5 (4†)	20
total	~~30~~	15	15	60

Table 1.1: Summary of INRC problems—the problems which are yet to be solved to proven optimality are indicated by † and crossed-out entries denote problems not considered in this study.

distributed over a period of 4 weeks. The INRC competition comprised 60 problems in aggregate—30 sprint problems, 15 medium problems, and 15 long problems. The problems were further segmented such that one-third of the problems of each type corresponded to 'early' instances, one-third to 'late' instances, and one-third to 'hidden' instances. Table 1.1 summarises the aforementioned INRC segments.

Bulog (2011) has already modelled and solved a small subset of the problem instances using Genie++. However, they did not find optimal solutions to any of the long instances and did not model any of the late or hidden instances. As the late and hidden instances use an extended formulation with more constraints, they are significantly harder to solve.

Due to the reported easy nature of the sprint problems (Santos et al., 2016) and the fact that all of these problems have already been solved optimally, we did not utilize this set of 30 problems in our experiments (as indicated in Table 1.1). Out of the remaining 30 problem instances, eleven of the problem instances are known to exhibit gaps between the best-known lower bounds and the best-known solutions; see §3.3 for more details. Thus, there was still an opportunity for us to find better solutions than had been found previously.

1.3 Waikato District Health Board

In 2019, the NZ Ministry of Health (2019) reported that 58,206 nurses were working in NZ, which is a record high. These nurses were distributed among 19 different District Health Boards (DHBs) NZ-wide. We interviewed staff from Waikato DHB, Auckland DHB, Waitemata DHB and Christchurch DHB, and all of these staff stated that they were creating rosters manually in Excel. In some cases, these staff were charge nurses, who manage a single nurse ward, and in some cases, they were professional rostering or operational staff.

All NZ nurses are working under a single multi-employer collective agreement (MECA) negotiated by the NZ Nurses Organisation (2018), which dictates strict rules for how rostering staff can roster each nurse for a given month. Furthermore, NZ hospital culture demands a high level of fairness between individual nurses and the acknowledgement of specific individual preferences (Employment NZ, 2020). The complexity of the MECA, fairness constraints and individual preferences, along with the growing nurse workforce in NZ, means that rostering staff spend a great

deal of time creating rosters. From the details provided by the staff we interviewed, we estimate that rostering staff spend around three to four days per month rostering for a single ward.

Balancing all of these rules and preferences is also tricky, often leading to understaffed wards and overworked nurses. Martin and Kilmister (2018) report understaffed wards as a significant problem in New Zealand. Similarly, overstaffed wards represent a financial burden on a system already suffering constraints. We strive to reduce the burden on rosterers and District Health Boards (DHBs) in creating rosters. Furthermore, we wish to reduce the number of understaffed and overstaffed wards.

During 2019 and 2020, we have been cooperating with the professional rostering staff at Waikato DHB to automatically produce four-week rosters for an operating theatre ward and some maternity wards. All of the wards have a legal requirement to follow the NZ Nurse MECA. Waikato DHB also has its own rules and requirements for rostering. Lastly, we must consider all of the individual nurse requests and requirements. We provide a detailed example of how we modelled these rules related to each nurse in the Waikato DHB maternity wards problem in §4.3.2.

Many of the rules are related to fairness, and there was no formal definition provided to us for these rules by Waikato DHB. We interpreted fairness rules from our conversations with rosterers. The rules we built were later validated by the rosterers, who examined a number of our roster solutions. They also have not provided us with an exact objective function for evaluating rosters' quality. As such, we provided a Likert scale to describe the relative penalty for violating each rule accurately, i.e., a minimal penalty of 1, a small penalty of 10, a medium penalty of 100, a large penalty of 1000 or infeasible.

The process of creating an accurate model for each of their rostering problems was an iterative process. Each iteration involved attempting to model their rules given our conversations and then receiving feedback on that roster from the professional rostering staff. It took around four iterations until they were happy with the rosters being produced.

We are solving two different rostering problems for the Waikato DHB, the *operating theatre ward problem* and the *maternity wards problem*. There are two significant differences between the operating theatre ward problem and the maternity wards problem. The first is that we are solving for three maternity wards simultaneously, whereas we are only solving for a single operating theatre ward. Thus, we have to consider around 100 nurses[1] for the maternity wards problem and around 30 nurses for the operating theatre ward problem.

The second is that the maternity ward nurses can work both eight-hour and twelve-hour shifts, whereas the operating theatre ward nurses only work eight-hour shifts. Thus, the maternity wards nurses can work up to 11 different shift types, whereas the operating theatre ward nurses can only work up to five shift types. The much larger number of shift types greatly increases the difficulty of the rostering problem, which we discuss in §4.3.2. Because of these two differences, the problem of finding a high-quality roster for the maternity wards is much harder than finding a high-quality roster for the operating theatre ward, as we demonstrate in Chapters 6-7.

[1]Nurses were hired and left while we were rostering

Name	Description	Time	Fixed	Maternity only
AM	8 hour morning shift	7am to 3:30pm	✗	✗
PM	8 hour afternoon shift	3pm to 11:30pm	✗	✗
N	8 hour night shift	11pm to 7:30am	✗	✗
J	8 hour late morning shift	11am to 7:30pm	✗	✓
a	12 hour day shift	7am to 7.30pm	✗	✓
O	12 hour night shift	7pm to 7.30am	✗	✓
E	8 hour Education day	8am to 4:30pm	✗*	✗
e	8 hour Education day + work	8am to 7.30pm	✗	✓
t	12 hour orientation day	7am to 7.30pm	✓	✓
L	8 hours leave	8am to 4:30pm	✓	✗
l	12 hours leave	7am to 7.30pm	✓	✓

Table 1.2: Shift types provided to us for the maternity and operating theatre wards. The "fixed" column indicates that we can only assign this shift to an employee if the roster requirements specify that the shift has been preallocated to that employee. The "maternity only" column indicates whether the shift-type appears only in the maternity wards problem and not in the operating theatre ward problem. *The E shift was fixed for the operating theatre ward problem but was not fixed for the maternity wards problem. This was because the solver could choose between an 'E' shift and an 'e' shift for a given maternity nurse if an education day was preallocated for that day.

Name	Time
AM period	From 7am to 3pm
PM1 period	From 3pm to 7:15pm
PM2 period	From 7:15pm to 11pm
N period	From 11pm to 7am

Table 1.3: Four periods of the day, each of which has requirements on the number of nurses that need to be working.

Although the MECA does not enforce any particular start or end time for shifts, Waikato DHB has provided us with a set of standard shifts predesigned from the rules outlined in the MECA. For a list of these different shift types, see Table 1.2. Note that no 12-hour shifts are worked in the operating theatre ward.

We divided up the day into four periods covered by the shifts. Please refer to Table 1.3 for a list of these periods. For each ward, there is a different minimum, target and maximum total number of nurses that need to work during a given period on each day.

Further, for each ward, there is also a minimum, target and maximum number of nurses with specific qualifications or skills that need to work during a given period on each day. There are five different types of nurses at Waikato DHB. Each nurse is a head nurse, intermediate nurse,

Nurses	Minimum	Target	Maximum
All	5	6	8
Head nurses	0	1	n/a
Head nurses and intermediate nurses	1	2	n/a
Nurses with coordinator skill	1	1	n/a
Student nurses	0	1	1
Student nurses and enrolled nurses	0	1	2

Table 1.4: The number of nurses we require with certain qualifications and skills from 7 am to 3 pm on a Wednesday. We need a sufficient amount of senior nurses (head and intermediate) and nurses with a coordinator skill, but we don't want too many student nurses or enrolled nurses.

regular nurse, enrolled nurse or student nurse skill. Some head nurses and intermediate nurses also have the coordinator skill. In Table 1.4, we provide some example data for the number of nurses we require from 7 am to 3 pm on a Wednesday.

Each ward has an insufficient number of nurses to meet the target number of nurses working in every period. We have hard constraints to at least ensure we meet all the minimum nurse requirements. To encourage the model to distribute the nurses evenly, we give an exponential penalty cost for the number of nurses below our target for a given period. For example, having one fewer nurse incurs a minimal penalty; and having two fewer nurses incurs a large penalty within our MIP. We implement these penalties using slack and surplus variables; see §4.1.1.

1.4 Thesis outline

In Chapter 3, we present a summary of the best performing algorithms in literature for solving the INRC problems as well as a comprehensive review on how researchers have used column generation to solve staff rostering problems. In Chapter 4, we present a generic method for modelling and solving staff rostering problems with column generation. We then use this method to model an example INRC problem and an example maternity wards problem. In Chapter 5, we discuss how we found proven optimal solutions to every difficult problem in the INRC. In Chapters 6-7, we discuss how we found high-quality rosters for the Waikato DHB using column generation based matheuristics. In Chapter 6, we discuss the use of neighbourhood restricted branch and price to perform local search and improve the quality of a roster solution. Then in Chapter 7, we discuss the use of column generation subproblem heuristics to enhance a branch-and-price dive and find good rosters quickly. Lastly, in Chapter 8, we conclude with an outline of our achievements and results.

1.5 Contribution

As part of this work, we have provided a generic mathematical description of nested shortest path problems with resource constraints for staff rostering problems; see §4.2 for details. We have provided two example models using our modelling language, a model for an INRC problem and a model for a Waikato DHB maternity wards problem.

Further, we have designed a series of novel improvements to standard column generation methods for solving staff rostering problems, which have been necessary to find proven optimal solutions to all of the difficult INRC problem instances. To this end, the following improvements have been made: a new dominance technique in §5.2.1 which we call 'dominance cost functions', an objective function perturbation in §5.2.2, new branching techniques in §5.3 and a shift aggregation technique in §5.4.

Our improved Genie++ algorithm could not solve the Waikato DHB problems to find a sufficiently high-quality solution in a reasonable time. Thus, we have developed a suite of column generation based matheuristics and compared their effectiveness in solving the operating theatre and maternity wards problems. With these techniques, we are able to find high-quality roster solutions quickly.

The first set of matheuristics involves a series of local search matheuristics using column generation. Except for "fixed employees" neighbourhood and "maximum shift changes per employee" neighbourhood, we believe all of these neighbourhood restricted column generation matheuristics to be novel within the field of staff rostering. See Chapter 6 for more details.

The second set of matheuristics involves a series of novel column generation subproblem heuristics, including entity restriction and neighbourhood pricing, to improve the speed of finding negative reduced cost columns in the column generation subproblem. Except for "entity restriction by cost", we believe all of these column generation based local search heuristics to be novel within the field of staff rostering. See Chapter 7 for more details.

Within the literature for staff rostering problems, we also found no other comprehensive comparison of the effectiveness of multiple column-generation-based matheuristics for solving staff rostering problems.

We established the novelty of each of our techniques by systematically reviewing every paper on using column generation to solve staff rostering problems. For a detailed explanation of how we performed this systematic review, refer to §3.2.

Chapter 2

Background

This chapter provides a comprehensive background on staff rostering problems, the core concepts of column generation and some of the local search techniques we will be employing to solve staff rostering problems.

2.1 Staff rostering problems

This thesis considers the "staff rostering problem", which we initially define in §1.1. Our goal is to solve a broad range of staff rostering problems, but we still make two core assumptions for the type of staff rostering problems that Genie++ can solve. The first is that the quality of a staff roster can be measured by features that are either embedded in an individual's roster-line or associated with the number of staff of a particular type working a shift. We do not, for example, handle staff who want to work together on four days of the fortnight. The second is that we can define each employee's roster-line or line of work as a series of *activities*, one for each day, where an activity is a shift or a day off. This definition for staff rostering problems, therefore, precludes other problems such as crew rostering, where staff are instead assigned crew pairings, which are sequences of flights that start from and end at the same location. We could extend these algorithms to apply to problems in which more than one activity is started per day; however, we opted out of this as it was unnecessary for the problems we were attempting to solve and would introduce code inefficiencies.

Instead of representing a roster-line purely as a set of activities, we can segment a roster-line into several component *entities*, where an *entity* represents a subset of the roster-line's activities over a series of consecutive days. Using the terminology of Mason and Smith (1998), we define six different entities as part of our *entity model*. A roster-line is made of a series of consecutive *work-stretches*, each of which contains a single *on-stretch* followed by a single *off-stretch*. An on-stretch is a series of consecutive days worked and is commonly also referred to in the literature as a stint (Glass and Knight, 2010) or rotation (Legrain et al., 2020). An off-stretch is a series of consecutive days off. A full diagram of how the component entities build into a roster-line can be seen in Figure 2.1. For a mathematical description of our entity model, see §4.2.

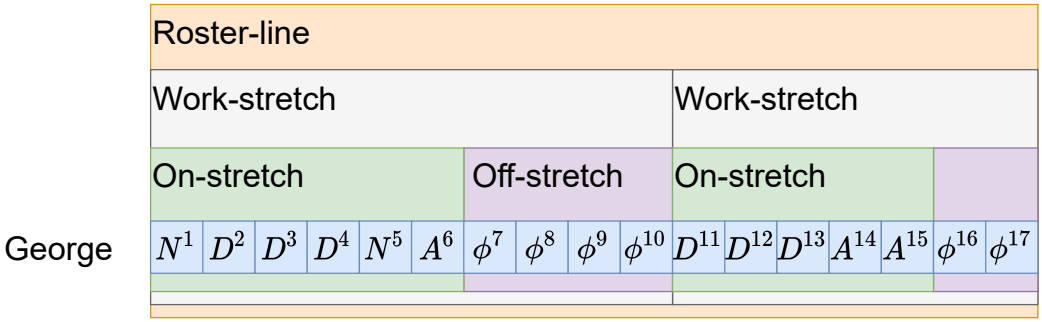


Figure 2.1: An example of our entity model. Here, we show a roster-line for the employee, George, with two work-stretch components. Each work-stretch has an on-stretch and an off-stretch component. Each on-stretch has a series of consecutive shifts (D, A, N) as components, and each off-stretch has a series of consecutive off-days (ϕ) as components.

We break down a roster-line into component entities as it can help to model rules more easily and construct roster-lines more efficiently; refer to the work of Dohn and Mason (2013) for more details on the effectiveness of decomposing a roster-line into component entities. Because we model a roster-line as a set of component entities, we refer to our column generation algorithm as being *nested*.

We define the span of any entity as the longest set of consecutive days for which the entity defines activities. We define a roster-line that starts on the first day and ends on the last day of our staff rostering problem as a *full roster-line*. We also define a roster-line that starts on the first day and ends on any other day as a *partial roster-line*.

We define a rostering rule as describing the criteria by which entities are deemed infeasible or have an associated penalty, given the set of activities worked in that entity or the span of that entity. For example, a rostering rule might describe a penalty for any entity which contains a night shift followed by a morning shift. Another example is a rostering rule that deems any on-stretch infeasible if it has too long a span.

As well as fulfilling a set of rostering rules for each employee, we also need to meet certain requirements on the number of employees who need to work during certain periods of the day. We even sometimes need to meet requirements on the number of employees with a specific skill or seniority. We call these requirements *demands*.

We note that a single shift can contribute to multiple demands, and a single demand can be contributed to by multiple shifts. For example, in the Waikato DHB maternity wards problem, we have a demand for a minimum number of head nurses working from 7:15 pm to 11:00 pm on day 5. Both the PM shift and the 12-hour day shift on day 5 cover this period, and thus, if a head nurse works one of these shifts, they cover this demand.

We can represent any roster-line as a set of shifts worked or a set of demands contributed to as we can easily convert between the two representations.

2.2 Column generation

The classic formulation of a staff rostering problem as a mixed-integer program involves modelling the rostering problem using employee-shift variables, i.e., x_{eS} is equal to one if employee e works shift S and zero otherwise; for an example of this formulation, see Santos et al. (2016). Another formulation involves modelling the rostering problem as a generalised set partitioning problem with roster-line variables, i.e., λ_e^r is equal to one if employee e works roster-line r and zero otherwise; for an example of this formulation, see Burke and Curtois (2014). In the second formulation, many of the rules are modelled implicitly. The existence of a given roster-line in the formulation implies it is a legal roster-line and the cost of the roster-line implies violations of soft constraints. In this thesis, we model rostering problems using the second formulation.

Formulating a rostering problem with roster-line variables tends to be a stronger formulation than with employee-shift variables. This is in the sense that the gap between the solution to the *root node* of the search tree, solved with no branching constraints, and the optimal integer solution is smaller using a formulation with roster-line variables than a formulation with employee-shift variables (we compare these two formulations in §2.2.1). We provide some evidence of this in §3.3 where both formulations have been used to solve INRC problems, allowing their root node objective function values to be compared.

Because there are a very large number of legal roster-lines for a given employee, we use column generation. Column generation is an approach used to efficiently solve large linear and integer programs involving a substantial number of variables. Instead of solving the entire linear program (LP), this method creates a restricted LP master problem by considering only a subset of the variables (columns). Then, by using the Simplex Algorithm (Dantzig, 1963) to solve this linear program and produce a dual solution, we can generate negative reduced-cost variables with a separate *column generation subproblem* or *pricing problem* and add these variables to the restricted LP. This LP is then resolved, and the process is repeated until no more negative reduced cost columns can be generated. The final solution is an optimal one for the complete LP with all possible variables. Column generation can be extended to solve mixed-integer programs (MIPs) with the use of a branch-and-bound tree; this is commonly known as branch-and-price (Barnhart et al., 1998). For further information on column generation, consult the comprehensive reviews of column generation by Desaulniers et al. (2005) and Lübbecke (2010).

Using the language of Desrosiers and Lübbecke (2005), we often refer to the formulation of the staff rostering problem with employee-shift variables as a "compact" formulation and the equivalent roster-line formulation as an "extensive" formulation.

2.2.1 Dantzig-Wolfe decomposition

Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960) involves the reformulation of a compact LP into a restricted master problem and n subproblems. For staff rostering problems with a compact formulation, the reformulation typically gives rise to the extensive formulation as described above.

The modelling approach described in this thesis of using column generation and constraint branching was developed by Ryan and Foster (1981) independently of a Dantzig-Wolfe formulation. With Ryan and Foster's approach, we do not formulate our staff rostering problem with a compact formulation and instead skip straight to modelling the staff rostering problem as a generalised set partitioning problem with roster-line variables. Ryan and Foster saw the value of column generation as a modelling tool, not just in the strengthening of the formulation but also the ability to model otherwise complex constraints in the column structure. Even though we do not directly use Dantzig-Wolfe decomposition, we still provide a brief overview of how our formulation could be derived using Dantzig-Wolfe decomposition, albeit with arbitrary non-linear constraints.

Dantzig-Wolfe decomposition is a method of solving linear programs with a special structure. In this special structure, a subset of the constraints are "coupling" in that they have non-zero coefficients for a large number of variables; the remainder of the constraints can be divided into a number of independent groups where if a variable has a non-zero coefficient in one group, then it must have a zero coefficient in every other group. Within the context of staff rostering, each independent group corresponds to the constraints placed upon each individual employee in order to model their rostering rules. The coupling constraints correspond to the *demands* for a specified minimum or maximum number of employees to work during certain times throughout the day. We model each of these requirements using a *demand* $D \in \mathcal{D}$ where $\mathcal{D} = \{D_1, D_2, \ldots, D_{|\mathcal{D}|}\}$ is the set of all demands.

We show a representation of a generalised, compact LP for a staff rostering problem in (2.1)-(2.4). As we skip straight to modelling the staff rostering problem as a generalised set partitioning problem, we have created the compact formulation with our actual formulation in mind. In this LP, the set of employee indices is given by $\mathcal{E} = \{1, 2, \ldots, |\mathcal{E}|\}$, where $|\mathcal{E}|$ denotes the total number of employees. The variable x_{eD} is one if employee e is contributing to demand D and zero otherwise. The vector $\mathbf{x}_e = (x_{eD_1}, x_{eD_2}, \ldots, x_{eD_{|\mathcal{D}|}})$ represents a roster-line for a single employee. Note that because there is a one-to-one mapping between roster-lines represented by shifts and roster-lines represented by demands contributed to, a roster-line can be represented as a set of employee-shift variables or a set of employee-demand variables in a compact formulation. In this example, we use employee-demand variables because it is more straightforward to reformulate into our actual formulation.

The function $f_e(\mathbf{x}_e)$ represents the sum of penalties for violating certain rostering constraints in employee e's roster-line $\mathbf{x}_e$. The constraint set (2.2) is a set of coupling constraints that ensure we are meeting each rostering demand b_D. The constraint set (2.3) is a set of independent constraints H_e for each employee $e \in \mathcal{E}$ which enforce rostering constraints for an individual employee and the mapping between working shifts and contributing to demands.

$$\text{Compact LP:} \quad \text{Minimize} \quad \sum_{e\in\mathcal{E}} f_e(\mathbf{x}_e) \tag{2.1}$$

$$\text{s.t.} \quad \sum_{e\in\mathcal{E}} x_{eD} = b_D \qquad \forall D \in \mathcal{D} \quad [\pi_D] \tag{2.2}$$

$$g_{e\eta}(\mathbf{x}_e) = 0 \qquad \forall \eta \in H_e,\ \forall e \in \mathcal{E} \tag{2.3}$$

$$0 \le x_{eD} \le 1, \qquad \forall e \in \mathcal{E}, D \in \mathcal{D} \tag{2.4}$$

We can use Dantzig-Wolfe decomposition to reformulate our compact LP into a master program and $|\mathcal{E}|$ subproblems. This reformulation relies on the fact that every point of a non-empty, bounded convex polyhedron can be represented as a convex combination of its extreme points. Minkowski and Weyl's theorem (Schrijver, 1998) says that any polyhedron can be represented as a convex combination of its extreme points and rays. However, there are no rays because in staff rostering problems all the variables (x_{eD}) in our compact formulation are bounded. Therefore, we represent the set of feasible variables x_{eD} in our compact formulation for employee e that satisfy constraint sets (2.3) and (2.4) as a convex combination of the set of extreme points for that employee e, with indices $\mathcal{R}_e = \{1, 2, \ldots, |\mathcal{R}_e|\}$, i.e.,

$$x_{eD} = \sum_{r\in\mathcal{R}_e} a^r_{eD}\lambda^r_e \tag{2.5}$$

$$\text{where} \sum_{r\in\mathcal{R}_e} \lambda^r_e = 1 \quad \forall e \in \mathcal{E} \tag{2.6}$$

$$0 \le \lambda^r_e \le 1, \qquad \forall e \in \mathcal{E}, r \in \mathcal{R}_e \tag{2.7}$$

where we ensure each extreme point $\mathbf{a}^r_e = (a^r_{eD_1}, a^r_{eD_2}, \ldots, a^r_{eD_{|\mathcal{D}|}})$ for employee $e \in \mathcal{E}$ satisfies the constraints $g_{e\eta}(\mathbf{x}_e) = 0\ \forall \eta \in H_e$ from the compact formulation and λ^r_e is the scalar for extreme point r of employee e.

In our problem reformulation, each extreme point is a column in our extensive LP shown in (2.8)-(2.10). Column r for employee e represents a roster-line, $\hat{R}^r_e$, generated for employee e. The cost of column r for employee e, $r \in \mathcal{R}_e$, is given by $c^r_e = f_e(\mathbf{a}^r_e)$. The cost of a given column indicates its quality, where a lower cost indicates a higher-quality roster-line. The decision variable, λ^r_e, is equal to one if column $r \in \mathcal{R}_e$ for employee e lies in the optimal roster solution and zero otherwise. The coefficient a^r_{eD} is one if employee e works a shift which contributes to demand D in column r.

$$\text{'Extensive LP:} \quad \text{Minimize} \quad \sum_{e\in\mathcal{E}}\sum_{r\in\mathcal{R}_e} c^r_e\lambda^r_e \tag{2.8}$$

$$\text{s.t.} \quad \sum_{e\in\mathcal{E}}\sum_{r\in\mathcal{R}_e} a^r_{eD}\lambda^r_e = b_D \qquad \forall D \in \mathcal{D} \quad [\pi_D] \tag{2.9}$$

$$\sum_{r\in\mathcal{R}_e} \lambda^r_e = 1 \qquad \forall e \in \mathcal{E} \quad [\pi_e] \tag{2.10}$$

$$0 \le \lambda^r_e \le 1, \qquad \forall e \in \mathcal{E}, r \in \mathcal{R}_e \tag{2.11}$$

The number of possible columns or roster-lines is too large to enumerate every single possible roster-line and solve this master problem as a whole with an LP solver. Thus, we instead work with a significant subset of the possible columns, forming the *restricted master problem*. More columns are instead generated as needed using a *subproblem*. Like the simplex method, in every iteration of column generation, we find a new column or set of columns to enter the basis each with a negative reduced cost. Thus, we need to define a subproblem that finds entering columns with the lowest reduced cost for each employee $e \in \mathcal{E}$. As we need the duals for each row, the restricted master problem needs to also be solved each iteration to find these duals.

We define the subproblem in (2.12)-(2.14). This subproblem is also referred to as the pricing problem. The reduced cost for an entering column $\mathbf{a}_e = (a_{eD_1}, a_{eD_2}, \ldots, a_{eD_{|\mathcal{D}|}})$ for employee e is given by $f_e(\mathbf{a}_e) - \pi_e - \sum_{D\in\mathcal{D}} \pi_D a_{eD}$, where π_D is the dual for row D in constraint set (2.9) and π_e is the dual for row e in constraint set (2.10). We find the lowest reduced cost column by minimising over the vector $\mathbf{a}_e$. If the reduced cost of that column is less than 0, we add that column to the restricted master problem as $\mathbf{a}_e^r$ for some appropriate r.

$$\text{Subproblem } (e): \quad \underset{a_{eD}\ D\in\mathcal{D}}{\text{Minimize}} \quad f_e(\mathbf{a}_e) - \pi_e - \sum_{D\in\mathcal{D}} \pi_D a_{eD} \tag{2.12}$$

$$\text{s.t.} \quad g_{e\eta}(\mathbf{a}_e) = 0 \qquad \forall \eta \in H_e \tag{2.13}$$

$$a_{eD} \in 0, 1, \qquad \forall e \in \mathcal{E},\ D \in \mathcal{D} \tag{2.14}$$

2.2.2 Pricing problem

The two most common ways to model the pricing problem are as a MIP or as a *shortest path problem with resource constraints* (SPPRC) as defined by Irnich and Desaulniers (2005). The MIP is typically solved using a standard MIP solver. The SPPRC is modelled as a digraph and typically solved using dynamic programming, using either a label setting or a label correcting algorithm (Irnich and Desaulniers, 2005). Since we have no cycles in our graph, we use a label setting algorithm, which we detail in §4.4.

When solving an SPPRC, the label represents a vector of resources or *resource vector*. We calculate how each resource varies along a path through *resource extension functions (REFs)* which define how the resource vector changes when the label is extended along an arc. The resources also have associated hard and soft constraints. Genie++ represents all employee rostering rules using resources, resource extension functions, and resource constraints. As we model our rostering rules in an SPPRC within the pricing problem, we do not need to model them as linear constraints as is typical within a MIP formulation. We can instead model all rostering rules with generic code

Genie++ defines a set of resources, resource extension functions, and resource constraints using generic C code. The resource extension functions and resource constraints are arbitrary functions. These functions form the basic building blocks of our modelling approach and take the place of what might normally be expressed with linear or non-linear functions and decision

variables in a linear program. Because these functions are the building blocks of our model, the presentation in this thesis is given in terms of these arbitrary functions rather than a set of linear constraints and decision variables.

These arbitrary functions are our model's building blocks and differ from problem to problem, as different problems have different structures. The core aspect of our modelling approach is that these functions exist, and despite their arbitrary complexity, we can implement them in a column generator.

We refer to the *state-space* in our label setting algorithm as the number of possible labels which we need to generate in order to solve our pricing problem. When solving staff rostering problems with many resources and over a long span, the state-space can become large, so it is standard practice to apply additional strategies to prune labels such as dominance and heuristic techniques to reduce the number of labels (Irnich and Desaulniers, 2005).

2.2.3 Branch-and-price

We require an integer solution to the staff rostering problem, i.e., each staff member must be assigned exactly one roster-line. Thus, we use *branch-and-price* (Barnhart et al., 1998) which is a *branch-and-bound* method for solving *mixed integer programs* (MIPs) where each node in the *search tree* is solved using column generation.

In branch-and-bound, the entire solution space is explored when solving the *root node*, then *branches* of the root node are explored, which form a subset of the solution space represented by further nodes of which can, in turn, be branched on. The corresponding branches and nodes are referred to as a search tree.

Branching is performed to remove non-integer solutions from the solution space while retaining all possible integer solutions. Since we are solving a minimisation problem, the best-known integer solution found by solving any node in the search tree represents an upper bound on the quality of the solution, also known as the primal bound. The lowest possible solution in the solution space found by solving any set of LPs that contain the entire integer problem space forms a lower bound on the quality of the solution, also known as the dual bound.

Creating two branches from each node in the binary search tree requires a *branching rule*. A branching rule typically involves a set of integer variables to branch on and a method for selecting the particular variable to be branched on. Instead of branching on variables, we use constraint branching by Ryan and Foster (1981). In the context of staff rostering, this typically involves branching on an employee-shift pair, (e, S), i.e., the left branch enforces that employee e must work shift S and the right branch enforces that employee e must not work shift S. Branching on these pairs is effectively branching on an employee-shift variables from a typical compact formulation.

A search strategy involves deciding which node in the branch-and-bound search tree to solve next. Some common search strategies include depth-first search, which is a LIFO queue-based implementation and best-first search, which is a priority queue that prioritises nodes in ascending order based on their lower bound (Clausen, 1999), both of which are used in this thesis.

2.2.4 Matheuristics

In order to find high-quality roster solutions more quickly, we can combine our *mathematical programming* method with a *metaheuristic*. This combination is known as a matheuristic (Maniezzo et al., 2021).

A popular metaheuristic is *variable neighbourhood search* (Hansen et al., 2016) where multiple *neighbourhoods* of an incumbent solution are explored. This technique proves useful as local minima for one neighbourhood are not necessarily local minima for another neighbourhood. A subclass of variable neighbourhood search which involves iteratively increasing the size of a neighbourhood is called *variable neighbourhood descent* (Hansen et al., 2016).

We can use variable neighbourhood search in combination with column generation as a matheuristic in two ways. The first way is generating new rosters more quickly by generating them in the neighbourhood of previous rosters. The second way is generating new roster-lines more quickly by generating them in the neighbourhood of old roster-lines.

2.3 Chapter summary

In §2.1, we have carefully defined the staff rostering problem we will be solving in this thesis and our entity model for defining roster-lines. In §2.2, we explained how Dantzig-Wolfe decomposition, which is one theoretical basis for column generation, fits in with our approach. We also describe the core concepts of column generation, including the pricing problem and branch-and-price. Lastly, we cover how column generation can potentially be sped up using matheuristics.

Chapter 3

Literature Review

This chapter explores two segments of the staff rostering problem literature. In §3.1, we introduce common usages of column generation in literature for solving complex integer programs. In §3.2, we perform a comprehensive literature search on solving staff rostering problems with column generation with a particular focus on modelling and acceleration strategies. In §3.3, we introduce some of the most successful attempts at solving the International Nurse Rostering Competition (INRC) problems.

3.1 Column generation

Column generation has typically been used to solve a wide range of complex integer linear programs including broader personnel scheduling problems (Van Den Bergh et al., 2013, Ernst et al., 2004, Burke et al., 2004a, Cheang et al., 2003), airline crew scheduling problems (Gopalakrishnan and Johnson, 2005, Heil et al., 2020), and vehicle routing problems (Desaulniers et al., 2011). In this section, we briefly discuss the use of column generation in solving crew scheduling and vehicle routing problems. We then provide a comprehensive review of literature on staff rostering problems solved using column generation. As our focus is on extending an existing column generation framework for solving staff rostering problems. We are primarily focused on literature eminently relevant to this topic.

We first define the airline crew scheduling problem. Airline crew scheduling is usually broken down into the airline crew pairing and airline crew rostering problems (Quesnel et al., 2020). The airline crew pairing problem (Aggarwal et al., 2020, Desaulniers et al., 2020, Tahir et al., 2019, Zeren and Özkol, 2016, Muter et al., 2013) involves constructing a set of crew pairings, each of which is a legal set of flights starting from a crew base and ending at the same crew base. Within our staff rostering terminology, the airline crew pairing problem is similar to building on-stretches from tasks with precedence constraints (see §3.2.1 for more details on tasks). Crew pairs are usually constructed over one or a few days.

The airline crew rostering or airline crew assignment problem (Kasirzadeh et al., 2017, Boubaker et al., 2010, Zeighami and Soumis, 2019, Maenhout and Vanhoucke, 2010b) involves

assigning each crew pairing to a staff member. Airline crew rostering also involves considering employee preferences, requests, rest periods, training and annual leave. Within staff rostering terminology, the airline crew rostering problem is similar to building roster-lines from on-stretches. Crew rosters are usually constructed over many days.

The vehicle routing problem (VRP) (Desaulniers et al., 2011, Dabia et al., 2013, Gutiérrez-Jarpa et al., 2010) involves assigning routes to vehicles to deliver goods to a number of customers. Typically vehicle routes are constructed on a single day.

Personnel scheduling problems, airline crew scheduling problems, and vehicle routing problems are all similar because they are fundamentally set covering and set partitioning problems. Each of these problems involves choosing a set of activities or destinations which need to be covered by a set of people or vehicles. They also both frequently involve the use of a Shortest Path Problem with Resource Constraints (SPPRC) to solve their column generation subproblem. In preparing our literature survey, we examined papers that use SPPRCs to solve column generation subproblems. SPPRCs are useful for modelling a large variety of linear and non-linear constraints.

Both staff rostering and airline crew rostering problems usually have a separate column generation subproblem for each staff member. This is because these problems are sensitive to the wants and needs of each staff member. However, crew pairings and vehicle routes are usually anonymous, and so these two problems usually only have a single column generation subproblem.

In the papers which involve using an SPPRC to solve the column generation subproblem, staff rostering problems, crew pairing problems and crew rostering problems all considered many different resources. Crew pairings often involve many legal requirements, and crew rostering problems involve contractual and legal requirements as well as preferences and requests from each staff member to consider. However, VRPs usually had few resources in their SPPRCs. VRPs typically focus on a day's worth of activities for a vehicle. Thus, we don't need to embed as many qualities in the SPPRC. For example, a standard vehicle routing problem with time windows, as described by Desaulniers (2010), only has three resources: time, vehicle load, and whether a customer has been visited.

As discussed, there are significant similarities between staff rostering, and crew scheduling. Although we consider notable crew scheduling literature, we only performed a systematic literature review on the staff rostering problem.

3.2 Column generation to solve staff rostering problems

The staff rostering problem has multiple aliases within the literature. Many researchers refer to the staff rostering problem as the nurse rostering problem, even though it applies to other staff types. Other forms include the staff scheduling problem and the personnel scheduling problem. Further, column generation is regularly referred to as both "column generation" and "branch and price" within the literature. To emphasise the use of column generation in our search, we only reviewed papers that specifically mention column generation or branch and price in the title.

Hence, we performed a search in Google Scholar with the following search terms:

> "staff rostering" OR "nurse rostering" OR "staff scheduling" OR "personnel scheduling" intitle: "column generation" OR intitle: "branch and price"

This search produced 101 results. Of these 101 results, 28 were either duplicates or inaccessible. All pertinent features of the remaining 73 papers are summarised in §3.2.1-§3.2.7.

56 of the 73 papers were published in 2010 or later, and 36 of the 73 papers were published in 2015 or later. This indicates that using column generation to solve staff rostering problems is becoming more popular. Thus, we believed there were opportunities to make significant contributions to this research field and explored the identified literature for potential opportunities.

3.2.1 Problem types

We broadly define a staff rostering problem within the literature as assigning shifts or activities to a set of employees over multiple days. Out of our 73 results, 54 of the papers were staff rostering problems according to this broad definition. However, these 54 problems still had significant variations in the exact problem being solved. This section classifies the 54 papers on staff rostering problems according to five prominent features.

According to the review of staff rostering by Ernst et al. (2004), there are six possible components to the staff rostering process: demand modelling, days-off scheduling, shift scheduling, line of work construction, task assignment, and staff assignment. Many researchers within the literature used this set of components to classify staff rostering problems. However, we found that column generation is generally used in literature to solve problems that integrate most of these components to various extents. Thus, we have extracted some of the prominent differences between the different staff rostering problems solved using column generation.

One complexity of solving staff rostering problems comes from many complex rules with soft constraints. Maenhout and Vanhoucke (2010a) defined the "multiple objective nurse scheduling problem" as one which not only involves minimising under-staffing and over-staffing but also seeks to include various types of nurse preferences, nurse special requests and measurements of fairness. We found that explicitly modelling fairness with a complex cost structure dramatically increases the difficulty of solving the problem; see §4.3.2 for more details. Allowing for preferences and special requests also means that there must be a separate column generation subproblem for each employee, leading to symmetry if the employees are mostly the same; see §5.2.2 for more details. Thus, our first two classification features are whether fairness is modelled and whether requests and preferences are modelled.

Many researchers increased their staff rostering problem's complexity with the explicit modelling of tasks/activities. There is some confusion within the literature over the differences between a task and an activity. For example, Boyer et al. (2014) referred to tasks and activities as separate and distinct entities. In their definition, tasks are small, have sequence constraints and have hard time constraints. Further, activities are daily operations and have penalties for undercoverage. Conversely, Gérard et al. (2016) considered a task as a time interval where a single activity is worked. For the remainder of this section, we refer to tasks as worked components of a

shift. Thus, our third classification feature is whether shifts are constructed from multiple tasks within their column generation algorithm.

Some researchers further complicated task assignment by giving the tasks precedence constraints, i.e., a given task must be worked before another task. In some cases, these tasks were components of multiple separate projects. Thus, our fourth classification feature is task precedence constraints.

Our final classification feature includes solving cyclic rostering problems, which involves adding additional constraints to the rostering problem.

Table 3.1 provides a summary of all 54 papers in terms of the five classification features. This table also includes the two problem sets modelled in this work: the INRC problems and the Waikato DHB problems.

The INRC problems are a relatively standard staff rostering problem within the literature with the only complex feature of employee preferences. This is a well-studied class of problem as many of the problems in literature have specific employee preferences and requirements (31/54). However, by being the first to solve each instance to proven optimality, we have demonstrated that the techniques outlined in this work are a new contribution even to this well-studied class of problem.

In contrast, the Waikato DHB problems are relatively unique within the literature due to the explicit modelling of fairness with a complex, non-linear objective function.

We only found two explicit mentions of modelling fairness (2/54) in the literature. Dohn and Mason (2013) mentioned that "a sophisticated cost structure applies that distributes weekend shifts fairly and distributes off-days evenly over the weeks. This cost structure makes it very hard to find near-optimal solutions." Although Bard and Purnomo (2005b) did not explicitly model fairness within a single rostering problem, they did track the number of preference violations for a given nurse and attempted to provide that nurse with a better schedule in their next roster. This was to achieve fairness over the long term.

We did not choose to solve such a uniquely difficult problem but were provided with this problem by the Waikato DHB.

Solving an integrated staff scheduling problem, involving the construction of shifts from tasks and roster-lines from shifts, is a common use of column generation. We found that 18 of the 54 papers explicitly constructed shifts from tasks. Six of those 18 papers had precedence constraints on those tasks. Although we did not directly model the construction of shifts from tasks, we believe that our generic formulation and acceleration strategies that provided proven optimality for the INRC problems (i.e., problems with only employee preferences) could easily be modified to include the modelling of tasks. We would consider a task as a separate entity and construct a set of feasible shifts using a separate SPPRC in our nested column generation subproblem, similar to Gérard et al. (2016). Hence, the techniques outlined in this work also have the potential for the feature combination of tasks and employee preferences.

It is also worth noting that very few approaches considered both tasks and employee preferences (5/54), and only one research group considered tasks, task precedence and employee

preferences (1/54). Thus, we believe integrating tasks in our already uniquely difficult problem is unnecessary to demonstrate the effectiveness of our column generation algorithm.

We also identified eight papers that solved cyclic rostering problems using column generation. Further research is required to investigate how our solver could be used to solve generic cyclic rostering problems.

3.2.2 Formulating and solving the column generation subproblem

In this section, we summarise the methods used in staff rostering literature to formulate and solve the column generation subproblem.

Table 3.2 shows a summary of the methods used in each of the 54 staff rostering papers to formulate their column generation subproblem. The most common two methods used to formulate the column generation subproblem in staff rostering literature were as a shortest path problem with resource constraints (SPPRC) as defined by Irnich and Desaulniers (2005) (which is typically solved using dynamic programming) and as a mixed-integer program (MIP). 23 of the 54 papers on staff rostering used an SPPRC and 26 of the 54 papers used a MIP. Otherwise, four papers used constraint programming, and four papers used other methods.

In the rest of this section, we discuss any paper which used a non-standard formulation or solution methodology. We define this as any paper which defined multiple separate SPPRCs, MIPs or constraint programs, any paper with a method in the "other" category, or any paper which did not generate roster-lines with their column generation subproblem.

Some researchers compared multiple methods of formulating column generation subproblems. Novak (2015) compared MIP, constraint programming and A* to solve the column generation subproblem. They concluded that their A* heuristic was preferable for smaller column generation subproblems but produced a too low-quality solution for the larger instances, and it was better to use a MIP for these instances. Novak (2015) also deemed constraint programming ineffective. Manuel and Barbosa (2018) used MIP with employee-shift variables (i.e. whether a given employee works a given shift), MIP with network flow variables and constraint programming to solve the column generation subproblem. They concluded that a MIP with employee-shift variables was the most effective option.

He and Qu (2012) used constraint programming to model the column generation subproblem with two improvements to the solution methodology. One of these was to generate diverse columns by using Depth-bounded Discrepancy Search (Walsh, 1997). The other was using a cost threshold where the first column with a cost below the threshold is returned by the column generation subproblem. The cost threshold was adaptively decreased as the rostering problem was solved. He and Qu (2012) argued that there is a powerful expressiveness to modelling the column generation subproblem using constraint programming; constraint programming can model a large variety of complicated nurse rostering constraints.

Some papers used a nested column generation subproblem where the column generation subproblem is decomposed into multiple SPPRCs and the output from one SPPRC becomes the input to another. We use the same nested column generation as first discussed in Dohn and Mason

Paper	Tasks	Task precedence	Fair	Preferences	Cyclic
Total papers (out of 54)*	18	6	2	31	8
Cleland (INRC)				✓	
Cleland (Waikato DHB)			✓	✓	
Van Den Eeckhout et al. (2020)	✓	✓			
Strandmark et al. (2020)				✓	
Lensing (2020)				✓	
Legrain et al. (2020)				✓	
Den Eeckhout et al. (2020)	✓	✓			
Bender et al. (2019)				✓	
Wang (2019)					
Pakpoom and Charnsethikul (2019)					✓
Swahn (2019)	✓				
Zeng et al. (2019)					
Václavík et al. (2018)				✓	
Restrepo et al. (2018)	✓				
Meijer (2018)				✓	✓
Dopheide and Spliet (2018)				✓	
Gomes et al. (2017)				✓	
Zamorano and Stolletz (2017)	✓	✓			
Karl and Volland (2017)	✓	✓			✓
Volland et al. (2017)	✓	✓			✓
Zeng et al. (2016)					
Restrepo et al. (2016)	✓			✓	
Firat et al. (2016)				✓	
Gérard et al. (2016)	✓				
Novak (2015)				✓	
Ohara and Tamaki (2015)					
Dahmen et al. (2015)				✓	
Boyer et al. (2014)	✓	✓		✓	
Brunner and Stolletz (2014)					
Côté et al. (2013)	✓			✓	
Dohn and Mason (2013)			✓	✓	
Brunner and Bard (2013)				✓	
Lusby et al. (2012)					✓
Restrepo et al. (2012)	✓				
Lim and Mobasher (2012)				✓	
He and Qu (2012)				✓	
Brunner et al. (2011)				✓	
Brunner and Edenharter (2011)					
Bulog (2011)				✓	
Mason et al. (2011)				✓	
Burke and Curtois (2010)				✓	
Maenhout and Vanhoucke (2010a)				✓	
Beliën and Demeulemeester (2008)				✓	✓
Al-Yakoob and Sherali (2008)				✓	
Purnomo and Bard (2007)				✓	✓
Maenhout and Vanhoucke (2007)				✓	
Ni and Abeledo (2007)	✓				
Beliën and Demeulemeester (2007)	✓			✓	
Hoogeveen and Penninkx (2007)				✓	
Beliën and Demeulemeester (2006)	✓			✓	
Demassey et al. (2006)	✓				
Wang (2006)					
Bard and Purnomo (2005b)			✓**	✓	
Belien and Demeulemeester (2004)	✓				
Mehrotra et al. (2000)	✓				✓

Table 3.1: Classification of staff rostering problems solved with column generation. The two "Cleland" rows are a classification of the two problem types we attempt to solve in this work. *Not including "Cleland". **Bard and Purnomo (2005b) model fairness over multiple rostering periods, not within a single rostering period.

(2013). For more details on our nested structure, see §4.2. Gérard et al. (2016) also used nested column generation to solve a granular nurse rostering problem. Firstly, they used an SPPRC to construct a set of tasks from 15 minute time periods. Secondly, they used an SPPRC to construct "timeslots" from tasks. Thirdly, they used an SPPRC to construct a set of shifts from timeslots, and finally, they used an SPPRC to construct lines of work from shifts. Restrepo et al. (2018) solved a similar nurse rostering problem where they used separate SPPRCs to create shifts from tasks and create roster-lines from shifts. The type of nested column generation described above is not to be confused with the nested column generation used by Tilk et al. (2019), and Enoch (2018), in which the column generation subproblem is solved using a MIP and that MIP is itself solved using column generation.

Some papers did not solve their SPPRC to produce roster-lines. Legrain et al. (2020) and Lensing (2020) used an SPPRC to generate only rotations or a sequence of shifts worked consecutively without a day off, which we define in §4.2 as an on-stretch. The on-stretches are combined to form roster-lines in the restricted master problem. Volland et al. (2017) used two different column generation subproblems, which generated two different types of columns: roster-lines from shifts and shifts from tasks, and added both types of columns to the restricted master problem. They used a MIP to solve both of these column generation subproblems. Similarly, Beliën and Demeulemeester (2008) generated two types of columns to add to the restricted master problem, roster-lines and surgery schedules. They generated roster-lines using an SPPRC and surgery schedules using a MIP.

Beliën and Demeulemeester (2007) compared two different decompositions for their column generation subproblem. The first was standard, which was solving the column generation subproblem to find all the activities that a given staff member works. The second was solving a column generation subproblem to find all of the staff who work a given activity. They concluded that the activity-decomposed column generation subproblem leads to better solve times, but the staff-decomposed column generation subproblem allows for more flexible modelling of rules. They used a MIP to solve both of these subproblems.

Václavík et al. (2018), and Lensing (2020) used regression with the dual solution of the restricted master problem to calculate an approximate lower bound on the reduced cost of the possible columns for a given employee. They trained the regression parameters while solving the nurse rostering problem, and as such, they did not need to precompute these parameters. They used this method to reduce the number of times the column generation subproblem required to be solved.

Four papers did not model the column generation subproblem as a SPPRC, MIP or constraint program. Boyer et al. (2014) and Côté et al. (2013) constructed their lines of work via a "minimum cost parse tree" which relates to their grammar-based formulation. Dopheide and Spliet (2018) and Bard and Purnomo (2005b) both used local search to generate columns; see 3.2.4 for more details.

We use an SPPRC to model the column generation subproblem and a dynamic program to solve it, a common method in the literature. Our column generation subproblem is nested, similarly to Gérard et al. (2016) and Restrepo et al. (2018). However, both research groups solved

Paper	SPPRC	MIP	CP	Other
Total papers (out of 54)*	23	26	4	5
Us (INRC, Waikato DHB)	✓			
Van Den Eeckhout et al. (2020)		✓		
Strandmark et al. (2020)	✓			
Lensing (2020)	✓			
Legrain et al. (2020)	✓			
Den Eeckhout et al. (2020)	✓			
Bender et al. (2019)	✓			
Wang (2019)	✓			
Pakpoom and Charnsethikul (2019)		✓		
Swahn (2019)		✓		
Zeng et al. (2019)	✓			
Manuel and Barbosa (2018)		✓	✓	
Václavík et al. (2018)		✓		
Restrepo et al. (2018)	✓			
Meijer (2018)	✓			
Dopheide and Spliet (2018)				✓
Gomes et al. (2017)		✓		
Zamorano and Stolletz (2017)	✓			
Karl and Volland (2017)		✓		
Volland et al. (2017)		✓		
Zeng et al. (2016)	✓			
Restrepo et al. (2016)	✓			
Firat et al. (2016)		✓		
Gérard et al. (2016)	✓			
Novak (2015)		✓	✓	✓
Ohara and Tamaki (2015)		✓		
Dahmen et al. (2015)		✓		
Boyer et al. (2014)				✓
Brunner and Stolletz (2014)		✓		
Côté et al. (2013)				✓
Dohn and Mason (2013)	✓			
Brunner and Bard (2013)		✓		
Lusby et al. (2012)	✓			
Restrepo et al. (2012)	✓			
Lim and Mobasher (2012)		✓		
He and Qu (2012)			✓	
Brunner et al. (2011)		✓		
Brunner and Edenharter (2011)		✓		
Bulog (2011)	✓			
Mason et al. (2011)	✓			
Burke and Curtois (2010)	✓			
Maenhout and Vanhoucke (2010a)	✓			
Beliën and Demeulemeester (2008)	✓	✓		
Al-Yakoob and Sherali (2008)		✓		
Purnomo and Bard (2007)		✓		
Maenhout and Vanhoucke (2007)	✓			
Ni and Abeledo (2007)		✓		
Beliën and Demeulemeester (2007)		✓		
Hoogeveen and Penninkx (2007)	✓			
Beliën and Demeulemeester (2006)		✓		
Demassey et al. (2006)			✓	
Wang (2006)		✓		
Bard and Purnomo (2005b)				✓
Belien and Demeulemeester (2004)		✓		
Mehrotra et al. (2000)		✓		

Table 3.2: Method used to solve column generation subproblem. *Not including "Cleland"

a single SPPRC to build roster-lines from shifts and days off. Genie++ (Dohn and Mason, 2013) is the only algorithm that divides up the construction of roster-lines, from shifts and days off, into multiple nested SPPRCs.

Many researchers model their column generation subproblem as an SPPRC. Thus, the techniques in this work which apply only to SPPRCs, i.e., dominance cost functions (§5.2.1), aggregate resource branching (§5.3.1), and our column generation subproblem heuristics (Chapter 7), are also applicable to many of the algorithms used in the literature, i.e., 23/54 papers.

3.2.3 Generic staff rostering problem modelling

Most column generation subproblems are modelled using SPPRCs, MIPs or constraint programming. MIPs are naturally modelled through an objective function and a set of linear constraints. Similarly, constraint programs are naturally modelled through an objective function and a set of logical constraints. However, SPPRCs are mostly constructed without such an abstraction. Thus, in this section, we present all usages of a generic methodology for modelling a large variety of staff rostering constraints in an SPPRC from the papers defined in our comprehensive literature review.

Restrepo et al. (2018) and Restrepo et al. (2016) used context-free grammar to define the make-up of a shift from tasks and breaks. Gérard et al. (2016) stated that using context-free grammar results in a huge hyper-graph and means the column generation subproblem can take much longer to solve.

Burke and Curtois (2014) and Novak (2015) used flexible pattern definitions to define rules for a roster-line. These patterns include days on, days off or specific shift types. They can be of various lengths and can be assigned complex costs. These patterns can describe shift sequences, days on/off patterns, weekend definitions and many other rules. Both Burke and Curtois (2014) and Novak (2015) used the patterns to define the cost of a roster-line but only Burke and Curtois (2014) also used the patterns to define hard constraints. Václavík et al. (2018) used the pattern definition from Burke and Curtois (2014).

Bender et al. (2019) created a domain-specific language where they can define a set of rules to map a customer's database to the inputs of their column generation algorithm. They modelled rules as SQL-like queries on a database where one query's output can form the input of another query. As far as we can tell, the rule engine does not map to resources in their SPPRC but more simple rules such as which employee can work which shift.

Dohn and Mason (2013) defined a set of entities such as on-stretches, work-stretches and roster-lines, each with a set of attributes that define the rules for an employee's line of work. These attributes can be modelled with a single C++ header file.

In §4.2, we present a novel improvement on the work of Dohn and Mason (2013) to model a large variety of staff rostering constraints. The model is given in a concise set of tables for each type of entity. These tables precisely describe the resource extension functions, dominance, feasibility, and cost of each resource in our resource constrained shortest path problems. We used these tables to accurately model all employee-specific rules of the INRC problems and Waikato

DHB problems. This generic model can also be easily optimised using the other techniques outlined in this work.

3.2.4 State-space reduction

In some of the more challenging column generation subproblems, the *state-space* is reduced to solve the column generation subproblem heuristically; refer to §2.2.2 for more details on the state-space. This section outlines all usages of heuristic state-space reduction from the papers defined in our comprehensive literature review.

The problem described by Den Eeckhout et al. (2020) is an integrated personnel and project scheduling problem and involves sharing personnel between multiple projects. They reduced the state-space by adding artificial restrictions to how many projects a single employee can be assigned. If they could not find a negative reduced cost column, they increased the maximum number of projects until they could, or the column generation subproblem is solved optimally.

Some researchers chose to have a greedy state-space reduction to keep high-quality components of a roster-line. Dohn and Mason (2013), and Burke and Curtois (2014) used a greedy strategy of keeping the lowest cost resource vectors at each node to solve the column generation subproblem heuristically. However, Burke and Curtois (2014) increased the number of resource vectors kept at each node until they found a negative reduced cost column or the column generation subproblem was solved optimally.

Gérard et al. (2016) heuristically reduced the number of shifts in their nested SPPRC by taking the minimum reduced cost shift from a set of shifts with the same day, start time, and working time, but different finish times. Barbosa et al. (2015) built a roster-line for a given driver by greedily assigning all duties in order from the most negative reduced cost duty to the least. They then added the roster-line to the restricted master problem. If they found no negative reduced cost column using this method, they solved the column generation subproblem exactly.

Some researchers have only considered a subset of the resources in their SPPRC for dominance. Dominance is a common technique for removing partial paths from an SPPRC which are strictly worse than other paths. If they are not considering a given resource for dominance, this resource could have a worse value and could still dominate a resource vector with a better value for that resource. Thus, this heuristic dominance strategy removes proof of optimality for the SPPRC. Zamorano and Stolletz (2017) and Restrepo et al. (2016) only considered a subset of the resources for dominance to solve the column generation subproblem heuristically. If they could not find a negative reduced cost column, they considered every resource for dominance and solved the column generation subproblem optimally. Strandmark et al. (2020) only included resources for minimum and maximum total work time and minimum and maximum consecutive shifts in their SPPRC to reduce the state-space heuristically. They modelled all other constraints by checking if they are violated in the shortest path and, if they are, resolving the SPPRC with large penalties applied to the appropriate employee-shift variables. For example, if the shortest path has too many working weekends, they assigned large penalties to the weekends exceeding this limit and resolve their SPPRC.

Maenhout and Vanhoucke (2010a) used a local search algorithm to take several existing columns with a negative reduced cost close to zero and performed single shift swaps to find negative reduced cost columns. They also employed a greedy shuffling heuristic to make all feasible swaps between the nurse schedules' sub-parts. If they could not find a negative reduced cost column using swaps, then they reverted to an SPPRC to solve the column generation subproblem exactly. Gomes et al. (2017) used neighbourhood swaps to find negative reduced cost columns in the neighbourhood of the best roster solutions found. They made swaps with one of four neighbourhood restrictions: "Change Allocation of one Day", which allows for the change of the activity worked on a single day; "Swap Allocation", which swaps any two activities worked on two days; "Change Allocation Working Windows", which changes which shifts are worked but maintains the same day on/off pattern; and lastly, "Invert Working Windows", which first inverts the day on/off pattern and then performs the same search as in "Change Allocation Working Windows". If they could not find a negative reduced cost column with neighbourhood search, they instead used a MIP to solve the column generation subproblem to optimality. Bard and Purnomo (2005b) generated columns through swaps to each roster-line in the incumbent solution. They then added all those columns to the restricted master problem and solved the restricted master problem as a MIP. They repeated this process until they could not find a better incumbent solution. Dopheide and Spliet (2018) solved their column generation subproblem through a greedy construction heuristic and then a series of swaps to find negative reduced cost columns.

One research group has tried multiple methods for state-space reduction. Beliën and Demeulemeester (2008) solved a problem to find surgery schedules in two phases. Initially, they solved their column generation subproblem using a MIP with a time limit to hopefully find negative reduced cost columns/surgery schedules. Later, they used a MIP without a time limit to find the most negative reduced cost column to prove optimality. Beliën and Demeulemeester (2006) solved a column generation subproblem for each activity in the roster to build a set of employees who work that activity. They restricted the state-space of this column generation subproblem by initially removing all staff from an activity's column generation subproblem if the staff have specified that they are not available to work on that day. This method is heuristic as the non-availability is a soft constraint. Lastly, Belien and Demeulemeester (2004) added additional hard constraints to the column generation subproblem, which are not often violated in an optimal solution, to reduce the state-space heuristically. They also explored a limited number of paths through their SPPRC but randomised which activities they considered first, so over successive SPPRC solves, they considered most parts of the feasible path state-space.

In Chapter 7, we compare greedy state-space reduction, resource-based state-space reduction, and local search for state-space reduction. However, our resource-based state-space reduction is novel in that we apply it to a generic model. Further, our local search for state-space reduction is novel in that it is generic and is solved using an SPPRC.

3.2.5 Resource dominance

When solving an SPPRC using dynamic programming, the dominance strategies from Irnich and Desaulniers (2005) are used most of the time. Although there have been multiple heuristic dominance techniques used to solve SPPRCs as seen in §3.2.4, we have not seen any attempts at improving the resource dominance in SPPRCs specifically for staff rostering problems. We could not find any novel description of dominance in SPPRCs that is not directly from Irnich and Desaulniers (2005).

However, two researchers in our field have mentioned the difficulty of dominance for solving the column generation subproblem for staff rostering problems. Gérard et al. (2016) stated, "[SPPRC] is much more efficient when only upper bounds are considered. When both lower and upper bounds co-exist, the dominance relations, used to reduce enumeration, are weaker." Legrain et al. (2020) found that soft constraints are bad for SPPRCs as if the value of a resource goes above its soft upper bound, the corresponding path still cannot be removed from the SPPRC. Further, if the value of a resource is below its soft lower bound, then the corresponding path cannot be dominated. Instead of modifying their dominance technique, Legrain et al. (2020) increased the size of their column generation subproblem's graph or used enumeration to remove the resources with soft lower and upper bounds. For example, they managed the penalties on the maximum number of consecutive assignments by adding network layers and arcs. They handled the penalties on the minimum number of consecutive assignments by enumeration of all possible states.

We discuss a novel dominance technique called dominance cost functions which makes dominance in SPPRCs for staff rostering problems more efficient in §5.2.1.

3.2.6 Branching techniques

Innovative branching techniques can quickly find solutions in the branch-and-price tree and drive up the lower bound. Common branches for staff rostering problems involve branching on the roster-line variables or constraint branching, i.e., that a given employee must work a given shift or not (see Ryan and Foster (1981)). In this section, we discuss all mentions of other branching techniques from the papers defined in our comprehensive literature review.

Legrain et al. (2020) used two branch types in their branch-and-price tree. Their first branch type was "branching on days" which involves enforcing that a particular day is a work-day or a rest day for a given employee. This branching rule is not complete, i.e., the rule does not guarantee integrality, so if they cannot branch on a day, they instead utilise constraint branching.

Zeng et al. (2019) solved a staff rostering problem to decide the number of employees required with different skill levels. They also used an incomplete branching method by branching on the number of employees with a particular skill level. If they could not find an integer solution through this branching method, they do not revert to a complete branching rule but instead solve a MIP with the columns at hand.

Mehrotra et al. (2000) used several unique, incomplete branching rules for shift scheduling: "Work period based branching," which is branching on the total number of shifts that cover a

specific work period; "Break period based branching," which is branching on the total number of shifts that cover a specific break period; and "Duty period based branching," which is branching on the total number of shifts that cover a set of consecutive periods.

Beliën and Demeulemeester (2006) compared two branching techniques on employee-shift variables: branching on whether an employee should work a given shift and branching on whether one employee must work a shift before another employee.

Maenhout and Vanhoucke (2010a) compared three different complete branching techniques on employee-shift variables. The first technique involved branching on whether an employee should work a given shift. The second technique involved creating non-binary branches where each branch corresponds to a different shift that a person can work or a day off. Their last branching technique involved branching on the 'minimal consecutive working days of the same shift type'. This branching method created three branches for a given shift: 1. Enforcing that the employee works the previous shift, and this shift; 2. Enforcing that the employee works this shift and the next shift; 3. Enforcing that the employee does not work this shift.

In §5.3, we discuss generic branching techniques for staff rostering problems which are novel in that we branch on generic rules and groups of employees. We also discuss prioritising branching on employees with higher skill levels and the order of application for our incomplete branching rules.

3.2.7 Neighbourhood search with branch and price

We have already reviewed the work of several research groups using neighbourhood search as a method of column generation in §3.2.4. Instead, in this section, we present all of the researchers who combine branch and price with neighbourhood search to find roster solutions within the neighbourhood of an incumbent roster solution from the papers defined in our comprehensive literature review.

We found three research groups using neighbourhood restricted column generation, i.e., adding restrictions to a column generation algorithm such that the algorithm only produced roster solutions that are similar to an incumbent solution. Legrain et al. (2020) used neighbourhood restricted column generation to improve sub-optimal roster solutions. First, they randomly selected either a subset of the nurses from all the nurses, or nurses working a given contract, or all the nurses with a certain skill. They fixed the lines of work for every other nurse and solved to find lines of work for the unfixed nurses using branch and price. We refer to this neighbourhood as "fixed employees neighbourhood" which we further elaborate on in §6.3.2.

Dahmen et al. (2015) created an initial roster solution by solving a branch-and-price problem for each of their "departments" independently. They then solved the full problem with a neighbourhood restriction, specifying that shifts must be similar to those produced in the incumbent solution, e.g., similar start times and lengths.

Bulog (2011) constrained the maximum number of changes to each employee's roster-line within the column generation subproblem and used this neighbourhood in a near-steepest de-

scent local search. We refer to this neighbourhood as "maximum roster-line modifications neighbourhood" which we further elaborate on in §6.2.1.

Apart from Legrain et al. (2020), Dahmen et al. (2015) and Bulog (2011), we did not find any other research group using neighbourhood constrained branch and price. Thus, we performed a brief search of some high profile researchers using neighbourhood constrained MIPs (without column generation) to solve staff rostering problems. Rahimian et al. (2017b) fixed the roster-lines for all but a small number of employees in an incumbent solution and solved a MIP to find optimal roster-lines for the remaining employees. Santos et al. (2016) fixed a subset of the days to match those in an incumbent and solved a MIP to find the optimal roster solution for the remaining days. They initially only had a single day unfixed and unfixed more days iteratively. Smet et al. (2016) used three neighbourhoods in their search: unfixing a small subset of days such as Santos et al. (2016), unfixing a small number of employees such as Rahimian et al. (2017b), and using local branching, which involves restricting the Hamming distance (sum of differences in variable values between the LP solution and an incumbent solution) with a constraint and solving the MIP to optimality with this restriction. We draw inspiration from the neighbourhood constrained MIPs of Smet et al. (2016), Santos et al. (2016) and Rahimian et al. (2017b) for our neighbourhood constrained column generation techniques in Chapter 6.

Two researchers used the entire set of columns produced so far from their column generation subproblems to find new incumbent solutions. Manuel and Barbosa (2018) used the set of columns generated from the column generation subproblems in different meta-heuristics to create improved roster solutions. These meta-heuristics include genetic algorithms and variable neighbourhood search. Brunner and Stolletz (2014) used simulated annealing to construct a high-quality roster solution at a given node in their branch-and-price search tree. They started with a roster solution created by selecting random columns in the node's column set. Then they used simulated annealing on that roster solution. The neighbourhood they explored using simulated annealing involved replacing a single roster-line in a roster solution with a column in the node's column set. They mentioned that this procedure is time-consuming, and as such, we would have liked to see a comparison with constructing a roster solution by solving a MIP instead of using simulated annealing.

In Chapter 6, we present several novel methods of neighbourhood restricted column generation for staff rostering problems. Apart from Legrain et al. (2020) who used a fixed employees neighbourhood and Bulog (2011) who used a maximum roster-line modifications neighbourhood, all our neighbourhood restricted column generation heuristics are novel for staff rostering problems.

3.3 International Nurse Rostering Competition

As covered by the reviews of Van Den Bergh et al. (2013), Ernst et al. (2004), Burke et al. (2004a) and Cheang et al. (2003), there are many different algorithms that are used to solve staff rostering problems. A large proportion of these algorithms have been used to solve the

International Nurse Rostering Competition (INRC), both in the competition itself as well as papers published after the competition. Over 100 papers have cited the competition problems created by Haspeslagh et al. (2010). In this section, we present the five papers which reported the highest-quality solutions.

The two most effective staff rostering algorithms in solving these problems were by Burke and Curtois (2014) and Santos et al. (2016). Burke and Curtois (2014) used column generation and Santos et al. (2016) used matheuristics. Because of the effectiveness of these two algorithms, we focus deeply on these two topics in this thesis.

We judge the quality of each group's results in terms of the two following criteria—the lowest cost roster solution identified and, in the case of optimisation-based algorithms, the identified lower bound for the costs of possible roster solutions. Solution times are not considered within the scope of this thesis. We chose to include a paper if its results are non-dominated, i.e., the paper must have beaten every other paper in the quality of the solution of at least one problem. The only exception is Valouxis et al. (2012) who is honourably mentioned as they achieved the highest score in the competition itself.

Table 3.3 depicts the best results obtained by each of the research groups available in the literature. As we prove the optimality of the solutions proposed in this thesis corresponding to each of the problem instances, the "Cleland" column presents the best possible results for comparison. In addition, corresponding to problems in which we have identified a new lower bound or upper bound, the proven optimal solution is displayed in bold, and the corresponding previous best upper or lower bound has been crossed out.

Burke and Curtois (2014) demonstrated the effectiveness of column generation by applying it to the INRC problems and concluded that strong lower bounds could be generated simply by solving the root node of their branch-and-price search tree. They produced the best-known roster solutions for the majority of the problems that they considered using a branch-and-bound dive. However, they reported no further improvements to the lower bounds via branching. Further, they did not report solutions to the most challenging instances—the 20 *hidden* ones.

We are pleased to see that we found identical root node objectives to every problem that Burke and Curtois (2014) solved when solving our root node, as this verifies that our formulation is correct. Any improvements we made to Burke and Curtois (2014)'s best known lower bounds were through branching.

Santos et al. (2016) solved the INRC problems as a MIP with a novel clique separation procedure to generate cuts to improve the lower bounds. Using this procedure with the COIN-OR branch-and-cut solver (Cbc), they produced some strong lower bounds. However, they discovered that commercial solvers operating over compact formulations are ineffective for the identification of lower bounds and low-cost roster solutions to relatively difficult INRC problems. In order to identify low-cost roster solutions, Santos et al. used two separate variable neighbourhood descent (VND) matheuristics with two separate neighbourhoods. Each matheuristic's neighbourhood was parameterised by k, which indicated its size. The first neighbourhood involved forcing each employee to *work the same shifts* as in an incumbent roster solution on each day except $n - k$ consecutive days, where n denotes the total number of days. In §6.1, we term the neighbourhood

used in this matheuristic, the "fixed days neighbourhood." They then used CPLEX to identify the optimal neighbouring solution by changing the shifts worked on the remaining days. The second neighbourhood involved forcing each employee to *work on the same days* as in an incumbent roster solution on each day except $n - k$ consecutive days. In §6.1, we term the neighbourhood used in this matheuristic, the "fixed days (on/off) neighbourhood."

Valouxis et al. (2012) developed a two-phase matheuristic to identify low-cost roster solutions. The first phase involved the utilisation of a MIP with full column enumeration (128 columns per person) to identify four separate week-long rosters for each of the weeks in a given problem instance, each with the lowest possible cost given a subset of the original problem instance's penalties. During this initial phase, the shifts worked by each employee are ignored, and only the assigned work-days of each employee are considered. An example of a week-long roster solution during this initial phase is as follows: on-on-off-off-on-on-on. Each of the four-week-long rosters were then combined, and a randomised hill-climbing heuristic was used with three different swaps to improve the solution quality. Phase 2 involved the implementation of another integer program to assign shifts to each of the employee's work-days specified in phase one. This algorithm produced the highest overall score in the INRC competition itself but has since been outperformed by methods proposed by Rahimian et al. (2017a) and Santos et al. (2016).

Rahimian et al. (2017a) used a MIP to solve the INRC problems with several enhancements to improve solution times. The first enhancement involved the application of a constraint programming algorithm to generate low-cost roster solutions that could be added to the MIP solver as upper bounds. The second enhancement solved several relaxations of the original problem to identify lower bounds. For example, similar to the approach discussed in Valouxis et al. (2012), a MIP was used to identify an optimal solution corresponding to each of the four weeks. Then, the solutions were combined together to constitute a single four-week roster. The third enhancement solved a modified MIP with some of the soft constraints modelled instead as hard constraints to identify high-quality roster solutions quickly to add to the MIP solver as upper bounds. Over one hour, the enhanced MIP identified several of the best-known roster solutions and proved some of them to be optimal.

Lü and Hao (2012) matched the majority of the best-known upper bounds to the INRC problem instances using an adaptive neighbourhood search heuristic with neighbourhood swaps.

Despite many researchers having attempted to solve these problems, 11 medium and long problems remained for which the cost of the best-known roster solution identified by previous works is larger than the corresponding best known lower bound. In Chapter 5, we demonstrate how we identified optimal solutions for all of the 11 problems where the best known lower bound in literature was not equal to the best known upper bound, along with every other medium and long problem, within a reasonable period of time (less than four hours).

It is worth noting that there are two core methods of modelling a staff rostering problem using integer programming. Santos et al. (2016) and Rahimian et al. (2017a), for example, use

employee shift variables, i.e.,

$$x_{esd} = \begin{cases} 1, & \text{if employee } e \text{ works shift } s \text{ on day } d. \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

Whereas Burke and Curtois (2014) and we use roster-line variables, i.e.,

$$\lambda_e^r = \begin{cases} 1, & \text{if employee } e \text{ works roster-line } r. \\ 0, & \text{otherwise} \end{cases} \tag{3.2}$$

Even with Santos et al. (2016) using clever cut generation and Rahimian et al. (2017a) using various relaxations to drive up the lower bound, Burke and Curtois (2014) and we were both able to attain better lower bounds merely from solving the root node. This is a testament to the strength of the roster-line variables based formulation.

There are too many possible roster-line variables to enumerate all of them in a MIP without column generation for a reasonably sized staff rostering problem. Thus, one must either choose a subset of the roster-line variables to use within the MIP but give up proof of optimality or use column generation, which can efficiently solve MIPs with a very large number of variables. Thus, MIP with column generation can lead to a stronger formulation for staff rostering problems than MIP without column generation.

3.4 Chapter summary

In §3.2, we investigated column generation techniques for solving staff rostering problems in the literature. We also found that there has not been extensive research into resource dominance or branching for solving staff rostering problems to proven optimality with column generation. This motivates our novel resource dominance and branching strategies described in Chapter 5.

Although many researchers have used heuristics to speed up column generation, most have used one or two heuristics at most, and no one has extensively compared their use. This motivates our comprehensive comparison of column generation based matheuristics in Chapters 6-7.

In §3.3, we established that no one has found proven optimal solutions to every problem in the INRC. Thus, in Chapter 5, we develop new techniques to find proven optimal solutions to these problems.

We also found that column generation has been a useful technique for solving the INRC problems, even though no one has yet used column generation to solve every INRC problem. Thus, in the following chapters, we continue to develop on the column generation work of Dohn and Mason (2013), instead of switching to another technique of solving these problems.

Instances	Us	Previous best bounds			Burke		Santos		Rahimian		Val.	Lü
	LB &UB	LB	UB	Gap	LB	UB	LB	UB	LB	UB	UB	UB
Medium												
Early												
01	240	240	240	0	240	240	240	240	240	240	240	240
02	240	240	240	0	240	240	240	240	240	240	240	240
03	236	236	236	0	236	236	236	236	236	236	236	236
04	237	237	237	0	237	237	237	237	237	237	237	237
05	303	303	303	0	303	303	303	303	303	303	303	303
Late												
01	**157**	~~156~~	157	1	156	157	156	157	144	157	158	164
02	18	18	18	0	18	18	18	18	18	18	18	20
03	29	29	29	0	29	29	29	29	22	29	29	30
04	35	35	35	0	35	35	35	35	32	35	35	35
05	107	107	107	0	107	107	107	107	107	107	107	112
Hidden												
01	**111**	~~89~~	111	22	?	?	88	111	89	117	130	117
02	**219**	~~197~~	~~220~~	23	?	?	197	221	190	248	221	220
03	**34**	~~28~~	34	6	?	?	28	34	23	36	36	35
04	**78**	~~73~~	78	5	?	?	73	78	68	81	81	79
05	**118**	~~91~~	~~119~~	28	?	?	91	119	56	129	122	119
Long												
Early												
01	197	197	197	0	197	197	197	197	197	197	197	197
02	219	219	219	0	219	219	219	219	219	219	219	222
03	240	240	240	0	240	240	240	240	240	240	240	240
04	303	303	303	0	303	303	303	303	303	303	303	303
05	284	284	284	0	284	284	284	284	284	284	284	284
Late												
01	235	235	235	0	235	235	232	235	231	235	235	237
02	229	229	229	0	229	229	229	229	229	229	229	229
03	**220**	~~219~~	220	1	219	220	219	220	218	221	220	222
04	221	221	221	0	221	221	215	222	213	230	221	227
05	83	83	83	0	83	83	83	83	79	94	83	83
Hidden												
01	**346**	~~341~~	346	5	?	?	341	346	337	368	363	346
02	**89**	~~86~~	89	3	?	?	86	89	81	92	90	89
03	**38**	~~36~~	38	2	?	?	36	38	17	47	38	38
04	**22**	~~19~~	22	3	?	?	19	22	14	29	22	22
05	41	41	41	0	?	?	41	41	40	41	41	45

Table 3.3: Best known upper and lower bounds available in the literature: us, Burke and Curtois (2014), Santos et al. (2016), Valouxis et al. (2012), Rahimian et al. (2017a), and Lü and Hao (2012) compared to our best upper and lower bounds. Our improvements to the previous best bounds in literature are highlighted in bold. The previous best LBs and UBs have been crossed out to indicate that our proven optimal solution improves upon these bounds. “?" is added to indicate the problems that are not considered in the relevant paper.

Chapter 4

Standard branch and price for staff rostering problems

This chapter introduces how we solve staff rostering problems within a nested column generation framework. In §4.1, we introduce our column generation algorithm. In §4.2-4.4, we show how we model and construct roster-lines. In §4.5, we introduce branching for finding integer solutions and in §4.6, we introduce dual stabilisation, sprint pricing and the tools we are using to perform our tests.

Note that most of the column generation subproblem modelling and solution methodologies in this chapter are derived from Genie++ which has been developed by Dohn and Mason (2013), Mason et al. (2011). There have only been slight modifications in order to model Waikato DHB problems which are first described in §4.2. Furthermore, the linear programming formulation is standard for staff rostering with column generation.

In this Chapter, however, we do present a new mathematical terminology for precisely defining the column generation subproblem in a generic way. For our novel improvements on column generation modelling and solution methodologies, see Chapters §5-§7.

4.1 Column generation

The fundamental algorithm of Genie++ is a branch-and-price algorithm, which is a branch-and-bound algorithm in which the linear programming relaxation is solved at each node in the branch-and-bound search tree via column generation. The execution of column generation involves the solution of two core problems—the restricted master problem (RMP) and the column generation subproblem. The RMP is a linear program to identify the lowest-cost convex combination of a given subset of possible full roster-lines for each employee. We define our RMP in §4.1.1 The column generation subproblem is used to identify feasible full roster-lines for the RMP; consult §4.2-4.4 for more details on solving the column generation subproblem.

4.1.1 Restricted master problem

This section introduces how we model our RMP for both the INRC problems and the Waikato DHB problems. For an introduction to using column generation for staff rostering, see §2.2.

We can not necessarily meet all demands specifically for the Waikato DHB problems. Therefore, we also define a set of slack y_D^- and surplus y_D^+ variables with associated costs c_D^- and c_D^+ respectively.

We show the RMP for the Waikato DHB problems in (4.1)-(4.5). The objective, depicted in (4.1), is to minimise the roster's cost, which is defined as the sum of each column's costs plus the costs of slacks and surpluses on each demand. The constraint set introduced by (4.2) ensures each employee works a single roster-line. The constraint set presented by (4.3) ensures that the slack and surplus variables are correct for a given demand.

$$\text{RMP:} \quad \text{Minimize} \quad \sum_{e\in\mathcal{E}}\sum_{r\in\mathcal{R}_e} c_e^r\lambda_e^r + \sum_{D\in\mathcal{D}} c_D^- y_D^- + \sum_{D\in\mathcal{D}} c_D^+ y_D^+ \tag{4.1}$$

$$\text{s.t.} \quad \sum_{r\in\mathcal{R}_e} \lambda_e^r = 1 \qquad \forall e \in \mathcal{E} \quad [\pi_e] \tag{4.2}$$

$$\sum_{e\in\mathcal{E}}\sum_{r\in\mathcal{R}_e} a_{eD}^r\lambda_e^r = b_D - y_D^- + y_D^+ \qquad \forall D \in \mathcal{D} \quad [\pi_D] \tag{4.3}$$

$$0 \le \lambda_e^r \le 1 \qquad \forall e \in \mathcal{E}, \forall r \in \mathcal{R}_e \tag{4.4}$$

$$y_D^- \ge 0, y_D^+ \ge 0 \qquad \forall D \in \mathcal{D} \tag{4.5}$$

Note that there are no upper limits on the slack and surplus variables. We can still model a demand with strict requirements by providing a very large slack cost c_D^- or surplus cost y_D^+ for that demand. This model ensures that we can always find a feasible roster solution, provided we can generate a legal roster-line for each employee. This makes constructing a feasible roster solution trivial. We can use a feasible roster solution to warm-start our RMP or as a starting solution for our column generation based neighbourhood search (see Chapter 6).

Also note that non-linear demands can effectively be modelled with multiple demands, i.e., a demand for being one nurse short can be modelled with a small cost and a separate demand for being three nurses short can be modelled with a large cost.

Because we model our pricing problem using shifts and our restricted master problem using demands, we need to translate between shifts and demands. Thus, we define the set of demands fulfilled by employee e if they work shift S as $\mathcal{D}_e^S$. We define a_{eD}^r as one if shift S is worked in roster-line $\hat{R}_e^r$ and fulfils demand D and zero otherwise, i.e.,

$$a_{eD}^r = \begin{cases} 1, & \text{if } \exists S \in \hat{R}_e^r \text{ s.t. } D \in \mathcal{D}_e^S \\ 0, & \text{otherwise} \end{cases} \tag{4.6}$$

Unlike our Waikato DHB problems, the INRC problems utilise a shift demand. This means there is a one to one relationship between shifts and demands. No changes to our RMP are required to model this simplified case.

As we model a given roster-line in our pricing problem as a set of dated shifts, we calculate the reduced cost of an entering column in terms of the cost of the roster-line for a given employee, the employee dual, and a set of shifts duals for shifts worked in that column. Thus, we need to convert our demand duals from 4.3 to shift duals with the following equation:

$$\pi_S = \sum_{D \in \mathcal{D}_e^S} \pi_D \tag{4.7}$$

Thus, we can calculate the reduced cost $f_{\mathrm{R}}(\hat{R}_e^r)$ of entering column $\hat{R}$ with the following equation:

$$f_{\mathrm{R}}(\hat{R}_e^r) = c_e^r - \sum_{S \in \hat{R}} \pi_S - \pi_e \tag{4.8}$$

4.1.2 Column generation subproblem

To identify the input columns for our RMP, we identify the lowest reduced-cost, feasible roster-line corresponding to each employee.

In §4.2, we demonstrate how we model the reduced cost and feasibility of a given roster-line, given the rules which apply to that employee. In §4.3, we show how we model a set of rules for the INRC problems and a set of rules for the Waikato DHB problems. Lastly, in §4.4, we show our nested Shortest Path Problem with Resource Constraints (SPPRC) for finding the most negative reduced cost roster-line.

4.2 Modelling employee rules

We define a roster-line for a given employee as a sequence of shifts worked and days off corresponding to that employee's work schedule. We define the set of shifts worked on day d as $\mathcal{S}^d$ and a day off on day d as ϕ^d. Hence, we can denote a roster-line R as a set of activities, with one *activity* per day d, where an activity is either a work-shift $S \in \mathcal{S}^d$ or a day off ϕ^d. For example, we can denote a given roster-line as $R = \{M^1, M^2, \phi^3, \ldots, N^{n-1}, N^n\}$ where M^1 and M^2 denote morning shifts on days 1 and 2 respectively, ϕ^3 is a day off on day 3, and N^{n-1} and N^n are night shifts on days $n-1$ and n respectively.

A *roster-line* entity,

$$R = W_1 \cup W_2 \cup \ldots \cup W_{|R|_W} = (W_1, W_2, \ldots, W_{|R|_W})$$

can be modelled as a set of $|R|_W$ work-stretch W entities, each of which is a series of consecutive work-days followed by a series of consecutive off-days. We utilise the set notation, $\cup$, to indicate that an entity is the union of multiple sets of daily activities. Further, we utilise the vector notation to indicate that the component entities can be naturally ordered based on the days on which they are defined. Henceforth in this thesis, we denote the number of Y entities in entity X, by $|X|_Y$, e.g., $|R|_W$ is the number of work-stretches in a roster-line. We denote the number of activities in entity X with $|X|$. Each roster-line begins on day 1. For each employee, we seek

to derive a *full roster-line*, $\hat{R}$, which is a roster-line ending on day n, where n denotes the total number of days considered in the problem. However, we also consider (partial) roster-lines which end before day n.

Each *work-stretch*

$$W = O \cup F = (O, F)$$

can be further segmented into two component entities, an on-stretch, O, which is defined to be a series of consecutive work-days, and an off-stretch, F, which is defined to be a series of consecutive off-days.

We define an *on-stretch*

$$O = \{S_1, S_2, \ldots, S_{|O|}\} = (S_1, S_2, \ldots, S_{|O|})$$

as a sequence of $|O|$ shifts S worked consecutively without a day off. When modelling the (less complex) INRC problems, we consider an on-stretch to be a series of consecutive calendar days, on which shifts are worked, with no calendar days off. However, in the NZ Nurse MECA, it specifies a day off as a 24-hour break between shifts. In this case, if you work the shift sequence: $(N, -, A)$ where the night shift starts on day one, and the AM shift starts on day 3, there is less than 24 hours off, so we define this shift sequence pattern as an on-stretch. Conversely, the shift sequence (A, N) is a single shift on-stretch followed by 24 hours off followed by another single shift on-stretch. The first on-stretch in a roster-line can be of length 0 since a roster-line can start with a day off.

We define an *off-stretch*,

$$F = \{\phi^d, \phi^{d+1}, \ldots, \phi^{d+|F|}\} = (\phi^d, \phi^{d+1}, \ldots, \phi^{d+|F|})$$

as $|F|$ consecutive off-days beginning on day d. When modelling the INRC problems, there is one off-stretch for each start day $1 \leq d \leq n$ and length $1 \leq |F| \leq n - d + 1$ as an off-stretch is a series of consecutive calendar days off and there are no hard constraints on off-stretch length. However, as mentioned previously, the NZ Nurse MECA specifies a day off as a 24 hour break between shifts and thus, there is one off-stretch for each feasible combination of end times and start times of the different shifts.

Each off-stretch must have at least 1 day off except the dummy off-stretch $F = (\phi^{n+1}, \phi^{n+1})$, since a roster-line can end on a worked day.

We define $\mathcal{S}$ to be the set of all shifts, $\mathcal{F}$ to be the set of all off-stretches, $\mathcal{O}$ to be the set of all on-stretches, $\mathcal{W}$ to be the set of all work-stretches, and $\mathcal{R}$ to be the set of all roster-lines.

We note that although we have elected to break down a roster-line into work-stretch, on-stretch and off-stretch component entities, this decomposition is arbitrary. For example, we could easily break down a roster-line into fortnight and week entities.

4.2.1 Roster-line cost model

For the remaining sections, we refer to the reduced cost of a full roster-line as the 'cost' and model each component entity as having its own associated cost. As each entity is composed of dated

shifts and days off, the cost of an entity is defined as the sum of the costs of all rule violations by the entity minus the sum of the duals for each of the shifts worked within the entity.

As we decompose some of our entities into a set of component entities, we can model each entity's cost with the sum of the costs of its component entities and an *interaction cost* corresponding to the particular sequence of its component entities. For example, an on-stretch cost includes the cost of its component shift entities and an interaction cost, e.g., a cost for working too many days in a row and a cost for working a night shift followed by a morning shift.

The implementation of such a nested cost structure enables the costs, which are solely applicable to work-stretches, to be reduced to a single work-stretch cost. Thus, they are no longer required to be considered during the calculation of the costs of roster-line entities. Likewise, the rules that are solely applicable to on-stretches are no longer required to be considered during the calculation of the costs of work-stretch entities. This nested cost structure simplifies the process of modelling and costing a roster-line. It also enhances the efficiency of the identification of the lowest-cost roster-line corresponding to a particular employee (Dohn and Mason, 2013).

We model the cost of a shift, S, with the following function:

$$f_{\mathrm{S}}(S) = f'_{\mathrm{S}}(S) - \pi_S,$$

where $f'_{\mathrm{S}}(S)$ is a cost used to model the aversion of an employee to work shift, S, or on the day containing shift, S and π_S is the shift dual calculated from our RMP.

By definition of the proposed nested cost model, the cost of an on-stretch, $O = (S_1, S_2, \ldots, S_{|O|})$, is given by the following function:

$$f_{\mathrm{O}}(O) = \sum_{S \in O} f_{\mathrm{S}}(S) + g_{\mathrm{O}}(O) \tag{4.9}$$

where $f_{\mathrm{S}}(S_i)$ denotes the cost of the shift, S_i, and $g_{\mathrm{O}}(O)$ denotes the on-stretch interaction cost of the on-stretch, O. Certain rules, such as that involving the attribution of a cost to an excessive or inadequate number of consecutive work-days, can be modelled solely using the on-stretch interaction cost, $g_{\mathrm{O}}(O)$.

The cost of a work-stretch, $W = (O, F)$, is given by the following function:

$$f_{\mathrm{W}}(W) = f_{\mathrm{O}}(O) + f_{\mathrm{F}}(F) + g_{\mathrm{W}}(W) \tag{4.10}$$

where $g_{\mathrm{W}}(W)$ denotes the work-stretch interaction cost of the work-stretch, W. The cost of an off-stretch, $f_{\mathrm{F}}(F)$, models a given employee's preference to work one of the shifts on the same day as this off-stretch. Certain rules, such as that involving the attribution of a cost to an employee working incomplete weekends, can be modelled purely with the work-stretch interaction cost, $g_{\mathrm{W}}(W)$.

Finally, the cost of a roster-line, $R = (W_1, W_2, \ldots, W_{|R|_W})$, is given by the following function:

$$f_{\mathrm{R}}(R) = \sum_{W \in R} f_{\mathrm{W}}(W) + g_{\mathrm{R}}(R) \tag{4.11}$$

where $g_{\mathrm{R}}(R)$ denotes the roster-line interaction cost of the roster-line, R. Certain rules, such as that involving the attribution of a cost to an excessive number of consecutive working-weekends,

need to be modelled using the roster-line interaction cost, $g_{\mathrm{R}}(R)$. Since the employee dual π_e is fixed for a given employee, we can ignore this dual when finding the minimum reduced cost roster-line.

The procedure to calculate the interaction cost of each entity is described in detail in the following section.

4.2.2 Building entities

Later in the paper, the proposed model is used to solve a nested Shortest Path Problem with Resource Constraints (SPPRC) for each employee. Thus, we adopt the terminology introduced by Irnich and Desaulniers (2005) to describe our model in this section. We use resource vectors to model the rules that have been provided in the INRC contracts and that we have built for the Waikato DHB. Five sets of resources are considered—Θ_{S} for shifts, Θ_{O} for on-stretches, Θ_{F} for off-stretches, Θ_{W} for work-stretches, and Θ_{R} for roster-lines. Each entity, $\tau \in \mathcal{S} \cup \mathcal{F} \cup \mathcal{O} \cup \mathcal{W} \cup \mathcal{R}$, corresponds to one associated *resource vector*, $\mathcal{T}_\tau$, which comprises one *resource variable* for each *resource*, θ. Hence, a resource vector corresponding to a work-stretch includes $|\Theta_{\mathrm{W}}|$ variables. A rule may involve multiple resource variables within its constituent entities of varying types.

In Genie++, new entities and their associated resource vectors are created by adding a component entity to the end of a shorter entity. The individual cases are described in detail below.

A new on-stretch, O', is created by adding a single shift, S, to the end of a shorter on-stretch, $O = (S_1, S_2, \ldots, S_{|O|})$. In other words,

$$O' = O \cup S = (S_1, S_2, \ldots, S_{|O|}, S) \tag{4.12}$$

where the shift, S, is scheduled directly after shift, $S_{|O|}$ with no day off in between. Therefore, the resource vector, $\mathcal{T}_{O'}$, corresponding to the on-stretch, O', is calculated based on the associated resource vector, $\mathcal{T}_O$, of the shorter on-stretch, O, and the resource vector, $\mathcal{T}_S$, of the shift, S, using a vector of *resource extension functions* (REFs):

$$\mathcal{T}_{O'} = \mathcal{E}_{\mathrm{O}}(\mathcal{T}_O, \mathcal{T}_S) = \left(\mathcal{E}_{\mathrm{O}}^{\theta}(\mathcal{T}_O, \mathcal{T}_S),\ \forall \theta \in \Theta_{\mathrm{O}}\right) \tag{4.13}$$

where $\mathcal{E}_{\mathrm{O}}^{\theta}(\mathcal{T}_O, \mathcal{T}_S)$ denotes the REF corresponding to the on-stretch resource, θ. The resource vector, $\mathcal{T}_S$, corresponding to each shift, S, is selected to optimise the modelling of rules included in the problem description. As an illustrative example of an on-stretch REF, let us consider the "last shift type" resource of an on-stretch, O. We retain the shift type of the final shift in the on-stretch with resource variable, $\mathcal{T}_O^{\mathrm{LastShiftType}}$. Then, this is used in the REFs in connection with resources related to shift sequence patterns (e.g., a night shift followed by a morning shift). In this case, the REF is given by $\mathcal{E}_{\mathrm{O}}^{\mathrm{LastShiftType}}(\mathcal{T}_O, \mathcal{T}_S) = \mathcal{T}_S^{\mathrm{ShiftType}}$ where $\mathcal{T}_S^{\mathrm{ShiftType}}$ denotes the shift resource representing the shift type.

A work-stretch, W, is created by adding an on-stretch, O, to an off-stretch, F. In other words,

$$W = O \cup F \tag{4.14}$$

where the first day of the off-stretch, F, follows the final day of the on-stretch, O. The resource vector, $\mathcal{T}_W$, corresponding to the work-stretch, W, is calculated from the resource vector, $\mathcal{T}_O$, of the on-stretch, O, and the resource vector, $\mathcal{T}_F$, of the off-stretch F, using the following equation:

$$\mathcal{T}_W = \mathcal{E}_{\mathrm{W}}(\mathcal{T}_O, \mathcal{T}_F) = \left(\mathcal{E}^{\theta}_{\mathrm{W}}(\mathcal{T}_O, \mathcal{T}_F),\ \forall\theta \in \Theta_{\mathrm{W}}\right) \tag{4.15}$$

where $\mathcal{E}^{\theta}_{\mathrm{W}}(\mathcal{T}_O, \mathcal{T}_F)$ denotes the REF corresponding to the work-stretch resource, θ. The resource vector, $\mathcal{T}_F$, corresponding to each off-stretch entity, F, is explicitly enumerated as only a relatively small number of combinations of consecutive off-days are possible.

Finally, a new roster-line, R', is created by adding a single work-stretch, W, to a shorter roster-line, $R = (W_1, W_2, \ldots, W_{|R|_W})$. In other words,

$$R' = R \cup W = (W_1, W_2, \ldots, W_{|R|_W}, W) \tag{4.16}$$

where the first day of the work-stretch, W, follows the final day of the roster-line, R. The resource vector $\mathcal{T}_{R'}$ corresponding to the roster-line, R', is calculated based on the resource vector, $\mathcal{T}_R$, of the roster-line, R, and the resource vector $\mathcal{T}_W$, of the work-stretch, W, using the following equation:

$$\mathcal{T}_{R'} = \mathcal{E}_{\mathrm{R}}(\mathcal{T}_R, \mathcal{T}_W) = \left(\mathcal{E}^{\theta}_{\mathrm{R}}(\mathcal{T}_R, \mathcal{T}_W),\ \forall\theta \in \Theta_{\mathrm{R}}\right) \tag{4.17}$$

where $\mathcal{E}^{\theta}_{\mathrm{R}}(\mathcal{T}_R, \mathcal{T}_W)$ denotes the REF corresponding to the roster-line resource, θ.

4.2.3 Evaluating the cost of entities

Using the resource vectors calculated for any given on-stretch, work-stretch, or roster-line, $\tau \in \mathcal{O} \cup \mathcal{W} \cup \mathcal{R}$, we can calculate the interaction cost for each entity, introduced in §4.2.1. The interaction cost, $g_\tau(\tau)$, for the entity, τ, is given by the following equation in terms of its associated resource vector, $\mathcal{T}_\tau$.

$$g_\tau(\tau) = \sum_{\theta \in \Theta_\tau} g^{\theta}_{\tau}(\mathcal{T}^{\theta}_{\tau}) \tag{4.18}$$

where $g^{\theta}_{\tau}(\mathcal{T}^{\theta}_{\tau})$ denotes the cost of the resource, θ (c.f. overall entity cost presented in §4.2.1). Four primary types of resource cost functions are included in the INRC problems and three additional resource cost functions are included in the Waikato DHB problems. The first resource cost function for a resource, θ, is given by

$$C^{\emptyset}: \qquad g^{\theta}_{\tau}(\mathcal{T}^{\theta}_{\tau}) = 0, \tag{4.19}$$

and is used if the resource, θ, does not contribute directly to the cost of an entity and is, instead, only used in the REF calculations corresponding to other resources.

The second resource cost function is given by

$$C^{\mathrm{linear}}: \qquad g^{\theta}_{\tau}(\mathcal{T}^{\theta}_{\tau}) = \alpha^{\theta}\mathcal{T}^{\theta}_{\tau}, \tag{4.20}$$

where α^{θ} is a constant penalty cost. This cost function is used if resource θ directly corresponds to rule violations.

The third resource cost function is given by

$$C^{\mathrm{UB}}: \qquad g^{\theta}_{\tau}(\mathcal{T}^{\theta}_{\tau}) = \begin{cases} \alpha^{\theta}(\mathcal{T}^{\theta}_{\tau} - \gamma^{\theta}), & \text{if } \mathcal{T}^{\theta}_{\tau} > \gamma^{\theta} \\ 0, & \text{otherwise} \end{cases}, \tag{4.21}$$

where α^{θ} is a penalty cost applied if the value of $\mathcal{T}^{\theta}_{\tau}$ is greater than an upper bound, γ^{θ}. This cost function is used if resource θ tracks a value corresponding to a rule violation if above a certain upper bound.

The fourth resource cost function is given by

$$C^{\mathrm{LBUB}}: \qquad g^{\theta}_{\tau}(\mathcal{T}^{\theta}_{\tau}) = \begin{cases} \alpha^{\theta}(\mathcal{T}^{\theta}_{\tau} - \gamma^{\theta}), & \text{if } \mathcal{T}^{\theta}_{\tau} > \gamma^{\theta} \\ \beta^{\theta}(\delta^{\theta} - \mathcal{T}^{\theta}_{\tau}), & \text{if } \mathcal{T}^{\theta}_{\tau} < \delta^{\theta} \\ 0, & \text{otherwise} \end{cases}, \tag{4.22}$$

where α^{θ} is a penalty cost applied if the value of $\mathcal{T}^{\theta}_{\tau}$ is greater than an upper bound, γ^{θ}, and β^{θ} is a penalty cost applied if the value of $\mathcal{T}^{\theta}_{\tau}$ is less than a lower bound, δ^{θ}. This cost function is used if resource θ tracks a value corresponding to rule violations if above a certain upper bound or below a certain lower bound. For examples of all four of these resource cost functions, see §4.3.1.

Note that the first three resource cost functions are just special cases of the fourth one so in principle one could model all of these with just alternative parameter choices for γ^{θ} and δ^{θ}. However, we choose to separate them as it helps better express the structure of the problem.

Our fifth, sixth and seventh cost functions are only used for the more complex Waikato DHB problems. The fifth resource cost function is given by

$$C^{\mathrm{LB}}: \qquad g^{\theta}_{\tau}(\mathcal{T}^{\theta}_{\tau}) = \begin{cases} \alpha^{\theta}(\delta^{\theta} - \mathcal{T}^{\theta}_{\tau}), & \text{if } \mathcal{T}^{\theta}_{\tau} < \delta^{\theta} \\ 0, & \text{otherwise} \end{cases}, \tag{4.23}$$

where α^{θ} is a penalty cost applied if the value of $\mathcal{T}^{\theta}_{\tau}$ is less than a lower bound, δ^{θ}. This cost function is used if resource θ tracks a value corresponding to a rule violation if below a certain lower bound.

The sixth resource cost function is given by

$$C^{\mathrm{2UB}}: \qquad g^{\theta}_{\tau}(\mathcal{T}^{\theta}_{\tau}) = \begin{cases} (\alpha^{\theta} + \beta^{\theta})(\mathcal{T}^{\theta}_{\tau} - \gamma^{\theta}), & \text{if } \mathcal{T}^{\theta}_{\tau} > \gamma^{\theta} \\ \alpha^{\theta}(\mathcal{T}^{\theta}_{\tau} - \delta^{\theta}), & \text{if } \mathcal{T}^{\theta}_{\tau} \leq \gamma^{\theta} \text{ and } \mathcal{T}^{\theta}_{\tau} > \delta^{\theta} \\ 0, & \text{otherwise} \end{cases}, \tag{4.24}$$

where α^{θ} is a penalty cost applied if the value of $\mathcal{T}^{\theta}_{\tau}$ is greater than an upper bound, γ^{θ} and β^{θ} is a further penalty cost applied if the value of $\mathcal{T}^{\theta}_{\tau}$ is also above a greater upper bound, δ^{θ}. This cost function is used if resource θ tracks a value corresponding to rule violations if above a certain upper bound and a large rule violation if above a larger upper bound.

The final resource cost function is given by

$$C^{\exp}: \qquad g^{\theta}_{\tau}(\mathcal{T}^{\theta}_{\tau}) = (\alpha^{\theta})^{\mathcal{T}^{\theta}_{\tau}} \tag{4.25}$$

where α^{θ} is the base of an exponential cost. This cost function is used if resource θ tracks a value where we simultaneously prefer lower values but also require fairness. For examples of the fifth, sixth and seventh resource cost functions, see §4.3.2.

4.2.4 Entity feasibility

Likewise, from the resource vectors we have calculated for any given on-stretch, work-stretch or roster-line $\tau \in \mathcal{O} \cup \mathcal{W} \cup \mathcal{R}$, we can calculate whether a given entity is feasible.

An entity τ is feasible if its corresponding resource vector $\mathcal{T}_{\tau}$ is feasible, i.e., $\mathcal{F}(\mathcal{T}_{\tau}) = \text{True}$. We say that a resource vector is feasible if every resource variable $\mathcal{T}^{\theta}_{\tau}$ in the resource vector is feasible, i.e., $\mathcal{F}(\mathcal{T}_{\tau}) = \text{True} \iff \mathcal{F}(\mathcal{T}^{\theta}_{\tau}) = \text{True}\ \forall \theta \in \Theta_{\tau}$.

We have four functions used to calculate feasibility for a given resource variable. The first feasibility function, F^{all}, is used in cases where the resource variable is always feasible, i.e.,

$$F^{all}: \qquad \mathcal{F}(\mathcal{T}^{\theta}_{\tau}) = \text{True} \iff 0 \leq \mathcal{T}^{\theta}_{\tau} \leq \infty \tag{4.26}$$

Because there is no complex infeasibility in the INRC problems, all resources in these problems have the F^{all} feasibility function.

The Waikato DHB problems have contractual rules with legal consequences for being broken. Thus, we model these rules with feasibility. We use a feasibility function if a given resource variable is infeasible if it is below a lower bound, above an upper bound or both, i.e.,

$$F^{lb}: \qquad \mathcal{F}(\mathcal{T}^{\theta}_{\tau}) = \text{True} \iff \mathcal{T}^{\theta}_{\tau} \geq \mathcal{T}^{\theta}_{\tau,\text{lower}} \tag{4.27}$$

$$F^{ub}: \qquad \mathcal{F}(\mathcal{T}^{\theta}_{\tau}) = \text{True} \iff \mathcal{T}^{\theta}_{\tau} \leq \mathcal{T}^{\theta}_{\tau,\text{upper}} \tag{4.28}$$

$$F^{lbub}: \qquad \mathcal{F}(\mathcal{T}^{\theta}_{\tau}) = \text{True} \iff \mathcal{T}^{\theta}_{\tau,\text{lower}} \leq \mathcal{T}^{\theta}_{\tau} \leq \mathcal{T}^{\theta}_{\tau,\text{upper}} \tag{4.29}$$

For examples of all four of these resource feasibility functions, please see §4.3.2.

4.2.5 Standard entity dominance

Entities are compared in terms of dominance, and entities that can be replaced by another entity in any feasible full roster-line without decreasing the cost of the line or rendering the line infeasible are discarded (please consult §5.2.1 for further details). We model Genie++'s original dominance rule using our novel terminology as follows:

Definition 1. *Suppose that two on-stretch, work-stretch, or roster-line entities are of the same type with the same initial and final days—entity, $\tau^1 \in \mathcal{O} \cup \mathcal{W} \cup \mathcal{R}$, with the associated resource vector, $\mathcal{T}_{\tau^1}$, and entity, $\tau^2 \in \mathcal{O} \cup \mathcal{W} \cup \mathcal{R}$, with the associated resource vector, $\mathcal{T}_{\tau^2}$. Then, τ^2 is said to be dominated by τ^1 if the cost, $f_{\tau}(\tau^1)$, of τ^1 is less than the cost, $f_{\tau}(\tau^2)$, of τ^2 and*

each resource variable, $\mathcal{T}^{\theta}_{\tau^1}$, in τ^1's resource vector is dominated by the corresponding resource variable, $\mathcal{T}^{\theta}_{\tau^2}$, in entity τ^2's resource vector. In other words,

$$\mathcal{T}^{\theta}_{\tau^1} \preceq \mathcal{T}^{\theta}_{\tau^2} \; \forall \theta \in \Theta_\tau \quad \text{and} \quad f_\tau(\tau^1) \leq f_\tau(\tau^2) \Rightarrow \tau^1 \preceq \tau^2 \tag{4.30}$$

where $\tau^1 \preceq \tau^2$ denotes that entity 2 is dominated by entity 1 and $\mathcal{T}^{\theta}_{\tau^1} \preceq \mathcal{T}^{\theta}_{\tau^2}$ denotes that the resource variable, $\mathcal{T}^{\theta}_{\tau^2}$, is dominated by the resource variable, $\mathcal{T}^{\theta}_{\tau^1}$.

If we consider the cost of each entity to be a separate resource, this definition is standard for SPPRCs; consult Irnich and Desaulniers (2005). To each resource, $\theta \in \Theta_{\mathrm{O}} \cup \Theta_{\mathrm{W}} \cup \Theta_{\mathrm{R}}$, we assign an individual dominance rule.

To define which resource variables dominate other resource variables, we provide an individual dominance rule for each resource $\theta \in \Theta_{\mathrm{O}} \cup \Theta_{\mathrm{W}} \cup \Theta_{\mathrm{R}}$. The first rule, *equality dominance*, asserts that for any two entities, $\tau^1 \in \mathcal{O} \cup \mathcal{W} \cup \mathcal{R}$ and $\tau^2 \in \mathcal{O} \cup \mathcal{W} \cup \mathcal{R}$, of the same type, the resource variable, $\mathcal{T}^{\theta}_{\tau^2}$, is dominated by the resource variable, $\mathcal{T}^{\theta}_{\tau^1}$, only if their values are equal. In other words,

$$D^{=}: \qquad \mathcal{T}^{\theta}_{\tau^1} \preceq \mathcal{T}^{\theta}_{\tau^2} \iff \mathcal{T}^{\theta}_{\tau^1} = \mathcal{T}^{\theta}_{\tau^2}. \tag{4.31}$$

This rule is applicable when it is unclear whether a higher or lower value for θ will lead to a lower resource cost or "less infeasibility" in any full roster-line. Note, we define "less infeasibility" as having fewer required changes to the resource variables to become feasible. For example, consider a resource that captures the number of shifts worked in an on-stretch. In the INRC problems, this resource influences the resource representing the number of shifts worked in a roster-line, which possesses the cost function, C^{LBUB}. Therefore, in a full roster-line involving multiple shifts, a shorter on-stretch may induce a lower cost. In contrast, in a full roster-line involving few shifts, a shorter on-stretch may yield a higher cost. Thus, we use equality dominance for this resource.

The second rule, *less than dominance*, asserts that the resource variable, $\mathcal{T}^{\theta}_{\tau^2}$, is dominated by the resource variable, $\mathcal{T}^{\theta}_{\tau^1}$, only if $\mathcal{T}^{\theta}_{\tau^1} \leq \mathcal{T}^{\theta}_{\tau^2}$. In other words,

$$D^{\leq}: \qquad \mathcal{T}^{\theta}_{\tau^1} \preceq \mathcal{T}^{\theta}_{\tau^2} \iff \mathcal{T}^{\theta}_{\tau^1} \leq \mathcal{T}^{\theta}_{\tau^2}. \tag{4.32}$$

This rule is applicable when lower values of θ always induce a lower resource cost or less infeasibility in any full roster-line. For example, consider a resource with the cost function, C^{UB}, and a non-decreasing REF, e.g., the resource variable $\mathcal{T}^{\theta}_{R}$ has a non-decreasing REF if $\mathcal{T}^{\theta}_{R} \leq \mathcal{E}^{\theta}_{\mathrm{R}}(\mathcal{T}_R, \mathcal{T}_W)$, for any roster-line resource vector, $\mathcal{T}_R$ and work-stretch resource vector, $\mathcal{T}_W$. An example of a resource with a non-decreasing REF is the total number of shifts in a roster-line; the value of this resource can never decrease as the roster-line is extended. Evidently, a lower value of a resource with a non-decreasing REF and the cost function, C^{UB}, always induces a lower cost in any full roster-line as the value of the resource can only increase as the entity is extended. Thus, we use less than dominance for this resource.

Our third rule, *greater than dominance*, indicates that resource variable $\mathcal{T}^{\theta}_{\tau^2}$ is dominated by resource variable $\mathcal{T}^{\theta}_{\tau^1}$ only when $\mathcal{T}^{\theta}_{\tau^1} \geq \mathcal{T}^{\theta}_{\tau^2}$, i.e.,

$$D^{\geq}: \qquad \mathcal{T}^{\theta}_{\tau^1} \preceq \mathcal{T}^{\theta}_{\tau^2} \iff \mathcal{T}^{\theta}_{\tau^1} \geq \mathcal{T}^{\theta}_{\tau^2}. \tag{4.33}$$

This rule is used when higher values for resource θ always lead to a higher resource cost or more infeasibility in any full roster-line. For example, consider a resource with the feasibility function, F^{lb}, and a non-decreasing REF. A greater value for this resource will always be more likely to lead to an infeasible full roster-line as the value can only increase as the roster-line is extended. Thus, we use greater than dominance for this resource.

The fourth rule, *bounded dominance*, asserts that the resource variable, $\mathcal{T}^{\theta}_{\tau^2}$, is dominated by the resource variable, $\mathcal{T}^{\theta}_{\tau^1}$, if their values are equal, or both values are above a certain lower bound γ^{θ} and resource variable $\mathcal{T}^{\theta}_{\tau^2}$ is greater than resource variable $\mathcal{T}^{\theta}_{\tau^1}$. In other words,

$$D^{\gamma^{\theta}}: \qquad \mathcal{T}^{\theta}_{\tau^1} \preceq \mathcal{T}^{\theta}_{\tau^2} \iff \mathcal{T}^{\theta}_{\tau^1} = \mathcal{T}^{\theta}_{\tau^2} \text{ or } \gamma^{\theta} \leq \mathcal{T}^{\theta}_{\tau^1} \leq \mathcal{T}^{\theta}_{\tau^2}, \tag{4.34}$$

For example, consider a resource with a non-decreasing REF and the cost function, C^{LBUB}. If the value of this resource is smaller than the lower bound, then it is unclear whether a higher or lower value will induce a lower resource cost or less infeasibility in any full roster-line. However, if the value of this resource exceeds the lower bound, lower values always induce a lower resource cost in any full roster-line. Thus, we use bounded dominance for this resource. For examples of all four of these dominance rules, see §4.3.2.

4.2.6 Roster history

We modelled the INRC problems as if the nurses had worked no shifts prior to this roster. However, when building real rosters, this is unrealistic. For example, if a nurse can work a maximum of five days in a row and has just worked five days in a row immediately preceding the first day of the roster we are building, then they can't start the roster with another five-day on-stretch.

For each Waikato DHB problem, we used the two weeks immediately preceding the roster as the roster's history. We used the roster's history to generate a set of initial resource values for any on-stretch or work-stretch that starts with the dummy shift.

For example, in the Waikato DHB problems, there is a resource tracking the number of consecutive days on within on-stretches, $\mathcal{T}^{\text{DaysOn}}_{O}$. If there is an on-stretch O^h that finishes on the last day of the roster's history, then the first on-stretch we create, $O^1 = (\text{Dummy})$, with a single dummy shift, can be considered as already part of on-stretch O^h. Thus, we give the dummy on-stretch the following value for the DaysOn resource:

$$\mathcal{T}^{\text{DaysOn}}_{O^1} = \mathcal{T}^{\text{DaysOn}}_{O^h} \tag{4.35}$$

In the literature, more complex history models use a rolling horizon to model the history of the rostering problem. Lusby et al. (2012) model a six-month problem by using a rolling time horizon with history. Their history model involves including part of the previous roster in their problem as fixed shifts. However, our history model worked fine for the Waikato DHB problems.

4.2.7 Compile-time customisation

To accelerate finding the solution to the column generation subproblem, Genie++ utilises the C++ Boost Preprocessor library to customise the column generation subproblem code during compilation. The column generation subproblem's code is generated directly based on the model description, which is in the form of a C++ header file. The model description file defines the vector of resources corresponding to each entity type, including the REF, cost function, and dominance rule associated with each resource. This functionality ensures that the code is compile-time optimised by the C++ compiler, which accelerates it by up to 10 times compared to simply reading the resource data from input files (Dohn and Mason, 2013).

4.2.8 Other column generation subproblem models

Although we have discussed our generic modelling terminology in terms of a nested model with shifts, off-stretches, on-stretches, work-stretches and roster-lines, our model can also be applied to other decompositions. For example, we could break down roster-lines into shift, day-off, week and fortnight component entities. Likewise, our model could have tasks as component entities, and these tasks could build into shifts in a separate nested subproblem; please see §3.2.2 for more details on tasks.

Although using a nested column generation subproblem is computationally efficient, it is not strictly necessary. Instead, we could solve a single SPPRC to generate roster-lines from shift and day-off entities.

4.3 Example models

In order to facilitate the understanding of the proposed formulation of Genie++'s entity model, we discuss two illustrative entity models: the model for the "medium hidden 01" INRC problem instance and the model for a Waikato DHB maternity wards problem instance.

4.3.1 Example INRC problem

The first problem instance we are describing is: "medium hidden 01". For further details regarding this problem, please refer to the official INRC website (Causmaecker, 2010). Note that because every combination of shifts and days off is feasible for the INRC problem instances, we have omitted feasibility functions in this section.

First, a new on-stretch, O', is constructed based on an existing on-stretch, O, and a shift, S, i.e., $O' = (O, S)$. We construct the resource vector

$$\mathcal{T}_{O'} = \left(\mathcal{T}_{O'}^{\text{StartDay}}, \mathcal{T}_{O'}^{\text{EndDay}}, \ldots, \mathcal{T}_{O'}^{\text{SkillsUnmet}}\right)$$

of on-stretch O' from the resource vector

$$\mathcal{T}_{O} = \left(\mathcal{T}_{O}^{\text{StartDay}}, \mathcal{T}_{O}^{\text{EndDay}}, \ldots, \mathcal{T}_{O}^{\text{SkillsUnmet}}\right)$$

of on-stretch O and the resource vector

$$\mathcal{T}_S = \left(\mathcal{T}_S^{\text{StartDay}}, \mathcal{T}_S^{\text{EndDay}}, \ldots, \mathcal{T}_S^{\text{DayOfWeek}}\right)$$

of shift S using $\mathcal{T}_{O'}^{\theta} = \mathcal{E}_{\text{O}}(\mathcal{T}_O, \mathcal{T}_S) = \left(\mathcal{E}_{\text{O}}^{\theta}(\mathcal{T}_O, \mathcal{T}_S),\ \forall\theta \in \Theta_{\text{O}}\right)$. A representative set of on-stretch resources and their associated REFs are presented in Table 4.1 along with their associated dominance rules and cost functions.

The resources associated with a shift in "medium hidden 01" include the shift's initial day, $\mathcal{T}_S^{\text{StartDay}}$; final day, $\mathcal{T}_S^{\text{EndDay}}$; type,

$$\mathcal{T}_S^{\text{ShiftType}} \in \{\text{Dummy}, \text{Morning}, \text{Afternoon}, \text{Late}, \text{Night}, \text{Head nurse}\};$$

and its day of the week, $\mathcal{T}_S^{\text{DayOfWeek}} \in \{\text{Monday}, \text{Tuesday}, \ldots, \text{Sunday}\}$.

Table 4.1 includes the resources corresponding to the initial day of an on-stretch, $\mathcal{T}_{O'}^{\text{StartDay}}$, the final day of an on-stretch, $\mathcal{T}_{O'}^{\text{EndDay}}$, and the number of days in the on-stretch, $\mathcal{T}_{O'}^{\text{DaysOn}}$. As certain REFs require knowledge of the previous two shift-types, Table 4.1 includes resources representing each of these shifts-the last shift-type resource, $\mathcal{T}_{O'}^{\text{LST}}$, and the second-to-last shift-type resource, $\mathcal{T}_{O'}^{\text{SLST}}$. Further, Table 4.1 includes the "identical shift weekend" resource, $\mathcal{T}_{O'}^{\text{ISW}}$, which represents the number of shifts worked in a non-symmetrical weekend (i.e., only one shift is worked or two shifts of different types are worked during the weekend). Table 4.1 also includes the resource, $\mathcal{T}_{O'}^{\text{P1}}$, representing the frequency of occurrence of the shift sequence pattern, (D, N, D). Although a total of eight resources are used to represent the frequencies of occurrence of certain shift sequence patterns, these resources share similar associated REFs and dominance rules. As a result, only one resource of this type has been presented in Table 4.1 as an illustration, and the remaining seven shift sequence resources have been omitted. Table 4.1 also includes the resource, $\mathcal{T}_{O'}^{\text{WW}}$, which represents the total number of full or split weekends that have been worked in the on-stretch being considered, and the resource, $\mathcal{T}_{O'}^{\text{SkillsUnmet}}$, which represents the total number of head-nurse shifts worked by a regular nurse.

Next a work-stretch, W, is constructed based on an on-stretch, O, and an off-stretch, F, i.e., $W = (O, F)$. We construct the resource vector

$$\mathcal{T}_W = \left(\mathcal{T}_W^{\text{StartDay}}, \mathcal{T}_W^{\text{EndDay}}, \ldots, \mathcal{T}_W^{\text{WeekendsOff}}\right)$$

of work-stretch W from the resource vector

$$\mathcal{T}_O = \left(\mathcal{T}_O^{\text{StartDay}}, \mathcal{T}_O^{\text{EndDay}}, \ldots, \mathcal{T}_O^{\text{SkillsUnmet}}\right)$$

of on-stretch O and the resource vector

$$\mathcal{T}_F = \left(\mathcal{T}_F^{\text{StartDay}}, \mathcal{T}_F^{\text{EndDay}}, \ldots, \mathcal{T}_F^{\text{SinglesOff}}\right)$$

of off-stretch F using the vector of REFs $\mathcal{E}_{\text{W}}(\mathcal{T}_O, \mathcal{T}_F) = \left(\mathcal{E}_{\text{W}}^{\theta}(\mathcal{T}_O, \mathcal{T}_F),\ \forall\theta \in \Theta_{\text{W}}\right)$. The full set of work-stretch resources and their associated REFs are presented in Table 4.2 along with their associated dominance rules and cost functions.

Resources for O'	REF: $\mathcal{E}^{\theta}_{\mathrm{O}}(\mathcal{T}_O, \mathcal{T}_S)$		Dom.	Cost
$\mathcal{T}^{\mathrm{StartDay}}_{O'}$	$\mathcal{T}^{\mathrm{StartDay}}_{O}$		$D^{=}$	$C^{\emptyset}$
$\mathcal{T}^{\mathrm{EndDay}}_{O'}$	$\mathcal{T}^{\mathrm{EndDay}}_{O} + 1$		$D^{=}$	$C^{\emptyset}$
$\mathcal{T}^{\mathrm{DaysOn}}_{O'}$	$\mathcal{T}^{\mathrm{DaysOn}}_{O} + 1$		$D^{=}$	C^{LBUB}
$\mathcal{T}^{\mathrm{LST}}_{O'}$	$\mathcal{T}^{\mathrm{ShiftType}}_{S}$		$D^{=}$	$C^{\emptyset}$
$\mathcal{T}^{\mathrm{SLST}}_{O'}$	$\mathcal{T}^{\mathrm{LST}}_{O}$		$D^{=}$	$C^{\emptyset}$
$\mathcal{T}^{\mathrm{ISW}}_{O'}$	$\mathcal{T}^{\mathrm{ISW}}_{O}$ $\mathcal{T}^{\mathrm{ISW}}_{O} + 1$ $\mathcal{T}^{\mathrm{ISW}}_{O} - 1$ $\mathcal{T}^{\mathrm{ISW}}_{O} + 1$	if $\mathcal{T}^{\mathrm{DayOfWeek}}_{S}$ *is weekday* if $\mathcal{T}^{\mathrm{DayOfWeek}}_{S} =$ *Saturday* if $\mathcal{T}^{\mathrm{DayOfWeek}}_{S} =$ *Sunday* & $\mathcal{T}^{\mathrm{ShiftType}}_{S} = \mathcal{T}^{\mathrm{SLST}}_{O}$ if $\mathcal{T}^{\mathrm{DayOfWeek}}_{S} =$ *Sunday* & $\mathcal{T}^{\mathrm{ShiftType}}_{S} \neq \mathcal{T}^{\mathrm{SLST}}_{O}$	$D^{=}$	C^{linear}
$\mathcal{T}^{\mathrm{P1}}_{O'}$	$\mathcal{T}^{\mathrm{P1}}_{O} + 1$ $\mathcal{T}^{\mathrm{P1}}_{O'}$	if $(\mathcal{T}^{\mathrm{ShiftType}}_{S}, \mathcal{T}^{\mathrm{LST}}_{O}, \mathcal{T}^{\mathrm{SLST}}_{O})$ *is* $(day, night, day)$ otherwise	$D^{\leq}$	C^{linear}
$\mathcal{T}^{\mathrm{WW}}_{O'}$	$\mathcal{T}^{\mathrm{WW}}_{O} + 1$ $\mathcal{T}^{\mathrm{WW}}_{O}$	if $\mathcal{T}^{\mathrm{DayOfWeek}}_{S} =$ *Saturday* otherwise	$D^{=}$	$C^{\emptyset}$
$\mathcal{T}^{\mathrm{SkillsUnmet}}_{O'}$	$\mathcal{T}^{\mathrm{SkillsUnmet}}_{O} + 1$ $\mathcal{T}^{\mathrm{SkillsUnmet}}_{O}$	if employee $\neq$ head nurse & $\mathcal{T}^{\mathrm{ShiftType}}_{S} =$ head nurse otherwise	$D^{\leq}$	C^{linear}

Table 4.1: Representative set of on-stretch resources corresponding to the INRC problem, "medium hidden 01". For each resource, θ, the table includes the corresponding resource variable, $\mathcal{T}^{\theta}_{O'}$, associated REF, $\mathcal{E}^{\theta}_{\mathrm{O}}(\mathcal{T}_O, \mathcal{T}_S)$, cost function, and dominance rule. One shift sequence pattern, "P1", has been included, but the remaining 7 have been omitted as their dominance rules and cost functions are identical to those of "P1" and their REFs are similar.

The set of off-stretches are pre-enumerated using associated resources. In this problem, they include resource variables representing the off-stretch's initial day, $\mathcal{T}^{\mathrm{StartDay}}_{F}$; its final day, $\mathcal{T}^{\mathrm{EndDay}}_{F}$; its total number of off-days, $\mathcal{T}^{\mathrm{DaysOff}}_{F}$; its total number of off weekends, $\mathcal{T}^{\mathrm{WeekendsOff}}_{F}$; and its number of single off-days, $\mathcal{T}^{\mathrm{SinglesOff}}_{F}$.

Table 4.2 includes the resource variables: $\mathcal{T}^{\mathrm{StartDay}}_{W}$, $\mathcal{T}^{\mathrm{EndDay}}_{W}$, $\mathcal{T}^{\mathrm{DaysOn}}_{W}$, and $\mathcal{T}^{\mathrm{WW}}_{W}$, which are similar to their on-stretch counterparts. The resource variables, $\mathcal{T}^{\mathrm{WeekendsOff}}_{W}$ and $\mathcal{T}^{\mathrm{DaysOff}}_{W}$, are similar to their off-stretch counterparts. The incomplete weekend resource variable, $\mathcal{T}^{\mathrm{IncompleteWeekend}}_{W}$, calculates the number of half-worked weekends in the work-stretch.

Finally, a new roster-line, R', is constructed based on a work-stretch, W, and a shorter roster-line R, i.e., $R' = (R, W)$. We construct the resource vector

$$\mathcal{T}_{R'} = \left(\mathcal{T}^{\mathrm{EndDay}}_{R'}, \mathcal{T}^{\mathrm{DaysOn}}_{R'}, \ldots, \mathcal{T}^{\mathrm{P2}}_{R'}\right)$$

of roster-line R' from the resource vector

$$\mathcal{T}_{R} = \left(\mathcal{T}^{\mathrm{EndDay}}_{R}, \mathcal{T}^{\mathrm{DaysOn}}_{R}, \ldots, \mathcal{T}^{\mathrm{P2}}_{R}\right)$$

Resources for W	REF: $\mathcal{E}_{\mathrm{W}}^{\theta}(\mathcal{T}_O, \mathcal{T}_F)$	Dom.	Cost
$\mathcal{T}_W^{\mathrm{StartDay}}$	$\mathcal{T}_O^{\mathrm{StartDay}}$	$D^{=}$	$C^{\emptyset}$
$\mathcal{T}_W^{\mathrm{EndDay}}$	$\mathcal{T}_F^{\mathrm{EndDay}}$	$D^{=}$	$C^{\emptyset}$
$\mathcal{T}_W^{\mathrm{DaysOn}}$	$\mathcal{T}_O^{\mathrm{DaysOn}}$	$D^{=}$	$C^{\emptyset}$
$\mathcal{T}_W^{\mathrm{DaysOff}}$	$\mathcal{T}_F^{\mathrm{DaysOff}}$	$D^{=}$	C^{LBUB}
$\mathcal{T}_W^{\mathrm{WeekendsOff}}$	$\mathcal{T}_F^{\mathrm{WeekendsOff}}$	$D^{=}$	$C^{\emptyset}$
$\mathcal{T}_W^{\mathrm{IncompleteWeekend}}$	$(\mathcal{T}_O^{\mathrm{StartDay}} = \mathit{Sunday}) + (\mathcal{T}_O^{\mathrm{EndDay}}\ \mathit{on\ Saturday})$	$D^{\leq}$	$C^{\emptyset}$
$\mathcal{T}_W^{\mathrm{WW}}$	$\mathcal{T}_O^{\mathrm{WW}}$	$D^{=}$	C^{linear}

Table 4.2: Full set of work-stretch resources in the INRC problem, "medium hidden 01". For each resource, θ, the table includes the corresponding resource variable, $\mathcal{T}_W^{\theta}$, associated REF, $\mathcal{E}_{\mathrm{W}}^{\theta}(\mathcal{T}_O, \mathcal{T}_F)$, cost function, and dominance rule.

of roster-line R and the resource vector

$$\mathcal{T}_W = \left(\mathcal{T}_W^{\mathrm{StartDay}}, \mathcal{T}_W^{\mathrm{EndDay}}, \ldots, \mathcal{T}_W^{\mathrm{WeekendsOff}}\right)$$

of work-stretch W using the vector of REFs $\mathcal{E}_{\mathrm{R}}(\mathcal{T}_R, \mathcal{T}_W) = \left(\mathcal{E}_{\mathrm{R}}^{\theta}(\mathcal{T}_R, \mathcal{T}_W),\ \forall\theta \in \Theta_{\mathrm{R}}\right)$. The full set of roster-line resources and their associated REFs is presented in Table 4.2 along with their associated dominance rules and cost functions.

Table 4.2 includes the final day resource variable, $\mathcal{T}_{R'}^{\mathrm{EndDay}}$; and the final consecutive weekends resource variable, $\mathcal{T}_{R'}^{\mathrm{ECW}}$, which represents the number of consecutive weekends worked, counting backwards from the last weekend in the roster-line, R'. The final consecutive weekends resource variable, $\mathcal{T}_{R'}^{\mathrm{ECW}}$, is used in the REF of both the maximum number of consecutive weekends worked resource variable, $\mathcal{T}_{R'}^{\mathrm{MCW}}$, and the number of lone weekends resource variable, $\mathcal{T}_{R'}^{\mathrm{LoneWeekend}}$, which represents the number of work-weekends that are not followed or preceded by a work-weekend. Finally, the table includes the resource variable, $\mathcal{T}_{R'}^{\mathrm{OneDayOff}}$, representing the number of single off-days. This is used to calculate the second shift sequence pattern, $\mathcal{T}_{R'}^{\mathrm{P2}}$, which represents the number of times no shifts are worked on Friday, but at least one shift is worked on the following weekend.

4.3.2 Example Waikato DHB problem

The second problem instance we are describing is the Waikato DHB maternity wards problem (MWP). The set of resources in this problem is a super-set of the resources in the Waikato DHB operating theatre ward problem.

Resources for R'	REF: $\mathcal{E}^{\theta}_{\mathrm{R}}(\mathcal{T}_R, \mathcal{T}_W)$		Dom.	Cost
$\mathcal{T}^{\mathrm{EndDay}}_{R'}$	$\mathcal{T}^{\mathrm{EndDay}}_{W}$		$D^{=}$	$C^{\emptyset}$
$\mathcal{T}^{\mathrm{DaysOn}}_{R'}$	$\mathcal{T}^{\mathrm{DaysOn}}_{R} + \mathcal{T}^{\mathrm{DaysOn}}_{W}$		$D^{\gamma^{\theta}}$	C^{LBUB}
$\mathcal{T}^{\mathrm{ECW}}_{R'}$	0 $\mathcal{T}^{\mathrm{ECW}}_{R} + \mathcal{T}^{\mathrm{WW}}_{W}$	if $\mathcal{T}^{\mathrm{WeekendsOff}}_{W} > 0$ otherwise	$D^{=}$	$C^{\emptyset}$
$\mathcal{T}^{\mathrm{MCW}}_{R'}$	$\mathcal{T}^{\mathrm{ECW}}_{R} + \mathcal{T}^{\mathrm{WW}}_{W}$ $\mathcal{T}^{\mathrm{MCW}}_{R}$	if$\mathcal{T}^{\mathrm{ECW}}_{R} + \mathcal{T}^{\mathrm{WW}}_{W} \geq 2$ and $\mathcal{T}^{\mathrm{ECW}}_{R} + \mathcal{T}^{\mathrm{WW}}_{W} > \mathcal{T}^{\mathrm{MCW}}_{R}$ otherwise	$D^{\leq}$	C^{UB}
$\mathcal{T}^{\mathrm{LoneWeekend}}_{R'}$	$\mathcal{T}^{\mathrm{LoneWeekend}}_{R} + 1$ $\mathcal{T}^{\mathrm{LoneWeekend}}_{R}$,	if $\mathcal{T}^{\mathrm{ECW}}_{R} + \mathcal{T}^{\mathrm{WW}}_{W} = 1$ and $\mathcal{T}^{\mathrm{WeekendsOff}}_{W} > 0$ otherwise	$D^{\leq}$	C^{linear}
$\mathcal{T}^{\mathrm{OneDayOff}}_{R'}$	true false	if $\mathcal{T}^{\mathrm{DaysOff}}_{W} = 1$ otherwise	$D^{\leq}$	C^{linear}
$\mathcal{T}^{\mathrm{P2}}_{R'}$	$\mathcal{T}^{\mathrm{P2}}_{R} + 1$ $\mathcal{T}^{\mathrm{P2}}_{R}$	if ($\mathcal{T}^{\mathrm{EndDay}}_{R}$ *is Friday* or $\mathcal{T}^{\mathrm{OneDayOff}}_{R} = False$) and $\mathcal{T}^{\mathrm{StartDay}}_{W}$ *is Weekend* otherwise	$D^{\leq}$	C^{linear}

Table 4.3: Full set of roster-line resources for the INRC problem, "medium hidden 01". For each resource, θ, the table includes the corresponding resource variable, $\mathcal{T}^{\theta}_{R'}$, associated REF, $\mathcal{E}^{\theta}_{\mathrm{R}}(\mathcal{T}_R, \mathcal{T}_W)$, cost function, and dominance rule.

The resources associated with a shift in the MWP include the shift's initial time, $\mathcal{T}^{\mathrm{StartTime}}_{S}$; final time, $\mathcal{T}^{\mathrm{EndTime}}_{S}$; type, $\mathcal{T}^{\mathrm{ShiftType}}_{S}$; day of the week, $\mathcal{T}^{\mathrm{DayOfWeek}}_{S} \in \{\mathrm{Monday}, \mathrm{Tuesday}, \ldots, \mathrm{Sunday}\}$ and ward $\mathcal{T}^{\mathrm{ward}}_{S}$.

The resources associated with an on-stretch in the MWP shown in Table 4.4 include the on-stretch's initial time, $\mathcal{T}^{\mathrm{StartTime}}_{O'}$; final time, $\mathcal{T}^{\mathrm{EndDay}}_{O'}$; total number of worked hours, $\mathcal{T}^{\mathrm{HoursOn}}_{O'}$; total number of hours worked in fortnight x, $\mathcal{T}^{\mathrm{F}x\mathrm{Hours}}_{O'}$; total number of hours worked in week x, $\mathcal{T}^{\mathrm{W}x\mathrm{Hours}}_{O'}$; type of the last shift worked, $\mathcal{T}^{\mathrm{LST}}_{O'}$; number of times there are different shift types worked on consecutive days, $\mathcal{T}^{\mathrm{TotalShiftChanges}}_{O'}$; number of different wards worked, $\mathcal{T}^{\mathrm{NumDifferentWards}}_{O'}$, whether the on-stretch starts on a Sunday or ends on a Saturday, $\mathcal{T}^{\mathrm{StartsSunday}}_{O'}$ and $\mathcal{T}^{\mathrm{EndsSaturday}}_{O'}$; total number of PM and night shifts, $\mathcal{T}^{\mathrm{TotalPM}}_{O'}$ and $\mathcal{T}^{\mathrm{TotalN}}_{O'}$; number of times an education shift is worked immediately followed by a PM shift, $\mathcal{T}^{\mathrm{EToPM}}_{O'}$; and number of night shifts worked in week x, $\mathcal{T}^{\mathrm{W}x\mathrm{HasN}}_{O'}$.

In some cases, we reworked the rules provided to us by Waikato DHB to work more efficiently within our on-stretch/off-stretch/work-stretch model. For example, initially, the staff at Waikato DHB told us they wanted a maximum of two different shift types in any given week. We would have to model this with separate resources in on-stretches, work-stretches and roster-lines to model which shift types have already been worked each week. We would also need roster-line resources to track the maximum number of shift types worked each week. We instead approximated this rule with a single resource, the maximum number of shift changes resource, $\mathcal{T}^{\mathrm{TotalShiftChanges}}_{O'}$. Using this resource, we could produce roster solutions that still met their requirements. Further, we could model this rule purely within the on-stretch entity.

Resources for O'	REF: $\mathcal{E}_{\mathrm{O}}^{\theta}(\mathcal{T}_O, \mathcal{T}_S)$	Dom.	Cost	Feas.
$\mathcal{T}_{O'}^{\text{StartTime}}$	$\mathcal{T}_O^{\text{StartTime}}$	$D^{=}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_{O'}^{\text{EndTime}}$	$\mathcal{T}_S^{\text{EndTime}}$	$D^{=}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_{O'}^{\text{HoursOn}}$	$\mathcal{T}_O^{\text{HoursOn}} + \mathcal{T}_S^{\text{HoursOn}}$	$D^{=}$	C^{UB}	F^{ub}
$\mathcal{T}_{O'}^{\text{F}x\text{Hours}}$	$\mathcal{T}_O^{\text{F}x\text{Hours}} + \mathcal{T}_S^{\text{F}x\text{Hours}}$	$D^{=}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_{O'}^{\text{W}x\text{Hours}}$	$\mathcal{T}_O^{\text{W}x\text{Hours}} + \mathcal{T}_S^{\text{W}x\text{Hours}}$	$D^{=}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_{O'}^{\text{LST}}$	$\mathcal{T}_S^{\text{ShiftType}}$	$D^{=}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_{O'}^{\text{TotalShiftChanges}}$	$\mathcal{T}_O^{\text{TotalShiftChanges}} + \begin{cases} 1, & \mathcal{T}_O^{\text{LastShiftType}} \neq \mathcal{T}_S^{\text{ShiftType}} \\ 0, & \text{otherwise} \end{cases}$	$D^{\leq}$	C^{UB}	F^{all}
$\mathcal{T}_{O'}^{\text{NumDifferentWards}}$	$\mathcal{T}_O^{\text{NumDifferentWards}} + \begin{cases} 1, & \mathcal{T}_S^{\text{Ward}} \text{ is not employee's ward} \\ 0, & \text{otherwise} \end{cases}$	$D^{\leq}$	$C^{\emptyset}$	F^{ub}
$\mathcal{T}_{O'}^{\text{StartsSunday}}$	$\mathcal{T}_O^{\text{StartsSunday}}$	$D^{=}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_{O'}^{\text{EndsSaturday}}$	$\mathcal{T}_S^{\text{DayType}}$ is Saturday	$D^{=}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_{O'}^{\text{TotalPM}}$	$\mathcal{T}_O^{\text{TotalPM}} + \begin{cases} 1, & \mathcal{T}_S^{\text{Shift Type}} \text{ is PM} \\ 0, & \text{otherwise} \end{cases}$	$D^{\leq}$	C^{UB}	F^{all}
$\mathcal{T}_{O'}^{\text{TotalN}}$	$\mathcal{T}_O^{\text{TotalN}} + \begin{cases} 1, & \mathcal{T}_S^{\text{Shift Type}} \text{ is N} \\ 0, & \text{otherwise} \end{cases}$	$D^{\leq}$	$C^{\emptyset}$	F^{ub}
$\mathcal{T}_{O'}^{\text{EToPM}}$	$\mathcal{T}_O^{\text{EToP}} + \begin{cases} 1, & \mathcal{T}_S^{\text{Shift Type}} \text{ is PM and } \mathcal{T}_O^{\text{Last shift type}} \text{ is E} \\ 0, & \text{otherwise} \end{cases}$	$D^{\leq}$	C^{linear}	F^{all}
$\mathcal{T}_{O'}^{\text{W}x\text{HasN}}$	$\begin{cases} 1, & \mathcal{T}_O^{\text{W}x\text{HasN}} = 1 \text{ or } (\mathcal{T}_S^{\text{Shift Type}} \text{ is N and } \mathcal{T}_S^{\text{Week}} = x) \\ 0, & \text{otherwise} \end{cases}$	$D^{\leq}$	$C^{\emptyset}$	F^{all}

Table 4.4: Representative set of on-stretch resources corresponding to the MWP. For each resource, θ, the table includes the corresponding resource variable, $\mathcal{T}_{O'}^{\theta}$, associated REF, $\mathcal{E}_{\mathrm{O}}^{\theta}(\mathcal{T}_O, \mathcal{T}_S)$, cost function, dominance rule and feasibility rule. One shift sequence pattern, "P1", has been included, but the remaining 7 have been omitted as their dominance rules and cost functions are identical to those of "P1" and their REFs are similar.

The resources associated with an off-stretch in the MWP include the off-stretch's start time, $\mathcal{T}_F^{\text{StartTime}}$; final time, $\mathcal{T}_F^{\text{EndTime}}$; total number of off-days, $\mathcal{T}_F^{\text{DaysOff}}$; number of off-days in week x, $\mathcal{T}_F^{\text{WxDaysOff}}$; and number of whole weekends off, $\mathcal{T}_F^{\text{WholeWeekendsOff}}$.

The resources associated with a work-stretch in the MWP include: $\mathcal{T}_W^{\text{StartTime}}$, $\mathcal{T}_W^{\text{NumDifferentWards}}$, $\mathcal{T}_W^{\text{F}x\text{Hours}}$, $\mathcal{T}_W^{\text{W}x\text{Hours}}$, $\mathcal{T}_W^{\text{TotalN}}$, and $\mathcal{T}_W^{\text{W}x\text{HasN}}$ which are similar to their on-stretch counterparts. They also include $\mathcal{T}_W^{\text{EndTime}}$, $\mathcal{T}_W^{\text{WxDaysOff}}$ and $\mathcal{T}_W^{\text{WholeWeekendsOff}}$ which are similar to their off-stretch counterparts. There are also resources representing the number of half-worked weekends in the work-stretch, $\mathcal{T}_W^{\text{SplitWeekend}}$; whether there is a 48 hour consecutive break,

$\mathcal{T}_W^{\text{WxConsecutive48hOff}}$; having only a single day off after working a night shift, $\mathcal{T}_W^{\text{OneDayOffAfterNight}}$; working only a single day on, $\mathcal{T}_W^{\text{SingleDayOn}}$; and having only a single day off, $\mathcal{T}_W^{\text{SingleDayOff}}$.

Resources for W	REF: $\mathcal{E}_{\text{W}}^{\theta}(\mathcal{T}_O, \mathcal{T}_F)$	Dom.	Cost	Feas.
$\mathcal{T}_W^{\text{StartTime}}$	$\mathcal{T}_O^{\text{StartTime}}$	$D^{=}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_W^{\text{EndTime}}$	$\mathcal{T}_F^{\text{EndTime}}$	$D^{=}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_W^{\text{F}x\text{Hours}}$	$\mathcal{T}_O^{\text{F}x\text{Hours}}$	$D^{=}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_W^{\text{W}x\text{Hours}}$	$\mathcal{T}_O^{\text{W}x\text{Hours}}$	$D^{\leq}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_W^{\text{NumDifferentWards}}$	$\mathcal{T}_O^{\text{NumDifferentWards}}$	$D^{\leq}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_W^{\text{TotalN}}$	$\mathcal{T}_O^{\text{TotalN}}$	$D^{\leq}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_W^{\text{W}x\text{HasN}}$	$\mathcal{T}_O^{\text{W}x\text{HasN}}$	$D^{\leq}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_W^{\text{SplitWeekend}}$	$\mathcal{T}_O^{\text{StartsSunday}} + \mathcal{T}_O^{\text{EndsSaturday}}$	$D^{\leq}$	C^{linear}	F^{all}
$\mathcal{T}_W^{\text{WxConsecutive48hOff}}$	$\mathcal{T}_F^{\text{WxDaysOff}} \geq 2$	$D^{\geq}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_W^{\text{WxDaysOff}}$	$\mathcal{T}_F^{\text{WxDaysOff}}$	$D^{\geq}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_W^{\text{WholeWeekendsOff}}$	$\mathcal{T}_F^{\text{WholeWeekendsOff}}$	$D^{\geq}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_W^{\text{OneDayOffAfterNight}}$	$\mathcal{T}_O^{\text{LastShiftType}}$ is N and $\mathcal{T}_F^{\text{DaysOff}} = 1$	$D^{\leq}$	C^{linear}	F^{all}
$\mathcal{T}_W^{\text{SingleDayOn}}$	$\mathcal{T}_O^{\text{DaysOn}} = 1$	$D^{\leq}$	C^{linear}	F^{all}
$\mathcal{T}_W^{\text{SingleDayOff}}$	$\mathcal{T}_F^{\text{DaysOff}} = 1$	$D^{\leq}$	C^{linear}	F^{all}

Table 4.5: Full set of work-stretch resources in the MWP. For each resource, θ, the table includes the corresponding resource variable, $\mathcal{T}_W^{\theta}$, associated REF, $\mathcal{E}_{\text{W}}^{\theta}(\mathcal{T}_O, \mathcal{T}_F)$, cost function, dominance rule and feasibility rule.

The resources associated with a roster-line in the MWP shown in Table 4.3 include the roster-lines's final time, $\mathcal{T}_{R'}^{\text{EndTime}}$; number of hours worked in a fortnight x, $\mathcal{T}_{R'}^{\text{F}x\text{Hours}}$; number of hours worked in week x, $\mathcal{T}_{R'}^{\text{W}x\text{Hours}}$; total number of different wards worked, $\mathcal{T}_{R'}^{\text{NumDifferentWards}}$; total number of night shifts worked, $\mathcal{T}_{R'}^{\text{TotalN}}$; whether there is a consecutive 48 hour break in week x, $\mathcal{T}_{R'}^{\text{WxConsecutive48hOff}}$; total number of weeks containing a consecutive 48 hour break, $\mathcal{T}_{R'}^{\text{TotalConsec48hOffWeeks}}$; total number of days off in week x, $\mathcal{T}_{R'}^{\text{WxDaysOff}}$; whether there is a whole weekend off, $\mathcal{T}_{R'}^{\text{WholeWeekendOff}}$; whether a night shift is worked in week x, $\mathcal{T}_{R'}^{\text{W}x\text{HasN}}$; and whether there is a two week break between working night shifts, $\mathcal{T}_{R'}^{\text{DenseN}}$.

Resources for R'	REF: $\mathcal{E}_{\mathrm{R}}^{\theta}(\mathcal{T}_R, \mathcal{T}_W)$	Dom.	Cost	Feas.
$\mathcal{T}_{R'}^{\mathrm{EndTime}}$	$\mathcal{T}_W^{\mathrm{EndTime}}$	$D^{=}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_{R'}^{\mathrm{F}x\mathrm{Hours}}$	$\mathcal{T}_R^{\mathrm{F}x\mathrm{Hours}} + \mathcal{T}_W^{\mathrm{F}x\mathrm{Hours}}$	$D^{=}$	$C^{\emptyset}$	F^{lbub}
$\mathcal{T}_{R'}^{\mathrm{W}x\mathrm{Hours}}$	$\mathcal{T}_R^{\mathrm{W}x\mathrm{Hours}} + \mathcal{T}_W^{\mathrm{W}x\mathrm{Hours}}$	$D^{\leq}$	C^{2UB}	F^{all}
$\mathcal{T}_{R'}^{\mathrm{NumDifferentWards}}$	$\mathcal{T}_R^{\mathrm{NumDifferentWards}} + \mathcal{T}_W^{\mathrm{NumDifferentWards}}$	$D^{\leq}$	$C^{\emptyset}$	F^{ub}
$\mathcal{T}_{R'}^{\mathrm{TotalN}}$	$\mathcal{T}_R^{\mathrm{TotalN}} + \mathcal{T}_W^{\mathrm{TotalN}}$	$D^{\leq}$	C^{exp}	F^{ub}
$\mathcal{T}_{R'}^{\mathrm{WxConsecutive48hOff}}$	$\min(\mathcal{T}_R^{\mathrm{WxConsecutive48hOff}} + \mathcal{T}_W^{\mathrm{WxConsecutive48hOff}}, 1)$	$D^{\geq}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_{R'}^{\mathrm{TotalConsec48hOffWeeks}}$	$\min(\mathcal{T}_R^{\mathrm{W1Consecutive48hOff}} + \mathcal{T}_W^{\mathrm{W1Consecutive48hOff}}, 1)$ $+ \min(\mathcal{T}_R^{\mathrm{W2Consecutive48hOff}} + \mathcal{T}_W^{\mathrm{W2Consecutive48hOff}}, 1)$ $+ \min(\mathcal{T}_R^{\mathrm{W3Consecutive48hOff}} + \mathcal{T}_W^{\mathrm{W3Consecutive48hOff}}, 1)$ $+ \min(\mathcal{T}_R^{\mathrm{W4Consecutive48hOff}} + \mathcal{T}_W^{\mathrm{W4Consecutive48hOff}}, 1), 1)$	$D^{\geq}$	C^{LB}	F^{lb}
$\mathcal{T}_{R'}^{\mathrm{WxDaysOff}}$	$\min(\mathcal{T}_R^{\mathrm{WxDaysOff}} + \mathcal{T}_W^{\mathrm{WxDaysOff}}, 2)$	$D^{\geq}$	C^{LB}	F^{all}
$\mathcal{T}_{R'}^{\mathrm{WholeWeekendOff}}$	$\min(\mathcal{T}_R^{\mathrm{WholeWeekendOff}} + \mathcal{T}_W^{\mathrm{WholeWeekendOff}}, 1)$	$D^{\geq}$	C^{LB}	F^{all}
$\mathcal{T}_{R'}^{\mathrm{W}x\mathrm{HasN}}$	$\mathcal{T}_R^{\mathrm{W}x\mathrm{HasN}}$	$D^{\leq}$	$C^{\emptyset}$	F^{all}
$\mathcal{T}_{R'}^{\mathrm{DenseN}}$	if $\mathcal{T}_R^{\mathrm{W00HasN}} + \mathcal{T}_R^{\mathrm{W0HasN}} + \mathcal{T}_R^{\mathrm{W1HasN}} + \min(\mathcal{T}_W^{\mathrm{W00HasN}} + \mathcal{T}_W^{\mathrm{W0HasN}} + \mathcal{T}_W^{\mathrm{W1HasN}}, 1) > 1$ or $\mathcal{T}_R^{\mathrm{W0HasN}} + \mathcal{T}_R^{\mathrm{W1HasN}} + \mathcal{T}_R^{\mathrm{W2HasN}} + \min(\mathcal{T}_W^{\mathrm{W0HasN}} + \mathcal{T}_W^{\mathrm{W1HasN}} + \mathcal{T}_W^{\mathrm{W2HasN}}, 1) > 1$ or $\mathcal{T}_R^{\mathrm{W1HasN}} + \mathcal{T}_R^{\mathrm{W2HasN}} + \mathcal{T}_R^{\mathrm{W3HasN}} + \min(\mathcal{T}_W^{\mathrm{W1HasN}} + \mathcal{T}_W^{\mathrm{W2HasN}} + \mathcal{T}_W^{\mathrm{W3HasN}}, 1) > 1$ or $\mathcal{T}_R^{\mathrm{W2HasN}} + \mathcal{T}_R^{\mathrm{W3HasN}} + \mathcal{T}_R^{\mathrm{W4HasN}} + \min(\mathcal{T}_W^{\mathrm{W2HasN}} + \mathcal{T}_W^{\mathrm{W3HasN}} + \mathcal{T}_W^{\mathrm{W4HasN}}, 1) > 1$ or $\mathcal{T}_R^{\mathrm{W3HasN}} + \mathcal{T}_R^{\mathrm{W4HasN}} + \mathcal{T}_R^{\mathrm{W5HasN}} + \min(\mathcal{T}_W^{\mathrm{W3HasN}} + \mathcal{T}_W^{\mathrm{W4HasN}} + \mathcal{T}_W^{\mathrm{W5HasN}}, 1) > 1$ then $\mathcal{T}_R^{\mathrm{DenseN}} + 1 : \mathcal{T}_R^{\mathrm{DenseN}}$	$D^{\leq}$	C^{linear}	F^{all}

Table 4.6: Full set of roster-line resources for the MWP. For each resource, θ, the table includes the corresponding resource variable, $\mathcal{T}_{R'}^{\theta}$, associated REF, $\mathcal{E}_{\mathrm{R}}^{\theta}(\mathcal{T}_R, \mathcal{T}_W)$, cost function, dominance rule, and feasibility rule.

Several of the resources we modelled appeared to have a significant effect on the difficulty of solving the problem. We modelled fairness in the number of night shifts with an exponential cost function C^{exp} and fairness in the number of hours worked in a week with the double upper bound cost function C^{2UB}.

Further, the maternity wards problem has a much larger number of entities generated in the column generation subproblem due to the additional hard-constrained resource, $\mathcal{T}_{R'}^{\mathrm{NumDifferentWards}}$. Without this constraint, we can choose to work each shift type from the ward with the largest dual, simplifying the problem.

Lastly, the maternity wards problem has 8-hour and 12-hour shifts instead of only 8-hour shifts. Different length shifts significantly increase the number of entities generated in the column generation subproblem compared to the operating theatre ward problem, which only has 8-hour shifts. This is because there are many more possible values for $\mathcal{T}_{R'}^{\mathrm{F}x\mathrm{Hours}}$. For example, if an employee works 80 hours a fortnight, then in the operating theatre problem, $\mathcal{T}_{R'}^{\mathrm{F}x\mathrm{Hours}} \in \{0, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80\}$ whereas in the maternity wards problem, $\mathcal{T}_{R'}^{\mathrm{F}x\mathrm{Hours}} \in \{0, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80\}$, which is almost twice as many possible values. $\mathcal{T}_{R'}^{\mathrm{F}x\mathrm{Hours}}$ has a hard equality constraint which means that it effectively prevents dominance if the values of this resource being compared are not equal.

We found that when we removed the fairness resources, number of ward restrictions and 12 hours shifts, our standard dive consistently produced an integer roster solution within 5% of the

objective function value of solution to the root node of the branch-and-price search tree. With these three resources included, there is a significant gap; see §7.4 for more details.

Our model arbitrarily breaks down a roster-line into on-stretches, off-stretches, and work-stretches, which are all sets of dated activities (shifts or days off). Suppose we instead broke down a roster-line into weeks and fortnights, which could also be sets of dated activities. In that case, we believe we could model the Waikato DHB rules more efficiently. For example, the number of hours worked in a week, $\mathcal{T}_{R'}^{\mathrm{W}x\mathrm{Hours}}$, needs to be modelled in the on-stretch, work-stretch and roster-line. However, it would only need to be modelled once in the week entity. Likewise, the number of hours in a fortnight, $\mathcal{T}_{R'}^{\mathrm{F}x\mathrm{Hours}}$, would only need to be modelled in the week and fortnight entities, but not the roster-line itself.

4.4 Nested column generation

The primary purpose of our entity model is to solve the column generation subproblem to find the lowest reduced cost roster-line for a given employee.

The column generation subproblem can be sub-classified into three separate *nested subproblems*—the generation of on-stretches, work-stretches, and roster-lines. These nested subproblems are considered to be nested as the solutions to each subproblem define the structure of the following subproblem. The nested subproblems only differ between individual employees in terms of the shift costs and the parameters defined in the resource cost functions, feasibility functions and dominance rules; consult §4.2.2. The first and third nested subproblems involve a modified version of the shortest path problem with resource constraints (SPPRCs) defined by Irnich and Desaulniers (2005). In this section, we use a modified version of the terminology introduced by Irnich and Desaulniers to model and solve each of the three aforementioned nested subproblems.

As with Irnich and Desaulniers, we solve both the first and the third nested subproblem with a *dynamic program*. Each resource vector represents a state in the dynamic program. In both nested subproblems, we find a Pareto-optimal set of states ending on each day.

Note that although the primary purpose of the entity model is to be used in the context of a shortest path problem with resource constraints to find negative reduced cost columns for the RMP, it can also be used to quickly find the cost of an employee's roster-line when enumerating neighbouring roster-lines for local search.

4.4.1 Dominance algorithm

To efficiently define an algorithm to solve each of the aforementioned nested subproblems, we first define the generic dominance algorithm for any entity, $\tau \in \mathcal{O} \cup \mathcal{W} \cup \mathcal{R}$. Given a set of entities $\mathcal{U}$ of the same type and with the same initial and final days, we identify the set of non-dominated entities, $\mathcal{P}$, using the following Algorithm 1.

Algorithm 1 Generic dominance algorithm to reduce the size of a set of entities

Require: A set of entities, $\mathcal{U}$.
1: $\mathcal{P} = \emptyset$ ▷ Initialising set of non-dominated entities
2: Order the set, $\mathcal{U}$, in ascending order of the cost of each entity.
3: **for each** Entity $\tau^1 \in \mathcal{U}$ **do** ▷ Both "for each"s maintain the ascending order of cost.
4: isDominated = False
5: **for each** Entity $\tau^2 \in \mathcal{P}$ **do**
6: **if** $\tau^2 \preceq \tau^1$ **then**
7: isDominated = True
8: **if** isDominated = False **then**
9: ADD τ^1 to $\mathcal{P}$.
10: **return** non-dominated set of entities, $\mathcal{P}$.

4.4.2 On-stretch subproblem

The first nested subproblem, involving the generation of on-stretches, is a modified SPPRC to identify the set of non-dominated feasible on-stretches between each possible pair of initial and final days, i.e., $\forall i_1, i_2 \in (1, 2, ..., n)$ s.t. $i_1 < i_2$ where n denotes the total number of days considered in the problem. The SPPRC is modified in that we are building paths from any start day to any other start day in a single SPPRC.

This modified SPPRC is defined on the on-stretch digraph

$$G_{\mathrm{O}} = (\mathcal{S}, \sigma)$$

where $\mathcal{S} = \{D^0, \mathcal{S}^1, \mathcal{S}^2, \ldots, \mathcal{S}^n\}$ denote the set of possible shifts starting on each day; D^0 is a dummy shift and $\mathcal{S}^i$ represents the set of possible shifts starting on day i. We require the dummy shift D^0 as we are building roster-lines from a collection of work-stretches. To model a day off on day 1, we require a one day on-stretch containing only the dummy shift. σ is a set of functions, $\sigma^S(S)$, for each shift S. Each function $\sigma^S(S)$ defines the set of shift successors to a given shift S, e.g., the set of shift successors to a PM shift P^3 on day 3, $\sigma^S(P^3) = \{A^4, P^4, N^4\}$. A given path in the SPPRC corresponds to an on-stretch $O = (S_1, S_2, \ldots, S_{|O|})$. An example of the on-stretch digraph for the Waikato DHB problems is depicted in Figure 4.1.

The NZ nurse MECA specifies that more than 24 hours between shifts is considered a day off. This can be compared with the INRC problems which specify that there is a day off on a calendar day if no shift is started on that day. For example, in the Waikato DHB problems, a roster could contain an AM shift, which finishes at 3.30 pm, followed by an N shift, which begins at 11.30 pm on the following day. Because there are more than 24 hours between these two shifts, it is counted as a day off and thus, this combination cannot be included in an on-stretch, and we must instead provide an off-stretch between these two shifts. Please consult Figure 4.2 for an example of 24 hours off between shifts that start on consecutive calendar days. Because not all shifts on consecutive days can form a feasible on-stretch, the legal set of shift successors for a given shift S^i which starts on day i, $\sigma^S(S^i)$, may not include all shifts worked on day $i + 1$.

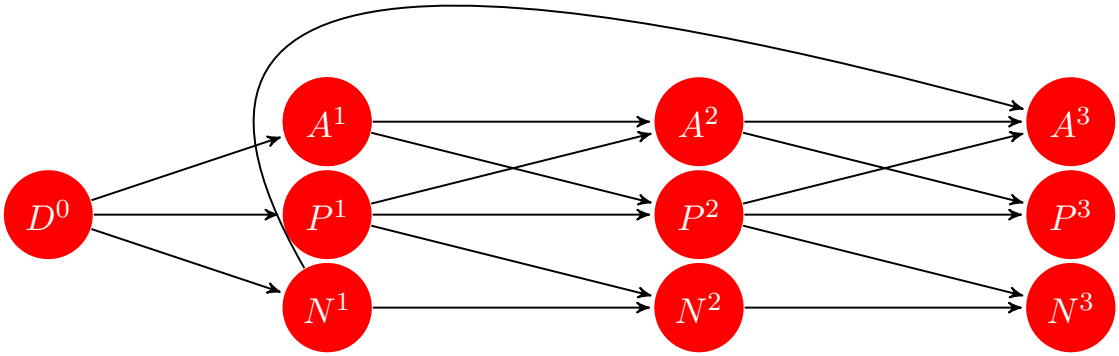


Figure 4.1: On-stretch digraph, G_{O}, for a three day Waikato DHB problem instance. Each vertical set of shift nodes represents the shifts starting on a given day. The D^0 node is a dummy node, so a roster-line, constructed from work-stretches, can start with a day off on day one. The arcs between nodes represent legal shift successions; e.g., the set of shift successors to an N shift N^1 on day 1, $\sigma^S(N^1) = \{N^2, A^3\}$.

Likewise, in the Waikato DHB problems, a roster could contain an N shift, which starts on day x and finishes on day $x+1$ at 3.30 pm, followed by an AM shift, which begins at 8.00 am on day $x+2$. The end of this N shift is less than 24 hours from the start of the AM shift, and as such, the shifts are considered as part of an on-stretch even though there is no shift starting on day $x+1$. Thus, in our on-stretch graph, an N shift starting on day x has an arc to an AM shift starting on day $x+2$. We model this with the set of shift successors $\sigma^S(S^i)$ potentially including shifts starting on day $i+2$.

We solve the on-stretch SPPRC using a label setting algorithm to identify a set of non-dominated, feasible on-stretches, $\mathcal{P}_{\mathrm{O}}^{i_1,i_2}$, between each possible pair of initial and final days, i.e., $\forall i_1, i_2 \in (1, 2, ..., n)$ s.t. $i_1 < i_2$. $\mathcal{U}_{\mathrm{O}}^{i_1,i_2}$ is taken to denote a set of possible on-stretches starting on day i_1 ending on day i_2, before dominance and feasibility checks. We solve the on-stretch SPPRC using Algorithm 2.

4.4.3 Work-stretch subproblem

The second nested subproblem involves the generation of work-stretches. We enumerate the set of non-dominated, feasible work-stretches by combining each non-dominated on-stretch with a feasible off-stretch, as illustrated in Algorithm 3. The set of constructed work-stretches starting on day i_1 and ending on day i_3 before dominance is denoted by $\mathcal{U}_{\mathrm{W}}^{i_1,i_3}$. We define the set of all legal off-stretches from an on-stretch ending with shift S to day i_3 as $\sigma_{i_3}^F(S)$. We model off-stretches as starting from a specific shift so we can model an AM shift starting on day i and an N shift starting on day $i+1$ as having a one-day off-stretch between them for the Waikato DHB problems.

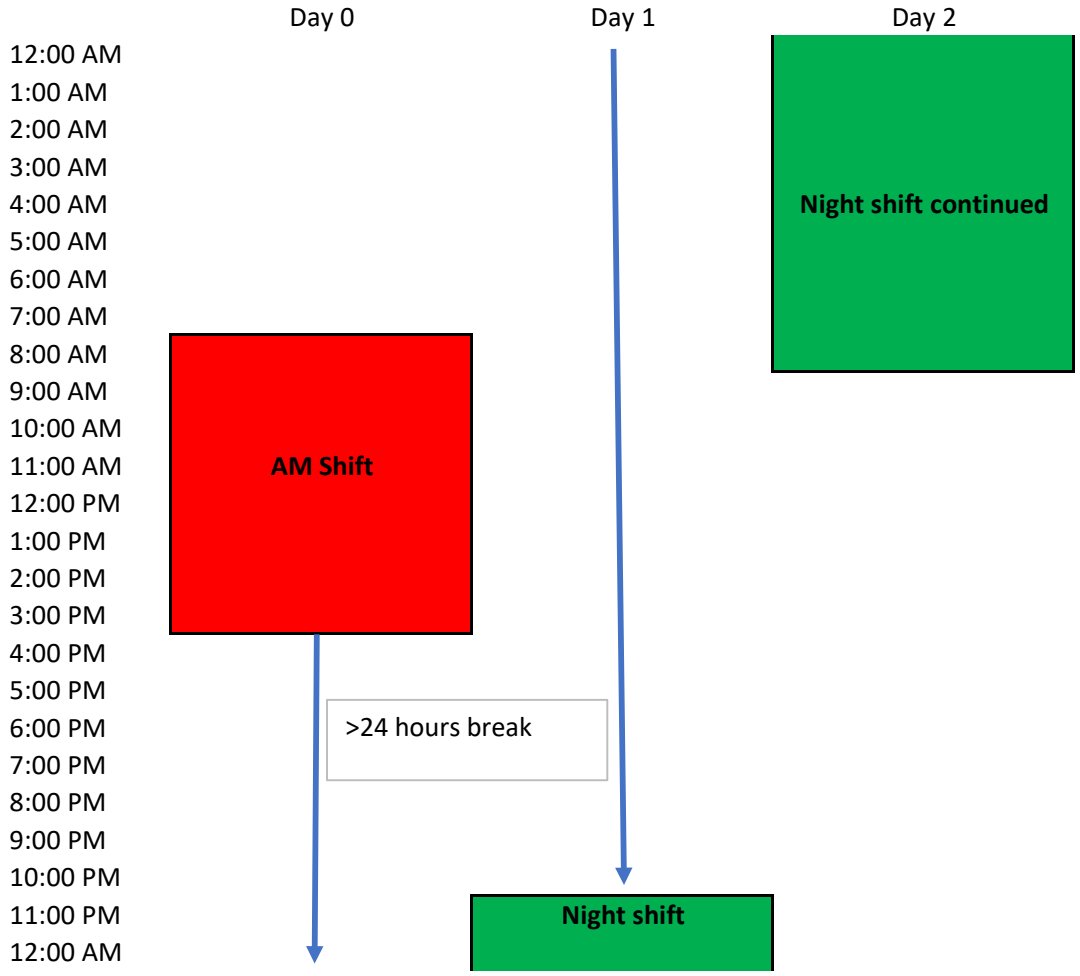


Figure 4.2: An example of an AM shift starting on day 0 and a night shift starting on day 1. In INRC, this would be considered two consecutive days worked as shifts are started in both day 0 and day 1. In the Waikato DHB problems, there is a day off between these two shifts as there is a $>$ 24-hour break between the AM shift on day 0 and the night shift starting on day 1.

Algorithm 2 Solving the first nested subproblem's SPPRCs to generate on-stretches

Require: An on-stretch digraph $G_{\mathrm{O}} = (\mathcal{S}, \sigma)$.

1: $\mathcal{U}_{\mathrm{O}}^{i_1,i_2} = \emptyset \quad \forall i_1 \in (0, 1, \ldots, n), i_2 \in (i_1, i_1 + 1, \ldots, n)$
2: **for** $i_2 \leftarrow 0$ to n **do** ▷ For each possible on-stretch end day.
3: **for each** shift $S \in \mathcal{S}^{i_2}$ **do**
4: ADD $O = (S)$ to $\mathcal{U}_{\mathrm{O}}^{i_2,i_2}$ ▷ Construct an on-stretch out of a single shift.
5: **for** $i_1 \leftarrow 0$ to i_2 **do** ▷ For each possible on-stretch start day.
6: APPLY Algorithm 1 to $\mathcal{U}_{\mathrm{O}}^{i_1,i_2}$ to find $\mathcal{P}_{\mathrm{O}}^{i_1,i_2}$
7: **for each** on-stretch $O \in \mathcal{P}_{\mathrm{O}}^{i_1,i_2}$ **do**
8: **for each** shift $S \in \sigma^S(S_{|O|})$ **do** ▷ where $S_{|O|}$ represents the last shift of on-stretch O and $\sigma^S(S_{|O|})$ represents the legal set of shift successors for shift $S_{|O|}$.
9: ADD $O' = (O, S)$ to $\mathcal{U}_{\mathrm{O}}^{i_1,\mathcal{T}_{O'}^{\mathrm{EndDay}}}$ ▷ Add each shift to the end of each on-stretch in the set of non-dominated on-stretches, $\mathcal{P}_{\mathrm{O}}^{i_1,i_2}$. $\mathcal{T}_{O'}^{\mathrm{EndDay}}$ is the end day of O'.
10: REMOVE infeasible on-stretches from $\mathcal{P}_{\mathrm{O}}^{i_1,i_2}$
11: **return** $\mathcal{P}_{\mathrm{O}}^{i_1,i_2} \quad \forall i_1 \in (0, 1, \ldots, n), i_2 \in (i_1, i_1 + 1, \ldots, n)$

Note that in the case we have an AM shift starting on day 1 and a night shift starting on day 2, our on-stretch starts and finishes on day 1, and our off-stretch also starts on day 1 and finishes on day 2. In Algorithm 3, this is represented by any work-stretch created when $i_1 = i_2$ and $i_3 = i_2 + 1$. In the INRC, work-stretches must be at least two days long.

4.4.4 Checks for work-stretch feasibility with respect to potential roster-lines

In Step 10 of Algorithm 3, we remove work-stretches because their resource vector is infeasible. However, Dohn and Mason (2013) also remove work-stretches if no feasible full roster-line can be constructed containing the given work-stretch.

Firstly, we define a function $B_{\max}^{\theta}(\mathcal{T}_W)$ which is an upper bound on how much roster-line resource $\theta \in \Theta_{\mathrm{R}}$ can increase by extending any roster-line with work-stretch W, i.e.,

$$B_{\max}^{\theta}(\mathcal{T}_W) \geq \max_{\mathcal{T}_R}(\mathcal{E}_{\mathrm{R}}^{\theta}(\mathcal{T}_R, \mathcal{T}_W) - \mathcal{T}_R^{\theta}) \tag{4.36}$$

For example, if we have a roster-line resource representing the number of days worked in a roster-line, then the function, $B_{\max}^{\mathrm{DaysOn}}(\mathcal{T}_W)$ is equal to the number of days worked in the work-stretch, i.e., $B_{\max}^{\mathrm{DaysOn}}(\mathcal{T}_W) = \mathcal{T}_W^{\mathrm{DaysOn}}$.

Dohn and Mason (2013) assume that the value of function $B_{\max}^{\theta}(\mathcal{T}_W)$ is equal to the value of roster-line resource θ when the associated roster-line is only made up of only the work-stretch

Algorithm 3 Generating set of non-dominated work-stretches

Require: A set of non-dominated on-stretches, $[\mathcal{P}_{\mathrm{O}}^{i_1,i_2}]$ and the complete set of off-stretches, $[\mathcal{P}_{\mathrm{F}}^{i_1,i_2}]$

$\mathcal{U}_{\mathrm{W}}^{i_1,i_3} = \emptyset \quad \forall i_1 \in (0,1,\ldots,n)\ \forall i_3 \in (i_1+1, i_1+2, \ldots, n+1)$
for $i_2 \leftarrow 0$ to n **do** ▷ For each on-stretch end day
  for $i_1 \leftarrow 0$ to i_2 **do** ▷ For each on-stretch start day
    for each on-stretch $O \in \mathcal{P}_{\mathrm{O}}^{i_1,i_2}$ **do**
      for $i_3 \leftarrow i_2 + 1$ to $n+1$ **do** ▷ For each off-stretch end day
        for each off-stretch $F \in \sigma_{i_3}^{F}(S_{|O|})$ **do** ▷ where $S_{|O|}$ represents the last shift of on-stretch O and $\sigma_{i_3}^{F}(S_{|O|})$ represents the legal set of off-stretches starting with shift $S_{|O|}$ and ending on day i_3.
          ADD work-stretch $W = (O, F)$ to $\mathcal{U}_{\mathrm{W}}^{i_1,i_3}$.
for $i_1 \leftarrow 0$ to n **do** ▷ For each work-stretch start day
  for $i_3 \leftarrow i_1$ to $n+1$ **do** ▷ For each work-stretch end day
    REMOVE infeasible work-stretches from $\mathcal{U}_{\mathrm{W}}^{i_1,i_3}$
    APPLY Algorithm 1 to $\mathcal{U}_{\mathrm{W}}^{i_1,i_3}$ to find $\mathcal{W}^{i_1,i_3}$
return Set of work-stretch arcs, $\mathcal{W} = (\mathcal{W}^{0,1}, \mathcal{W}^{0,2}, \ldots, \mathcal{W}^{n-1,n+1}, \mathcal{W}^{n,n+1})$.

itself, i.e., $B_{\max}^{\theta}(\mathcal{T}_W) = \mathcal{T}_R^{\theta}$ where $R = (W)$. However, this is not true for any possible resource extension function, and so resource extension functions must be modelled with this assumption in mind.

Next, we define a lower limit $\mathcal{T}_{R,\text{lower}}^{\theta}(d)$ for a roster-line resource θ as a function of the end day d of the partial roster-line. For example, if we have no work-stretches with shifts worked on days 10 to 14 in a 14-day rostering problem, then $\mathcal{T}_{R,\text{lower}}^{\text{DaysOn}}(10) = \gamma^{\theta}$ where γ^{θ} is the lower bound for the full 14-day full roster-line. We find this bound through backpropagation; see Algorithm 4.

Lastly, we define a maximum $\mathcal{T}_{R,\text{forward upper}}^{\theta}(d)$ possible value of roster-line resource θ ending on day d. For example, if we have a work-stretch with days worked on days one to four, then $\mathcal{T}_{R,\text{forward upper}}^{\text{DaysOn}}(4) = 4$. We find this maximum through forward propagation; see Algorithm 5.

Thus, if we have a work-stretch W starting on day d^1 and ending on day d^2, and the maximum possible value $\mathcal{T}_{R,\text{forward upper}}^{\theta}(d^1)$ of resource θ for a roster-line ending on day d^1 plus the maximum possible increase $B_{\max}^{\theta}(\mathcal{T}_W)$ in resource θ from extension with work-stretch W is less than the lower bound $\mathcal{T}_{R,\text{lower}}^{\theta}(d^2)$ on day d^2, then no feasible full roster-line can contain work-stretch W, i.e.,

$$\mathcal{T}_{R,\text{forward upper}}^{\theta}(d^1) + B_{\max}^{\theta}(\mathcal{T}_W) < \mathcal{T}_{R,\text{lower}}^{\theta}(d^2) \Rightarrow \mathcal{F}(W) = \text{False} \tag{4.37}$$

Similarly, if we have a work-stretch W starting on day d^1 and ending on day d^2, and the minimum possible value $\mathcal{T}_{R,\text{forward lower}}^{\theta}(d^1)$ of resource θ for a roster-line ending on day d^1 plus

the minimum possible increase $B^{\theta}_{\min}(\mathcal{T}_W)$ in resource θ from extension with work-stretch W is greater than the upper bound $\mathcal{T}^{\theta}_{R,\text{upper}}(d^2)$ on day d^2, then no feasible full roster-line can contain work-stretch W, i.e.,

$$\mathcal{T}^{\theta}_{R,\text{forward lower}}(d^1) + B^{\theta}_{\min}(\mathcal{T}_W) > \mathcal{T}^{\theta}_{R,\text{upper}}(d^2) \Rightarrow \mathcal{F}(W) = \text{False} \tag{4.38}$$

Algorithm 4 Finding upper and lower bounds on roster-line resource variables as a function of the day

Require: The set of all feasible, non-dominated work-stretches, $\mathcal{W}$.
1: **for each** Day $d^1 \in (n, n-1, \ldots, 1)$ **do**
2: **for each** Day $d^2 \in (d^1+1, d^1+2, \ldots, n)$ **do**
3: **for each** Work-stretch W starting on day d^1 and ending on day d^2 **do**
4: $\mathcal{T}^{\theta}_{R,\text{lower}}(d^2) = \min(\mathcal{T}^{\theta}_{R,\text{lower}}(d^1) - B^{\theta}_{\max}(\mathcal{T}_W), \mathcal{T}^{\theta}_{R,\text{lower}}(d^2))$

Algorithm 5 Finding upper and lower bounds on roster-line resource variables as a function of the day

Require: The set of all feasible, non-dominated work-stretches, $\mathcal{W}$.
1: **for each** Day $d^1 \in (1, 2, \ldots, n)$ **do**
2: **for each** Day $d^2 \in (d^1+1, d^1+2, \ldots, n)$ **do**
3: **for each** Work-stretch W starting on day d^2 and ending on day d^1 **do**
4: $\mathcal{T}^{\theta}_{R,\text{forward upper}}(d^2) = \max(\mathcal{T}^{\theta}_{R,\text{forward upper}}(d^1) - B^{\theta}_{\max}(\mathcal{T}_W), \mathcal{T}^{\theta}_{R,\text{forward upper}}(d^2))$

4.4.5 Roster-line subproblem

The third nested subproblem, involving the generation of the lowest reduced cost, feasible full roster-line, is an SPPRC defined on the roster-line digraph,

$$G_{\text{R}} = (\mathcal{D}, \mathcal{W})$$

where $\mathcal{D} = (D^0, D^1, \ldots, D^n, D^{n+1})$ denotes the set of nodes representing each day considered in the problem, including dummy nodes, D^0 and D^{n+1}, to account for roster-lines beginning with an off-day and ending with a worked shift, respectively. The set of arcs between day nodes is denoted by $\mathcal{W} = (\mathcal{W}^{0,1}, \mathcal{W}^{0,2}, \ldots, \mathcal{W}^{n-1,n})$ where $\mathcal{W}^{i_1,i_2}$ denotes the set of arcs between day node D^{i_1} and day node D^{i_2}. Each arc $W \in \mathcal{W}^{i_1,i_2}$ represents a work-stretch starting on day i_1 and finishing on day i_2. Any path in this SPPRC starting from dummy node D^0 corresponds to a roster-line, R and if the path also finishes on dummy node D^{n+1}, it corresponds to a full roster-line, $\hat{R}$. A simplified example of this digraph is depicted in Figure 4.3.

We solve the roster-line SPPRC using a label setting algorithm to identify the lowest reduced-cost full roster-line. The set of possible roster-lines ending on day i before dominance is denoted by $\mathcal{U}^i_{\text{R}}$ and the set of non-dominated roster-lines ending on day i is denoted by $\mathcal{P}^i_{\text{R}}$. We also define a transition function $\sigma^W(R, W)$ which defines whether partial roster-line R can legally

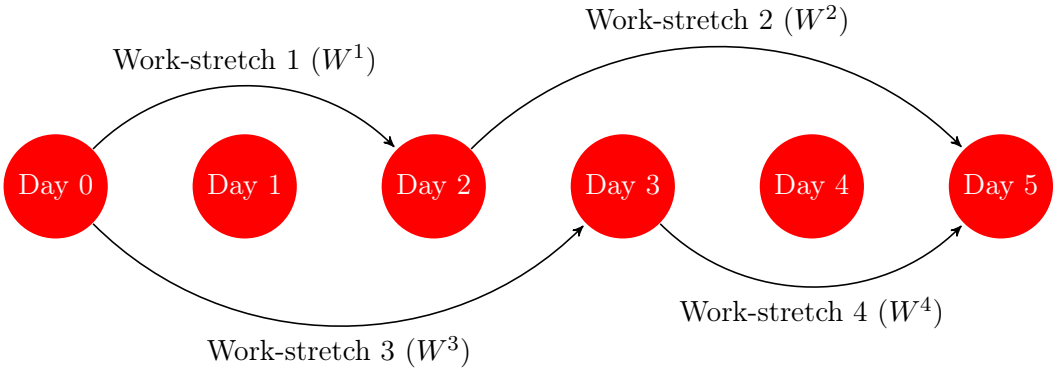


Figure 4.3: Roster-line digraph, G_{R}, corresponding to a 4-day problem. The nodes, Day 0 and Day 5, represent dummy nodes to account for roster-lines beginning with off-days and ending with worked shifts. Four possible work-stretches are depicted—work-stretch 1, $W^1 \in \mathcal{W}^{0,2}$; work-stretch 2, $W^2 \in \mathcal{W}^{2,5}$; work-stretch 3, $W^4 \in \mathcal{W}^{0,3}$; and work-stretch 4, $W^4 \in \mathcal{W}^{3,5}$. There are two possible full roster-lines—$\hat{R}^1 = (W^1, W^2)$ and $\hat{R}^2 = (W^3, W^4)$.

be extended by work-stretch W. This is required as, for example, a one-day off-stretch can only connect an AM shift and an N shift in the Waikato DHB problems. $\sigma^W(R, W) =$ True indicates the extension is legal.

The aim is to identify the lowest reduced-cost full roster-line from the set of all full roster-lines, $\mathcal{U}_{\mathrm{R}}^{n+1}$, where n denotes the total number of days considered in the problem. We only require the lowest reduced cost full roster-line to guarantee our column generation algorithm solves optimally but in practice, adding some of the other generated low-cost full roster-lines to the aforementioned set improves the overall solution time of the LP. We solve the roster-line SPPRC using Algorithm 6. The final solution to all three nested subproblems is a set of full roster-lines generated for a given employee, including the full roster-line with the lowest possible reduced cost.

4.4.6 Checks for partial roster-lines feasibility

In Algorithm 6, we only check the feasibility of full roster-lines. This is because we cannot discard partial roster-lines deemed infeasible using the feasibility criteria of full roster-lines. For example, say we have a partial roster-line with eight work-days and a lower bound on the number of work-days of 10. By extending it with a work-stretch that contains at least two work-days, the partial roster-line becomes feasible. However, similarly to in §4.4.4, we can create a feasibility criteria for partial roster-lines through backward propagation. Dohn and Mason (2013) used lower and upper limits, $\mathcal{T}^{\theta}_{R,\mathrm{lower}}(d)$ and $\mathcal{T}^{\theta}_{R,\mathrm{upper}}(d)$ for each roster-line resource $\theta \in \Theta_{\mathrm{R}}$ for roster-lines ending on each day d calculated through back propagation. This feasibility range can be used to discard infeasible partial roster-lines while solving the roster-line SPPRC.

Algorithm 6 Solving the third nested subproblem's SPPRC to generate the lowest-cost full roster-line

Require: A roster-line digraph $G_{\mathrm{R}} = (\mathcal{D}, \mathcal{W})$ and the number, x, of full roster-lines to be generated.

1: $\mathcal{U}_{\mathrm{R}}^{i} = \emptyset \quad \forall i \in (1, \ldots, n+1)$
2: **for** $i \leftarrow 1$ to n **do**
3: **for each** work-stretch $W \in \mathcal{W}^{0,i}$ **do**
4: ADD $R = (W)$ to $\mathcal{U}_{\mathrm{R}}^{i}$.
5: **for each** day $i \in (1, 2, \ldots, n)$ **do**
6: APPLY Algorithm 1 to $\mathcal{U}_{\mathrm{R}}^{i}$ to find $\mathcal{P}_{\mathrm{R}}^{i}$.
7: **for each** roster-line $R \in \mathcal{P}_{\mathrm{R}}^{i}$ **do**
8: **for each** day $j \in (i+1, i+2, \ldots, n+1)$ **do**
9: **for each** Work-stretch $W \in \mathcal{W}^{i,j}$ **do**
10: **if** $\sigma^{W}(R, W) =$ True **then**
11: ADD $R' = (R, W)$ to $\mathcal{U}_{\mathrm{R}}^{j}$.
12: Order the set, $\mathcal{U}_{\mathrm{R}}^{n+1}$, in ascending order of the cost of each roster-line. ▷ Note: $\mathcal{U}_{\mathrm{R}}^{n+1}$ is the set of full roster-lines.
13: **return** First x feasible roster-lines in $\mathcal{U}_{\mathrm{R}}^{n+1}$.

A roster-line resource is infeasible if it is outside of these bounds for the end day d of its roster-line, i.e.,

$$\mathcal{T}_{\tau}^{\theta} < \mathcal{T}_{R,\mathrm{lower}}^{\theta}(d) \text{ or } \mathcal{T}_{\tau}^{\theta} > \mathcal{T}_{R,\mathrm{upper}}^{\theta}(d) \Rightarrow \mathcal{F}^{d}(\mathcal{T}_{\tau}^{\theta}) = \text{False} \tag{4.39}$$

Using this equation, we can remove partial roster-lines due to infeasibility at Step 6 of Algorithm 6.

4.5 Branching rules

We use two separate branch-and-price algorithms in parallel to solve any given staff rostering problem. Each algorithm has a separate *branching rule* and search strategy.

In staff rostering problems, constraint branching typically involves branching on an employee-shift pair, (e, S). A 1-branch on (e, S) enforces employee e to work shift S. During the solution of the column generation subproblem for a given employee, we enforce the 1-branch by removing all shifts $\mathcal{S}^{\hat{d}}$ worked on the start day, $\hat{d}$, of the enforced shift, $\hat{S}$, i.e., we remove $S \in \mathcal{S}^{\hat{d}}, S \neq \hat{S}$. We also remove all off-stretches that include day $\hat{d}$, i.e., we remove all $F = (\phi^{d}, \phi^{d+1}, \ldots, \phi^{d+|F|})$, where $d \leq \hat{d} \leq d + |F|$. Similarly, a 0-branch prohibits employee e from working a given shift, S. During the column generation subproblem solution, the enforcement of the 0-branch involves the removal of the prohibited shift from the set of possible shifts.

We use a separate method to select which employee-shift pair to branch on for both finding high-quality roster solutions and driving up the lower bound. Firstly, we define the corresponding value of an employee-shift pair (e, S) in a solution to the RMP as the sum of the values of employee

e's roster-line variables λ_e^r where shift S is worked in each roster-line r, i.e., $x_{eS} = \sum_{r \in \mathcal{R}_e | S \in \hat{R}_e^r} \lambda_e^r$. To find lower bounds, we choose to branch on the employee-shift pair (e, S) with the least integer value i.e.,

$$\min_{e,S}(\text{abs}(0.5 - x_{eS}))$$

where $\text{abs}(X)$ is the absolute value of X. To find high quality roster solutions, we used a dive heuristic, which involved branching on the largest corresponding non-integer value of any employee-shift pair (e, S), i.e.,

$$\max_{e,S}(x_{eS}) \text{ s.t. } x_{eS} \text{ is fractional}$$

We also used a separate search strategy for both finding high-quality roster solutions and driving up the lower bound. To drive up the lower bound, we choose nodes to explore next based on their parent node having the lowest objective function value. Since branching rarely produced an increase in the objective, if two nodes have parents with the same objective function value, we prefer 1-branching. To find high quality roster solutions, we chose to only 1-branch (a diving heuristic). However, this dive heuristic was sometimes not producing the optimal roster solution, so we extended the algorithm to use the Limited Discrepancy Search (LDS) by Prosser and Unsworth (2011) to search for additional roster solutions. LDS has already been applied to column generation in the work of Sadykov (2019). However, we use LDS on the employee-shift variables instead of the roster-line variables. Essentially, since we assume that if not for a small number of wrong branching decisions, our branch-and-bound dive would have found a better solution. Thus, we search our branch-and-bound tree in increasing order of branching *discrepancies* to perform a series of similar dives.

4.6 Column generation framework

Genie++ was implemented using COIN-OR's Branch, Cut and Price (BCP) library, which is a framework for handling column generation (Ralphs, 2013). However, for our tests, we created our own staff rostering based column generation framework using only the COIN-OR linear programming (Clp) solver (Forrest et al., 2020).

4.6.1 Dual stabilisation

We found that dual stabilization was effective for reducing the time taken to solve each LP. We took a simple approach to dual stabilization, using the algorithm by du Merle et al. (1999). There are many different ways to apply the algorithm, but we used the current duals as the dual estimate (one of du Merle et al.'s recommendations) and solved the LP for increasingly smaller dual penalties and tolerances (another suggestion by du Merle et al.).

4.6.2 Sprint pricing

We found that the sprint pricing employed by Barnhart et al. (1998) was effective at improving the time to solve the linear relaxation. For some INRC problem instances, in which the structure of the roster is heavily dependent upon which shifts the head nurses work, sprint pricing leads to the root node being solved up to 5 times as fast. Sprint pricing in Genie++ involves having a set of employees that we are generating columns for and removing employees from that set if they do not return a negative reduced cost column. After there are no longer any employees in the set, we add every employee back into the set and continue until no negative reduced cost columns can be generated for any employee. Often, only column generation subproblems for the head nurses produced negative reduced cost columns, so significant time is saved by having fewer regular nurses for whom we are generating columns.

4.6.3 Optimization suite

For our tests, we are using the following CPU: Intel Xeon CPU E5-2650 v2 @ 2.60GHz. We run each of our tests single-threaded, although multiple tests can be run simultaneously.

As our software is compile-time optimized, each model needs to have a separately compiled executable. We do not include the compilation time overhead in our solve times as multiple solves are possible with a single model. However, in practice, this overhead is necessary when modifying the resources.

For enhanced customization, we have implemented our own branch-and-bound tree, dual stabilization and other generic features of column generation. We have used best practices in all the major cases. However, there may be a performance deficit compared with highly tuned column generation frameworks.

4.7 Chapter summary

In this chapter, we have introduced a novel way of modelling a nested column generation subproblem for solving staff rostering problems. We have also introduced example models for the two core applications of our algorithm described in this thesis: solving the INRC problems and solving the Waikato DHB rostering problems. However, both of these applications required novel extensions to our column generation algorithm in order to solve, which is introduced in the following two chapters.

Chapter 5

Proving optimality to all INRC problems

This chapter demonstrates how we can identify proven optimal solutions to every INRC problem by implementing a new dominance technique, an objective function perturbation technique, new branching techniques, and a shift aggregation relaxation. These techniques can be applied to any column generation based staff rostering solver and apply to other column generation problems with complex resource constrained shortest path problems.

5.1 Results before enhancements to Genie++

We solve each INRC problem using Genie++'s branch-and-price framework as described in Chapter 4.

We set a four-hour time limit following Santos et al. (2016) to prove the optimality of the generated solutions to several of the INRC problems. We solved 10 of the 30 problems and established the solutions as optimal. Each of these ten problems exhibited no gap between the cost of its linear relaxed solution and its optimal integer roster solution. We were unable to identify any feasible integer roster solutions to the remaining 20 problems and only managed to increase the lower bound past that of the root node in the case of the problem, "medium hidden 02". Further, we were unable to solve the root node of the five "long hidden" problems.

Although the proposed algorithm was incapable of identifying optimal roster solutions in most cases, the column generation formulation induced strong lower bounds by solving the linear relaxation. Only six problems exhibited gaps between the linear relaxed solutions obtained via the proposed algorithm and the best-known roster solutions in the literature. Our algorithm's success in finding lower bounds is in contrast with the two MIP-based methods mentioned in §1.2. Both methods exhibited gaps between their best-known lower bounds and best-known roster solutions in literature corresponding to a more significant number of problems, notwithstanding the various improvements to the identification of lower bounds; consult Table 3.3 for details.

Two observations help to increase the number of problems solved to proven optimality. The first is that on average, around 5% of the time is spent solving the restricted master problem, and 95% of the time is spent solving the column generation subproblem. Therefore, we can significantly reduce the solution time by implementing a more efficient column generator. The second observation is that constraint-branching rarely increases the lower bound. We contend that this is because of the symmetry present within the INRC problems. Please consult §5.2.2 for more details. Based on these observations, we propose a set of improvements to the column generation subproblem. We present these improvements in the following section, §5.2.

5.2 Enhancement #1: Improving the column generator

Given the shortcomings of our standard column generation approach, our first enhancement to Genie++ is to improve standard dominance to significantly shorten the time taken to solve each column generation subproblem. We also tilted the objective function to produce columns which lead to more naturally integer roster solutions.

5.2.1 Improved resource dominance

In this section, we discuss the extension of standard dominance to increase the number of entities removed by dominance while maintaining proof of optimality.

As discussed in §3.2.5, there are no unique, exact dominance techniques described in the literature on column generation for staff rostering problems. However, there is a need for good dominance strategies as soft constraints and constraints with lower/upper bounds are known to be inefficient for SPPRCs (Legrain et al., 2020) and staff rostering problems typically have multiple instances of these types of constraints.

In §4.4.4 and 4.4.6, we show how Mason and Smith (1998) implemented a standard bidirectional algorithm to calculate hard bounds on the values of resources for roster-lines ending on each day. One naive way of extending this method for resources with soft bounds is to bound each resource's minimum and maximum potential costs. If the sum of the maximum costs of resources for one entity is less than the sum of the minimum costs of resources for a second entity, then the second entity is dominated. These bounds could then be used to improve the dominance results by translating the differences in resource value into differences in the final cost.

However, because of the complex dependencies between resources and non-linear resource cost functions, the gap between the lower and upper bound for the cost of each resource is often large and thus, the dominance condition is weak. For example, it is often possible for an entity with zero weekends worked to violate either the "working too many consecutive weekends" rule, "working too few consecutive weekends" rule, or not violating any consecutive weekends rules at all. Complex dependencies between resources and non-linear cost functions happen naturally as the user is free to define custom resource extension functions and cost functions in object-based C++ code.

Instead of using this naive strategy, we have developed a novel dominance algorithm which we call *dominance cost functions*. With dominance cost functions, instead of calculating the costs for one resource variable at a time, we consider multiple resource values at once. Also, instead of bounding the costs of each entity individually, we bound the difference in costs of two entities. These two factors lead to a very effective dominance condition.

However, since we are considering a maximum bound on the difference in cost between two whole entities with dependencies amongst the resources, calculating the bound automatically would be challenging and computationally expensive. Thus, we push the complexity onto the user and their understanding of how the resources act and have the user create custom code which achieves the same effect. The goal of genie++ is to allow the expert modeller to embed their problem into the framework to allow the framework to run very efficiently, which has generated huge rewards in terms of the actual run times.

Recall from §4.2.5, when performing typical dominance, we compare if each resource in the resource vector dominates another resource separately. However, consider a rostering problem with two resources with binary values; one resource has a guaranteed cost of one million, and one resource might eventually lead to a cost of one. The first resource is so bad that any resource vector that has a value of one for the first resource would be almost always worse than a resource vector that does not. However, every paper we looked at was comparing each resource individually for dominance, and so the cost trade-offs between multiple resources were overlooked. Further, other approaches only confirm cost contribution at a later stage, but we confirm contribution immediately. The dominance cost functions introduced in this section give insight into how we can consider multiple resources at once when performing dominance and make cost trade-offs immediately without sacrificing proof of optimality.

Consider a very simplified example in which the cost of a roster-line is given by the sum of a staff member's preferences for working each shift. There is also a cost of three for each day worked above 20 days or below 10 days in a full roster-line. Consider two partial roster-lines $R^1 = \{N^1, \phi^2, D^3, N^4, D^5, D^6, D^7, D^8, \phi^9\}$[1] with $\mathcal{T}_{R^1}^{\text{DaysOn}} = 7$ days worked and a shift preference sum of $f_{\text{R}}(R^1) = 10$, and $R^2 = \{N^1, N^2, \phi^3, D^4, D^5, D^6, \phi^7, \phi^8, \phi^9\}$ of the same length with $\mathcal{T}_{R^2}^{\text{DaysOn}} = 5$, and $f_{\text{R}}(R^2) = 20$. We want to be able to say that roster-line R^1 dominates roster-line R^2 because of its lower cost. However, under a standard dominance definition, roster-line R^1 could not dominate roster-line R^2 because roster-line R^1 has more days worked, and there is a cost for too many days worked in a full roster-line.

We observe that by replacing roster-line R^2 with roster-line R^1 in any full roster-line containing roster-line R^2, you would increase the total number of days worked by two. In the worst-case scenario, this would lead to an additional cost of six; see Figure 5.1. However, by replacing roster-line R^2 with roster-line R^1, you would also have a guaranteed reduction in the cost of the full roster-line by ten because of the differences due to staff preferences. Thus, roster-line R^1 can still dominate roster-line R^2. This concept of considering the maximum additional cost from a difference in resource values is the motivation behind our novel dominance rule. We denote the maximum additional cost from a difference in resource values, a *dominance cost function*.

[1] where N^1 is a night shift on day 1, ϕ^2 is a day off on day 2 and D^3 is a day shift on day 3.

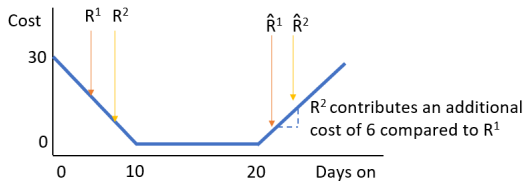


Figure 5.1: The cost for each number of days worked in a full roster-line. On the left, we show the current number of days worked in partial roster-lines R^1 and R^2. On the right, we show the number of days worked in full roster-line $\hat{R}^2$ containing partial roster-line R^2 and that same full roster-line but with roster-line R^2 replaced with roster-line R^1: $\hat{R}^1$.

There is an art to calculating the dominance cost function from a difference in resource values. This is because resources both implicitly and explicitly influence the values of other resources in complex ways. For example, in the INRC problems, the resources representing the number of worked weekend and weekends off in a work-stretch are explicitly mentioned in the resource extension function for the maximum number of consecutive weekends worked resource; see §4.3.1. However, the maximum number of consecutive weekends resource is also implicitly affected by the end day resource, as, after a certain day, there are no further weekends to be worked. We provide an example of how we can create a complex dominance cost function in §5.2.1.4.

The proposed novel dominance rule significantly reduced the number of entities in every single rostering problem that we have solved. In the "long late 03" problem, which is a typical example of one of the more complex INRC problems, via the proposed novel dominance rule, the total number of entities generated during the solution of the column-generation subproblem was observed to be reduced by 89%. The generation of a reduced number of entities induced a 95% reduction in the time required to solve the column-generation subproblem.

This novel dominance rule was the most significant improvement to the column generator. Thus, we consider the proposal of this approach to dominance to be a significant contribution of this study.

Dominance cost functions are most useful at solving SPPRC with multiple resources with costs. Staff rostering problems often have many costs associated with staff preferences which make this technique very useful. Airline scheduling also considers many staff preferences, and we are aware of at least one airline company where dominance cost functions could be useful.

5.2.1.1 Dominance notation

To explain the novel dominance method, we first define some notations. A *horizon* is defined to be a sequence of n days indexed by $1, \ldots, n$. A *span*, $(i_1, i_2) \equiv i_1, i_1 + 1, \ldots, i_2$, is defined to be any subset of consecutive days of the horizon, such that $1 \leq i_1 \leq i_2 \leq n$. Any entity, τ, within a span, (i_1, i_2), denotes a sequence of activities—one for each day in the span—where an activity is defined to be either a worked shift or an off-day. A full roster-line, $\hat{R}$, is an entity within the

span, $(1, n)$, i.e., it defines an activity corresponding to each day in the horizon. The set of all possible full roster-lines is denoted by $\hat{\mathcal{R}}$.

The complement, $\overline{(i_1, i_2)}$, of a span, (i_1, i_2), is defined to be the set of days in the horizon not accounted for by the span, i.e., $\overline{(i_1, i_2)} = \{1, \ldots, n\} \setminus \{i_1, i_1 + 1, \ldots, i_2\}$. The complement of a span does not necessarily comprise consecutive days. An extension, $\overline{\tau}$, of an entity, τ, is defined to be a collection of activities corresponding to each day in the complement of the entity's span. Hence, the union of an entity and an extension of that entity comprise a full roster-line, i.e., a roster-line with a worked shift or an off-day assigned to each day in the horizon. Thus, for an appropriate pair of full roster-line, $\hat{R}$, and entity, τ,

$$\hat{R} = \tau \cup \overline{\tau}, \quad \overline{\tau} = \hat{R} \setminus \tau \tag{5.1}$$

As the INRC problems do not involve any hard constraints, the set of possible complements to two entities, τ^1 and τ^2, of the same type belonging to the same span, are identical. In other words,

$$\tau^1 \cup \overline{\tau^1} \in \hat{\mathcal{R}} \iff \tau^2 \cup \overline{\tau^1} \in \hat{\mathcal{R}} \tag{5.2}$$

5.2.1.2 Dominance definition

In this section, dominance of a given on-stretch, work-stretch, or roster-line, $\tau^2 \in \mathcal{O} \cup \mathcal{W} \cup \mathcal{R}$, by another entity, $\tau^1 \in \mathcal{O} \cup \mathcal{W} \cup \mathcal{R}$ of the same type and belonging to the same span within the column generation subproblem is defined. To ensure that the lowest reduced cost full roster-line is always identified for the column generation subproblem being considered, only entities that are known to not be a subset of the lowest reduced cost full roster-line are discarded. If the replacement of τ^2 with τ^1 in any full roster-line either reduces or preserves the reduced cost of that roster-line, then it is not necessary to include τ^2 in the lowest reduced-cost full roster-line. Thus, it can be safely discarded. This yields the following definition for dominance.

Definition 2. *Given the reduced cost function, f_{R}, corresponding to a roster-line and two entities, $\tau^1, \tau^2 \in \mathcal{O} \cup \mathcal{W} \cup \mathcal{R}$, of the same type and belonging to the same span, τ^2 is considered to be dominated by τ^1, denoted by $\tau^1 \preceq \tau^2$, if the replacement of τ^2 by τ^1 in any full roster-line, $\hat{R} = \tau^2 \cup \overline{\tau^2}$, does not increase the reduced-cost of that roster-line. In other words,*

$$f_{\mathrm{R}}(\underbrace{\tau^1 \cup \overline{\tau^2}}_{\substack{\hat{R} \text{ with } \tau^2 \\ \text{substituted} \\ \text{by } \tau^1}}) \leq f_{\mathrm{R}}(\underbrace{\tau^2 \cup \overline{\tau^2}}_{\hat{R} \in \hat{\mathcal{R}}}) \quad \forall \overline{\tau^2} : \tau^2 \cup \overline{\tau^2} \in \hat{\mathcal{R}} \Rightarrow \tau^1 \preceq \tau^2 \tag{5.3}$$

or equivalently

$$\max_{\overline{\tau^2} : \tau^2 \cup \overline{\tau^2} \in \hat{\mathcal{R}}} \left(f_{\mathrm{R}}(\tau^1 \cup \overline{\tau^2}) - f_{\mathrm{R}}(\tau^2 \cup \overline{\tau^2}) \right) \leq 0 \Rightarrow \tau^1 \preceq \tau^2 \tag{5.4}$$

It is to be noted that the proposed dominance methodology functions equally well in the case of hard-constrained problems by considering the cost, $f_{\mathrm{R}}(\tau^1 \cup \overline{\tau^2})$, of an infeasible roster-line, $\tau^1 \cup \overline{\tau^2}$, to be very high.

5.2.1.3 Work-stretch dominance

Dominance is calculated differently, corresponding to each entity type. In this section, we explain the application of the aforementioned dominance definition to work-stretches, which correspond to the simplest calculation procedure for the determination of dominance. First, $\tau^1 = W^1 \in \mathcal{W}$ and $\tau^2 = W^2 \in \mathcal{W}$ are substituted into (5.4). This yields the following dominance definition for any two work-stretches, W^1 and W^2, belonging to the same span.

$$\max_{\overline{W^2}:\overline{W^2}\cup W^2\in\hat{\mathcal{R}}} \left(f_{\mathrm{R}}(W^1 \cup \overline{W^2}) - f_{\mathrm{R}}(W^2 \cup \overline{W^2}) \right) \leq 0 \implies W^1 \preceq W^2. \tag{5.5}$$

where $\overline{W^2}$ denotes an extension of the work-stretch, W^2.

Substituting the roster-line reduced cost definition given by (4.11) into the LHS of (5.5) yields the following definition of work-stretch dominance.

$$f_{\mathrm{W}}(W^1) + \max_{\overline{W^2}:W^2\cup\overline{W^2}\in\hat{\mathcal{R}}} \left(g_{\mathrm{R}}(W^1 \cup \overline{W^2}) - g_{\mathrm{R}}(W^2 \cup \overline{W^2}) \right) \leq f_{\mathrm{W}}(W^2) \implies W^1 \preceq W^2. \tag{5.6}$$

By substituting (4.18) into the LHS of (5.6), the following definition of work-stretch dominance is obtained in terms of roster-line resource costs.

$$f_{\mathrm{W}}(W^1) + \max_{\overline{W^2}:W^2\cup\overline{W^2}\in\hat{\mathcal{R}}} \sum_{\theta\in\Theta_{\mathrm{R}}} \left(g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{W^1\cup\overline{W^2}}) - g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{W^2\cup\overline{W^2}}) \right) \leq f_{\mathrm{W}}(W^2) \implies W^1 \preceq W^2. \tag{5.7}$$

Explicit calculation of the LHS of (5.7) is complex and computationally expensive. Therefore, before solving the column generation subproblem, a new dominance cost function, $\psi^{\theta}_{\mathrm{W}}(\mathcal{T}_{W^1}, \mathcal{T}_{W^2})$, is constructed corresponding to each roster-line resource, $\theta \in \Theta_{\mathrm{R}}$, such that:

$$\psi^{\theta}_{\mathrm{W}}(\mathcal{T}_{W^1}, \mathcal{T}_{W^2}) \geq \max_{\overline{W^2}:W^2\cup\overline{W^2}\in\hat{\mathcal{R}}} \left(g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{W^1\cup\overline{W^2}}) - g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{W^2\cup\overline{W^2}}) \right) \quad \forall\theta \in \Theta_{\mathrm{R}} \tag{5.8}$$

$$\implies \sum_{\theta\in\Theta_{\mathrm{R}}} \psi^{\theta}_{\mathrm{W}}(\mathcal{T}_{W^1}, \mathcal{T}_{W^2}) \geq \sum_{\theta\in\Theta_{\mathrm{R}}} \max_{\overline{W^2}:W^2\cup\overline{W^2}\in\hat{\mathcal{R}}} \left(g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{W^1\cup\overline{W^2}}) - g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{W^2\cup\overline{W^2}}) \right) \tag{5.9}$$

$$\geq \max_{\overline{W^2}:W^2\cup\overline{W^2}\in\hat{\mathcal{R}}} \sum_{\theta\in\Theta_{\mathrm{R}}} \left(g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{W^1\cup\overline{W^2}}) - g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{W^2\cup\overline{W^2}}) \right) \tag{5.10}$$

Definition 3. *Based on* (5.7) *and* (5.10)*, the following weaker definition for dominance between any two work-stretches,* W^1 *and* W^2*, belonging to the same span can be obtained.*

$$f_{\mathrm{W}}(W^1) + \sum_{\theta\in\Theta_{\mathrm{R}}} \psi^{\theta}_{\mathrm{W}}(\mathcal{T}_{W^1}, \mathcal{T}_{W^2}) \leq f_{\mathrm{W}}(W^2) \implies W^1 \preceq W^2. \tag{5.11}$$

In this definition, we can consider the dominance cost function, $\psi^{\theta}_{\mathrm{W}}(\mathcal{T}_{W^1}, \mathcal{T}_{W^2})$*, to represent the potential degradation in the quality of the resource vector,* $\mathcal{T}_{W^1}$*, compared to the resource vector,* $\mathcal{T}_{W^2}$*. Thus, the obtained dominance definition represents a trade-off between the difference between the current costs of the two entities and the difference between future potential costs incurred by differences between the resource vectors.*

It is useful to compare the aforementioned definition with the standard definition of dominance applied to work-stretches (consult Definition 1). The standard definition involves two conditions for dominance—$\mathcal{T}^{\theta}_{W^1} \preceq \mathcal{T}^{\theta}_{W^2}\ \forall \theta \in \Theta_{\mathrm{W}}$ and $f_{\mathrm{W}}(W^1) \leq f_{\mathrm{W}}(W^2)$. If both these conditions are true, then (5.11) is also true. In this case, the new definition of dominance is observed to be a relaxation of the standard definition of dominance while preserving the proof of optimality.

However, the dominance definition given by (5.11) can be used to construct dominance cost functions, $\psi^{\theta}_{\mathrm{W}}(\mathcal{T}_{W^1}, \mathcal{T}_{W^2})$, such that W^2 is dominated by W^1 even if $\mathcal{T}^{\theta}_{W^1} \not\preceq \mathcal{T}^{\theta}_{W^2}$ for any work-stretch resource, $\theta \in \Theta_{\mathrm{W}}$. For example, if the cost, $f_{\mathrm{W}}(W^1)$, of W^1 is significantly less than the cost, $f_{\mathrm{W}}(W^2)$, of W^2, while the resource vector of the former is only slightly worse than that of the latter, i.e., $\sum_{\theta \in \Theta_{\mathrm{R}}} \psi^{\theta}_{\mathrm{W}}(\mathcal{T}_{W^1}, \mathcal{T}_{W^2}) < f_{\mathrm{W}}(W^2) - f_{\mathrm{W}}(W^1)$, then the domination remains valid under the new definition of dominance.

5.2.1.4 Work-stretch dominance example

In this section, we illustrate the construction of a dominance cost function, $\psi^{\mathrm{MCW}}_{\mathrm{W}}(\mathcal{T}_{W^1}, \mathcal{T}_{W^2})$, that satisfies (5.8) for the 'maximum-number-of-consecutive-weekends-worked' roster-line resource, $\theta = \mathrm{MCW}$, corresponding to work-stretches, W^1 and W^2, belonging to the same span.

Each INRC problem includes four weekends which can be worked. Therefore, for a given full roster-line, $\hat{R} = W^2 \cup \overline{W^2}$, we can define $P_{W^2 \cup \overline{W^2}} = (p_1, p_2, p_3, p_4)$, where $p_i \in \{w, -\}\ \forall i \in 1, 2, 3, 4$ represents the weekends that are worked. Here, w denotes a worked weekend and $-$ indicates an unworked weekend. For example, in a full roster-line, $W^2 \cup \overline{W^2}$, $P_{W^2 \cup \overline{W^2}} = (w, w, -, w)$ indicates that 3 weekends were worked in total, with the maximum number of consecutive weekends worked being $\mathcal{T}^{\mathrm{MCW}}_{W^2 \cup \overline{W^2}} = 2$.

In order to construct the dominance cost function, $\psi^{\mathrm{MCW}}_{\mathrm{W}}(\mathcal{T}_{W^1}, \mathcal{T}_{W^2})$, we first identify the exact cost associated with the MCW resource, $g^{\mathrm{MCW}}_{\mathrm{R}}(\mathcal{T}^{\mathrm{MCW}}_{W^2 \cup \overline{W^2}})$, in terms of any given work-stretch, W^2, and any corresponding feasible extension, $\overline{W^2}$, for work-stretch W^2. The resource cost of the MCW resource is a linear cost, α^{MCW}, for each consecutive weekend worked exceeding a user-specified upper bound, $\gamma^{\mathrm{MCW}} \in \{0, 1, 2, 3\}$, for each employee, in a given roster-line, $W^2 \cup \overline{W^2}$. This cost is given by:

$$g^{\mathrm{MCW}}_{\mathrm{R}}(\mathcal{T}^{\mathrm{MCW}}_{W^2 \cup \overline{W^2}}) = \alpha^{\mathrm{MCW}} \max(\mathcal{T}^{\mathrm{MCW}}_{W^2 \cup \overline{W^2}} - \gamma^{\mathrm{MCW}}, 0) \tag{5.12}$$

The complement, $\overline{(i_1, i_2)}$, of a span, (i_1, i_2), of a given work-stretch, W^2, can comprise non-consecutive days, i.e., $\overline{(i_1, i_2)} = \{1, \ldots, n\} \backslash \{i_1, i_1{+}1, \ldots, i_2\}$, for some $i_1 \neq 1, i_2 \neq n$. To simplify subsequent equations, we decompose any extension of the work-stretch, W^2, and its extension, $\overline{W^2}$, into a partial roster-line, R^1, and a corresponding extension, R^2, to the partial roster-line $R^1 \cup W^2$, which can be expressed as follows.

$$\hat{R} = W^2 \cup \overline{W^2} = R^1 \cup W^2 \cup R^2 \tag{5.13}$$

where R^1 belongs to the span, $(1, i_1 - 1)$, and R^2 belongs to the span, $(i_2 + 1, n)$. Both R^1 and R^2 possess spans with consecutive days.

$\mathcal{T}^{\text{WW}}_{W^1}$	$\mathcal{T}^{\text{WO}}_{W^1}$	$\mathcal{T}^{\text{WW}}_{W^2}$	$\mathcal{T}^{\text{WO}}_{W^2}$	$\mathcal{T}^{\text{MCW}}_{R^1\cup W^1\cup R^2}$	$\mathcal{T}^{\text{MCW}}_{R^1\cup W^2\cup R^2}$
2	0	1	1	4	2

The worked weekends in a given work-stretch, W^1, and the corresponding work-stretch extension, R^1, can be denoted in a manner similar to that for $\hat{R}$. For example, for $P_{R^1\cup W^1\cup R^2} = (w,-,w,w)$, we might have $P_{R^1} = (w,-)$, $P_{W^1} = (w)$, and $P_{R^2} = (w)$.

The objective is to identify a dominance cost function, $\psi^{\text{MCW}}_{\text{W}}(\mathcal{T}_{W^1},\mathcal{T}_{W^2})$, corresponding to each $\gamma^{\text{MCW}} \in \{0,1,2,3\}$ that satisfies (5.8) for all 325 legal combinations of P_{W^1}, P_{W^2}, P_{R^1}, and P_{R^2} such that $|P_{R^1}|+|P_{W^1}|+|P_{R^2}| = 4$, $|P_{W^1}| = |P_{W^2}|$. As the roster-line resource, $\theta = \text{MCW}$, is only affected by two work-stretch resources—the "weekends worked" work-stretch resource, $\theta = \text{WW}$, and the "weekends off" work-stretch resource, $\theta = \text{WO}$, the dominance cost function can be expressed as follows. $\psi^{\text{MCW}}_{\text{W}}(\mathcal{T}_{W^1},\mathcal{T}_{W^2}) = \psi^{\text{MCW}}_{\text{W}}(\mathcal{T}^{\text{WW}}_{W^1},\mathcal{T}^{\text{WO}}_{W^1},\mathcal{T}^{\text{WW}}_{W^2},\mathcal{T}^{\text{WO}}_{W^2})$.

As an example, let us consider one possible combination—$P_{W^1} = (w,w)$, $P_{W^2} = (w,-)$, and an extension comprising $P_{R^1} = (w)$ and $P_{R^2} = (w)$. First, the values of the resource variables corresponding to worked and unworked weekends over the two work-stretches, W^1 and W^2, are calculated. Further, the values of the maximum consecutive weekends resource variable over the two full roster-lines, $R^1 \cup W^1 \cup R^2$ and $R^1 \cup W^2 \cup R^2$, created from those work-stretches are also calculated.

So, in this example,

$$\begin{aligned} g^{\text{MCW}}_{\text{R}}(\mathcal{T}^{\text{MCW}}_{W^1\cup\overline{W^2}}) - g^{\text{MCW}}_{\text{R}}(\mathcal{T}^{\text{MCW}}_{W^2\cup\overline{W^2}}) &= \alpha^{\text{MCW}}\max(4-\gamma^{\text{MCW}},0) - \alpha^{\text{MCW}}\max(2-\gamma^{\text{MCW}},0) \\ &= \begin{cases} 2\alpha^{\text{MCW}}, & \gamma^{\text{MCW}} \in \{0,1\} \\ \alpha^{\text{MCW}}(4-\gamma^{\text{MCW}}) & \gamma^{\text{MCW}} \in \{2,3\} \end{cases} \\ &\leq \begin{cases} 2\alpha^{\text{MCW}}, & \gamma^{\text{MCW}} \in \{0,1,2\} \\ \alpha^{\text{MCW}}, & \gamma^{\text{MCW}} = 3 \end{cases} \end{aligned}$$

This yields the value of $g^{\text{MCW}}_{\text{R}}(\mathcal{T}^{\text{MCW}}_{W^1\cup\overline{W^2}}) - g^{\text{MCW}}_{\text{R}}(\mathcal{T}^{\text{MCW}}_{W^2\cup\overline{W^2}})$ for the combination considered.

Using complete enumeration on all 1300 legal combinations of W^1, W^2, P_{R^1}, P_{R^2}, and γ^{MCW}, establishes that the dominance cost function

$$\psi^{\text{MCW}}_{\text{W}}(\mathcal{T}_{W^1},\mathcal{T}_{W^2}) = \begin{cases} \alpha^{\text{MCW}}(4-\gamma^{\text{MCW}}), & \mathcal{T}^{\text{WO}}_{W^1} = 0 \text{ and } \mathcal{T}^{\text{WO}}_{W^2} > 0 \\ \alpha^{\text{MCW}}(\max(\mathcal{T}^{\text{WW}}_{W^1} - \mathcal{T}^{\text{WW}}_{W^2},0)), & \text{otherwise} \end{cases} \tag{5.14}$$

always satisfies (5.8) corresponding to each upper bound, γ^{MCW}.

Under standard work-stretch dominance rules, W^1 can be dominated by W^2 only if $\mathcal{T}^{\text{WW}}_{W^1} \leq \mathcal{T}^{\text{WW}}_{W^2}$ and $\mathcal{T}^{\text{WO}}_{W^1} \geq \mathcal{T}^{\text{WO}}_{W^2}$. However, under the newly proposed definition of dominance, if $\mathcal{T}^{\text{WW}}_{W^1} > \mathcal{T}^{\text{WW}}_{W^2}$ or $\mathcal{T}^{\text{WO}}_{W^1} < \mathcal{T}^{\text{WO}}_{W^2}$, the domination may still hold if the cost, $f_{\text{W}}(W^1)$, of W^1 is sufficiently lower than the cost, $f_{\text{W}}(W^2)$, of W^2.

Careful consideration of each of the resources yields a dominance cost function, $\psi^{\theta}_{\mathrm{W}}(\mathcal{T}_{W^1}, \mathcal{T}_{W^2})$, corresponding to each resource, $\theta \in \Theta_{\mathrm{W}}$, that is an improvement on its counterpart under standard dominance functions. This enables the domination of a greater number of entities.

5.2.1.5 Roster-line dominance

In this section, we explain how our dominance definition can apply to partial roster-lines. As with work-stretches, we start by putting $\tau^1 = R^1 \in \mathcal{R}$ and $\tau^2 = R^2 \in \mathcal{R}$ in (5.4). This gives us the following dominance definition for any two roster-lines, $R^1 = (W_1^{R^1}, W_2^{R^1}, \ldots, W_{|R^1|}^{R^1})$ and $R^2 = (W_1^{R^2}, W_2^{R^2}, \ldots, W_{|R^2|}^{R^2})$ on the same span:

$$\max_{\overline{R^2}: \overline{R^2} \cup R^2 \in \hat{\mathcal{R}}} \left(f_{\mathrm{R}}(R^1 \cup \overline{R^2}) - f_{\mathrm{R}}(R^2 \cup \overline{R^2}) \right) \leq 0 \implies R^1 \preceq R^2. \tag{5.15}$$

where $\overline{R^2}$ is an extension of roster-line R^2.

Substituting our roster-line reduced cost definition from (4.11) into the LHS of (5.15) gives the following roster-line dominance definition:

$$\max_{\overline{R^2}: R^2 \cup \overline{R^2} \in \hat{\mathcal{R}}} \left(\sum_{W \in R^1} f_{\mathrm{W}}(W) + g_{\mathrm{R}}(R^1 \cup \overline{R^2}) - \sum_{W \in R^2} f_{\mathrm{W}}(W) - g_{\mathrm{R}}(R^2 \cup \overline{R^2}) \right) \leq 0 \tag{5.16}$$

$$\iff \max_{\overline{R^2}: R^2 \cup \overline{R^2} \in \hat{\mathcal{R}}} \Bigg([g_{\mathrm{R}}(R^1 \cup \overline{R^2}) - g_{\mathrm{R}}(R^1)] - [g_{\mathrm{R}}(R^2 \cup \overline{R^2}) - g_{\mathrm{R}}(R^2)] + \sum_{W \in R^1} f_{\mathrm{W}}(W) - \sum_{W \in R^2} f_{\mathrm{W}}(W) + g_{\mathrm{R}}(R^1) - g_{\mathrm{R}}(R^2) \Bigg) \leq 0 \tag{5.17}$$

$$\iff f_{\mathrm{R}}(R^1) + \max_{\overline{R^2}: R^2 \cup \overline{R^2} \in \hat{\mathcal{R}}} \left([g_{\mathrm{R}}(R^1 \cup \overline{R^2}) - g_{\mathrm{R}}(R^1)] - [g_{\mathrm{R}}(R^2 \cup \overline{R^2}) - g_{\mathrm{R}}(R^2)] \right) \leq f_{\mathrm{R}}(R^2) \tag{5.18}$$

Substituting (4.18) into the LHS of (5.18), we get our roster-line dominance definition in terms of resource costs:

$$f_{\mathrm{R}}(R^1) + \max_{\overline{R^2}: R^2 \cup \overline{R^2} \in \hat{\mathcal{R}}} \sum_{\theta \in \Theta_{\mathrm{R}}} \left([g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{R^1 \cup \overline{R^2}}) - g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{R^1})] - [g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{R^2 \cup \overline{R^2}}) - g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{R^2})] \right) \leq f_{\mathrm{R}}(R^2) \tag{5.19}$$

As with work-stretches resources, we construct a new function $\psi^{\theta}_{\mathrm{R}}(\mathcal{T}_{R^1}, \mathcal{T}_{R^2})$, for each roster-line resource $\theta \in \Theta_{\mathrm{R}}$, such that:

$$\psi^{\theta}_{\mathrm{R}}(\mathcal{T}_{R^1}, \mathcal{T}_{R^2}) \geq \max_{\overline{R^2}: R^2 \cup \overline{R^2} \in \hat{\mathcal{R}}} \left([g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{R^1 \cup \overline{R^2}}) - g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{R^1})] - [g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{R^2 \cup \overline{R^2}}) - g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{R^2})] \right) \tag{5.20}$$

Definition 4. *From* (5.19) *and* (5.20)*, we can deduce the following weaker dominance definition for any two roster-lines,* R^1 *and* R^2 *on the same span:*

$$f_{\mathrm{R}}(R^2) + \sum_{\theta \in \Theta_{\mathrm{R}}} \psi^{\theta}_{\mathrm{R}}(\mathcal{T}_{R^1}, \mathcal{T}_{R^2}) \leq f_{\mathrm{R}}(R^1) \implies R^1 \preceq R^2. \tag{5.21}$$

For our INRC problem instances, we found $\psi^{\theta}_{\mathrm{R}}(\mathcal{T}_{R^1}, \mathcal{T}_{R^2})$ for each resource $\theta \in \Theta_{\mathrm{R}}$ either through enumeration as shown in §5.2.1.4 or through exploiting the structure of the resource's cost. For example, the pattern counter resource $\theta = \mathrm{P2}$ has the following property:

$$g^{\mathrm{P2}}_{\mathrm{R}}(\mathcal{T}^{\mathrm{P2}}_{R^1 \cup \overline{R^2}}) = g^{\mathrm{P2}}_{\mathrm{R}}(\mathcal{T}^{\mathrm{P2}}_{R^1}) + g^{\mathrm{P2}}_{\mathrm{R}}(\mathcal{T}^{\mathrm{P2}}_{\overline{R^2}}) \tag{5.22}$$

where $\mathcal{T}^{\mathrm{P2}}_{\overline{R^2}}$ represents the number of times there is a day off on the Friday immediately before a worked weekend in $\overline{R^2}$. Therefore, through substitution into (5.20), we can identify the following function:

$$\psi^{\mathrm{P2}}_{\mathrm{R}}(\mathcal{T}_{R^1}, \mathcal{T}_{R^2}) = 0.$$

5.2.1.6 On-stretch dominance

In this section, we explain how our dominance definition can apply to on-stretches. As with roster-line and work-stretch dominance, we start by putting $\tau^1 = O^1 \in \mathcal{O}$ and $\tau^2 = O^2 \in \mathcal{O}$ in (5.4). This gives us the following dominance definition for any two on-stretches, O^1 and O^2 on the same span:

$$\max_{\overline{O^2}: \overline{O^2} \cup W^2 \in \hat{\mathcal{R}}} \left(f_{\mathrm{R}}(O^1 \cup \overline{O^2}) - f_{\mathrm{R}}(O^2 \cup \overline{O^2}) \right) \leq 0 \implies O^1 \preceq O^2. \tag{5.23}$$

where $\overline{O^2}$ is an extension of on-stretch O^2.

Unlike roster-lines and work-stretches, on-stretches can be a component of two different entities in our model: a longer on-stretch or a work-stretch; refer to (4.12) and (4.14). Thus, on-stretches have two roles in which they can be dominated. They can be dominated as a full on-stretch, $\hat{O}$, which builds into a work-stretch, i.e., $W = \hat{O} \cup F$. They can also be dominated as a partial on-stretch, O, which builds into a full on-stretch which then builds into a work-stretch, i.e., $W = O \cup O^3 \cup F$ where O^3 is a set of shifts on a consecutive span and $O \cup O^3$ is a full on-stretch.

Firstly, we describe dominance between two on-stretches with the assumption that they are partial on-stretches. As with work-stretches, for a given partial on-stretch O^2, any extension of that on-stretch, $\overline{O^2}$, can have a non-consecutive span. Therefore, we decompose a given partial on-stretch O^2 and on-stretch extension $\overline{O^2}$ into consecutive components as follows:

$$\hat{R} = O^2 \cup \overline{O^2} = R^1 \cup O^2 \cup O^3 \cup F \cup R^2. \tag{5.24}$$

where $O^2 \cup O^3$ is a full on-stretch, $O^2 \cup O^3 \cup F$ is a work-stretch, R^1 is a partial roster-line, and R^2 is an extension of partial roster-line $R^1 \cup O^2 \cup O^3 \cup F$.

Substituting our roster-line reduced cost definition from (4.11) and our decomposition of on-stretch extension $\overline{O^2}$ shown in (5.24) into the LHS of (5.23) gives the on-stretch dominance condition:

$$\max_{R^1,O^3,F,R^2:R^1\cup O^2\cup O^3\cup F\cup R^2\in\hat{\mathcal{R}}} \Big(f_{\mathrm{W}}(O^1\cup O^3\cup F) - f_{\mathrm{W}}(O^2\cup O^3\cup F) + g_{\mathrm{R}}(R^1\cup O^1\cup O^3\cup F\cup R^2) - g_{\mathrm{R}}(R^1\cup O^2\cup O^3\cup F\cup R^2)\Big) \leq 0. \tag{5.25}$$

Further substitution of our work-stretch reduced cost definition from (4.10) into (5.25) gives:

$$\max_{R^1,O^3,F,R^2:R^1\cup O^2\cup O^3\cup F\cup R^2\in\hat{\mathcal{R}}} \Big(f_{\mathrm{O}}(O^1\cup O^3) + g_{\mathrm{W}}(O^1\cup O^3\cup F) - f_{\mathrm{O}}(O^2\cup O^3) - g_{\mathrm{W}}(O^2\cup O^3\cup F) + g_{\mathrm{R}}(R^1\cup O^1\cup O^3\cup F\cup R^2) - g_{\mathrm{R}}(R^1\cup O^2\cup O^3\cup F\cup R^2)\Big) \leq 0. \tag{5.26}$$

Lastly substitution of our on-stretch reduced cost definition from (4.9) into (5.26) gives:

$$\max_{R^1,O^3,F,R^2:R^1\cup O^2\cup O^3\cup F\cup R^2\in\hat{\mathcal{R}}} \Big(\sum_{S\in O^1} f_{\mathrm{S}}(S) + g_{\mathrm{O}}(O^1) - \sum_{S\in O^2} f_{\mathrm{S}}(S) - g_{\mathrm{O}}(O^2) + [g_{\mathrm{O}}(O^1\cup O^3) - g_{\mathrm{O}}(O^1)] - [g_{\mathrm{O}}(O^2\cup O^3) - g_{\mathrm{O}}(O^2)] + g_{\mathrm{W}}(O^1\cup O^3\cup F) - g_{\mathrm{W}}(O^2\cup O^3\cup F) + g_{\mathrm{R}}(R^1\cup O^1\cup O^3\cup F\cup R^2) - g_{\mathrm{R}}(R^1\cup O^2\cup O^3\cup F\cup R^2)\Big) \leq 0. \tag{5.27}$$

$$\iff f_{\mathrm{O}}(O^1) + \max_{R^1,O^3,F,R^2:R^1\cup O^2\cup O^3\cup F\cup R^2\in\hat{\mathcal{R}}} \Big([g_{\mathrm{O}}(O^1\cup O^3) - g_{\mathrm{O}}(O^1)] \tag{5.28}$$

$$- [g_{\mathrm{O}}(O^2\cup O^3) - g_{\mathrm{O}}(O^2)] + g_{\mathrm{W}}(O^1\cup O^3\cup F) - g_{\mathrm{W}}(O^2\cup O^3\cup F) + g_{\mathrm{R}}(R^1\cup O^1\cup O^3\cup F\cup R^2) - g_{\mathrm{R}}(R^1\cup O^2\cup O^3\cup F\cup R^2)\Big) \leq f_{\mathrm{O}}(O^2). \tag{5.29}$$

By following a process similar to partial roster-line and work-stretch dominance (See Sections 5.2.1.5 and 5.2.1.3), we find the following on-stretch dominance definition:

Definition 5. *We define dominance for two partial on-stretches, O^1 and O^2, on the same span, in terms of a function $\psi^{\theta}_{\mathrm{O}}(\mathcal{T}_{O^1},\mathcal{T}_{O^2})$ for each resource θ, i.e.,*

$$f_{\mathrm{O}}(O^1) + \sum_{\theta\in\Theta_{\mathrm{R}}\cup\Theta_{\mathrm{W}}\cup\Theta_{\mathrm{O}}} \psi^{\theta}_{\mathrm{O}}(\mathcal{T}_{O^1},\mathcal{T}_{O^2}) \leq f_{\mathrm{O}}(O^2) \implies O^1 \preceq O^2. \tag{5.30}$$

where we construct each function $\psi^{\theta}_{\mathrm{O}}(\mathcal{T}_{O^1}, \mathcal{T}_{O^2})$ *so that it satisfies one of the following equations, depending on whether* θ *is an on-stretch, work-stretch or roster-line resource:*

$$\psi^{\theta}_{\mathrm{O}}(\mathcal{T}_{O^1}, \mathcal{T}_{O^2}) \geq \max_{O^3 : O^2 \cup O^3 \in \mathcal{O}} \left[g^{\theta}_{\mathrm{O}}(\mathcal{T}^{\theta}_{O^1 \cup O^3}) - g^{\theta}_{\mathrm{O}}(\mathcal{T}^{\theta}_{O^1}) \right] - \left[g^{\theta}_{\mathrm{O}}(\mathcal{T}^{\theta}_{O^2 \cup O^3}) - g^{\theta}_{\mathrm{O}}(\mathcal{T}^{\theta}_{O^2}) \right] \qquad \forall \theta \in \Theta_{\mathrm{O}}$$

$$\psi^{\theta}_{\mathrm{O}}(\mathcal{T}_{O^1}, \mathcal{T}_{O^2}) \geq \max_{O^3, F : O^2 \cup O^3 \cup F \in \mathcal{W}} g^{\theta}_{\mathrm{W}}(\mathcal{T}^{\theta}_{O^1 \cup O^3 \cup F}) - g^{\theta}_{\mathrm{W}}(\mathcal{T}^{\theta}_{O^2 \cup O^3 \cup F}) \qquad \forall \theta \in \Theta_{\mathrm{W}}$$

$$\psi^{\theta}_{\mathrm{O}}(\mathcal{T}_{O^1}, \mathcal{T}_{O^2}) \geq \max_{R^1, O^3, F, R^2 : R^1 \cup O^2 \cup O^3 \cup F \cup R^2 \in \hat{\mathcal{R}}} g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{R^1 \cup O^1 \cup O^3 \cup F \cup R^2}) - g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_{R^1 \cup O^2 \cup O^3 \cup F \cup R^2}) \; \forall \theta \in \Theta_{\mathrm{R}}$$

The dominance definition for any two full on-stretches on the same span is the same as Definition 5 but with an empty on-stretch extension i.e., $O^3 = \emptyset$. Thus, $O^1 \cup O^3 = O^1$, $O^2 \cup O^3 = O^2$ and therefore $\psi^{\theta}_{\mathrm{O}}(\mathcal{T}_{O^1}, \mathcal{T}_{O^2}) = 0$ for all $\theta \in \Theta_{\mathrm{O}}$.

5.2.1.7 Implementation recommendations

Because dominance cost functions involve multiple resources at once and can be complicated, it can be easy to make mistakes when creating these functions. Thus, we would recommend starting with traditional resource dominance and transitioning over to dominance cost functions for one resource at a time. If we use both dominance methods simultaneously, we first check for traditional resource dominance and then evaluate our dominance cost functions. Sometimes, just changing one or two resources to dominance cost functions can significantly differ the time taken to solve the column generation subproblem.

Each iteration of our column generation subproblem solves to proven optimality with either method of dominance. Thus, we can compare our column generation subproblem with only traditional dominance to one with dominance cost functions to ensure they are solving identically and our dominance cost functions are correct.

5.2.2 Arbitrary shift preferences

Symmetry was a significant feature of the INRC problem instances. Because of the identical or near-identical formulations of employees working the same contract, entire roster-lines can be swapped between employees with no change to the roster solution's cost. Furthermore, because of the sparseness of both shift preferences and penalised shift sequences, many shift swaps between employees are possible, with no change to the cost of the roster solution. Vanderbeck (2009) reports that when multiple columns have identical formulations, branching on variables in the compact space (in our case, employee-shift pairs) can induce symmetries, which make column generation problems challenging to solve.

Even when there is no gap between the optimal roster solution and the solution to the root node of the branch-and-price search tree, problem instances can be challenging to solve. This is because of the large number of optimal integer solutions. A solution to any node in the branch-and-price search tree can be a convex combination of many optimal roster solutions with the same objective function value, so the solution is rarely integer unless many of the shifts have already been branched on.

To improve each node's natural integrality, we attempt to 'tilt the objective towards a single integer roster solution by making each roster's cost unique. If each roster has a unique cost, then there is one optimal integer solution to each problem instance. We propose making this change by adding a small modification $\delta_{eS} \geq 0$ to the cost of each shift S for a given employee e. If the sum of all of these shift cost modifications is less than one, then any optimal roster solution to our modified formulation is also optimal for our original formulation as all costs in the original formulation are integer. Ideally, we create these modifications so that the sum of any combination of them is unique, so we know that any roster solution is also unique. However, according to the result of Erdos (1957), to define a set of unique shift modifications with a unique sum for any subset and a total sum of less than 1, we would need at least 268 significant decimal digits. Therefore, it is not feasible within the precision of CLP.

We instead find a set of unique random values for our shift cost modifications with a sum of less than one, so our roster solutions are mostly unique. To find this set of values, we find a vector of integers, $(1, 2, \ldots, |\mathcal{E}| \times |\mathcal{S}|)$ where $|\mathcal{E}|$ is the number of employees and $|\mathcal{S}|$ is the number of shifts and divide each value in the vector by the total length of the vector, $|\mathcal{E}| \times |\mathcal{S}|$. We can then randomly choose a value from this vector without replacement for each shift cost modification δ_{eS}.

Mason (2001) uses a similar objective function perturbation and coins it elastic constraint branching. The addition of this improvement means we could solve many of the INRC problem instances with zero branching.

5.2.3 Results after improving the column generator

Based on the first set of improvements to Genie++, we optimally solved all but seven of the INRC problems within the prescribed four-hour time window (medium late 01, long late 03, medium hidden 01, medium hidden 02, medium hidden 04, and medium hidden 05 remained unsolved). Further, we identified better lower bounds than those available in the literature corresponding to nine problems. We provide a detailed analysis of the lower and upper bounds achieved for the eleven most INRC challenging instances in Table 5.1 where either the optimal lower or upper bound had not been reported in existing literature. Based on these results, it is evident that the medium hidden category contains the hardest problem instances.

One of the fundamental problems that persisted despite the improvements was that branching rarely improved the lower bound by more than merely the sum of the shift-cost perturbations. To optimally solve the remaining problems, we developed better branching techniques to increase the lower bounds in the branch-and-bound trees more efficiently.

5.3 Enhancement #2: Improving the branching rules

As described in §4.5, our branching rule involves using constraint branching, i.e., branching on a given employee working or not working a certain shift. However, using standard constraint branching has been ineffective at driving up the lower bounds of any of our problem instances

	Problem										
		medium						long			
	late	hidden					late	hidden			
Obj./Bounds	01	01	02	03	04	05	03	01	02	03	04
Genie++ LB	156	96	215	34	76	118	219	n/a	n/a	n/a	n/a
Best Known LB	156	89	197	28	73	91	219	341	86	36	19
LB (after Enh. #1)	156	96	215	34	76	118	219	345	89	38	22
UB (after Enh. #1)	157	139	228	34	80	120	220	346	89	38	22
Best Known UB	157	111	229	34	78	119	220	346	89	38	22
Genie++ UB	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Proof	✗	✗	✗	✓	✗	✗	✗	✗	✓	✓	✓

Table 5.1: Results corresponding to the 11 problems, where either the optimal lower or upper bound had not been reported in existing literature. These results are obtained via Genie++, before and after enhancement #1. A ✓ in the Proof column indicates that the optimality of the solutions obtained for this problem has been established using Genie++ with enhancements, i.e. the highest lower bound (LB) = the objective of the lowest cost roster solution (UB) found. All of these results were identified within the prescribed four-hour time limit. "n/a" indicates a result could not be found within four hours.

for which there is a gap between the best-known roster solution and the solution to the column generation linear relaxation. This ineffectiveness is most likely due to employees that are working the same contract having near identical formulations and near symmetry between different shift types; see §5.2.2 for more details. Therefore, when branching on an employee not working a specific shift (0-branching), another shift can often be assigned to that employee with no change to the overall roster solution's cost. Since 0-branching is ineffective at raising the lower bound, we want to change to a more effective branching rule.

Thus, our second enhancement to Genie++ is to use a more effective branching rule that circumvents the natural symmetry of this problem.

5.3.1 Aggregate resource branching

There has been some discussion on branching with regards to the staff rostering problem's rules in the literature for using column generation to solve staff rostering problems. This includes branching on the minimum number of consecutive shifts of the same type by Maenhout and Vanhoucke (2010a), branching on the total number of shifts which cover different periods by Mehrotra et al. (2000) and branching on whether one employee must work an activity before another employee by Beliën and Demeulemeester (2006). Please see §3.2.6 for more details. However, no one is branching on the values of resources within a given roster-line in a generic way.

Our first branching rule is an aggregate resource branch. An aggregate resource branch involves branching on the sum of the corresponding values of a given roster-line resource, $\theta \in$

Θ_{R}, over all full roster-lines, $\hat{R}_r^e$, in the roster solution. Branching on the total number of non-symmetrical weekends (i.e., only one shift is worked or two shifts of different types are worked) worked in the roster solution by any employee serves as an example. Roster-line resources exert a direct effect on the roster's cost, increasing the effectiveness of branching on the sum of roster-line resources in increasing the lower bound. Further, branches over groups of employees are less affected by the symmetry between employees.

The first resource branching method involves the enforcement of an upper bound, $B^{\theta\text{-UB}}$, and a lower bound, $B^{\theta\text{-LB}}$, on the total sum of the values of resource variables, $\mathcal{T}^{\theta}_{\hat{R}_r^e}$, corresponding to each roster-line resource, $\theta \in \Theta_{\mathrm{R}}^{C^{\text{linear}}} = \{\theta \in \Theta_{\mathrm{R}} | g^{\theta}_{\mathrm{R}} \equiv C^{\text{linear}}\}$, with a linear cost over each full roster-line, $\hat{R}_r^e$, in the roster solution. In other words, the following constraints are added to the RMP:

$$B^{\theta\text{-LB}} \leq \sum_{e \in \mathcal{E}} \sum_{r \in \mathcal{R}_e} \mathcal{T}^{\theta}_{\hat{R}_r^e} \lambda_e^r \leq B^{\theta\text{-UB}} \quad \forall \theta \in \Theta_{\mathrm{R}}^{C^{\text{linear}}} \quad [\pi^{\theta}] \tag{5.31}$$

where π^{θ} denotes the dual associated with the resource, θ. We choose to branch on roster-line resources with linear costs, $\Theta_{\mathrm{R}}^{C^{\text{linear}}}$, because the values of the corresponding resource variables directly affect the overall cost of the roster (please consult §5.3.1.1 for further details).

Our second branching rule is similar to the first, but is only applicable to full roster-lines in the roster solution that are worked by employees $\mathcal{E}_c$ working a given contract, c. In other words, the following constraints are added to the RMP:

$$B_c^{\theta\text{-LB}} \leq \sum_{e \in \mathcal{E}_c} \sum_{r \in \mathcal{R}_e} \mathcal{T}^{\theta}_{R_r} \lambda_e^r \leq B_c^{\theta\text{-UB}} \quad \forall \theta \in \Theta_{\mathrm{R}}^{C^{\text{linear}}}, \ \forall c \in \mathcal{C} \quad [\pi_c^{\theta}] \tag{5.32}$$

where π_c^{θ} denotes the dual associated with θ for employees working the contract, c.

The reduced cost calculations within the column generation subproblem also need to be altered to account for the differences in the restricted master problem. The new reduced cost, associated with the resource, θ, for employee e working the contract, c, is given by $g^{\theta}_{\mathrm{R}}(\mathcal{T}^{\theta}_R) = (\alpha^{\theta} - \pi^{\theta} - \pi_c^{\theta})\mathcal{T}^{\theta}_R$.

These branching methods are equivalent to the addition of auxiliary original variables to the proposed formulation to represent the value of each roster-line resource for each employee and branching on the sum of those variables. Please consult Desrosiers and Lübbecke (2011) for further details. As the values of all resource variables are always integral, i.e., $\mathcal{T}^{\theta}_{R_r} \in \mathbb{Z} \quad \forall \theta \in \Theta_{\tau}, e \in \mathcal{E}, r \in \mathcal{R}_e$, these branches are valid corresponding to any resource.

5.3.1.1 Branching on resources with other cost functions

Branching on the sum of the values of resources with the cost functions, C^{LBUB} or C^{UB} can be ineffective. This is because resources with values that are not outside the bounds for any roster-line can exhibit sums similar to those of resources where half of the roster-lines have values exceeding the upper bounds and half of the roster-lines have values lower than the lower bounds. An example of this phenomenon concerning the workdays resource, $\mathcal{T}_R^{\text{DaysOn}}$, is depicted in Figure 5.2. Therefore, the sum of the values of these resource variables does not exhibit

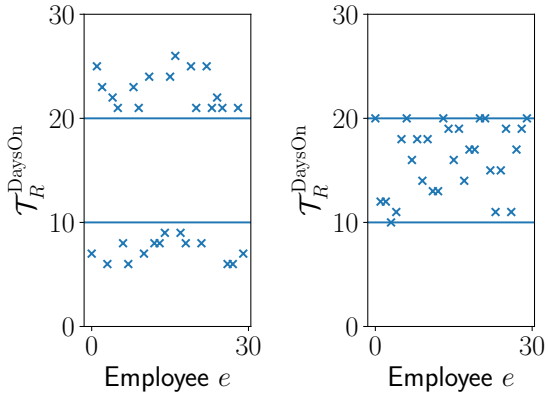


Figure 5.2: The value of the workdays resource, $\mathcal{T}_R^{\text{DaysOn}}$, corresponding to each employee in two different roster solutions. $\mathcal{T}_R^{\text{DaysOn}}$ represents the number of days worked by an employee over the roster-line and exhibits a cost function, C^{LBUB}. Blue lines indicate the lower bound, $\gamma^\theta = 10$, and the upper bound, $\delta^\theta = 20$, for the considered resource's cost function. The sum of this resource variable's values over all employees is similar for each example roster solution. However, the sum of the costs generated by this resource is high for the roster solution on the left and zero for the roster solution on the right.

consistent information about the degree of influence exerted by the resource on the cost of the roster solution.

To branch on resource variables with the cost functions, C^{LBUB} or C^{UB}, we add a new resource variable, $\mathcal{T}_R^{\theta\text{-surplus}}$, representing the amount by which the soft upper bound, γ^θ, is exceeded by resource variable $\mathcal{T}_R^\theta$, and, if relevant, a new resource variable, $\mathcal{T}_R^{\theta\text{-slack}}$, is also added to represent the amount by which resource variable $\mathcal{T}_R^\theta$ falls short of the soft lower bound, δ^θ. For example, the original cost of the roster-line workdays resource variable, $\mathcal{T}_R^{\text{DaysOn}}$, is C^{LBUB}. In other words,

$$g_{\text{R}}^{\text{DaysOn}}(\mathcal{T}_R^{\text{DaysOn}}) = \begin{cases} \alpha^{\text{DaysOn}}(\mathcal{T}_R^{\text{DaysOn}} - \gamma^{\text{DaysOn}}), & \text{if } \mathcal{T}_R^{\text{DaysOn}} > \gamma^{\text{DaysOn}} \\ \beta^{\text{DaysOn}}(\delta^{\text{DaysOn}} - \mathcal{T}_R^{\text{DaysOn}}), & \text{if } \mathcal{T}_R^{\text{DaysOn}} < \delta^{\text{DaysOn}} \\ 0, & \text{otherwise} \end{cases}.$$

This cost is altered to $C^\emptyset$, i.e., $g_{\text{R}}^{\text{DaysOn}}(\mathcal{T}_R^{\text{DaysOn}}) = 0$, and is instead represented by the costs of two new resources—$\mathcal{T}_R^{\text{DaysOn-surplus}}$ with the cost function,

$$g_{\text{R}}^{\text{DaysOn-surplus}}(\mathcal{T}_R^{\text{DaysOn-surplus}}) = \alpha^{\text{DaysOn}}\mathcal{T}_R^{\text{DaysOn-surplus}}$$

and REF,

$$\mathcal{E}_{\text{R}}^{\text{DaysOn-surplus}}(\mathcal{T}_R, \mathcal{T}_W) = \max[\mathcal{T}_R^{\text{DaysOn}} + \mathcal{T}_W^{\text{DaysOn}} - \gamma^{\text{DaysOn}}, 0]$$

and $\mathcal{T}_R^{\text{DaysOn-slack}}$ with the cost function,

$$g_{\text{R}}^{\text{DaysOn-slack}}(\mathcal{T}_R^{\text{DaysOn-slack}}) = \beta^{\text{DaysOn}}\mathcal{T}_R^{\text{DaysOn-slack}}$$

and REF,

$$\mathcal{E}_{\mathrm{R}}^{\text{DaysOn-slack}}(\mathcal{T}_R, \mathcal{T}_W) = \max[\delta^{\text{DaysOn}} - \mathcal{T}_R^{\text{DaysOn}} - \mathcal{T}_W^{\text{DaysOn}}, 0].$$

5.3.1.2 Branching on on-stretch and work-stretch resources

To branch on work-stretch and on-stretch resource variables, we must track these resource variables at the roster-line level. To track an on-stretch resource, θ, at the work-stretch level, we track its value using a new work-stretch resource variable, $\mathcal{T}_W^{\theta\text{-tracker}}$, with the REF,

$$\mathcal{T}_W^{\theta\text{-tracker}} = \mathcal{E}_{\mathrm{W}}^{\theta\text{-tracker}}(\mathcal{T}_O, \mathcal{T}_F) = \mathcal{T}_O^{\theta}.$$

To track a work-stretch resource, θ, at the roster-line level, we simply sum the values of the resources corresponding to each constituent work-stretch using a new roster-line resource variable, $\mathcal{T}_R^{\theta\text{-tracker}}$, with the REF,

$$\mathcal{T}_{R'}^{\theta\text{-tracker}} = \mathcal{E}_{\mathrm{R}}^{\theta\text{-tracker}}(\mathcal{T}_R, \mathcal{T}_W) = \mathcal{T}_R^{\theta\text{-tracker}} + \mathcal{T}_W^{\theta}$$

where $\mathcal{T}_R^{\theta\text{-tracker}}$ is used to track the work-stretch resource, θ, at the roster-line level. This enables branching on the resource variable, $\mathcal{T}_R^{\theta\text{-tracker}}$.

5.3.1.3 Selection method for resource branching

We chose to select the most fractional candidate to branch on in each case. The most fractional value x in a set X is given by:

$$\underset{x\in X}{\text{most fractional}}(x) = \min_{x\in X}(\text{abs}(0.5 - x - \lfloor x \rfloor))$$

where the function abs() gives the absolute value. If we branch over all roster-lines, the up branch is given by

$$B^{\theta\text{-UB}} = \left\lfloor \underset{\theta\in\Theta_{\mathrm{R}}}{\text{most fractional}} \sum_{e\in\mathcal{E}} \sum_{r\in\mathcal{R}_e} \mathcal{T}_{\hat{R}_r^e}^{\theta} \lambda_e^r \right\rfloor \tag{5.33}$$

and the down branch is given by

$$B^{\theta\text{-LB}} = \left\lceil \underset{\theta\in\Theta_{\mathrm{R}}}{\text{most fractional}} \sum_{e\in\mathcal{E}} \sum_{r\in\mathcal{R}_e} \mathcal{T}_{\hat{R}_r^e}^{\theta} \lambda_e^r \right\rceil. \tag{5.34}$$

If we branch over the roster-lines within a contract, the up branch is given by

$$B_c^{\theta\text{-UB}} = \left\lfloor \underset{\theta\in\Theta_{\mathrm{R}}, c\in\mathcal{C}}{\text{most fractional}} \sum_{e\in\mathcal{E}_c} \sum_{r\in\mathcal{R}_e} \mathcal{T}_{\hat{R}_r^e}^{\theta} \lambda_e^r \right\rfloor \tag{5.35}$$

and the down branch is given by

$$B_c^{\theta\text{-LB}} = \left\lceil \underset{\theta\in\Theta_{\mathrm{R}}, c\in\mathcal{C}}{\text{most fractional}} \sum_{e\in\mathcal{E}_c} \sum_{r\in\mathcal{R}_e} \mathcal{T}_{\hat{R}_r^e}^{\theta} \lambda_e^r \right\rceil. \tag{5.36}$$

Branching on the aggregate sum of resource only does not guarantee integrality, and thus, we may also need to use (the less effective) constraint branches to reach optimal solutions.

5.3.2 Aggregate demand branching

Our third branching rule is an aggregate demand branch. Aggregate demand branching is another method to circumvent the symmetry of the INRC problems. It involves branching on the set of aggregate demand variables, z_{cD}, which is defined as the set of employees under each contract, $c \in \mathcal{C}$, working each demand, $D \in \mathcal{D}$. These variables are known as aggregate original variables. Please consult Desrosiers and Lübbecke (2011) for further details.

To add aggregate demand variables to the proposed RMP, we divide the original demand constraints (consult (4.3)) into two different constraint sets. The first constraint set defines the aggregate demand variables, z_{cD}. In other words,

$$\sum_{e \in \mathcal{E}_c} \sum_{r \in \mathcal{R}_e} a^r_{eD} \lambda^r_e = z_{cD} \quad \forall c \in \mathcal{C} \quad \forall D \in \mathcal{D} \quad [\pi_{cD}]. \tag{4.3'a}$$

where $\pi_{c\,D}$ denotes the dual corresponding to each of the constraints.

The second constraint set ensures the fulfilment of the each demand, $D \in \mathcal{D}$ over all employees. In other words,

$$\sum_{c \in \mathcal{C}} z_{cD} = b_D \quad \forall D \in \mathcal{D}. \tag{4.3'b}$$

We also need to modify the calculation of shift duals (see 4.7). When using aggregate demand branching, the shift duals need to be calculated through the following equation:

$$\pi_S = \sum_{D \in \mathcal{D}^S_e} \pi_{cD} \tag{5.37}$$

where columns corresponding to each employee's working contract, $c \in \mathcal{C}$, are generated.

We branch on each aggregate demand variable, z_{cD}, by adding the following additional constraint set to the RMP.

$$B^{\text{Lower}}_{cD} \leq z_{cD} \leq B^{\text{Upper}}_{cD} \quad \forall c \in \mathcal{C} \quad \forall S \in \mathcal{S} \tag{5.38}$$

where B^{Lower}_{cD} and B^{Upper}_{cD} denote the lower and upper bounds for the aggregate demand variables. These branches remain unaffected by the symmetry of employees under the same contract.

As with aggregate resource branching, we chose to select the most fractional candidate to branch on in each case. The up branch involves setting upper bound B^{Upper}_{cD} to be the floor of the most fractional aggregate demand variable, i.e.,

$$B^{\text{Upper}}_{cD} = \left\lfloor \underset{D \in \mathcal{D}, c \in \mathcal{C}}{\text{most fractional}}\ z_{cD} \right\rfloor \tag{5.39}$$

and the down branch involves setting lower bound B^{Lower}_{cD} to be the ceiling of the most fractional aggregate demand variable, i.e.,

$$B^{\text{Lower}}_{cD} = \left\lceil \underset{D \in \mathcal{D}, c \in \mathcal{C}}{\text{most fractional}}\ z_{cD} \right\rceil \tag{5.40}$$

Branching on aggregate demand branches only does not guarantee integrality, and thus, we may also need to use (the less effective) constraint branches to reach optimal solutions.

Algorithm	branching order	selection strategy	search strategy
lower bound search #1	order #1	most fractional	best first
lower bound search #2	order #2	most fractional	best first
upper bound search #1	order #1	least fractional	LDS
upper bound search #2	order #2	least fractional	LDS

Table 5.2

5.3.3 Priority first shift branching

The last branching technique we used to improve the lower bounds of some INRC problem instances is *priority first shift branching*. With priority first shift branching we divide up employee-shift pairs into two different set of pairs: priority employee-shift pairs, i.e., (e, S) where shift $S \in \hat{\mathcal{S}}$ and $\hat{\mathcal{S}}$ is the set of priority shifts, and non-priority employee-shift pairs, i.e., (e, S) where shift $S \in \mathcal{S} \setminus \hat{\mathcal{S}}$.

In the case of INRC, the set of priority shifts is the set of head nurse shifts, i.e.,

$$\hat{\mathcal{S}} = (S \in \mathcal{S} | \mathcal{T}_S^{\text{ShiftType}} = \text{head nurse}) \tag{5.41}$$

where $\mathcal{T}_S^{\text{ShiftType}}$ is the shift type resource variable for shift S. Experimentally, we found that branches on head nurse shifts have a larger effect on the lower bound than branches on regular shifts.

Branching on priority shifts only does not guarantee integrality, and thus, we may also need to branch on non-priority employee-shift pairs to reach optimal solutions.

5.3.4 Overall branching rules

As discussed so far in §5.3, we have implemented multiple different branching rules. We have used two algorithms for this set of tests for finding lower bounds and two algorithms for finding high-quality roster solutions. Each algorithm has a separate order of branching rules, selection strategy within each branching rule and branch-and-bound tree search strategy. A summary of each algorithm is provided in Table 5.2. For more details on most fractional branching, least fractional branching, best first search and LDS, refer to §4.5.

We had two different orders in which we prioritise different branching rules. Order #1 is as follows:

1. Aggregate resources (see §5.3.1)
2. Priority shift-employee pairs (see §5.3.3)
3. Aggregate demands (see §5.3.2)
4. Non-priority shift-employee pairs (see §4.5)

We have ordered these branching rules by the number of possible branches they can produce. As the first three branching rules do not guarantee integrality, if we cannot find a fractional candidate within the first branching rule, then we use the second branching rule and so forth. Order #2 is as follows:

1. Aggregate resources (see §5.3.1)
2. Aggregate demands (see §5.3.2)
3. Priority shift-employee pairs (see §5.3.3)
4. Non-priority shift-employee pairs (see §4.5)

We ran all four algorithms in parallel for each INRC problem instance until either the time limit was reached or the best lower bound found was equal to the highest quality roster solution.

5.3.5 Results after improving the branching rules

Table 5.3 presents the results obtained by resolving the remaining unsolved problems using the aforementioned improved branching techniques. Using the improved branching strategies, we were able to identify optimal lower bounds in all problems. Their optimality follows from their equality with the objective functions of the best roster solutions obtained subsequently (see Table 5.4).

We increased the lower bound above the root node solution in five problem instances. Further, we identified higher quality roster solutions for all five medium hidden instances than had been solved without the improved branching rules.

5.4 Enhancement #3: Shift aggregation

Our third enhancement to Genie++ is to aggregate multiple shift types into a single shift type. This aggregation accelerates the lower bounds' improvement and leads to the production of higher quality integer roster solutions.

By tracking both the non-integer and integer roster solutions produced within the proposed branch-and-bound tree, we concluded that there were almost zero undesirable shift sequence patterns or non-identical shift weekends. This observation led to the hypothesis that differentiation between shift types did not significantly affect the solution quality.

Therefore, we removed the on-stretch resources related to different shift types from the column generation subproblem. These on-stretch resources include:

- The eight on-stretch shift sequence pattern resources, $\mathcal{T}_O^{\mathrm{Pi}} \quad \forall i \in (1, 2, \ldots, 8)$
- The identical shift weekend resource, $\mathcal{T}_O^{\mathrm{ISW}}$

Please see §4.3.1 for more details about these resources.

We also aggregated the shift types in both the RMP and the column generation subproblem so that we are only concerned with which days the employees are working and not which shifts

	Problem										
	medium						long				
	late	hidden					late	hidden			
Obj./Bounds	01	01	02	03	04	05	03	01	02	03	04
Genie++ LB	156	96	215	34	76	118	219	n/a	n/a	n/a	n/a
Best Known LB	156	89	197	28	73	91	219	341	86	36	19
LB (after Enh. #1)	156	96	215	34	76	118	219	345	89	38	22
LB (after Enh. #2)	157	111	219	-	78	118	220	346	-	-	-
UB (after Enh. #2)	157	111	224	-	78	119	220	346	-	-	-
UB (after Enh. #1)	157	139	228	34	80	120	220	346	89	38	22
Best Known UB	157	111	229	34	78	119	220	346	89	38	22
Genie++ UB	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Proof	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓

Table 5.3: Results corresponding to the seven problems remaining unsolved after Enhancement #1. These results are obtained via Genie++, equipped with all the aforementioned enhancements, using the proposed four tree-searching algorithms. As in Table 5.1, all of these results were identified within the prescribed four-hour time limit. "n/a" indicates a result could not be found within four hours.

they are working on those days. We removed the early shifts, day shifts, late shifts and night shifts from our set of shifts, $\mathcal{S}$, and replaced them with a single shift for each day, which we refer to as a "Regular shift". The one shift type we found that we could not remove without significantly changing the quality of the roster solutions produced is the head nurse shift. Therefore, our new shift aggregation formulation has regular shifts and head nurse shifts. We calculated the demand for each regular shift each day by summing up the demands of the removed shifts.

We demonstrate how shifts were aggregated using the "medium hidden 04" problem instance. This instance has the following shifts with respective demands (on the left) on a Wednesday that we aggregated as shown (on the right):

- Early shift: demand 6
- Late shift: demand 6
- Day shift: demand 3
- Night shift: demand 4

→ • Regular shift: demand 19

- Head nurse shift: demand 1 → • Head nurse shift: demand 1

As we are only removing potential roster-line costs and relaxing the demand constraints in our RMP, our shift aggregation technique is a relaxation of the original formulation. Thus, the cost of a given roster solution in this shift aggregation formulation is always less than that roster solution's cost in the full formulation. Therefore, the lower bound for the shift aggregation formulation is still a valid lower bound for the original formulation.

We used the integer roster solutions obtained by solving the problem using the shift aggregation formulation mentioned above to identify integer roster solutions valid for the original formulation. We identified these roster solutions by solving the original formulation using a hard constraint in the column generator, mandating that all roster solutions be of the same form as the identified solution to the shift aggregation formulation. For example, suppose the shift aggregation formulation's identified solution has assigned a particular employee to a regular shift on a particular day. In that case, constraints are added to the column generator to ensure that the employee is necessarily assigned an early shift, late shift, day shift, or night shift on that day. This enabled us to find integer roster solutions valid for the original formulation with the same cost as the corresponding solution to the proposed shift aggregation formulation in every case. Therefore, although this relaxation is a heuristic, the lower bounds it produces are provably valid lower bounds for the original problem and the upper bounds happened to always be valid.

Aggregating shift types provides several benefits over the standard formulation. There are fewer feasible entities in the column generation subproblem since there are only two shift types in the shift aggregation formulation. Thus, we can solve the column generation subproblem faster. Also, as there are fewer demand constraints, the LP size is reduced and takes less time to solve. Lastly, the algorithm produces more naturally integer solutions because there is less shift-type symmetry than without the aggregation.

A similar decomposition was used successfully by Valouxis et al. (2012); please see §3.3. The novel differences are that we do not decompose by week, and we differentiate between normal shifts and head nurse shifts instead of just days on and days off.

Another heuristic that is likely to significantly improve the solve time would be to aggregate all employees with the same contract. However, this heuristic would be ineffective in real rosters as employees usually differ by hard constraints (e.g., different FTEs) as well as history. Since we had already solved every instance to proven optimality, we did not deem it necessary to experiment with this heuristic.

5.4.1 Final results

Table 5.4 presents the improved results obtained by executing the proposed shift aggregation technique or a period of one hour. The optimality of the solutions to all 30 difficult INRC problems were established within the prescribed four-hour time limit. Further, we identified new solutions, which have been depicted on the official website: `https://nrpcompetition.kuleuven-kulak.be/instances-results/`.

The five most difficult roster solutions in terms of the identification of optimal solutions exhibit gaps between the root node solution and the optimal roster solution. These are also the most difficult problems to solve based on heuristics, as evidenced by the identification of new solutions to these problems in this study several after years their introduction.

Figure 5.3 shows the branch-and-bound tree of our best first search strategy applied to "long late 03". With only resource branching and our shift aggregation technique, we can prove

	Problem										
	medium						long				
	late	hidden					late	hidden			
Obj./Bounds	01	01	02	03	04	05	03	01	02	03	04
Genie++ LB	156	96	215	34	76	118	219	n/a	n/a	n/a	n/a
Best Known LB	156	89	197	28	73	91	219	341	86	36	19
LB (after Enh. #1)	156	96	215	34	76	118	219	345	89	38	22
LB (after Enh. #2)	157	111	219	-	78	118	220	346	-	-	-
LB (after Enh. #3)	-	-	219	-	-	118	-	-	-	-	-
UB (after Enh. #3)	-	-	219	-	-	118	-	-	-	-	-
UB (after Enh. #2)	157	111	224	-	78	119	220	346	-	-	-
UB (after Enh. #1)	157	139	228	34	80	120	220	346	89	38	22
Best Known UB	157	111	229	34	78	119	220	346	89	38	22
Genie++ UB	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Proof	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 5.4: Results corresponding to the two problems remaining unsolved after Enhancement #2. These results are obtained by Genie++, equipped with all the aforementioned improvements, using the proposed four tree-searching algorithms. As in Tables 5.1 and 5.3, all of these results were identified within the prescribed four-hour time period. "n/a" indicates a result could not be found within four hours.

optimality to this problem instance with a small tree; our column generation algorithm did not require any constraint branches.

5.5 Chapter summary

The results obtained in this chapter establish that we can identify provably optimal solutions for all 30 of the hardest INRC problems within a reasonable amount of time. We achieved this using a generic column generation algorithm with novel dominance cost functions, arbitrary shift preferences, novel branching strategies, and a shift aggregation technique. This study represents the first successful effort to prove the optimality of solutions to all of the INRC problems.

We have also established that problems that are difficult to solve via column generation solvers due to their inherent symmetry remain challenging to solve with heuristics. With improvements to circumvent the symmetries in the problems, the proposed column generation algorithm was more effective than standard heuristics in solving these problems.

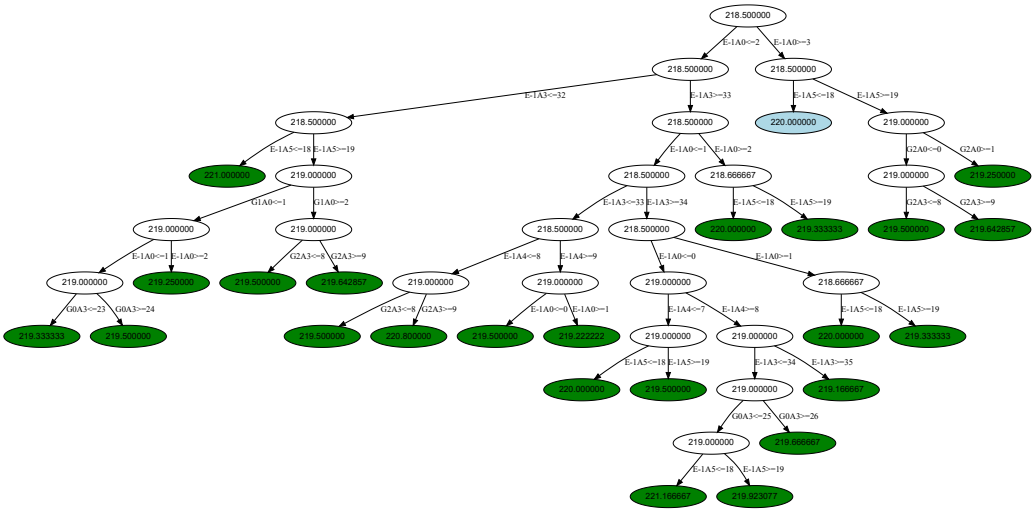


Figure 5.3: Example branch-and-bound tree for "long late 03." The integer solution found is shown in light blue. Bounded nodes are shown with green. Resource branches are shown with "E-1Ax" where x is the resource index. Contract resource branches are shown with "GyAx" where y is the contract index and x is the resource index.

Chapter 6

Neighbourhood search strategies using branch and price

The following two Chapters, Chapters 6 and 7, demonstrate how we can identify high-quality solutions to the operating theatre ward and maternity wards problems. Using the strategies outlined in Chapter 5 dramatically increases our chances of finding a proven optimal solution to a staff rostering problem. However, in more challenging problems such as the maternity wards problem, our algorithm cannot find a proven optimal solution in a reasonable amount of time. However, by using the strategies outlined in Chapter 5, we can construct an initial roster solution that can be improved through neighbourhood search.

Thus, in this chapter, we demonstrate how we can perform neighbourhood search to improve a sub-optimal integer solution's quality quickly by using branch and price.

Section 6.1 introduces 14 different neighbourhoods that can be used to search for improvements to incumbent roster solutions. In Sections 6.2-6.3, we demonstrate how we implemented a subset of these neighbourhoods with our column generation model. In Sections 6.4-6.5, we perform comprehensive tests to find which neighbourhoods work best with column generation and how the problem's difficulty and the initial incumbent solution's quality impact these neighbourhoods' effectiveness.

6.1 Neighbourhoods

This section details 14 neighbourhoods that can apply to integer incumbent solutions to staff rostering problems. The following neighbourhood definitions are not column generation specific.

Firstly, we must define some notation to describe our different neighbourhoods accurately. With any neighbourhood, we are classifying whether a given candidate roster solution, X, is similar to an incumbent roster solution, $\hat{X}$. As these neighbourhoods are not column generation specific, we define a candidate roster solution in terms of each employee-shift variable's values, $x_{eS} \in X$, i.e., $x_{eS} = 1$ if employee e works shift S and 0 otherwise. We also define an incumbent

roster solution in terms of the values of each employee-shift variable $\hat{x}_{eS} \in \hat{X}$. Lastly, we define the set of shifts on day d as $\mathcal{S}^d$.

Given some candidate roster X and an incumbent roster $\hat{X}$, we define a *shift modification* δ_{ed} to employee e's incumbent roster-line on day d as follows:

$$\delta_{ed} = \begin{cases} 0, & x_{eS} = \hat{x}_{eS} \quad \forall S \in \mathcal{S}^d \\ 1, & \text{otherwise} \end{cases} \tag{6.1}$$

Likewise, we can define an *on/off modification* δ^{on}_{ed}, which indicates a modification for an employee from working a shift on a given day to having a day off and vice versa. The on/off modification is defined as follows:

$$\delta^{\text{on}}_{ed} = \begin{cases} 0, & \sum_{S \in \mathcal{S}^d} x_{eS} = \sum_{S \in \mathcal{S}^d} \hat{x}_{eS} \\ 1, & \text{otherwise} \end{cases} \tag{6.2}$$

From these two definitions for roster modifications, we define four aggregate modifications to an incumbent roster. The first aggregate modification is the *aggregate employee modification*, δ_e. This modification indicates whether there are any changes to the roster-line of employee $e \in \mathcal{E}$. The aggregate employee modification is defined as follows:

$$\delta_e = \begin{cases} 0, & \sum_{d \in \mathcal{D}} \delta_{ed} = 0 \\ 1, & \text{otherwise} \end{cases} \tag{6.3}$$

The second aggregate modification is the *aggregate day modification* δ_d. This modification indicates whether there are any changes to any employee's activity on a given day, $d \in \mathcal{D}$ where $\mathcal{D} = (1, 2, \ldots, n)$ is the set of all day indices. The aggregate day modification is defined as follows:

$$\delta_d = \begin{cases} 0, & \sum_{e \in \mathcal{E}} \delta_{ed} = 0 \\ 1, & \text{otherwise} \end{cases} \tag{6.4}$$

The on/off aggregate modifications are identical to the first two aggregate modifications except instead use the on/off modifications. The *aggregate on/off day modification* is defined as follows:

$$\delta^{\text{on}}_e = \begin{cases} 0, & \sum_{d \in \mathcal{D}} \delta^{\text{on}}_{ed} = 0 \\ 1, & \text{otherwise} \end{cases} \tag{6.5}$$

Lastly, the *aggregate on/off employee modification is defined as follows*:

$$\delta^{\text{on}}_d = \begin{cases} 0, & \sum_{e \in \mathcal{E}} \delta^{\text{on}}_{ed} = 0 \\ 1, & \text{otherwise} \end{cases} \tag{6.6}$$

An example of how this notation can apply to an incumbent solution and a candidate roster solution is shown in Figure 6.1.

Using the notation above, we define all 14 neighbourhood restrictions that we use for a rostering problem. Please refer to Table 6.1 for a list of all 14 neighbourhoods. We categorise

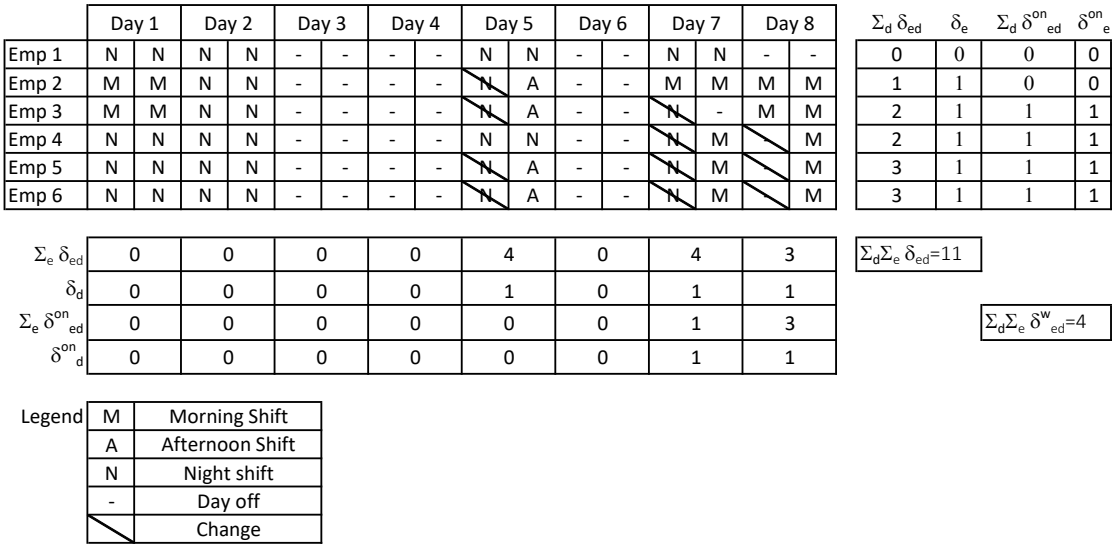


Figure 6.1: Example of each modification variable for a given incumbent solution (shown on the left column of each day) compared to a candidate roster solution (shown on the right column of each day).

the neighbourhoods into two different types: "maximum changes neighbourhood" and "fixed sub-roster neighbourhood". For the maximum changes neighbourhoods, we specify a neighbourhood function, $N(\hat{X}, k)$, which takes an incumbent roster solution $\hat{X}$ and a neighbourhood distance parameter k and modifies the restricted master problem or column generation subproblem appropriately to apply the neighbourhood restrictions. The neighbourhood distance parameter, k, specifies how many changes can be made to the incumbent roster solution.

We define a fixed sub-roster neighbourhood as one that allows for changes only in some regions of the problem. For the fixed sub-roster neighbourhoods, we specify a neighbourhood function, $N(\hat{X}, k, j)$, which takes an incumbent roster solution $\hat{X}$, a neighbourhood distance parameter k, and a neighbourhood location parameter j and modifies the column generation subproblem appropriately to apply the neighbourhood restrictions. The neighbourhood distance parameter, k, specifies how much of the search space is unfixed, and the neighbourhood location parameter, j, determines where the search space is unfixed.

We also categorise the neighbourhoods in terms of whether they count shift modifications or on/off modifications.

It is possible to use any combination of the 14 neighbourhoods simultaneously. For example, we could use both fixed days neighbourhood, $\delta_d = 0 \; \forall \; d \notin \{3, \ldots, 5\} \subseteq \mathcal{D}$ and maximum shift changes per employee neighbourhood, $\sum_d \delta_{ed} \leq 1 \; \forall \; e$, which indicates that days three, four, and five can have modifications, but there can only be one modification per employee's roster-line.

Within the column generation literature for staff rostering problems, only Legrain et al. (2020) and Bulog (2011) have used neighbourhood restricted column generation: Legrain et al. (2020) used fixed employees neighbourhood $\delta_e = 0 \; \forall \; e \notin \{j, \ldots, j + k\} \subseteq \mathcal{E}$ and Bulog (2011)

Type	Modification	Neighbourhood	Formula (shorthand*)	C.G.
Maximum changes neighbourhood	Shift (δ_{ed})	Maximum shift changes per employee	$\sum_d \delta_{ed} \leq k \; \forall \; e$	✓
		Maximum shift changes per day	$\sum_e \delta_{ed} \leq k \; \forall \; d$	✗
		Maximum shift changes per roster	$\sum_e \sum_d \delta_{ed} \leq k$	✗
		Maximum employee changes	$\sum_e \delta_e \leq k$	✗
		Maximum day changes	$\sum_d \delta_d \leq k$	✗
	On/off (δ^{on}_{ed})	Maximum on/off changes per employee	$\sum_d \delta^{\text{on}}_{ed} \leq k \; \forall \; e$	✓
		Maximum on/off changes per day	$\sum_e \delta^{\text{on}}_{ed} \leq k \; \forall \; d$	✗
		Maximum on/off changes per roster	$\sum_e \sum_d \delta^{\text{on}}_{ed} \leq k$	✗
		Maximum employee on/off changes	$\sum_e \delta^{\text{on}}_{e} \leq k$	✗
		Maximum day on/off changes	$\sum_d \delta^{\text{on}}_{d} \leq k$	✗
Fixed sub-roster neighbourhood	Shift (δ_{ed})	Fixed days	$\delta_d = 0 \; \forall \; d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$	✓
		Fixed employee	$\delta_e = 0 \; \forall \; e \notin \{j, \ldots, j+k\} \subseteq \mathcal{E}$	✓
	On/off (δ^{on}_{ed})	Fixed days (on/off)	$\delta^{\text{on}}_{d} = 0 \; \forall \; d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$	✓
		Fixed employees (on/off)	$\delta^{\text{on}}_{e} = 0 \; \forall \; e \notin \{j, \ldots, j+k\} \subseteq \mathcal{E}$	✓

Table 6.1: Shows a comprehensive list of the neighbourhood rules we define using our modification notation. The table shows our two separate neighbourhood types, "maximum changes neighbourhood" and "fixed sub-roster neighbourhood." It also specifies whether the neighbourhood is defined by counting shift modifications δ_{ed} or on/off modifications δ^{on}_{ed}. Lastly, the "C.G." column indicates whether the neighbourhood can be enforced purely within the column generation subproblem. *Note: We give formulas for each neighbourhood in shorthand. $\sum_d$ indicates $\sum_{d \in \mathcal{D}}$, and $\sum_e$ indicates $\sum_{e \in \mathcal{E}}$. Likewise, $\forall e$ indicates $\forall e \in \mathcal{E}$ and $\forall d$ indicates $\forall d \in \mathcal{D}$.

used maximum shift changes per employee neighbourhood $\sum_d \delta_{ed} \leq k \; \forall \; e$. However, several researchers have used one of these 14 neighbourhood restrictions on a standard MIP without column generation. Rahimian et al. (2017b) used fixed employees neighbourhood $\delta_e = 0 \; \forall \; e \notin \{j, \ldots, j+k\} \subseteq \mathcal{E}$; Santos et al. (2016) used fixed days neighbourhood $\delta_d = 0 \; \forall \; d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$ and fixed days (on/off) neighbourhood $\delta^{\text{on}}_d = 0 \; \forall \; d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$; Rahimian et al. (2017b) used fixed days neighbourhood, fixed employees neighbourhood and maximum roster modifications neighbourhood $\sum_e \sum_d \delta_{ed} \leq k$. Please see §3.2.7 for more details.

In practice, each of these neighbourhoods requires modifying different parts of our branch-and-price algorithm. The fixed sub-roster neighbourhoods only require modifying the input data to the column generation subproblem for each employee. These neighbourhoods are the easiest to implement as we do not need to change the structure of our column generation subproblem or restricted master problem; see §6.3 for details. The maximum employee changes neighbourhood only requires adding a single constraint to the restricted master problem and no changes to the column generator or column generator input. Thus, it is also easy to implement; see §6.2.3. The maximum shift changes per employee and maximum on/off changes per employee neighbourhoods only require adding a single resource to the column generation subproblem to track shift or on/off changes; see §6.2.1-6.2.2. This resource is easy to add using our generic resource modelling and automatic code generation. The remaining neighbourhoods require modifying both the restricted master problem and the column generation subproblem and, thus, are more challenging to implement.

The majority of time spent solving a column generation instance is spent solving the column generation subproblem. Thus, we implemented all of the neighbourhoods which modify the structure of or input to the column generation subproblem; please see Table 6.1 for more details. We refer to these neighbourhoods as "IP neighbourhood pricing" as we are solving our column generation subproblem or *pricing problem* in the neighbourhood of an IP solution. Later, we extend a subset of these methods in §7.2 to solve the pricing problem in the neighbourhood of an LP solution, which we refer to as "LP neighbourhood pricing".

We also implemented the maximum employee changes neighbourhood ($\sum_e \delta_e \leq k$) as the implementation was trivial. In the following sections, we show how these 14 neighbourhoods can be applied to a column generation framework for maximal effectiveness. We also compare these neighbourhoods' effectiveness in detail.

6.2 Maximum changes neighbourhood search

The first type of neighbourhood search we discuss is maximum changes neighbourhood search. As seen in §6.1, the maximum changes neighbourhoods are defined by a single parameter, k, which defines the maximum allowed distance from the incumbent roster. Thus, we represent a given maximum changes neighbourhood with a maximum changes neighbourhood function $N(\hat{X}, k)$ which applies the maximum changes neighbourhood to either the column generation subproblem or the RMP as appropriate.

In our experiments, we observed that the time taken to solve our neighbourhood constrained branch-and-price algorithm is typically exponential with respect to the neighbourhood distance parameter k; see §6.5.1 for more details. Thus, we first solve with the smallest possible neighbourhood distance parameter $k = 1$ and only increase our neighbourhood distance if our algorithm did not produce a lower cost roster solution with that neighbourhood distance. We define the cost $f_X(X)$ of a roster solution X as the objective function value of the RMP defined in (4.1), i.e., $f_X(X) = \sum_{e \in \mathcal{E}} \sum_{r \in \mathcal{R}_e} c_e^r \lambda_e^r + \sum_{D \in \mathcal{D}} c_D^- y_D^- + \sum_{D \in \mathcal{D}} c_D^+ y_D^+$.

We are essentially implementing a "basic sequential variable neighbourhood descent" procedure as outlined by Hansen et al. (2016) for each maximum changes neighbourhood.

For each maximum changes neighbourhood we discuss, there is a maximum possible neighbourhood distance $k_{\max}$. For example, the maximum shift changes neighbourhood has $k_{\max} = n$ where n is the total number of days. If the neighbourhood distance parameter is equal to its maximum value, i.e., $k = k_{\max}$, then the neighbourhood includes all possible solutions. Thus, each maximum changes neighbourhood search would find an optimal solution if a sufficient maximum run time was permitted. Therefore, each variable neighbourhood descent algorithm converges to global optimality and so is technically not a heuristic. However, in practice, our neighbourhood restricted branch and price only solves efficiently for the lower values of k.

Our sequential variable neighbourhood descent algorithm is shown in Algorithm 7. We continue solving our neighbourhood restricted branch-and-price algorithm with a given neighbourhood distance parameter k until no improvements to the incumbent roster can be made. The

neighbourhood distance parameter is then increased by one. To speed up the time taken to find an integer roster solution with each neighbourhood constraint, we perform a branch-and-price dive find an integer roster solution. Refer to §4.5 for more details on diving.

For a better understanding of how our variable neighbourhood descent algorithms work, refer to the graphs in §6.5.1. These graphs show the objective of all solutions to the restricted master problem and all integer incumbent solutions found over time.

Algorithm 7 Sequential variable neighbourhood descent with maximum changes neighbourhoods

procedure VND($X, k_{\max}, N(\hat{X}, k)$) ▷ We require an initial incumbent solution, X, the largest possible value of our neighbourhood distance parameter, $k_{\max}$ and a maximum changes neighbourhood function $N(\hat{X}, k)$.
  $k \leftarrow 1$
  while $k \leq k_{\max}$ **do**
    $\hat{X} \leftarrow X$
    APPLY neighbourhood $N(\hat{X}, k)$ to RMP or column generator
    $X \leftarrow$ solution to branch-and-price dive with neighbourhood restrictions
    if $f_X(X) < f_X(\hat{X})$ **then**
      $\hat{X} \leftarrow X$
      $k \leftarrow 1$
    else
      $k \leftarrow k + 1$
  return X

6.2.1 Maximum shift changes per employee

The first maximum changes neighbourhood search method uses the maximum shift changes per employee neighbourhood ($\sum_d \delta_{ed} \leq k \; \forall \; e$). This neighbourhood limits the number of differences between each employee's roster-line in the incumbent solution and that employee's roster-line in any new solution produced. In practice, we enforce this neighbourhood purely in the column generator. If we are generating a roster-line for a given employee, we define the incumbent roster-line as the roster-line worked by that employee in the incumbent roster solution.

We enforce this neighbourhood in the column generator by adding an additional resource to the resource vectors for each of the entities within our nested Shortest Path Problem with Resource Constraints (SPPRC). We denote each of these resources *incumbent difference*, since they are a count of the number of days with different activities from the incumbent roster-line. For a given employee e, the incumbent difference, $\mathcal{T}_S^{\text{Incumbent difference}}$, for each shift, S, is one if this shift is not in employee e's incumbent roster-line, i.e.,

$$\mathcal{T}_S^{\text{Incumbent difference}} = \begin{cases} 0, & \text{if } \hat{x}_{eS} = 1 \\ 1, & \text{otherwise} \end{cases}$$

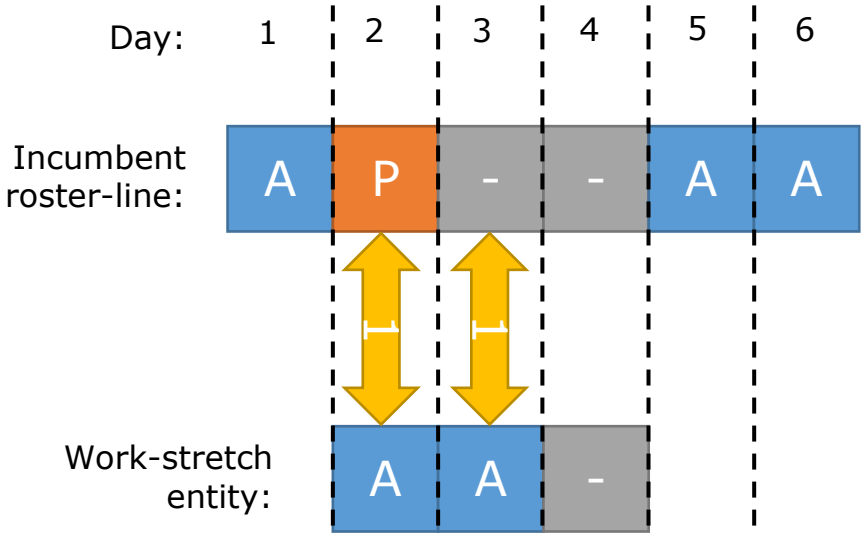


Figure 6.2: In this example, we count the number of incumbent differences for a 3 day work-stretch entity starting on day 2 in comparison with an incumbent full roster-line. Here we have an incumbent difference of two in the work-stretch entity shown above. Each arrow indicates one incumbent difference.

We define the incumbent difference $\mathcal{T}_F^{\text{Incumbent difference}}$ for each off-stretch F as the number of shifts in the incumbent roster-line worked during the days off covered by that off-stretch, i.e.,

$$\mathcal{T}_F^{\text{Incumbent difference}} = \sum_{d \in (d^1,\ldots,d^2)} \sum_{S \in \mathcal{S}^d} \hat{x}_{eS}$$

where d^1 and d^2 are the first and last days of the off-stretch respectively.

For a given on-stretch O, work-stretch W or roster-line R, the incumbent difference is merely the sum of the incumbent differences of its respective component entities. The REFs for each of these entities are as follows:

$$\begin{aligned}
\mathcal{E}_{\text{O}}^{\text{Incumbent difference}}(\mathcal{T}_O, \mathcal{T}_S) &= \mathcal{T}_O^{\text{Incumbent difference}} + \mathcal{T}_S^{\text{Incumbent difference}} \\
\mathcal{E}_{\text{W}}^{\text{Incumbent difference}}(\mathcal{T}_O, \mathcal{T}_F) &= \mathcal{T}_O^{\text{Incumbent difference}} + \mathcal{T}_F^{\text{Incumbent difference}} \\
\mathcal{E}_{\text{R}}^{\text{Incumbent difference}}(\mathcal{T}_R, \mathcal{T}_W) &= \mathcal{T}_R^{\text{Incumbent difference}} + \mathcal{T}_W^{\text{Incumbent difference}}
\end{aligned}$$

Each incumbent difference resource has an upper bound equal to the neighbourhood distance parameter k, i.e. $\gamma_O^{\text{Incumbent difference}} = \gamma_W^{\text{Incumbent difference}} = \gamma_R^{\text{Incumbent difference}} = k$. This neighbourhood distance parameter k represents the maximum number of differences from our incumbent roster-line. Figure 6.2 provides an example of how we can calculate the incumbent difference resource for a given work-stretch.

This neighbourhood significantly reduces the number of entities generated in our column generation subproblem. We do not create any entities which contain more roster-line modifications than we are allowing.

6.2.2 Maximum on/off changes per employee

The second maximum changes neighbourhood search method uses the maximum on/off changes per employee neighbourhood ($\sum_d \delta^{\text{on}}_{ed} \leq k \; \forall \; e$). This neighbourhood is very similar in implementation to the neighbourhood described in §6.2.1 except in how we define the incumbent difference resource for shift entities.

For a given employee e, the incumbent difference, $\mathcal{T}_S^{\text{Incumbent difference}}$, for a given shift S worked on day d, is zero if any shift $S' \in \mathcal{S}^d$ is worked on day d in employee e's incumbent roster-line, i.e.,

$$\mathcal{T}_S^{\text{Incumbent difference}} = \begin{cases} 0, & \text{if } \sum_{S' \in \mathcal{S}^d} \hat{x}_{eS'} = 1 \text{ for } d \text{ s.t. } S \in \mathcal{S}^d \\ 1, & \text{otherwise} \end{cases}$$

This neighbourhood reduces the number of entities generated in our column generation subproblem to a lesser extent than maximum shift changes per employee neighbourhood for a given neighbourhood distance parameter k. As such, the column generation subproblem solves slower.

6.2.3 Maximum employee changes

The third maximum changes neighbourhood search method uses the maximum employee changes neighbourhood ($\sum_e \delta_e \leq k$). This neighbourhood limits the total number of employees with a roster-line that differs from their roster-line in the incumbent roster solution.

In practice, we enforce this neighbourhood in the RMP with the following constraint:

$$\sum_{e \in \mathcal{E}} \sum_{r \in \mathcal{R}_e} \epsilon_r \lambda_e^r \geq |\mathcal{E}| - k \tag{6.7}$$

where

$$\epsilon_r = \begin{cases} 1, & \text{if roster-line } r \text{ is in the incumbent solution} \\ 0, & \text{otherwise} \end{cases}$$

If we generate a further column r that is not part of the incumbent solution, we set $\epsilon_r = 0$. However, if we generate an incumbent column due to numerical error, we can still set $\epsilon_r = 0$. This is because any integer roster solution that is feasible with an incumbent column r s.t. $\epsilon_r = 0$ would also be feasible with an identical column but with $\epsilon_r = 1$ due to the "$\geq$" constraint.

As we always set $\epsilon_r = 0$ in any further columns generated, we do not need to consider the dual for this constraint in our column generation subproblem. Thus, this neighbourhood search has a trivial implementation overhead.

6.3 Fixed sub-roster neighbourhood search

The second type of incumbent search we discuss is fixed sub-roster neighbourhood search. As seen in §6.1, the fixed sub-roster neighbourhoods are defined by two parameters, the neighbourhood

distance parameter k and the neighbourhood location parameter j. Thus, we represent a given fixed sub-roster neighbourhood with a fixed sub-roster neighbourhood function $N(\hat{X}, k, j)$ which applies the fixed sub-roster neighbourhood to the column generation subproblem.

Like the maximum changes neighbourhood, each fixed sub-roster neighbourhood has a maximum possible neighbourhood distance parameter $k_{\max}$. If the neighbourhood distance parameter is equal to its maximum value, i.e., $k = k_{\max}$ and $j = 0$, then the problem will be solved to global optimality, given an infinite amount of time. Thus, we could use the same neighbourhood search as in §6.2 with $j = 0$. In our experiments, we observed that the time taken to solve our neighbourhood constrained branch-and-price algorithm is typically exponential with respect to the neighbourhood distance parameter k, but not with respect to the neighbourhood location parameter j; see §6.5 for more details. Thus, it is usually better to search in different parts of the solution space with a lower neighbourhood distance than increase the neighbourhood distance. Therefore, we only increase the neighbourhood distance parameter k if we can not find any improved solutions by changing the neighbourhood location parameter j.

We are practically implementing the "mixed variable neighbourhood descent" procedure outlined by Hansen et al. (2016) for each fixed sub-roster neighbourhood. This algorithm is a "cyclic variable neighbourhood descent" nested inside a "sequential variable neighbourhood descent."

Our mixed variable neighbourhood descent algorithm is shown in Algorithm 8. For a given neighbourhood distance parameter k, we solve for every possible neighbourhood location parameter j. If no improvements were made, the neighbourhood distance parameter is then increased by one.

Algorithm 8 Mixed variable neighbourhood descent with fixed sub-roster neighbourhoods

1: **procedure** VND($X, k_{\max}, j_{\max}, N(\hat{X}, k, j)$) ▷ We require an initial incumbent solution, X, the largest possible value of our neighbourhood distance parameter, $k_{\max}$, the largest possible value of our location parameter j and a fixed sub-roster neighbourhood function $N(\hat{X}, k, j)$.
2: $k \leftarrow 1$
3: $\hat{X} \leftarrow X$
4: **while** $k \leq k_{\max}$ **do**
5: **for** $j \in (0, 1, 2, \ldots, j_{\max})$ **do**
6: APPLY neighbourhood $N(\hat{X}, k, j)$
7: $X \leftarrow$ solution to branch-and-price dive
8: **if** $f_X(X) < f_X(\hat{X})$ **then**
9: $\hat{X} \leftarrow X$
10: **if** $f_X(X) < f_X(\hat{X})$ **then**
11: $k \leftarrow 1$
12: **else**
13: $k \leftarrow k + 1$
14: **return** X

6.3.1 Fixed days

The first fixed sub-roster neighbourhood search method uses the fixed days neighbourhood ($\delta_d = 0 \ \forall \ d \notin \{j, \dots, j+k\} \subseteq \mathcal{D}$). This neighbourhood involves fixing a subset of the days in any solution produced to be equal to those of an incumbent solution.

We impose this neighbourhood by modifying the input to the column generation subproblem for each employee $e \in \mathcal{E}$. We remove any shifts from e's column generation subproblem if they are on one of the set of fixed days $d \in (1, 2, \dots, j-1, j+k+1, \dots, n)$ and are not in employee e's incumbent roster-line. Likewise, we remove any off-stretches from employee e's column generation subproblem if they cover a fixed day, which is worked in employee e's incumbent roster-line. The process of modifying the input to employee e's column generation subproblem is shown in Algorithm 9.

Algorithm 9 Fixed days column generator preprocessor

```
1: procedure FIX DAYS(X̂, j, k)        ▷ We require an incumbent solution, X̂, the first
   unfixed day, j, and the total number of unfixed days k.
2:     for d ∈ (1, 2, ..., j − 1, j + k + 1, ..., n) do
3:         if Σ_{S∈S^d} x̂_eS > 0 then        ▷ If day d is worked in the incumbent solution.
4:             for F ∈ F^{d1,d2} do ▷ F^{d1,d2} is the set of off-stretches from day d^1 to day d^2
5:                 if d^1 ≤ d ≤ d^2 then
6:                     REMOVE off-stretch F        ▷ i.e. we don't consider off-stretch F
                                                      in the work-stretch SPPRC graph
7:         for S ∈ S^d do
8:             if x̂_eS = 0 then        ▷ If shift S is not worked in the incumbent solution.
9:                 REMOVE shift S    ▷ i.e. we don't consider shift S in the on-stretch
                                         SPPRC graph
```

6.3.2 Fixed employees

The second fixed sub-roster neighbourhood search method uses the fixed employees neighbourhood ($\delta_e = 0 \ \forall \ e \notin \{j, \dots, j+k\} \subseteq \mathcal{E}$). This neighbourhood involves choosing a subset of employees and fixing their roster-lines to be equal to their roster-lines in the incumbent solution.

In practice, initially, we remove all columns except the incumbent column for each fixed employee $e \in (1, 2, \dots, j-1, \dots, j+k+1, \dots, |\mathcal{E}|)$ from the RMP. While solving the neighbourhood restricted branch and price, we do not generate any further columns for the fixed employees.

6.3.3 Fixed days (on/off)

The third fixed sub-roster neighbourhood search method uses the fixed days (on/off) neighbourhood ($\delta_d^{\text{on}} = 0 \ \forall \ d \notin \{j, \dots, j+k\} \subseteq \mathcal{D}$). This neighbourhood involves fixing certain days such that if an employee works on a fixed day $d \in (1, 2, \dots, j-1, j+k+1, \dots, n)$ in their incumbent

roster-line, they also work that day d in any solutions produced. Conversely, if an employee has a day off on a fixed day in their incumbent roster-line, they also have that day off in any solutions produced.

We impose this neighbourhood by modifying the input to the column generation subproblem for each employee $e \in \mathcal{E}$ similarly to in §6.3.1. However, we only remove a given shift from the column generation subproblem if it resides on a day off in employee e's incumbent roster-line. The process of modifying the input to employee e's column generation subproblem is shown in Algorithm 10.

Algorithm 10 Fixed days (on/off) column generator preprocessor

procedure FIX DAYS (ON/OFF)($\hat{X}, j, k$) ▷ We require an incumbent solution, $\hat{X}$, the first unfixed day, j, and the total number of unfixed days k.
  for $d \in (1, 2, \ldots, j-1, j+k+1, \ldots, n)$ **do**
    if $\sum_{S \in \mathcal{S}^d} \hat{x}_{eS} > 0$ **then** ▷ If day d is worked in the incumbent solution.
      for $F \in \mathcal{F}^{d^1,d^2}$ **do** ▷ $\mathcal{F}^{d^1,d^2}$ is the set of off-stretches from day d^1 to day d^2
        if $d^1 \leq d \leq d^2$ **then**
          REMOVE off-stretch F ▷ i.e. we don't consider off-stretch F in the work-stretch SPPRC graph
    else
      for $S \in \mathcal{S}^d$ **do**
        REMOVE shift S ▷ i.e. we don't consider shift S in the on-stretch SPPRC graph

6.3.4 Fixed employees (on/off)

The fourth fixed sub-roster neighbourhood search method uses the fixed employees (on/off) neighbourhood ($\delta_e^{\text{on}} = 0 \; \forall \; e \notin \{j, \ldots, j+k\} \subseteq \mathcal{E}$). This neighbourhood involves fixing a subset of the employees. If a fixed employee works on any given day in their incumbent roster-line, they also work that day in any solutions produced. Conversely, if a fixed employee has a day off on any given day in their incumbent roster-line, they also have that day off in any solutions produced.

In practice, we impose this neighbourhood by modifying the input to the column generation subproblem for each fixed employee $e \in (1, 2, \ldots, j-1, j+k+1, \ldots, |\mathcal{E}|)$. The process of modifying the input to fixed employee e's column generation subproblem is shown in Algorithm 11. The implementation of this neighbourhood is similar to fixed days (on/off) neighbourhood in §6.3.1; however, we are fixing every day instead of a subset of days.

6.4 Testing methodology

As we first introduced in §1.3, we have two different problems, the (easier) operating theatre ward problem and the (more challenging) maternity wards problem. We test our neighbourhood

Algorithm 11 Fixed employees (on/off) column generator preprocessor

```
1: procedure FIX EMPLOYEES (ON/OFF)(X̂)        ▷ We only require an incumbent solution, X̂ as we fix all days for the fixed employees.
2:     for d ∈ (1, 2, ..., n) do
3:         if Σ_{S∈S^d} x̂_eS > 0 then        ▷ If day d is worked in the incumbent solution.
4:             for F ∈ F^{d1,d2} do ▷ F^{d1,d2} is the set of off-stretches from day d^1 to day d^2
5:                 if d^1 ≤ d ≤ d^2 then
6:                     REMOVE off-stretch F    ▷ i.e. we don't consider off-stretch F in the work-stretch SPPRC graph
7:         else
8:             for S ∈ S^d do
9:                 REMOVE shift S    ▷ i.e. we don't consider shift S in the on-stretch SPPRC graph
```

restricted column generation algorithms on both of these problems to show how the matheuristics behave when solving these two problems.

We compare each algorithm's effectiveness in terms of the objective of the best incumbent solution found over time. We observed that some of our matheuristics are more effective at improving higher cost roster solutions, and some are more effective at improving lower cost roster solutions; see §6.5.2 for details. Thus, we have a high-quality and low-quality initial incumbent roster solution for each problem. With two problems and two initial incumbent roster solutions, we have four experiments total.

The first initial incumbent solution for each problem was created by generating the lowest cost roster-line for each employee. When generating these roster-lines, we ignore the duals from each demand, i.e., $\pi_D = 0 \quad \forall D \in \mathcal{D}$. Thus, if there were no constraints on the demand, this would be the lowest cost solution possible. However, there are soft constraints with high costs on demands in practice, so this was a low-quality solution; see §4.1.1 for more details on demand costs and duals. The other initial incumbent solution is the first integer solution found by a single dive from an optimal LP solution with no heuristics or neighbourhood restrictions. This initial incumbent solution is of a much higher quality than the first.

One of the problems with performing experiments that involve a branch-and-bound dive is that small changes to the problem (such as simply reordering the constraints; see Fischetti and Monaci (2014)) can have a large effect on which branches are selected and the resulting solution quality. This effect is mitigated as we are running a large number of branch-and-bound dives per experiment during each variable neighbourhood descent and reporting averaged results. However, to further mitigate the variance, we decided to create ten instances of each of our four experiments, with slight differences in the employee shift costs. As roster history (see §4.2.6) only affects two resources in the whole problem, and employee contracts don't change from month to month, the main difference from month to month for the Waikato DHB problems was employee preferences. Thus, this was an adequate emulation of how the roster could change in real life from month to month.

To create each of these instances in a repeatable way, we set 10 different seeds for the random shift cost perturbations, first described in §5.2.2. We performed a brief experiment to verify that the random shift cost perturbations changed the branch-and-bound tree in a meaningful way. In this experiment, we performed a standard dive from an optimal LP solution with no neighbourhood restrictions and random shift cost perturbations generated with each of the ten seeds. We identified, on average, a 5% similarity in branches made by diving between the ten sets of random shift costs. We calculated the similarity of branches between two dives as the number of employee shift variables which were branched on in both dives divided by the average number of branches for the two dives. Thus, we can infer that our branch-and-bound dive was different each time, which suggests we have sufficient variability to ensure we are not repeating the same local search for each instance. We also observed that we obtained a different solution for each dive.

As the total cost contribution from the random shift cost perturbations is minimal compared to the cost of the original objective function, we can use the same incumbent solution across all ten instances of a given experiment; those incumbent solutions all start with a similar objective.

Thus, with two problems, two initial incumbent solutions and ten instances of each experiment, we performed 40 runs in total for each variable neighbourhood descent heuristic. As we are comparing seven different heuristics, we needed to complete 280 runs in total. Due to the difference in problem difficulty, the 140 runs involving the maternity wards problem had a 12-hour time limit, and the 140 runs for the operating theatre ward problem had a one hour time-limit. If we were to run all the tests on a single-core, they would take 1820 hours (about 2.5 months) total.

We record the lowest cost roster solution found at various times to show the improvement over time for our variable neighbourhood descent algorithms. We recognise that it is possible that by imposing a neighbourhood search in the column generation subproblem, all we are doing is speeding up the column generation subproblem. Thus, we track how many column generation iterations and dives we perform in the set time with each variable neighbourhood descent algorithm.

Further, we are interested in whether the neighbourhood structure strongly changes the nature of the solves, specifically the "integrality" or how close solutions to the root node of the branch-and-price search tree are to the integer solution produced by diving. Thus, we also recorded the time taken to solve the root node and two integrality measures for each dive in each run: the number of non-zero columns in the root node solution and the number of branches required for each dive. Each variable neighbourhood descent algorithm involved many dives, so we report the average of each of these metrics across the run.

6.5 Results

The following sections detail the results of all 280 runs. Not only do we compare the relative effectiveness of each of our seven variable neighbourhood descent algorithms, but we also show

how the neighbourhood restrictions affect solving the root node and the resulting branch-and-price search tree.

6.5.1 Single run example

To demonstrate how our variable neighbourhood descent algorithms work, in this section, we show the results from a single run using the "maximum shift changes per employee" neighbourhood ($\sum_d \delta_{ed} \leq k \; \forall \; e$). During this run, we recorded the objective of every solution of the restricted master problem (RMP) generated during the branch-and-price dives. Note that we refer to a dive as inclusive of solving the root node in the following sections.

A graph of the RMP solution's objective over time is shown in Figure 6.3. We show the objective of the best incumbent integer solution found at the end of each branch-and-price dive in the variable neighbourhood descent with an 'x'. Lighter colours/shades for the 'x's indicate the end of a neighbourhood restricted dive with a larger neighbourhood distance parameter k. We also provide a graph of a small subset of the RMP solutions from the same run in Figure 6.4. With the smaller values of k, the improvements are small, but the dives take a very short amount of time to complete.

As we are trying to find the incumbent solution with the smallest objective, if a neighbourhood restricted dive does not find an improved incumbent solution, the best incumbent solution remains the same. Thus, if the objective values of the RMP solutions (indicated by dots) produced during the dive are higher than the objective value of the best incumbent integer solution (indicated by an 'x'), then the next 'x' will be at the same objective value as the previous 'x'.

On the later solves in Figure 6.3, we can see a noticeable dip in the objective while the root node is being solved, followed by a rise caused by branching. However, in Figure 6.4, we observe that many of the early dives have naturally integer root nodes and thus, require little or no branching. Thus, in some of the early dives, we can see no rise from branching.

The first dive in Figure 6.4 has a neighbourhood distance $k = 4$. An improved incumbent solution is found, and thus the neighbourhood distance resets to one. The next two dives with $k = 1$ are naturally integer and lead to a very small improvement in the incumbent solution. However, the next dive leads to no improved incumbent solution, and so the neighbourhood distance is increased to $k = 2$.

One observation we made was that after a large enough neighbourhood distance parameter, k, our dives are ineffective at producing improved incumbent solutions. By definition, solving with a large neighbourhood distance k is the equivalent of having no neighbourhood restriction. For example, if we are using the maximum shift changes per employee neighbourhood $\sum_e \sum_d \delta_{ed} \leq k$ and $k = n$, then every day can change for every employee, and there are effectively no neighbourhood restrictions. By extrapolation, we can assume that diving with no neighbourhood restrictions is unlikely to produce a better solution than variable neighbourhood descent can produce.

To confirm our assumption and to further demonstrate why column generation based local search is so effective compared to a branch-and-price dive without neighbourhood restrictions, we

compare both methods on the same graph in Figure 6.5. This graph shows that solving an LP all the way down and branching all the way up is less effective than imposing a neighbourhood and avoiding a descent and climb. The much higher quality (lower objective) of the root node solution is wasted as many more branches are required to reach an integer solution. Those branches drive up the objective significantly.

Another observation is that while solving the start of the root node for the last two dives in Figure 6.3, columns are added to the RMP, but no improvement is made to the RMP objective. Thus, we observe a small flat line at the start of solving these two root nodes. We propose that this occurs because we are starting the RMP with only the columns from the previous best incumbent integer solution along with the slack and surplus columns. Because the previous best incumbent integer solution is of a very high quality, any solution we generate by swapping columns in the best incumbent integer solution with newly generated columns is likely to be worse than the incumbent solution because it requires additional slacks and surpluses to be at non-zero values which increases the objective value. Also, since during these branch-and-price dives, we are generating columns with a large number of possible changes to the incumbent, i.e., $k \geq 8$, the columns generated tend to have much larger differences to the columns in the incumbent solution. This means that replacing a column in the incumbent solution with the newly generated column is more likely to require non-zero slacks and surpluses.

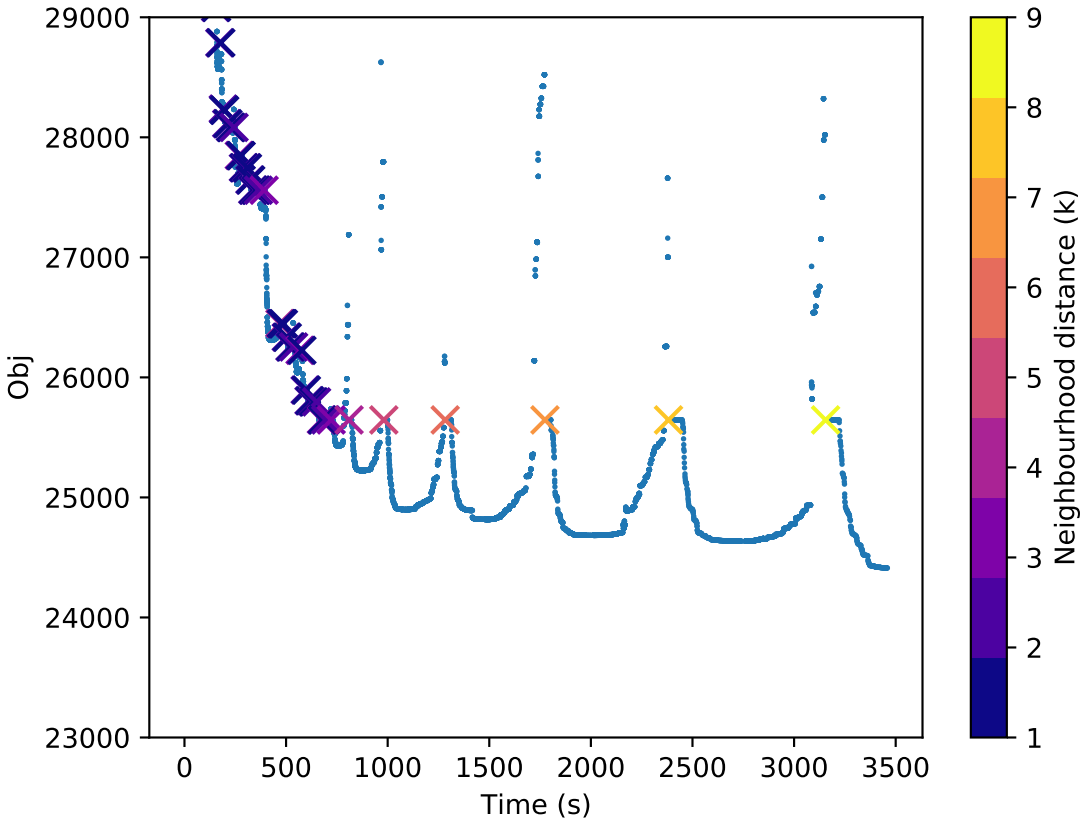


Figure 6.3: The objective of each solution to the RMP during a single run of solving the operating theatre ward problem with the maximum shift changes per employee neighbourhood ($\sum_d \delta_{ed} \leq k \; \forall \; e$). Each 'x' is the best incumbent roster solutions found after a neighbourhood restricted dive. Lighter colours/shades of 'x' indicate starting a dive with a larger neighbourhood distance parameter k where $1 \leq k \leq 9$.

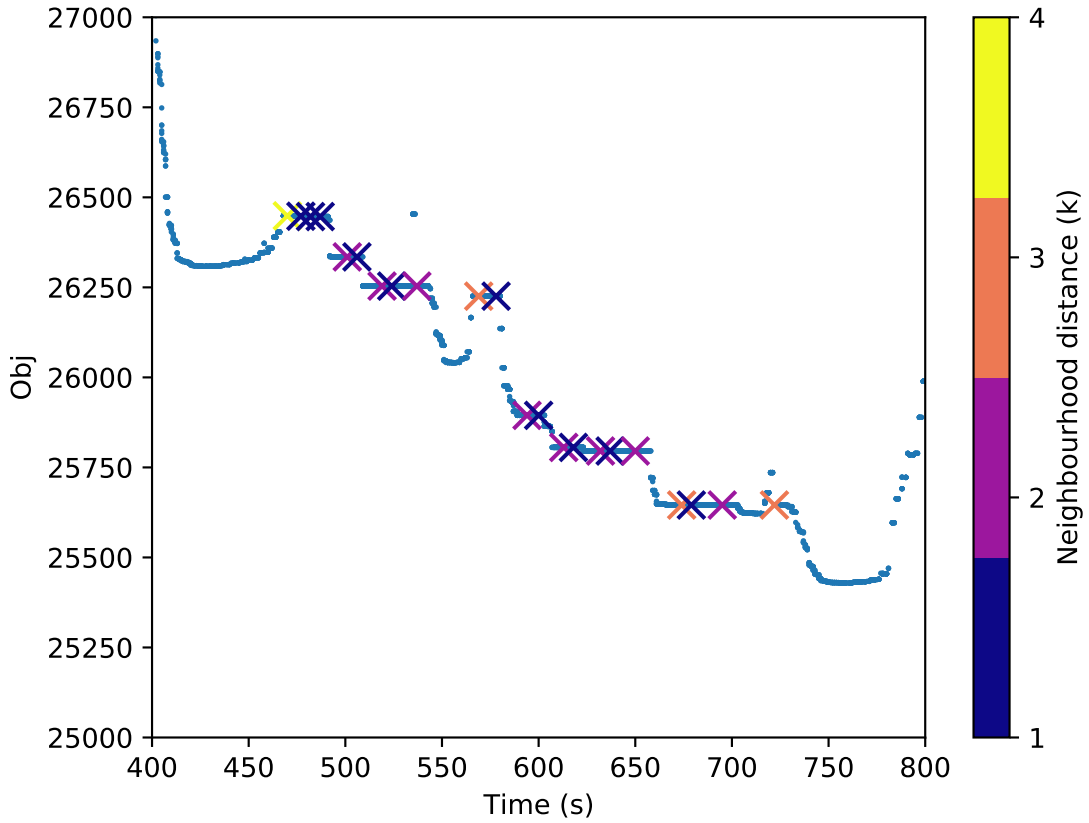


Figure 6.4: The same as Figure 6.3 except zoomed in over a small subset of the total time and with a small range of neighbourhood distance parameter k: $1 \leq k \leq 4$.

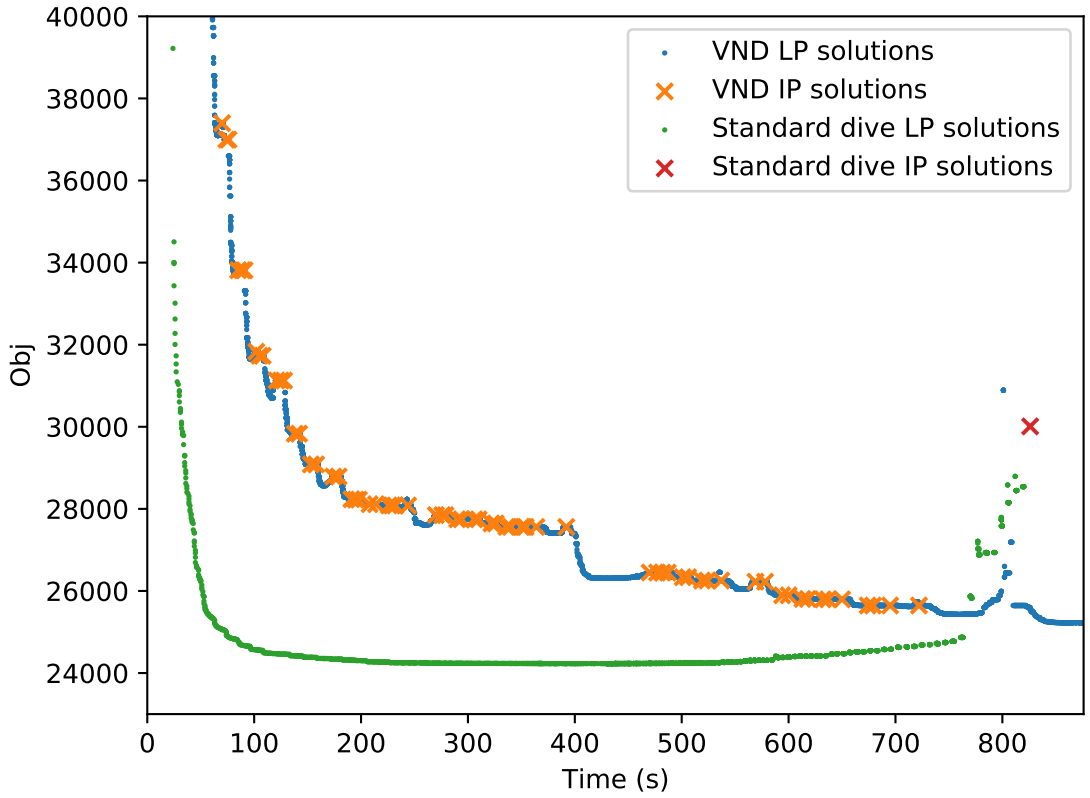


Figure 6.5: The objective of each solution to the RMP during a single run of solving the operating theatre ward problem with the maximum shift changes per employee neighbourhood ($\sum_d \delta_{ed} \leq k \; \forall \; e$) compared to a single branch-and-price dive without neighbourhood restrictions. Each 'x' is an integer roster solution found. The objective drops while the root node is being solved and then rises with branching until an integer solution is found. For the VND, this pattern is replicated multiple times for each iteration.

6.5.2 All runs results

This section presents the results for all 280 runs. We performed all 280 runs and recorded the objective of the best incumbent solution found at various intervals for each problem. As the Maternity wards problem is significantly more difficult than the Operating theatre ward problem, we recorded the objective of the best incumbent solution produced at different times for the two problems. We recorded the incumbent solution objective for the maternity wards problem after 10 minutes, one hour and 12 hours, and for the operating theatre ward problem after one minute, 15 minutes and one hour. The results for the 140 operating theatre ward problem runs are shown using bar and whisker plots in Figure 6.6 and the results for the 140 maternity wards problem runs are shown using bar and whisker plots in Figure 6.7. Each box and whisker represents ten runs of solving one of our two problems with a neighbourhood descent algorithm and with a low-quality or high-quality initial incumbent solution.

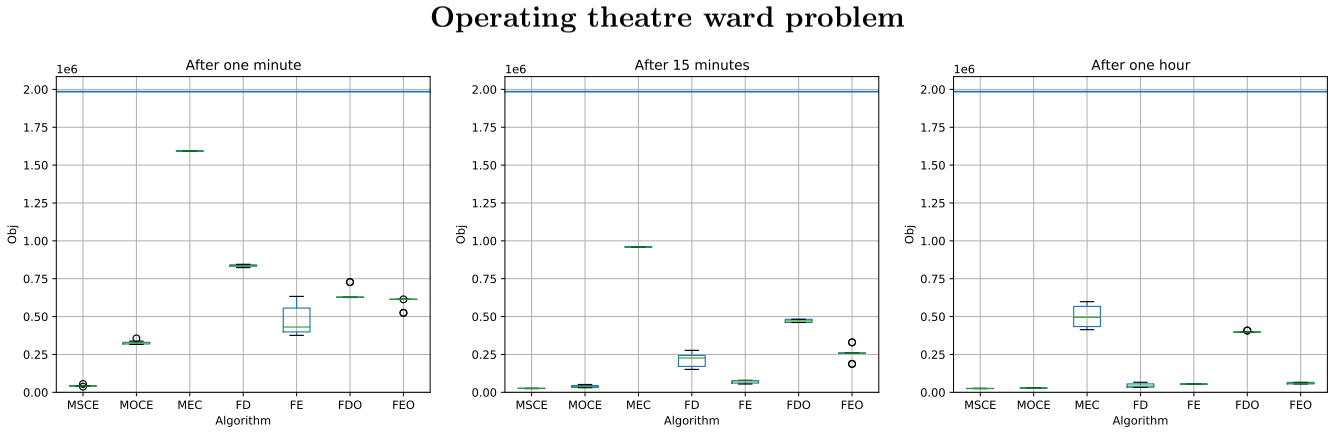
We observed that there is no best algorithm for all experiments at every measured interval. We define the best algorithm for a given experiment and interval as the one with the lowest average objective. However, there are some algorithms that are best to use under certain conditions. When starting with a low-quality initial incumbent solution and a limited amount of time ($\leq$ one hour) for both problems, the maximum shift changes per employee neighbourhood ($\sum_d \delta_{ed} \leq k \; \forall \; e$) produces the lowest average objective incumbent solution by a significant margin.

When starting with a high-quality initial incumbent solution to the (more challenging) maternity wards problem, the fixed days ($\delta_d = 0 \; \forall \; d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$) neighbourhood produces the lowest average objective incumbent solutions. However, when starting with a high-quality initial incumbent solution to the operating theatre ward problem, multiple algorithms have similar average incumbent solution quality.

We also note that the neighbourhood search algorithms which generate a larger number of entities are better in solving the operating theatre ward problem than the (more challenging) maternity wards problem. The on/off modification equivalent of a neighbourhood always generates a larger number of entities and performs relatively better when solving the (easier) operating theatre ward problem. For example, the maximum on/off changes per employee neighbourhood ($\sum_d \delta^{\text{on}}_{ed} \leq k \; \forall \; e$) performs better relative to the maximum shift changes per employee neighbourhood ($\sum_d \delta_{ed} \leq k \; \forall \; e$) in the maternity wards problem than in the operating theatre wards problem.

We observe that some heuristics have less spread of the objective value of the best incumbent solution found at certain times than others. For example, the objective value of all the best incumbent solutions produced by the fixed days algorithm ($\delta_d = 0 \; \forall \; d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$) are relatively similar. However, there is a much greater spread with the maximum on/off changes per employee algorithm ($\sum_d \delta^{\text{on}}_{ed} \leq k \; \forall \; e$). We suspect this is because of our observation that most dives within the fixed days algorithm have a naturally integer root node. In contrast, most dives within the maximum on/off changes per employee algorithm require many branches to reach an integer incumbent solution. We discuss this further in §6.5.3.

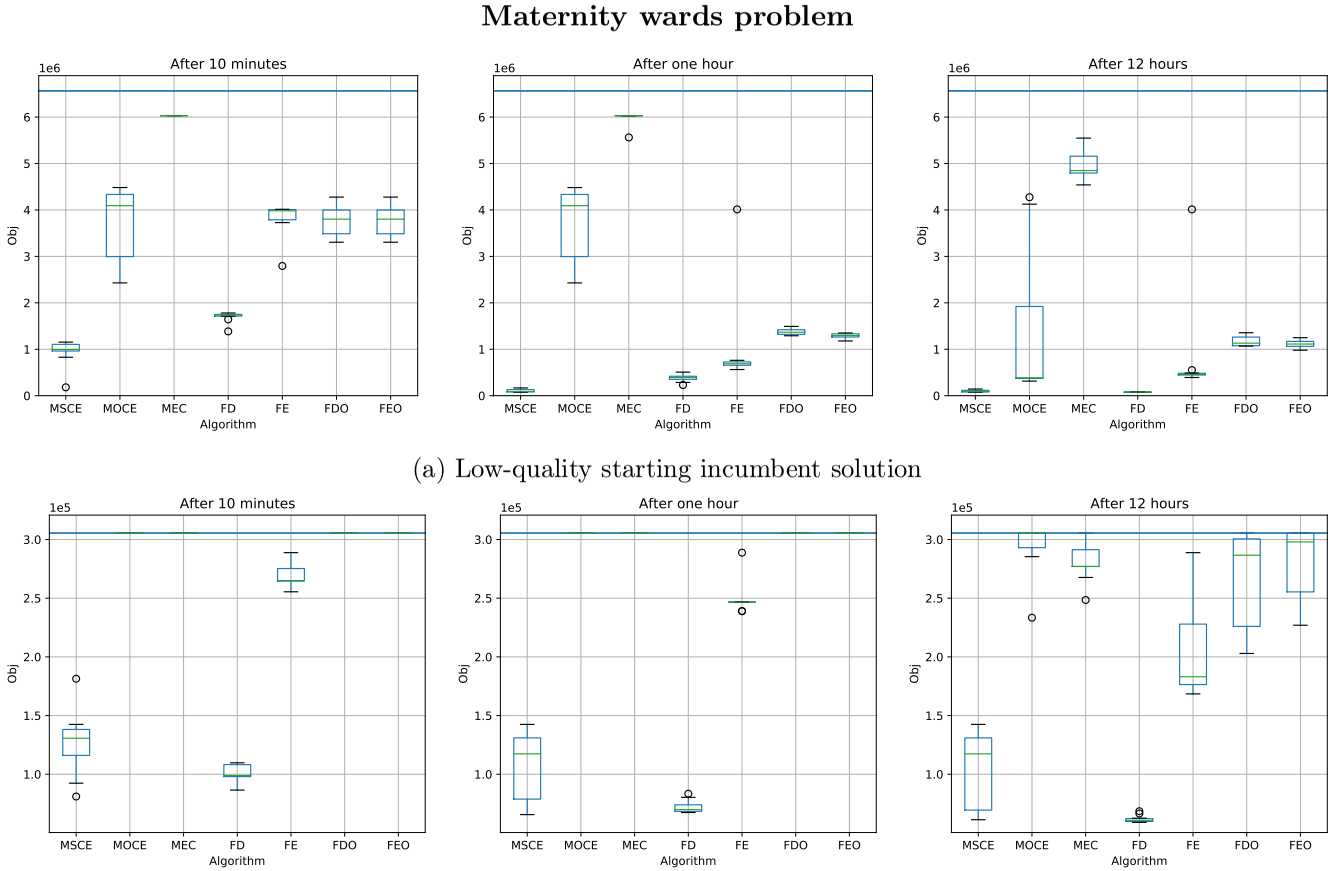
We observe that for the maternity wards problem, some of the algorithms gave an immediate and significant improvement in the quality of the high-quality initial incumbent solution. Within our cost structure, a cost of 10,000 indicates violation of a relaxed hard constraint. In only 10 minutes, the fixed days algorithm ($\delta_d = 0 \; \forall \; d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$) reduced the equivalent number of 'hard' constraints violated by around 20. This reduction is a considerable improvement in the quality of the roster solution. The high-quality initial incumbent solution to the operating theatre ward problem was of a much higher quality and, thus, could not be improved as much.



(a) Low-quality starting incumbent solution

(b) High-quality starting incumbent solution

Figure 6.6: Comparison of the objective of the lowest cost incumbent roster solution produced after one minute, 15 minutes and one hour in the 140 runs for the **operating theatre ward problem**. The graphs on the top show the results from the runs starting with a low-quality initial incumbent solution, and the graphs on the bottom show the results from the runs starting with a high-quality initial incumbent solution. The objective of each initial incumbent solution is shown with a blue line. Each algorithm is a variable neighbourhood descent algorithm with the following neighbourhoods: maximum shift changes per employee (MSCE, $\sum_d \delta_{ed} \leq k \ \forall \ e$), maximum on/off changes per employee (MOCE, $\sum_d \delta^{\text{on}}_{ed} \leq k \ \forall \ e$), maximum employee changes (MEC, $\sum_e \delta_e \leq k$), fixed days (FD, $\delta_d = 0 \ \forall \ d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$), fixed employees (FE, $\delta_e = 0 \ \forall \ e \notin \{j, \ldots, j+k\} \subseteq \mathcal{E}$), fixed days (on/off) (FDO, $\delta^{\text{on}}_d = 0 \ \forall \ d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$), and fixed employees (on/off) (FEO, $\delta^{\text{on}}_e = 0 \ \forall \ e \notin \{j, \ldots, j+k\} \subseteq \mathcal{E}$).



(a) Low-quality starting incumbent solution

(b) High-quality starting incumbent solution

Figure 6.7: Comparison of the objective of the lowest cost incumbent roster solution produced after 10 minutes, one hour and 12 hours in the 140 runs for the **maternity wards problem**. The graphs on the top show the results from the runs starting with a low-quality initial incumbent solution, and the graphs on the bottom show the results from the runs starting with a high-quality initial incumbent solution. The objective of each initial incumbent solution is shown with a blue line. Each algorithm is a variable neighbourhood descent algorithm with the following neighbourhoods: maximum shift changes per employee (MSCE, $\sum_d \delta_{ed} \leq k \ \forall \ e$), maximum on/off changes per employee (MOCE, $\sum_d \delta^{\text{on}}_{ed} \leq k \ \forall \ e$), maximum employee changes (MEC, $\sum_e \delta_e \leq k$), fixed days (FD, $\delta_d = 0 \ \forall \ d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$), fixed employees (FE, $\delta_e = 0 \ \forall \ e \notin \{j, \ldots, j+k\} \subseteq \mathcal{E}$), fixed days (on/off) (FDO, $\delta^{\text{on}}_d = 0 \ \forall \ d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$), and fixed employees (on/off) (FEO, $\delta^{\text{on}}_e = 0 \ \forall \ e \notin \{j, \ldots, j+k\} \subseteq \mathcal{E}$).

6.5.3 Properties of dives

As mentioned in §6.4, we are interested in if the neighbourhood structure strongly changes the nature of the dives itself or is merely speeding up the column generator. This section presents our findings on each neighbourhood restriction's effect on the column generator and branch-and-price dives.

Firstly, we found the average number of dives and the average number of calls to the column generator for each employee per run with each neighbourhood across the two different problems. This data was collected over 12 hours for the maternity wards problem and 1 hour for the operating theatre ward problem. We show this in Figure 6.8. We found a wide discrepancy between our neighbourhoods in the number of branch-and-price dives that occurred within the given time. However, we note that the neighbourhoods with fewer dives still perform well in some cases. The average improvement in the objective of an incumbent solution per dive is higher in these cases to compensate for the fewer dives. For example, in some cases, variable neighbourhood descent using maximum shift changes per employee neighbourhood leads to lower average objective value incumbent solutions than using fixed days neighbourhood even though using fixed days neighbourhood means a much larger number of dives can occur.

We also observed that the neighbourhoods that can generate fewer possible entities tended to generate columns more quickly. For example, when using fixed days neighbourhood ($\delta_d = 0 \; \forall \; d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$), fewer possible entities are generated than using maximum shift changes per employee neighbourhood ($\sum_d \delta_{ed} \leq k \; \forall \; e$), and as a result, the column generator solves faster on average.

We then compared three different metrics for each run to see if the nature of each dive is changing or the change is just in the speed of solving the column generator. The first metric, "average number of non-zero columns by algorithm," is a measure of the root node solution's integrality. If the number of non-zero columns is equal to the number of employees, then our root node solution is naturally integer. The larger the number of non-zero columns, the more fractional our root node solution is. The second metric, "average time spent solving root node by algorithm", is a measure of the average time taken to solve each root node in the dives. The third metric, "average number of branches in dive by algorithm" is a measure of the average number of branches required to reach an integer roster solution in the dives.

In Figure 6.9, we compare these metrics for all 240 runs with box and whisker plots. We also added a "None" box and whisker to each plot which is the same metrics but for ten instances of a single dive with no neighbourhood restrictions.

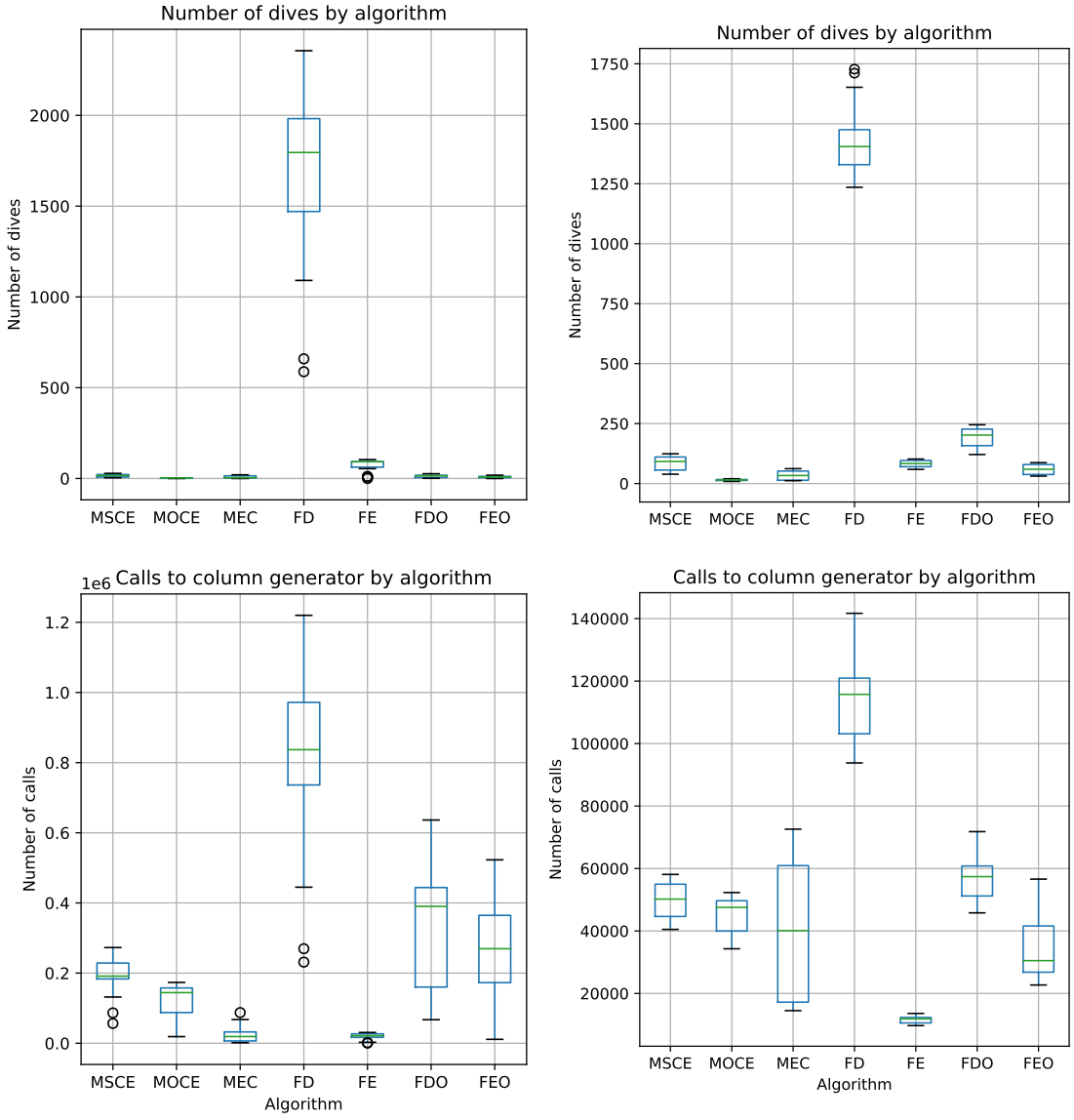
As shown by the top and bottom graphs in Figure 6.9, the neighbourhood restricted branch-and-price dives are more naturally integer than branch-and-price dives with no neighbourhood restriction. Thus, we suspect that neighbourhood restriction is having a larger effect on the branch-and-price dive than reducing the number of times that the column generator needs to be called.

We also observed that the relative difference in the time taken to solve the root node between the branch-and-price dives with neighbourhood restrictions and those without is much larger

for the maternity wards problem. However, the relative difference in integrality and number of branches between the branch-and-price dives with neighbourhood restrictions and those without were similar for the two problems.

Although these metrics vary considerably between the different algorithms, they also vary between branch-and-price dives with different neighbourhood distance parameters within the same algorithm. Thus, we took all dives in all 20 runs for the maximum shift changes per employee neighbourhood ($\sum_d \delta_{ed} \leq k \; \forall \; e$) and recorded the same metrics as above but for each neighbourhood distance parameter k. A set of box and whisker plots showing these experiments' results is demonstrated in Figure 6.10.

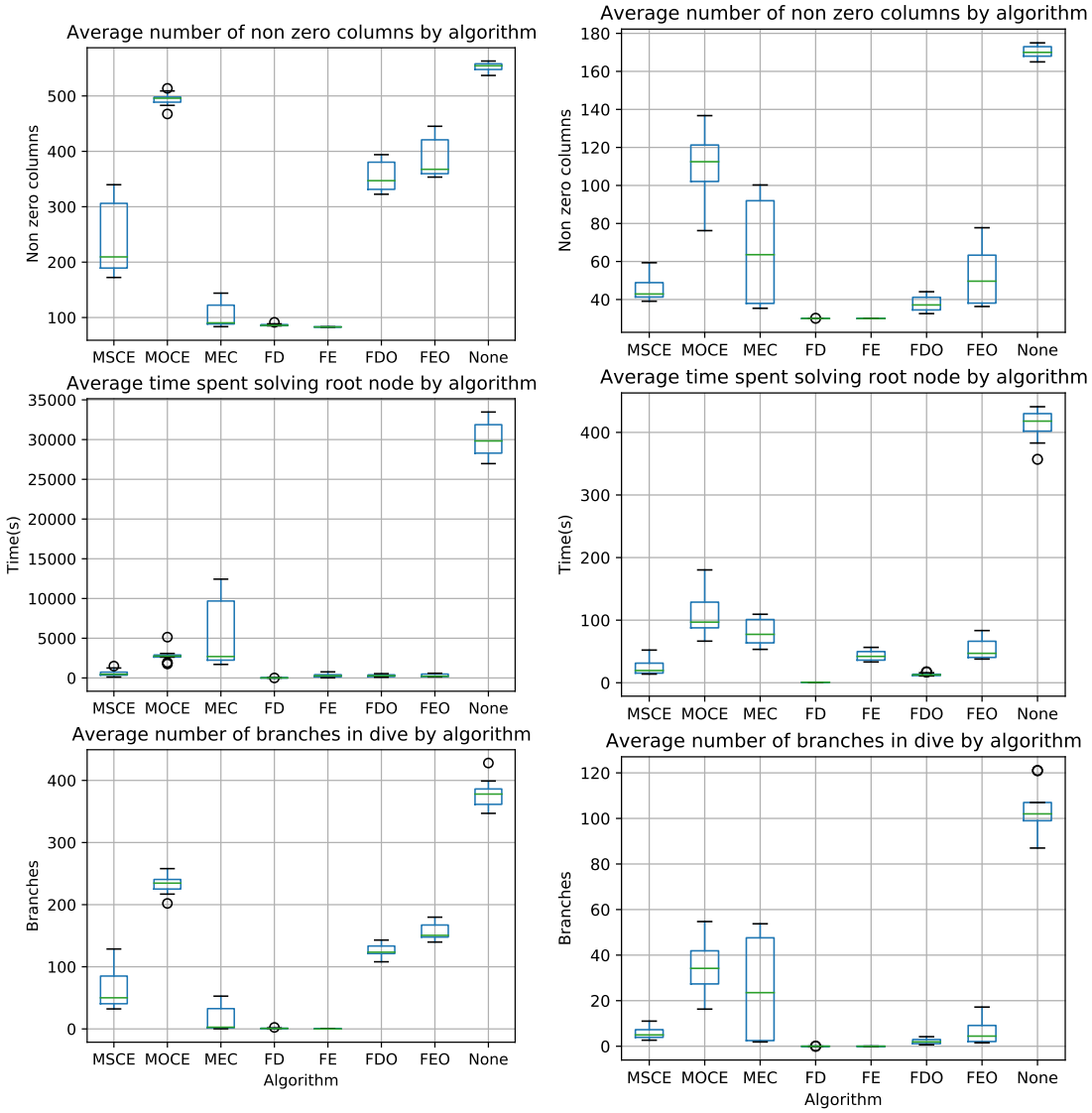
Recall that if the neighbourhood distance parameter reaches a maximum value, i.e., $k = k_{\max}$, then this is the equivalent of no neighbourhood restrictions. Therefore, as expected, the metrics all tend towards those of a dive with no neighbourhood restrictions. For example, in Figure 6.9, we can see that there are on average 560 non-zero columns in the root node solution to the maternity wards problem solved with no neighbourhood restrictions. We can compare this to the top left graph in Figure 6.10 where when our neighbourhood distance parameter $k = 6$, there are on average 500 non-zero columns in the root node solution. This is much less integer than the same neighbourhood but with $k = 1$.



(a) Maternity wards problem

(b) Operating theatre ward problem

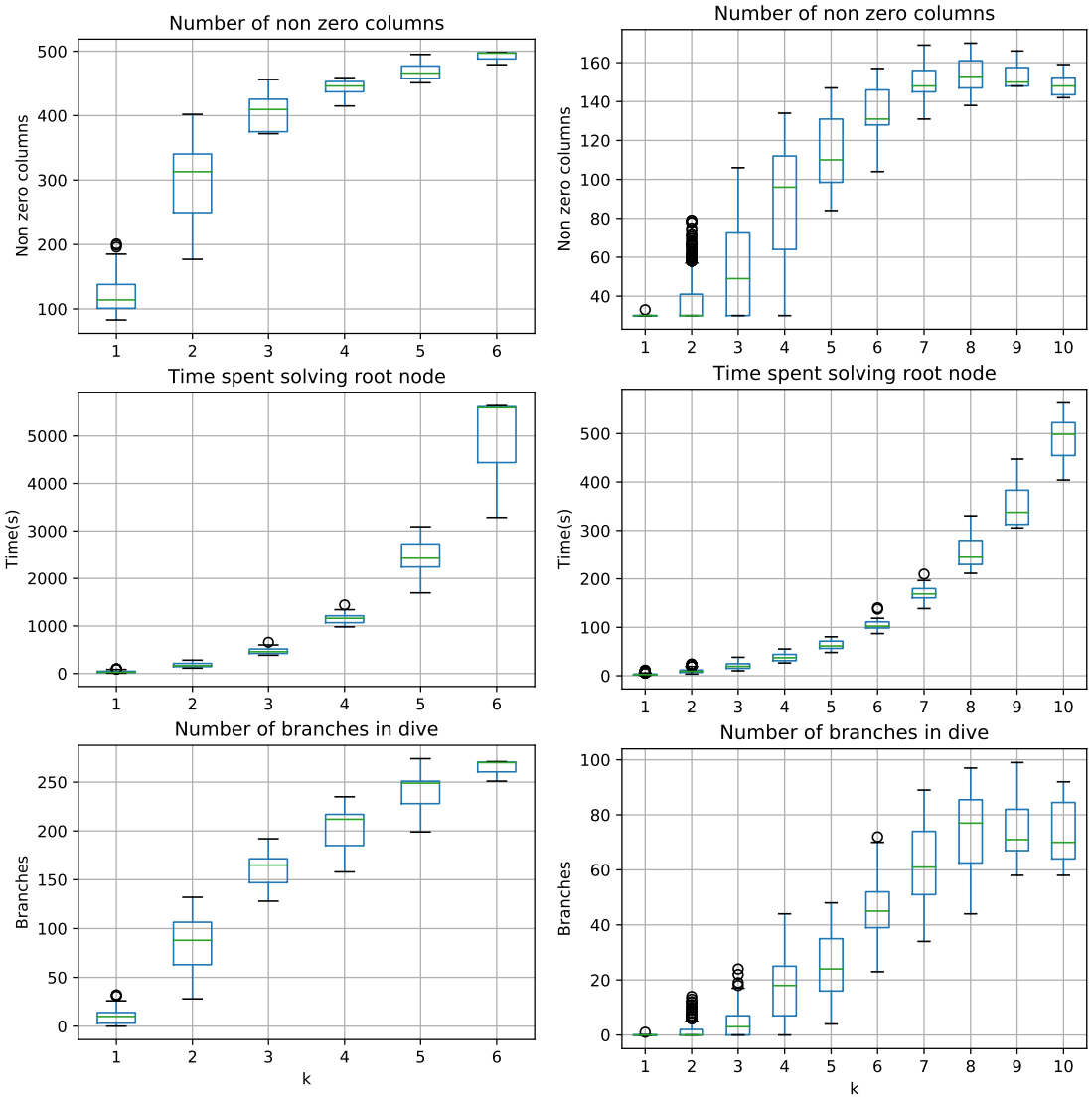
Figure 6.8: Comparison of the number of dives per run and number of calls to the column generator for a single employee per run with the following neighbourhoods: maximum shift changes per employee (MSCE, $\sum_d \delta_{ed} \leq k \ \forall \ e$), maximum on/off changes per employee (MOCE, $\sum_d \delta^{\text{on}}_{ed} \leq k \ \forall \ e$), maximum employee changes (MEC, $\sum_e \delta_e \leq k$), fixed days (FD, $\delta_d = 0 \ \forall \ d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$), fixed employees (FE, $\delta_e = 0 \ \forall \ e \notin \{j, \ldots, j+k\} \subseteq \mathcal{E}$), fixed days (on/off) (FDO, $\delta^{\text{on}}_d = 0 \ \forall \ d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$), and fixed employees (on/off) (FEO, $\delta^{\text{on}}_e = 0 \ \forall \ e \notin \{j, \ldots, j+k\} \subseteq \mathcal{E}$).



(a) Maternity wards problem

(b) Operating theatre ward problem

Figure 6.9: Comparison of various metrics for a single dive from an optimal root node solution with no neighbourhood constraints ("None") against average dive metrics from our variable neighbourhood descent algorithm with the following neighbourhoods: maximum shift changes per employee (MSCE, $\sum_d \delta_{ed} \leq k \; \forall \; e$), maximum on/off changes per employee (MOCE, $\sum_d \delta^{\text{on}}_{ed} \leq k \; \forall \; e$), maximum employee changes (MEC, $\sum_e \delta_e \leq k$), fixed days (FD, $\delta_d = 0 \; \forall \; d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$), fixed employees (FE, $\delta_e = 0 \; \forall \; e \notin \{j, \ldots, j+k\} \subseteq \mathcal{E}$), fixed days (on/off) (FDO, $\delta^{\text{on}}_d = 0 \; \forall \; d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$), and fixed employees (on/off) (FEO, $\delta^{\text{on}}_e = 0 \; \forall \; e \notin \{j, \ldots, j+k\} \subseteq \mathcal{E}$).



(a) Maternity wards problem

(b) Operating theatre ward problem

Figure 6.10: Comparison of the number of non zero columns in the solution to the root node, the average time spent solving the root node and the average number of branches per dive by neighbourhood distance parameter k. The dives are collated from all runs with the maximum shift changes per employee neighbourhood ($\sum_d \delta_{ed} \leq k \; \forall \; e$).

6.6 Chapter summary

This chapter has shown how we perform neighbourhood search on an integer roster solution using branch and price. In §6.1, we define 14 different neighbourhoods that can be used with

rostering problems. We then showed how we implemented seven of these neighbourhoods within our generic column generation model introduced in Chapter 4.

Through experimentation, we found using the maximum shift changes per employee ($\sum_d \delta_{ed} \leq k \ \forall \ e$) and fixed-days ($\delta_d = 0 \ \forall \ d \notin \{j, \ldots, j+k\} \subseteq \mathcal{D}$) were the two most effective neighbourhoods to use with branch and price out of those we tested. The effectiveness of each neighbourhood restriction algorithm was sensitive to the problem being solved and the incumbent roster solution's quality.

We found that every neighbourhood restrictions we tested had several significant effects on a branch-and-price dive. Each neighbourhood restriction led to solving the column generation subproblem fewer times, a less fractional root node solution, and fewer branches in a given dive. These factors lead to a given branch-and-price dive with neighbourhood restrictions taking significantly less time than without neighbourhood restrictions.

Chapter 7

Column generation subproblem heuristics

In Chapter 6, we presented various methods of performing neighbourhood search using branch and price to quickly improve an initial roster solution. One of the methods we used to construct a given initial roster solution is to perform a single branch-and-price dive with no heuristics.

However, finding an initial roster solution with a dive for the maternity wards problem takes around 16 hours. 92% of this time is spent solving the column generation subproblem. This is because of the vast number of entities we are constructing in the nested Shortest Path Problem with Resource Constraints (SPPRC). Our SPPRC suffers from the "curse of dimensionality" from having many state variables (resources), which is common for many types of dynamic programs (Powell, 2009). The recommendation by Powell (2009) in this case is to solve an approximate dynamic program in which the number of states are heuristically reduced by approximation of the value function or cost-to-go.

In this chapter, we present two core heuristic methods for solving the column generation subproblem much faster. The first method, which we call "entity restriction", involves reducing the number of entities that we choose to extend at each node in our nested SPPRC. Although we lose proof of optimality within our column generation subproblem, we can still find low reduced cost roster-lines. We introduce four entity restriction methods in §7.1.

The second method, which we call "LP neighbourhood pricing", involves putting neighbourhood restrictions on the column generation subproblem to find the most negative reduced cost column in the neighbourhood of the current LP solution. By adding neighbourhood restrictions to the column generation subproblem, we reduce the number possible entities generated significantly. We also found that by using LP neighbourhood pricing, our RMP produced more naturally integer solutions; see §7.4.4. We introduce three LP neighbourhood pricing methods in §7.2.

In the column generation literature for staff rostering problems, several groups have used entity restriction in various forms. Dohn and Mason (2013), Burke and Curtois (2014), Gérard et al.

(2016), and Barbosa et al. (2015) have heuristically kept only the lowest reduced cost entities. We discuss the implementation of this method in §7.1.1. This method essentially approximates the value function or cost-to-go as zero. Since certain resources such as the hours worked per fortnight are infeasible if not equal to an exact value, approximating a non-zero cost-to-go for these resources is challenging.

Zamorano and Stolletz (2017), Restrepo et al. (2016) and Strandmark et al. (2020) heuristically remove non-dominated entities based on the corresponding resource vectors. However, each of these research groups does this in a problem specific way. In §7.1.2-§7.1.4, we explore generic methods of removing non-dominated entities based on the corresponding resource vectors. We believe these methods to be novel within the staff rostering literature. However, the techniques we use are similar to those used in the field of multi-constrained optimal paths (Garroppo et al., 2010). Using the classification of Garroppo et al. (2010), precomputed buckets (§7.1.4) and systematic sampling (§7.1.3) are "interval partitioning" techniques, and entity restriction by variability (§7.1.2) is a "separation' technique.

Maenhout and Vanhoucke (2010a), Gomes et al. (2017) and Bard and Purnomo (2005a) have used neighbourhood swaps to enumerate columns and find columns with negative reduced costs. In §7.2, we discuss using a modified version of our nested SPPRC to find the most negative reduced cost column in a given neighbourhood instead of enumerating all possible swaps. No-one has used a neighbourhood restricted SPPRC with an LP incumbent solution within the staff rostering literature to find new negative reduced cost columns.

See §3.2.4 for more details on each of the staff rostering papers mentioned in this section.

7.1 Entity restriction

In the following sections, we report on heuristically removing a subset of the entities at each node in a nested SPPRC either because they have a high reduced cost or because there is not enough variability between the resource vectors.

7.1.1 Entity restriction by cost

The first entity restriction strategy, as discussed by Dohn and Mason (2013), involves keeping only the M lowest reduced cost entities. We call this strategy "entity restriction by cost". This strategy is a greedy heuristic in that we are trying to find the lowest reduced cost full roster-line and are, hence, keeping the lowest reduced cost component entities of that full roster-line. This strategy is shown in Algorithm 12 where Steps (10) and (11) are the only differences to our standard dominance algorithm (see Algorithm 1). Recall that this algorithm is used for entities of a particular type at a particular node in the SPPRC of a particular employee.

As our dominance algorithm already involves ordering the entities by their cost, this strategy involves little additional computation.

Algorithm 12 Modified dominance algorithm with entity restriction by cost

Require: A set of entities before dominance, $\mathcal{U}$.

```
P = ∅                                   ▷ Initialising set of non-dominated entities
Order set, U, by the cost of each entity in ascending order.
for each Entity τ^1 ∈ U do ▷ Both "for each"s maintain the ascending order of cost.
    isDominated = False
    for each Entity τ^2 ∈ P do
        if τ^2 ⪯ τ^1 then
            isDominated = True
    if isDominated = False then
        ADD τ^1 to P.
    if |P| > M then
        EXIT for loop
return non-dominated set of entities, P.
```

Figure 7.1 provides a contrived example of a set of non-dominated entities, which start and end on the same day with a single resource. In this example, we are keeping the five lowest-cost, non-dominated entities.

The single resource in Figure 7.1 has a "prefer lower" dominance rule, which means that we consider lower resource values as better. Thus, after dominance, we tend to observe a correlation between higher costs and lower resource values; if an entity has a high cost and a high resource value, it would have been dominated. For example, there is very strong evidence of a correlation between the cost of a work-stretch and the values of resources with the $C^{\emptyset}$ cost function for a single column generation iteration; see Appendix B. Therefore, the main drawback to entity restriction by cost is that we often do not consider entities with good resource values, which could be necessary for a low reduced cost full roster-line.

7.1.2 Entity restriction by variability in resource vectors

Instead of keeping entities with the lowest cost, we can also attempt to keep entities that have desirable resource vectors. Evaluating the quality of a resource vector is difficult without solving the SPPRC to completion, and thus, we instead find a set of dissimilar resource vectors.

Thus, our second strategy for reducing the number of entities at each node in our nested SPPRC is "entity restriction by variability". With this strategy, we only keep entities with dissimilar resource vectors at each node in the nested SPPRC.

We denote resource vector $\mathcal{T}_{\tau^1}$ as being similar to resource vector $\mathcal{T}_{\tau^2}$ with the notation, $\mathcal{T}_{\tau^1} \sim \mathcal{T}_{\tau^2}$. We deem two resource vectors similar if the weighted Euclidean distance between them is less than a given parameter d_τ, i.e.,

$$\mathcal{T}_{\tau^1} \sim \mathcal{T}_{\tau^2} \iff \sqrt{\sum_{\theta \in \Theta_\tau} \left(\frac{\mathcal{T}^\theta_{\tau^1} - \mathcal{T}^\theta_{\tau^2}}{\xi^\theta} \right)^2} \leq d_\tau \tag{7.1}$$

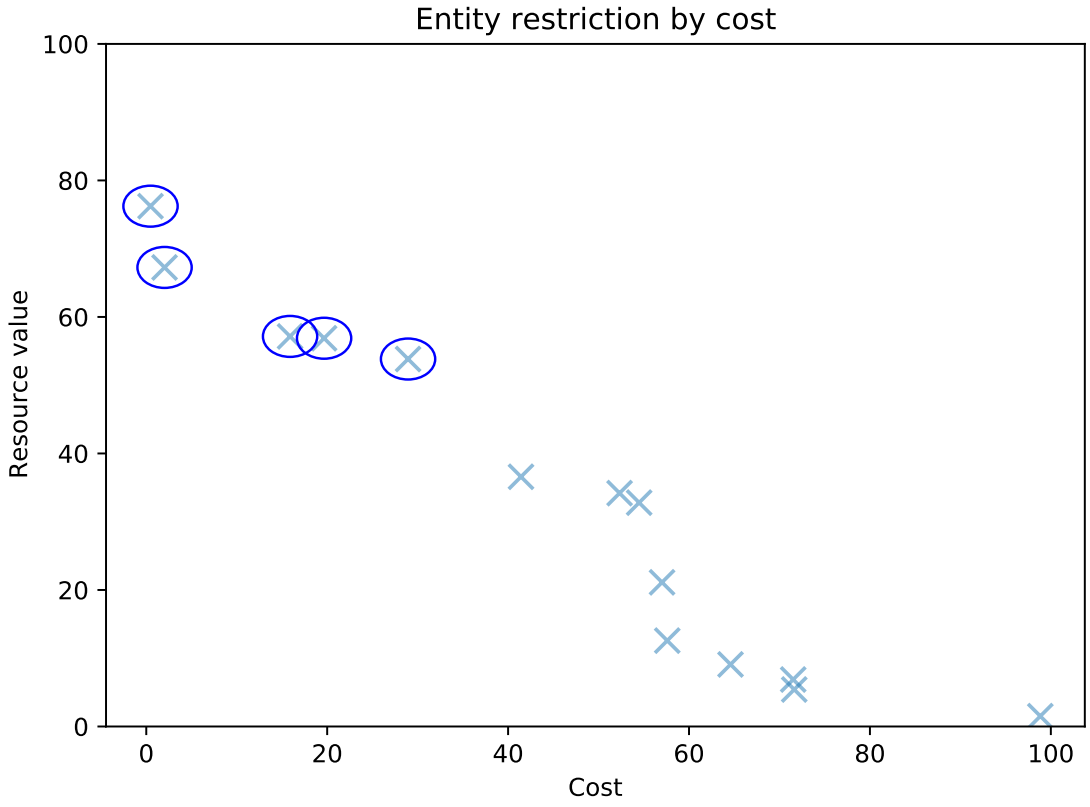


Figure 7.1: Example of selecting the five lowest cost entities, irrespective of the resource value. In this figure, the kept entities are circled.

where ξ^θ is an estimate of the relative importance per 1 unit difference in the values for resource θ as determined by the user. If resource θ has a cost α^θ, we choose $\xi^\theta = \alpha^\theta$ to be our resource weight for resource θ, otherwise, we choose $\xi^\theta = \alpha^{\theta^2}$ where resource θ affects the value of resource θ^2 and α^{θ^2} is the associated cost of resource θ^2. For example, the on-stretch resource θ = OnStretchWeekOneHours, which represents the number of hours worked in a given week, doesn't have a direct cost, but does contribute to the equivalent roster-line resource θ = RosterLineWeekOneHours which does have a cost. So in this case: $\xi^{\text{OnStretchWeekOneHours}} = \alpha^{\text{RosterLineWeekOneHours}}$

Entity restriction by variability in resource vectors involves only keeping entities if they are not similar to any other entity we have already kept in our non-dominated, dissimilar set. We demonstrate this strategy in Algorithm 13 where Step 9 involves an additional check for similarity to other entities in our non-dominated set, which is used in Step 11 to prune entities. Note that we have retained Step 13 from our entity restriction by cost algorithm in case our parameter d_τ is too low, and there are too many non-similar entities.

Figure 7.2 shows an example of us keeping the best entities by cost, which are of distance $d_\tau = 10$ away from the resource vectors of other entities which we are choosing to keep. The

Algorithm 13 Modified dominance algorithm with entity restriction by variability in resource vectors

Require: A set of entities, $\mathcal{U}$.

1: $\mathcal{P} = \emptyset$ ▷ Initialising set of non-dominated entities
2: Order set, $\mathcal{U}$, by the cost of each entity in ascending order.
3: **for each** Entity $\tau^1 \in \mathcal{U}$ **do** ▷ Both "for each"s maintain the ascending order of cost.
4: isDominated = False
5: isSimilar = False
6: **for each** Entity $\tau^2 \in \mathcal{P}$ **do**
7: **if** $\tau^2 \preceq \tau^1$ **then**
8: isDominated = True
9: **if** $\tau^2 \sim \tau^1$ **then**
10: isSimilar = True
11: **if** isDominated = False and isSimilar = False **then**
12: ADD τ^1 to $\mathcal{P}$.
13: **if** $|\mathcal{P}| > M$ **then**
14: EXIT for loop
15: **return** non-dominated set of entities, $\mathcal{P}$.

main disadvantage of this algorithm is in finding appropriate parameters. If d_τ is too large, we remove too many entities and reduce our likelihood of finding low reduced cost full roster-lines. If d_τ is too small, we will not achieve a high enough variability of resource vectors. The choice in the number of entities that will be kept is also sensitive to the resource weight vector, ξ^θ, that we have estimated.

7.1.3 Systematic sampling of entities based on cost

Our third strategy for reducing the number of entities at each node in our nested SPPRC is "systematic sampling of entities based on cost". Since we know that entities with higher costs often have worse resource values, we can find a restricted set of entities with an even distribution of costs and hope this will lead to a good balance of the cost and the quality of the resource vectors.

This algorithm involves the systematic sampling of the full set of entities $\mathcal{U}$ before performing dominance checks. Algorithm 14 shows us reducing the number of unprocessed entities by ordering the entities by cost in Step 2 and only keeping every $(|\mathcal{U}|/M)$th entity in Steps 3-5, where $|\mathcal{U}|$ is the total number of unprocessed entities and M is the number of entities we want to keep. Then dominance is calculated as normal.

Figure 7.3 shows an example of us selecting an even spread of entities with respect to the entity cost. The main drawback to this algorithm is that we choose to keep entities with a high cost that are unlikely to be part of the low reduced cost full roster-lines.

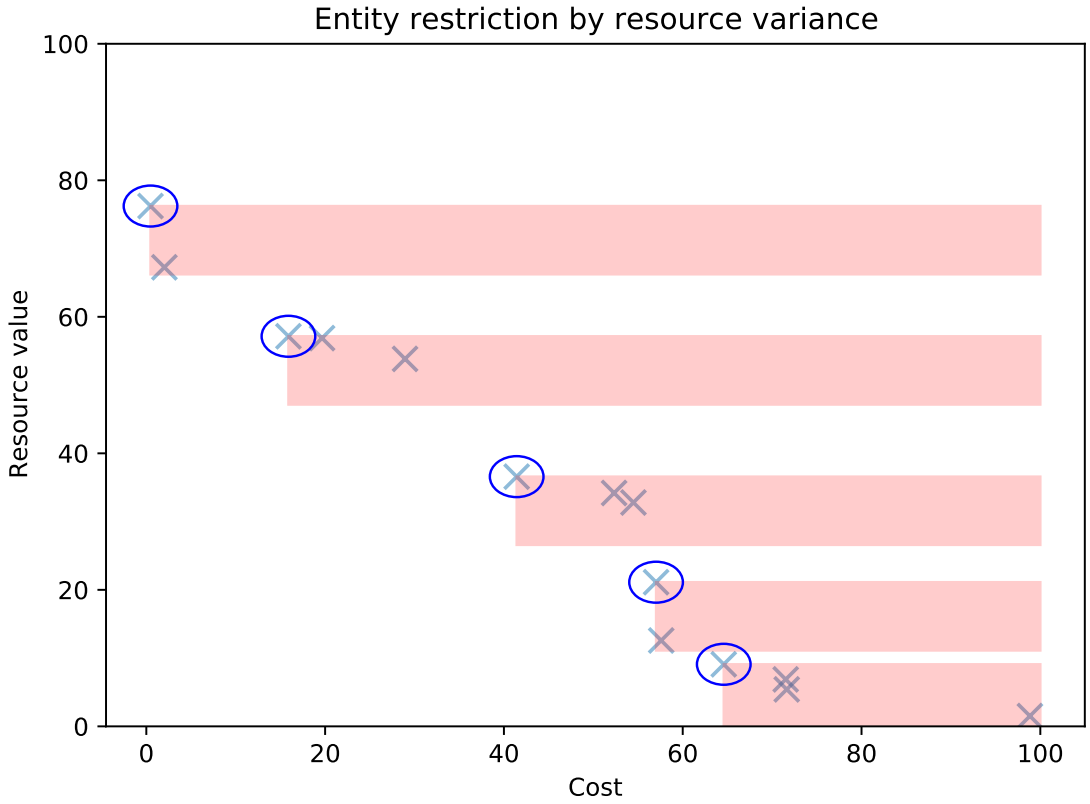


Figure 7.2: Example of selecting the lowest cost entities with a minimum distance of 10 between the resource value of all other entities we are keeping. In this example, there is a single resource with weight $\xi^\theta = 1$ and maximum distance $d_\tau = 10$. In this figure, the kept entities are circled. We show the minimum distance with a red bar and prune any entities within the red bars as they are too similar to another entity we are already keeping. Note that because we apply this algorithm to entities in order of increasing cost and increasing resource quality, the bar only needs to be below the entity we are selecting.

Algorithm 14 Modified dominance algorithm with systematic sampling

Require: A set of entities, $\mathcal{U}$.

$\mathcal{P} = \emptyset$ ▷ Initialising set of non-dominated entities
Order set, $\mathcal{U}$, by the cost of each entity in ascending order.
for $i \leftarrow 1$ to $|\mathcal{U}|$ **do**
    if $(i-1)\%(|\mathcal{U}|/M) \neq 0$ **then** ▷ Where we define $a\%b$ as the remainder from dividing a by b. In this step we are only keeping every $(|\mathcal{U}|/M)$th entity.
        REMOVE $[\mathcal{U}]_k$
for each Entity $\tau^1 \in \mathcal{U}$ **do** ▷ Both "for each"s maintain the ascending order of cost.
    isDominated = False
    for each Entity $\tau^2 \in \mathcal{P}$ **do**
        if $\tau^2 \preceq \tau^1$ **then**
            isDominated = True
    if isDominated = False **then**
        ADD τ^1 to $\mathcal{P}$.
return non-dominated set of entities, $\mathcal{P}$.

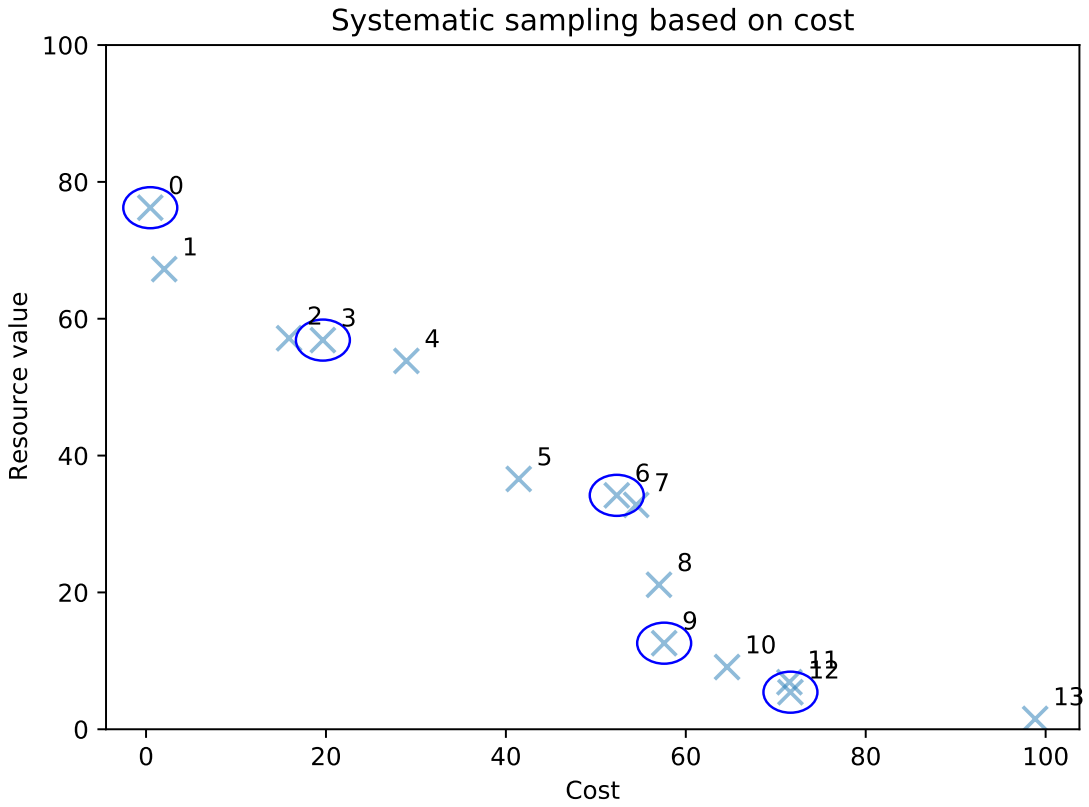


Figure 7.3: Example of keeping every third entity ordered by cost. In this figure, the kept entities are circled.

7.1.4 Precomputed buckets based on resource vectors

Our fourth strategy for reducing the number of entities at each node in our nested SPPRC is "precomputed buckets based on resource vectors". This strategy involves clustering the entities with similar resource vectors and only keeping a certain number of non-dominated entities from each cluster by cost.

Sadykov et al. (2020) clustered resource vectors into several "buckets". These buckets were chosen by equally dividing up the time and capacity resources for the column generation subproblem of a vehicle routing problem. Sadykov et al. (2020) used the buckets to speed up dominance calculations and, in some cases, heuristically reduce the number of entities in the SPPRC.

With more than ten resources, this process becomes more complicated. If we divide each resource's possible values into two ranges, high and low, then for 15 resources, there are 32768 different buckets.

We wanted to precompute a set of k buckets, which appropriately divide up the resource space.

To achieve this, we called the column generator with all duals set to zero to generate a full roster-line for each employee. We then collated the resource vectors of every on-stretch, work-stretch and partial roster-line constructed.

We performed k-means clustering on the resource vectors for each set of entities to divide the entities into k clusters. To efficiently calculate which cluster each entity belongs to, we built a binary classification tree for on-stretches, work-stretches and roster-lines.

Our approach to creating the buckets is naive. We are producing the lowest cost roster-lines for each employee, which may not represent the best roster-lines that we expect to find in an optimal solution. However, our experiments show that this method still has reasonable performance even with our naive approach to creating the buckets.

When solving each node of our SPPRC, we first divide our set of unprocessed entities into bins using the binary classification tree. We then perform Algorithm 12 on each bin and combine the set of non-dominated, low-cost entities from each bin.

Figure 7.4 shows an example of us selecting the lowest cost entity in each of five buckets. In our real algorithm, we select more than one entity per bucket.

7.2 LP neighbourhood pricing

This section focuses on finding the most negative reduced cost roster-line in the neighbourhood of an incumbent LP solution to the Restricted Master Problem (RMP). We deem this method "LP neighbourhood pricing." Specifically, we are enforcing a neighbourhood within each of the SPPRCs of our nested subproblems. Enforcing a neighbourhood in an SPPRC allows us to prune any entity outside our neighbourhood rule. By pruning entities, we can speed up the time taken to generate columns considerably.

To demonstrate the effectiveness of LP neighbourhood pricing, we have chosen to use the maximum shift changes per employee neighbourhood $\sum_d \delta_{ed} \leq k \ \forall \ e$, introduced in §6.1. This

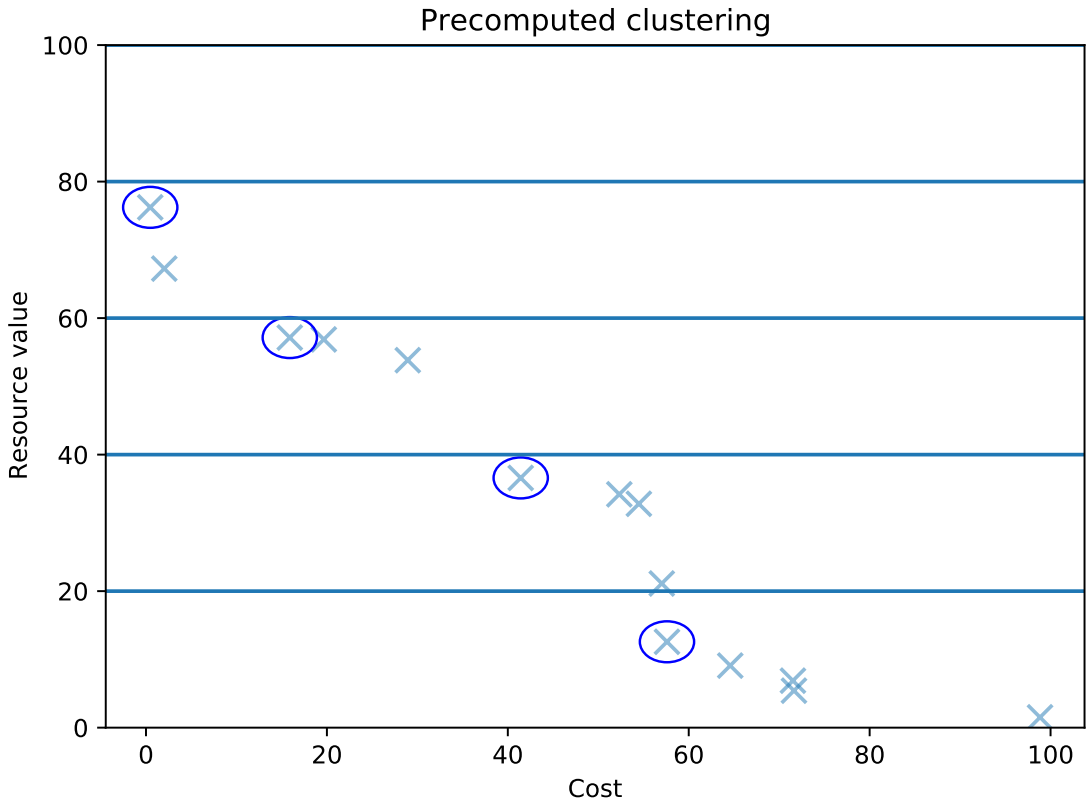


Figure 7.4: Example of keeping lowest cost entity from 5 different clusters of the resource value. In this figure, the kept entities are circled. The lines represent each bucket for a single resource value.

neighbourhood quickly improves low-quality solutions, especially with the smaller neighbourhood sizes ($k \leq 3$). However, this neighbourhood can only be used with an integer incumbent roster solution, and so is a form of IP neighbourhood pricing. Because an LP solution is not necessarily integer, we modified this neighbourhood in three different ways so that it can apply to an LP solution (which can be integer or non-integer). We describe each of these three resulting LP neighbourhoods in §7.2.1.

One of the benefits of LP neighbourhood pricing is that we produce many integer roster solutions while solving the root node; see §7.4.4 for more details. This means we can rapidly produce some good roster solutions without having to branch or fully solve the root node.

Another benefit of enforcing a neighbourhood in the column generation subproblem is that it is less likely to be affected by dual instability. Generated columns must be similar to those already part of the best known primal solution. Thus, the generated columns are more likely to improve the primal solution than unrestricted columns generated using oscillating duals.

Neighbourhood pricing is a particular form of partial pricing. We never do a full price with LP neighbourhood pricing but update our incumbent to partial price in a new neighbourhood.

Partial pricing has the advantage of finding a good entering column very quickly. We see the same benefit here by applying our neighbourhood.

7.2.1 LP neighbourhoods

Unlike an integer incumbent, where there is one roster-line worked per employee, an incumbent LP solution to the RMP can have a single employee work a convex combination of multiple roster-lines. Thus, in this section, we consider how we can generate a candidate roster-line in the neighbourhood of an LP incumbent solution.

Similar to §6.1, we represent a candidate roster-line in terms of a set of binary, employee-shift variables, x_{eS}, i.e., $x_{eS} = 1$ if employee e works shift S in their associated roster-line and 0 otherwise.

Unlike §6.1, we define the incumbent LP solution $\hat{X}$ to the RMP in terms of a set of incumbent roster-lines $\hat{X}_e^\kappa$ for each employee e. Each incumbent roster-line is a roster-line with a non-zero value in the incumbent LP solution. We can represent each incumbent roster-line $\hat{X}_e^\kappa$ as a set of binary employee-shift variables, i.e., $\hat{X}_e^\kappa = (\hat{x}_{eS}^\kappa,\ \forall S \in \mathcal{S})$, where $\hat{x}_{eS}^\kappa$ is a binary variable representing whether employee e works shift S in incumbent roster-line κ. We can also represent the incumbent LP solution $\hat{X}$ using the same employee-shift variables, i.e.,

$$\hat{X} = (\hat{x}_{eS}^\kappa,\ e = 1, 2, \ldots, |\mathcal{E}|, \kappa = 1, 2, \ldots, K, \forall S \in \mathcal{S}).$$

Consider all roster-lines $r \in \mathcal{R}_e$ within an incumbent LP solution generated for an employee e (see §4.1.1 for more details on the RMP). We order these roster-lines by decreasing value in the LP solution λ_e^r and index this ordering by $\kappa = 1, 2, \ldots$, i.e., the incumbent roster-lines $\hat{X}_e^\kappa$ have decreasing value in the LP solution as κ increases.

As we tend to LP optimality, we often produce LP solutions with many incumbent roster-lines. For this reason, we only use the first K incumbent roster-lines for our LP neighbourhood, i.e., we consider $\hat{X}_e^\kappa\ \forall \kappa \in (1, \ldots, K)$. Considering fewer incumbent roster-lines lowers the complexity of the neighbourhood search.

From these definitions, we can define three new modification variables that we can use to define neighbourhoods to a given incumbent LP solution. We define a *modification* $\delta_{ed\kappa}$ to an LP incumbent as zero if employee e is assigned the same shift on day d in both the generated column and the κth incumbent column, and one otherwise, i.e.,

$$\delta_{ed\kappa} = \begin{cases} 0, & \text{if } x_{eS} = \hat{x}_{eS}^\kappa \quad \forall S \in \mathcal{S}^d \\ 1, & \text{otherwise} \end{cases} \tag{7.2}$$

The second and third modifications to an LP incumbent involve generating a single roster-line, x_{eS}, in the neighbourhood of the first K incumbent roster-lines. We define the *average modification* $\delta_{ed}^{\text{Average}}$ to an LP incumbent as the sum of the modifications for each incumbent roster-line divided by the number of incumbent roster-lines, i.e.,

$$\delta_{ed}^{\text{Average}} = \frac{\sum_{\kappa \in 1,\ldots,K} \delta_{ed\kappa}}{K} \tag{7.3}$$

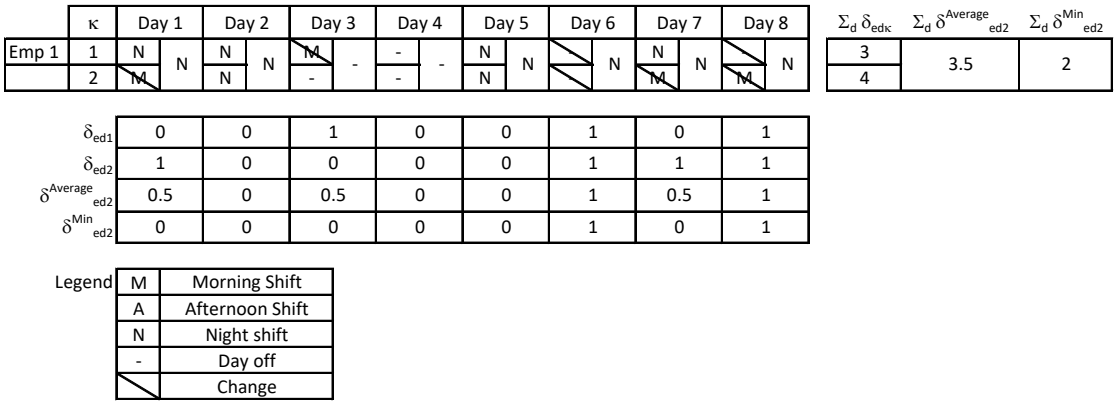


Figure 7.5: Example of each of our new day modification variables for the top $K = 2$ roster-lines by value in an incumbent LP solution (shown on the left column of each day) compared to a generated roster-line (shown on the right column of each day).

Initially, our average modification was weighted towards the value of each incumbent roster-line κ in the RMP. However, in our initial experiments, we found this approach to not work as well in practice. There are more possible values for each of the resources which track the sum of the weighted average modifications in our column generation subproblem. This increases the number of possible entities that the column generation subproblem can generate significantly. For more details on this resource, see §7.2.3.

Lastly, we define the *minimum modification* δ^{Min}_{ed} to an LP incumbent as zero if employee e is assigned the same shift on day d in the generated column as occurs in any of the first K incumbent roster-lines, and one otherwise, i.e.,

$$\delta^{\text{Min}}_{ed} = \min_{\kappa \in 1,\ldots,K} \delta_{ed\kappa} \tag{7.4}$$

An example of how this notation can describe the modifications, from an LP incumbent solution to a generated roster-line, is shown in Figure 7.5.

From these three new modification variables, we can define three LP neighbourhoods which are all enforced in the column generation subproblem:

1. Maximum shift changes per employee by column, $\min_\kappa(\sum_d \delta_{ed\kappa}) \leq k \; \forall \; e$
2. Maximum average modifications per employee, $\sum_d \delta^{\text{Average}}_{ed} \leq k \; \forall \; e$
3. Maximum minimum modifications per employee, $\sum_d \delta^{\text{Min}}_{ed} \leq k \; \forall \; e$

Please note that we give formulas for each neighbourhood in shorthand. $\sum_d$ indicates $\sum_{d \in \mathcal{D}}$, $\forall e$ indicates $\forall e \in \mathcal{E}$ and $\min_\kappa$ indicates $\min_{\kappa \in K}$.

In practice, for each of our neighbourhoods, we only considered the first $K = 2$ incumbent roster-lines to reduce the number of entities we generate. A larger number of incumbent roster-lines can increase the number of entities we generate significantly; see §7.2.6 for more details.

7.2.2 LP variable neighbourhood descent

To solve each node in the branch-and-bound tree using LP neighbourhood pricing, we adapted the "pipe variable neighbourhood descent" procedure described by Hansen et al. (2016) for use with LP neighbourhood pricing. Our implementation of this procedure is shown in Algorithm 15. We use a different implementation of LP neighbourhood pricing for each of our LP neighbourhoods discussed in §7.2.1. However, we are solving a modified instance of an SPPRC with an additional resource in each implementation.

Algorithm 15 Pipe variable neighbourhood descent to solve a node in the branch-and-bound tree

```
1:  procedure VND
2:      k ← 1
3:      while k ≤ k_max = 3 do
4:          improvement = false
5:          for e ∈ ℰ do
6:              RUN LP neighbourhood pricing for employee e within the neighbourhood
    of that employee's incumbent columns X̂_e^κ ∀κ ∈ (1, ..., K) to generate negative
    reduced cost columns and with neighbourhood size k
7:              if negative reduced cost columns were generated then
8:                  ADD negative reduced cost columns to RMP
9:                  SOLVE restricted RMP
10:                 X̂ ← solution to RMP
11:                 improvement = true
12:         if improvement = false then
13:             k ← k + 1
14:     return X̂
```

We solve the column generation subproblem using LP neighbourhood pricing with a given neighbourhood distance parameter k until we can find no further negative reduced cost columns. The neighbourhood is then increased by one. The maximum neighbourhood distance $k_{\max}$ we use for LP neighbourhood pricing is three. We found that a larger value than three for the neighbourhood distance significantly increased the column generation subproblem solve time.

We need to find an initial set of columns on which to perform LP neighbourhood pricing. To do this, we use the construction heuristic described in §6.4, i.e., we generate the lowest cost, legal roster-line for each employee with no neighbourhood restrictions and combine them to form a (low quality) starting solution.

In the following sections, we discuss how we model LP neighbourhood pricing for our three LP neighbourhoods within our column generation framework introduced in Chapter 4. We first discuss the maximum average modifications per employee neighbourhood ($\sum_d \delta_{ed}^{\text{Average}} \leq k \; \forall \; e$) and the maximum minimum modifications per employee neighbourhood ($\sum_d \delta_{ed}^{\text{Min}} \leq k \; \forall \; e$) as they only require solving a single nested SPPRC per employee. Afterwards, we discuss the

maximum shift changes per employee by column neighbourhood ($\min_\kappa(\sum_d \delta_{ed\kappa}) \leq k \ \forall \ e$) in which we need to solve more than one nested SPPRC per employee.

7.2.3 Maximum average modifications per employee

The first LP neighbourhood pricing method uses the maximum average modifications per employee neighbourhood ($\sum_d \delta_{ed}^{\text{Average}} \leq k \ \forall \ e$).

All three of our LP neighbourhood pricing techniques involve adding resources to calculate incumbent difference to each entity, similar to the maximum shift changes per employee neighbourhood described in §6.2.1. However, we use different values for the incumbent difference resources for shifts and off-stretches to model each neighbourhood. For the $K = 2$ incumbent roster-lines $\kappa \in (1, 2)$ worked by employee e, we define the incumbent difference $\mathcal{T}_S^{\text{Incumbent difference}}$ for a given shift, S, as follows:

$$\mathcal{T}_S^{\text{Incumbent difference}} = 1 - \frac{\hat{x}_{eS}^1 + \hat{x}_{eS}^2}{2}$$

The off-stretch incumbent difference $\mathcal{T}_F^{\text{Incumbent difference}}$ is defined by:

$$\mathcal{T}_F^{\text{Incumbent difference}} = \sum_{S \in \mathcal{S}^F} \frac{\hat{x}_{eS}^1 + \hat{x}_{eS}^2}{2}$$

where $\mathcal{S}^F$ is the set of shifts which overlap with off-stretch F.

The on-stretch, work-stretch and roster-line incumbent difference resources, $\mathcal{T}_O^{\text{Incumbent difference}}$, $\mathcal{T}_W^{\text{Incumbent difference}}$ and $\mathcal{T}_R^{\text{Incumbent difference}}$, are modelled the same as in §6.2.1, i.e., with the following resource extension functions:

$$\mathcal{E}_{\text{O}}^{\text{Incumbent difference}}(\mathcal{T}_O, \mathcal{T}_S) = \mathcal{T}_O^{\text{Incumbent difference}} + \mathcal{T}_S^{\text{Incumbent difference}}$$
$$\mathcal{E}_{\text{W}}^{\text{Incumbent difference}}(\mathcal{T}_O, \mathcal{T}_F) = \mathcal{T}_O^{\text{Incumbent difference}} + \mathcal{T}_F^{\text{Incumbent difference}}$$
$$\mathcal{E}_{\text{R}}^{\text{Incumbent difference}}(\mathcal{T}_R, \mathcal{T}_W) = \mathcal{T}_R^{\text{Incumbent difference}} + \mathcal{T}_W^{\text{Incumbent difference}}$$

and a hard upper bound equal to the neighbourhood distance, $\gamma_O^{\text{Incumbent difference}} = \gamma_W^{\text{Incumbent difference}} = \gamma_R^{\text{Incumbent difference}} = k$.

Note that all five of these incumbent resources can potentially be non-integer.

7.2.4 Maximum minimum modifications per employee

The second LP neighbourhood pricing method uses the maximum minimum modifications per employee neighbourhood ($\sum_d \delta_{ed}^{\text{Min}} \leq k \ \forall \ e$).

In practice, for the $K = 2$ incumbent roster-lines $\kappa \in (1, 2)$ worked by employee e, we define the incumbent difference resource variable for shift S as follows:

$$\mathcal{T}_S^{\text{Incumbent difference}} = \min(1 - \hat{x}_{eS}^1, 1 - \hat{x}_{eS}^2)$$

Similarly, we define the off-stretch incumbent difference resource variable as follows:

$$\mathcal{T}_F^{\text{Incumbent difference}} = \sum_{S \in \mathcal{S}^F} \min(\hat{x}_{eS}^1, \hat{x}_{eS}^2)$$

where $\mathcal{S}^F$ is the set of shifts which overlap with off-stretch F.

The on-stretch, work-stretch and roster-line incumbent difference resources, $\mathcal{T}_O^{\text{Incumbent difference}}$, $\mathcal{T}_W^{\text{Incumbent difference}}$ and $\mathcal{T}_R^{\text{Incumbent difference}}$, are modelled the same as in §6.2.1 and §7.2.3.

7.2.5 Maximum shift changes per employee by column

The third LP neighbourhood pricing method uses the maximum shift changes per employee by column neighbourhood ($\min_\kappa(\sum_d \delta_{ed\kappa}) \leq k \; \forall \; e$).

In practice, we solve a separate nested SPPRC within the neighbourhood of each of employee e's incumbent roster-lines $\kappa \in (1, 2)$. We enforce this neighbourhood in each of the two nested SPPRCs identically to how we enforced the maximum shift changes per employee neighbourhood but separately for each incumbent roster-line (§6.2.1).

From solving these two nested SPPRCs, we find two separate roster-lines for each employee $e \in \mathcal{E}$. One roster-line with the most negative reduced cost within the neighbourhood of incumbent roster-line $\kappa = 1$, i.e., $\sum_d \delta_{ed1} \leq k$, and one roster-line with the most negative reduced cost within the neighbourhood of incumbent roster-line $\kappa = 2$, i.e., $\sum_d \delta_{ed2} \leq k$. Out of these two roster-lines, the one with the most negative reduced cost is the roster-line with the most negative reduced cost within the maximum shift changes per employee by column neighbourhood ($\min_\kappa(\sum_d \delta_{ed\kappa}) \leq k \; \forall \; e$). However, in practice, we also add the other column to our RMP as it is often also a low reduced cost column.

7.2.6 Comparison of LP neighbourhood pricing techniques

As mentioned previously, the incumbent difference resources for the maximum average modifications per employee neighbourhood ($\sum_d \delta_{ed}^{\text{Average}} \leq k \; \forall \; e$) can take fractional values. Thus, the total number of possible values for each incumbent difference resource is equal to $K \times k + 1$, e.g. if $k = 2$ and $K = 2$ then $\mathcal{T}_\tau^{\text{Incumbent difference}} \in \{0, 0.5, 1, 1.5, 2\}$. The higher number of possible values significantly increases the number of non-dominated entities as an entity with a lower value for the incumbent difference resource can not be dominated by an entity with a higher value for that resource. In contrast, the other two neighbourhoods can only have integer incumbent differences.

The maximum minimum modifications per employee neighbourhood ($\sum_d \delta_{ed}^{\text{Min}} \leq k \; \forall \; e$) allows for much fewer entities to be removed at each node than the other two neighbourhood strategies. This is because the value of the minimum modification is less than the value of the average modification and the modification per column by definition, i.e., $\delta_{ed}^{\text{Min}} \leq \delta_{ed\kappa}$ and $\delta_{ed}^{\text{Min}} \leq \delta_{ed}^{\text{Average}}$. In each neighbourhood, we only remove entities if the sum of the modifications over each day d for the given employee e is above the neighbourhood distance parameter k. Thus, a smaller value for each modification variables means we remove fewer entities at each node in our nested SPPRC.

Lastly, only the maximum shift changes per employee by column neighbourhood ($\min_\kappa(\sum_d \delta_{ed\kappa}) \leq k \; \forall \; e$) requires solving the column generation subproblem more than once.

7.3 Testing methodology

We have thus far discussed seven different column generation subproblem heuristics: four entity restriction methods and three LP neighbourhood pricing methods. To compare each of these column generation subproblem heuristics, we want to understand how many entities are generated in each nested SPPRC within our column generation subproblem. We also want to understand how they reduce the time it takes to solve the nested SPPRC.

Further, we wish to compare each column generation subproblem heuristic's effectiveness within the broader context of finding high-quality integer roster solutions. To find a high-quality integer roster solution, we used a separate branch-and-price dive with each column generation subproblem heuristic. Please refer to Section 4.5 for more details on diving.

To demonstrate the behaviour of the different column generation subproblem heuristics on solving rostering problems with different levels of difficulty, we solve both the (easier) operating theatre ward problem and the (more challenging) maternity wards problem. Similar to our experiments with variable neighbourhood descent in §6.4, we used random shift cost perturbations to create ten instances of each problem to ensure more accurate results by mitigating the variance in the branch-and-bound tree. Unlike variable neighbourhood descent, each instance is a single dive, so it is more affected by variance in the branch-and-bound tree.

Thus, with two problems and ten instances of each problem, we performed 20 runs with each column generation subproblem heuristic. As we compare seven different column generation subproblem heuristics, we needed to perform 140 runs in total.

We also recorded the average number of entities produced in each solve of the nested SPPRC and the average time taken to solve the nested SPPRC using each column generation subproblem heuristic.

We deem a column generation subproblem heuristic more effective if using that heuristic with a branch-and-price dive can produce integer roster solutions with a lower cost than using another heuristic in the same amount of time. Thus, we also recorded all integer roster solutions produced and their quality to measure the branch-and-price dive's effectiveness with each column generation subproblem heuristic.

Further, we are interested in determining if the column generation subproblem heuristics strongly change the nature of the solves, specifically the "natural integrality" (i.e., how close the solution to the root node is to integrality). Thus, we also recorded the objective of the root node solution for each run and two measures of natural integrality for the root node: the number of non-zero columns in the root node solution and the number of branches required for each dive.

Lastly, for our LP neighbourhood pricing algorithms, we recorded all incumbent LP solutions to the RMP produced while solving the root node. We compare these solutions with how the LP incumbent solution to the RMP is changing over time. This is to gain insight into why our LP neighbourhood pricing algorithm consistently produces multiple high-quality integer roster solutions while solving the root node. No high-quality integer roster solutions were produced while solving the root node when using entity restriction or solving the column generation subproblem optimally.

7.4 Results

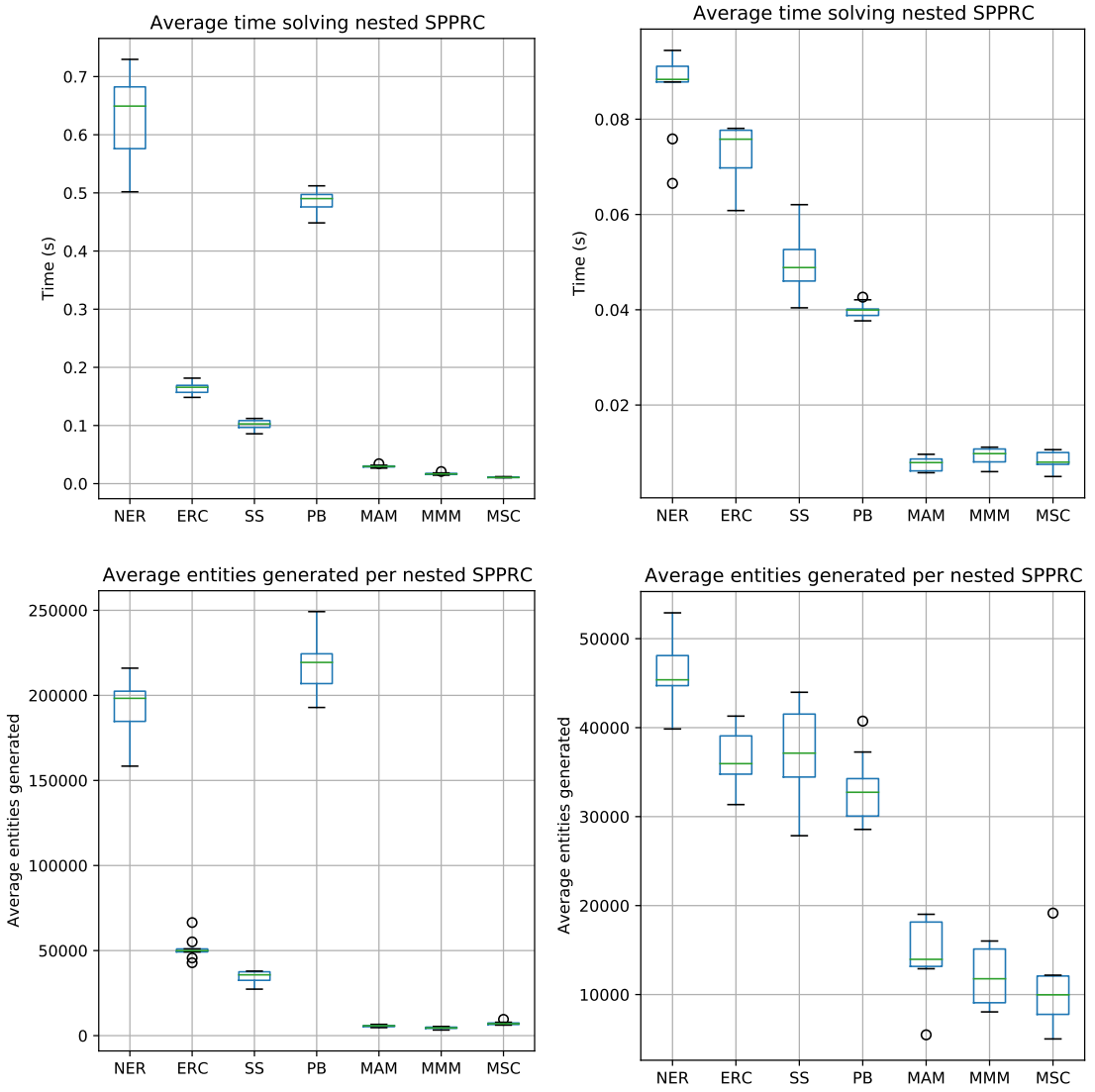
The following sections detail the results of the 120 runs with entity restriction by cost, systematic sampling, precomputed buckets, and our three LP neighbourhood pricing methods. We do not report on the 20 runs with entity restriction by variability in resource vectors as, despite spending significant efforts in testing, no configuration for this method produced reasonable quality integer roster solutions. We also provide results for 20 runs with no column generation subproblem heuristic.

7.4.1 Column generation speed

In this section, we discuss the effect of column generation subproblem heuristics on solving our nested SPPRC. While solving each dive with column generation subproblem heuristics, we recorded the average time spent solving each nested SPPRC and the average number of entities generated (including on-stretches, work-stretches and roster-lines). The average time to solve the SPPRC and the average number of entities generated for each run is shown with box and whisker plots in Figure 7.6.

Entity restriction by cost, systematic sampling and the three LP neighbourhood pricing algorithms all lead to a significant reduction in the time taken to solve the nested SPPRC and the number of entities generated compared with no column generation subproblem heuristic. This difference is exacerbated in the (more challenging) maternity wards problem as we impose the same upper limit on the number of entities extended M at each node in our nested SPPRC (or same neighbourhood distance parameters $1 \leq k \leq 3$) for each column generation subproblem heuristic across the two problems. However, there is a much larger number of entities created with no column generation subproblem heuristic in the maternity wards problem. Out of these five column generation subproblem heuristic techniques, the three LP neighbourhood pricing techniques solve our nested SPPRC much faster than entity restriction by cost and systematic sampling.

Although we solve the column generation subproblem faster using precomputed buckets than not using a heuristic, we still generated more entities in the maternity wards problem. This is because we only run dominance for each individual bin and not after combining the bins. Since a large amount of time is spent calculating dominance, the time taken to solve the nested SPPRC with precomputed buckets is still shorter than solving the column generation subproblem optimally, even though more entities are created.



(a) Maternity wards problem

(b) Operating theatre ward problem

Figure 7.6: Average time spent generating entities and average number of entities generated with the following column generation subproblem heuristics: no entity restriction (NER), entity restriction by cost (ERC), systematic sampling (SS), precomputed buckets (PB), maximum average modifications per employee (MAM), maximum minimum modifications per employee (MMM), and maximum shift changes per employee by column (MSC).

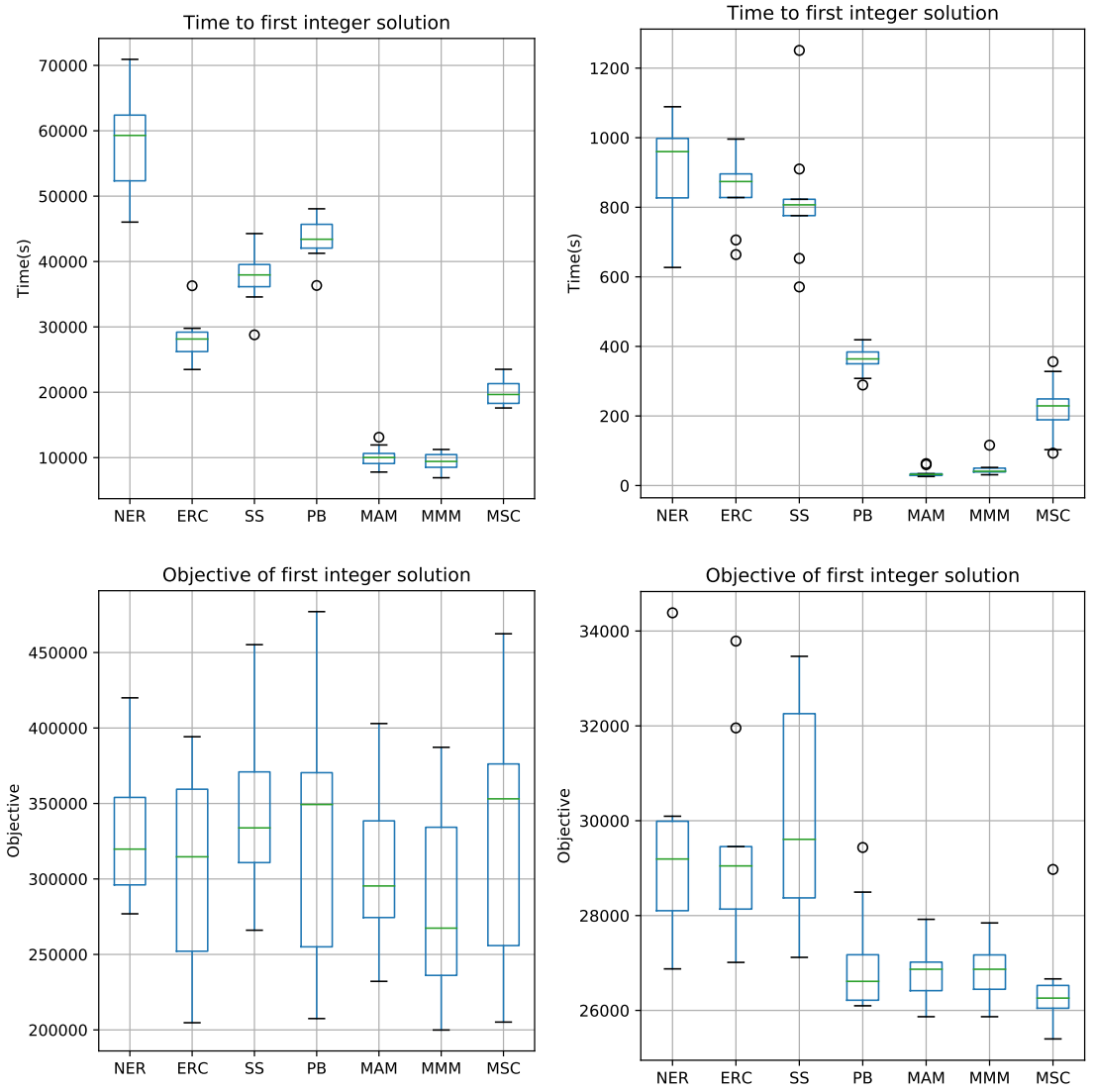
7.4.2 All run results

In this section, we discuss performing a branch-and-price dive until an integer roster solution is found for all 140 runs described in §7.3.

We summarise the time taken to find the first integer solution by diving and the quality of that solution for all 140 runs with box and whisker plots in Figure 7.7.

The results show that finding integer solutions with LP neighbourhood pricing is significantly faster than cost and resource restriction. Furthermore, for the operating theatre ward problem, the integer roster solutions produced by diving with LP neighbourhood pricing are of a considerably lower cost than with entity restriction by cost, systematic sampling and with no column generation subproblem heuristic and have a similar cost to precomputed buckets. This suggests the most effective method of neighbourhood restriction when performing a dive is LP neighbourhood pricing.

Out of our cost and resource restriction algorithms, restriction by cost appears to perform the best for the Waikato DHB problems, and precomputed buckets appears to perform the best for the operating theatre ward problem. Although precomputed buckets doesn't perform as well on the Maternity wards problem, it is the most complex entity restriction method and has a lot of potential for further optimisation. For example, we could have used entities generated using non-zero duals for the clustering algorithm or used a different clustering algorithm. Thus, we may have happened across a more effective set of buckets for the operating theatre ward problem by chance.



(a) Maternity wards problem

(b) Operating theatre ward problem

Figure 7.7: Comparison of the time and objective of the first integer solution produced by diving with the following column generation subproblem heuristics: no entity restriction (NER), entity restriction by cost (ERC), systematic sampling (SS), precomputed buckets (PB), maximum average modifications per employee (MAM), maximum minimum modifications per employee (MMM), and maximum shift changes per employee by column (MSC).

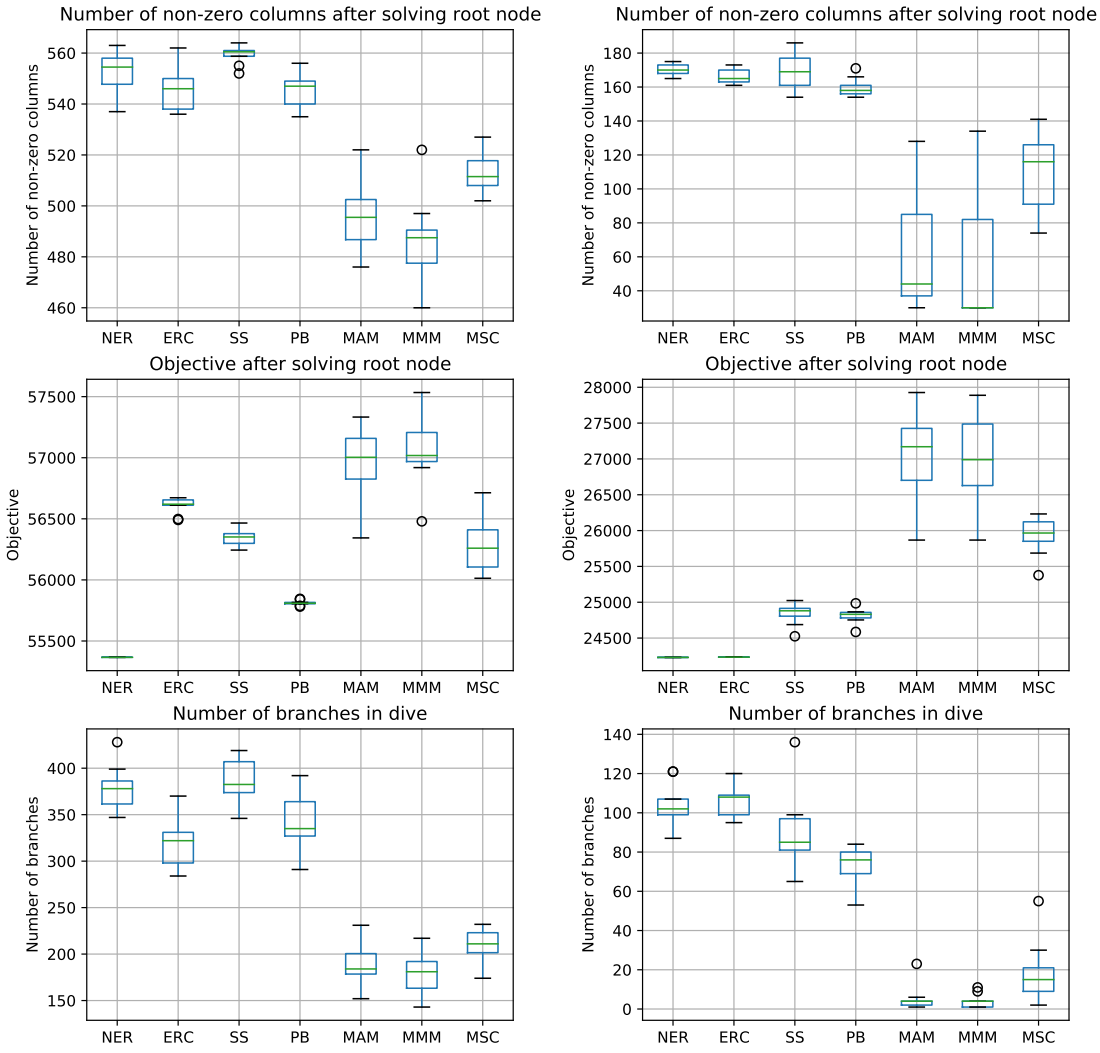
7.4.3 Properties of dives

In this section, we show metrics for all 140 runs with regards to the objective of the solution to the root node and how far the solution to the root node is from integrality. We also measure the

total number of branches in the branch-and-bound dive. We show these metrics for all 140 runs with a box and whisker plot in Figure 7.8

Our entity restriction strategies had a similar number of non-zero columns in the solution to the root node and took a similar number of branches to dive to an integer solution as when solving the column generation subproblem with no heuristics. However, each run with LP neighbourhood pricing had significantly fewer non-zero columns in the solution to the root node and took fewer branches to dive to an integer solution than the other strategies. Further, for many of the runs which involved solving the operating theatre problem with LP neighbourhood pricing, the solution to the root node was naturally integer (as shown by zero branches being required).

We observed that although the cost and resource restriction methods produced higher quality solutions to the root node than the LP neighbourhood pricing methods, this did not correlate with higher quality integer solutions after diving. The LP neighbourhood pricing methods mostly had a higher objective value solution to the root node and an equal or lower objective value integer solution produced with a branch-and-price dive; cf. §7.4.2. These observations indicate that LP neighbourhood pricing methods produce root node solutions that are closer to being integer. The natural integrality of LP neighbourhood pricing methods is discussed next.



(a) Maternity wards problem

(b) Operating theatre ward problem

Figure 7.8: Comparison of root node metrics and the number of branches in the dive with the following column generation subproblem heuristics: no entity restriction (NER), entity restriction by cost (ERC), systematic sampling (SS), precomputed buckets (PB), maximum average modifications per employee (MAM), maximum minimum modifications per employee (MMM), and maximum shift changes per employee by column (MSC).

7.4.4 Natural integrality properties of LP neighbourhood pricing

This section reports on one of the other core benefits of using LP neighbourhood pricing: LP neighbourhood pricing often produces integer solutions from non-integer solutions without branching, which never happened with the entity restriction methods or with no column generation subproblem heuristic. Since we often still converge to an IP solution with no IP restriction,

there is no point doing further column generation iterations to improve the root node solution if these improvements are undone in the diving process.

To demonstrate this phenomenon, we provide detailed metrics showing the progression of solving the root node for a single run with maximum minimum modifications per employee LP neighbourhood pricing. As in §7.4.3, we measure the integrality of the root node solution with the number of incumbent roster-lines. If the number of incumbent roster-lines is equal to the number of employees, the solution is integer.

We compare each LP solution's integrality level with the total number of changes from one incumbent LP solution to the next incumbent LP solution. We calculate the total number of changes between two incumbent LP solutions,

$$\hat{X}^{\text{old}} = (\hat{x}_{eS}^{\kappa,\text{old}},\ e = 1, 2, \ldots, |\mathcal{E}|, \kappa = 1, 2, \ldots, K, \forall S \in \mathcal{S})$$

and

$$\hat{X}^{\text{new}} = (\hat{x}_{eS}^{\kappa,\text{new}},\ e = 1, 2, \ldots, |\mathcal{E}|, \kappa = 1, 2, \ldots, K, \forall S \in \mathcal{S}),$$

with the following equation:

$$\sum_{e \in \mathcal{E}} \sum_{S \in \mathcal{S}} \text{abs} \left(\sum_{\kappa \in 1,\ldots,K} (\hat{x}_{eS}^{\kappa,\text{old}} - \hat{x}_{eS}^{\kappa,\text{new}}) \right) \tag{7.5}$$

where $\hat{x}_{eS}^{\kappa,\text{old}}$ and $\hat{x}_{eS}^{\kappa,\text{new}}$ are binary variables representing whether employee e works shift S in incumbent roster-line κ. This metric is often referred to as the Hamming distance.

The other two metrics include the incumbent LP solution's objective and the current neighbourhood distance parameter k.

We show how each of these four metrics changes over time with three line graphs in Figure 7.9. The dashed line shown in the first graph indicates the total number of employees in the problem. As shown by the dashed blue line and the solid blue line intersecting, multiple integer roster solutions are found while solving the root node LP. This occurs in other examples of LP neighbourhood pricing even when the root node is not naturally integer. Further, we can see that the LP solutions often go from non-integer to integer while solving the root node. We observe that the jumps from non-integer to integer solutions usually correlate with a large change to the incumbent LP solution. However, we can't find a strong link between changes to the objective or neighbourhood distance and the jump to an integer solution.

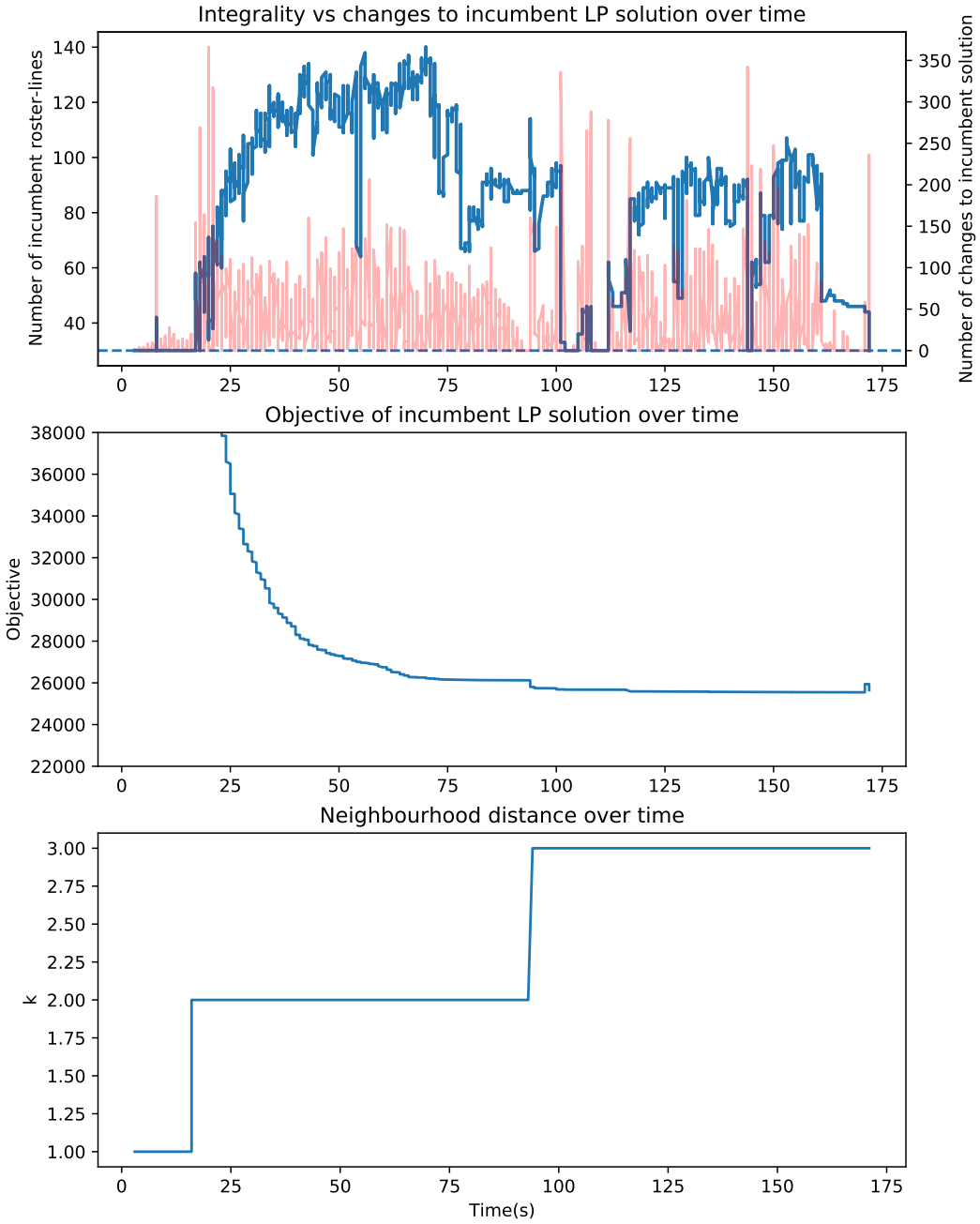


Figure 7.9: Solving the root node of a single operating theatre ward problem with the maximum minimum modifications per employee neighbourhood. The first graph shows the number of incumbent roster-lines in blue, with the incumbent change over time shown in red. In this graph, a dotted blue line indicates the total number of employees. If there is one incumbent roster-line per employee, i.e. the solid blue line and the dotted blue line intersect, then the LP incumbent solution is integer. The other two graphs show the LP incumbent solution's objective over time and the neighbourhood distance (k, respectively.

7.5 Real-world rostering experience

Using the algorithms shown in Chapters 6 and 7, we made four rosters for the operating theatre ward, which the rostering staff at Waikato DHB implemented in practice with minimal changes. Unfortunately, although they verified our solution to an old Maternity wards problem, due to changing priorities caused by COVID-19, they were unable to implement rosters for the Maternity wards before submission of this thesis.

Our process for real-world rostering involved creating a template form for the rostering staff to fill out in Excel and using Python to process the Excel template into our column generation model's input data. The data they provided consistently had multiple mistakes as this was data they usually provided to human rosterers. Human rosterers can easily interpret which data was incorrect using their expertise, but our algorithm could not.

To find mistakes in the input data, we first constructed a decent roster solution using a branch-and-price dive with LP neighbourhood pricing using the maximum minimum modifications neighbourhood (§7.2.3). We then performed 10 minutes of neighbourhood search on that roster solution using the maximum shift changes per employee neighbourhood (§6.2.1). Using these two matheuristics would usually lead to a roster solution that would be good enough to spot the roster model's mistakes. This strategy allowed us to test and fix the input data rapidly.

Once we had validated that the inputs to our model were correct, we attempted to find the highest quality roster solution possible. Because we had access to a server with many CPU cores, we ran all of the column generation based neighbourhood search strategies from Chapter 6 in parallel overnight to improve the incumbent found when validating inputs and attempt to find the best roster solution possible.

The rostering staff would send the template to us by email, and we would use the process outlined in this section to produce a solution. If the solution was almost perfect, then the rostering staff would accept it. Otherwise, the rostering staff would provide feedback about how we could improve the model by adding in more constraints or by modifying the relative importance of the constraints that we already have.

The whole process was mostly manual and tedious, and it would be ideal if we could automate some of this process with a high-quality user interface and a web server to solve the rostering problems.

7.6 Chapter summary

This chapter has shown how we use heuristics within the column generation subproblem to speed up branch and price. We have compared two different types of column generation subproblem heuristics. The first is entity restriction, in which we remove a subset of the entities at each node in our SPPRC. Entity restriction is a standard method of speeding up the column generation subproblem in the literature.

The second type of column generation subproblem heuristic is LP neighbourhood pricing. Although we have seen researchers enumerate neighbourhood swaps to generate columns from

an LP incumbent, we did not find anyone using a dynamic program to explore a neighbourhood to an LP incumbent solution to generate columns.

All of our column generation subproblem heuristics were useful at speeding up the time taken to perform a branch-and-price dive. However, we found that the LP neighbourhood pricing heuristics could be used to find a solution of the same or better quality in less time. Further, the LP neighbourhood pricing heuristics were much more likely to produce a naturally integer solution to the root node and produce integer roster solutions while solving the root node.

Thus, we would recommend implementing LP neighbourhood pricing if the goal is to find high-quality roster solutions quickly within a column generation framework.

Chapter 8

Conclusions

In this chapter, we summarise the significance of this work, provide ideas for future work on this topic, and finally provide insights about how to solve difficult staff rostering problems.

8.1 Achievements

The primary objective of this research has been to improve upon Genie++ to build a state-of-the-art generic system for modelling and solving challenging staff rostering problems automatically. This work has improved upon standard column generation methods for solving staff rostering problems found within the literature. Further, we have compared and implemented a large selection of novel column generation based matheuristics. By doing so, we believe that we have achieved our primary objective and built a state-of-the-art generic staff rostering solver.

Our secondary objective has been to solve two challenging sets of staff rostering problems: the International Nurse Rostering Competition (INRC) problems and the problems given to us by the Waikato DHB. We used our staff rostering solver to solve these two challenging sets of staff rostering problems successfully.

The first set of staff rostering problems was from the INRC. Within a 4-hour time limit, we solved 30 of the most difficult INRC problems to proven optimality. In doing so, we produced 11 new lower bounds and two new upper bounds (best-known roster solutions) that had not been found before. As many researchers have tested their rostering algorithms on these problems, this was a significant achievement. Not only did we show the value of the improvements we made to Genie++, but we also confirmed the value of a column generation decomposition for solving staff rostering problems. Furthermore, most column generation based algorithms use a single dive to find a high-quality solution but do not prove optimality for that solution. Being able to prove optimality to a large number of problems with a column generation based algorithm shows the effectiveness of the techniques we have employed.

The second set of staff rostering problems was from the Waikato DHB. These problems were complex as they involved non-linear cost functions that incorporated complex fairness rules.

Few researchers in the literature have been modelling complex fairness rules. Even with the novel improvements to standard column generation that we used to solve the INRC problems, our solver took a long time to generate roster solutions; further, these solutions were initially low-quality.

After developing and implementing new matheuristics, including neighbourhood pricing, we were able to quickly generate high-quality solutions to these problems through a set of column generation-based matheuristics used to construct an initial roster solution and perform local search. The rosters we generated were used in practice at the Waikato hospital for four consecutive months. Their usage of our rosters is a testament to the solution's quality and the model's accuracy; they did not accept our solution until we had implemented their required complex cost functions and fairness features into our model.

The fact that we were able to build rosters for two very different classes of problems, the INRC and Waikato DHB problems, using our nested column generation framework, is a testament to the strength of our modelling framework.

8.2 Contributions

By building our state-of-the-art roster solver, we believe that we have contributed to the literature for solving staff rostering problems with column generation in several ways.

Firstly, having found no existing comprehensive, generic column generation model for the multi-objective nurse rostering problem, we introduced a novel way of modelling a generic nested column generation subproblem (§4.6). We also believe that our model could be modified to include modelling tasks that build into shifts. Our model works seamlessly with all the techniques we have introduced, including dominance cost functions, complex branching rules, relaxations and matheuristics. The framework we presented and math notation was adequate to describe all these additional features.

Secondly, we found that there had not been extensive research in resource dominance or branching for solving staff rostering problems with column generation. Thus, we developed a novel dominance cost function (§5.2.1) and novel branching strategies (§5.3) to identify provably optimal solutions for all 30 of the hardest INRC problems within a reasonable amount of time. The dominance cost function was pivotal in solving the INRC problems to provable optimality within a four-hour time limit. With respect to the dominance cost functions, instead of requiring each resource to have a better value in one resource vector than another, our approach exploits the fact that resources eventually contribute to the objective function of the roster-line. Thus, dominance cost functions allow cost trade-offs to be made early. Using dominance cost functions, we could solve the column generation subproblem around 20x faster than with standard methods. This approach was a significant improvement to our overall solver as the column generation subproblem accounted for most of the time spent solving rostering problems.

The novel branching strategies were pivotal to increasing the upper bound. Our branching strategies allowed us to effectively branch over multiple employees at once or over key employees

and break the rostering problem's underlying symmetry. Without their use, we only had an increase in the upper bound of a single INRC problem. By using these strategies, in one case, we proved optimality for the entire branch-and-bound tree with less than 30 nodes for a problem that was previously unsolved to proven optimality.

In addition to novel dominance and branching techniques, we also developed a novel arbitrary shift preference technique (§5.2.2) and a shift aggregation technique (§5.4) which were helpful in decreasing the solve time.

Thirdly, we found that no one has extensively compared neighbourhood search strategies within branch-and-price frameworks. In Chapter 6, we defined 14 different neighbourhoods and applied seven of these neighbourhoods to our generic rostering system using a variable neighbourhood descent (VND) algorithm. Out of these seven neighbourhoods, we believe "maximum on/off changes per employee", "maximum employee changes", "fixed days", "fixed days (on/off)," and "fixed employees (on/off)" to be novel neighbourhoods used in column generation matheuristics within the field of staff rostering. Although "fixed employees" neighbourhood and "maximum shift changes per employee" neighbourhood are not novel column generation matheuristics within the field of staff rostering, our VND which utilises each of our neighbourhoods is novel. We then demonstrated how our novel applications of "maximum roster-line modifications" neighbourhood and "fixed-days neighbourhood" were the two most effective neighbourhood restricted column generation techniques of the seven that we tried.

Fourthly, we found that no one has extensively compared column generation subproblem heuristics for staff rostering. In Chapter 7, we defined two different types of column generation subproblem heuristics, "entity restriction" and "LP neighbourhood pricing". Out of the four entity restriction heuristics, we believe "entity restriction by variability in resource vectors", "systematic sampling of entities based on cost", and "precomputed buckets based on resource vectors" to be novel. We also consider LP neighbourhood pricing in general to be novel. The three novel LP neighbourhoods we used with LP neighbourhood pricing include "maximum average modifications per employee", "maximum minimum modifications per employee", and "maximum shift changes per employee by column". All of these column generation subproblem heuristics meant the column generation subproblem could be solved significantly faster. Using any column generation subproblem heuristics with a branch-and-price dive allowed us to find integer roster solutions more quickly than solving a branch-and-price dive without column generation subproblem heuristics.

We demonstrated that using our novel LP neighbourhood pricing methods to find negative reduced cost columns can be more effective than generating columns with entity restriction. Using LP neighbourhood pricing with a branch-and-price dive finds roster solutions of the same or higher quality in less time than the same branch-and-price dive using entity restriction or no column generation subproblem heuristics. Further, LP neighbourhood pricing often found good-quality integer roster solutions while solving the root node to our Waikato DHB problems. It also consistently found naturally integer solutions to the initial LP root node of our operating theatre ward problem.

Our LP neighbourhood pricing and IP neighbourhood restrictions both lead to smaller branch-and-price dives. The success of these algorithms over a branch-and-price dive with no restrictions is evidence of the concept that solving an LP all the way down and branching all the way up is less effective than imposing a neighbourhood and avoiding a descent and climb. We can even consider imposing a neighbourhood as branching on some distance from a solution before solving the LP. By using LP neighbourhood pricing with a branch-and-price dive, we found IP roster solutions of the same or better quality 7-20x faster than with a standard branch-and-price dive. We found that by performing column generation restricted local search, we can find up to 3x better quality solutions to the Maternity wards problem than can be found with a standard branch-and-price dive in the same time.

We found no other work in the literature that systematically explored and compared this many column generation based matheuristics to solve staff rostering problems. Most papers we found implemented at most two matheuristics.

8.3 Future work

We believe that the techniques outlined in this work are already very effective at solving staff rostering problems. Thus, the most important piece of future work is expanding our research to cover more classes of staff rostering problems and applying our techniques to other fields.

A prevalent model for solving staff rostering problems is to break down shifts into individual tasks as described in §3.2.1. Thus, our next step would be to model task entities and solve modified SPPRCs to build shift entities from task entities within our column generation subproblem.

We also found that breaking down a roster-line into on-stretches, off-stretches, and work-stretches is somewhat arbitrary. We believe that breaking down a roster-line into a set of weeks and fortnights would be better for the Waikato DHB problems as more of their rules were modelled by week and fortnight than by on-stretch, off-stretch and work-stretch. We would like to experiment on how best to break down a roster-line into component entities for maximum efficiency in the future.

We would also like to see the techniques described in this work applied to airline crew scheduling problems. As described in §3.1, airline crew scheduling is a similar class of problem to staff rostering. Hence we believe that our techniques, particularly dominance cost functions, would be similarly effective in solving these problems.

Our work has explored a large variety of techniques, many of which have the potential to be investigated further and improved. Our dominance cost functions (§5.2.1) are very effective if they are constructed well. However, in our current implementation, they need to be constructed by hand. In future, we would like to see an automatic method for constructing effective dominance cost functions.

We found that random, arbitrary shift preferences (§5.2.2) help a lot when there are multiple optimal solutions to the same rostering problem. Their use leads to a less fractional root node solution and fewer branches required to find an integer roster solution. We also found they were

an effective means of creating 'fake' new problem instances with sufficient variability, which we first elaborate on in §7.3. On experimentation with different random seeds to create these shift preferences, we found that the fractionality of the root node solution can change a lot depending on the arbitrary shift preferences; see §7.4 for details. In future, we would like to look into creating these arbitrary shift preferences systematically instead of randomly to see if we can lead the branch-and-price dive towards quickly finding an integer roster solution with fewer branches.

In some of the INRC problems, branching on resources over groups of employees (§5.3.1) was the most effective method we tested at driving up the lower bound. On other INRC problems, branching on high priority nurses (head nurses) was more effective (§5.3.3). We want to investigate why different problems respond so differently to our two different branching rules in future work. Furthermore, we want to investigate which resources are most effective to branch on and which nurses are of the highest priority to branch on.

The experiments we performed in Chapters 6 and 7 could be extended to consider modelling and solving more rostering problems to verify the effectiveness of our techniques further. Although we utilised arbitrary shift preferences to simulate having different problems, which did change the nature of the solves, we were not testing on truly different problems.

With each variable neighbourhood descent (VND) algorithm in Chapter 6, we changed the neighbourhood distance but did not change the neighbourhood type. In future, we would like to explore the effectiveness of combining multiple neighbourhood types into a single VND. As different neighbourhoods were more effective for solving different difficulties of problems and improving solutions to the same problem but with different objective values, combining multiple neighbourhoods might work better to solve a wide variety of problems. It is also possible to combine multiple neighbourhoods into a composite neighbourhood type as described in §6.1. We have yet to test composite neighbourhood types.

In Chapter 7, we solved several branch-and-price dives, each with a different column generation subproblem heuristic. By only solving a heuristic column generation subproblem, we could produce high-quality roster solutions with a dive faster than by solving the column generation subproblem to optimality. We also discovered the natural integrality properties of LP neighbourhood search. However, many researchers in the literature solve each node in the branch-and-price tree in two stages. In the first stage, they use a heuristic to solve the column generation subproblem, and in the second stage, they solve the column generation subproblems optimally. We have yet to test this two-stage method with our column generation subproblem heuristics.

We also wish to investigate the capabilities and advantages of LP neighbourhood pricing further. We still do not have a firm understanding of why LP neighbourhood pricing consistently generates so many integer solutions from the neighbourhood of non-integer solutions. We also wish to experiment with different neighbourhoods for LP neighbourhood pricing. For example, we could implement the fixed days neighbourhood from §6.3.1 with an LP incumbent solution to perform neighbourhood pricing.

Several researchers in the literature have generated negative reduced cost columns by enumerating shift swaps (see §3.2.4). We are interested in comparing using an SPPRC to perform

LP neighbourhood pricing with the enumeration of shift swaps to evaluate the same neighbourhood. We are also interested in soft neighbourhood restrictions in which we model the maximum neighbourhood distance with a cost. Soft neighbourhood restrictions may work well with our dominance cost functions.

Although we did not formally report in detail on this aspect, our compile-time customised entity model for modelling the feasibility and cost of a roster-line could also evaluate the cost and feasibility of a roster-line very quickly. Thus, our single model is easy to use with both column generation and with local search by enumeration. It is worth exploring how we can exploit this to improve our state-of-the-art generic system for solving rostering problems.

We found that, for many of the INRC problems, the gap between the objective of the LP solution and the IP solution found using a single branch-and-price dive was zero when using column generation. However, the gap was very large for the maternity wards problem. We want to investigate the primary cause of a large gap when solving staff rostering problems with column generation.

8.4 Recommendations

This section provides the reader with our insights into how they might solve their specific staff rostering problem.

Firstly, one must decide on whether to use column generation at all. We found that certain constraints lend themselves to column generation very well (instead of a MIP formulated without column generation). For example, any constraints modelled purely within on-stretches or work-stretches, such as shift precedence constraints or the ratio between days on and days off, do not significantly increase the time taken to solve the rostering problem with column generation. However, constraints that apply over the whole roster-line, such as a maximum number of nights, require a separate resource in each of three entity types: on-stretches, work-stretches, and roster-lines, and this can significantly increase the time taken to solve the column generation subproblem.

Next, one must decide on how to program the resources in the SPPRC. We used generic programming (i.e. compile-time optimised code) to build our SPPRC. Generic programming is significantly faster than having the resources as run-time inputs to the column generator. When modelling multiple classes of problems, only a single, small C++ file needs to be modified. In staff rostering, the column generation subproblem can vary significantly between different hospitals in the resources required. This variance contrasts with problems like VRPs, where there is less variety in resource types. Thus, if modelling many classes of problems, we would recommend generic programming. However, when modelling a single class of problems, debugging is made easier by directly programming the resources into the column generation subproblem.

When modelling the problem, our recommendation is to model as realistically as possible. In our experience, more accurate modelling trumps better quality solutions to a less accurate model. Our client could quickly tell if we were not correctly modelling any of the constraints they

described. However, it was difficult to tell the difference between a high-quality and an optimal roster solution once we had built the model correctly. Having non-linear constraints was also critical to modelling the problem correctly. Non-linear constraints are much easier to implement when using an SPPRC to model the column generation subproblem.

Lastly, one must decide which techniques to apply.

One technique we suggest that should always be applied is arbitrary shift preferences. This technique is an effortless way to improve the natural integrality of each node in the branch-and-price tree with little additional computation.

Another technique we suggest that should always be applied is dominance cost functions. This technique gave us a significant improvement in the time taken to solve the column generation subproblem, so we would highly recommend using these for dominance of at least some SPPRC resources.

With these two techniques applied, a standard branch-and-price dive should provide a high-quality roster solution in a reasonable amount of time unless the rostering problem is particularly challenging. However, if the problem is complex, we would recommend performing the branch-and-price dive using LP neighbourhood pricing with the "maximum minimum modifications per employee" neighbourhood to produce an incumbent solution. Then, we would recommend using branch-and-price neighbourhood search with a combination of "maximum roster-line modifications" neighbourhood and "fixed-days neighbourhood".

If the goal is to increase the calculated lower bound, we recommend using aggregate resource branching and priority first shift branching. A clever relaxation can also be helpful, such as using a relaxation that removes part of the symmetry. For example, if the shift types are near symmetrical such as in INRC, we recommend aggregating the shift types. If the employees are near-symmetrical, we recommend modelling the employees anonymously. However, employees can never be completely symmetrical due to roster history, and shifts can never be completely symmetrical; otherwise, they would not be separate shifts. Thus, there must be a process for dis-aggregating them as appropriate.

8.5 Final words

To my reader, I hope this thesis had been helpful for you in solving difficult staff rostering problems.

People are complex, and there is a potential in staff rostering problems to model the many complex facets of work-life, including fairness, health, preferences, work-life balance, and complex regulations. The high ceiling for problem complexity means there is a need for a complex modelling language and a fast solver. We hope that others may improve upon this work to capture the complexities inherent in effective human-centred rostering.

Personally, the journey has been long and, at times, frustrating. However, I am proud to have been given the opportunity to contribute to the international body of knowledge on staff rostering and column generation.

I look forward to hearing from anyone who has used my work to improve people's lives through rostering.

Appendices

Appendix A

Cost dominance function example enumeration proof

```
# For a number of weekends, 'n', find all combinations of
# weekends worked and weekends off with no restrictions
def get_all_rosterline_weekends(n):
    possibilities = [[]]
    for i in range(n):
        new_possibilities = []
        for P in possibilities:
            new_possibilities.append(P + ['-'])
            new_possibilities.append(P + ['w'])
        possibilities = new_possibilities
    return possibilities

# For a number of weekends, 'n', find all combinations of
# weekends worked and weekends off such that a weekend
# worked cannot follow a weekend off
def get_all_workstretch_weekends(n):
    possibilities = [[]]
    for i in range(n):
        new_possibilities = []
        for P in possibilities:
            if len(P) == 0 or P[-1] == "w":
                new_possibilities.append(P + ['w'])
            new_possibilities.append(P + ['-'])
```

```
            possibilities = new_possibilities
        return possibilities

# Get the maximum number of consecutive weekends in a roster-line
def get_mcw(R):
    mcw = 0
    ecw = 0

    for X in R:
        if X == "w":
            ecw += 1
        else:
            ecw = 0
        mcw = max(ecw, mcw)

    return mcw

# Find the number of worked weekends in a work-stretch
def get_ww(W):
    return W.count("w")

# Find the number of weekends off in a work-stretch
def get_wo(W):
    return W.count("-")

# Find the cost of a given maximum number of consecutive weekends
def get_cost(mcw, gamma):
    return max([mcw - gamma, 0])

# Calculate work-stretch dominance for MCW
def calculate_dominance(W1, W2, R1, R2, gamma):
    return get_cost(get_mcw(R1 + W1 + R2), gamma) \
           - get_cost(get_mcw(R1 + W2 + R2), gamma)

# Calculate the work-stretch psi function for MCW
def calculate_bound(W1, W2, gamma):
    if get_wo(W1) == 0 and get_wo(W2) > 0:
        return 4 - gamma
    else:
        return max(get_ww(W1) - get_ww(W2), 0)
```

```
total = 0
gamma_combinations = [0, 1, 2, 3]

for w_start_day in range(0, 4):
    for w_length in range(1, 5 - w_start_day):
        R1_combinations = get_all_rosterline_weekends(w_start_day)
        W1_combinations = get_all_workstretch_weekends(w_length)
        W2_combinations = get_all_workstretch_weekends(w_length)
        R2_combinations = get_all_rosterline_weekends(4 - w_start_day - w_length)
        for gamma in gamma_combinations:
            for W1 in W1_combinations:
                for W2 in W2_combinations:
                    for R1 in R1_combinations:
                        for R2 in R2_combinations:
                            total += 1
                            assert (calculate_dominance(W1, W2, R1, R2, gamma)
                                    <= calculate_bound(W1, W2, gamma))

print(total)
```

Appendix B

Regression on work-stretch resources

Dep. Variable:	cost	**R-squared:**	0.254
Model:	OLS	**Adj. R-squared:**	0.253
Method:	Least Squares	**F-statistic:**	176.0
Date:	Tue, 04 Aug 2020	**Prob (F-statistic):**	0.00
Time:	10:55:05	**Log-Likelihood:**	-44257.
No. Observations:	6205	**AIC:**	8.854e+04
Df Residuals:	6192	**BIC:**	8.863e+04
Df Model:	12		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
Intercept	788.7305	15.763	50.038	0.000	757.830	819.631
wF1Hours	-5.585e+13	2.54e+13	-2.202	0.028	-1.06e+14	-6.12e+12
wF2Hours	-3.937e+12	1.79e+12	-2.202	0.028	-7.44e+12	-4.32e+11
ww1Hours	5.585e+13	2.54e+13	2.202	0.028	6.12e+12	1.06e+14
ww2Hours	5.585e+13	2.54e+13	2.202	0.028	6.12e+12	1.06e+14
ww3Hours	3.937e+12	1.79e+12	2.202	0.028	4.32e+11	7.44e+12
ww4Hours	3.937e+12	1.79e+12	2.202	0.028	4.32e+11	7.44e+12
ww1Consec48hOff	-290.2046	14.728	-19.704	0.000	-319.077	-261.332
ww2Consec48hOff	-264.1684	12.001	-22.012	0.000	-287.695	-240.642
ww4Consec48hOff	-37.6502	21.852	-1.723	0.085	-80.488	5.187
wwholeWeekendsOff	-40.9434	9.853	-4.155	0.000	-60.259	-21.628
wtotalN	43.0155	4.904	8.771	0.000	33.401	52.630
ww2HasN	-39.1821	10.091	-3.883	0.000	-58.964	-19.400
ww3HasN	-98.2029	25.285	-3.884	0.000	-147.770	-48.636
ww4HasN	-75.8744	31.553	-2.405	0.016	-137.730	-14.019

Omnibus:	1134.082	**Durbin-Watson:**	0.477
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2098.198
Skew:	1.140	**Prob(JB):**	0.00
Kurtosis:	4.709	**Cond. No.**	2.28e+15

Bibliography

Divyam Aggarwal, Dhish Kumar Saxena, Thomas Bäck, and Michael Emmerich. A novel column generation heuristic for airline crew pairing optimization with large-scale complex flight networks. *arXiv*, 7744:0–3, 2020. ISSN 23318422.

Uwe Aickelin and Kathryn A Dowsland. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling*, 3(3):139–153, 2000. ISSN 1099-1425. doi: 10.1002/(SICI)1099-1425(200005/06)3:3<139::AID-JOS41>3.0.CO;2-2.

S. M. Al-Yakoob and H. D. Sherali. A column generation approach for an employee scheduling problem with multiple shifts and work locations. *Journal of the Operational Research Society*, 59(1):34–43, 2008. ISSN 14769360. doi: 10.1057/palgrave.jors.2602294.

Vítor Barbosa, Ana Respício, and Filipe Alvelos. A Column Generation Based Heuristic for a Bus Driver Rostering Problem. In *Progress in Artificial Intelligence*, volume 1, pages 143–156. 2015. ISBN 9783319234854. doi: 10.1007/978-3-319-23485-4.

Jonathan F. Bard and Hadi W. Purnomo. Preference scheduling for nurses using column generation. *European Journal of Operational Research*, 164(2):510–534, 2005a. ISSN 03772217. doi: 10.1016/j.ejor.2003.06.046.

Jonathan F. Bard and Hadi W. Purnomo. A column generation-based approach to solve the preference scheduling problem for nurses with downgrading. *Socio-Economic Planning Sciences*, 39(3):193–213, 2005b. ISSN 00380121. doi: 10.1016/j.seps.2004.04.001.

Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, W P Martin, No May Jun, Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin W P Savelsbergh, and Pamela H Vance. Branch-and-Price : Column Generation for Solving Huge Integer Programs Savelsbergh and Pamela H . Vance Published by : INFORMS Stable URL : http://www.jstor.org/stable/222825 REFERENCES Linked references are available on JSTOR for this article : You may n. *Operations Research*, 46(3):316–329, 1998.

Nicholas Beaumont. Scheduling staff using mixed integer programming. *European Journal of Operational Research*, 98(3):473–484, 1997. ISSN 03772217. doi: 10.1016/S0377-2217(97)00055-6.

URL http://www.scopus.com/inward/record.url?eid=2-s2.0-0031140763&partnerID=tZOtx3y1.

Jeroen Belien and Erik Demeulemeester. Heuristic branch-and-price for building long term trainee schedules. *DTEW Research Report 0422*, pages 1–22, 2004. URL https://lirias.kuleuven.be/bitstream/123456789/85433/1/OR_0422.pdf.

Jeroen Beliën and Erik Demeulemeester. Scheduling trainees at a hospital department using a branch-and-price approach. *European Journal of Operational Research*, 175(1):258–278, 2006. ISSN 03772217. doi: 10.1016/j.ejor.2005.04.028.

Jeroen Beliën and Erik Demeulemeester. On the trade-off between staff-decomposed and activity-decomposed column generation for a staff scheduling problem. *Annals of Operations Research*, 155(1):143–166, 2007. ISSN 02545330. doi: 10.1007/s10479-007-0220-2.

Jeroen Beliën and Erik Demeulemeester. A branch-and-price approach for integrating nurse and surgery scheduling. *European Journal of Operational Research*, 189(3):652–668, 2008. ISSN 03772217. doi: 10.1016/j.ejor.2006.10.060.

F. Bellanti, G. Carello, F. Della Croce, and R. Tadei. A greedy-based neighborhood search approach to a nurse rostering problem. *European Journal of Operational Research*, 153(1): 28–40, 2004. ISSN 03772217. doi: 10.1016/S0377-2217(03)00096-1.

Marco Bender, Sebastian Berckey, Michael Elberfeld, and Jörg Herbers. Real-World Staff Rostering via Branch-and-Price in a Declarative Framework. pages 445–451, 2019. doi: 10.1007/978-3-030-18500-8{_}55.

Khaled Boubaker, Guy Desaulniers, and Issmail Elhallaoui. Bidline scheduling with equity by heuristic dynamic constraint aggregation. *Transportation Research Part B: Methodological*, 44 (1):50–61, 2010. ISSN 01912615. doi: 10.1016/j.trb.2009.06.003. URL http://dx.doi.org/10.1016/j.trb.2009.06.003.

Vincent Boyer, Bernard Gendron, and Louis Martin Rousseau. A branch-and-price algorithm for the multi-activity multi-task shift scheduling problem. *Journal of Scheduling*, 17(2):185–197, 2014. ISSN 10946136. doi: 10.1007/s10951-013-0338-9.

Jens O. Brunner and Jonathan F. Bard. Flexible weekly tour scheduling for postal service workers using a branch and price. *Journal of Scheduling*, 16(1):129–149, 2013. ISSN 10946136. doi: 10.1007/s10951-011-0265-6.

Jens O. Brunner and Günther M. Edenharter. Long term staff scheduling of physicians with different experience levels in hospitals using column generation. *Health Care Management Science*, 14(2):189–202, 2011. ISSN 13869620. doi: 10.1007/s10729-011-9155-x.

Jens O. Brunner and Raik Stolletz. Stabilized branch and price with dynamic parameter updating for discontinuous tour scheduling. *Computers and Operations Research*, 44:137–145, 2014. ISSN 03050548. doi: 10.1016/j.cor.2013.11.004. URL `http://dx.doi.org/10.1016/j.cor.2013.11.004`.

Jens O. Brunner, Jonathan F. Bard, and Rainer Kolisch. Midterm scheduling of physicians with flexible shifts using branch and price. *IIE Transactions (Institute of Industrial Engineers)*, 43 (2):84–109, 2011. ISSN 0740817X. doi: 10.1080/0740817X.2010.504685.

Eduard Bulog. A Nurse Rostering Algorithm with Compile-Time Customisation and Neighbourhood-Constrained Column Generation. 1994, 2011.

E K Burke, T Curtois, R Qu, and G Vanden Berghe. A scatter search methodology for the nurse rostering problem. *Journal of the Operational Research Society*, 61(11):1667–1679, 2010a. ISSN 0160-5682. doi: 10.1057/jors.2009.118. URL `http://link.springer.com/10.1057/jors.2009.118`.

Edmund K. Burke and Tim Curtois. New approaches to nurse rostering benchmark instances. *European Journal of Operational Research*, 237(1):71–81, 2014. ISSN 03772217. doi: 10.1016/j.ejor.2014.01.039. URL `http://dx.doi.org/10.1016/j.ejor.2014.01.039`.

Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7(6):441–449, 2004a. ISSN 10946136. doi: 10.1023/B:JOSH.0000046076.75950.0b.

Edmund K. Burke, Timothy Curtois, Gerhard Post, Rong Qu, and Bart Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, 188(2):330–341, 2008. ISSN 03772217. doi: 10.1016/j.ejor.2007.04.030.

Edmund K. Burke, Jingpeng Li, and Rong Qu. A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research*, 203(2):484–493, 2010b. ISSN 03772217. doi: 10.1016/j.ejor.2009.07.036.

Edmund K Ek Burke and Tim Curtois. An ejection chain method and a branch and price algorithm applied to the instances of the first international nurse rostering competition, 2010. *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling PATAT*, 10:13, 2010. URL `https://www.kuleuven-kortrijk.be/~u0041139/nrpcompetition/abstracts/s4.pdf%5Cnhttp://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:An+ejection+chain+method+and+a+branch+and+price+algorithm+applied+to+the+instances+of+the+first+international+nurse`.

Ek K Burke, Patrick De Causmaecker, and Greet Vanden Berghe. Novel meta-heuristic approaches to nurse rostering problems in Belgian hospitals. *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, 18(September):1–26, 2004b. doi: doi:10.1201/9780203489802.ch44. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1.5770&rep=rep1&type=pdf`.

Patrick De Causmaecker. International Nurse Rostering Competition instance set, 2010. URL `https://www.kuleuven-kulak.be/nrpcompetition/instances`.

B. Cheang, H. Li, A. Lim, and B. Rodrigues. Nurse rostering problems - A bibliographic survey. *European Journal of Operational Research*, 151(3):447–460, 2003. ISSN 03772217. doi: 10.1016/S0377-2217(03)00021-3.

Jens Clausen. Branch and Bound Algorithms-Principles and Examples. Technical report, 1999.

Marie Claude Côté, Bernard Gendron, and Louis Martin Rousseau. Grammar-based column generation for personalized multi-activity shift scheduling. *INFORMS Journal on Computing*, 25(3):461–474, 2013. ISSN 10919856. doi: 10.1287/ijoc.1120.0514.

Said Dabia, Stefan Ropke, Tom Van Woensel, and Ton De Kok. Branch and price for the time-dependent vehicle routing problem with time windows. *Transportation Science*, 47(3):380–396, 2013. ISSN 15265447. doi: 10.1287/trsc.1120.0445.

Sana Dahmen, Rekik Monia, Francois Soumis, and Guy Desauliniers. A Branch-and-Price-and-Cut Algorithm for Adjusting Schedules in a Multi-Department Context. *11e Congres International de Genie Industriel*, 2015. URL `http://www.simagi.polymtl.ca/congresgi/cigi2015/Articles/CIGI_2015_submission_105.pdf`.

George Dantzig. *Linear Programming and Extensions*. Princeton University Press, 12 1963. ISBN 9781400884179. doi: 10.1515/9781400884179.

George B. Dantzig and Philip Wolfe. Decomposition Principle for Linear Programs. *Operations Research*, 8(1):101–111, 2 1960. ISSN 0030-364X. doi: 10.1287/opre.8.1.101.

Sophie Demassey, Gilles Pesant, and Louis Martin Rousseau. A cost-regular based hybrid column generation approach. *Constraints*, 11(4):315–333, 2006. ISSN 13837133. doi: 10.1007/s10601-006-9003-7.

M. Van Den Eeckhout, M. Vanhoucke, and B. Maenhout. A column generation-based diving heuristic to solve the multi-project personnel staffing problem with calendar constraints and resource sharing. *Computers & Operations Research*, page 105163, 2020. ISSN 03050548. doi: 10.1016/j.cor.2020.105163. URL `https://doi.org/10.1016/j.cor.2020.105163`.

Guy Desaulniers. Branch-and-price-and-cut for the split-delivery vehicle routing problem with time windows. *Operations Research*, 58(1):179–192, 2010. ISSN 0030364X. doi: 10.1287/opre.1090.0713.

Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon. *Column Generation*, volume 82. 2005. ISBN 0-387-25486-2. doi: 10.1007/b135457. URL `http://www.springerlink.com/content/978-0-387-25485-2/`.

Guy Desaulniers, Jacques Desrosiers, and Simon Spoorendonk. The Vehicle Routing Problem with Time Windows: State-of-the-Art Exact Solution Methods. *Wiley Encyclopedia of Operations Research and Management Science*, (1):1–8, 2011. doi: 10.1002/9780470400531.eorms1034.

Guy Desaulniers, François Lessard, Mohammed Saddoune, and François Soumis. Dynamic Constraint Aggregation for Solving Very Large-scale Airline Crew Pairing Problems. *SN Operations Research Forum*, 1(3):1–23, 2020. doi: 10.1007/s43069-020-00016-1.

Jacques Desrosiers and Marco E. Lübbecke. A primer in column generation. *Column Generation*, (May 2014):1–32, 2005. doi: 10.1007/0-387-25486-2{_}1.

Jacques Desrosiers and Marco E. Lübbecke. Branch-Price-and-Cut Algorithms. *Wiley Encyclopedia of Operations Research and Management Science*, (January), 2011. doi: 10.1002/9780470400531.eorms0118.

Anders Dohn and Andrew Mason. Branch-and-price for staff rostering: An efficient implementation using generic programming and nested column generation. *European Journal of Operational Research*, 230(1):157–169, 2013. ISSN 03772217. doi: 10.1016/j.ejor.2013.03.018. URL `http://dx.doi.org/10.1016/j.ejor.2013.03.018`.

J.J. Dopheide and R. Spliet. Solving Nurse Rostering Problems with Lagrangian Relaxation and Column Generation. 2018.

O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194(1-3):229–237, 1999. ISSN 0012365X. doi: 10.1016/S0012-365X(98)00213-1.

Employment NZ. Rostering, 2020. URL `https://www.employment.govt.nz/hours-and-wages/hours-of-work/rostering/`.

Julian Enoch. Advanced Column Generation Decompositions for Optimizing Provisioning Problems in Optical Networks. (June), 2018.

P. Erdos. Colloque sur la Théorie des Nombres. *The Mathematical Gazette*, 41:127–137, 1957.

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*, 153(1):3–27, 2004. ISSN 03772217. doi: 10.1016/S0377-2217(03)00095-X.

M. Firat, D. Briskorn, and A. Laugier. A Branch-and-Price algorithm for stable workforce assignments with hierarchical skills. *European Journal of Operational Research*, 251(2):676–685, 2016. ISSN 03772217. doi: 10.1016/j.ejor.2015.11.039.

Matteo Fischetti and Michele Monaci. Exploiting erraticism in search. *Operations Research*, 62 (1):114–122, 2014. ISSN 0030364X. doi: 10.1287/opre.2013.1231.

John J Forrest, Stefan Vigerske, Ted Ralphs, Lou Hafer, Haroldo Gambini Santos, Matthew Saltzman, Bjarni Kristjansson, and Alan King. coin-or/Clp: Version 1.17.6, 2020. URL `https://doi.org/10.5281/zenodo.3748677`.

Rosario G. Garroppo, Stefano Giordano, and Luca Tavanti. A survey on multi-constrained optimal path computation: Exact and approximate algorithms. *Computer Networks*, 54(17): 3081–3107, 2010. ISSN 13891286. doi: 10.1016/j.comnet.2010.05.017. URL `http://dx.doi.org/10.1016/j.comnet.2010.05.017`.

Matthieu Gérard, François Clautiaux, and Ruslan Sadykov. Column generation based approaches for a tour scheduling problem with a multi-skill heterogeneous workforce. *European Journal of Operational Research*, 252(3):1019–1030, 2016. ISSN 03772217. doi: 10.1016/j.ejor.2016.01.036.

Celia A. Glass and Roger A. Knight. The nurse rostering problem: A critical appraisal of the problem structure. *European Journal of Operational Research*, 202(2):379–389, 2010. ISSN 03772217. doi: 10.1016/j.ejor.2009.05.046.

Rafael A.M. Gomes, Túlio A.M. Toffolo, and Haroldo Gambini Santos. Variable neighborhood search accelerated column generation for the nurse rostering problem. *Electronic Notes in Discrete Mathematics*, 58:31–38, 2017. ISSN 15710653. doi: 10.1016/j.endm.2017.03.005.

Balaji Gopalakrishnan and Ellis L. Johnson. Airline crew scheduling: State-of-the-art. *Annals of Operations Research*, 140(1):305–337, 2005. ISSN 02545330. doi: 10.1007/s10479-005-3975-3.

Gabriel Gutiérrez-Jarpa, Guy Desaulniers, Gilbert Laporte, and Vladimir Marianov. A branch-and-price algorithm for the Vehicle Routing Problem with Deliveries, Selective Pickups and Time Windows. *European Journal of Operational Research*, 206(2):341–349, 2010. ISSN 03772217. doi: 10.1016/j.ejor.2010.02.037. URL `http://dx.doi.org/10.1016/j.ejor.2010.02.037`.

Walter J. Gutjahr and Marion S. Rauner. An ACO algorithm for a dynamic regional nurse-scheduling problem in Austria. *Computers and Operations Research*, 34(3):642–666, 2007. ISSN 03050548. doi: 10.1016/j.cor.2005.03.018.

Pierre Hansen, Nenad Mladenović, Raca Todosijević, and Saïd Hanafi. Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, 5(3):423–454, 2016. ISSN 21924414. doi: 10.1007/s13675-016-0075-x.

Stefaan Haspeslagh, Patrick De Causmaecker, Martin Stølevik, and Andrea Schaerf. First international nurse rostering competition 2010. *PATAT 2010 - Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling*, pages 498–501, 2010.

Fang He and Rong Qu. A constraint programming based column generation approach to nurse rostering problems. *Computers and Operations Research*, 39(12):3331–3343, 2012. ISSN 03050548. doi: 10.1016/j.cor.2012.04.018. URL http://dx.doi.org/10.1016/j.cor.2012.04.018.

Julia Heil, Kirsten Hoffmann, and Udo Buscher. Railway crew scheduling: Models, methods and applications. *European Journal of Operational Research*, 283(2):405–425, 2020. ISSN 03772217. doi: 10.1016/j.ejor.2019.06.016.

Han Hoogeveen and Eelko Penninkx. Finding Near-Optimal Rosters Using Column Generation Han Hoogeveen Eelko Penninkx Finding Near-Optimal Rosters Using Column Generation. *Knowledge Creation Diffusion Utilization*, 2007.

Stefan Irnich and Guy Desaulniers. Shortest Path Problems with Resource Constraints. In Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon, editors, *Column Generation*, pages 33–65. Springer US, Boston, MA, 2005. ISBN 978-0-387-25486-9. doi: 10.1007/0-387-25486-2{_}2. URL https://doi.org/10.1007/0-387-25486-2_2.

Jonas Karl and Christoph Volland. Strategic and Tactical Scheduling of Logistics Assistants Leveraging Flexibility in Shifts and Tasks Using Column Generation – An Opportunity for Hospital Logistics. 2017.

Richard M. Karp. Reducibility among combinatorial problems. *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 219–241, 2010. ISSN 00224812. doi: 10.1007/978-3-540-68279-0{_}8.

Atoosa Kasirzadeh, Mohammed Saddoune, and François Soumis. Airline crew scheduling: models, algorithms, and data sets. *EURO Journal on Transportation and Logistics*, 6(2):111–137, 2017. ISSN 21924384. doi: 10.1007/s13676-015-0080-x.

Antoine Legrain, Jérémy Omer, and Samuel Rosat. *A rotation-based branch-and-price approach for the nurse scheduling problem*, volume 12. Springer Berlin Heidelberg, 2020. ISBN 1253201900. doi: 10.1007/s12532-019-00172-4. URL https://doi.org/10.1007/s12532-019-00172-4.

Warner Lensing. *Heuristic Branch-and-Price algorithms for the nurse rostering problem.* PhD thesis, University of Groningen, 2020.

Gino Lim and Arezou Mobasher. Operating suite nurse scheduling problem: A heuristic approach. *62nd IIE Annual Conference and Expo 2012*, pages 1071–1079, 2012.

Zhipeng Lü and Jin Kao Hao. Adaptive neighborhood search for nurse rostering. *European Journal of Operational Research*, 218(3):865–876, 2012. ISSN 03772217. doi: 10.1016/j.ejor.2011.12.016. URL http://dx.doi.org/10.1016/j.ejor.2011.12.016.

Marco E. Lübbecke. Column Generation. *Wiley Encyclopedia of Operations Research and Management Science*, 2010. ISSN 0031918X. doi: 10.1134/S0031918X15030102.

R. Lusby, A. Dohn, T. M. Range, and J. Larsen. A column generation-based heuristic for rostering with work patterns. *Journal of the Operational Research Society*, 63(2):261–277, 2012. ISSN 14769360. doi: 10.1057/jors.2011.27. URL http://dx.doi.org/10.1057/jors.2011.27.

Broos Maenhout and Mario Vanhoucke. A branch-and-price procedure for nurse staffing incorporating roster preferences. *Multidisciplinary scheduling : theory and applications*, (May), 2007. URL http://hdl.handle.net/1854/LU-666135.

Broos Maenhout and Mario Vanhoucke. Branching strategies in a branch-and-price approach for a multiple objective nurse scheduling problem. *Journal of Scheduling*, 13(1):77–93, 2010a. ISSN 10946136. doi: 10.1007/s10951-009-0108-x.

Broos Maenhout and Mario Vanhoucke. A hybrid scatter search heuristic for personalized crew rostering in the airline industry. *European Journal of Operational Research*, 206(1):155–167, 2010b. ISSN 03772217. doi: 10.1016/j.ejor.2010.01.040. URL http://dx.doi.org/10.1016/j.ejor.2010.01.040.

Vittorio Maniezzo, Marco Antonio Boschetti, and Thomas Stutzle. *Matheuristics: Algorithms and Implementations*. Springer Nature, 2021.

Victor Manuel and Meneses Barbosa. Bus Driver Rostering by Hybrid Methods Based on Column Generation. 2018.

Hannah Martin and Sam Kilmister. Short-staffing of New Zealand hospitals is putting patients and staff at risk – report, 2018. URL https://www.stuff.co.nz/national/health/107793693/shortstaffing-of-new-zealand-hospitals-putting-patients-staff-at-risk-report.

Andrew Mason, Ed Bulog, and Anders Dohn. Using Nested Column Generation & Generic Programming to solve Staff Scheduling Problems: Using Compile-time Customisation to create a Flexible C++ Engine for Staff Rostering. 2011.

Andrew J. Mason. Elastic Constraint Branching, the Wedelin/Carmen Lagrangian Heuristic and Integer Programming for Personnel Scheduling. *Annals of Operations Research*, 108(1-4): 239–276, 2001. ISSN 15729338. doi: 10.1023/A:1016023415105.

Andrew J. Mason and Mark C Smith. A Nested Column Generator for solving Rostering Problems with Integer Programming. pages 827–834, 1998. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.1832.

Anuj Mehrotra, Kenneth E. Murphy, and Michael A. Trick. Optimal Shift Scheduling: A Branch-and-Price Approach. *Naval Research Logistics*, 47(3):185–200, 2000. ISSN 0894069X. doi: 10.1002/(SICI)1520-6750(200004)47:3<185::AID-NAV1>3.0.CO;2-7.

Coen Meijer. Cyclical personnel scheduling using column generation. 2018.

Florian Mischek and Nysret Musliu. Integer Programming and Heuristic Approaches for a Multi-Stage Nurse Rostering Problem. *Proceedings of the 11th International Confenference on Practice and Theory of Automated Timetabling (PATAT-2016)*, pages 245–262, 2016.

Ibrahim Muter, Ş Ilker Birbil, Kerem Bülbül, Güvenç Şahin, Hüsnü Yenigün, Duygu Taş, and Dilek Tüzün. Solving a robust airline crew pairing problem with column generation. *Computers and Operations Research*, 40(3):815–830, 2013. ISSN 03050548. doi: 10.1016/j.cor.2010.11.005.

Hua Ni and Hernán Abeledo. A branch-and-price approach for large-scale employee tour scheduling problems. *Annals of Operations Research*, 155(1):167–176, 2007. ISSN 02545330. doi: 10.1007/s10479-007-0212-2.

Koji Nonobe. INRC2010: An approach using a general constraint optimization solver. 2010.

Antonin Novak. Methods of the efficient state space search for the Nurse Rostering Problem using branch-and-price approach. *Master's Thesis*, 2015.

NZ Ministry of Health. New Zealand's nursing workforce the largest it's ever been, 2019. URL `https://www.health.govt.nz/news-media/media-releases/new-zealands-nursing-workforce-largest-its-ever-been`.

NZ Nurses Organisation. DHB NZNO MECA, 2018. URL `https://www.nzno.org.nz/groups/health_sectors/dhb`.

Makoto Ohara and Hisashi Tamaki. Mathematical programming approach based on column generation for a class of staff scheduling problems. *2015 54th Annual Conference of the Society of Instrument and Control Engineers of Japan, SICE 2015*, pages 240–245, 2015. doi: 10.1109/SICE.2015.7285552.

Pattarapong Pakpoom and Peerayuth Charnsethikul. A Column Generation Approach for Personnel Sched uling with Discrete Uncertain Requirements. *2018 2nd International Conference on Informatics and Computational Sciences, ICICoS 2018*, pages 185–190, 2019. doi: 10.1109/ICICOS.2018.8621664.

Warren B. Powell. What You Should Know About Approximate Dynamic Programming. *Naval Research Logistics*, 55(April 2007):541–550, 2009. ISSN 0894069X. doi: 10.1002/nav. URL `http://www.interscience.wiley.com/jpages/0894-069X/`.

P. Prosser and C. Unsworth. Limited discrepancy search revisited. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 16(1):607–613, 2011. ISSN 10846654. doi: 10.1145/1963190.2019581.

Hadi W. Purnomo and Jonathan F. Bard. Cyclic preference scheduling for nurses using branch and price. *Naval Research Logistics*, 54(2):200–220, 2007. ISSN 0894069X. doi: 10.1002/nav.20201.

Frédéric Quesnel, Guy Desaulniers, and François Soumis. Improving air crew rostering by considering crew preferences in the crew pairing problem. *Transportation Science*, 54(1):97–114, 2020. ISSN 15265447. doi: 10.1287/trsc.2019.0913.

Erfan Rahimian, Kerem Akartunal, and John Levine. A hybrid integer and constraint programming approach to solve nurse rostering problems. *Computers and Operations Research*, 82: 83–94, 2017a. ISSN 03050548. doi: 10.1016/j.cor.2017.01.016. URL `http://dx.doi.org/10.1016/j.cor.2017.01.016`.

Erfan Rahimian, Kerem Akartunalı, and John Levine. A hybrid Integer Programming and Variable Neighbourhood Search algorithm to solve Nurse Rostering Problems. *European Journal of Operational Research*, 258(2):411–423, 2017b. ISSN 03772217. doi: 10.1016/j.ejor.2016.09.030. URL `http://dx.doi.org/10.1016/j.ejor.2016.09.030`.

Ted Ralphs. COIN/BCP User's Manual. (September 2001), 2013.

Maria I. Restrepo, Bernard Gendron, and Louis Martin Rousseau. Combining Benders decomposition and column generation for multi-activity tour scheduling. *Computers and Operations Research*, 93:151–165, 2018. ISSN 03050548. doi: 10.1016/j.cor.2018.01.014. URL `https://doi.org/10.1016/j.cor.2018.01.014`.

María I. Restrepo, Leonardo Lozano, and Andrés L. Medaglia. Constrained network-based column generation for the multi-activity shift scheduling problem. *International Journal of Production Economics*, 140(1):466–472, 2012. ISSN 09255273. doi: 10.1016/j.ijpe.2012.06.030.

María I. Restrepo, Bernard Gendron, and Louis Martin Rousseau. Branch-and-price for personalized multiactivity tour scheduling. *INFORMS Journal on Computing*, 28(2):334–350, 2016. ISSN 15265528. doi: 10.1287/ijoc.2015.0683.

D. M. Ryan and B. A. Foster. An integer programming approach to b-coloring. *Discrete Optimization*, 32(October):43–62, 1981. ISSN 15725286. doi: 10.1016/j.disopt.2018.12.001.

Ruslan Sadykov. Modern Branch-Cut-and-Price. 2019.

Ruslan Sadykov, Eduardo Uchoa, and Artur Pessoa. A Bucket Graph–Based Labeling Algorithm with Application to Vehicle Routing. *Transportation Science*, pages 1–53, 2020. ISSN 0041-1655. doi: 10.1287/trsc.2020.0985.

Haroldo G. Santos, Túlio A.M. Toffolo, Rafael A.M. Gomes, and Sabir Ribas. Integer programming techniques for the nurse rostering problem. *Annals of Operations Research*, 239(1): 225–251, 2016. ISSN 15729338. doi: 10.1007/s10479-014-1594-6. URL `http://dx.doi.org/10.1007/s10479-014-1594-6`.

Alexander Schrijver. *Theory of linear and integer programming.* John Wiley and Sons, 1998.

Pieter Smet, Andreas T. Ernst, and Greet Vanden Berghe. Heuristic decomposition approaches for an integrated task scheduling and personnel rostering problem. *Computers and Operations Research*, 76:60–72, 2016. ISSN 03050548. doi: 10.1016/j.cor.2016.05.016. URL `http://dx.doi.org/10.1016/j.cor.2016.05.016`.

Petter Strandmark, Yi Qu, and Timothy Curtois. First-order linear programming in a column generation-based heuristic approach to the nurse rostering problem. *Computers and Operations Research*, 120, 2020. ISSN 03050548. doi: 10.1016/j.cor.2020.104945.

Lars Sundqvist Swahn. A Column Generation Method for Minimization of Shift Costs at an Airport. 2019.

Adil Tahir, Guy Desaulniers, and Issmail El Hallaoui. *Integral column generation for the set partitioning problem*, volume 8. THE AUTHORS. Published by Elsevier on behalf of the Association of European Operational Research Societies (EURO)., 2019. ISBN 0123456789. doi: 10.1007/s13676-019-00145-6. URL `http://dx.doi.org/10.1007/s13676-019-00145-6`.

Christian Tilk, Michael Drexl, and Stefan Irnich. Nested branch-and-price-and-cut for vehicle routing problems with multiple resource interdependencies. *European Journal of Operational Research*, 276(2):549–565, 2019. ISSN 03772217. doi: 10.1016/j.ejor.2019.01.041. URL `https://doi.org/10.1016/j.ejor.2019.01.041`.

Nikola Todorovic and Sanja Petrovic. Bee Colony Optimization Algorithm for Nurse Roster. *Ieee Transactions on Systems, Man, and Cybernetics: Systems, Vol. 43, No. 2, March 2013*, VOL. 43, N(2):467–73, 2013. ISSN 2168-2216. doi: 10.1109/TSMCA.2012.2210404.

Roman Václavík, Antonín Novák, Přemysl Šůcha, and Zdeněk Hanzálek. Accelerating the Branch-and-Price Algorithm Using Machine Learning. *European Journal of Operational Research*, 271(3):1055–1069, 2018. ISSN 03772217. doi: 10.1016/j.ejor.2018.05.046.

Christos Valouxis, Christos Gogos, George Goulas, Panayiotis Alefragis, and Efthymios Housos. A systematic two phase approach for the nurse rostering problem. *European Journal of Operational Research*, 219(2):425–433, 2012. ISSN 03772217. doi: 10.1016/j.ejor.2011.12.042. URL `http://dx.doi.org/10.1016/j.ejor.2011.12.042`.

Jorne Van Den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385, 2013. ISSN 03772217. doi: 10.1016/j.ejor.2012.11.029. URL `http://dx.doi.org/10.1016/j.ejor.2012.11.029`.

M. Van Den Eeckhout, M. Vanhoucke, and B. Maenhout. A decomposed branch-and-price procedure for integrating demand planning in personnel staffing problems. *European Journal of Operational Research*, 280(3):845–859, 2020. ISSN 03772217. doi: 10.1016/j.ejor.2019.07.069.

François Vanderbeck. Branching in Branch-and-Price : a Generic Scheme. 2009.

Jonas Volland, Andreas Fügener, and Jens O. Brunner. A column generation approach for the integrated shift and task scheduling problem of logistics assistants in hospitals. *European Journal of Operational Research*, 260(1):316–334, 2017. ISSN 03772217. doi: 10.1016/j.ejor.2016.12.026. URL `http://dx.doi.org/10.1016/j.ejor.2016.12.026`.

Toby Walsh. Depth-bounded discrepancy search. *IJCAI International Joint Conference on Artificial Intelligence*, 2(1934):1388–1393, 1997. ISSN 10450823.

Shu Wang. Workforce scheduling with large-scale mixed integer programming using column generation and 2d genetic algorithms: an application to airport ground staff scheduling. (December), 2019.

Yong Min Wang. A column generation approach for stochastic optimization problems. *ProQuest Dissertations and Theses*, page 151, 2006. URL `https://search.proquest.com/docview/304979135?accountid=188395`.

Emilio Zamorano and Raik Stolletz. Branch-and-price approaches for the Multiperiod Technician Routing and Scheduling Problem. *European Journal of Operational Research*, 257(1):55–68, 2017. ISSN 03772217. doi: 10.1016/j.ejor.2016.06.058. URL `http://dx.doi.org/10.1016/j.ejor.2016.06.058`.

Vahid Zeighami and François Soumis. Combining benders' decomposition and column generation for integrated crew pairing and personalized crew assignment problems. *Transportation Science*, 53(5):1479–1499, 2019. ISSN 15265447. doi: 10.1287/trsc.2019.0892.

Lishun Zeng, Mingyu Zhao, and Chunlei Mu. Airport Ground Staff-sizing with Hierarchical Skills Using Column Generation. *PATAT 2016 - Proceedings of the 11th International Conference on the Practice and Theory of Automated Timetabling*, pages 569–572, 2016.

Lishun Zeng, Mingyu Zhao, and Yangfan Liu. Airport ground workforce planning with hierarchical skills: a new formulation and branch-and-price approach. *Annals of Operations Research*, 275(1):245–258, 2019. ISSN 15729338. doi: 10.1007/s10479-017-2624-y.

Bahad Ir Zeren and Ibrahim Özkol. A novel column generation strategy for large scale airline crew pairing problems. *Expert Systems with Applications*, 55:133–144, 2016. ISSN 09574174. doi: 10.1016/j.eswa.2016.01.045.