Train Driver Scheduling with Windows of Relief Opportunities

Ignacio Eduardo Laplagne

Submitted in accordance with the requirements for the degree of Doctor of Philosophy.

UNIVERSITY OF LEEDS

School of Computing

January 2008

Abstract

Train driver scheduling is the problem of finding an assignment of drivers to cover all train vehicle work, such that cost is minimized while all constraints are satisfied. Relieving of drivers happens mostly at train stops; in many cases the train will stop for several minutes, giving rise to a window of relief opportunities (WRO), but current industry practice is to consider relieving only at the time the train arrives at the station. This thesis studies an extension of the train driver scheduling problem to exploit relieving drivers within WROs at times other than the arrival. While extending the problem in this way may lead to a more expressive model, and better solutions, it also gives rise to problem sizes too large for existing scheduling methods.

Two different approaches to solve the problem of driver scheduling with WROs are presented. In the first, relief opportunities inside WROs are evaluated in terms of their role in generating solutions unavailable to the relief-on-arrival formulation. Heuristics based on this evaluation result in instance sizes that are manageable with current generate-and-select (GaS) methods. Experiments produce new best-known solutions for some real-life instances of the problem.

The second approach is a hybridized system that generates an initial solution using GaS on a relief-on-arrival formulation, which is fed into a local search method running on a full WRO model. This method is then extended by allowing infeasible solutions to be considered during the search. A novel way of costing infeasible solutions is introduced, where the cost of an infeasible solution is derived from a repaired, feasible version of that solution. This strategy is embedded in a local search framework that alternates between exploration of feasible and infeasible regions, where infeasibility arises from complex moves that remove undesirable structural features in the current feasible solution.

Acknowledgements

I would first like to thank the Engineering and Physical Sciences Research Council for their generous support (grant no GR/S20949/1), as well as the School of Computing for waiving the difference between EU and overseas fees when I wasn't yet formally a British Citizen.

I would also like to thank my supervisor Raymond Kwan, who was always available and with whom we have spent countless sessions of healthy discussion and thinking aloud. I am also grateful for the trust he put in me, be it for presenting work in conferences, helping organize them or meeting TRACSiS customers on my own.

The work developed in this thesis is built upon the foundations laid by many in the Scheduling Group at the School of Computing, University of Leeds, before me. I would like to thank Ann Kwan (now in TRACSiS) and Margaret Parker (now retired) for their patience and disposition, and their help in making it possible to integrate my tools with `TrainTRACS`.

From the very start, Raymond and I wanted this PhD project to work on the full scheduling problem, using real-life problem instances, and I was fortunate enough during these years to be involved in solving real-life problems with/for a number of train operators. I would like to thank Jerry Farquharson and Mark Quinn from First Scotrail for their valuable advice, Mike Salmon from Southern Trains for his help and enthusiasm, and the other TOCs which also provided test data or taught me more about railway operations.

Finally and most of all, I would like to thank my wife Natalia and our son Nicolás, who gave up holidays and weekends with dad for the last year and more. This thesis is for them.

Declarations

Parts of the work presented in this thesis have been reported in the following articles and conferences:

Ignacio Laplagne, Raymond S K Kwan, and Ann S K Kwan, *A hybridised integer programming and local search method for robust train driver schedules planning*, Selected papers from *PATAT 2004*, Lecture Notes in Computer Science, vol. 3616, Springer, 2005.

Ignacio Laplagne, Raymond S K Kwan, and Ann S K Kwan, *Exploiting constraint boundaries in train driver scheduling with windows of relief opportunities.* Submitted (2007) to Public Transport: Planning and Operations.

Raymond S K Kwan, Ignacio Laplagne, and Ann S K Kwan, *Train driver scheduling with time windows of relief opportunities.* Presented at CASPT 2004, San Diego, USA, 2004.

Ann S K Kwan, Ignacio Laplagne, and Raymond S K Kwan, *Building robustness into train driver schedules.* Presented at the 2^{nd} Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2005), New York, USA.

Ignacio Laplagne, Raymond S K Kwan, and Ann S K Kwan, *Repair-cost-guided local search for scheduling.* Presented at the 2^{nd} Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2005), New York, USA, and at the 6^{th} Metaheuristics International Conference (MIC 2005), Vienna, Austria.

Ignacio Laplagne, Raymond S K Kwan, and Ann S K Kwan, *Time windows and constraint boundaries for public transport scheduling.* Presented at the 6^{th} International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006), Brno, Czech Republic.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Rail Transport in the UK

In the mid-nineties, rail transport in the UK was privatized. As of 2007, the rail infrastructure is owned by Network Rail, and train operations are split into approximately 25 regional franchises, which are awarded by the Department of Transport (DfT) to private Train Operating Companies (TOCs). The government, through the DfT, controls many aspects of the rail services provided; in particular, it controls to a great extent the so-called *train service patterns* (or *service levels*), which define among others origins and destinations of services, and minimum peak and off-peak frequencies for train services. As an example, we include here an excerpt from a Stakeholder Briefing Document issued by the DfT in relation to the new East Midlands franchise due to start in November 2007 [19]. Note that although the actual specification of services in the franchise bid documents is necessarily more detailed (if only from a legal point of view), it still leaves TOCs considerable room for their own planning/optimization:

Train service pattern

The timetable from June 2006 will continue to operate from the start of the new franchise in November 2007, until December 2008. From the timetable change date in December 2008 the following will operate.

Midland Main Line (MML)

[. . .] the specification from December 2008 will require the introduction of a dedicated service between London and Kettering. [. . .] The off-peak weekday service on the MML will continue to provide 4 trains per hour to and from London St Pancras. Additional trains will run in the London commuter peaks. [. . .] It is likely extra capacity via train lengthening will be proposed by bidders to accommodate the anticipated growth.

- *The hourly London-Sheffield service will operate as now, however, [. . .] the off-peak calls at East Midlands Parkway and Luton stations will be at the bidders' discretion.*
- *The existing hourly fast London-Nottingham service will include a train portion for Derby, with portions splitting and joining at Leicester. The services will run fast between London and Leicester in the off-peak. The Nottingham portion will run fast; the Derby portion will call at Loughborough, East Midlands Parkway and Long Eaton.*
- *The existing hourly Nottingham to London semi-fast service will operate as now, but with additional calls at East Midlands Parkway and the Luton call swapped for Luton Airport Parkway.*
- *A new service will operate between London and Kettering, with intermediate calls at Luton, Bedford and Wellingborough.*

Long-term operations' planning for TOCs in the UK is then driven by the need to implement a timetable of services, including working vehicle and crew schedules, for the train service patterns agreed between the TOC and the DfT as part of the franchise bid. The process of going from an agreed level of service to a fully-formed daily, weekly and monthly set of timetables, vehicle and crew assignment is complex

and long. In the UK, the processes laid by the DfT and other agencies specify in detail milestones, deliverables and interactions with various entities over a period stretching for almost a year before putting a new set of timetables in place.

Conceptually, the planning process is often split into a number of stages that are executed sequentially. We show these in Figure 1.1. In the first stage –*timetabling*– a timetable of *services* or *trips* that satisfies the service level agreed is built. In the next stage, *unit scheduling* (also called *vehicle scheduling*), train formations are assigned to cover those trips. The vehicle schedule specifies the trips for each vehicle in the fleet, along with other operations such as fueling, and cleaning, along with supplementary schedules such as for maintenance, stabling at night, cyclic links between consecutive days, etc. The third stage, *crew scheduling*, consists of assigning notional drivers (*shifts*) to cover the vehicle schedule; other types of crew are usually also part of this stage, e.g. conductors or catering staff. A crew schedule will usually span a period of one day of operation, although in some cases this is extended (for example in long inter-urban services where drivers cannot return to their home depot within the same day). Finally, the *rostering* stage deals with specifying the workload for each individual member of the crew. An individual roster will usually specify the assignments for between a week to several months' work. Rostering involves defining the patterns of work days and rest days, and assigning notional shifts to individual members of the crew.

This conceptual division of the planning process into stages is also frequently followed in practice, although the separation may be less strict; also, there is usually some room for back-tracking or iterating over the stages, particularly between unit and driver scheduling. It is also worth noting that many of these stages involve active negotiation with entities that are external to the TOCs. For example, TOCs in the UK almost always share the rail infrastructure with other TOCs, and they must negotiate its use with Network Rail. Similarly, crew schedules frequently undergo revision from the workers' unions, even if they obey all agreed labour rules.

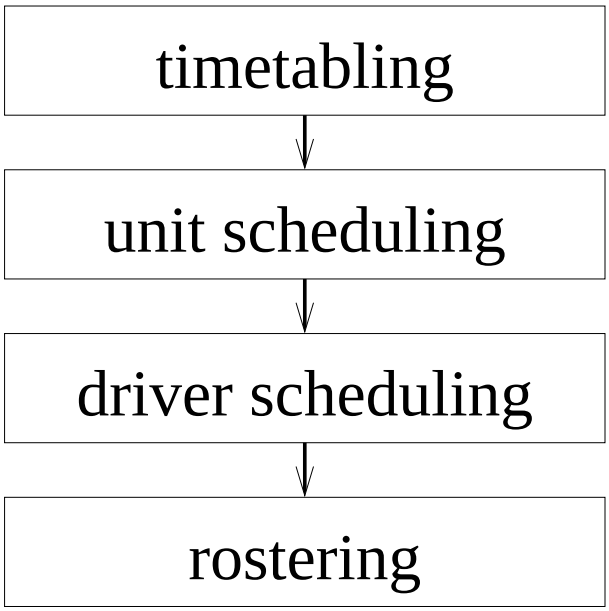

Figure 1.1: Stages in the planning of rail operations.

1.2 Train Driver Scheduling

In this thesis we will concentrate on the problem of driver scheduling in railway operations. Driver scheduling is a key stage in planning of railway operations, since crew costs are frequently the single biggest component of the operating costs of a TOC. In line with the description of the process of planning of public transport above, we will use the following definition:

Definition 1 (Train driver scheduling)**.** *Train driver scheduling* (TDS) is the problem of assigning crew to cover all work as specified in the vehicle schedule, maximizing the efficiency of the resulting schedule while meeting all constraints.

Measures of efficiency/cost and constraints are described in detail in Chapter 2. Notice that although we refer to the problem in terms of assigning drivers, we allow for other types of crew to be part of the scheduling exercise – throughout this thesis, we will use the terms 'driver scheduling' and 'crew scheduling' interchangeably. It should be clear however that any personnel that is not part of the train crew (e.g. staff at the ticket offices) is not encompassed in this problem.

Computational Complexity Driver scheduling belongs to the class of *combinatorial optimization* problems, where the set of feasible solutions is discrete; the reader is directed to Section 3.1.2 for a formulation of the problem where this discreteness is easy to appreciate. Many authors have shown that crew scheduling problems belong in general to the class of *NP-hard* problems. In this thesis, we follow the results of Fischetti et al. [27, 28], who link the bus driver scheduling problem to the problems of *Fixed Job Scheduling with Spread-Time Constraints* (FJSS) and *Working-Time Constraints* (FJSW), and show that these are NP-hard, effectively making even the simplest versions of bus driver scheduling NP-hard. Train driver scheduling is regarded as an extension (or generalization) of bus driver scheduling, because of the introduction of multiple depots, and more complex constraints. It can be shown that, in complexity theory terms, bus driver scheduling can be *reduced* to TDS, which means that TDS also belongs to the class of NP-hard problems.

The fact that a problem p belongs to the class of NP-hard problems is an almost-conclusive indication that it is impossible to find an algorithm that guarantees obtaining an optimal solution for any instance of p in polynomial time; doing so would imply proving $P = NP$, while most researchers in combinatorial optimization would seem to work under the assumption that $P \neq NP$. However, it is also true that many NP-hard problems admit algorithms that produce solutions of good enough quality on real-life instances, including optimal solutions in many cases; TDS is one such case. Another important consideration is that even if a problem can be pinpointed to a specific class from the point of view of complexity theory, different algorithms will have different behaviours in practice – this observation underlies all applied research in solving NP-complete and NP-hard problems. In particular, research in the TDS problem is still producing improvements in the solutions obtainable in practice on real-life instances. As an example, recent work by Kwan and Kwan [56] has been shown to improve on best-known solutions for several UK real-life instances, and the algorithm is in regular use by a number of TOCs in the UK.

1.3 Motivation of the Thesis

Research in crew scheduling, and in particular the TDS problem, has been going on for over 40 years. Most of the recent work has concentrated on applying new algorithmic approaches to basically the same underlying model for the problem, with two main aims: improving the cost of solutions, and handling increased problem sizes. For example, the Scheduling Group at the University of Leeds has in the last 10 years experimented with Mathematical Programming and Column Generation, Evolutionary Algorithms, Tabu Search, Constraint Programming, Fuzzy Logic and Hyperheuristics, among others. Other two active research areas in crew scheduling are: a) integrated transport scheduling, where consecutive steps in the diagram in Figure 1.1 are tackled as a single problem (Section 2.1.5), and b) looking at shorter-term transport scheduling, and especially online re-scheduling, where crew schedules are adjusted dynamically as they are executed to adjust for disruptions and delays (Section 2.1.4). However, we believe that there is an area of research in driver scheduling that has been neglected, which is that of looking for ways to improve or extend the existing models for the problem, particularly where the extensions may lead to solutions (driver schedules) that were unavailable on previous models. In particular, we believe the current models for driver scheduling impose limitations on the way that drivers are relieved which are for the most part artificial – and removing them could allow for more expressiveness and better solutions.

In train driver scheduling, opportunities for relieving a driver frequently stretch over a time interval. For example, if a train stops at a station for several minutes, most of the period (usually all but the last minute before departure) can be used for relieving drivers. We will denote any non-empty interval of time that can be used for relieving purposes a *window of relief opportunities* (WRO). A WRO can then be described by a pair of times (start and end time of the interval) and a location. A common practice among both schedulers and scheduling software is to only consider relieving at the start of the interval – on WROs arising from train stops, the start of the interval corresponds to the arrival time of the train at the station. We will then concentrate on two distinct formulations of the problem in relation to the use of WROs:

Definition 2 (relief-on-arrival formulation)**.** We will call any formulation of the driver scheduling problem where drivers are only allowed to relieve on train stops at the arrival time of the train to the station a *relief-on-arrival formulation* (abbr. *RoA formulation*).

Definition 3 (WRO formulation)**.** We will call any formulation of the driver scheduling problem where drivers are allowed to relieve at any valid times within an attended train stop a WRO formulation. Valid relief times within stops are understood to satisfy all other problem-specific constraints.

This thesis deals with the extension of the driver scheduling problem to consider WROs more fully. We call this extension the problem of *Train Driver Scheduling with Windows of Relief Opportunities* (TDSW). The main motivations of incorporating WROs into a model of the train driver scheduling problem are:

It allows the scheduling model to be enriched. Information such as window length and attendance constraints, for example, allows users to describe better the real-world situation. A specific example on how this added flexibility allows to express ways of adding robustness into a schedule which are unavailable on a relief-on-arrival formulation is shown later in this section.

More efficient schedules could be obtained. Given an instance $\mathcal{I}$ of the scheduling problem, the solution space of a model for $\mathcal{I}$ which approximates WROs (e.g. by restricting relieving to certain times within the WROs) is –all other things being equal– a subset of the solution space of a model with no approximations; therefore, optimal solutions in the latter model are always better than or equal to those obtainable in the former. In Section 2.3.1 we build an instance of the problem where the optimal solution on the WRO-aware model is strictly better than that of a RoA approximation. It is worth noting that this is a theoretical property of the solution sets, and this relation may not hold in practice when the algorithms used over these spaces are not exact.

Robustness could be enhanced. Although this may be thought of as a special form of cost reduction, it is worth considering robustness separately; firstly, because

it is becoming one of the main priorities for train operators, particularly in the UK. Second, WROs provide for a very specific way of creating buffers which may help avoid the cascading of delays.

To illustrate how WROs can be used to increase schedule robustness, consider the example in Figure 1.2. Here, the driver in shift s_1 is relieved by the driver in shift s_2. The existence of a 10-minute time window within vehicle v can be exploited to create a 3-minute buffer for the driver of shift s_2, by requiring s_2 to start covering v three minutes before s_1 leaves v (the length of the buffer is arbitrary, and could be set in principle to any other length). Because there is a 3-minute period in which both s_1 and s_2 are covering (part of) the WRO, the schedule effectively allows the driver of s_2 to begin its spell on vehicle v up to three minutes late without causing any disruption to the execution of s_1, hence no disruption to the execution of the whole schedule. This kind of buffer could be easily incorporated into a formulation for driver scheduling with WROs e.g. by adding a constraint that all relieving within WROs (that are at least three minutes long) is subject to a minimum overlap of three minutes. On the contrary, if WROs are modelled as a single $< time, location >$ pair there is no opportunity to consider the creation of this kind of buffer.

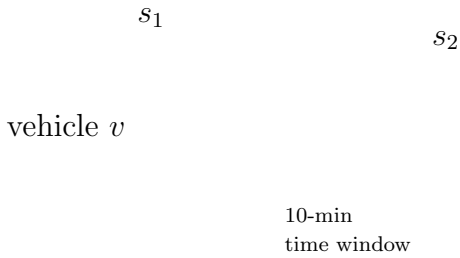

Figure 1.2: An example on how WROs can be used to enhance robustness. Vehicle v contains a 10-minute long WRO (red lines delimit minutes within the time window). By requiring s_2 to start covering the WRO three minutes before s_1 leaves the vehicle, the schedule can absorb a delay of three minutes of s_2 at this relief point without any further propagation.

1.4 Research Questions and Theme of the Thesis

The problem of driver scheduling with WROs has so far been studied in only a very limited way, and there are many basic questions that have not been tackled

properly. In this section we describe what we think are the main questions that need addressing, given the current state of research in the area.

The fundamental questions about any extension or substitute for an existing model for a problem any researcher would probably want to answer are: is this model better than the existing ones, and if so, in what way, and under what circumstances? The comparison between models can be carried out at a theoretical level, at a practical level, or both. In this thesis we put the emphasis on the impact in real-life applications; hence, the theme of the thesis is that of investigating whether exploiting WROs allows to obtain better schedules in practice, on real-life instances.

Still, and as a starting point, a question that has not been answered formally for the train driver scheduling problem is whether it can be proved that a model that considers WROs fully can contain solutions that are not available in an otherwise equivalent relief-on-arrival formulation. Although this is intuitively true, we tackle this question formally by constructing instances of the driver scheduling problem where the optimal cost achievable on the WRO formulation is strictly less than the one on the RoA formulation.

As we move closer to the practical impact of the model, it is important to study whether there are structural differences between the problem of driver scheduling with WROs and the one where relieving is done on arrival, and if so what are these differences. Again, although this analysis could be done from a purely theoretical point of view, we are interested in finding out how these differences impact on the application of existing algorithms for the RoA formulation to the WRO formulation.

In the thesis we find that current approaches to solving the TDS problem in real-life problems break down when applied to their corresponding WRO formulations; this is likely one of the main reasons why this extension has not received enough attention in the literature. However, the fact that there are noticeable structural differences between the two formulations suggests that this structure might be exploited effectively when designing new algorithms for the WRO formulation. The central part of this thesis is then devoted to finding and studying new algorithms that are specific to the WRO formulation, with the aim of improving on the results that can be obtained with current algorithmic approaches for the RoA formulation.

1.4.1 Hybridization

Research in hybridization of metaheuristics and other optimization techniques has gathered considerable momentum recently. For example, the 2007 edition of the *Metaheuristics International Conference* (MIC) series included a session for hybrid methods; in the previous edition (2005), 15 out of the 150 papers presented were classified by the organizers as related to hybridizations with exact methods.

In their recent overview of metaheuristics in combinatorial optimization, Blum and Roli [7] present a categorization of hybridization techniques, and make the case for more research into this kind of approach. They also suggest that there are 'subtasks' in the search process where some metaheuristics perform better than others. In the same sense, is the view of the author that implementing an optimization technique 'from the book' is only necessary when the aim of the research is to evaluate that specific technique. Instead, when the objective is to solve a given optimization problem, doing it through the canonical application of an existing technique is likely to limit the quality of results achievable.

Specifically in the case of the driver scheduling problem, our survey of the literature in Chapter 3 suggests no evidence that a particular technique or metaheuristic is clearly best suited for solving the problem. At the same time, there is an extensive body of work in solving the TDS problem in the RoA formulation, particularly using mathematical approaches based on the set-covering formulation. Even if these approaches do not work well when applied to the WRO formulation, from a methodological point of view it makes sense to make use of or adapt these tools for the new formulation. Hybridization, as a form of composition, can be a powerful technique to re-purpose the work on the driver scheduling problem over the RoA formulation to account for WROs.

From the observations above, we derive a set of criteria that guide the research conducted in this thesis:

- we do not limit ourselves to the canonical application of one specific technique or metaheuristic; rather, we use and combine elements of existing techniques as and when deemed necessary

- we emphasize the reuse of existing research in the driver scheduling problem;

we give special consideration to hybridization as a means of integrating existing techniques with new proposals

- hybridization works both at the design and implementation levels – when feasible, we interface with existing software packages, rather than re-implementing their functionality

- we note that in order to use hybridization effectively it is essential to understand the motivations, benefits and limitations of each technique or tool

1.5 Organization of the Thesis

We start our thesis by first describing the train driver scheduling problem in detail, presenting WROs more thoroughly and discussing modeling issues associated with them. We then construct a minimal instance of the problem where it is easy to show that the optimal solution involves relieving at points unavailable to a RoA formulation. We also study the structural differences between the usual formulation and the one with WROs, and examine the implications of these differences. All of this is done in Chapter 2.

In Chapter 3 we review the existing literature on the driver scheduling problem, both with and without WROs. We also present an overview of certain optimization techniques that we use as a basis of our research in this thesis. We look at the challenges they set to solve, and compare them with the problems we face when exploiting WROs. By doing this matching, we can then reinterpret the solutions they propose in the context of our problem; we extract valuable ideas that guide our developments throughout the rest of the thesis.

In Chapters 4 to 8 we present original approaches to solving the problem of driver scheduling with WROs. In Chapter 4 we present a first stab at exploiting WROs by using the popular generate-and-select approach to solve the driver scheduling problem without WROs as a starting point to a local search over a WRO formulation. Experiments with this approach suggest that better schedules can be obtained in practice by considering WROs; however, those improvements are limited. We also look at the potential of iterating between generate-and-select and local search phases.

In Chapter 5 we take a very different approach to exploiting WROs, based on the observation that most scheduling constraints can be used to determine strict intervals for the start or end of a spell or shift. Given an instance of the scheduling problem, these limits can be exploited to heuristically reduce the set of WROs to be considered, to a point where generate-and-select approaches can be used on the resulting model. Experiments with this technique show significant improvements on real-life instances.

We then return to study the use of a hybridized generate-and-select and local search approach, with the aim of making the local search phase more effective. Experience on earlier chapters leads us to consider relaxing the constraint that all intermediate solutions in the search must be feasible. This in turn introduces the problem of comparing infeasible solutions among themselves, and against feasible solutions. We tackle these issues in Chapter 6, where we propose and explore a new methodology to cost schedules containing infeasible shifts, based on repairing the infeasible schedule into a feasible one, and returning the cost of this feasible schedule. At the same time, our experiments in Chapters 4 and 5 suggest that the initial proposals in Chapter 4 make limited use of domain knowledge. In Chapter 7 we look into exploiting domain knowledge in the form of reducing or eliminating perceived local inefficiencies in the current solution.

In Chapter 8 we consolidate several of the tools developed in earlier chapters into a local search framework that takes the starting solution using a generate-and-select approach on a relief-on-arrival formulation, and then iterates between exploring the spaces of feasible and infeasible solutions. Switching from feasible to infeasible solution exploration is triggered by the search reaching a local optimum, and the execution of an inefficiency-correcting move. We conduct experiments on this proposal, and also suggest and test variations including the use of a reduced WRO formulation derived from the work in Chapter 5, both during the generate-and-select phase and/or during the local search phase, and starting the search from an empty schedule.

We conclude the thesis in Chapter 9 with a summary of results, and a discussion of open issues and further research questions.

Chapter 2

The Train Driver Scheduling Problem

2.1 Description of the Problem

In this section we describe the train driver scheduling problem in more detail. We do so by first defining the inputs to the problem, then discussing the cost function, and finally describing the constraints most commonly enforced in practice, and where they arise from.

2.1.1 Inputs

Vehicle schedule, relief opportunities The main output of the vehicle scheduling phase, a vehicle schedule is usually represented as a set of *vehicle blocks*, each of which indicates the trips to be made by a given vehicle over the day, along with other operations such as fueling, cleaning, preparation (putting the train in a state that a driver can start driving it) and disposal (setting the train in a state that is safe for leaving unattended, when finishing the day). Unless specified, all trips have to be covered by a driver; some of the other operations, such as preparation and

disposal, are also usually carried out by drivers, before driving or leaving the train.

Figure 2.1 shows a schematic representation of a vehicle block. Vehicle blocks are annotated with *relief opportunities* and *windows of relief opportunities*, which –as discussed in Chapter 1– are (time, location) points during the operation of the vehicle where it is feasible to change or *relieve* drivers. We call the stretch of driving work between two consecutive relief opportunities in a vehicle block a *piece of work*. Most if not all relief opportunities arise either when a train stops at a station, or when a train is in a depot. Not all train stops are relief opportunities; some will be only used for the purposes of passenger-travel of drivers, and some will not be used at all for driver scheduling. Relief opportunities play a key role in driver scheduling. Throughout this thesis, we will study how relief opportunities can be better modelled and exploited in the context of driver scheduling.

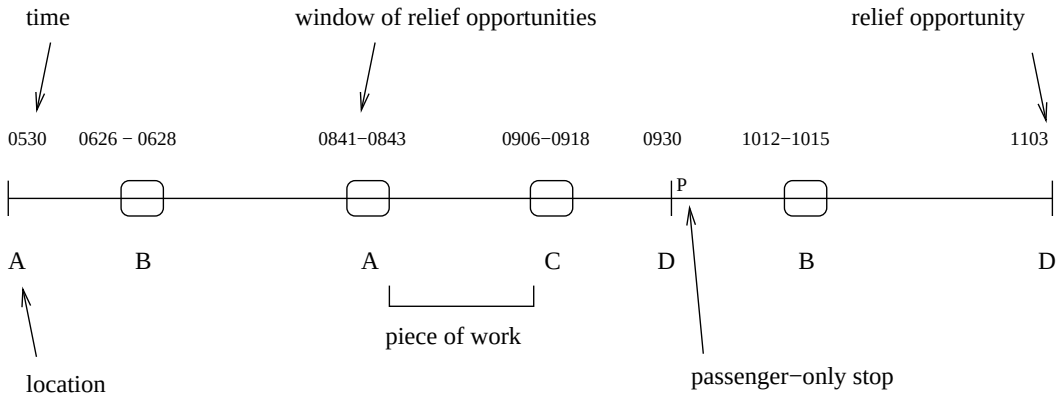

Figure 2.1: Schematic representation of a vehicle block.

Labour agreement rules A legal schedule must satisfy labour agreement rules that are usually determined by local practice, as agreed between the TOC and the unions, as well as government regulations. We describe the constraints derived from labour rules in detail in Section 2.1.3.

Route and traction knowledge Complex operations may be such that their vehicle fleets contain more than one vehicle or *traction* type. Different rules may apply to different traction types, including restrictions on which drivers can be assigned to each type; we call this extra information *traction knowledge*. Similarly,

drivers may be allowed to drive only those sections of the network they are trained to operate on, and have maintained a minimum frequency of driving over them; we call this *route knowledge*. Often, TOCs and their schedulers find it useful to define associations between shift 'classes' and routes / traction types, such that only drivers of specific classes will be allowed to drive certain routes / traction types; in this case, it can be useful to think of traction and route knowledge input as represented by binary matrices, with one row per shift type and one column per traction type or route, with a 1 in position i, j indicating that a shift of type i can be assigned to cover route or traction j. Limits on the number of shifts of each type in a schedule can be established based on the skills of the staff available to the TOC (taking this approach to the extreme, each real driver in the TOC could be associated to an individual shift type, but this is usually not feasible because the workforce is not fixed).

2.1.2 Output and Objectives

Structure of a schedule The main output of the driver scheduling phase is a *driver schedule*, which is a set of driver shifts. Figure 2.2 shows a schematic representation of a schedule of four shifts, covering a vehicle schedule of three vehicles. Each shift is usually described as a sequence of 1 or more *spells*, where each spell is an uninterrupted sequence of work on the same vehicle. In general, each shift will be composed of not more than four or five spells, although in some cases it may be useful to concentrate as much of the preparation and disposal operations to a few drivers, whose shifts will then include many more spells than usual.

For a schedule to be a valid solution to the problem, all pieces of work must be covered by at least one driver. Covering of a piece by more than one driver is called *overcovering*. Although overcover may not be desirable in itself, overcover pieces do not make a schedule infeasible (since given any overcovered piece of work, all but one driver can always be assigned as passengers).

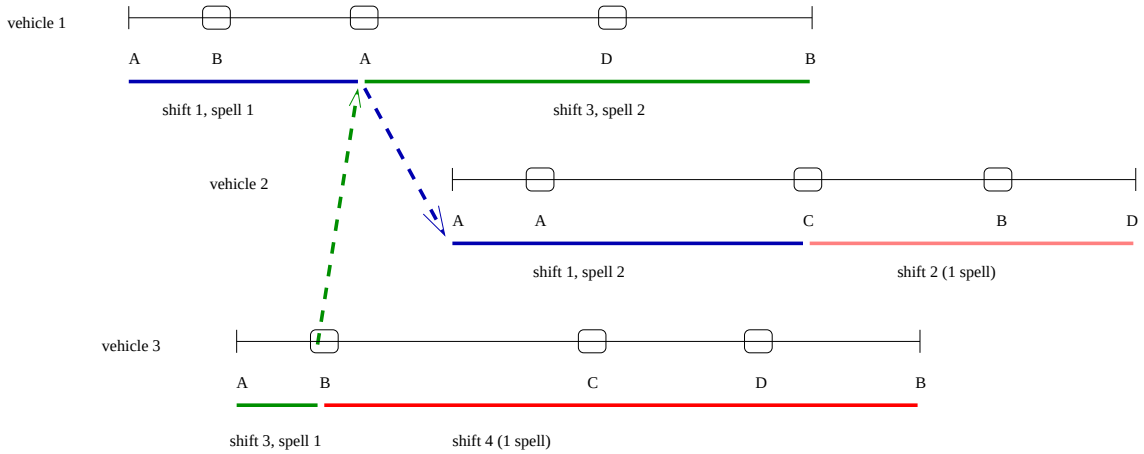

Figure 2.2: Schematic representation of a four-shift driver schedule.

Objective function The most commonly used indicators of the cost of a schedule $\mathcal{S}$ ($cost(\mathcal{S})$) with shifts $s_1, \ldots s_k$ are

(a) the number of shifts in $\mathcal{S}$:

$$cost(\mathcal{S}) = |\mathcal{S}| = k \tag{2.1}$$

(b) the sum of individual shift cost $c(s)$ for every shift s in $\mathcal{S}$:

$$cost(\mathcal{S}) = \sum_{i=1}^{k} c(s_i) \tag{2.2}$$

(c) a combination of the two above:

$$cost(\mathcal{S}) = W_1 \sum_{i=1}^{k} c(s_i) + W_2|\mathcal{S}| \tag{2.3}$$

where W_1 and W_2 are fixed weighting constants.

In this context, the objective of the driver scheduling problem is to build a feasible schedule $\mathcal{S}$ (i.e. one that meets all the constraints) that minimizes $cost(\mathcal{S})$.

Notice that while the cost $c(s)$ will usually be driven by the overall amount of hours worked or paid to the driver of shift s, it may also include other elements such as penalties for undesirable attributes in s (e.g. a shift requiring a taxi connection between two consecutive spells). The same also applies to the overall schedule cost, which might include schedule-level penalties (for example, for a schedule requiring the opening of a specific canteen for mealbreak purposes).

In some approaches to solving the driver scheduling problem, the cost function is augmented with other terms, for example to penalize certain constraints when violated, or to force certain properties on the solutions (e.g. no overcover). This is done with the aim of helping the algorithms find better solutions, or find them more quickly. However, these are part of the solution to the problem, rather than its description, and as such will not be covered here.

On the effect of the choice of cost function on the optimization problem When the cost function in use is based on the size (i.e. number of shifts) of the schedule, the set of possible values for the function is quite limited; for example, a daily schedule for a UK operator usually contains between 50 and 300 shifts. We would expect a competitive search algorithm for the problem to be within 10–15% of the optimal schedule size soon after starting the search. This means that the number of possible function values throughout the crucial part of the search is extremely limited – if the optimal schedule is expected to be of around 100 shifts, an optimization is expected to spend most of its time evaluating solutions whose cost is in a range of only about 10 distinct values, and even fewer when close to the optimum. This creates a real problem for any algorithm that is based on iteratively improving a solution, since it is very difficult to differentiate solutions in terms of their cost.

Common ways of tackling this problem in optimization include 'smoothing' the cost function by augmenting it with other (artificial) measures of the quality of a solution, which are designed to be smooth. However, it must be noted that these extra terms are not directed by a need of the user of the solution, and hence may in principle even prevent certain good-quality solutions from being found; for example, Equation 2.2 is not a good proxy for Equation 2.1, as e.g. shifts with a short spread-over tend to be inefficient for Equation 2.1 but are generally useful when optimizing on Equation 2.2. Fortunately, most of the times the TOC will be interested in a combination of schedule size and total shift cost, which allows search algorithms to work on smoother landscapes.

Most experiments in this thesis will be conducted on the cost function in Equation 2.3, which combines both measures. Some experiments in Chapter 8 will be

carried on the objective function in Equation 2.2, i.e. the sum of shift costs only.

Schedule Robustness

In the last few years, and particularly in the UK, the cost of a train driver schedule is increasingly being assessed by its *robustness*, which can be defined as the capability of a schedule to withstand operational disruption. Disruptions mainly take the form of delays/cancellations of train services (e.g. a train not available at the time defined in the schedule), and delays/cancellations in driver resources (e.g. a driver not available to drive a train unit at the time defined in the schedule). Causes for these delays are multiple, from signaling problems that affect the execution of the vehicle schedule, to drivers being off sick.

Alterations to the vehicle or driver schedule can feed on themselves (and on each other), causing more alterations on later stages of the execution of the schedules, triggering what is known as a *cascading of delays* – a situation very well known by rail commuters in this country. Therefore, one of the main ways of measuring robustness in a static driver (or vehicle) schedule is by evaluating or estimating its capacity to absorb delays and avoid their propagation. As of today, research in the area is still at the point of determining which measures of robustness are most suitable, both in terms of their correlation to actual robust schedules as perceived by operators and/or the general public, and in terms of their applicability to existing computational approaches to solve these problems. As an example, Yen and Birge [86] devise a stochastic airline crew scheduling model along with a solution methodology for integrating disruptions into the evaluation of crew schedules. We introduce specific measures of robustness for train driver scheduling in Section 4.2.2. Other models for assessing disruptions appear in the context of online re-scheduling, e.g. Walker et al. [78], and it is likely that these can also be adapted to assess static schedules during a long-term crew scheduling exercise.

Multi-objective optimization

Most optimization problems are such that there is a number of objectives that should ideally be optimized simultaneously. For example, the cost function formulation in Equation 2.3 specifies two objectives (total cost and schedule size). Similarly,

TOCs are increasingly faced with the usually conflicting objectives of minimizing the operating cost the schedule, in terms of man-hours paid, while maximizing its robustness. There is considerable work in the literature on transport scheduling as a multi-objective optimization problem, although mostly within airline operations; for example, Ageeva [2] discusses the tradeoff between robustness and optimality in the context of airline (vehicle) scheduling, and proposes methods to incorporate robustness into the model. Ehrgott and Ryan [23] present a (cost, robustness) bicriteria airline crew scheduling model, and generate Pareto-optimal crew schedules on their formulation.

In this thesis we will mostly restrict ourselves to single-objective formulations of the problem. When dealing with more than one objective simultaneously, we will either use a weighted formulation like that of Equation 2.3, or rely on a sequential optimization on the different objectives (e.g. optimize schedule size first, then total man-hours).

2.1.3 Constraints

In the research community in combinatorial optimization, constraints in the problem of train driver scheduling are generally regarded as very difficult to satisfy. Adding constraints to a problem will usually make the space of feasible solutions smaller. However, algorithms that rely on iterative improvement, and in particular local search algorithms, might find it harder to generate feasible solutions in the neighbourhood of the current solution. Throughout this thesis we will discuss in more detail the effect of constraints on local search approaches for the train driver scheduling problem.

Shift-level Constraints

Individual shifts are subject to a number of constraints. The most basic constraint derives from the recognition that the driver is a human being:

- a driver cannot be assigned to cover more than one piece of work simultaneously

Many constraints on individual shifts arise from the labour agreement; these include

- The restriction that a driver must start and end (i.e. sign on and sign off) its daily shift at the same depot; there are some exceptions to this rule, e.g. in the now-finished `GNER` operations on the East Coast Main Line, where drivers were sometimes required to do overnight stays in a city far away from the sign-on point.

- The *spreadover*, which measures the time elapsed between sign-on and sign-off in a shift; for example, a full-time shift may have a maximum spreadover of 9 hours, while for a part-time shift it might be 5 hours instead.

- The *total working time*, which may differ from the spreadover if not all time in a shift (e.g. mealbreaks or non-driving time) is classified as working time.

- The *spell length*, which controls the amount of continuous driving time; for example, a driver about to reach the maximum spell length must either finish its shift, or take a break in driving.

- *PNBs*: constraints on spell length give rise to periods or rest between spells; these are usually called *PNBs* (Physical Needs Break!) or *mealbreaks*; constraints in the labour agreement may control their number, length and location or separation within a shift.

- The *joinup time*, which is the time elapsing between the driver leaving a train and taking the next train (in the event that no PNB or mealbreak takes place); minimum joinup times are sometimes enforced to add robustness to the schedule.

- *Minimum travel times* between particular pairs of locations; these may also be used to increase robustness

- Restrictions on *sign-on and sign-off* times (e.g. through valid intervals for sign-on and sign-off). These are usually associated with particular shift types, and may be used to control the distribution of shifts in a schedule throughout the day.

Other constraints include route and traction knowledge, as discussed in 2.1.1. In the same way as route and traction knowledge, all shift-level constraints related to the

labour agreement can in principle be fine-grained, by associating different values to different shift types. For example, a shift type 'diesel' (which would only be able to drive diesel traction types) could eventually be assigned a shorter maximum spreadover to compensate for the perceived extra effort involved in driving a diesel engine, as opposed to driving an electric one.

Schedule-level Constraints

Apart from constraints that govern the validity of individual shifts, schedules will also usually be subject to constraints that involve the set of shifts as a whole. Examples of usual schedule-level constraints are:

- capping of the number of shifts belonging to a specific depot; this constraint may be determined by patterns in geographical availability of drivers, or may respond to restrictions in infrastructure (e.g. the size of a depot canteen)
- composition of the schedule in terms of full-time vs. part-time shifts (e.g. no more than 30% part-time shifts)
- maximum average shift length per depot

2.1.4 Short-term Scheduling and On-line Re-Scheduling

In this thesis we concentrate on long-term planning of transport operations: schedules obtained through these methods are expected to be available usually months before their actual implementation. However, transport planning in general and driver scheduling in particular are also subject to shorter-term requirements. In crew scheduling, these are frequently categorized into short-term scheduling and online re-scheduling.

Short-term scheduling is necessary –at least in UK railway operations– in response to either engineering work in the rail infrastructure, special events (e.g. the 2005 G8 Summit in Gleneagles), or major disruptions due to e.g. flooding. Although details are different in each case, in general short-term planning involves considerable short-term alterations to service timetables, vehicle schedules and driver schedules. The objectives in short-term scheduling are usually less related to cost, and more to

minimizing (as much as possible) changes to the existing assignments (particularly in the case of timetables). Hence the cost function has to be adapted to incorporate measures of change in relation to existing schedules. The amount of time available to produce the new timetables and schedules will obviously depend on the causes, and it will usually range from a couple of months (pre-planned engineering work) to becoming online re-scheduling, e.g. when flooding occurs. Balancing similarity between previous and new schedules with efficiency is a delicate issue. Kwan et al. [54] describe a 'minimum-change' approach, where shifts in a pre-existing schedule are added to the set of candidate shifts with a cost of zero, so that they will be artificially favoured by the selection phase.

Online re-scheduling occurs when there is disruption to the execution of the pre-planned daily schedule, e.g. due to delays, signalling failures, or a driver calling in sick. As with short-term scheduling, re-scheduling can involve service timetables, and vehicle and crew schedules. The main difference between short-term scheduling and online re-scheduling, apart from the timeframe for obtaining new solutions, is the quality and nature of information available: while in short-term scheduling information is likely to be stable and well-known, online re-scheduling often suffers from unreliable information, which is also short-lived in nature as the disruption progresses or spreads across the network. As with short-term scheduling, minimizing change is important. Walker et al. [78] discuss models and methods for real-time recovery from disruptions which generate alterations to the train timetable and crew schedules simultaneously.

2.1.5 Integrated Vehicle and Crew Scheduling

In integrated transport scheduling, two or more consecutive stages in operations planning (Figure 1.1) are solved together as a single problem. Within this area, recent research efforts have concentrated on integrating vehicle and crew scheduling, mostly in the airline and bus industries. The main motivation for solving these stages as a single problem is to obtain increased efficiency. In his PhD thesis [45], Huisman states that this increased efficiency is due to the fact that crew costs dominate vehicle costs, and that the feasibility of a crew schedule is much more restricted than that of a vehicle schedule. It is worth noting that an integrated

approach may however be more difficult to implement for a TOC since it involves negotiating two sets of decisions (vehicle and crew assignments) simultaneously with most likely independent entities.

Because most of the recent work in integration of vehicle and crew scheduling is based on column-generation approaches, the reader is referred to Section 3.1.5 for an overview of related work.

2.2 Modeling of WROs

2.2.1 Attended and Unattended WROs

When a train stops at a station, it is sometimes possible to leave the train unattended (i.e. unmanned); we call such a WRO an *unattended WRO*. This can be exploited to make the schedule more efficient –especially if the train stops for a long period of time– since the work to be covered can be reduced by not covering the train with a driver while the train is stopped.

In order to leave a train unattended, drivers must perform a set of predefined actions to secure the train whilst unattended and to start it up again shortly before the train departs again. These operations take a non-negligible amount of time to execute; for example, one UK operator allows drivers two minutes when leaving a train ('immobilizing') and five minutes when resuming driving ('mobilizing'). This means that the choice of leaving a train unattended is only feasible when the stop is long enough, e.g. more than seven minutes in this case (in many cases these mobilization/immobilization times are longer).

The scheduling system we will compare to and integrate with on this thesis, `TrainTRACS` (Section 3.1.7) allows users to specify unattended WROs; the system can then use an unattended WRO u in one of two ways:

1. u is not used to relieve drivers. In this case, the same driver covers all of u and both pieces of work at the sides of u; the train remains attended during the stop (and no mobilization or immobilization operation will be carried on u during the execution of the schedule).

2. A relief takes place in u, and the train remains unattended during the stop.

The driver leaving the train immobilizes it before leaving, and the driver relieving him mobilizes the train before departure.

While this model is sufficient to exploit WROs that can be left unattended in the sense of not covering the stop, it also omits a third option, namely that of leaving the train attended during the stop and relieving drivers at any given point within the stop. Note that relieving while leaving the train unattended forces the driver leaving the train to do so only after immobilizing, which could be too late to take another train later (and similarly the driver taking up the train is forced to arrive at the train in time to perform the mobilization). Since mobilization and immobilization are only required if the train is effectively left unattended, the restrictions on relieving times just described do not apply when the train is left attended, and therefore omitting the third option formally limits the schedules that can be generated.

Nevertheless, in this thesis we treat unattended WROs in the same way that `TrainTRACS` does, i.e. we will not consider the third option described above. We do so mostly because the two options supported by `TrainTRACS` are usually adequate enough for exploiting unattended WROs, and in any case the time needed to immobilize and mobilize a train usually makes unattended WROs a small fraction of all the stops. Hence we believe that restricting our study to attended WROs does not affect the results or applicability of our research significantly, while simplifying the models and implementation.

2.2.2 One-minute expansion of WROs

Many mathematical models for events that can occur within a time window $[t_s, t_e]$ use a variable t to represent the time at which the event occurs, then bound the value of t through two inequalities,

$$t_s \leq t \leq t_e$$

In general, TOCs in the UK use the minute as the smallest unit of time for scheduling purposes (although some work on 30-second units). Allowing for drivers to be relieved at a non-integer number of minutes (or half-minutes) would not then make a model more general, since in practice relief times obtained on this model would

have to be adjusted to minutes or half-minutes. Hence if t is specified at a minute level, the above formulation is equivalent to

$$t \in \{t_s, t_s + 1, \ldots, t_e\}$$

Although the two formulations describe the same sets of ROs, they can be seen to represent different 'views' on what WROs provide: the first one related to the concept of an interval of time where the driver can be relieved, the second one suggesting a discrete, contiguous set of relief times that can be used for relieving a driver. Throughout this thesis we will take the latter view.

Following from the view of a WRO as a set of contiguous relief times, a possible representation for a WRO at location l starting at time t_s and ending at time $t_e = t_s + k$ is that of a set of $k + 1$ relief opportunities of the type $< t_s + i, l >, i = 0, \ldots, k$. These ROs would delimit pieces of work that are one minute long, and have the same rules regarding covering by drivers than all other pieces of work. We call this formulation a one-minute expansion of WROs:

Definition 4 (1-minute WRO expansion)**.** We call the representation of a WRO w as a set of 1-minute-apart ROs a *1-minute expansion* of w. By extension, we call the formulation of a TDSW instance $\mathcal{I}$ where all WROs are represented through one-minute expansions a *1-minute expansion of* $\mathcal{I}$.

2.3 Solving the Problem of Train Driver Scheduling with WROs

In the remainder of this chapter, we look at the structure of the TDS and TDSW problems, and give an initial insight into how WROs may help in obtaining more efficient schedules, and why the methods that are designed for the RoA formulation may break down when used in a WRO formulation, e.g. by applying the same methods over a 1-minute expansion of WROs as described in the previous section.

2.3.1 An Example on the Limitations of the RoA Formulation

In this section, we show that considering WROs in the scheduling model may allow for strictly better solutions to be produced. We do so by building a minimal instance of the problem, where it is easy to see that the (unique) optimal solution with WROs is strictly better than the (unique) one on the RoA formulation.

The example is built using three vehicles. Figure 2.3 shows vehicle work for the three vehicles, and the optimal schedule when WROs are considered. Work for each of the vehicles v_1, v_2, v_3 is formed of two pieces of work; vehicles v_1 and v_3 contain WROs at 11.00–11.01 and 10.59–11.01 respectively.

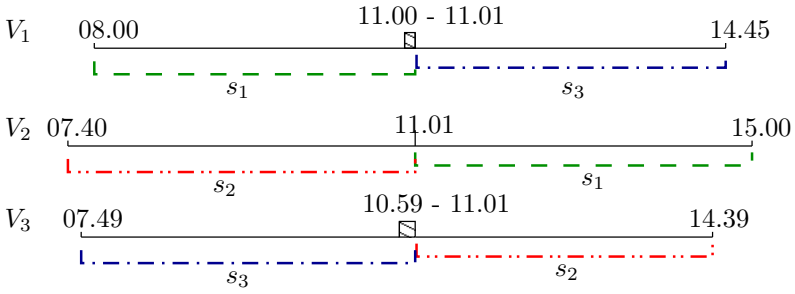

Figure 2.3: A 3-vehicle instance of the problem, with WROs.

Under the conditions that (a) maximum shift spreadover[1] is seven hours, and (b) travel time between relief points is zero, the optimal schedules in the 1-minute WRO expansion and the RoA formulations contain 3 and 4 shifts respectively. This can be shown by observing that the total work on v_2 is over 7h and therefore to achieve a 3-shift schedule, work on v_2 must be split up (i.e. the driver on v_2 must be relieved at 11.01). With RoA, the ROs are 11.00, 11.01, 10.59 on v_1, v_2 and v_3 respectively. The driver on v_2 could be relieved by either that on v_1 or v_3, but then he would be too late to start the second portion of work on either v_1 or v_3 and therefore an extra driver would be needed. It is also easy to see that if time is discretized in 1-minute intervals, then the optimal schedule for this instance on each model is unique. The unique optimal schedule on the 1-minute expansion is:

[1] the maximum total shift time from sign-on to sign-off

S1: v_1 08.00 – 11.01
v_2 11.01 – 15.00

S2: v_2 07.40 – 11.01
v_3 11.01 – 14.39

S3: v_3 07.49 – 11.01
v_1 11.01 – 14.45

while the unique optimal schedule for the RoA formulation has the following four shifts:

S1: v_1 08.00 – 14.45

S2: v_2 07.40 – 11.01

S3: v_2 11.01 – 15.00

S4: v_3 07.49 – 14.39

This minimal example proves that –at least on some instances– the solution space for the RoA formulation will not contain *any* of the optimal solutions for the original scheduling problem, and so any algorithm that searches this space will unavoidably lead to sub-optimal solutions.

We built this example so that the optimal schedule for each model is unique; we also did it so that both solutions differ in all of their shifts (and in fact only share a single spell). Moreover, this approach can be extended to create examples involving more vehicles and requiring more shifts on the optimal solutions, without losing any of the properties described. All of this would suggest that the gains available from a full consideration of WROs may require a very extensive transformation, which would be difficult to obtain from solving local, independent subproblems.

A considerable part of this thesis will be devoted to studying methods for building an optimal solution on a WRO formulation by starting from an optimal solution for the RoA formulation. In this context, the example above –and the fact that it can be extended to bigger instances– suggests that the problem of transforming an optimal solution on an approximated model to an optimal solution for the WRO

formulation may effectively be as hard as the full driver scheduling problem for WRO formulations.

2.3.2 Structural Differences between the RoA and WRO Formulations

Establishing the existence of differences in structure between two formulations of a problem is very important as this can be used to characterize them and relate them, helping answer questions as essential as whether one formulation is a generalization of the other. In turn, this helps evaluate whether existing algorithms for a formulation can be reused for a different formulation, and if so whether any adaptation is needed to reuse them. In this section we study the differences in structure between the sets of ROs derived from a RoA formulation and those obtained when considering WROs. We will consider a 'real-world' problem instance $\mathcal{I}_R$, and two sets of ROs R_{roa} and R_{wro} implicit on the RoA and WRO formulations for $\mathcal{I}_R$, when using a 1-minute expansion of the WRO formulation as discussed in Section 2.2.2.

A first immediate observation is that $|R_{roa}| \leq |R_{wro}|$. Moreover, $|R_{roa}| = |R_{wro}|$ if an only if $\mathcal{I}_R$ contains no WROs. This holds for any instance of the driver scheduling problem. A look at real-life instances of the problem (e.g. those in Table 5.1) shows that usually $|R_{wro}|$ is more than double $|R_{roa}|$. This in principle already separates the two formulations in terms of the feasibility of solving the problem in practice, since the size of the solution space is in general exponential in the number of relief opportunities.

In our opinion, the second crucial difference between the sets R_{roa} and R_{wro} is the way in which ROs within those sets are distributed in time and space. It is important to consider this aspect carefully, as the implications are central to any attempt at solving the driver scheduling problem with WROs. We claim that the distribution of relief opportunities is completely different in R_{roa} and R_{wro}, because of the way new ROs in R_{wro} cluster together, to the point where most algorithms designed to solve the driver scheduling problem in the RoA formulation will not be suited to deal with the extension to consider WROs.

To investigate this, we took a real-life instance $\mathcal{I}$ from *Wessex Trains* and

computed first the time distances between consecutive relief opportunities in a vehicle, for all vehicle work specified in $\mathcal{I}$, under the relief-on-arrival formulation. $|R_{roa}| = 809$. Figure 2.4 shows a histogram of these distances, at 5-minute intervals between 0 and 120 minutes (which account for more than 96% of all distances).

We also run a curve-fitting package (available online at `www.zunzun.com`) that fitted the data from the histogram using least-squares, for more than 100 different functions commonly used in experimental sciences. The best fit was achieved with a Gaussian function, i.e. a function of the form

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}} \tag{2.4}$$

where a defines the height of the Gaussian peak, b controls the position (x-value) of the center of the peak, and c is related to the concentration of the distribution around the peak. In this fit, $b = 54m7s$ (54 minutes and 7 seconds); with distances above 120 minutes considered, the mean sample value is $56m5s$ instead.

Although it is beyond the scope of this thesis to analyse the 'real-world' motivations for the distribution of train stops, we can argue that a Normal distribution of distances between relief opportunities is to be expected: time distances in the RoA formulation are very strongly correlated to physical distances between stations in the underlying rail network; in turn, train stations (and hence relief opportunities) would often be 'evenly' spaced, the spacing being subject to possibly Gaussian noise in the exact spatial distribution of population centres / town centres. The existence of peaks that do not correspond to the peak of the Gaussian fit could be explained by the fact that rail networks will usually have an uneven concentration of services throughout their physical network (since some origin-destination pairs have more passenger demand than others, for example), and this translates into some pairs of consecutive trips appearing more frequently than others in the specification of vehicle work. As an aside, it is worth noting that the distance between two consecutive relief opportunities is also the length of the piece of work delimited by them.

We repeated the analysis above for the WRO formulation. For this test, unattended WROs were also expanded, resulting in $|R_{wro}| = 6854$ ($|R_{wro}| \approx 8.47|R_{roa}|$). The corresponding histogram of distances between consecutive ROs is shown in Figure 2.5. In this case, more than 80% of the distances are smaller than 5 minutes; it

is clear that the data cannot be explained by a normal distribution. Moreover, there is no meaningful probability distribution that would explain the data by itself[2]. In our view, any explanation of this distribution would need to consider ROs arising from WROs separately from the ones arising from the relief-on-arrival formulation.

Discussion The experiments above make it clear that introducing ROs within WROs into the problem changes the distribution of relief opportunities dramatically. Throughout this thesis we will see that this difference is at the center of the need for algorithms that tackle ROs within WROs explicitly, as opposed to considering them the same as non-windowed ROs in the model.

It is interesting to see that, as far as we know, none of the literature on TDS studies the distribution of lengths of pieces of work in real-life instances explicitly. However, it can be argued that when no explicit assumption is made on the distribution of a random variable, the implicit assumption is that the distribution is normal. Hence, we could say that the assumptions implicit in the literature would match our empirical observation that these lengths actually appear to be distributed normally.

On the other hand, any approach to solving the driver scheduling problem on the WRO formulation that treats ROs within WROs as any other RO, and considers the (1-min) piece between two consecutive ROs within WROs a piece of work like any other, would be wrong to assume (implicitly or explicitly) that the distribution of piece lengths is normal. As an introduction to the issue from the point of view of algorithms, consider a hypothetical algorithm to solve the driver scheduling problem on the RoA formulation. This algorithm works by first constructing a feasible solution, and then iteratively selecting a piece of work randomly from the set $\mathcal{P}$ of all pieces in the problem instance, removing it from the shift that is currently covering it (assuming no overcover), and evaluating re-assigning it to some other shift in the schedule. In the *Wessex* problem instance described earlier, this would mean that on each 'move' this algorithm would be re-assigning on average approximately 56 minutes of work on each move (the amount of work moved having a Gaussian distribution around that value). Consider now a modification of this algorithm for

[2]A Gaussian fit is still relatively descriptive of the sample data if all distances below 5 minutes are removed, although the least-squares fit fails to explain the second small peak around 65 minutes

the WRO formulation, which consists of adding all ROs within WROs as new ROs, and considering the space between two consecutive ROs within a WRO a separate piece of work. If we apply the same algorithm, 8 out of every 10 moves would be re-assigning less than 5 minutes of work.

Based on the analysis presented in this section, and the study of the literature presented in Chapter 3, and more specifically in Section 3.2, the proposals for tackling the driver scheduling problem with WROs in this thesis will in most cases (and when under our control) consider relief opportunities arising from WROs –and the work in between them– separately from relief opportunities present in the relief-on-arrival formulation.

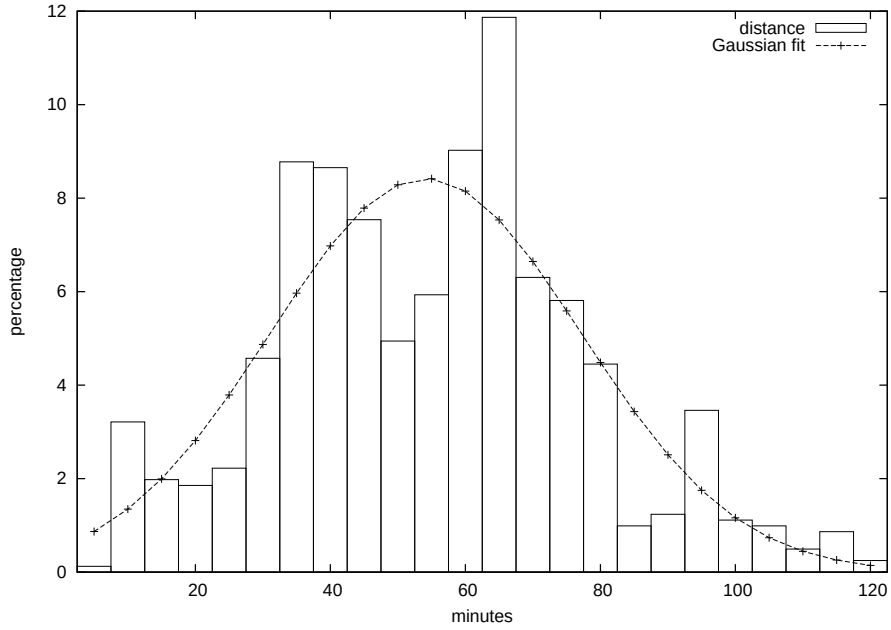

Figure 2.4: Histogram of time differences between consecutive relief opportunities in the relief-on-arrival model (Wessex Trains), along with a fitted Gaussian function.

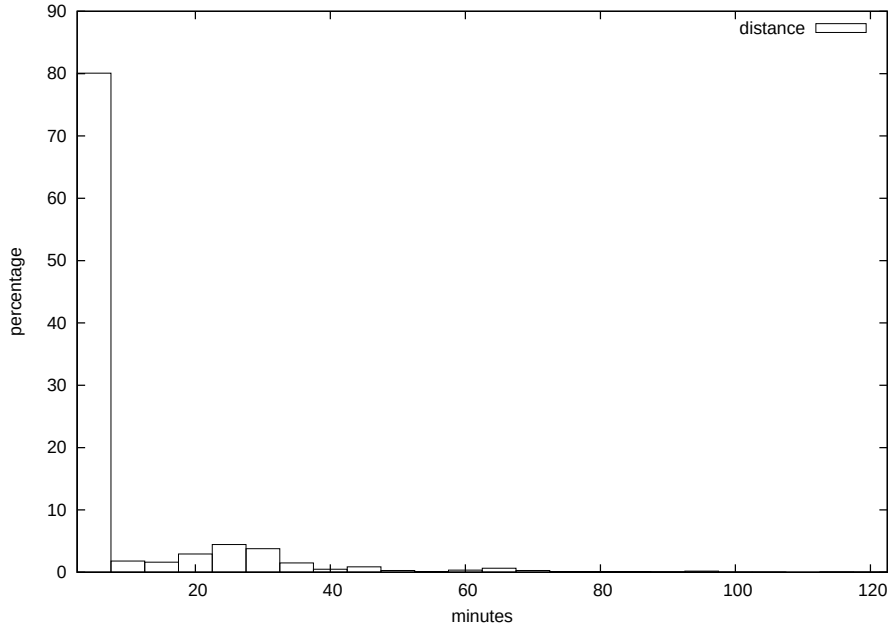

Figure 2.5: Histogram of time differences between consecutive relief opportunities when considering WROs as sets of 1-minute-apart ROs (Wessex Trains).

Chapter 3

Literature Review

3.1 Solving the Crew Scheduling Problem

Crew scheduling in public transport has been the subject of considerable research in the last forty years. A number of conference series are directly or indirectly related to the study of the problem. Chief among them is the series of *International Conferences on Computer-Aided Scheduling of Public Transport* (CASPT), which started in 1975 and is held roughly every three years; selected papers from these are available in [42, 77, 81, 15, 22, 16, 68, 82]. Other conference series, such as *PATAT* [11, 9, 10] and *MIC* [46] usually include tracks specific to transport scheduling.

In this chapter we introduce the main computational approaches to solving the public transport crew scheduling problem. This is presented from a historic perspective, highlighting past and current research trends. Other surveys in the literature include Wren and Rousseau [84], who cover the bus driver scheduling problem, including commercial packages; this however only spans work up to 1993. Kwan [55] presents a more recent review on bus and train driver scheduling, concentrating on current algorithmic approaches; Gopalakrishnan and Johnson [38] provide a review on airline crew scheduling.

Notice that, unless specified otherwise, all work described in this Section tackles

the problem in the relief-on-arrival formulation, i.e. the TDS problem, rather than TDSW.

3.1.1 Heuristics

According to Wren and Rousseau [84], most computerized crew scheduling systems up to the late 1970's were based on heuristics (many of which were attempts to mimic a rule used by human schedulers), their suggestion being that although researchers knew that the problem could be formulated with integer linear programming, the technology at that point in time didn't allow for problems of reasonable size to be solved with that kind of approach.

As mathematical programming took over, the use of heuristics subsided, although they still played a role in complementing mathematical approaches, or helping reduce the problem sizes so that the mathematical approach could find a solution in a reasonable amount of time. This is still true of many academic and commercial crew scheduling tools – we will see that no mathematical approach presented so far (including those involving column generation) has been able to solve large-size, real-life crew scheduling problem instances without incorporating heuristics on at least one of the components of the system.

3.1.2 Generate and Select

The crew scheduling problem, along with some related problems in transportation, have been successfully modelled using integer linear programming. A frequently cited unified framework for many routing and scheduling problems is proposed by Desaulniers et al. [20].

The most widely used approach to solving the problem in practice is the so-called *Generate-and-Select* (GaS) approach. In the first phase of this algorithm –the *generation* phase– a (usually very large) set $\mathfrak{C}$ of candidate shifts is built. These shifts are valid according to labour agreement rules, and cover all of the vehicle work (usually many times over). Additional constraints such as *minimum spell length* are frequently added to limit the size of $\mathfrak{C}$; note that these constraints are artificial, in the sense that violating the restrictions they impose would not necessarily make the

shifts invalid in terms of the labour agreement. In the *selection* phase, a minimal cost subset $S \subset \mathfrak{C}$ such that all vehicle work is covered is selected to form a schedule.

The Set Covering Formulation

The selection phase (for the formulation where the schedule cost is described by Equation 2.3) can be described mathematically using a set covering model:

$$\begin{array}{lll} \min & \sum_{j=1}^{n}(W_1 c_j + W_2)x_j & \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij}x_j \geq 1, & 1 \leq i \leq m \\ & x_j \in \{0,1\}, & 1 \leq j \leq n \end{array} \tag{3.1}$$

where $n = |\mathfrak{C}|$ is the number of candidate shifts, m is the number of pieces of work, x_j indicates whether shift j is selected ($x_j = 1$) or not ($x_j = 0$), c_j is the cost of shift j, and a_{ij} is 1 if and only if shift j covers the piece of work i. W_1 and W_2 are used to weight the different objectives (total cost and number of shifts).

Some domain-specific constraints, and in particular all schedule-level constraints, will appear as added constraints to the model above; for example, the number of shifts with drivers from a specific depot k can be capped to b_k by defining a vector $d^k, |d| = n$, such that $d^k_j = 1$ if and only if shift j belongs to depot k, and then adding the constraint $\sum_{j=1}^{n} d^k_j x_j \leq b_k$.

Notice that this formulation hides the structure of individual shifts – therefore, constraints governing the validity of individual shifts are not explicit in it. However, these constraints will determine the structure of all shifts in the pool, which is represented in the mathematical formulation through the matrix $\{a_{ij}\}$. Likewise, individual components of the cost of each shift s_j are encapsulated in the coefficient c_j. It is worth remembering that any computational approach that is strictly based on this model will not have access to the 'hidden' information described here.

An important aspect of this model is that it allows for pieces of work to be covered by more than one driver simultaneously; we call this *overcovering* a piece of work. This feature of the model may seem counter-intuitive, especially since many early approaches to driver scheduling actively worked on discouraging overcover. However, set partitioning is a more constrained problem than set covering, and it is generally regarded as more difficult to solve. Crucially, allowing for overcover is

not problematic from a modeling point of view, since when executing a schedule, a piece of work p that is covered by two or more drivers will have only one driver assigned to drive it, while the other driver(s) assigned to p will travel as passengers. In some cases, overcover work can be subtracted from the schedule cost by checking whether overlapping shifts can be modified to remove the overcover. Also, even when overcover is allowed on the model, it may still be penalized; for example, `TrainTRACS` generally adds a cost component equal to three times the amount of overcover to the total cost of a schedule.

It is worth noting that Garey and Johnson [34] have shown that the set covering problem is NP-hard in the number of sets, or shifts in this case. Worse, we show in Section 3.2.2 that the number of shifts n in $\mathfrak{C}$ can grow exponentially with the number of relief opportunities, resulting in an extremely hard problem to solve.

3.1.3 Constraint Programming

Public transport crew scheduling, and in particular train driver scheduling, are regarded to be highly constrained problems. Hence these problems seem naturally suited to be solved using so-called *constraint programming* approaches. A constraint satisfaction problem (CSP) consists of a set of variables $X = \{x_1, \ldots, x_n\}$, each of which can take a finite set D_i of possible values, and a set of constraints that restrict the values that the variables can take simultaneously. When using a CSP to solve an optimization problem, we are interested in finding an assignment of values to all variables, such that all constraints are satisfied, and the cost of the solution is optimal (or as close to optimal as possible). As with ILP problem formulations, CSP formulations are declarative, in the sense that they do not impose a particular computational method for solving the problem. However, authors such as Curtis et al. [14] suggest that the choice of modelling (as a constraint satisfaction problem) can greatly affect the performance of the algorithm.

Constraint programming approaches have been successfully applied to scheduling of crews in public transport. Curtis et al. [14] solve the problem of bus driver scheduling using a constraint programming approach based on the set partitioning formulation. The authors consider two alternative models of the set partitioning problem, and propose specific methods and reduction techniques to make the algo-

rithms more efficient; they also incorporate heuristics derived from analysing the solution for a LP relaxation of a set-covering formulation of the problem. Preliminary results suggest ILP-based systems can produce solutions faster and on larger problems, but it is hoped that constraint programming approaches will allow to add more constraints that are hard to model in an LP formulation.

More recently, De Silva [18] and Fahle et al. [25] incorporate constraint programming techniques in the context of solving a crew scheduling problem (bus and airline respectively) using column generation. As discussed later in Section 3.1.5, most column generation approaches to the crew scheduling problem solve the generation subproblem through a constrained shortest path formulation, but this has the drawback that many of the more complex rules arising in real life cannot be expressed entirely in that formulation, or can only be expressed at the expense of the subproblem becoming too difficult to solve in practice. As with Curtis before, the authors of both papers claim that using a constraint programming formulation instead allows them to model real-life constraints more fully.

3.1.4 Metaheuristics

Although there is no agreed consensus on the exact definition of a metaheuristic, we will take Blum and Roli's view [7], who say that a metaheuristic is

> a new kind of approximate algorithm [...] which tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space.

In the last twenty years or so research in metaheuristics and its application to scheduling problems has become very popular. Most of the major metaheuristics have been (and are still) adapted to and tested on the crew scheduling or rostering problem for either either airline, bus or train operations.

We present here a brief overview of how different metaheuristics have been applied to crew scheduling. We concentrate on the choice of solution representation, where a very wide range of proposals is exhibited. This is undoubtedly linked to the fact that metaheuristics impose very few constraints on the underlying representation (as opposed to e.g. linear programming). However, it must be noted that

some questions are only relevant to particular metaheuristics – for example, genetic algorithms seem to be much more sensitive to solution representation than other metaheuristics, in particular when the standard mutation and crossover operators are used. Where relevant, we also highlight how metaheuristics are modified to incorporate elements that are not part of the standard formulation. We refer the reader to Section 3.3 for a more comprehensive look at the key metaheuristics in the context of their potential application to problem of driver scheduling with WROs.

Tabu Search Cavique et al. [12] propose algorithms for a decision-support system for crew scheduling at Lisbon Underground. Initial solutions are built with a well-known problem-specific heuristic. In their first Tabu Search proposal, the next solution in the search is built by removing inefficient shifts from the current solution and replacing them with more efficient shifts (which the authors claim to be a form of 'strategic oscillation'). In their second proposal, moves do not operate on the solution space directly, but rather indirectly on the space of possible partitions of the set of vehicle blocks into pieces. A matching graph G is built over this partition, and finally a schedule is obtained by solving a maximum-cardinality matching problem over G. Moves modifying the partition are implemented as sequences of simple moves; these sequences are usually referred to as 'subgraph ejection chains' in Tabu Search literature. Shen and Kwan [70, 69] propose a tabu search approach for the TDSW problem; their approach is described in more detail in Section 3.2.1.

Simulated Annealing Emden-Weinert and Proksch [24] present a simulated annealing approach for airline crew pairing. Their application of simulated annealing is quite straightforward, although they introduce an extension where after the completion of each temperature level a deterministic local improvement phase is fired, exploring specific compound moves.

Evolutionary Computation Levine [58] presents a steady-state genetic algorithm (GA) for airline crew scheduling, where in each iteration the worst-ranked solutions (usually one or two) in the current set are replaced with new ones. A chromosome contains one bit per 'column' (i.e. candidate shift) in the set-covering formulation (each bit can then be directly associated with a 0-1 variable in the

set-covering formulation). Recently, Kotecha et al. [52] claim to have improved on the results from Levine on another GA proposal which uses a different, row-based encoding, and a cost-biased crossover.

Kwan [53] and Kwan et al. [57] study variants of GA where so-called 'combinatorial traits' (features of a solution that are deemed positive) are detected on elite solutions, and are carried forward to offsprings. Initial solutions are built with information derived from solving the LP relaxation of a set covering model, which provides with an initial list $\mathcal{L}$ of *preferred* shifts. A chromosome does not represent a full solution; instead, it contains one bit per preferred shift in $\mathcal{L}$. Their chromosome representation and crossover/mutation operators mean that in most cases the chromosomes will not describe a valid schedule; hence, a greedy repair heuristic stage is added. The repair heuristic presented in this work predates those introduced in our proposal in Chapter 6.

Current Research Research in metaheuristics for transportation scheduling is still very active. As we discuss in more detail in Section 3.3, there is currently a research trend in hybridizing search techniques in order to solve a combinatorial optimization problem. Among the presentations on the most recent CASPT conference (2006), Moz et al. [64] propose two evolutionary (meta)heuristics for a bi-objective formulation of the bus driver rostering problem, one of which integrates local search into an evolutionary framework. Similarly, Souai and Teghem [72] propose a hybridization of genetic algorithms and simulated annealing, where the latter is used to restore feasibility to the solutions generated by the genetic algorithm under their crossover and mutation operators.

Another promising development that can be linked to the area of metaheuristics is the so-called 'hyper-heuristic' approach introduced by Soubeiga et al. [13]. The key idea of hyper-heuristics is to provide a high-level, problem-independent framework to control the execution of lower-level, presumably very simple problem-specific heuristics. A key suggestion is the use of a 'choice function' that dynamically ranks the low-level heuristics during the search, ideally using only minimal information from the heuristics, such as improvements generated by each call to a specific heuristic. The approach has been applied to the bus driver scheduling problem

by Rattadilok and Kwan [67], using an extension of the choice-function approach where the controller dynamically selects a set of parameters to instantiate a low-level heuristic to be executed during the search.

3.1.5 Column Generation

Column generation (CG) is a very popular technique in the area of linear programming, and it targets problems where the number of variables ('columns') is too big to be considered explicitly, while the number of basic variables is expected to be very small in comparison. Lübbecke and Desroisiers [62] describe the approach comprehensively and give an up-to-date survey of applications.

Column generation works by decomposing the problem into a (restricted) master problem, which is the original problem restricted to a subset of the original variables, and a pricing subproblem where new variables with negative reduced cost are found to be added to the master problem. Both problems are solved iteratively in a loop, the master problem finding the optimal solution for the current (sub)set of variables, the pricing subproblem selecting new variables to add to the master problems based on updated reduced costs. This loop is in theory finished when there are no new variables with reduced cost to add to the master problem.

When solving the driver scheduling problem using the set covering formulation, it is easy to see that the vast majority of the x_j variables will be zero (i.e. non-basic) in all but extremely inefficient solutions. For example, a typical instance of the problem may result in 100,000 valid shifts, but its optimal solution contain 100 shifts in it (hence 99.9% of the x_j variables will be zero). Therefore, driver scheduling would fit the conditions under which CG can be useful. The master problem is still the set covering problem described in Section 3.1.2, but the set $\mathfrak{C}$ of shifts available to the set covering phase will change (grow) over time, and it is the task of the pricing subproblem to select new (negative reduced-cost) shifts to add to $\mathfrak{C}$.

Column generation is actively applied to the driving scheduling problem both in academic research and in commercial scheduling systems (such as `TrainTRACS` and `CrewOpt`). We briefly discuss some of these applications.

`TRACS II` In her PhD thesis [30], Fores investigates the use of CG techniques for the bus driver scheduling problem, using the bus driver scheduling system `TRACS II` as a starting point and benchmark. Her work aims principally at allowing systems like `TRACS II` to be able to (implicitly) consider bigger sets of candidate shifts. Her proposal is different from most current CG approaches in that, instead of resorting to CG in the context of solving an ILP, she uses CG during the construction of the optimal solution for the LP relaxation of the problem. The system runs a loop where it alternates between (re-)optimization and CG stages, until the relaxed LP solution is optimal over the original set of candidate shifts. Fores notes that some of the constraints handled in `TRACS II` mean that the usual network formulations for the pricing problem in set covering formulations cannot be applied, and the pricing problem would in some instances be NP-hard in itself. Hence, she extends her CG proposal by adding a number of heuristics to accelerate and bound execution times.

This CG strategy was later incorporated into `TRACS II` [32]. This system is the predecessor to `TrainTRACS`, the train driver scheduling system we will use or interact with throughout this thesis. `TrainTRACS` has inherited the CG phase from `TRACS II` virtually unmodified.

Integrated Vehicle and Crew Scheduling Within the area of integrated vehicle and crew scheduling, most of the research has concentrated in applying column-generation approaches to solve it. After the foundational paper by Desaulniers et al. [20] several authors (often working together) have made sustained progress in the problem sizes and the complexity of the constraints handled, particularly in the crew scheduling subproblem, which is the most complex part of the integration. Haase et al. [39] and Stojkovic et al. [74] introduce models for urban mass transit systems and airlines, respectively. Freling et al. [33] and Huisman [45] solve integrated bus scheduling problems. Recent PhD work by Weider [79] claims to improve solution quality and execution times in a number of instances solved by Huisman in [45], while also applying his algorithms on real-life instances of a German bus operator.

Embedding Heuristics within the Column Generation Framework

One of the main problems faced by column generation approaches for crew scheduling is that the column pricing problem, which is usually modelled as a constrained shortest path problem, is frequently NP-hard for models containing anything more than the simplest constraints. This means that researchers usually have to resort to limiting their models severely; for example, in his thesis [45] Huisman only considers crew constraints that are defined on an individual shift – rules which e.g. require that a (maximum/minimum) number (percentage) of the duties has certain properties cannot be taken into account. Weider claims to allow for more general constraints than Huisman, but his work is still restricted to bus operations, which are usually regarded as simpler than railways.

Common approaches to tackle NP-hard pricing subproblems include heuristics for the shortest path problem, and also adding many columns per pricing phase. An interesting approach is taken by Borndörfer et al. [8], who resort to a method they call 'callback', where they "*ignore the rule[s] in their pricing model, construct a pairing, and send it to a general rule verification oracle that either accepts or rejects the pairing*". This is interesting in that the use of such a mechanism seems to break with all the mathematical foundation of CG and its roots in mathematical programming. Moreover, we will see that we resort to a similar technique when generating neighbouring solutions in our local search proposals, where we rely on an external oracle (`TrainTRACS`'s `CHECKER`) to validate the new solutions generated during the search.

Criticism and Ongoing Work

Column generation for driver scheduling is a very active area of research. However, although one company (`GIRO`) reports to use CG in their widely-used `CrewOpt` scheduling system [17], to our knowledge there are no peer-reviewed reports on applications of CG for the bus or train driver scheduling problem that can fully handle the size and complexity of real-life instances within the pricing problem.

As an example of current research on this area, recent work by Steinzen et al. [73] proposes a time-space network representation for the shift generation subproblem,

which is claimed to result in reduced network sizes. However, no results are reported on the integration of this representation into the main driver scheduling problem, neither in terms of schedule sizes nor costs; also, although the authors claim that complex constraints are considered, they are not described in the paper.

3.1.6 Current Trends

Driver scheduling is a very active field of research. In this section we highlight what we consider two of the most interesting areas of development.

Increased Problem Size and Complexity

In the last decade, theoretical development, along with more powerful computer hardware, have triggered a major commercial takeover of computerized driver scheduling systems. This in turn has resulted in demands for systems to be able to solve bigger and more complex real-life problem instances, and take into consideration more sophisticated objectives and constraints.

An example of current work in this area is that by Kwan and Kwan [56]. In their previous system, `TRACS II`, a large problem would have been handled by first manually decomposing it into smaller subproblems and solving them separately, and later joining the individual solutions into an overall solution for the original problem, with possibly some final re-run to optimize on the quality of the end solution. However, this approach results in suboptimal solutions. Their new `PowerSolver` tool tackles the problem instead by initially solving a simplified version of the problem, where most relief opportunities are eliminated from the problem description. Based on this first solution, the system then iteratively explores more refined formulations of the original problem, by selectively reinstating parts of the set of relief opportunities that were initially left out and re-running the optimization on this augmented problem instance. Results obtained on big-size instances improve over results on earlier systems. In practice, all railway companies currently using `TrainTRACS` obtain their final solutions using this new method.

It is worth noting that `PowerSolver` can be seen as an iterative hybridization between GaS and a tool that generates a new formulation (in terms of the ROs

available to GaS) for the same problem instance on each iteration. In this sense, the proposal can be linked to some of the proposals in this thesis, in particular the GaS and Local Search loop introduced in Chapter 4.

Crew Scheduling as an Inspiration for New Search Techniques

Driver scheduling is an interesting problem that has inspired many new techniques while being studied, some which have been generalized to general-purpose search techniques. As an example, early work on fuzzy evolutionary bus and rail driver by Li [60] and Li and Kwan [59] was generalized for the set covering problem [61]; recently Li et al. [4] present a continuation of this work through hybridization between evolutionary algorithms and Squeaky Wheel Optimization.

It is likely that the driver scheduling problem will remain an inspiration for new search techniques. In particular, the proposals on repair costing developed in this thesis (Chapters 6 and 8) have arisen while studying the driver scheduling problem, but are likely to be generalizable to a much broader range of problems.

3.1.7 `TrainTRACS`

`TrainTRACS` is a commercial package developed over the last 30 years by the University of Leeds [54, 83, 31]. `TrainTRACS` is becoming the *de facto* standard in the UK railway industry – as an illustration, in a recent UK franchise bid all the bidders used `TrainTRACS` to assess their crew requirements. Through extensive testing by experienced schedulers in the transport industry, `TrainTRACS` solutions have consistently produced solutions at least as good, most often better, than the best previous known solutions, including those generated manually by human schedulers. Hence, `TrainTRACS` solutions obtained on RoA formulations are regarded as near-optimal (for that formulation) throughout this thesis.

The implementation of most of the algorithms presented in this thesis will either rely on or integrate with parts of the `TrainTRACS` system. Therefore, we include here a description of its main components in terms of their functionality, expected inputs, and outputs generated.

The `TRAVEL` module is used at a pre-processing stage to generate a set of feasible

travel links available to drivers for a given instance of the driver scheduling problem, which is vital for determining whether spells can be joined to form valid shifts. It does so by looking at a description of the railway network, the vehicle schedule for the current driver scheduling exercise, and other travel opportunities available (either through walking/taxi or other bus/rail services that are not part of the scheduling exercise), along with the set of labour rules and other constraints. Travel links are multi-modal and can incorporate up to two stages on train/bus services. `TRAVEL` is also used in some heuristics presented in Chapter 5 to evaluate earliest arrival times and latest departure times.

`TrainTRACS` uses a generate-and-select approach to solve the driver scheduling problem. Its generation phase is executed by a module called `BUILD`, which takes the vehicle schedule, the labour agreement and the outputs of executing `TRAVEL` to generate the set of all feasible shifts for the problem instance. Since the number of feasible shifts may be too big for the selection phase to run on, `TrainTRACS` allows for the specification of extra constraints (e.g. minimum spell length) to restrict the size of the resulting set. When these constraints are not enough, or the user wants to prioritize execution times over solution quality, a so called *reduced build* can be triggered. A reduced build incorporates some heuristics (mostly limiting the time elapsed between the end of a spell and the start of the next spell in a shift) to further reduce the number of shifts generated. Some test instances used in this thesis have to be run using reduced builds in order to get a solution.

`TrainTRACS` exposes the functionality to determine whether any given shift is valid or invalid according to labour agreement rules and extra constraints; we call this the `CHECKER`. Throughout the thesis, we will rely on `CHECKER`'s functionality (which is exposed as a Windows DLL) to determine whether shifts generated by our algorithms are valid. Apart from the usual advantages in terms of software re-use, this ensures consistency at a shift level between `TrainTRACS` and our proposals when comparing solutions.

The selection phase is executed by a module called `SCHEDULE`. It uses a set-covering formulation, operating over the set of shifts generated by `BUILD`, and is based on branch-and-bound. `SCHEDULE` first finds the optimal solution for an LP relaxation of the set covering problem, using column generation to iteratively add more

shifts (from those available from `BUILD`) to the model. This (possibly non-integer) solution is used to define a target schedule size, and also the set of relief opportunities available to the branch-and-bound phase. The branch-and-bound phase uses problem-specific branching rules to gradually take the non-integer solution to an integer one; it also incorporates a number of heuristics to make execution times smaller, including a limit on the number of tree nodes evaluated, and rules for aborting the evaluation of the tree when 'good enough' solutions are found.

As described in Section 3.1.6, `TrainTRACS` has recently been extended with a new module, `PowerSolver`, that runs `BUILD` and `SCHEDULE` (that is, generate and select) cycles iteratively, concentrating the efforts on each iteration on a particular subset of the relief opportunities available on the problem instance. Recent experience [56] shows that `PowerSolver` is able to consistently improve on the results of single runs of `BUILD` + `SCHEDULE` for large problem instances. Therefore, when testing extensions to the generation or selection phases (Chapter 5), we will use `PowerSolver` to assess their efficiency. Although the system provides the users the choice of specifying the way each iteration looks at relief opportunities, it also includes a tool to generate control files automatically; for the purposes of testing, we use the automatic mode as it reduces the opportunities for tailoring the test runs to favour a particular algorithm.

3.2 Solving the Problem of Driver Scheduling with WROs

Although we have explained that train operating companies will usually follow the policy of relieving drivers on arrival, we know from conversations with schedulers that they will sometimes consider relieving on times other on arrival if it results in big enough savings, either when building a driver schedule manually or when taking one obtained from a computerized system as a starting point. This means that although currently no automated scheduling software package seems to offer support for exploiting WROs, schedules implemented in real life are already making use of them.

3.2.1 Previous Work

In the area of transport scheduling and routing, the concept of a 'time window' is usually associated with a constraint that some action (e.g. a pick-up or delivery, arrival or departure) must happen within certain time bounds; in this sense, adding the consideration of time windows is usually a way of specializing a problem by adding new constraint dimension to it. Although WROs as introduced in the previous chapter could be seen as delimiting a range of times in which a certain action can happen, we observe that driver scheduling with WROs is instead a generalization of the original driver scheduling problem, as described in Section 1.3. From a methodological point of view, we are interested in comparing TDS and TDSW, and investigate the benefits in considering WROs. It is then important to differentiate research in time windows in terms of whether they restrict or expand the original problem.

Time windows have been studied for a long time in the area of vehicle routing, although mostly in a restrictive sense. Time windows in daily aircraft routing and scheduling are considered by Desaulniers et al. [21]. Two models are proposed: a set-partitioning one, which is solved using branch-and-bound and column generation, and a time-constrained multicommodity network flow formulation. Their column generation approach also involves the use of a multicommodity network model for the subproblem. Multicommodity network flow formulations have remained popular in vehicle scheduling [6, 85]; however, they do not translate easily to the crew scheduling problem.

Incorporation of time windows as an extension to vehicle or crew scheduling is more recent. Klabjan et al. [51] propose a variation to the crew scheduling problem in airlines, where the flight departure times are allowed to be modified within given time windows for the purposes of obtaining a better crew schedule. Although the problem is different than the one tackled in this thesis, in the sense that windows 'arise' from allowing the vehicle schedule to be modified, rather than existing in the original vehicle schedule, there is a direct association between the reasons for modifying a flight departure time and those for relieving inside a WRO in our problem. In principle, this formulation (and the algorithms proposed for it)

might be re-interpreted and reused for (T)DSW, by modeling the WROs in TDSW as time windows for vehicle departure times. However, it must be noted that the formulation is some ways more restrictive than TDSW, since given a choice of vehicle departure times, relieving is then restricted to those departure times only, whereas in our formulation the full WRO is in principle always available for relieving. However, in Chapter 4 we propose a method that loops over GaS and Local Search phases, where each GaS phase is run with a set of re-timed relief points within WROs derived from the previous local search phase; in this case the two underlying models are very similar.

Other authors also consider time windows in aircraft scheduling as a facilitator for increased efficiency within integrated planning of operations. Ahuja et al. [3] develop models for a combined through and fleet assignment problem with time windows. The authors claim that time windows during fleet assignment allow for greater opportunities between flight legs, which in turn allows for improvements on the combined through and fleet assignment problem. A multicommodity network flow model is first introduced, but the resulting problem is too large to be solved to a reasonable quality; hence, they develop a neighbourhood search algorithm instead, which they claim produce improvements over existing solutions.

A Tabu Search Approach for Driver Scheduling with WROs

To our knowledge, the problem of driver scheduling with windows of relief opportunities is only considered in Shen's PhD thesis [69] and a technical report by Shen and Kwan [71]. In her PhD thesis, Shen initially concentrates on solving the TDS problem without consideration of WROs. A neighbourhood search scheme is proposed, which considers three possible ways (moves) to modify the existing schedule:

1. 'swap links': this move is similar to the 1-point crossover described in Section 4.2.1.

2. 'replace AROs': this move is similar to the transfer of pieces of work (at piece-level only) in Section 4.2.1

3. adding a shift: because their solution can be infeasible and not cover all vehicle work, a specific move to efficiently add a new duty to an existing solution is

proposed. Some work may be removed from existing shifts in the schedule when adding a new shift.

Based on the neighbourhoods described above, Shen then proposes a Tabu Search approach that is embedded into the algorithm described in Algorithm 1.

Algorithm 1 HACS (Shen and Kwan)

```
1: construct a (possibly infeasible) initial schedule, using a heuristic
2: minimize total penalty (using Tabu Search)
3: minimize total cost (using Tabu Search)
4: if current best schedule is feasible then
5:    END
6: else if if total cost has been reduced then
7:    go to step 2
8: else
9:    use an extra shift to reduce penalty, go to 2
10: end if
```

Shen and Kwan then develop an extension of `HACS` for the TDSW problem. WROs are modelled using 1-minute expansions (Section 2.2.2). The 'replace AROs' move is modified to account for WROs. This is done by modifying the interpretation of a predecessor and successor of a RO in a vehicle block (where the order relation is given by the relief time of ROs) in the context of a 1-minute expansion of WROs. For example, given a vehicle block with one RO at location A at 12.00 and one (attended) WRO at location B between 12.30 and 12.31, the predecessor of (B, 12.30) is (A, 12.00), while the predecessor of (B, 12.31) is (B, 12.30). The ARO-replacing move is then redefined to look at predecessors and successors of AROs in terms of the new definition. This equates to considering the work between each pair of consecutive relief times within a WRO as a separate piece of block for the purposes of this move. New moves (swapping spells and inserting spells) are also added.

Experiments are carried in a set of 10 instances, mostly small in size (only one of the ten instances is comparable in size to those used in Chapter 5 in this thesis). Moreover, instances are simplified (e.g. by eliminating unattended WROs). It is not clear from the thesis or report which side constraints (e.g. labour agreement rules)

apply to the test instances. In most cases, the test instances are RoA formulations where WROs are added artificially, using a uniform 5-minute WRO length; our experience in this thesis suggests that this may overestimate the amount of time windows available on real-life instances (for example, most WROs in the *InterCity* dataset are two or three minutes long; most WROs in the *ThamesLink* dataset are only one or two minutes long). On these instances, experiments on `HACS` starting from solutions not obtained by `TrainTRACS` are on average 0.55% higher cost than those obtained running `TrainTRACS` on a RoA formulation, while experiments on `HACS` starting from the solutions obtained with `TrainTRACS` over the RoA formulation show an average decrease in cost of around 1%.

Discussion The work by Shen and Kwan provides a good starting point to the study of WROs in the context of train driver scheduling. Some central concepts are introduced, like the idea of a 1-minute expansion of WROs, and that of retiming an active relief on a WRO, although the latter is only exploited in a very limited way, i.e. by moving the relief point one minute at a time in the context of 'replace ARO' moves.

In our view, the main limitations in this work are, first, that no study is conducted on how the problem of driver scheduling changes when WROs are introduced; second, the algorithms are developed for simplified versions of the problem, and WROs in the test instances are artificially generated –hence, it is difficult to translate the gains in performance shown in the experiments with actual gains achievable in a real-life context. At the same time, existing tools for the TDS problem are only exploited in a very limited way, e.g. to provide starting solutions. In this thesis we try to address these three perceived limitations to advance the study of the TDSW problem.

3.2.2 Limitations of GaS

A first intuitive approach to considering WROs in the scheduling problem is to adapt the GaS approach to accommodate for them. To do so, it is sufficient to adapt the generation phase, since all inputs to the selection phase (including the set $\mathcal{C}$ of candidate shifts and its derived matrix $\{a_{ij}\}$ in the set covering representation)

are determined during the generation phase.

A relatively straightforward way of extending the generation phase to account for WROs is to model each WRO as a discrete set of individual ROs of the form $< time, location >$, as suggested in section 2.2.2. Since `TrainTRACS` works on minute units, it makes sense to discretize WROs as a set of ROs separated by one minute; therefore, a WRO in location L starting at 10:10 and ending at 10:12 would be represented as a set of three ROs at location L, occurring at 10:10, 10:11, and 10:12. Hence, the piece of the WRO between each of these new ROs becomes a separate, new piece of work, while the piece of work to the right of the WRO is reduced in length by two minutes when compared to the RoA formulation.

In theory, any algorithm that works on the original RoA instance should be able to generate shifts on this new 'extended' instance. However, there is a major drawback to this approach: the number of feasible shifts usually grows exponentially with the number of ROs considered. The problem is even worse when ROs are bundled together, as is the case with the 1-minute WRO expansion proposed. This is illustrated in Figure 3.1. Suppose s is a legal spell on vehicle v. The right

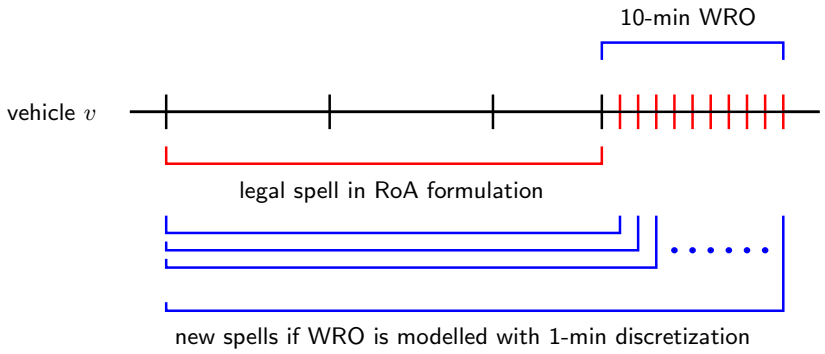

Figure 3.1: Combinatorial explosion in the number of shifts when incorporating time windows into the scheduling model. Spells which are legal in the non-windowed model usually give rise to many new legal spells if a WRO on one of its ends is expanded.

end of the spell is marked by the presence of a 10-minute WRO, which in a RoA formulation will be approximated by its arrival time. Now, if we model the WRO with the discretization proposed in the previous section, 10 new ROs will be added. Because these new ROs are so close in time to the original one, it is very likely that changing the end-RO for spell s to any of the new ROs will result in a (new)

legal spell. This means that where the generation phase on the RoA formulation would build a single valid spell, it will instead build ten legal spells on a WRO formulation. The nature in which spells are then combined to form shifts results in an exponential increase in the number of valid shifts. Incidentally, if one were to run the selection phase without any modifications to the usual mechanism, covering of WROs would follow the unconstrained model presented in Appendix A.1, which is a further indication of the difficulties a GaS approach would face in finding good solutions on this extension.

In practice, fully expanding WROs with 1-minute apart ROs results in instances that are intractable with current selection mechanisms and computer hardware. However, the fact that WROs will result in groups of bundled ROs also has a positive implication. Going back to the previous 3-minute WRO example introduced in Section 3.2.2, it is likely that for every optimal schedule using the RO at time 10:11 to relieve drivers, there is a schedule with the same cost that uses either 10:10 or 10:12 as the relief point instead. This would mean that we can skip the addition of the RO at time 10:11 without affecting the solution cost achievable. In the context of building an optimal solution on the WRO formulation from an optimal solution for the RoA formulation, this suggests that perhaps only some ROs within WROs will play a part in allowing for better solutions to be found (and so we may not need to consider all ROs within WROs). The interplay between the choice of formulation (in terms of the ROs made available to the solver), the resulting problem size, and the possibility of obtaining efficient schedules is one of the main themes of this thesis.

3.3 Overview of Optimization Techniques

In this section we look in more detail at a group of general-purpose optimization techniques whose underlying ideas and principles form the basis of the proposals in this thesis. We assess their motivations and key ideas in the context of solving the problem of driver scheduling with WROs, and also introducing notation and vocabulary that is later used throughout the thesis. Aarts and Lenstra [1] provide a comprehensive presentation of these techniques and many others, including the

application of some of them to vehicle routing and machine scheduling. For illustration purposes, and without loss of generality, we restrict ourselves to minimization problems.

Some techniques that are also related to our proposals but to a lesser degree are discussed in the context where the relation is more evident; these include *Evolutionary Algorithms*, *Squeaky Wheel Optimization*, *Simulated Annealing* and *Path Relinking*.

Neighbourhood Search A good starting point when looking at general-purpose optimization techniques within combinatorial optimization is neighbourhood search, since most successful metaheuristics can be seen as extensions to it – this agrees with the view implicit by Aarts and Lenstra, who in their book include simulated annealing, tabu search, genetic algorithms and artificial neural networks under the category of 'local search' algorithms. In this thesis we will use the terms 'neighbourhood search' and 'local search' interchangeably.

The key proposal in local search is that of building a sequence of increasingly better solutions to the problem, where each new solution S_{next} is selected from a set of *neighbouring* solutions to the current solution S_{curr}. Intuitively, neighbouring solutions are 'close' to each other in some measurable way. In general, the relation is defined through a *perturbation* or *move*, which determines how new solutions can be generated from S_{curr}; the minimum number of moves needed to get from one solution to another can be seen as the *distance* between them.

In order to get a sequence of increasingly better solutions, the selection of the new solution in the sequence is usually driven by a comparison of *costs*, i.e. the value of the objective function. In the simplest implementation, called *hill climbing*, a solution has to be strictly better in terms of cost than the current solution in order to be chosen as the next solution in the sequence: $cost(S_{next}) < cost(S_{curr})$. Variations exist within hill climbing in the way that the neighbourhood of the current solution S_{curr} is explored. In *simple hill climbing*, the first solution in the neighbourhood of S_{curr} whose cost is less than that of S_{curr} is chosen to be S_{next}. In *steepest descent hill climbing*, all neighbours of S_{curr} are evaluated, and S_{next} is chosen as the/a solution with the lowest cost (assuming its cost is lower than that of S_{curr}). Exploring

a neighbourhood fully can be extremely expensive (in some cases, neighbourhood sizes can even be exponential in the problem size) – in this thesis we will mostly follow a first-improvement criterion.

Criticism and Extensions The main drawback of local search approaches as described so far is that, by their very construction, they will become trapped in *local optima*, i.e. solutions for which there is no neighbouring solution with lower cost. In most combinatorial optimization problems, the search space –as defined by the problem representation and the local search implementation, including the neighbourhood structures– will contain a vast number of local optima, only some of which are also global optima. Therefore, the local optima to which local search converges will most of the times be sub-optimal solutions.

Many of the extensions to local search can be seen as attempts to augment the basic scheme with tools that allow it to 'escape' from local optima. A simple adaptation includes re-starting the search when it reaches a local optimum. In order to have successive runs arrive at different optima, some sort of randomization in the search process is needed; the most popular such technique, known as GRASP [26], proposes a greedy-randomized mechanism for constructing the initial solution, which is followed by a local search. Although we do not implement GRASP in the thesis, some similarities can be established between its greedy-randomized construction stage and the randomized repair proposals in Chapter 6.

A different approach is to allow for $cost(S_{next}) = cost(S_{curr})$; this however is usually not enough to escape most basins of attraction. Two very successful meta-heuristics, *simulated annealing* [50] and *tabu search*, take this further by allowing (under certain conditions) $cost(S_{next}) \geq cost(S_{curr})$. Both relaxations to the cost relationship between $cost(S_{next})$ and $cost(S_{curr})$ create a new problem in that the sequence of solutions in the search could become cyclical, since as soon as a solution of bad quality is inserted in the sequence the local search will tend to take the sequence back to lower-cost solutions, and previous solutions in the sequence are very likely to be selected as they are 'close' to the current bad-quality solution (at least in neighbourhoods that are symmetrical, i.e. $s_1 \in N(s_2) \Leftrightarrow s_2 \in N(s_1)$). We describe the solution to this problem proposed by tabu search in more detail later

in this chapter.

A second criticism to local search approaches is that (at least in its simplest forms) they rely almost completely on the cost function to compare solutions and therefore guide the search, making it difficult to consider elements at a more atomic level than a full solution. It is therefore difficult to force biases on the search procedure, for example towards a desirable particular structural property. We believe that two intuitive ways of dealing with this are:

1. adapting the cost function used in the search to include explicit consideration of structural attributes

2. integrating stages in the search where the solution is perturbed explicitly to favour or impose a particular type of structural attribute on the current solution

Notice that the two approaches are orthogonal, since one operates over the definition of the cost function, and the other by temporarily working outside of a traditional cost-comparison local search framework. We study the first type of mechanism in the context of the local search proposal introduced in Chapter 4 (Section 4.2.1, conditioning), while in Chapter 8 we make use of the second type of method to 'unlock' a search that stagnates.

Tabu Search As described earlier in this section, tabu search extends the local search framework by allowing $cost(S_{next}) \geq cost(S_{curr})$. To overcome the problem of the search getting trapped in a cycle, tabu search incorporates the concept of a *tabu list*, which stores solutions (or attributes of solutions); any solution appearing in the tabu list (or sharing an attribute with one stored in that list) is not allowed to be selected as the next solution in the search. The list is updated dynamically during the search, and at any point in time it usually contains recently-accepted solutions in the search sequence (or attributes of those solutions), effectively forbidding recent solutions from being accepted again.

When the tabu list is implemented by storing (thus forbidding) attributes of solutions, rather than complete solutions, it is possible that promising solutions that share certain attributes with past solutions in the search are unfairly prohibited from

being selected. To counter this, many implementations of tabu search also include an overriding *aspiration criterion*, which states that if a candidate solution has a cost better than that of all previous solutions in the sequence, then the solution is accepted even if it violates the tabu list.

Criticism and Possible Uses Tabu search introduces a number of very interesting concepts, and it has proven very popular and successful in practice (for an early survey of applications of tabu search in the literature, see Chapter 8 in [36]). The tabu list is a very clever way of preventing cycles when combined with accepting cost-increasing solutions. In particular, the variation in which the tabu list contains attributes of solutions is very attractive, as it can be argued that it allows to incorporate explicit structural considerations during the search. In many cases, these attributes are directly associated with the moves being used; for example in the context of *k-opt* neighbourhoods one might choose to include the reverse exchange in the tabu list.

A drawback of tabu search (which is shared by many metaheuristics, particularly those extending a simpler technique) is that any implementation of the technique must determine values for a number of parameters, such as list size or the length of time for which a solution or attribute is forbidden. There is usually no automatic way of determining or 'tuning' these, so most implementations either choose an arbitrary value for them, or determine them empirically based on a (usually small) set of problem-specific test instances.

It is worth noting that many authors use different terminology to describe the characteristics or main attributes of tabu search; in particular, tabu search is often described as a way of incorporating (short-term) memory into the search. However, there are other metaheuristics that can be said to extend local search by adding memory; as an example, when describing the motivations for their *Variable Neighbourhood Search* (VNS) metaheuristic, Hansen and Mladenović [40] consider memory as a key element in a local search framework by stating that 'a local optimum often provides some information about the global one'.

Throughout this thesis, there will be more than one time where we find ourselves in a situation where steps should or must explicitly be taken to prevent the search

from looping back to previously obtained solutions; in these cases, the solutions suggested or implemented will be similar in spirit to tabu search. In particular, the mechanism to avoid cycling in Chapter 8 can be seen as an attribute-based tabu list of size 1, with an aspiration criterion as defined above.

Column Generation Although it is our belief that column generation is not mature enough in the context of train driver scheduling to address the extension to WROs, we think it is relevant to examine its motivations and proposals, and evaluate whether these can be translated to a context other than linear programming, where they can be applied to the train driver scheduling problem with WROs.

It is commonly accepted that column generation was in great part spawned by a 1958 paper from Ford and Fulkerson on maximal multi-commodity network flows [29]. As early as this paper, the proposal addressed the issue of problem instances where the number of elements (non-basic variables) to be considered when finding a solution (using simplex) are too many to even be listed explicitly (when determining a vector to enter the basis). In terms of the proposed solution, column generation as a methodology can be seen as incrementally building the set of elements that is considered explicitly during the optimization. The choice of new elements to be added to this set is interleaved with the optimization process itself, and uses information from the optimization to help evaluate or find such elements.

In the case of the driver scheduling problem we have a similar problem at hand, since the set of feasible shifts sometimes becomes too large to be considered fully. In particular, when considering WROs the full set of feasible shifts will most of the times be too big to consider all shifts explicitly. Similarly, the solution proposed by column generation could be abstracted out of linear programming, with the description given above, and possibly applied to the driver scheduling problem. Finally, we note that authors like Fores [30] use column generation in a pre-processing stage. It can then be argued that our proposals in Chapter 5 address the same issue through a similar kind of approach than some existing applications of column generation, when the motivations and methods in column generation are generalized out of linear programming.

Chapter 4

A Hybridized Integer Programming and Local Search Approach

4.1 Motivation

In Chapter 3 we showed that the GaS approach is unsuited to the case where WROs are expanded in 1-minute intervals; this is mainly due to the exponential increase in valid shifts that can/would be produced during the generation phase. As its name suggests, GaS is based on pre-generating a set of candidate shifts, from which a schedule is then built. In the standard GaS approach, we can think of the generation of shifts as *static*, in the sense that no new shifts will be created during the selection phase. Because the number of valid shifts in a real-life instance of the TDSW problem is extremely big, any algorithm that is based on (pre-)generating candidate shifts statically needs to incorporate ways of constraining the size of the set generated. This is even the case for approximated models (such as RoA); for example, runs of `TrainTRACS` on real-life instances will usually be subject to artificial

constraints (such as minimum spell length), or may even require the execution of so-called *reduced builds*, where heuristics are applied to keep the size of the resulting pool of shifts manageable. Still, static shift generation approaches, and in particular GaS, have proved very effective in practice, at least for approximated models of the problem.

In contrast, algorithms that do not pre-generate a set of candidate shifts will in principle not be constrained by the number of potential valid shifts for a particular problem instance. This makes them especially attractive in our setting of 1-minute expansion of WROs. We will call any scheme that does not rely on a set of pre-generated shifts a *dynamic shift-generation approach*.

In summary, dynamic shift generation approaches seem to scale to large, real-life instances of the driver scheduling problem with WROs, while GaS (a static generation approach) has proven very effective at solving the same problem for approximated models. Our first proposal will then be an attempt at reconciling these two properties of dynamic and static shift generation.

4.2 The Proposal

Our first proposal for solving the train driver scheduling problem with WROs is a two-phase scheme. In the first phase, GaS is run on an approximated (i.e. non-windowed) model of the problem, yielding a solution $\mathcal{S}^{GaS}$. After that, a local search phase is run on a WRO formulation, using the schedule $\mathcal{S}^{GaS}$ as the initial solution. We use the set of ROs derived from a RoA formulation as the initial set of approximations for the WROs. The approach is illustrated in Figure 4.1.

Consistent with the motivations presented in the previous section, the scheme first relies on explicit shift generation/enumeration on an approximated model, where the number of feasible shifts is still manageable, banking on the efficiency of GaS approaches on approximated models to generate a near-optimal solution for the RoA formulation. It then switches to fully considering WROs, while simultaneously moving to a dynamic shift-generation approach, therefore avoiding the limitations of static-generation approaches on WRO formulations. It is worth noting that because

0. generate a set A of approximations for the WROs in the set of vehicle blocks B (using arrival times)

1. run *Generate and Select* on the model approximated by A, obtaining a schedule $\mathcal{S}^{GaS}$

2. run *Local Search* on an extended (i.e. non-approximated) model for B, with initial solution $\mathcal{S}^{GaS}$, to obtain an improved solution $\mathcal{S}^{LS}$

Figure 4.1: The proposed two-phase, GaS and Local Search approach for solving the TDSW problem

the solution set of the WRO formulation is a superset of *any* approximated model, $\mathcal{S}^{GaS}$ is always a feasible solution for the WRO formulation, and thus can be safely used as an initial solution for the local search phase.

4.2.1 The Local Search Phase

In this section we describe the main aspects of our local search proposal. We put special emphasis in describing how WROs are exploited, and how low-level moves are integrated into higher-level schemes, including the use of 'conditioning' phases to try and impose certain structural attributes into the active solution.

Local Search and the Artificial Constraints in GaS

Because of the way the scheme is structured, it is only in the local search phase that WROs can be exploited. This is therefore the main objective of the local search phase. However, local search is complementary to GaS in other areas. In particular, because the practicability of a local search phase is in principle not tied to the number of feasible shifts for the problem instance, we can relax any/all of the artificial constraints that are introduced on a GaS setting for real-life instances to limit the size of the output shift pool. This means the local search can effectively consider shifts that would be illegal on GaS because of artificial constraints. This adds another way of achieving improvements in the schedule, which is actually independent

of the introduction of WROs to the driver scheduling model.

A Generic Two-Step-Move Framework for Exploiting WROs

Since the main aim of the local search phase in our proposal is to exploit WROs, we prioritized creating moves that involve the use of WROs over moves that exploit the structure of the TDS problem in general. Our initial intention was then to design a set of WRO-exploiting moves, complemented as necessary by others that don't. However, it turned out that for every move that doesn't exploit WROs we could design a move with the same function, but which also exploited WROs.

Therefore, we modified the initial scheme by one where every move may potentially exploit WROs. This is done by dividing every move into two steps. In the first step, we apply a specific perturbation to a subset of the shifts of the current schedule. The second step is a generic procedure that, given a set R of (W)ROs that are active in the current schedule (after step 1), evaluates the possibility of altering the times at which driver reliefs take place inside any WROs in R. Each move selects this set R according to the characteristics of the perturbations it has performed on the first step. For example, if a move splits a shift s into two shifts s'_1, s'_2, the second phase will probably receive a set that consists of the ROs occurring at the ends of s'_1 and s'_2. A graphical example of this two-step methodology can be seen in Figure 4.2. After the specific perturbation (in this case, a swap of two spells) is performed, the ROs delimiting the gaps between the new spells are chosen as the set R to be passed to the window-evaluating routine (these are shown with arrows in the figure). The routine will consider combinations of re-timings of reliefs inside WROs in R, generating different sets of candidate shifts from s'_1, s'_2, which by construction only differ in the relief times inside WROs in R. Eventually, the best of these options will be chosen as the new solution.

Choosing a model for covering WROs is a balance between flexibility and complexity/running time. In this local search phase, as in later local search proposals in this thesis, we rely on a model of WRO covering in which all spells in shifts in the schedule that start or end at a given WRO w do so at the same time. This model, along with a number of alternatives, is analyzed in more detail in Appendix A. The model we use is the *constrained (version III)* in A.1.4, with $k = 0$.

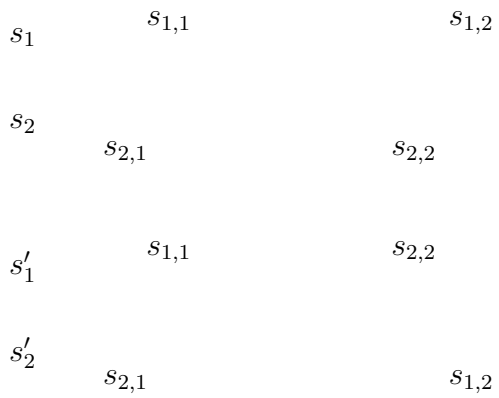

Figure 4.2: Example of a move: spell swapping. s_1 is a shift with two spells $s_{1,1}$, $s_{1,2}$; s_2 is a shift with two spells $s_{2,1}$, $s_{2,2}$. Dotted lines represent non-driving periods. First, spells $s_{1,2}$ and $s_{2,2}$ are swapped (move-specific perturbation). Then, ROs that are relevant to the perturbation just performed are inspected for optimal relief times within WROs.

Validation of Candidate Solutions

The feasibility of a schedule is governed by both shift- and schedule-level constraints. An example of the former is the maximum shift length; an example of the latter, the number of part-time shifts in relation to the total number of shifts in the schedule. Since a move is basically a perturbation of a subset of shifts of a schedule (the current solution), shift-level constraints need only be checked for shifts that are being perturbed – notice this may include shifts which were not initially selected by the move, but which took part in a relief re-timing in the second step of the move.

In principle, it would be desirable to be able to guarantee that a schedule generated by a move is always feasible, as this avoids spending time on generating solutions that then have to be discarded. Failing that, it would be desirable that at least the shifts generated by a move are feasible, since this would intuitively increase the probability of the new schedule being feasible as a whole. However, in real-life train driver scheduling shifts must adhere to a large and complex set of labour agreement rules. This not only makes the task of designing a move that generates only feasible shifts extremely hard: it even makes the task of assessing the feasibility of a shift difficult. Therefore, in our implementation we don't attempt to design moves that guarantee feasible shifts. Instead, local search moves only guarantee that the

most basic 'structural' constraints are satisfied (e.g. a driver is never assigned to drive two pieces of work that overlap in time). Most of the labour agreement rules are disregarded when executing a move, with the exception of maximum shift/spell length in some moves. It is worth noting that the relief re-timing trials in the second stage of a move may not only decrease the cost of a solution, but also turn an infeasible solution into a feasible one. Therefore, candidate solutions are not checked for validity during the first step of the move, but rather for each possible combination of relief timings inside the set R of ROs derived by the move.

In order to check that shifts are still valid after executing the perturbations, the local search interfaces in a black-box fashion with checking routines that are also used during the generation phase of the GaS solver. In this implementation, we interface with shift-validation routines (the `CHECKER`) available in the `TrainTRACS` package, by linking to an external Windows DLL provided by it. `TrainTRACS` does not natively support WROs; therefore, in order for `CHECKER` to be able to correctly assess the validity of shifts using WROs we use a 1-minute-expansion formulation of the problem for the purposes of validation; the association between relief points inside WROs in the WRO formulation and ROs in the 1-minute-expansion formulation is handled internally by our tools. `CHECKER` does not rely on building all shifts explicitly in order to validate shifts, and hence is not subject to the problem of combinatorial explosion in the number of feasible shifts.

Decoupling the validation of shifts from the perturbation of schedules not only simplifies the task of designing the moves, but also has a number of advantages from a software engineering point of view, including the fact that shift-level constraints are guaranteed to be treated consistently on the two phases, and even allowing for external tools to be used for this purpose, as described.

Design and Implementation of Moves

We designed a two-level hierarchy for moves. L_0 moves are atomic moves, i.e. they do not resort to other moves to perform the perturbation.

- *re-timing of a relief inside a time window.* In the first phase, an (active) WRO is selected. In the second phase, all possible re-timings of the relief inside that

window are evaluated.

- *1-point crossover.* Two shifts are selected. One relief opportunity (windowed or not) is selected on each shift. The portions of the shifts to the right of those ROs are exchanged (if certain basic checks are satisfied). Because selected ROs are not necessarily placed at spell boundaries, this move can in effect break existing spells. Also, if the shifts share an RO, it is possible to select this RO as the crossover point. In this case, the so-far active RO becomes inactive after the exchange. In the second phase, ROs selected as crossover points are tested for relief re-timing.

- *Transfer of spells and pieces of work.* Individual spells or pieces of work can be transferred from one shift to another. Begin and end ROs of the spell or piece transferred are tested for relief re-timing in the second phase of the move.

Moves similar to the 1-point crossover and transfer of pieces of work were originally proposed in Shen [69] and Shen and Kwan [70] for a simplified version of the problem, as described in Section 3.2.1, although as described their moves do not handle the concept of a WRO explicitly and hence no relief retiming is possible.

L_0 moves are used or combined into L_1 moves. Some of these moves are high-level, requiring perhaps a sequence of many L_0 moves to be called; for example, "shift dismembering" is an L_1 move in which a procedure systematically tries to transfer every piece of work in a shift to other shifts in the schedule, or a move in which the spells of a 2-spell shift are distributed to two other existing shifts in the current schedule. Other L_1 moves are aimed at avoiding the cost of calling L_0 moves where we have reason to believe that the solutions generated will be infeasible, and instead only calling L_0 moves for which certain checks are satisfied. Therefore, although the feasibility of a shift is tested with external, black-box-like checking routines, some domain-specific knowledge *is* incorporated into the local search for efficiency reasons.

The overall driving scheme used in this chapter (what we call an L_2 move) was determined by comparing several different loops of sequential calls to specific L_1 moves, sometimes combined with conditioning phases (see next section). For every alternative, the choice and ordering of L_1 calls and conditioning phases was

hardcoded. Tests presented in this chapter use the disposition that experimentally showed the best results.

Conditioning

All moves described in the previous section are called with the intention of decreasing the cost of the current solution; therefore, new solutions are accepted only if the cost is decreased (or, under certain circumstances, not increased). This may lead the local search to quickly stagnate in a local optimum. As discussed in Section 3.3, metaheuristic approaches like *Tabu Search* or *Simulated Annealing* tackle this problem by accepting certain cost-increasing moves; however, these approaches leave the cost function and (to a certain extent) the neighbourhoods unchanged.

We want to be more proactive in guiding the local search to good areas, and away from local optima; more specifically, we intend to favour certain structural properties in the solutions, which intuitively could lead the local search to better final solutions. As an example, short-spreadover shifts might be easier to "dismember" into other shifts; if the cost function is based on the number of shifts in the schedule, having a schedule with many short-spreadover shifts could make the task of reducing the number of shifts in the solution easier. However, a schedule with short-spreadover shifts will be inefficient in terms of schedule size, and hence will rarely be selected by a local search metaheuristic (including tabu search and simulated annealing).

The way we favour the appearance of these properties is by temporarily changing the cost function with which the local search is conducted. We call these the *conditioning phases*.

We investigated favouring two different properties:

1. Generate short-spreadover shifts. For that purpose, the cost of a move (which is a perturbation to a subset $\mathcal{S}_m \subseteq \mathcal{S}$ of shifts) is the decrease in the minimum spreadover of $\mathcal{S}_m$ ($\min_{s \in \mathcal{S}_m} spreadover(s)$), relative to the original minimum spreadover in $\mathcal{S}_m$ before executing the move.

2. Generate long inter-spell gaps of non-work time, which may increase the chances of adding new work to the shifts in those gaps, and also of performing some spell-swapping moves that were previously infeasible. For that purpose,

the cost of a move is defined to be the increase in the maximum inter-spell gap of $\mathcal{S}_m$, relative to the original maximum inter-spell gap in $\mathcal{S}_m$ before executing the move.

The first criterion produced better results than the second. Overall, this way of alternating between conditioning and non-conditioning phases has the inconvenience that moves executed in the conditioning phase may be later undone by moves made in the following cost-improving phase, because different, often conflicting and even opposite cost functions are used in each phase. In our experience, it is quite complex to tailor the specific moves and cost functions to stop this from happening. One such approach, borrowing from the concepts of the *tabu list* and *aspiration criteria* in tabu search, is presented in Chapter 8.

4.2.2 Experiments

The algorithm was tested using real-life instances from the Scottish train operator *Scotrail*. We used `TrainTRACS` as the GaS solver for the first phase or our algorithm. A typical instance from *Scotrail* contains around 1,000 relief opportunities, which under the usual set of constraints (including the artificial ones) results in a pot of around 170,000 candidate shifts generated during the execution of GaS. The solutions obtained after the local search phase adhere to all operating rules as defined by the operator for its scheduling exercises.

Schedule Cost

We first analyzed the potential of the new approach for reducing the cost of a schedule. Results for a sample run are shown in Table 4.1.

artificial constraints	*GaS*	*GaS+LS*	improvement
not relaxed	459.48h	458.13h	-0.34%
relaxed	N/A	455.18h	-0.98%

Table 4.1: Results of the GaS and Local Search approach for a real-life instance. Two variants of the hybridized method were analyzed: in the first one, the artificial constraints introduced during the GaS phase enforced during the local search phase; in the second one, they are removed.

For an average train operator, a 1% cost reduction in the drivers schedules would mean a saving of several hundred thousand pounds per year. The improvements obtained are thus considerable for train operators, especially given the near-optimality of the initial solution (on the approximated model).

Schedule Robustness

WROs could play a very important role in increasing schedules' robustness. Interaction with schedulers has allowed us to identify some aspects of a schedule that are key to enhance its capability of absorbing delays that may occur during operation. We have defined two such indicators. (1) *Slack* is the unproductive time during signing on, mealbreak or joinup. Slack occurs naturally in a schedule; it might be added to a shift so that it conforms to some labour union rules. Moreover, slack in a shift could be used to absorb some delay when it occurs. The scheduler has to strike a balance between producing a cost-efficient schedule and adding enough slack so that the schedule meets robustness criteria. (2) When a train is so delayed that the slack in the driver's shift is unable to absorb it, a pragmatic way which a train crew manager would employ is to switch an available driver, say d, at the same location to take over the next train that the delayed driver was due to take on. This way the train driver can be relieved and have his/her legal break, and then take on the train that was originally assigned to d. We refer to this as a *swap opportunity*.

To test the capability of our approach in tackling robustness, we translated these two measures of robustness into separate cost functions: given a schedule $\mathcal{S}$, $slack(\mathcal{S})$ computes the total minutes of slack in $\mathcal{S}$, and $swapOpp(\mathcal{S})$ computes the number of swap opportunities in $\mathcal{S}$. Let $cost(\mathcal{S})$ be the original cost function related to the number of shifts and total payable hours in $\mathcal{S}$. Then, the GaS phase is still executed using *cost* as the cost function, generating a solution $\mathcal{S}^{GaS}$. The local search phase, however, is now driven by one of the new, robustness-related cost functions. It is likely that train operators will want to balance cost and robustness in a schedule. For these initial tests we decided to force the local search to guarantee that every solution obtained during its execution is as cost-efficient as the starting solution, $\mathcal{S}^{GaS}$; this was achieved by adding a constraint $cost(\mathcal{S}') \leq cost(\mathcal{S}^{GaS})$, where $\mathcal{S}'$ is any candidate solution being considered during the local search phase.

Results for experiments on the same data used in the previous tests are presented in Figure 4.2. These suggest that both measures of robustness can be increased by using our approach, with no additional cost to the schedule. However, it should be noted that the GaS solver used for these tests is not tailored to optimizing any of the two robustness measures, so comparisons between the local search approach and `TrainTRACS` in terms of how they can improve schedule robustness would be misleading; similarly, the contribution of WROs to improving robustness cannot be assessed conclusively through these tests. The next logical step in this research direction would be to modify the GaS solver (in this case, `TrainTRACS`) so that the selection phase is guided by the same robustness-related cost functions than the local search. However, `TrainTRACS` does not provide access to modifying the cost function in use, and even if it did it is likely that the algorithms for both the generation and selection phases would have to be reconsidered thoroughly, including heuristics in the generation phase and branching rules and column-generation procedures in the selection phase. This research is not tackled in this thesis.

measure of robustness	$\mathcal{S}^{GaS}$	*after LS*	improvement
slack	1,604	1,648	2.74%
swapOpp	6	8	33.33%

Table 4.2: Results of the GaS and Local Search approach for a real-life instance, using measures of robustness in the LS phase. Both total slack and number of swap opportunities could be increased, at no extra cost in the schedule.

On the Reproducibility of Results

One of the main premises of research in optimization (and many other areas) is that experiments and results should be reproducible. We can refine on this concept by distinguishing to aspects of reproducibility: first, the researcher that conducted the experiments has to be able to repeat the experiment and obtain the same (or reasonably similar) results; second, it is expected that other researchers can also generate similar results by following the description of the experiments in question. With regards to the first item, it should be clear that for most (if not all) optimization problems, once the problem and a particular problem instance are described fully,

the researcher would be expected to be able to reproduce previous results faithfully, e.g. generate the same end-solution for that problem instance.

Many algorithmic approaches to optimization problems resort to randomization, i.e. incorporate points during the execution of the algorithm where a choice between two or more alternatives is selected on the basis of an instantiation of a random variable. True randomization would seem to make it impossible to guarantee reproducibility in those cases (other than in some statistical sense); however most implementations of randomization in computer systems resort to pseudo-random number generators, which allow some control over the sequence of 'random' numbers generated over time.

In this thesis, all implementations of algorithms that involve randomization and are under our control are done in a way that the sequence of numbers generated can be reproduced in later executions, mostly by having a single-threaded implementation that shares a single number generator across the whole execution, and allowing for the specification of 'seeds' for those number generators, thus making it easier to achieve reproducibility. However, we also interact in many cases with external tools, particularly `TrainTRACS`. We observe that `TrainTRACS` does not provide a way to guarantee that the results for a given problem instance will be the same in two separate runs; moreover, we have experienced `TrainTRACS` generating different results for the same problem instances while working on this thesis, and this behaviour has been corroborated by other `TrainTRACS` users. This means that when our experiments involve the use of `TrainTRACS` (particularly when using the selection phase of their GaS solver), reproducibility can be compromised.

An apparent manifestation of this problem can be observed in Figures 4.4 and 4.5 in pages 72 and 73, where the cost of the first solution in the loop varies from one experiment to another (roughly 690 hours in the first experiment against 695 hours in the second), under apparently the same problem instance. However in this case the difference is actually due to the fact that the `BUILD` tool used for the generation phase had to be run in a different mode in the second experiment than in the first, due to the addition of constraints, and this results in a slightly different set $\mathfrak{C}$ of candidate shifts generated for the first iteration. Hence one must also be careful when aiming for reproducibility in verifying that the problem and problem

instances are actually equivalent across experiments.

4.3 Looping Back to GaS

4.3.1 Motivation

The final step in the approach proposed in Section 4.2 is a call to a local search phase, which produces a schedule $\mathcal{S}^{LS}$. Now, even if the schedule was generated on the extended, windowed model, all driver reliefs in $\mathcal{S}^{LS}$ happen at precise time points. Therefore, $\mathcal{S}^{LS}$ is also a solution for a particular approximated model A^{LS}, namely that obtained by approximating all windows of relief opportunities by the times at which the reliefs are taking place in $\mathcal{S}^{LS}$ (inactive WROs can be approximated by e.g. their arrival times). If $cost(S^{LS}) < cost(S^{GaS})$, it can be argued that A^{LS} is a *better approximated model* than the one derived from A, which was used in the GaS phase.

The GaS approach has been very good in solving approximated models; in particular, `TrainTRACS` is being successfully used by transport operators for their scheduling tasks. Therefore, if restricted to an approximated model, it may happen that GaS/`TrainTRACS` can perform better than the (somehow basic) local search we have implemented in this chapter. It may thus be wise to attempt solving the problem for the approximated model A^{LS} using GaS. Once this first loop-back to GaS is established, it is natural to consider extending the idea to running GaS and local search in a loop, iterating until no further improvements to the approximation can be made. The algorithm is presented in the next section.

4.3.2 A Loop-back Version of the Algorithm

The loop-back version of the algorithm is presented in Figure 4.3. Again, an initial set A_1 of approximations is generated using the arrival times. GaS and Local Search phases are then carried out in sequence. In step 5, a new set of approximations A_{i+1} is generated according to the times at which reliefs are taking place inside windows in $\mathcal{S}_i^{LS}$, and the GaS and Local Search loop starts again with this new set of approximations.

0. set $i := 1$; generate a set A_1 of approximations for the WROs in the set of vehicle blocks B (using arrival times)

1. run *Generate and Select* on the model approximated by A_i, obtaining a schedule $\mathcal{S}_i^{GaS}$

2a. run *Local Search* on an extended (i.e. non-approximated) model for B, with initial solution $\mathcal{S}_i^{GaS}$, obtaining $\mathcal{S}_i^{LS}$

2b. generate a new set of approximations A_{i+1} for the windows in B, based on the active relief opportunities in $\mathcal{S}_i^{LS}$

2c. set $i := i + 1$; go to step 1

Figure 4.3: A loop-back version of the algorithm: a new set of approximations A_{i+1} is generated from the solution $\mathcal{S}_i^{LS}$ obtained in the last call to local search, and fed back to the GaS solver

4.3.3 Experiments

We present two sets of experiments on the loop-back mechanisms to GaS. These provide initial evidence that the approach is viable, pointing out at the same time the main issues arising when implementing such a scheme.

One: Unconstrained GaS phases

On the first set of experiments, the only information derived from the local search phase for the next call to GaS is that of the next set of approximations A_{i+1}. We call these *unconstrained experiments*, because even if we know that solutions with specific cost values exist (we obtained them on the local search phase), we don't force GaS to equal or better those costs. The results of a typical run are shown in Figure 4.4.

As shown in the figure, the behaviour of the loop is erratic. Since the local search is set to accept only cost-decreasing solutions for these experiments, it is intuitive to expect that the overall behaviour of the cost function during the loop is decreasing. The reason why this is not reflected in the actual results is that the GaS solver is not an exact algorithm, and some heuristics are built into it to speed up the CPU

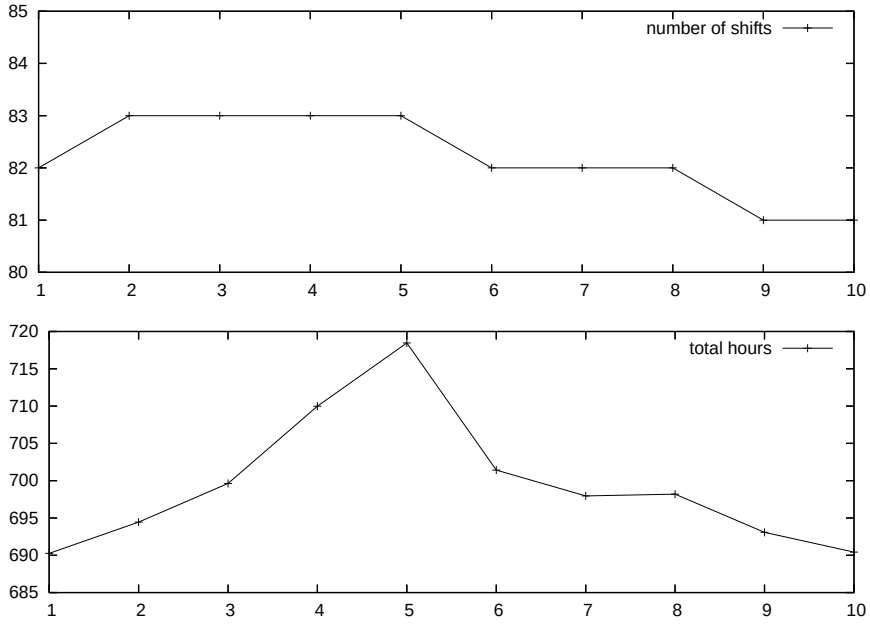

Figure 4.4: Unconstrained experiments for the GaS and Local Search loop. The x-axis shows the number of iteration of the loop; the y-axis show two measures of cost, number of shifts and payable hours. The objective is to first minimize the number of shifts, and then the total payable hours.

times; for example, with the default settings the branch and bound phase will stop as soon as it reaches a "good enough" solution, which might not be the best that can be achieved if all branches are explored. It is still interesting to see that, even with these settings, the loop has been able to generate better solutions (iterations 9 and 10) than the one obtained on the first iteration. This supports the intuition that considering time windows would lead to more efficient solutions.

Two: Constrained Schedule Size on GaS Phases

For the second set of experiments, we added a hard constraint on the calls to GaS, specifying that the number of shifts in the final solution must not exceed the one obtained in the previous call to the local search. The results are shown on Figure 4.5.

The results show that GaS now enforces the max-schedule-size condition. However, since there was no constraint issued on the total payable hours, the behaviour on that component of the cost is still erratic. It is easy to think of different ways of further constraining the GaS phase to control its behaviour. As an example, we

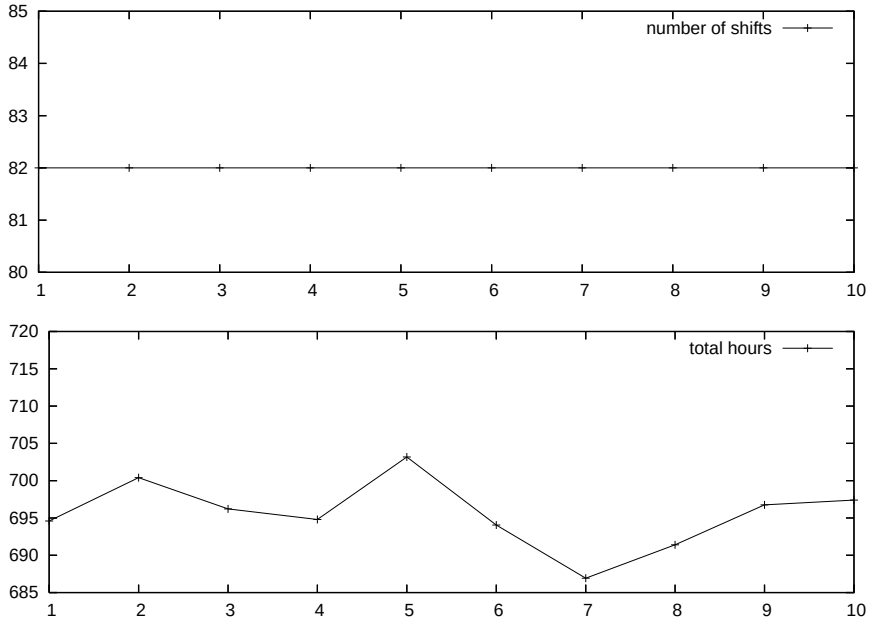

Figure 4.5: Cost-constrained experiments for the GaS and Local Search loop: schedule size and total payable hours vs. iteration number. While the constraint on schedule size succeeds in controlling the upper limit on size, the end result is worse than the one obtained when the constraint was not enforced. The algorithm is erratic when dealing with payable hours.

tried adding a constraint on the total payable hours; however, this seemed to render the solver unable to find a feasible solution. We don't have a final explanation for this behaviour. However, we can think of several possible reasons for this; among them:

1. The local search solution is present at some node of the branch-and-bound tree, but a limit on the number of nodes to expand prevents the algorithm from finding it; this limit is currently set to 5,000 nodes.

2. Because the GaS solver we are using includes a preprocessing phase, in which some relief opportunities deemed not useful are taken out of the problem, it may happen that the solution found in the local search (and every other solution with equal or better cost) is actually left out of the solution space considered by the branch-and-bound phase (no matter how many nodes are visited). Forcing `TrainTRACS` to consider all ROs is not viable, because of the resulting increase in problem size.

3. Because the generation phase is artificially constrained to restrict the number of shifts generated, it might happen that some of the shifts in the local search solution are not available to the next phase of GaS. We can circumvent this problem by explicitly adding those shifts to the pot available for the selection phase; in fact, the choice of shifts to add is not restricted to those in the final schedule $\mathcal{S}_i^{LS}$, but could also include shifts generated during the search process.

The reasons just described are quite independent from each other, and therefore it is possible that we have to tackle all of them before being able to get GaS to always find a solution that is better than or equal to the one obtained in the previous call to local search. However, tackling any of them would involve relaxing some heuristic rule which was originally added to make the execution time of the GaS phase feasible. This means that we must be intelligent in how to relax them, the way we solved the third problem being an example of that.

4.4 Conclusions

The experiments conducted in this chapter suggest that a local search approach is capable of overcoming the limitations of the Generate and Select approach on an extended scheduling model, which fully incorporates windows of relief opportunities. We developed different algorithms which show that cost improvements can be achieved; we also show that there is room for enhancing the robustness of the schedules. Better models need to be built to analyse the how cost and robustness can be tackled at the same time.

Experiments carried out in this chapter suggest many open areas. The local search phase should ideally be less conservative; on the other hand, constraints on shift structure mean that generating feasible perturbations of the current solution (i.e. altering the shifts in the current schedule) may be a hard task. These two observations suggest that it could be useful to accept infeasible intermediate solutions. This is studied in Chapter 6. The problem of cycling when including conditioning phases must also be properly studied. Also, while the test instances used did not contain schedule-level constraints, they should still be handled by the local search

phase in order to make this a general-purpose tool.

The loop-back version of the algorithm presented here is relatively simple, and our intuition is that much better results might be obtained with the right algorithms. These should be centered on better exploiting the information gathered during a local search phase for the next call to the GaS phase, which may include new shifts generated during the search, information about relief opportunities that were instrumental in producing cost improvements, etc. Feasible solutions obtained during the LS phase could even help in pruning the branch-and-bound tree on the following call to GaS.

Chapter 5

Exploiting Scheduling Constraints in the Generation Phase of GaS

5.1 Introduction

In the previous chapter we presented a two-phase approach to solving the driver scheduling problem with WROs, where we first run a GaS phase on a RoA approximation of the problem instance, to obtain an initial solution for a second phase which is a local search that works over the 1-minute WRO expansion. The search itself makes no explicit attempt at understanding the potential contribution of WROs to better schedules, hence treating all WROs equally when exploiting them. This would seem to contradict our observation that most of the ROs in a 1-minute expansion of WROs might be redundant, in the sense that eliminating them will not affect the theoretical optimal cost obtainable for that instance. Results on the experiments carried out on our first GaS + Local Search proposal suggest that only a small subset of the WROs is actually exploited during the search.

In this chapter, we propose a heuristic framework for exploiting WROs in the context of a GaS approach, where only those ROs within WROs that are deemed to

be potentially useful in getting lower-cost schedules are actually added to the model considered during the generation phase. To do so, we concentrate on identifying ROs within WROs that can be used to build new 'useful' shifts that were unavailable in the RoA formulation. The qualification of 'usefulness' arises from the fact that, although in general most of the ROs within WROs can be used to create shifts that are unavailable in a RoA formulation, many of these new shifts would only differ from shifts in the RoA formulation in the way WROs are covered – if we disregard work content inside WROs, most of the new shifts would actually have the same work content as an existing shift in the RoA formulation.

More formally, define spells sp_1, sp_2 to be *structurally equivalent* if, without considering work that occurs inside WROs, sp_1 and sp_2 cover the same pieces of work; if they cover different pieces of work, we say they are *structurally different*. By extension, shifts s_1, s_2 are structurally equivalent if for each spell in s_1 there is a structurally equivalent spell in s_2 and viceversa (and structurally different otherwise). In these terms, we claim that most of the shifts that can be created by using ROs within WROs are structurally equivalent to shifts in the RoA formulation. Our heuristic framework proposal is that of extending the set of ROs from the RoA formulation by adding only ROs within WROs that allow (or are likely to allow) GaS to generate shifts that are structurally different to any shift that can be generated in the RoA formulation.

5.2 Scheduling Constraints and Shift Boundaries

Driver scheduling can be seen as a way of partitioning vehicle work into a set of driver shifts (in the USA, for example, crew scheduling is often referred to as *run cutting*); looking at driver scheduling from this point of view, it can be noted that scheduling constraints give rise to limiting boundaries on the spells of work that can be 'carved out' of a given vehicle schedule. Figure 5.1 depicts such an example: given an RO r on vehicle v at time t, a maximum work spell length of x minutes will define a boundary in vehicle v at time $t-x$, such that any spell on vehicle v ending at r will satisfy this constraint if the spell starts at or after $t-x$, and will break the constraint otherwise. If $t-x$ falls inside a WRO w at vehicle v (but not at its arrival

time), then considering relieving inside w at $t - x$ leads to forming a spell which was invalid on the simplified, relief-on-arrival model. This would indicate that the

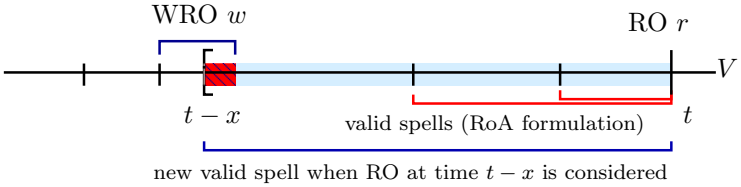

Figure 5.1: Looking at boundary conditions for the *maximum spell length* constraint. A maximum spell length of x restricts any spell ending at time t to start within the interval $[t - x, t)$. A new spell can be formed if the RO within WRO w at time $t - x$ is added to the model.

RO at vehicle v and time $t - x$ is a good candidate to be included in the extended model, because it allows for a new (structurally different) spell to be generated, and thus should allow for new (structurally different) shifts to be generated as well.

This kind of property is exhibited by many of the constraints in the driver scheduling problem, among them:

- maximum spell length
- the existence of feasible travel links between consecutive spells, sign-on and first spell, last spell and sign-off
- maximum shift spreadover
- mealbreak start/end times, lengths

5.3 Analyzing Individual Constraints

The approach outlined above can be applied to all constraints that are time-dependent, to produce a set of ROs within WROs to add to the ones derived from the RoA formulation. In its simplest form, the procedure can be tested on any given set $\mathfrak{I}$ of test instances $\mathcal{I}$ with the method described in Algorithm 2. The key step in this algorithm is step 4, where the scheduling constraint c is used to determine the set $W_{\mathcal{I}}$ of ROs to be added to $A_{\mathcal{I}}$. The implementation of this step will be specific to the chosen constraint c.

Algorithm 2 Testing the heuristic expansion of the RO set

given a scheduling constraint c that is time-dependent
for each instance $\mathcal{I} \in \mathfrak{I}$ of the scheduling problem **do**
    let $A_{\mathcal{I}}$ be the set of ROs from the RoA formulation
    compute $W_{\mathcal{I}}$, the set of 'useful' ROs within WROs in $\mathcal{I}$, according to c
    // run GaS on $A_{\mathcal{I}}$
    set $S_{RoA} := select(generate(A_{\mathcal{I}}))$
    // run GaS on $A_{\mathcal{I} \cup W_{\mathcal{I}}}$
    set $S_{WRO} := select(generate(A_{\mathcal{I}} \cup W_{\mathcal{I}}))$
    compare schedules S_{RoA} and S_{WRO}
end for

To test the applicability of this idea, we run the experiment outlined above by choosing the constraint that, for any given shift s to be valid, there must exist a feasible travel link between the end of each spell of work in s and the beginning of the next spell. The availability of these links is key in the process of joining individual spells into full shifts. The procedure for computing $W_{\mathcal{I}}$ is outlined in Algorithm 3, and it relies on a set of travel links computed by a module in `TrainTRACS` called `TRAVEL`, running on a 1-minute-expansion formulation of the problem. We run GaS

Algorithm 3 Computing $W_{\mathcal{I}}$ under the *feasible travel links* constraint

given an instance $\mathcal{I}$ of the scheduling problem
set $W_{\mathcal{I}} := \emptyset$
for each location l, relief opportunity $r \in A_{\mathcal{I}}$ **do**
    compute $t^{ea}(l, r)$, the earliest arrival time t for which there exists a feasible travel link between r and (l, t) according to $\mathcal{I}$
    if $(l, t^{ea}(l, r))$ is an RO within a WRO in $\mathcal{I}$ **then**
        set $W_{\mathcal{I}} := W_{\mathcal{I}} \cup (l, t^{ea}(l, r))$
    end if
end for

on $A_{\mathcal{I}}$ and $A_{\mathcal{I}} \cup W_{\mathcal{I}}$ using `PowerSolver`, an iterative GaS tool that is part of the `TrainTRACS` driver scheduling package. We use four recent real-life data sets from UK rail operators. The objective is to minimize schedule size first, then hours paid.

Results are shown in Table 5.1.

Results show a considerable decrease in cost (schedule size and/or hours paid) for the *Wessex* and *Wales* data sets, a relatively small improvement for *ThamesLink*, and an increase in cost for *InterCity*. This last result would seem contradictory, since $A_{\mathcal{I}} \subseteq (A_{\mathcal{I}} \cup W_{\mathcal{I}})$, and so every feasible schedule obtainable from $A_{\mathcal{I}}$ should also be obtainable from $A_{\mathcal{I}} \cup W_{\mathcal{I}}$; however, the reader should be aware that solutions in the test are not theoretical optimal solutions, but rather those obtained with a computer run of a (non-exact) software tool – indeed, the compromise between problem size and solution quality is precisely the driver for designing heuristics to restrict the set of ROs within WROs being added[1]. The fact that the proposed method generates new best-known solutions on three out of the four datasets is further proof that considering WROs may result in better schedules; moreover, the tool proposed can easily be applied in practice: a real-life scheduler, faced with the problem of assigning drivers to its operations, can always run `TrainTRACS` on both sets $A_{\mathcal{I}}$ and $A_{\mathcal{I}} \cup W_{\mathcal{I}}$ (or even have `TrainTRACS` do this automatically for him) and then simply choose the best solution from the two.

Results on the *Wessex* dataset are insightful. We investigated how many of the new ROs added by the procedure are actually functional in allowing for the 109-shift solution to be found. To do this, we replaced where possible all shifts that involve relieving at a newly-added RO with shifts covering the same work but relieving on arrival. Replacement is done in pairs (where one shift leaves the unit at that RO, and the other takes the unit at that point), so that all work –including that within WROs– remains covered after the replacement. Replacements that result in infeasible shifts were not carried out. The procedure allowed us to conclude that only *one* of the newly added ROs is responsible for the decrease in schedule size. It is still necessary to add more ROs to get the total number of hours paid down significantly; also, some of the ROs seem to play a part in allowing the selection phase to find a low-cost solution. This result backs our intuition that most of the

[1]It must be noted that although this heuristic may result in more ROs added than necessary for the solution space in the selection phase to contain an optimal-cost schedule equivalent to that of the full WRO formulation, finding a minimum-size set of ROs that provides such a guarantee will usually be a hard problem in itself

ROs derived from WROs are redundant, and it is worth designing algorithms that can sift ROs derived from WROs. Similar arguments have been stated when looking at individual ROs in the RoA formulation, where both early [66] and recent [56] work involves the use of heuristics to reduce the set of ROs used during the search.

In summary, the new method proposed is able to produce new best-known solutions for real-life instances of the TDS(W) problem. The proposal works a preprocessing stage, where instead of constructing the set of relief-on-arrival ROs $A_{\mathcal{I}}$ for $\mathcal{I}$ we generate an expanded set $A_{\mathcal{I}} \cup W_{\mathcal{I}}$. Therefore, from an implementation point of view, existing GaS systems should be easily amenable to this extension.

5.4 Exploiting a Group of Scheduling Constraints

Many of the constraints that apply to a driver scheduling problem instance can be exploited in a similar way to the *feasible travel links* constraint, to yield a set $W_{\mathcal{I}}$ of ROs within WROs which can be used to generate new, structurally different shifts. Therefore, a simple approach to exploit a group of scheduling constraints $C = \{c_1, \ldots, c_k\}$ is to build separate sets $W_{\mathcal{I}}^{c_j}$ for each constraint c_j, and then run GaS on the set $A_{\mathcal{I}} \cup (\bigcup_{j=1}^{k} W_{\mathcal{I}}^{c_j})$.

A first objection to this approach is that the resulting set of ROs available to the generation phase of GaS will grow quickly, exacerbating the effects seen in some of the instances of Table 5.1. The key for this behaviour in Algorithm 2 (and its extensions to sets of constraints) is the fact that, while a particular RO r is added because we have proof (or a suspicion) that there is a structurally new shift that depends on r being available, the generation phase will most likely also use r to generate new shifts that are structurally *equivalent* to shifts that could be obtained without using r.

At least two alternatives to adding ROs are available:

a. provide the generation phase with new, structurally different *spells* than those available under the RoA formulation

b. add *shifts* that are structurally different from those generated under the RoA formulation to the pool available in the selection phase

In either case, the pool of shifts available to the selection phase will not contain any structurally equivalent shifts.

A second observation about this proposal is that because several scheduling constraints impose the same kind of structural restrictions on shifts, they could be tackled within a single algorithm. Moreover, there are several scheduling constraints for which all structurally different shifts obtained by running the generation phase on $A_{\mathcal{I}} \cup W_{\mathcal{I}}$ can also be obtained directly by altering a specific shift $\mathcal{S}$ from the RoA pool, extending a particular spell in $\mathcal{S}$ so that it starts within an earlier WRO (but not at its arrival time).

Based on the previous two observations, we present a constructive algorithm (Algorithm 4) that implicitly exploits a group of scheduling constraints by adding new, structurally different shifts to an existing pool of shifts P (usually generated from a RoA formulation); these shifts are obtained from existing shifts in P, manipulating them in the way described in the previous paragraph.

To investigate what type of constraint this method can effectively exploit, we run Algorithm 4 (page 83) on an existing pool of shifts $\mathfrak{C}_{\mathcal{I}}$ generated with `TrainTRACS` on the RoA formulation. Using the notation in Algorithm 4, we then analyzed for every new shift s'_{wro} created with this method which constraints are violated by its structurally equivalent (infeasible[2]) shift s'_{roa} under the RoA formulation – this gives us an indication of which constraint we are implicitly exploiting when generating s'_{wro}. s'_{roa} is completely similar to s'_{wro} except that relief within w happens on arrival in s'_{roa}. To determine the constraint(s) violated by s'_{roa}, we feed it to the `CHECKER` on a RoA formulation of the problem instance; `CHECKER` returns only *one* of the constraints violated when the shift is infeasible; although knowing all reasons for a shift being infeasible would have been preferable, this still allow us to have a reasonably complete idea of the constraints being exploited.

In the following paragraphs we show three examples of how the new method is able to capture new structurally different shifts, and how they exploit scheduling constraints. For each example, we show on the right hand-side a new structurally different shift s'_{wro} obtained with this method, and on the left hand-side the shift s in the RoA pool that gave rise to it.

[2] s'_{roa} in infeasible, otherwise s'_{wro} would not have been generated by Algorithm 4

Algorithm 4 Adding structurally different shifts to an existing pool $\mathfrak{C}$

given an instance $\mathcal{I}$ of the scheduling problem, and a pool $\mathfrak{C}_\mathcal{I}$ of shifts obtained from $\mathcal{I}$ by running a generation phase on $A_\mathcal{I}$
for each shift s in $\mathfrak{C}_\mathcal{I}$ **do**
  for each spell sp in s, on vehicle v, starting at RO r **do**
    if the RO w immediately earlier than r on v is a WRO **then**
      let sp' be the 'extension to the left' of sp, so that it now starts in w, at its arrival time
      let s'_{roa} be the shift created from s by replacing sp with sp'
      if s'_{roa} is feasible **then**
        continue with next spell sp
      else
        if the start time of sp' within w can be changed to t so that it becomes feasible **then**
          set $\mathfrak{C}_\mathcal{I} := \mathfrak{C}_\mathcal{I} \cup s'_{wro}$, where s'_{wro} is built from s'_{roa} by modifying sp' to start at time t within w
        end if
      end if
    end if
  end for
end for

In the first example (Figure 5.2) the equivalent shift s'_{roa} in the RoA formulation turns out to be infeasible due to lack of travel opportunities – the earliest travel link from depot to the location of the WRO in vehicle 1752 containing the time 15:51 is 15:51, so starting the spell earlier than 15:51 in w is infeasible. This kind of situation wouldn't have been caught by Algorithm 3 since it only considers travel links during joinups, but not at sign-on/off, due to technical details on which feasible travel links are represented in the files generated by `TRAVEL`.

29	1752	**17:25**	19:30	29	1752	**15:51**	19:30
29	1754	20:38	21:48	29	1754	20:38	21:48
29	1755	22:46	23:57	29	1755	22:46	23:57

Figure 5.2: Results on Algorithm 4: new travel opportunity allows for an earlier sign-on. The shift on the right hand-side is created from that on the left hand-side by applying Algorithm 4.

In both the second and third cases (Figure 5.3, page 85), moving the start of the spell *sp* earlier makes the gap between the start of *sp* and the end of the previous spell in the shift less than 15 minutes. However, a hard constraint on the minimum joinup time of 15 minutes makes relieving on arrival infeasible in both cases. Notice how the joinup gap in both new shifts s'_{wro} (which benefit from the possibility of relieving later than arrival within the WRO) is exactly 15 minutes. Note also that the situation arising in these two examples can be linked to the potential of WRO formulations on robustness as described in Section 1.3 and Figure 1.2, where we suggest that the creation of certain buffers can be considered better in a WRO formulation.

Further analysis on the *Wessex* dataset show at least the following constraints to be exploited: *feasible travel links* (both for joinups and sign on/off); *maximum spell length*; *maximum shift spreadover*; *min/max joinup time*; at least some of the *mealbreak rules*. Although it would be difficult to determine precisely, initial analyses suggest that this mechanism is actually able to build most (if not all) of the relevant shifts that are not available under a RoA formulation because of violations to the constraints enumerated here.

It is worth noting that this extension to GaS is not as easy to implement as the one described in Section 5.3: the new shifts have to be generated at a point

71	1710	08:18	09:40	71	1710	08:18	09:40
71	1701	**10:24**	11:30	71	1701	**09:55**	11:30
71	1705	12:02	13:31	71	1705	12:02	13:31
71	1705	14:59	15:40	71	1705	14:59	15:40
73	1707	14:08	14:55	73	1707	14:08	14:55
73	1707	16:15	18:55	73	1707	16:15	18:55
73	1709	21:20	22:39	73	1709	21:20	22:39
73	1702	**23:05**	23:28	73	1702	**22:54**	23:28

Figure 5.3: Results on Algorithm 4: relieving later than the arrival time allows to enforce the hard constraint on *minimum joinup time*. The shift on the right hand-side is created from that on the left hand-side by applying Algorithm 4. In both cases, the joinup time between the affected spell and that preceding it is exactly 15 minutes, which is the minimum joinup time according to constraints for the dataset.

between the generation and the selection phases, so it cannot be implemented as a preprocessing stage. Moreover, the selection phase and the generation phase will need to be able to work on different sets of ROs, $A_{\mathcal{I}}$ and $A_{\mathcal{I}} \cup W_{\mathcal{I}}$ respectively (part of the matrix $\{a_{ij}\}$ in Equation 3.1 is now effectively related to pieces of work that are not available during the generation phase). Adapting existing software to accommodate for this, while keeping the system to produce operational schedules for real-life instances, is subject to further research.

5.5 Conclusions

In this chapter, we propose new mechanisms to address the perceived benefit of incorporating more domain knowledge when exploiting WROs.

We investigate how WROs may play a part in achieving more efficient solutions. WROs are looked at in terms of the new 'structurally different' feasible spells and shifts they may allow to be formed, compared to those available from the standard RoA formulation. This is first done by looking at individual scheduling constraints, and the role WROs may play in exploiting them. A second approach is then presented, in which so-called 'useful' ROs within WROs are detected by a constructive

procedure.

We derive a heuristic framework for adding selected ROs within WROs to a relief-on-arrival model (which can be seen as an extension to the usual generation phase of GaS). Tests conducted on real-life instances show substantial improvements in most of the test cases, and in a particular example, a reduction of more than 2.5% in the schedule size. Moreover, an analysis of this last instance shows that most of the improvement in schedule size can be attributed to a single new RO. This reinforces our view that most of the ROs within WROs will not be of use in obtaining more efficient solutions, and therefore can be eliminated from the model without affecting the quality of results, while at the same time showing that some of them are crucial to obtaining better schedules and therefore must be considered.

Although the tools presented here are relevant *per se*, there is much that can be done from here. To begin with, we would like to adapt `TrainTRACS` to fully implement our constructive proposal. At the same time, the algorithms and experiments presented on this chapter arise from an attempt at adding domain knowledge into the GaS + Local Search proposal. It would be important to investigate how the observations on the relationship between WROs and scheduling constraints may be translated into the GaS + Local Search framework; this includes the relatively straightforward modification to have the initial GaS phase run on a RoA formulation that is enhanced with selected 'useful' ROs, but also guided variants of the local search, where moves involving exploiting WROs and scheduling constraints are given preference.

	$A_{\mathcal{I}}$			$A_{\mathcal{I}} \cup W_{\mathcal{I}}$				
dataset	ROs	shifts	cost	ROs	shifts	cost	best	difference
Wessex	809	112	55,107	881 (2680)	109	54,279	$A_{\mathcal{I}} \cup W_{\mathcal{I}}$	-3 shifts
Wales	881	64	32,960	1010 (2535)	64	32,691	$A_{\mathcal{I}} \cup W_{\mathcal{I}}$	-269 mins
ThamesLink	722	91	48,807	776 (1581)	91	48,772	$A_{\mathcal{I}} \cup W_{\mathcal{I}}$	-35 mins
InterCity	547	115	59,437	582 (1293)	115	59,621	$A_{\mathcal{I}}$	+184 mins

Table 5.1: Execution of the experiment outlined in Algorithm 2, on the *feasible travel links* constraint. For each of the four test instances, $W_{\mathcal{I}}$ is created following Algorithm 3. Table shows, for both sets $A_{\mathcal{I}}$ and $A_{\mathcal{I}} \cup W_{\mathcal{I}}$, the total number of ROs for the generation phase, and cost of the solution obtained with `PowerSolver` (in schedule size and total minutes paid). For the $A_{\mathcal{I}} \cup W_{\mathcal{I}}$ sets, the number in brackets indicate the number of ROs in the full 1-minute expansion of WROs.

Chapter 6

Repair-Costing of Infeasible Solutions

6.1 Introduction

The hybrid approach in Chapter 4 is an initial attempt at explicitly exploiting WROs in the TDSW problem. While the search is able to improve on results obtained using GaS on the RoA model, the improvements are relatively limited, and the tool is outperformed by the second hybridization proposed in Chapter 5. In the remainder of this thesis, we develop and evaluate a more sophisticated local search scheme for the TDSW problem. This new scheme attempts at improving the first local search proposal by addressing two perceived limitations in its local search phase: difficulty in generating valid solutions in the neighbourhood of the current solution, and limited use of domain knowledge.

In this chapter we tackle the first limitation, and analyse the possibility of allowing infeasible solutions during the search, with particular emphasis in the problem of costing feasible and infeasible solutions. We propose a repair-based scheme, and we develop and compare several repair-costing heuristics.

In Chapter 7 we briefly study inefficiency in driver schedules. Finally, in Chapter 8 we integrate the ideas in Chapters 6 and 7 into a single framework, and evaluate its performance.

6.2 About Generating Neighbouring Solutions

In the proposal in Chapter 4, we enforce that all intermediate solutions in the search are feasible. This implies that all shifts in a schedule must satisfy all labour rules. Each move attempts to form new candidate shifts to replace shifts in the current solution. We implement the constraint that the resulting new shifts must be valid by first ignoring all but the most essential structural shift constraints while creating the shifts (for example, a driver is not assigned two different pieces of work at the same time), and once the shifts are built validating them using an external routine provided by `TrainTRACS` (the `CHECKER`). This approach has the advantage of automatically enforcing all the shift-level constraints enforced by `TrainTRACS`, and being very robust to changes in the specification of shift validity criteria. However, in practice it also means that during the local search only a tiny fraction of the shifts that are generated are valid according to the `CHECKER`; the majority of the solutions built are in fact discarded because they are not even valid.

Our experience in the experiments in Chapter 4 is that the search space is too sparsely connected under most neighbourhood moves if the search is restricted to feasible solutions. At the same time, the requirement for feasibility adds considerable complexity to the design and implementation of moves. Detaching that part of the complexity into a specific component would make the design of complex moves easier and cleaner. Moreover, if handling of infeasibility arising from moves could be achieved through a general-purpose mechanism, this would eliminate (or reduce substantially) the need for consideration of feasibility constraints from the process of move design. While our policy in Chapter 4 can be seen as establishing this kind of separation (if the call to `CHECKER` is taken as the component that handles infeasibility), in practice this results in too many potential solutions discarded. In this chapter we develop an alternative, general-purpose way of handling inefficiency in moves, which in effect allows for infeasible solutions during the search, but presents these solutions as feasible to the levels above. We call this method *repair costing*.

6.2.1 Search Space Connectivity

For a given instance of the train driver scheduling problem, Let $\mathfrak{F}$ be the space of schedules whose shifts are all structurally feasible, i.e. they satisfy the minimum physical constraints, such as not assigning a driver to cover two different pieces/vehicles at the same time. Then, calling `CHECKER` on each shift of the schedule defines a subspace $\mathfrak{F}_c \subset \mathfrak{F}$, $\mathfrak{F}_c = \{S \in \mathfrak{F} : \texttt{CHECKER}(S)\}$, where `CHECKER`$(S)$ is true if and only if all shifts in the schedule $\mathcal{S}$ are valid to `CHECKER`. Accordingly, we can think of two neighbourhood relations: the one defined by the moves themselves, $N(S)$, and its restriction to $\mathfrak{F}_c$, $N_c(S) = \{S' \in N(S) : S' \in \mathfrak{F}_c\}$. These are exemplified in Figure 6.1.

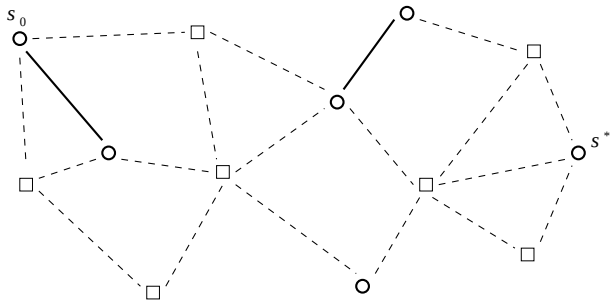

Figure 6.1: Two solution spaces, two neighbourhood structures. The set $\mathfrak{F}$ of all structurally feasible schedules can be partitioned into solutions in $\mathfrak{F}_c$ (represented by circles) and those in $\mathfrak{F} \setminus \mathfrak{F}_c$ (squares). The neighbourhood relation N is represented by dashed *and* solid arcs between solutions. Solid arcs represent the restricted neighbourhood N_c. Under N_c, $\mathcal{S}_0$ and $\mathcal{S}^\star$ are disconnected.

We are interested in obtaining the best possible solution in $\mathfrak{F}_c$. However, as suggested by Figure 6.1, it is likely that by allowing for the local search to work over the space $\mathfrak{F}$ (rather than $\mathfrak{F}_c$), connectivity between solutions in $\mathfrak{F}_c$ will increase, and this will result in better solutions in $\mathfrak{F}_c$ being reached. In this chapter we investigate and propose ways of costing infeasible solutions in a way that they are easily comparable with other solutions, both feasible and infeasible. In Chapter 8 we integrate these proposals into a local search framework that allows for certain intermediate solutions in the search to violate validity constraints.

6.3 Costing Feasible and Infeasible Solutions

Local search frameworks generally request that all intermediate solutions be feasible. Two reasons for this are that, first, there is usually no natural, fair way of comparing feasible and infeasible solutions in terms of the original cost function; second, the search might stray into the infeasible region and not be able to return to the feasible region, therefore rendering the rest of the search useless.

Many authors allow for infeasible solutions to be considered by augmenting the original objective function of the problem with a series of *penalty terms* [41]; through them, a solution becomes more unattractive (in terms of the augmented cost function) as the constraints get more violated. A first problem faced by this type of approach is that for many problems the original cost function may not be applicable to certain (or all) infeasible solutions. In the driver scheduling problem, we argue that for shifts violating structural constraints, the usual cost (defined as the number of hours between sign-on and sign-off) cannot be computed sensibly; for example, if a driver is assigned simultaneously to two vehicles during a period p, the sign-on to sign-off time would underestimate the number of man-hours driven (and even if p was counted twice, the 'extra' travel time will generally be impossible to compute). At a schedule level, a schedule not covering all of the work cannot be costed sensibly with the original cost function, as it would not account for uncovered work. Overall, this means that (even leaving the penalty terms aside) feasible and infeasible solutions may not be compared fairly. Most approaches involving penalty terms omit this discussion.

A second drawback of this approach is that the extended cost function mixes two separate evaluations of the solution that are extremely different in nature: on one hand, the quality of the solution in terms of the original problem, and on the other the 'amount' of infeasibility. The modeller must then tune the weighting of the original cost component against the amount of constraint violation, which usually involves empirically determined individual weights for each type of constraint violation that may occur. Some authors have proposed methods for adaptively adjusting these weights during the search [35]. A method that decouples these two evaluations, however, would give the modeller more control over the search process.

6.4 Using a Repair Heuristic to Cost Infeasible Solutions

6.4.1 Design Aims

Based on the limitations of the approaches based on penalty terms, we would like to find an alternative mechanism that allows for both feasible and infeasible solutions in the search space, and addresses the following issues:

1. how to compare feasible and infeasible solutions fairly, and if possible, in terms of the original cost function; this issue is especially critical when the current solution is on or near the boundary between the feasible and infeasible regions;
2. if and how to limit the 'amount' of infeasibility that is to be tolerated during the search process – if possible, in a way that is decoupled from objective-cost considerations;
3. the need to provide a feasible solution as a result of the search process.

Consistency in Costing Infeasible Solutions As stated earlier, we are not interested in infeasible solutions *per se*, but rather as a way of connecting different parts of the feasible space. Then, a cost function f_i that evaluates an infeasible solution $\mathcal{S}_i$ can also be thought of as a way of stating how promising $\mathcal{S}_i$ is, in terms of future feasible solutions that can be reached from it. Function f_i is then providing an estimation of the result of a particular process (for example, the quality of the next feasible solution in the local search, if it is continued from $\mathcal{S}_i$). This gives rise to the issue of *consistency*: if two solutions $\mathcal{S}_1, \mathcal{S}_2$ are equally promising, then we would expect $f_i(\mathcal{S}_1) \simeq f_i(\mathcal{S}_2)$; accordingly, if $\mathcal{S}_1$ is more promising than $\mathcal{S}_2$, we would expect $f_i(\mathcal{S}_1) < f_i(\mathcal{S}_2)$. Crucially, even if one is able to precisely specify what the function f_i should estimate, it might not be easy to evaluate how consistent f_i actually is; in particular, the process being estimated will most likely depend –among others– on the f_i itself.

Unbiased Costing of Feasible and Infeasible Solutions In a combinatorial optimization problem, a desirable property of any cost function is that solutions of

the same 'quality' are assigned the same (or very similar) costs. When designing a local search framework that allows both feasible and infeasible solutions during the search, we will in general want to be unbiased in relation to whether a solution is feasible or infeasible[1]. In other words, we want the 'quality' of a solution to be measured independently of whether the solution is feasible or infeasible (or of the amount of infeasibility).

It is then important that our proposed costing mechanism for feasible and infeasible solutions is not biased towards feasible or infeasible solutions. Even if we do not introduce any explicit bias when designing the method, any such method could nevertheless result in cost functions that are biased towards feasible or infeasible solutions. Therefore, the experiments on Section 6.6 will also look at the existence of biases as defined here. We will sometimes refer to this problem as that of an 'infeasibility step', since e.g. if the repair is inefficient, there might be a difference in cost in the limit between feasible and infeasible regions (against infeasible solutions) that is only attributable to that inefficiency in repairing.

6.4.2 The Proposal

Instead of relying on penalty terms, our approach to allowing infeasible solutions to be part of the search space is based on the use of a fast *repair heuristic*, which projects infeasible solutions into feasible ones. When considering the cost of an infeasible solution $\mathcal{S} \in \mathfrak{F} \setminus \mathfrak{F}_c$, the repair operator is first applied, obtaining a feasible solution $\mathcal{S}_r \in \mathfrak{F}_c$; the cost of $\mathcal{S}$ is then defined to be the cost of its repaired, feasible equivalent $\mathcal{S}_r$. In this way, the original cost function is naturally extended to infeasible solutions.

Previous work

Although repair costing is not frequently associated with neighbourhood search metaheuristics, the concept of repairing and repair costing has been studied considerably in the area of evolutionary algorithms, although using different terminology. In the

[1] A possible exception being that, if needed, we might at some point actively discourage infeasibility in order to return a feasible solution as a result of the search

context of evolution, the concept of repairing can be derived from a re-interpretation of the concept of learning, and the role of repairing during a search was originally studied in terms of the interaction between learning and evolution. In so-called *Lamarckian evolution*, learning during the lifetime of a phenotype is transferred back to its genotype, effectively feeding that information back into the evolutionary process. In the context of evolutionary algorithms, Lamarckian evolution can be equated to locally improving each new solution s generated during the evolutionary process, and replacing s with its repaired version. Local improvement can take the form of e.g. a greedy heuristic, or a short local search phase; if s was infeasible, local improvement would involve a repair procedure as well. Although Lamarckian evolution is nowadays considered an incorrect model of biological evolution, it is still widely used within the area of evolutionary computing.

A competing theory of evolution was proposed by Baldwin [5]. In this theory, good traits learnt during the lifetime of an individual affect its fitness, but those traits are not inherited by their offsprings. In optimization terms, this equates to using a locally-improved version of a solution s for the purposes of evaluating its cost, but crucially not replacing s with its improved version during the search process. A frequently-cited work on Baldwinian evolution was developed by Hinton and Nowlan [43]. Whitley et al. [80] compare Lamarckian and Baldwinian evolution in the context of genetic search, and suggest that Baldwinian evolution may be more efficient in reaching global optima. Mills and Watson [63] revisit the potential of Baldwinian evolution in escaping local optima and crossing fitness valleys.

As with Lamarckian evolution, local improvement of a solution s is in principle independent of s being feasible or infeasible – in biology this concept may not even exist. However, work by Ishibuchi et al. [47] contemporary to this thesis presents Lamarckian and Baldwinian evolution in the context of dealing with infeasibility arising during the execution of an evolutionary optimization algorithm. By comparing several variants of Lamarckian and Baldwinian evolution on a multi-objective 0/1 knapsack problem, they are able to conclude that Baldwinian evolution (i.e. that which relies on repairing for costing purposes only) performs better than Lamarckian evolution.

The exploration of the infeasible region as a means to connect feasible parts of the search space is also discussed in other applications or optimization techniques. A relatively recent metaheuristic called *Path Relinking* [37] makes an explicit consideration for traversing the space of infeasible solutions as part of the search procedure (*tunneling*). Ho and Gendreau [44] implement path relinking method for the vehicle routing problem; however, their implementation of tunneling involves the use of penalties in the cost function.

6.5 Designing a Repair Heuristic

As described in Chapter 4, our local search approach to the driver scheduling problem involves running a generate-and-select phase over a simplified model, obtaining a first solution $\mathcal{S}_0$, which is used as the starting point for a local search over the extended scheduling model. As a by-product of the first phase we obtain a set of shifts $\mathfrak{C}$ that covers (with huge redundancy) all of the drivers' work. This set provides for an efficient strategy to repair an infeasible schedule $\mathcal{S}$, which is based on first removing all infeasible shifts from the schedule, and then covering back any uncovered work by adding shifts from $\mathfrak{C}$ to $\mathcal{S}$. Kwan [57] and Li [59] have used similar strategies; however, they use the repair procedures to generate a feasible solution, rather than to cost an infeasible one.

This approach allows for computationally efficient implementations. Algorithm 5 depicts the framework shared by the heuristics we designed.

Algorithm 5 repair heuristic using the set $\mathfrak{C}$ of candidate shifts

1: given a schedule $\mathcal{S}$ containing infeasible shifts
2: remove all infeasible shifts from $\mathcal{S}$
3: **while** $\mathcal{S}$ does not cover all the work **do**
4: select a piece of work p_i which is not covered by $\mathcal{S}$
5: select a shift $s_j \in \mathfrak{C}$ such that s_j covers piece p_i (i.e. $a_{ij} = 1$)
6: $\mathcal{S} := \mathcal{S} \cup \{s_j\}$
7: **end while**

This is a greedy repair framework, since the repair is done incrementally and

with no backtracking. Different heuristics can be obtained by varying the criteria for selection in steps 3 and 4. Also, step 1 can be extended to consider removing not only infeasible shifts in the schedule $\mathcal{S}$, but also feasible ones, the intuition being that removing specific feasible shifts in the current schedule may make it easier for the repair heuristic to produce a good feasible schedule.

6.5.1 Data Structures to Support an Efficient Repair Heuristic

It is very important that the resulting repair heuristic is efficient. In the context of the framework described in algorithm 5, we add a pre-processing stage to the local search that takes the set $\mathfrak{C}$ and generates an efficient representation of how pieces of work in the current instance are covered by shifts in $\mathfrak{C}$. The result is an array `coveredBy`$[1 \dots |\mathcal{P}|]$, where $\mathcal{P}$ is the set of pieces of work for the current instance. `coveredBy`$[i]$ lists all shifts in $\mathfrak{C}$ that cover piece p_i, and is stored as a list of (pointers to) shifts. This representation makes it extremely efficient to:

- access all shifts covering a particular piece p: $O(1)$, or $O(k)$ to access all k shifts covering the piece;
- compare pieces of work in terms of the number of shifts that cover them: $O(1)$, since the number of shifts covering a piece p_i is the size of the list `coveredBy`$[i]$, which is stored/cached along with the list.

Additionally, during the repair we keep an array `powCovered`$[1 \dots |\mathcal{P}|]$ that counts the number of shifts covering each piece. This allows to find uncovered pieces easily (e.g. by traversing the array as if it were a list), and also to compare the number of uncovered pieces covered by different shifts efficiently. The choice of a count of the number of shifts covering a piece, rather than a boolean covered/uncovered description allows for an efficient update of `powCovered` as the repair progresses.

6.5.2 Selecting the Piece p to Cover

There are many different criteria that can be used to select the next currently-uncovered piece p to cover. We start by presenting deterministic criteria; after that,

we introduce randomized versions of them, where possible.

First uncovered piece The first uncovered piece in the list is selected. On each new selection, the search can be resumed from the last piece covered.

Least-covered piece It may be argued that it is better to start by covering the pieces that appear in the fewest number of shifts – by doing so, it is hoped that the last shifts to cover are those on which the algorithm has more choices. To implement this method, one can traverse the `powCovered` and `coveredBy` in parallel, keeping track of the uncovered piece (or one of the pieces) with the lowest `coveredBy` list size.

Start-of-run piece Experiments with the first heuristics show a problem in that the piece p selected would often be adjacent to a run of other uncovered pieces, which occur exactly before p on the same vehicle. This sometimes causes problems in that those adjacent pieces would then be left uncovered by the shift selected to cover p. If that run was short, it would later almost surely cause the need for an extra shift to just cover those few pieces in the run. A way to fight this problem is for example by restricting the choice of p to those pieces starting a run of uncovered pieces. This criterion is in principle orthogonal to the criteria presented before. In practice, we apply a slightly different method: we first find the least-covered piece p_1, and then select the piece p_2 that starts the run to which p_1 belongs (p_2 may eventually be p_1, if it starts one such run).

A symmetric criterion to the start-of-run piece is that of selecting end-of-run pieces (that is, pieces ending a run of uncovered work on a vehicle). Although the start-of-run criterion solves the problem at hand, start- and end-of-run criteria could be evaluated simultaneously during a repair (especially in a multi-run, randomized setting as described later in this section) to increase the likelihood of finding lower-cost repairs. Similar arguments of considering or traversing vehicle work in both directions have been proposed earlier in the literature, see e.g. [66].

Given a repair task, and since the algorithms to implement the methods above are very efficient, it makes sense to consider running a randomized version of them a

number of times and selecting the best repaired schedule from those runs, especially as it can help reduce the 'infeasibility step' as described in section 6.4.1. To see this, we note that both the framework and the selection heuristics proposed are greedy in nature. Greedy heuristics tend to be especially unreliable in terms of their approximation to the optimal solution, in the sense that it may be very close to the optimum for some instances but very far away from others. In order to minimize the number of situations where the repair heuristic results in a gap in cost between feasible an 'nearly feasible' solutions, randomizing the heuristics (and taking the best or average of a number of runs) could 'smooth' the results returned by them. In the following paragraphs, we sketch randomized versions of the methods above.

Random uncovered piece A random uncovered piece in the list is selected. This can be seen as a randomized version of the 'first uncovered piece' criterion.

Randomized least-covered piece When selecting an uncovered piece to cover, it is likely that there are a number of least-covered such pieces. In the randomized version, a set $\mathcal{LC}$ of least-covered pieces is built. Then, a random piece is selected from $\mathcal{LC}$.

Start-of-run of randomized least-covered piece In this method, we first randomly select a piece p_1 from the set of least-covered pieces, and then select the piece p_2 that starts the run of uncovered pieces to which p_1 belongs.

6.5.3 Selecting the Shift to Cover p

The methods we propose to select the shift to cover a given uncovered piece p share a greedy mechanism for selecting the shift to cover p: from those shifts that cover p, select the shift s that

a. maximises some measure of 'quality' of the uncovered pieces covered by s;

b. has the lowest cost.

If there is more than one such shift, we select the first on some pre-established order. We propose three measures of the 'quality' of the uncovered pieces:

1. Count the number of pieces covered by s.

2. For each uncovered piece p_i covered by s, measure the percentage c_{p_i} of shifts in $\mathfrak{C}$ that cover p. The overall quality of s is defined as $\sum_i 1/c_{p_i}$. This has the effect of giving a preference to shifts covering pieces that are 'hard to cover'.

3. Early experiments showed that the term $1/c_{p_i}$ might be too punishing for 'easy-to-cover' pieces (or, conversely, give too much preference to hard-to-cover pieces). An obvious alternative is to make that weighting less strong, e.g. by replacing that term with $1/\sqrt{c_{p_i}}$.

6.6 Experiments – Repair Efficiency

We are interested in investigating a number of properties of the repair heuristics outlined above. In particular,

1. **What their behaviour is when the schedule to be repaired contains a very small number of infeasible shifts.** This helps us understand better the behaviour of a search using such a repair heuristic in the limit between feasible and infeasible regions. We want specifically to verify that the heuristics do not result in any biases against infeasible solutions, which could for example happen if the heuristics are very inefficient, leading to what we have called an 'infeasibility step' in the limit between the two regions. The possible situations are depicted in Figure 6.2.

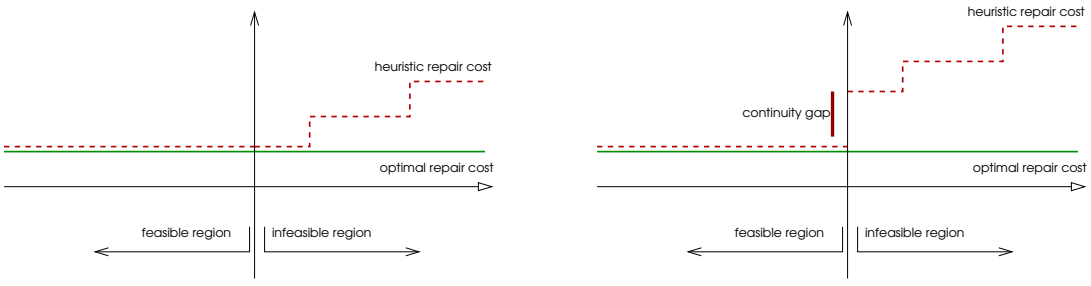

Figure 6.2: Possible scenarios for the behaviour of the repair-based cost function in the limit between the feasible and infeasible regions. The graphs represent a section of the search space with a uniform value for the optimal repair cost. left: no significant gap; right: significant gap.

2. **How their performance degrades as the number of infeasible shifts in the schedule to be repaired increases.** Because of inefficiencies in the repair mechanism, every repair heuristic will tend to introduce a bias against infeasible solutions as the number of infeasible shifts in a solution increases. We want to understand how (and particularly how fast) this occurs in our heuristics. The main consequence of a systematic bias against infeasible solutions would be that a lot of infeasible candidates are considered during the search only to be discarded; hence, knowing how performance degrades would give us the choice of controlling the amount of infeasibility tolerated to balance execution speed and capacity to accept new solutions with flexibility in exploring the infeasible region.

3. **How the different heuristics presented above compare.** This includes understanding how randomization affects them and what is the effect of increasing the number of runs on randomized heuristics.

To do so, we devised the following experiment. Given an instance of the driver scheduling problem, we first generate a schedule $\mathcal{S}$ using a generate-and-select approach; as a by-product of this stage, we also obtain a pool $\mathfrak{C}$ of candidate shifts. We then generate an infeasible schedule $\mathcal{S}_i^{(k)}$ by randomly removing k shifts from $\mathcal{S}$. This infeasible solution $\mathcal{S}_i^{(k)}$ is presented –along with $\mathfrak{C}$– to a repair heuristic, which returns a repaired schedule $\mathcal{S}_r^{(k)}$. We can then assert the efficiency of each heuristic by comparing $|\mathcal{S}|$ and $|\mathcal{S}_r^{(k)}|$: the smaller $|\mathcal{S}| - |\mathcal{S}_r^{(k)}|$, the better the heuristic is said to perform. This would correspond to a formulation of the scheduling problem where schedule cost is the number of shifts.

We carried out this experiment for a set of 6 different heuristic types. Of those, 3 where randomized; for those we tested different numbers of re-runs. Individual choices of heuristics or numbers of runs will be outlined later. In total, we evaluated 11 heuristics:

- random uncovered piece (rUP)
- least-covered piece (LCP) – pieces are scanned in *vehicle* → *time* order
- start-of-run LCP (SR-LCP)

- randomized SR-LCP – this heuristic was tried for 1, 3, 6 and 12 runs per instance (SR-rLCP-1,3,6,12 respectively)
- SR-rLCP, then select shift using linear weighting – tested with 3 and 12 runs per instance (SR-rLCP-LW-3,12)
- SR-rLCP, then select shift using square-root weighting – tested with 6 and 12 runs per instance (SR-rLCP-QW-6,12)

We used a real-life instance of the driver scheduling problem provided by Scottish train operator *Scotrail*; this instance contains approximately 1,000 relief opportunities, and the resulting set $\mathfrak{C}$ of candidate shifts contains more than 170,000 shifts; the resulting schedule $\mathcal{S}$ has $|\mathcal{S}| = 82$. We run each heuristic over a set of 300 schedules $\mathcal{S}_i^{(k)}$, for $k = 1, \ldots, 40$, effectively running each heuristic over 12,000 different inputs. Finally, for each heuristic h we computed the average of $|\mathcal{S}| - |\mathcal{S}_r^{(k)}(h)|$ for all 300 schedules repaired, for every value of k. For each heuristic we also added the results for every k, to obtain a single value E_h that allows us to compare heuristics quantitatively; lower values of E imply better performance. Figure 6.3 shows a scatterplot of all 11 heuristics tested. X-axis shows the number of runs per instance (1 for non-randomized heuristics). Y-axis shows P_h.

rUP and LCP The first surprising result is that the random-uncovered-piece heuristic performs almost as well as the least-covered-piece, even with no re-runs. We believe this is justified by the fact that the simplest version of LCP does not use the start-of-run criteria, therefore frequently causing the heuristic to leave gaps between the start of run and the piece selected, which in turn render it inefficient.

LCP and SR-LCP The explanation of the lack of added efficiency of LCP over rUP is backed by the comparison between LCP and its start-of-run variant, SR-LCP. The added performance obtained by adding the start-of-run criterion results in us adding this rule to all other heuristics tested.

SR-rLCP We then took SR-LCP as a first heuristic to test randomization. We did so by considering 3, 6 and 12 re-run versions, plus a 1 re-run version for control

against SR-LCP. Results show a steady increase in efficiency as the number of re-runs increases, suggesting the randomization approach is worth considering. The rate of increase in efficiency seems to decrease as the number of re-runs increases, possibly to a point where no further performance increases can be achieved by further re-runs. In the end, the choice of number of re-runs will be down to the specific weighting between speed and efficiency required. Finally, the control (SR-rLCP-1) has a value of P_h that is very close to that for SR-LCP, as expected.

SR-rLCP-LW The next two experiments test the addition of the criterion of weighting shifts by the quality of uncovered pieces they cover, where that quality is expressed in terms of the percentage c_p of shifts covering a piece p: $1/c_p$. Results show that the performance is increased by adding this criterion to SR-rLCP. Also, performance increases with the number of re-runs, like before.

SR-rLCP-QW Detailed analysis of the choices taken by SR-rLCP-LW show that the use of terms $1/c_p$ in the summation is inconvenient, as this results in too much importance being given to covering hard-to-cover shifts. More specifically, we would want each term in the summation to act both as an indication of quality, but also to signal that this shift is covering one more uncovered piece. As an example, when using a linear term $1/c_p$, covering a piece that is covered by 1% of the shifts would be worth as much as covering 10 pieces that are covered by 10% of the shifts.

In order to give less importance to hard-to-cover shifts, we propose the use of a new term $1/\sqrt{c_p}$. Results show a surprisingly big increase in performance; more in general, this also allows us to conjecture that selecting the right shift to cover the piece(s) selected is as important as selecting the piece to cover.

General results Figure 6.4 shows the average extra shifts in the schedules built by selected repair heuristics, $|\mathcal{S}| - |\mathcal{S}_r^{(k)}|$. The efficiency of these repair heuristics can be said to degrade linearly with the number of shifts removed, k. The particular criteria for selecting the piece p to cover, and how uncovered pieces are weighted when selecting a shift to cover p, make a difference in the rate at which performance degrades: while the average value of $|\mathcal{S}| - |\mathcal{S}_r^{(k)}|$ is 0 for all heuristics when $k = 1$, it varies between 8.83 and 5.96 (a 32% efficiency gap) when $k = 40$.

Results are also useful in another way: since the original schedule $\mathcal{S}$ can be regarded as optimal for practical purposes, this test would also indirectly provide an insight in how estimates obtained from a repair heuristic compare with the cost of 'optimally repairing' a solution. Figure 6.5 shows that for $k \leq 6$ the best two heuristics will return a schedule that is optimal the majority of times, since the average number of extra shifts is less than 0.5. Hence we argue that, at least in this case, these two heuristics can be considered unbiased against infeasible solutions containing 6 or less invalid shifts, or approximately 7.3% of the optimal schedule size. This could be used when determining a limit k in the maximum number of infeasible shifts allowed in a solution during the search. In that case the size of the optimal schedule wouldn't be known in advance, but the value of k could be adjusted dynamically based a current estimate of the optimal schedule size. As discussed earlier in this section, higher values of k could be tolerated during the search, but it is likely that the number of solutions that are rejected, as an artifact of the increased inefficiency of the heuristics to recover a good-cost feasible solution, will increase. We also argue that because the degradation of efficiency is basically linear, the price paid for tolerating more infeasible shifts will not be excessively high.

It is worth noting that although these tests provide some evidence on how the proposed heuristics behave, they are carried out in a different setting than the one in which they will be used. Here, *feasible* shifts are removed from the schedule; during local search, however, the shifts that will be removed are *infeasible*. In particular, the result on $|\mathcal{S}| - |\mathcal{S}_r^{(k)}| = 0$ for $k = 1$ will almost certainly not stand when the heuristics are set to work in the context of a local search algorithm.

6.7 Experiments – Computational Cost

Since the aim of our research into repair heuristics is to use them for costing every infeasible solution during a local search procedure, it is essential that the computational cost of executing the repair is reasonable. With this in mind, we have designed the repair heuristics to be linear or sub-linear in the main measures of size of this problem (including the size of the pool $\mathfrak{C}$ of candidate shifts used during the repair). However, it is important to study the actual running times (in CPU time or number

of operations carried) in real-life instances of the problem.

In this section we study the SR-rLCP-QW-1 repair heuristic, which is the heuristic used for testing the framework developed in Chapter 8. We first provide an introductory theoretical analysis of the order of a single call to the algorithm; we then conduct an empirical analysis of actual running times and number of operations carried out by running an *exploration* phase of the framework proposed in Chapter 8 on two of the instances used for testing in that chapter.

6.7.1 Theoretical Complexity

Let us define the following measures of size in the problem input:

n	number of shifts in the schedule being repaired
i	number of invalid shifts in the schedule
m	size of the pool of candidate shifts $\mathfrak{C}$ used by the repair
p	number of pieces of work to cover
k	average number of pieces per shift

Throughout this analysis we will assume no overcover in the input schedule to the repair heuristic. Although this is not true in many cases, the actual amount of overcover occurring in schedules is usually very small; an assumption of a fixed amount of overcover as a percentage of p could be used instead, and it would not affect the results of the analysis below. We note the following relations between variables:

- $i \leq n$; in general $i < n$, and thus $i/n < 1$;
- $k = p/n$; it makes sense to extend this estimate for k on all shifts in $\mathfrak{C}$;
- It is expected that $m >> p > n$.

We analyse the theoretical complexity using $O()$ notation; however where deemed appropriate we replace worst-order analysis with mean-based estimates (particularly when estimating pieces per shift, or number of shifts in $\mathfrak{C}$ covering a particular piece). We base the analysis on the pseudocode outlined in Algorithm 5 in page 95, and the data structures discussed in Section 6.5.1.

The repair algorithm consists of three main stages; first, removing all invalid shifts in the input schedule; second, initializing the `powCovered` array that records the number of shifts covering each piece in the current schedule (see Section 6.5.1); third, adding feasible shifts to the schedule until all pieces are covered. The first stage is $O(n)$, as it simply consists of iterating through the shifts in the schedule and reading a 'feasible' flag. The second stage is $O(p)$, since it can be achieved by iterating over the list of feasible shifts in the schedule, and for each shift s iterating over the pieces of work it covers, increasing the count in `powCovered`$[i]$ for every piece $i \in s$.

The third stage in the repair is the most costly, and it involves incrementally adding selected shifts from $\mathfrak{C}$ to the schedule being repaired – steps 3–7 in Algorithm 5. Looking at each operation individually:

- The outer loop is formally executed $O(i \times k) = O(i \times p/n)$ times, but this is a very pessimistic estimate as it implies that each shift added covers roughly only one uncovered piece of work. Based on the experiments in Section 6.6 we will assume that the number of shifts added during the repair is proportional to the number of infeasible shifts on the input schedule, hence this loop is actually executed $O(i)$ times.

- Step 4 (selecting an uncovered piece to cover) is $O(p)$ under SR-rLCP-QW-1.

- Step 5 (finding a shift to cover the uncovered piece) involves looping over all shifts in $\mathfrak{C}$ that cover that piece. This is done efficiently as we use the array `coveredBy`$[1 \ldots p]$ to record the list of shifts in $\mathfrak{C}$ covering each piece of work; in a local search setting, `coveredBy` would be computed as an initialization step before the start of the search, an would remain unchanged during the search. The number of shifts in $\mathfrak{C}$ covering a particular piece can be approximated by noting that each shift in $\mathfrak{C}$ covers approximately p/n pieces, and so m shifts cover $m \times p/n$ pieces, hence each piece is on average covered by m/n shifts in $\mathfrak{C}$. Hence step 5 is executed $O(m/n)$ times.

- For each shift s considered in step 5, a 'score' based on the uncovered pieces covered by s is carried out. For SR-rLCP-QW-1, the cost of computing this

score is $O(k)$ as it involves accessing `powCovered`$[i]$ for every piece $i \in s$. Overall, the cost of step 5 is $O(m/n \times k)$.

- Step 6 is $O(k)$ as it involves updating `powCovered` based on the shift being added to the schedule.

The overall cost of the third stage –and hence of the full repair algorithm– is then $O(i \times m/n \times k) \leq O(m \times k)$. Although the cost is not linear on the size m of $\mathfrak{C}$, it must be noted that the value k of average pieces per shift is usually bounded by a small value for all real-life instances of the problem, and can be treated as a constant, in which case the order of this heuristic is $O(m)$. An exception to this would be if WROs were modelled as sets of 1-minute-apart pieces of work, in which case the value of k can grow considerably. It is likely that a different algorithmic approach is needed in this case.

6.7.2 Empirical Analysis of Cost

To gain a better understanding of the expected computational cost of using the repair heuristics proposed in practice, we devised the following experiment. We take a real-life instance of the TDSW problem, and run one *exploration* phase of the local search framework proposed in Chapter 8. For the purposes of the experiment carried out here, it is sufficient to know that this *exploration* phase generates a big number of candidate infeasible solutions, which are then costed using the repair heuristic SR-rLCP-QW-1. For each test, we compute the average CPU time (in milliseconds) for each call to the repair heuristic, and the input sizes as enumerated in Section 6.7.1, taking average values wherever appropriate. We also measure the average total number of accesses to the `powCovered` array; this is a good proxy for the overall cost of the loop 3–7. We also compute the average number of times that the loop of steps 3–7 is executed on each call to the repair, to validate our claim that this is roughly similar to the number i of invalid shifts in the input schedule.

Results for two experiments on the *InterCity* and *Wessex* datasets (which are later used for the experiments in Chapter 8) are summarised in Table 6.1 (page 112). Tests were conducted on a computer based on an Intel Core2 4300 chip running at 1.8GHz, with 2GB of RAM. We observe that:

- The actual CPU time is extremely small (0.8 and 2.1 msec for the *InterCity* and *Wessex* datasets), making these heuristics completely suitable for integration to a local search scheme in terms of CPU time per call.
- As discussed earlier, the actual number of times the loop 3–7 is run in practice seems to be very close to the number of infeasible shifts in the schedule (1.14 and 1.27 times i for the *InterCity* and *Wessex* datasets).
- the average total accesses to `powCovered` per call to the repair heuristic is well below m, the size of the pool of candidate shifts $\mathfrak{C}$. This suppports our claim that the repair heuristics as proposed and implemented are linear (or sublinear) in m.
- Finally, it is important to note that the average number of infeasible shifts in the schedules input to the repair heuristic is considerably high (19% and 13% of n for the *InterCity* and *Wessex* datasets), so scalability to big values of i is already considered in these tests.

In summary, the computational experiments conducted in this section provide conclusive evidence that the repair heuristics proposed in this chapter are suitable for use within a local search framework. Moreover, since the instances used represent realistic instances of the TDS problem in terms of size and number of infeasible shifts in the schedule, scalability to big values in input size is proven in these tests, with the exception of k, for which an approach where WROs are modelled as sets of 1-minute-long pieces of work might increase the value of k to a level not covered by the experiments conducted in this section.

6.8 Conclusions

In this chapter we present the case for allowing the search to violate certain feasibility constraints (currently encompassed in the call to the external `CHECKER`). Working with feasible and infeasible solutions gives rise to the problem of costing those solutions, and particularly how feasible and infeasible solutions are compared. We propose a method of costing infeasible solutions through a fast repair heuristic, which projects infeasible solutions onto feasible ones. The heuristics developed

exploit the set of candidate shifts $\mathfrak{C}$ derived from the generation phase of GaS to re-cover work that was covered by infeasible shifts. We propose a number of criteria for selecting the next piece to cover and the shifts used to cover that piece, and we evaluate and compare the resulting heuristics. We also analyse the degradation in performance as the percentage of infeasible shifts in a schedule increases, as a first look into their use within a repair-costing framework in a local search. We estimate a theoretical order of execution for the best-performing repair heuristic, and conduct experiments on real-life instances of the problem to obtain empirical evidence about its running cost; we conclude that the heuristics are suitable to be integrated into a local search scheme in terms of their computational cost.

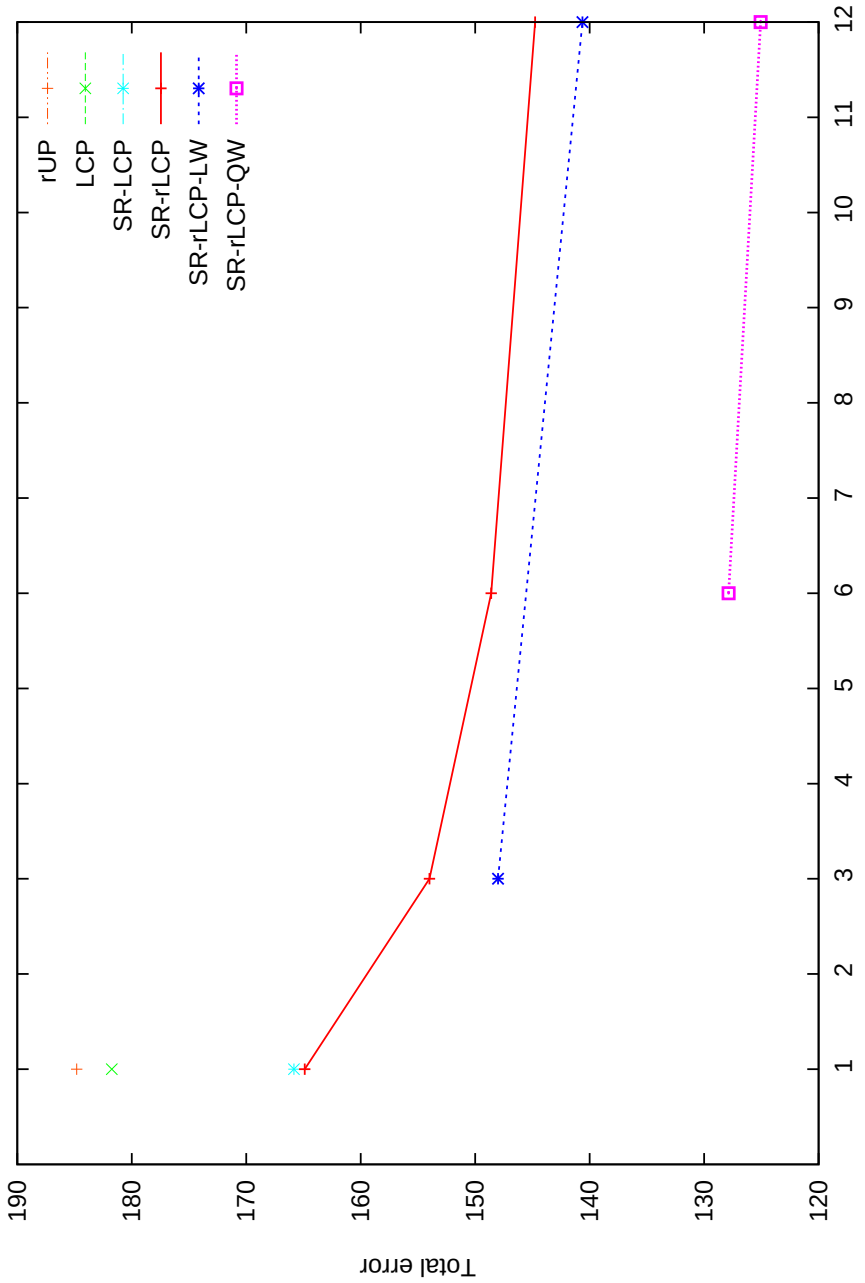

Figure 6.3: Comparison of repair heuristics: total error E after 12,000 repairs per heuristic. Lower error values imply better performance.

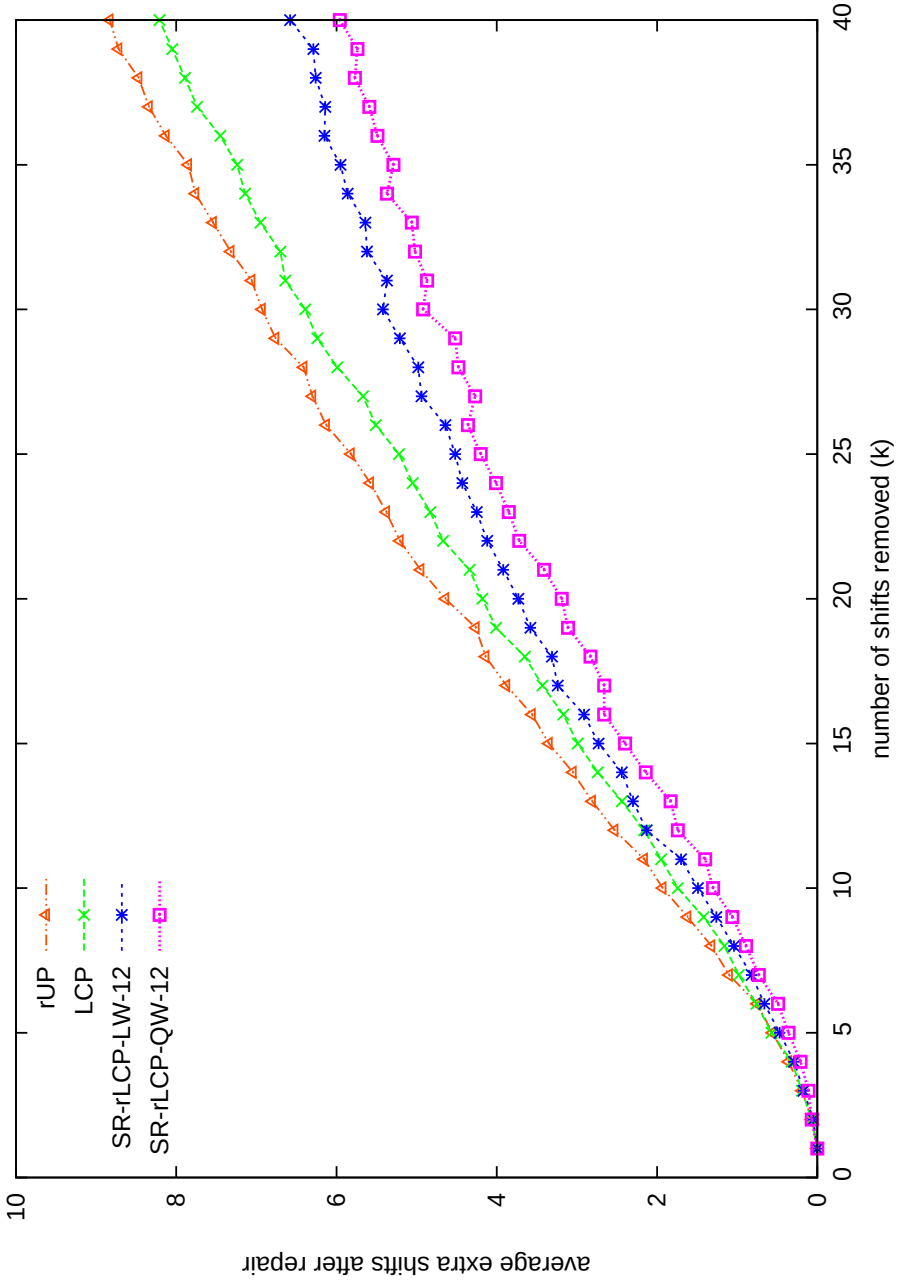

Figure 6.4: Comparison of repair heuristics: average extra shifts after repair. Degradation in performance is almost linear on the number k of shifts removed from the original schedule.

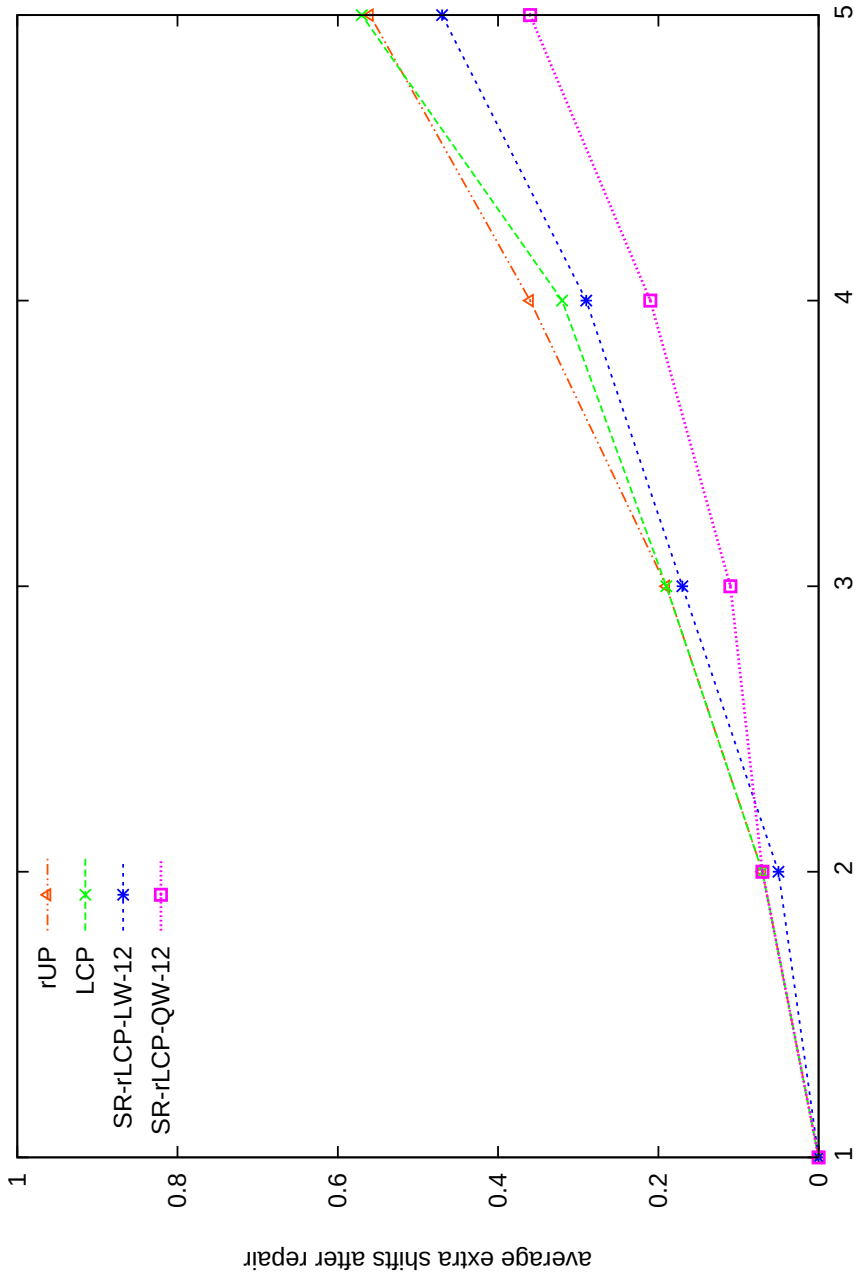

Figure 6.5: Comparison of repair heuristics: average extra shifts after repairing, $k \leq 5$.

dataset	*InterCity*	*Wessex*
p	475	719
m	19,028	87,173
number of calls to repair heuristic	2,161	3,015
average running time per call (msec)	0.8	2.1
n	116	117
i	21.7 (0.19 n)	14.7 (0.13 n)
k	4.1	6.1
average iterations on the loop (3–7)	24.8 (1.14 i)	18.6 (1.27 i)
average total accesses to `powCovered` per call	8,897 (0.47 m)	32,578 (0.37 m)

Table 6.1: Results for the empirical running-time analysis of the SR-rLCP-QW-1 repair heuristic. Numbers in brackets indicate the relation to other variables.

Chapter 7

Inefficiency in Driver Schedules

7.1 Introduction

Other than through specific mechanisms for exploiting WROs, the local search phase in Chapter 4 makes very limited use of domain knowledge. Our intuition is that it may be desirable to make the search explicitly biased towards desirable structural attributes (or against perceived 'inefficiencies'). In this chapter we briefly characterise inefficiency in driver schedules, with the aim of incorporating the rules derived from these analyses into a local search scheme with active biases against these inefficiencies.

In the context of solving a combinatorial optimization problem, it is tempting to consider whether it is possible to 'explain' the reasons for a given solution being sub-optimal. However, it is likely that a truly complete explanation would somehow need to describe how to generate an optimal solution instead, and this is impossible in general, at least for NP-complete or NP-hard problems under the premise that $P \neq NP$. In fact, it can be argued that most iterative optimization techniques (including local search metaheuristics but also ILP techniques such as branch and bound) acknowledge this fact and instead work on finding, or attempting to find,

partial explanations for sub-optimality. These 'partial explanations' can take different forms, for example a cost-improving move in a hill-climbing local search, or the discarding of a node/subtree in branch and bound. In hill-climbing local search frameworks, and particularly when using general-purpose moves and neighbourhoods, we argue that the inefficiency exploited by a move is mostly determined *a posteriori*, since it is only after the evaluation of the cost of the new solution that the move will be accepted or rejected. An inefficiency that is determined *a priori*, on the other hand, would guide (or bias) the search in determining the next move(s) to be executed. Determining inefficiencies *a priori* and *a posteriori* are not mutually exclusive choices, but can be integrated into a common framework.

7.2 Identifying Local Inefficiencies

Identifying desirable and/or undesirable attributes in solutions to an optimization problem is a good starting point to incorporate domain knowledge into an algorithm for solving that problem. In this section we revisit the problem of driver scheduling with the aim of extracting indicators of inefficiency that may help guide a local search metaheuristic in constructing the next solution in the sequence. For the reasons described in the introduction, we do not attempt to provide a complete explanation for sub-optimality. Instead, we work on identifying attributes or 'parts' of a solution that are inefficient. Notice that this approach is local in nature, and it may be necessary for a solution to contain an inefficient component in order to be efficient overall. This dichotomy is at the root of all combinatorial optimization problems, and is present in most optimization techniques.

7.2.1 Inefficiency at the Shift Level

Our first approach to identifying local inefficiencies is to look at an individual shift s in the current schedule, and study the relation between the amount of driving time (dt_s) and non-driving time (ndt_s). Intuitively, the efficiency of a shift s will increase when dt_s increases or ndt_s decreases. Standard ways of expressing this would be to maximize one of:

- dt_s/ndt_s
- $dt_s/(dt_s + ndt_s)$, or $dt_s/spreadover_s$
- $dt_s - ndt_s$

It may also make sense to limit the considerations to maximizing dt_s or minimizing ndt_s: these still indirectly look at the relation between dt_s and ndt_s because shifts are constrained in total length, i.e. $dt_s + ndt_s = spreadover_s < max\_spreadover$. We find that it is usually more intuitive to identify excessive non-driving time as an inefficiency, hence implicitly looking at minimizing ndt_s. In any case, it is possible to consider more than one of these criteria simultaneously, for example by checking them against pre-determined bounds and triggering specific actions if one of these bounds is exceeded.

In the following paragraphs, we decompose a shift into atomic components of driving and non-driving time and categorize the types of non-driving time, with the aim of finding ways to reduce those times. We start our analysis by considering a typical 3-spell shift s, with the following breakdown:

a: *sign on* at depot 3

b: *travel* to location 5

c: *drive* vehicle 7 for 2h15m, ending at location 8

d: *travel* to canteen

e: have a 30-minute mealbreak/PNB

f: *travel* to location 6

g: *drive* vehicle 2 for 1h42m, ending at location 2

h: *wait* for 20 minutes

i: *drive* vehicle 9 for 1h35m, ending at location 4

j: *travel* to depot 3

k: *sign off*

In this example, only stages c, g and i are strictly driving time; all other stages (traveling, waiting, mealbreaks, and signing on and off) are non-driving time. We look at these types of non-driving time in more detail, and briefly illustrate how inefficiency may arise in them; we present them in what we believe is their order of importance in terms of the role they play in determining the overall efficiency of a schedule.

1. **travel time**: drivers frequently need to travel from one location to another in the network; for example, to start the next spell of driving work (on a different vehicle). This may also happen between sign-on and the first spell of work, or before sign-off. Inefficient distribution of work among shifts in relation to the resulting required travel time is one of the major causes for inefficient schedules. In shift s, travel time is incurred in stages b, d, f and j.

2. **idle (slack) time**: it occurs when the shift includes some time on which the driver is neither driving, nor traveling, nor having a mealbreak. For example, it may happen that the travel time required to connect two consecutive spells of work in a shift is shorter that the time that is physically available, and it can't be used as a PNB. In this case, the driver will simply have nothing to do during that time, but will still be paid for it. It can also happen that some breaks between vehicles are longer than what is actually required by the labour agreements, which from the point of view of the operator is unnecessary paid non-working time. In shift s, this happens in stage h. It is likely that not all 20 minutes are 'wasted', since there may be a minimum joinup time (the time allowed to a driver between two consecutive spells) mandated by the labour agreement; however, even in this case it may be argued that a shift with fewer spells would incur less total absolute joinup time, since the total minimum joinup time is proportional to the number of spells in a shift.

   A second (and less intuitive) example arises when a shift cannot start less than a certain length of time t from the beginning of the first mealbreak. When a shift is breaking this rule, it is not uncommon to solve this by *padding* it with some minutes of non-driving time at the beginning of the shift, so that the mealbreak will start t minutes from the start of the shift.

3. **mealbreak/PNB time**: although PNBs are usually mandatory, the labour agreement frequently offers some options regarding the distribution of PNB time over the length of a shift; for example, a 3-spell shift may have one or two mealbreaks. It may be possible that the different options result in a different total amount of PNB time (say, one 30-minute PNB against two 20-minute PNB, with 30 vs 40 total PNB minutes). In these cases, it can be argued that a shift whose PNBs are distributed in a way that is not optimal regarding total PNB time is inefficient.

4. **sign-on/off time**: while the sign-on/off times are usually viewed as fixed, it may happen that the allowances for sign-on/off vary according to the depot where the procedure is carried out; in these cases, choosing the right depot for a shift may result in decreased sign-on/off times.

7.2.2 Inefficiency at the Schedule Level

The categorization above covers all of the instances of shift-level non-driving time in our formulation of the driver scheduling problem. However, because in our model we allow for pieces of work to be covered by more than one shift in a schedule, some of the time we categorized as driving time in the previous analysis may actually turn out not to be driving time when the shift is integrated into a schedule. As an example, consider shift s above; if another shift s' in the schedule was also assigned to drive vehicle 2 during the same period as s, then only one of the two can be actually driving vehicle 2 during that period, and hence the actual non-driving time for either s or s' would include time which we have categorized as driving time.

In an instance of k shifts (over)covering the same piece of work, rather than arbitrarily considering the overcovered period as driving time in one of the shifts, and as non-driving time on the other $k-1$ shifts, it seems convenient to classify this as a schedule-level inefficiency. We therefore identify our first schedule-level inefficiency:

5. **overcover driving time**: this occurs when two or more shifts in a schedule are assigned to cover the same piece of work.

7.2.3 Criticism and Extensions

While categorizing or detecting inefficiencies at a shift level can be easy, this approach has its drawbacks. One such drawback is suggested by the perception that, in many cases, an inefficiency in a shift s is actually forced by the presence of another shift s' in the schedule. This is especially frequent in methods that involve constructive steps/heuristics. As an example, consider a shift s that contains a spell that starts 5 min after the start of a certain vehicle block. If the schedule contains no overcover, this means that there is another shift s' which covers that 5-minute spell left uncovered by s. In constructive approaches, this situation will usually arise when s is added to the schedule before s'. With our categorization above, s' will probably be classed as inefficient (for example using the metric $dt_{s'}/ndt_{s'}$). However, most human schedulers would agree that in order to solve this local inefficiency, it is s that must be modified first.

One solution for this particular situation is to add a new shift-level inefficiency rule for shifts containing a spell that starts or ends too close to the start/end of a vehicle block (but not exactly at the start/end of that block). However, this approach has the inconvenience that the new inefficiency is not expressed in terms of driving or non-driving time, and therefore is difficult to compare or integrate with other categories.

Moreover, we argue that this situation is or can still be covered by driving/non-driving time considerations. We first observe that shifts s and s' are adjacent, in the sense that they share a relief opportunity (or WRO). In our first local search proposal in Chapter 4, most moves would consider both a shift and its adjacent shifts. For example a piece-swap would reassign a piece of work to an adjacent shift – and in the example above this move might solve the problem by assigning the 5-minute spell in s' to s.

In the situation of using a repair heuristic like those proposed in the previous chapter, consider the case where shift s' is invalid because the 5-minute spell is not allowed by a (hard or soft) constraint. The standard approach of removing s' wouldn't work, since any shift added to replace s' would either be invalid on a 5-minute spell, or otherwise include overcover, but s wouldn't be modified. However, there is a simple extension to our repair mechanism that solves this problem:

> **rule for repairing**: given a schedule S containing a set I of invalid shifts, determine the set A_I of shifts in S adjacent to shifts in I; then, when repair-costing S, remove all shifts in $I \cup A_I$ from S before repairing.

In the example above, this rule would remove s since it belongs to I, but would also remove s' because it belongs to A_I, therefore giving the repair cost an opportunity to modify both s and s'. Although in principle tests in the previous chapter show a linear degradation of repair heuristics with the number of shifts removed, it must be noted that the rule above is different in that the additional shifts removed by the rule are not randomly selected (as were the ones in the experiments in Chapter 6), and hence the results on degradation may not apply to this rule – on the contrary, we expect that the addition of this rule will help achieve better repair costs.

We believe that in most (if not all) situations where the initial perception is that there is a need for a new inefficiency rule, careful analysis should show that the rules introduced in this chapter regarding inefficient use of driving/non-driving time or overcover are enough to detect and tackle those situations.

Chapter 8

A Local Search Approach with Support for Exploration of Infeasible Solutions and Exploitation of Local Inefficiencies

---

In this chapter we integrate the tools and ideas developed in the previous two chapters, and present a new hybridized (GaS and Local Search) approach for the TDSW problem. We implement the proposed framework and evaluate it using a set of real instances of the problem, varying the starting solution and the formulation. We also consider alternatives to the framework proposed and evaluate them.

8.1 The Proposal

The proposal presented in this chapter is built on that of Chapter 4, and shares the skeleton of first running a GaS stage over a RoA model, followed by a local search stage over the WRO formulation. However, the local search stage now iterates

over two distinct phases, the *standard* phase and the *exploration* phase. During *standard* phases, the search is tied to all feasibility constraints, as is the case with the local search in Chapter 4. When a *standard* phase reaches a local optimum (or a similar criterion, e.g. a certain number of move trials without cost improvements), a move tailored to correct a specific inefficiency in the current solution is executed. This move is more complex in design than other moves in the search, but does not guarantee a feasible solution – in some cases, it doesn't even guarantee that all work is covered in the resulting schedule. After this move, the search switches to an *exploration* phase, where infeasible solutions are allowed, and which relies on repair costing to compare (feasible and infeasible) solutions. When a termination criterion is met, the current solution is repaired (if infeasible), and the search resumes in *standard* mode. A global termination criterion dictates the end of the search. The proposed framework is illustrated in Figure 8.1.

In combination, the execution of the inefficiency-reducing move and the subsequent *exploration* phase act as a diversification phase that 'bridges' parts of the feasible search phase through both intelligent exploration of the infeasible search space and use of domain knowledge. In this proposal, the inefficiency-reducing move and the *exploration* phase are linked inextricably: allowing for infeasible solutions during the *exploration* phase makes it possible to design a move that modifies the current solution in a way that the *standard* phase cannot; at the same time, the *exploration* phase relies on the inefficiency-correcting move to suggest a different direction in the exploration of the search space.

8.1.1 The *Standard* Phase

This phase is similar in spirit to the local search phase in Chapter 4, and it shares the same set of moves/neighbourhoods. Solutions are required to be feasible to be considered, and only cost-decreasing (or non cost-increasing) solutions are accepted. Moves exploit WROs in the same two-stage way as the local search in Chapter 4.

0.	set $i := 1$; run *Generate and Select* on a RoA formulation, obtaining a feasible solution $\mathcal{S}_i^{GaS}$ and a set $\mathfrak{C}^{RoA}$ of feasible shifts
1.	run a *standard* phase on a WRO formulation, starting with the current (feasible) solution, generating a new feasible solution $\mathcal{S}_i^{std}$
2a.	execute the inefficiency-correcting move on $\mathcal{S}_i^{std}$, leaving a possibly infeasible new solution $\mathcal{S}_i^{inf}$
2b.	add rules to prevent the next phase from undoing the inefficiency correction
3	run an *exploration* phase starting with the current (infeasible) solution $\mathcal{S}_i^{inf}$ and using the pool of shifts $\mathfrak{C}^{RoA}$, generating a new feasible solution $\mathcal{S}_i^{exp}$
4	remove rules to prevent undoing of the inefficiency correction; set $i := i + 1$; go to step 1

Figure 8.1: Scheme for the extended GaS and Local Search proposal.

8.1.2 The Inefficiency-Correcting Move

The primary aim of this move is to remove or correct a perceived (local) inefficiency in the current solution. At the point of executing this move the current solution is expected to be a local optimum for the previous phase. The intuition is that this local inefficiency might be preventing the local search to progress further, and cannot be removed by the search because either

1. the changes needed make it difficult to modify the solution in a way that the inefficiency is removed and the resulting solution is still feasible, or
2. any local modification of the solution to remove the attribute results in a higher-cost solution, which is then rejected by the search, or
3. both of the above

Although there are many approaches in the literature to detect recurring attributes in a sequence of solutions, in this proposal we don't attempt to discover these, and instead concentrate on the current solution when looking for inefficiencies. This is discussed further in Section 8.2.1.

Cycling As seen in Section 4.2.1 when attempting conditioning phases, correcting a local inefficiency will almost always result in an increase in the cost function (at least in the very short term). As with any local search approach in which the cost is allowed to increase, this creates a problem of cycling. Hence, inefficiency-correcting moves have to be designed taking into account this potential cycling problem. In this proposal, we put an emphasis in designing the moves in a way that it is easy to forbid the inefficiency that has been corrected from being undone by the following search phase(s), taking also into account that the phase following the move is an *exploration* phase. Details on the specific mechanisms are shown in Section 8.2.1.

8.1.3 The *Exploration* Phase

This phase starts at a point where an inefficiency-correcting move has been executed, which means that the initial solution is likely to be infeasible. During this phase, both feasible and infeasible solutions are considered, and all solutions are compared through a repair-cost function, which relies on the pool of shifts $\mathfrak{C}$ generated during the GaS phase to execute the repair. The repair heuristic first removes all infeasible shifts from the schedule (and in some configurations can also remove shifts adjacent to those infeasible shifts), and then covers the work left uncovered using shifts selected from $\mathfrak{C}$.

New solutions are accepted if their repair cost is lower than or equal to that of the current solution. Modifying the scheme to accept other solutions as well is straightforward, and could be used for example to increase the distance between the last solution in the previous *standard* phase and the resulting solution of the current *exploration* phase.

Repairing and WROs Because the generation phase in the GaS is run on the RoA model, all shifts in $\mathfrak{C}$ are such that all of their spells start and end at arrival

times within (W)ROs. This creates a problem for the repair heuristics proposed in Chapter 6, in that requiring the WROs to be fully covered by the repair may result in very inefficient repairs. To see this, consider an instance of the TDSW problem containing a vehicle v with two pieces of work p_1, p_2, and a WRO w in between the two pieces, as depicted in Figure 8.2. Now consider an infeasible schedule $\mathcal{S}^{inf}$ in

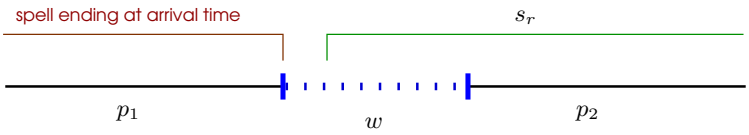

Figure 8.2: An example of the limitations of the repair mechanism on WRO instances. If WRO w is only partly covered by a shift s_r after removing infeasible shifts from a schedule $\mathcal{S}^{inf}$, a repair mechanism relying on a set shifts obtained from a RoA formulation will need to (over)cover piece of work p_2, which is inefficient.

which (after removing all infeasible shifts), p_2 is covered but p_1 is uncovered, and the shift s_r covering p_2 only covers part of WRO w. All shifts covering p_1 in $\mathfrak{C}$ either cover p_2 as well, or do not cover the WRO w. Therefore, strictly speaking the repaired schedule would only be valid if the shift chosen to cover p_1 also covers p_2. However, taking the approach of covering p_2 would mean overcovering most of p_2. When taken to a whole schedule, this approach is likely to make the resulting solution too inefficient – its repair cost too high. We present an alternative way to tackle this problem in Section 8.2.2.

Moves as a shift-selecting mechanism The repair heuristics in use are based on removing all infeasible shifts and then re-covering all uncovered work with feasible shifts. At the same time, our experience is that most of the times the move preceding the repair is such that all the shifts perturbed by the move become infeasible after the perturbation. Together, this means that most of the times *all* the shifts perturbed by a move during an *exploration* phase will be removed by the repair mechanism (and replaced with other shifts). In turn, since all moves in our proposal work by re-distributing the work in a subset of the shifts in the schedule, one might be tempted to conclude that the only effective role of the move executed before the repair is that of selecting a subset of shifts to remove from the current solution, leaving the repair to find a better-cost replacement for those shifts: since all shifts perturbed

become infeasible, the perturbation itself is lost to the repair.

The reasoning above would indeed be correct if the repaired solution were used as the next point in the search, and the consequences would be significant – for example, it might make sense to consider replacing the perturbation-based set of moves with e.g. some sort of simple subset-selecting mechanism to determine the shifts to remove. However, we note that although the repair will replace those infeasible shifts generated by the move, this replacement is considered while costing the solution only, but the shifts generated by the move will still be present in the accepted solution, as the repair is not enforced on the solution. Hence, the view of moves as limited to shift-selecting mechanisms is not correct in our repair-costing setting – the specific ways in which work is re-distributed in our moves *do* have an impact on the search process.

Dynamics of infeasibility during the *exploration* **phase** Experiments in Section 6.6 show that the performance of the heuristics proposed degrades as the number of infeasible shifts in the solution increases. Therefore, leaving the number of infeasible shifts to grow too much may result in the *exploration* phase to stall, simply because of the limitations in the repair heuristics in use. Although this degradation in performance may act as a self-constraining mechanism for the *exploration* phase not to create schedules with too many infeasible shifts, it may be sometimes preferable to repair the solution during the *exploration* phase; in particular, there are cases where it may be useful to use the repaired version of a solution as the next point in the search. In this proposal, we implemented this in a way similar to tabu-search's aspiration criterion: during the search, we keep track of the best-cost solution $s^\star$ (including both *standard* and *exploration* phases); if at any point during an *exploration* phase the solution s' under consideration has a cost lower than that of $s^\star$, then s' is repaired, and the repaired solution is accepted as the next solution in the search.

Generating a feasible solution for the *standard* **phase** At the end of an *exploration* phase, and before starting the next *standard* phase, it is necessary to build a feasible solution to feed to the *standard* phase. As discussed earlier, although every

solution in the *exploration* phase is guaranteed to cover all pieces of work, WROs may be partially uncovered. At the same time, the *standard* phase relies on a specific model of WROs to simplify the phase where relief relocation within WROs is tried, where all relieving within the same WRO in a given schedule must happen at the same time (Section 4.2.1 and Appendix A). In conjunction, these issues mean that generating a feasible solution out of the current solution at the end of an *exploration* phase may not be trivial. We tackle this problem together with the covering of WROs in Section 8.2.1.

8.2 Setup and Experiments

In this section we describe the setup and implementation of the proposal in more detail. We then run a set of experiments using the new framework on the same set of instances as those used in the experiments in Section 5.3. Finally, we discuss a number of issues arising from an analysis of the results obtained on those tests. The configuration used to run the main experiment in Section 8.2.4 is summarised in Appendix B, including the reasoning behind some of the choices behind that configuration, where that reasoning is not clear from the current chapter.

8.2.1 The Inefficiency-Correcting Move

In Chapter 7 we discussed a number of possible types of inefficiency to tackle when trying to improve a solution. In the current proposal, we investigate the possibility of removing occurrences of high joinup time, a shift-level measure that aggregates travel time and idle time between spells. Given a schedule $\mathcal{S}$, the move developed removes from $\mathcal{S}$ the shift s with the highest total joinup time, adding in its place a shift s' selected from the pool of shifts $\mathfrak{C}$ obtained during the GaS phase. The shift s' is selected by first finding the longest gap g between spells in s, and then finding a shift that covers all the work in s before gap g, and either ends at that point or has the smallest gap right after the spells shared with s. After that, any shift in $\mathcal{S}$ that shares a piece of work with s' is also removed from $\mathcal{S}$. The pseudocode is presented in Algorithm 6. Notice that by feeding this solution to the *exploration* phase, we are placing the repair heuristic in a slightly different situation than originally designed

for, since the schedule resulting from this inefficiency-correcting move will not meet the constraint that all vehicle work must be covered, which we would usually consider a structural constraint.

Algorithm 6 Correcting inefficient use of non-driving time

1: given a schedule $\mathcal{S}$
2: find the shift $s \in \mathcal{S}$ such that the total joinup time is the biggest in $\mathcal{S}$; call the spells in s $sp_1, \ldots, sp_n$
3: find the longest gap g between spells sp_k and sp_{k+1} in s
4: find the shift $s' \in \mathfrak{C}$, with spells $sp'_1, \ldots, sp'_{n'}$ such that

   1. $\forall i, 1 \leq i \leq k : sp_i = sp'_i$ (disregarding covering of WROs)
   2. $n' = k$, or the gap between sp'_k and sp'_{k+1} is the smallest from all shifts in $\mathfrak{C}$ that satisfy condition 1

5: remove s from $\mathcal{S}$
6: add s' to $\mathcal{S}$
7: remove all shifts in $\mathcal{S}$ that cover a piece of work that is covered by s'

The aim of this move is to force the gap g out of the schedule. By having a shift s' covering all the work in s before g, no move or repair mechanism after this move will find it useful to select shift s (or any other shift covering sp_k before gap g) as part of the new solution, therefore making it impossible for the search to reintroduce gap g into the solution. It must be noted however that is at least one instance where shift s will have to be reintroduced anyway, namely when s is the only valid shift that covers some of the work in $sp_{k+1}, \ldots sp_n$. In this case, and while we disallow shift s' to be removed, any solution produced by the repair heuristics would include both s and s'. As soon as s' ceases to be fixed in the schedules, it is likely that it will be replaced back by s. It can also happen that there is no shift that satisfies the requirements in step 4. In this case, the correcting move cannot be executed.

We want to enforce that the gap is not immediately re-introduced by the search, including discouraging the repair costing from considering re-introducing the gap as part of the repair. To do so, we resort to a tabu-like mechanism, where we 'fix' the new shift s' in the schedule, such that no move can modify it, and the repair

mechanism cannot remove it from the schedule (step 2b in the framework described in Figure 8.1). This guarantees that no shift s'' in a repaired schedule will contain this 100-minute gap, unless having that shift s'' in the schedule is so important that it has to be added even if it overcovers all work in spells $sp_1, \ldots sp_i$. The shift stays fixed in the schedule for the duration of the current *exploration* phase, after which the restriction is removed (step 4 in Figure 8.1).

Recurring local inefficiencies Even with a criterion to explicitly forbid attributes from being re-introduced to the current solution in the search, early experiments show that in many cases, once that restriction on an 'undesirable feature' is lifted, this feature makes it back to the active solution in the search. In extreme cases, every run of a pair of (*exploration*, *standard*) phases results in a solution that contains the same inefficient attribute in the end solution. In these cases, it may seem more useful to try and find a different inefficiency to work on. This can be tackled in different ways, and the choice depends mostly on the intuition of what may be happening:

- it is possible to argue that the attribute is recurring because the search is unable to escape from the basin of attraction of the local optimum that contains this feature;
- it is also possible to argue that this attribute is likely to be part of any (or most) local optima, including global optima.

In the first case, a possible action to take is to keep the attribute for longer in the tabu list, hence giving the search more time to escape from that optimum. In the second case, a possible conclusion is that it may be better to accept that attribute as part of the solution, for example by not attempting to remove this attribute in the remainder of the search. It is also possible that both explanations are true. We subscribe to this latter view, and in consequence modify the proposal by randomizing the selection of the inefficiency to correct. During the execution of the inefficiency-correcting move, we rank the shifts according to the measure of inefficiency, filter those above a certain measure, and select a shift randomly from this list, with increasing probability for more inefficient shifts – a strategy that is close to the greedy-randomized selection

approach in GRASP [26]. In our opinion, this approach is a satisfactory answer to both explanations for recurring inefficiencies: on one hand, it should allow the search frequent chances to work on removing this inefficiency from the solution, which is useful if indeed the inefficiency can and should be removed; on the other, it allows the search to work on other inefficiencies as well, which is useful when the feature is required to be present on every efficient solution.

An optimization metaheuristic called *Squeaky Wheel Optimization* (SWO) [48] uses the history of the search to prioritize certain solution attributes dynamically. Aickelin et al. [4] present a variant of SWO and apply it to the driver scheduling problem; their experiments include comparisons with results obtained with an early version of the `TrainTRACS` system. It is likely that the ideas on dynamic re-prioritization of solution attributes can be incorporated into our proposal, to obtain better ways of selecting the inefficiencies to correct/remove.

8.2.2 Repair Costing: Covering of WROs and Generation of Feasible Solutions

The repair-cost function used in these experiments is based on the SR-rLCP-QW repair heuristic described in Section 6.6. As discussed in Section 8.1.3, all repair heuristics based on the set $\mathfrak{C}$ generated on a RoA model have the problem that covering WROs may result in very inefficient repairs (in terms of cost). To tackle this problem, we considered the following alternatives:

- Add some intelligence to the repair mechanism. For example, in the case of a partially-covered WRO, the repair could try to modify the shifts at the sides of the WRO to find a combination in which all the WRO is covered, and the shifts are feasible. This would substantially increase the complexity and running time of the repair heuristic, and may still not be enough to solve the problem, since it is possible that no relocation of the relief times that covers all the WRO is feasible. More sophisticated methods could be envisaged, but in general we would prefer to keep the repair heuristic fast (even greedy) if possible.

- Disregard the covering of WROs during the repair, effectively allowing the

repair to leave WROs partly uncovered. A first drawback of this approach is that the repair costing ceases to provide a 'real' cost of an equivalent feasible solution, and instead becomes an estimate of the cost that might be achievable by some feasible solution. Worse, this still leaves a problem unsolved, since there are points where a feasible solution is needed, in particular at the end of the *exploration* phase, and also as a result of the search.

- Switching to a RoA model during the *exploration* phase. This has the advantage that the repair functions described in Chapter 6 will result in all WROs fully covered without any modification, hence making the repair cost reflect the cost of an actual feasible solution, which in turn means that feasible solutions are readily available at any point during the *exploration* phase.

In these experiments, we opt for the third option, i.e. switching to a RoA model during the *exploration* phase. We do so at the start of each *exploration* phase, after the inefficiency-correcting move has been executed. To achieve this, we move every relief time happening inside a WRO in the current solution to the arrival time of that WRO. It is likely that some shifts become infeasible as a result of this procedure; however, this is also expected to be the case with the solution obtained after correcting an inefficiency, and should be handled without problems by the repair cost function.

8.2.3 Other Settings

Choice of cost function All optimizations using the new framework are run using a cost function based on total cost only (Equation 2.2). This is mainly because the repair heuristics designed in Chapter 6 work on minimizing cost – which in itself is related to the discussion in Section 2.1.2 in relation to the difficulties in comparing solutions on a schedule-size-based cost function.

Design of L_0 and L_1 moves As in the proposal in Chapter 4 (Section 4.2.1), both *standard* and *exploration* phases are structured as a fixed sequence of L_1 move trials, each of them running hundreds or thousands of low-level, L_0 moves. For the experiments presented in this section, each *standard* phase iterates three times over

a fixed set of L_1 moves, while *exploration* phases iterate twice over a different set of L_1 moves; the active solution in an *exploration* phase is repaired after each of these two iterations.

Additional complexity in the problem instances All three test instances in the experiments presented in this section are more sophisticated than the ones used in Chapter 4, in that certain vehicle work happening very early in the morning is allowed to be covered by drivers scheduled on the previous day (as late-night work), and similarly late-night vehicle work can be carried out by drivers in scheduled on the following day (as early-day work). This is not trivial to model, and in our case it means that *standard* phases are unable to generate shifts that use early vehicle work as the end of late shifts, or late vehicle work as the start of early shifts. However, *exploration* phases are still able to exploit this, since the pool $\mathfrak{C}$ is generated using `TrainTRACS` and therefore contains shifts exploiting this possibility. It is then difficult and possibly unfair to compare results between `TrainTRACS` or `PowerSolver` and the new proposal, in particular to those results obtained in Chapter 5. It is worth mentioning however that although schedules obtained in our framework may not be the most efficient for these test cases, they are still fully operational as they satisfy all labour agreements and constraints.

8.2.4 Results

We run four different experiments over three real-life instances of the problem (*Wales*, *Wessex*, *InterCity*), which were previously used in Chapter 5:

1. Run the new proposal on a RoA formulation, starting from a solution obtained using `TrainTRACS` (but not `PowerSolver`) on a RoA formulation; notice running the proposal on a RoA formulation is unnatural as there are no WROs to exploit; we use this run only as a benchmark to compare other runs to.

2. Run the new proposal on a WRO formulation, starting from a solution obtained using `TrainTRACS` (but not `PowerSolver`) over a RoA formulation; for each instance, this is the experiment we are more interested in.

3. Run the new proposal on a WRO formulation, starting from an empty solution that is repaired into a feasible solution by a repair heuristic; although we are more interested in investigating how WROs can be exploited from a good initial solution for the RoA formulation, this run may show whether the new tool has potential to be used as a standalone solver for the TDS(W) problem[1].

4. Run the new proposal on a WRO formulation, starting from a solution obtained using `PowerSolver` over an extended formulation obtained as in Section 5.3 by looking at WROs in relation to the feasible-travel-links constraint; with the caveats in Section 8.2.3, this run is used to assess the potential of the new tool to optimize on the results of `PowerSolver`, which is run here over arguably the only kind of model where GaS could exploit WROs. Throughout this chapter, we will refer to this extended formulation as a *URO* ('Useful ROs') formulation.

Because one of the motivations for these experiments is to evaluate the role of inefficiency-correcting moves and *exploration* phases as diversification phases, we do not impose a general stopping criterion for the overall search in these experiments; instead, we let the search run for a fixed amount of time (in general, 2h, or 7,200 seconds, per instance); common stopping criteria such as limits on time without generating new best-cost solutions would be easy to integrate if desired. Experiments were run on a computer with a single Intel 1.5GHz Pentium M processor and 512MB or RAM. Results of the four tests on individual datasets are shown on Table 8.1 (page 146) and Figures 8.3, 8.4 and 8.5 (pages 147, 148 and 149). As part of the analysis on the role of *exploration* phases in Section 8.2.6, we also graph the evolution of the number of infeasible shifts during *exploration* phases for the *Wessex* and *InterCity* datasets (Figures 8.7 and 8.8 respectively, pages 151 and 152).

As a starting point to our analysis, it is worth noting that as a result of the use of our repair-costing scheme and heuristics, the cost displayed on the y-axis of all graphs is *always* the cost of an existing feasible solution, irrespective of whether the search is traversing the space of feasible or infeasible solutions. This is, in our view, an extremely important feature of our proposals. In effect, the only unusual aspects

[1] In effect we are replacing the selection phase in GaS with a different selection tool, as our heuristics still depend on a pool $\mathfrak{C}$ of feasible shifts being available and so the generation phase is still needed

in the shape of the curves are not due to the switch to the infeasible region, but rather to the execution of the inefficiency-correcting move.

Wales dataset

Although the *Wales* dataset is the set with the most relief opportunities of the three, it is the one with the smallest schedule sizes according to both `TrainTRACS` and `PowerSolver`. Similarly, this is the instance where our local search proposal works faster, which results in smaller CPU times per phase, and hence more *standard* and/or *exploration* phases per unit of time. Results for the *Wales* dataset are shown in Figure 8.3 on page 147.

Although not our main objective in the analysis, it is clear that the search mechanism is not powerful enough to beat `TrainTRACS` or `PowerSolver` in itself – the three searches on the WRO formulation organize around three clearly different areas in the range of possible costs. However, it is interesting to see that the search is able to improve on the starting solution in all cases, and particularly when starting from a solution from a RoA formulation, where it achieves an improvement of 2.7% over the initial solution.

More importantly, a comparison between the two searches starting from the `TrainTRACS` solution on the RoA formulation, but running over different formulations (RoA and WRO) suggests that WROs are instrumental in getting a more efficient schedule. The plot on the search using the RoA formulation also shows a peculiarity (not present in experiments in other datasets) that at some point the search is unable to find a replacement shift for the chosen inefficient shift; from that point onwards, the search cannot progress any further.

Results on the WRO formulation, both when starting from an empty solution and from the `TrainTRACS` solution over a RoA formulation, show how the *exploration* phases are able to recover from the increase in cost due to inefficiency-correcting moves. It is also interesting to see how the search traverses different regions of cost, and doesn't necessarily reach the best-cost solutions until late into the search. It is difficult to say whether the history of the search is instrumental in leading to those late good-cost solutions, or the improvements are due to quasi-random perturbations (bearing in mind that even the inefficiency-correcting move is partially randomized).

The questions of whether and how local search (meta)heuristics effectively exploit search history, and how they compare to random-restart methods (e.g. GRASP) in the way they explore the search space, are extremely interesting subjects, but also very hard and difficult to define, and are outside of the scope of this thesis.

Finally, it is interesting to see that the search on the WRO formulation that starts from an empty solution is within 3% of the `TrainTRACS`/RoA solution in less than 2 minutes, and eventually manages to find a solution that is better than the `TrainTRACS` RoA solution, even with the limitations in covering early/late vehicle work.

Wessex dataset

Results of the four tests are shown in Figure 8.4. A first observation is that the distribution of the costs of active solutions for different methods is not as clearly spread into distinct regions as it was on the *Wales* dataset; in fact, all three searches starting from a non-empty solution overlap with each other at some point during the search. As with the *Wales* dataset, the search starting from the `TrainTRACS`/RoA solution is able to make more improvements when working on the WRO formulation instead of the RoA formulation, which further supports the idea that WROs are instrumental in achieving better solutions.

It is interesting to give a more detailed look at the overlaps in cost between the three searches, and in particular the two searches that operate on the WRO formulation. It seems striking at first that although both searches reach stages where the relationship between costs is inverted, this is quickly undone. On further inspection, all such occurrences happen when the search that started from the `PowerSolver`/URO solution is executing an inefficiency-correcting move. Our intuition is that the `PowerSolver`/URO solution contains certain structural properties that are neither removed by the inefficiency-correcting moves, or capable of being introduced by the search mechanism; the temporary increase in cost by the inefficiency-correcting move is then either repaired or undone by a later *standard* phase. A possible way to exploit this observation is by adding inefficiency-correcting moves that tackle different inefficiencies.

InterCity **dataset**

Results on the WRO formulation are shown in Figure 8.5. In this case, all searches seem to converge to the same range of costs. As in the *Wales* dataset, the search starting from an empty-repaired solution is able to match the cost obtained using `TrainTRACS` on a RoA formulation. An interesting result arising from the graph is that the runs on the WRO and RoA formulation using the new framework are extremely similar in the range of costs spanned, and there is even a difference of 215 minutes (0.36% of the cost) in favour of using the RoA formulation when starting from the solution obtained with `TrainTRACS`. This is consistent with the perceived difficulties in `PowerSolver` to exploit UROs in the experiments in Section 5.3, and it suggests that the impact of considering WROs in the scheduling model is not equal among all instances.

Finally, results on the search using a starting solution from `PowerSolver`/URO show, similarly to those on the *Wessex* dataset, how the search reaches points where the cost is worse than that of solutions obtained on other searches, and seem to confirm the intuition expressed when analyzing the *Wessex* dataset, in that the solution generated with `PowerSolver`/URO contains certain structural attributes that our framework is both unable to generate and remove from a solution.

8.2.5 Comparison with Earlier Proposals

The proposals in this chapter can be seen as extensions to the initial proposal in Chapter 4. Therefore it makes sense to evaluate the new framework on the same data than that used for testing the initial proposal. We run two experiments on the *ScotRail* dataset, namely the first and second experiments run in Section 8.2.4, running the local search on ROA and WRO formulations respectively, starting from the solution obtained with `TrainTRACS` on a ROA formulation. Our aim is to compare the results obtained with the new framework with those obtained using the first proposal, specifically with the experiments conducted in Section 4.2.2. In both experiments we do not relax artificial constraints in the description of the problem instance, so results should be compared with the first row of Table 4.1 in Section 4.2.2.

Results for the two experiments are graphed in Figure 8.6. The best-cost solution obtained on the WRO formulation with the new proposal has a cost of 457h32m, a 43% increase in the gain in cost over the results in Section 4.2.2 (458h13m) when compared to the original solution obtained with `TrainTRACS` (459h48m). It is difficult to ascertain the relevance of this extra reduction in cost in terms of getting closer to the global optimum for this instance, as we do not know the value of the optimal cost under the WRO formulation. The experiments also show that, for this instance, the new proposal performs better when acting on the ROA formulation than when conducted on the WRO formulation, achieving an extra 112% decrease in cost (456h27m) over the best-cost solution in experiments in Section 4.2.2.

8.2.6 Discussion

It should be clear that the proposal discussed and tested in the previous sections is only one of many possible ways of addressing the limitations discussed in Chapter 6; in this sense, this proposal can be seen a proof of concept on how these limitations can be tackled. The new framework addresses the limitation on search space connectivity by lifting the restriction on the solutions being feasible during the *exploration* phase; the problems associated with comparing infeasible solutions are tackled through a repair-costing mechanism. Domain knowledge is further exploited by executing specific inefficiency-correcting moves after *standard* phases. In the following paragraphs, we briefly look at some issues arising from the experiments on this proposal.

Use of WROs As with previous proposals in this thesis to solve the TDSW problem, we are interested in determining whether the solutions found by the algorithms make use of the WROs available in the problem instance, either in the best-cost solutions or in intermediate solutions in the search. As in Section 5.3, we consider WROs to be exploited fully in a shift s if replacing active reliefs occurring in s at times other than arrival with relieving at arrival time results in s becoming infeasible.

To verify the use of WROs, we took the best-cost schedule from each of the three experiments in Section 8.2.4 and performed the substitution described in the previous paragraph. The number of infeasible shifts resulting from this procedure

was two for the *Wales* dataset, two for the *Wessex* dataset, and six for the *InterCity* dataset, out of 72, 117 and 125 shifts respectively. The same behaviour was also verified in intermediate solutions; for example, a (randomly selected) intermediate best-cost schedule for the *Wessex* dataset contained four infeasible shifts when replacing all active reliefs inside WROs with relieving on arrival. Overall, results suggest that the algorithm does make use of WROs during the search; moreover, best-cost solutions found make use of WROs in all three cases.

A closer look at *exploration* phases As a separate analysis, we plot the evolution on the number of infeasible shifts during the search, together with the changes in cost in the active solution. Our aims are, first, to look at the way in which the *exploration* phase makes use of the restriction on feasibility being relaxed and, second, to understand the evolution of cost during the search in relation to *standard* and *exploration* phases.

Results on the *Wessex* dataset for a search using the WRO formulation, starting from a schedule obtained running GaS on a RoA formulation, are shown in Figure 8.7. A first observation is that the *exploration* phases are making use of the possibility of keeping infeasible shifts on the active solution. In principle, results would show that *exploration* phases are able to find cost-equalling or cost-improving solutions even when the active solution has a sizable number of infeasible shifts; this is evidenced by the fact that the number of invalid shifts changes inside *exploration* phases (a change in the number of infeasible shifts implies that the active solution has been replaced; the reverse is not true, so the count of changes in the number of infeasible shifts is a lower bound on the number of changes in active solution). However, we must be more precise in the analysis, since the active solution is forcibly repaired after each iteration of the *exploration* phase, and at the end of the *exploration* phase. This means that on each *exploration* phase, two of the intermediate changes to zero infeasible shifts, as well as the final change in the cycle, are due to forced repairs. Hence for example the fourth, fifth, seventh and eight *exploration* phases in Figure 8.7 are such that almost all changes in the number of infeasible shifts result in more infeasible shifts in the schedule, and the *exploration* phase is not able to recover to a lower level of infeasible shifts until it is forced externally. Overall, this graph suggests

that most of the cost improvements in the search are due to changes in the *standard* phase; in particular, sharp decreases at the start of each *standard* phase are due to the first L_1 move in a *standard* iteration being devoted to analyze relocations of active reliefs within WROs – in effect 'reinstating' the use of WROs that is removed at the start of an *exploration* phase, as described in Section 8.2.2. It is difficult to ascertain how much these improvements are related to changes occurring during *exploration* phases, or the inefficiency-correcting move.

We plot the same graph for a test on the *InterCity* dataset (Figure 8.8). However, in this case we plot results from the search that starts from an empty-repaired solution. Although the evolution of the number of infeasible shifts over time seems to broadly follow the same pattern, in this case most of the cost improvement seems to happen *within* the *exploration* phases. Our intuition is that because in this case the search starts from a very high-cost solution, the problem resembles more the classical TDS problem than the TDSW problem, i.e. most of the improvements should come from being able to solve the TDS problem efficiently, rather than from exploiting WROs, and the repair heuristics would seem to be better at solving the TDS problem than our *standard* phases with the current configuration. Together with the observation in Section 8.1.3 that moves during *exploration* phases may usually simply act as a shift-selecting mechanism, this suggests a promising general-purpose approach for the design of moves in local search frameworks, where an efficient repair heuristic is combined with a suitable selection of elements to remove from the current solution.

On the need for *standard* phases With respect to the exploration of the infeasible region, this proposal differs from the original motivation in Chapter 6 in that infeasible solutions are not allowed at every point during the search, but only on specific (*exploration*) phases. The search could instead be structured into a single '*exploration*' phase, with inefficiency-correcting moves triggered by the search getting stuck in local optima, effectively removing the *standard* phases. This alternative proposal may look simpler and cleaner; however, two issues make this not viable with the current setup:

- *Exploration* phases work over a RoA model; hence, if the *standard* phases are removed, the search is effectively unable to exploit WROs;
- even without the problem above, *exploration* phases are inefficient when repairing schedules with a sizable number of invalid shifts, a problem that the *standard* phases are not subject to.

With regards to the second observation, an alternative way of interpreting the *standard + exploration* proposal in this chapter is to consider the *exploration* phases as the driving force of the search, while the *standard* phases are used to 'refine' the solutions obtained during the *exploration* phases – a local search within a local search. In Section 8.3 we propose a search mechanism where *standard* and *exploration* phases work on the same formulation; even then, it is still desirable to have *standard* phases during the search.

Exploration vs. exploitation Any search algorithm for a combinatorial optimization problem is subject to the exploration–exploitation disjunctive. In essence this refers to the problem of balancing the time/resources spent on exploring new areas of the search space, with that of exploiting the knowledge gathered in the search, e.g. by looking for the local optimum in a basin of attraction. This a central issue in all optimization algorithms based on iterative improvement, and also in all areas of research related to learning, e.g. reinforcement learning [75, 49]. In the proposals in this chapter, the *exploration* phase can be seen as doing most of the exploration, while the *standard* phase does most of the exploitation.

In combinatorial optimization, exploration is associated with or referred to as *diversification*; similarly, exploitation is associated with *intensification*. Some metaheuristics consider the balance of these two aspects explicitly (or at least expose the issue more prominently); for example, the cooling schedule in simulated annealing is a way of controlling the frequency of 'exploratory' moves, i.e. those that are taken even if the cost of the new solution is worse than that of the previous solution in the search. Unfortunately, there is no consensus in the literature over what constitutes a desirable balance between exploration and exploitation, and hence usually each implementation of a search procedure for an optimization problem will need tun-

ing of this balance (see e.g. [76, 65] for research on cooling strategies in simulated annealing).

The experiments in this chapter expose many areas where the algorithms can be altered to favour exploration over exploitation, or viceversa. For example, the inefficiency-correcting move between the *standard* and *exploration* phases is configured to remove all shifts overlapping with the shift replacing the inefficient one. This seems to be useful in allowing the repair some freedom to reshape the solution around the inefficiency. However, as the search progresses, and the cost of the solution after a *standard* phase goes down, there is a stage at which removing all adjacent shifts sometimes leaves the *exploration* phase unable to recover to a solution of comparable cost to the previous *standard* phases. Further contradictions in the role of the inefficiency-removing move can be seen in Figure 8.4, *Wessex* on the WRO formulation, starting from an empty solution, where the inefficiency-correcting move seems to be triggering a descent at around 1,000 seconds (first spike), only to stop the search from getting closer to a local optimum 1,300 seconds later (third spike).

The idea of a cooling schedule in simulated annealing could be adapted to this scheme; for example, the first issue described above could be tackled by only removing the overlapping shifts in the early stages of the search (or reducing the number of shifts removed, or the probability of removing them, over time). However, as in simulated annealing, the difficulty lies in determining what constitutes the best way of (gradually) switching from exploration to exploitation. We do not explore the impact of reducing exploration over time in this thesis.

Repairing at the end of an *exploration* phase In the current proposal, the active solution at the end of an *exploration* phase is repaired before the execution of the next phase; if two or more iterations of the *exploration* phase are executed in sequence (e.g. as described in Section 8.2.3), we execute the repair after each iteration of the *exploration* phase. This introduces a subtlety in that, because the repair heuristic is randomized, the repaired solution may not have the same cost as it originally had when first evaluated. Although this cost may be lower, in general it will most likely be the opposite, since the search has a hard bias to accept good-cost solutions. The effect is noticeable in the experiments conducted in this section. The

graph in Figure 8.8 shows an extreme occurrence of this: in the sixth *exploration* phase (at about 3,400 seconds into the local search), we can observe the cost of the current solution increase in value three times during the same phase. This can be explained by the cost going up after the inefficiency-correcting move before the start of the *exploration* phase, and then once per repair after each of the two iterations within the *exploration* phase.

This behaviour could be avoided by caching the original repaired version of each solution that is accepted during an *exploration* phase. We have however kept this unchanged because of a number of reasons: first, *exploration* phases are meant to be exploratory in nature; second, there are performance issues in keeping a repaired version of each accepted solution, particularly because we also accept solutions that keep the cost unchanged (admittedly, performance would not suffer excessively); thirdly, keeping a repaired version would imply that the repair is able to efficiently produce feasible solutions with the exact repair cost – although this is true in the current experiments, it is easily the case that a different repair heuristic would be unable to do so, and a different repair mechanism (slower but lower-cost-yielding) would be used at the end of an *exploration* phase instead. A setup with two distinct repair mechanisms can be reproduced in our proposal by using a higher number of trials when repairing at the end of an *exploration* phase than that used during repair costing.

8.3 Running the Search on a Reduced Set of Relief Opportunities

The work in Chapters 5 and 6 is in most part motivated by the observations that

a) most of the ROs within WROs in a 1-minute expansion may be redundant

b) most of the solutions generated by moves in the local search are infeasible (to `CHECKER`), hence unnecessarily generated

However, the proposals presented so far in this chapter do not address these two motivations fully. In particular, *standard* phases attempt to exploit WROs in the

same way and are subject to the same constraints as the original proposal in Chapter 4 when creating/evaluating new solutions, and therefore exhibit the two problems above.

In this section, we look at integrating the proposals in Chapter 5 into the local search framework. We start by observing that during the *standard* phases, when a move looks at exploiting WROs it considers all possible relocations of ROs within selected WROs (using the model in A.1.4). Revisiting the study of constraint boundaries in Section 5.2, it can be argued that each execution of the relief-relocation step of a move in an *standard* phase is, in terms of exploiting WROs, re-evaluating the use of ROs within WROs to generate structurally different shifts. From this point of view, it would make sense to replace these relief-relocation stages with a preprocessing of the set of ROs within WROs, as suggested in Sections 5.3 and 5.4.

Our modified proposal is then to run a preprocessing stage on the problem instance, generating a model M that contains all ROs in the RoA formulation, and those ROs within WROs that are instrumental in generating structurally different shifts from those available in the RoA formulation (a URO formulation), using one or more rules related to scheduling constraints, as discussed in Chapter 5. In this expanded RoA model, the interval of work between two consecutive ROs in a vehicle is considered as a separate piece of work, independently of whether that interval covers or has an overlap with an attended WRO. This means that any algorithm operating over the model M cannot distinguish WROs within the problem instance. The generation phase of GaS is still executed, to obtain a set $\mathfrak{C}$ of valid shifts for M. The search starts from an empty-repaired solution[2]. The new scheme is summarized in Figure 8.9.

An important aspect of this proposal is that both *standard* and *exploration* phases work on the same formulation of the problem. Hence the problems in covering WROs described in 8.2.2 effectively disappear. Explicit consideration of WROs is removed from the local search phase, so the WRO-relocation stage in moves is eliminated. Instead, most of the responsibility for exploiting WROs is shifted to both moves that re-distribute pieces of work (especially those targeting the start and end pieces of

[2]The selection phase of GaS can always be executed if one wants to generate an initial feasible solution for the local search phase.

spells), and the repair costing in *exploration* phases. We keep however the *standard* and *exploration* phases in the proposal, as they still serve the same purposes of exploitation and exploration.

8.3.1 Experiments and Results

We run the new proposal on the *Wales* dataset. We compare the results with those obtained from the search on the WRO formulation, starting from an empty-repaired solution as well. Both runs are graphed together in Figure 8.10. Results show that the search on the URO formulation is initially able to make progress more quickly; however, it soon reaches a point in the search where no cost improvements are possible; at the same time, the inefficiency-correcting is unable to find replacement shifts for the inefficient shifts it chooses to replace, which means that the search effectively gets trapped in a local optimum. On the other hand, the search on the WRO formulation is able to progress further, eventually reaching a better-cost solution. This result would suggest that in the case of local search approaches – where considering WROs in full does not imply an explosion in the problem size like it does for the generation phase of GaS– full consideration of WROs may allow for better solutions to be found.

8.4 Conclusions

This chapter presents a new hybridized, GaS and Local Search framework that extends the ideas presented in Chapter 4. Based on the limitations discussed at the start of Chapter 6, the work on repair heuristics on the same chapter, and the identification of local inefficiencies in Chapter 7, we develop a new local search framework where feasible and infeasible parts of the search space are iteratively explored in sequence, through *standard* and *exploration* phases. The start of an *exploration* phase is triggered by an inefficiency-correcting move (for which we design one specific move); this move is more sophisticated than regular moves in *standard* phases, but does not guarantee a feasible solution as an output. Solutions generated during *exploration* phases are compared through a repair-cost function, which for a given (possibly infeasible) solution s returns the cost of a feasible solution s' built

from s.

We evaluate the behaviour and performance of the new proposal over a set of three real-life instances of the TDSW problem, which were previously used in Chapter 5. Results suggest that the *exploration* phases are instrumental in taking the search from the infeasible solutions generated by the inefficiency-correcting back to cost-efficient feasible solutions. The new method is able to improve on all instances when starting from a solution generated by `TrainTRACS` over a RoA formulation, and two out of three instances (albeit only marginally) when starting from a solution generated with `PowerSolver` over a specific URO formulation. Both results add further support for the intuition that considering WROs in the scheduling model may allow for better-quality solutions to be found. An analysis of the role of *standard* and *exploration* phases in the evolution of the cost of the active solution suggests that responsibility for improving on cost varies according to the choice of starting solution: if the starting solution is a good solution for the TDS problem, then *standard* phases –which are designed to exploit WROs– are responsible for most of the improvement; if on the other hand the starting solution is of poor quality, most of the improvement would come from exploiting the structure of the TDS problem itself, rather than the use of WROs; in this case, *exploration* phases seem to generate most of the improvement.

We also evaluate the possibility of applying the new methodology over a URO formulation, where a preprocessing of the problem instance results in a reduced set of ROs being available to the solver (compared to the full 1-minute expansion of WROs). Results suggest that the new scheme as designed performs better when full consideration of WROs is allowed.

The proposed framework involves a considerable number of distinct components, and therefore there are many possible ways to try and improve on it. Better tuning of the balance between exploration and exploitation could be beneficial to the search; in particular, it would be desirable to make this a self-adaptive mechanism. Also, the use of a single type of inefficiency during inefficiency-correcting moves is clearly limited. We expect that considering more types of inefficiency would also result in more efficient solutions being found; however, selecting the best type of inefficiency

to tackle at any given point during the search is a complicated issue, since different types of inefficiency are not comparable in general, and especially since we want to avoid using the cost function as a decision tool when comparing inefficiencies. Finally, the repair heuristics in use force some restrictions on the framework that we would like to remove. In particular, the fact that the current repair heuristics work on the RoA formulation (and hence the active solution needs to be stripped of any reliefs occurring inside WROs) is undesirable; however, adapting any of the heuristics proposed to work on a WRO formulation seems very difficult, since all of them rely on the use of a pool of feasible shifts, and generating the full set of feasible shifts for a WRO formulation is usually impossible in practice.

method	T	LS	LS	LS	PS	LS	LS
formulation	RoA	RoA	WRO	WRO	URO	WRO	URO
starting solution	–	T/RoA	T/RoA	empty	–	PS/URO	empty
Wales	34,161	33,911	33,228	34,127	32,722	32,631	34,576
Wessex	55,871	55,575	55,331	56,548	54,658	54,646	
InterCity	60,630	59,959	60,174	60,341	59,668	59,584	

Table 8.1: Results for the experiments on the new framework. Cost is expressed in total paid minutes. Method is one of LS (new local search framework), T (`TrainTRACS`, without using `PowerSolver`), or PS (`PowerSolver`). Starting solutions for LS runs are labeled 'method/formulation'. Experiments on running the new framework on a RoA formulation (third column) are used as a control.

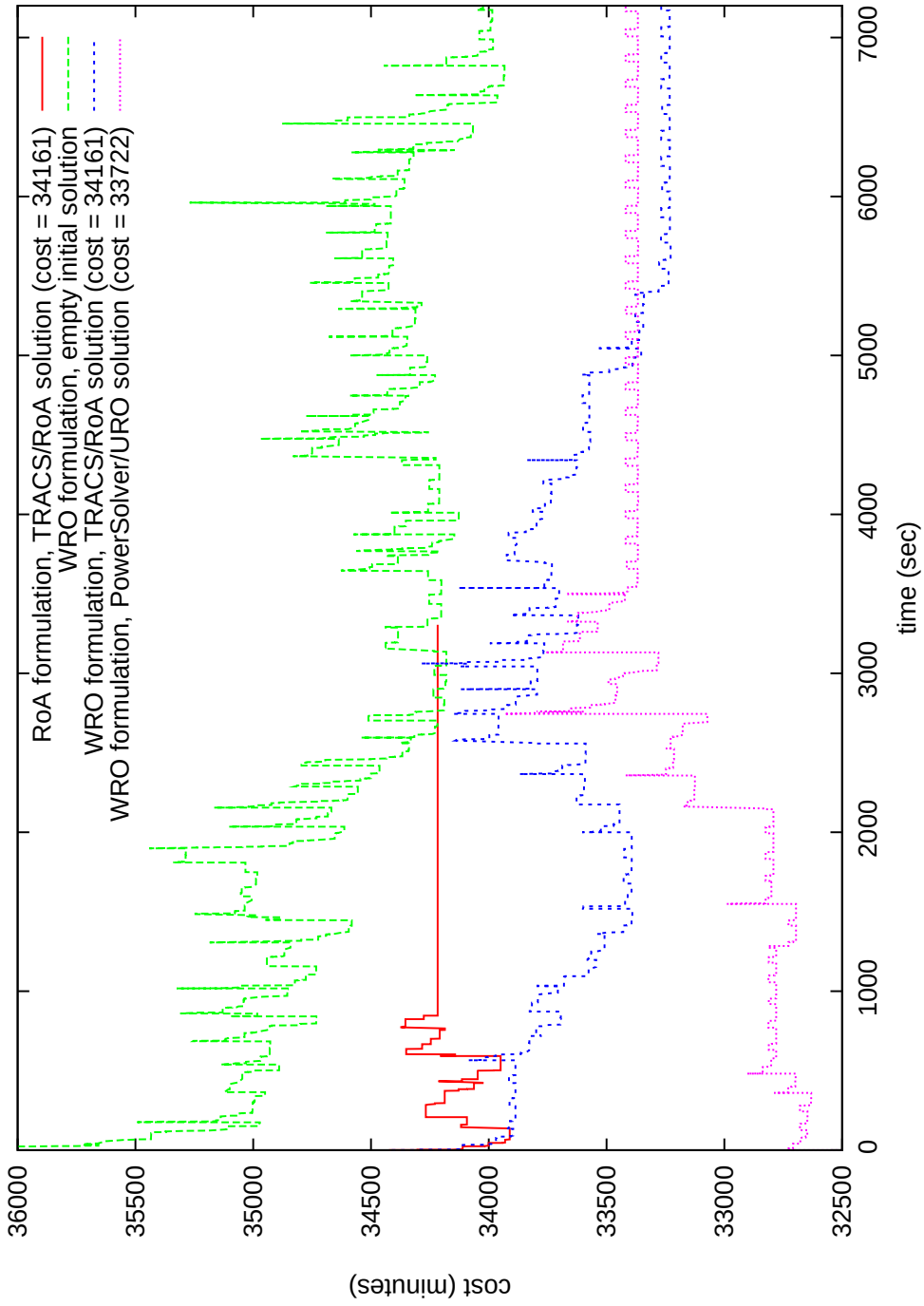

Figure 8.3: Execution of the new proposal on the WRO formulation, *Wales* dataset. Execution on a RoA formulation is included for comparison purposes.

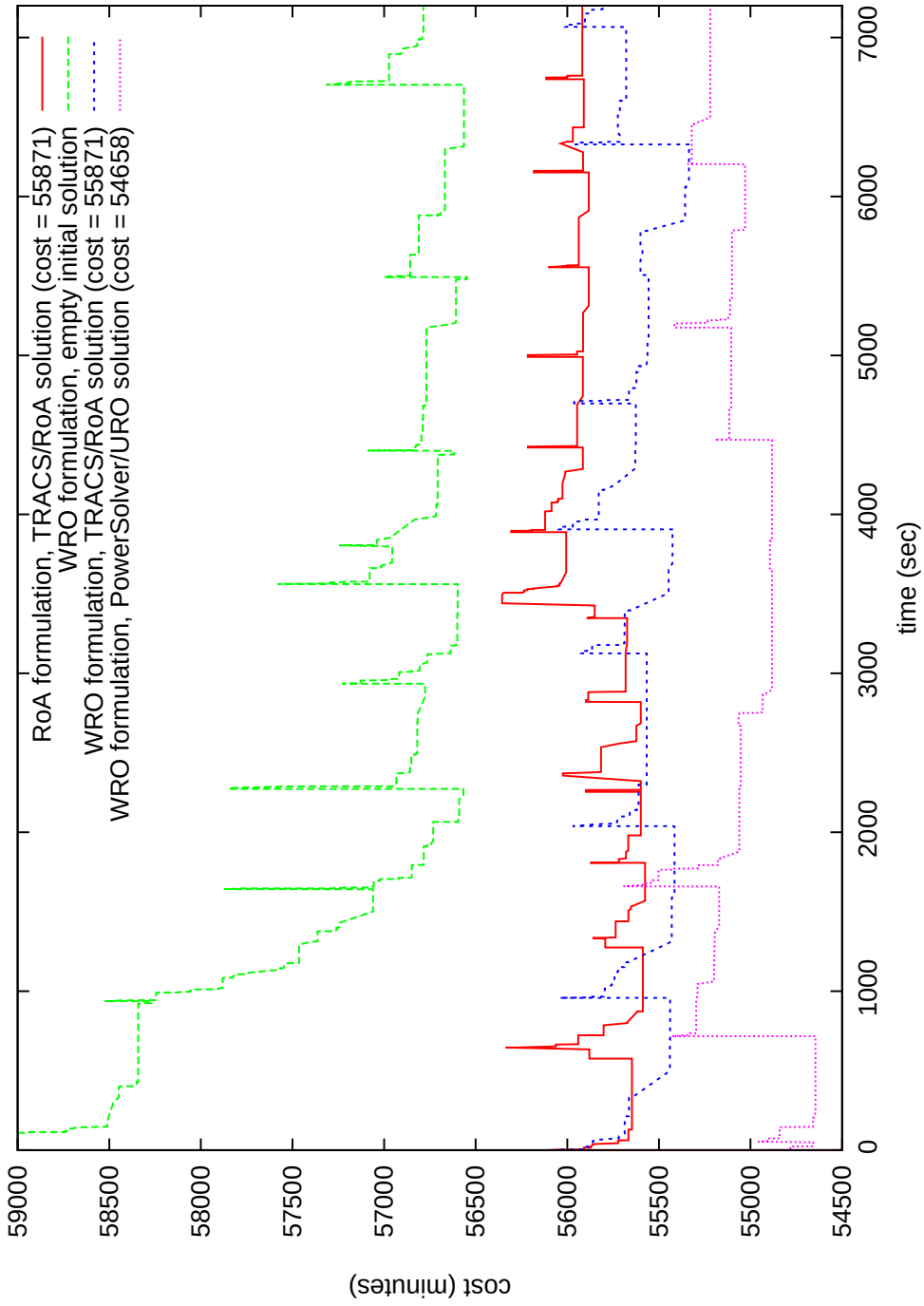

Figure 8.4: Execution of the new proposal on the WRO formulation, *Wessex* dataset. Execution on a RoA formulation is included for comparison purposes.

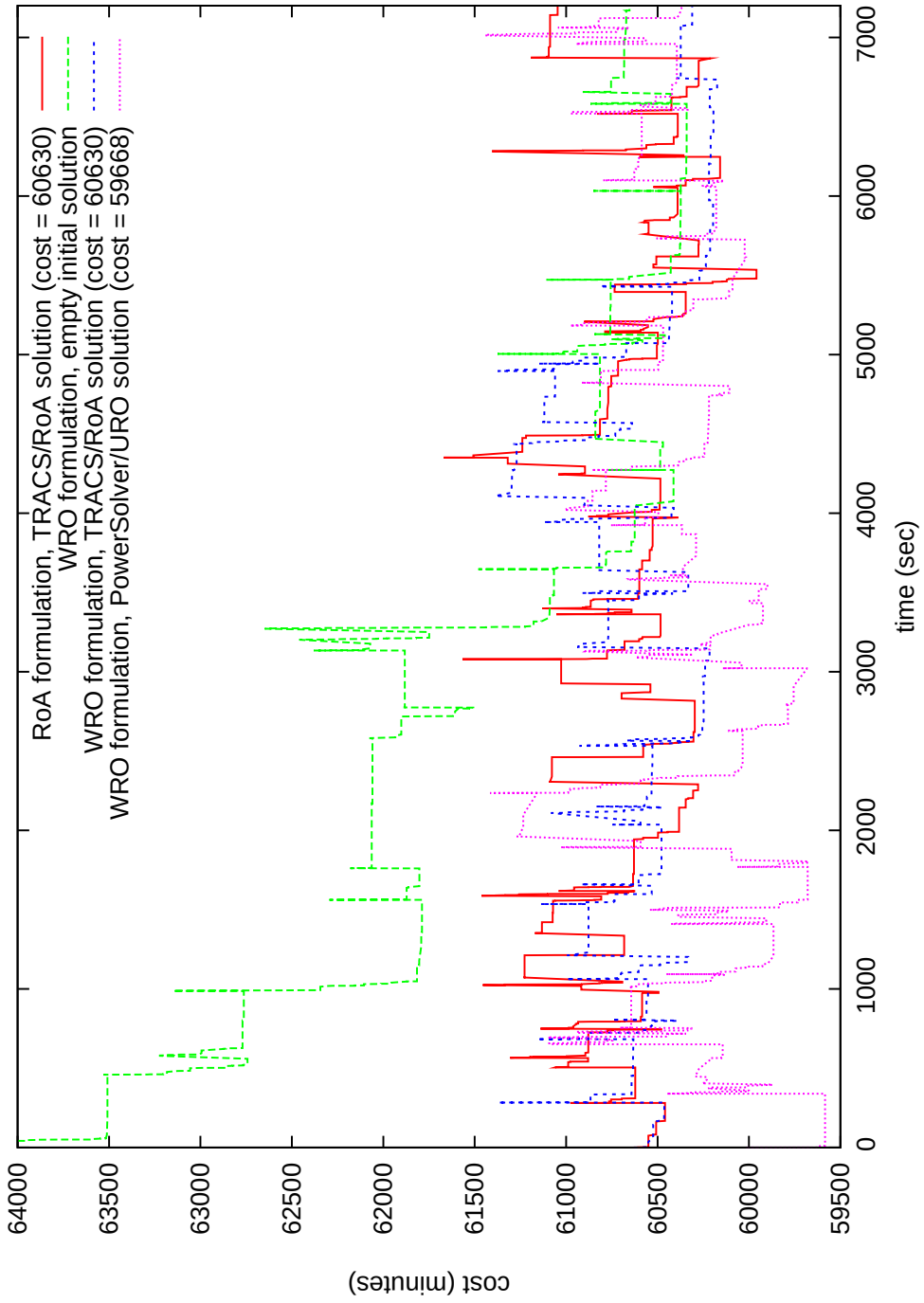

Figure 8.5: Execution of the new proposal on the WRO formulation, *InterCity* dataset. Execution on a RoA formulation is included for comparison purposes.

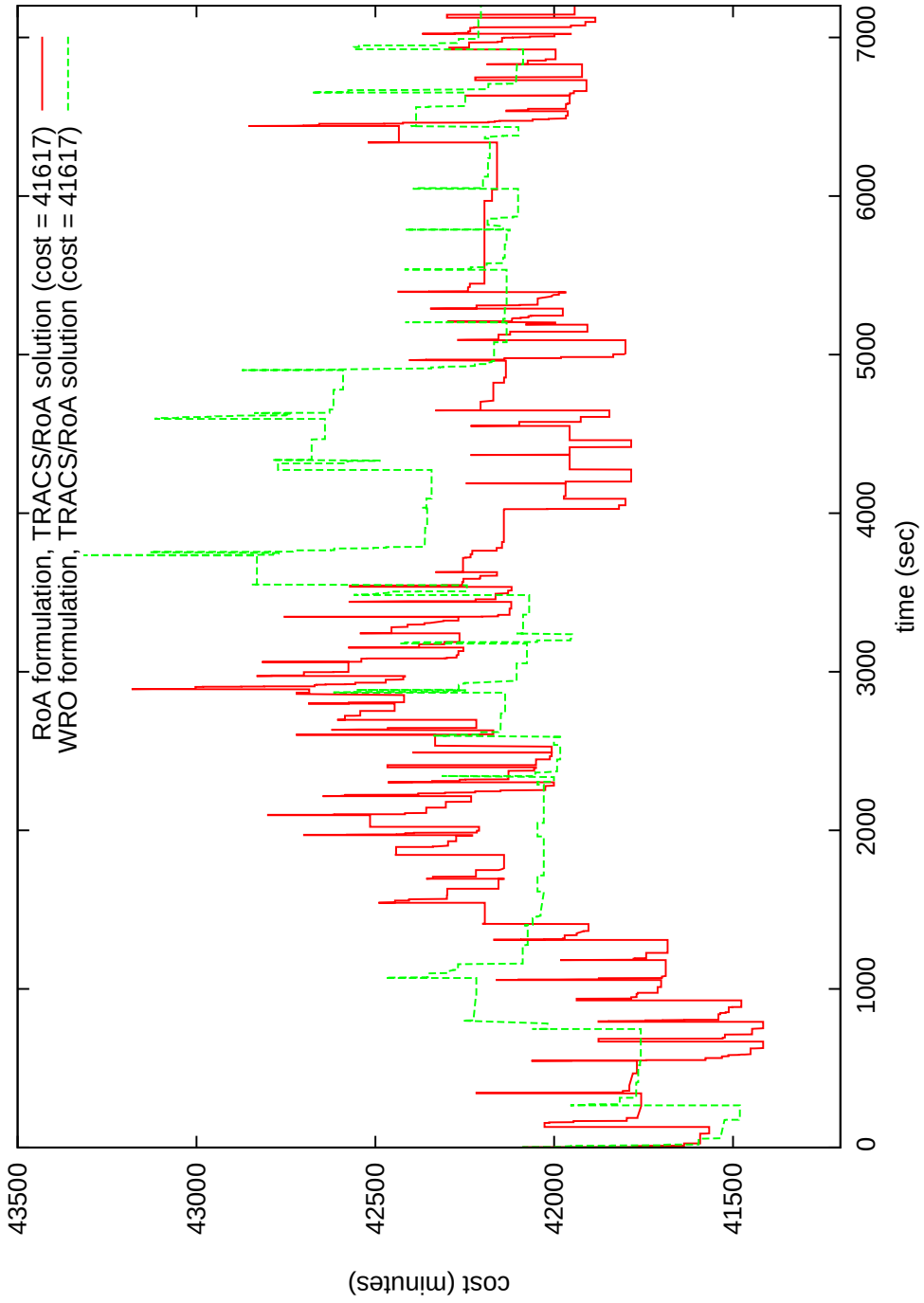

Figure 8.6: Execution of the new proposal on the ROA and WRO formulations, *ScotRail* dataset. Execution on a RoA formulation is included for comparison purposes.

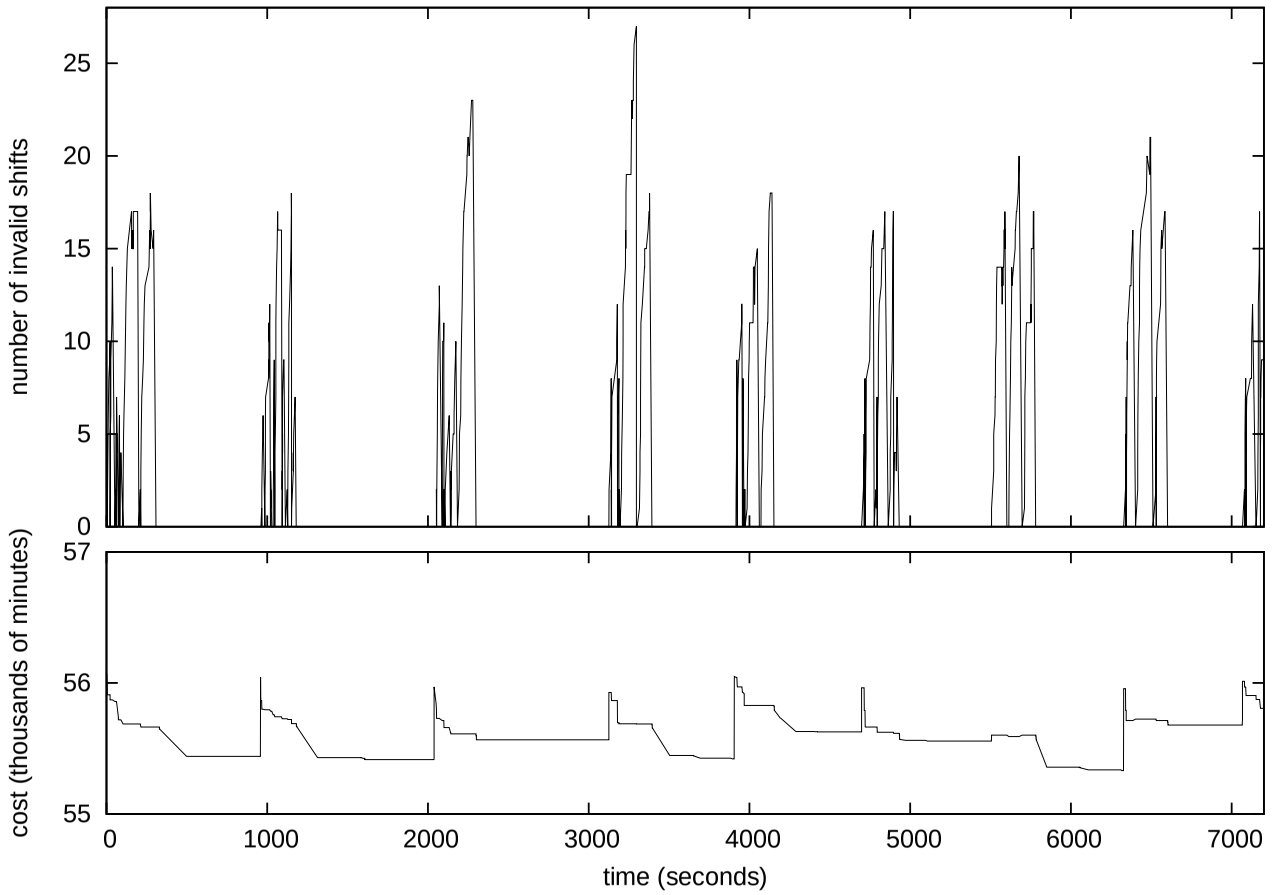

Figure 8.7: Number of invalid shifts in active solutions (*Wessex* dataset, WRO formulation, starting from a schedule obtained running GaS on a RoA formulation). Spikes in the number of infeasible shifts mark the start and end of *exploration* phases. A comparison with the cost of the active solution in the schedule (graphed below) suggests that most of the improvement in cost during the search happens right after *exploration* phases.

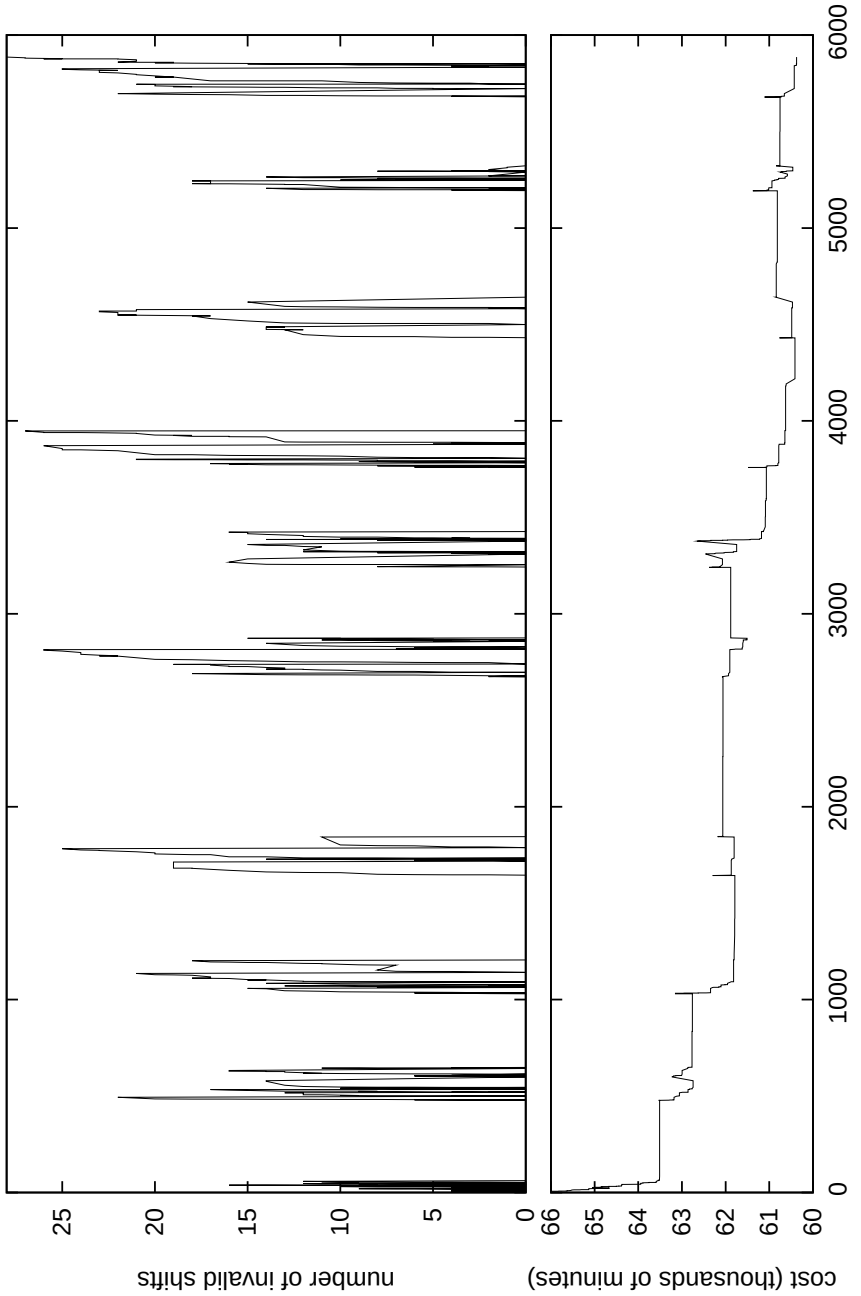

Figure 8.8: Number of invalid shifts in active solutions (*InterCity* dataset, WRO formulation, starting from an empty schedule). Spikes in the number of infeasible shifts mark the start and end of *exploration* phases. A comparison with the cost of the active solution in the schedule (graphed below) suggests that most of the improvement in cost during the search happens during *exploration* phases.

0.	generate a URO formulation U for the instance, by analyzing feasible travel links constraints
1.	set $i := 1$; run *Generate and Select* on formulation U, obtaining a feasible solution $\mathcal{S}_i^{GaS}$ and a set $\mathfrak{C}^U$ of feasible shifts
2.	run a *standard* phase on formulation U, starting with the current (feasible) solution, generating a new feasible solution $\mathcal{S}_i^{std}$
3a.	execute the inefficiency-correcting move on $\mathcal{S}_i^{std}$, leaving a possibly infeasible new solution $\mathcal{S}_i^{inf}$
3b.	add rules to prevent the next phase from undoing the inefficiency correction
4	run an *exploration* phase on formulation U, starting with the current (infeasible) solution $\mathcal{S}_i^{inf}$ and using the pool of shifts $\mathfrak{C}^U$, generating a new feasible solution $\mathcal{S}_i^{exp}$
5	remove rules to prevent undoing of the inefficiency correction; set $i := i + 1$; go to step 2

Figure 8.9: Scheme for the modified GaS and Local Search proposal, working on a reduced set of relief opportunities derived from looking at scheduling constraints. In this proposal, both *standard* and *exploration* phases work on the same formulation of the problem.

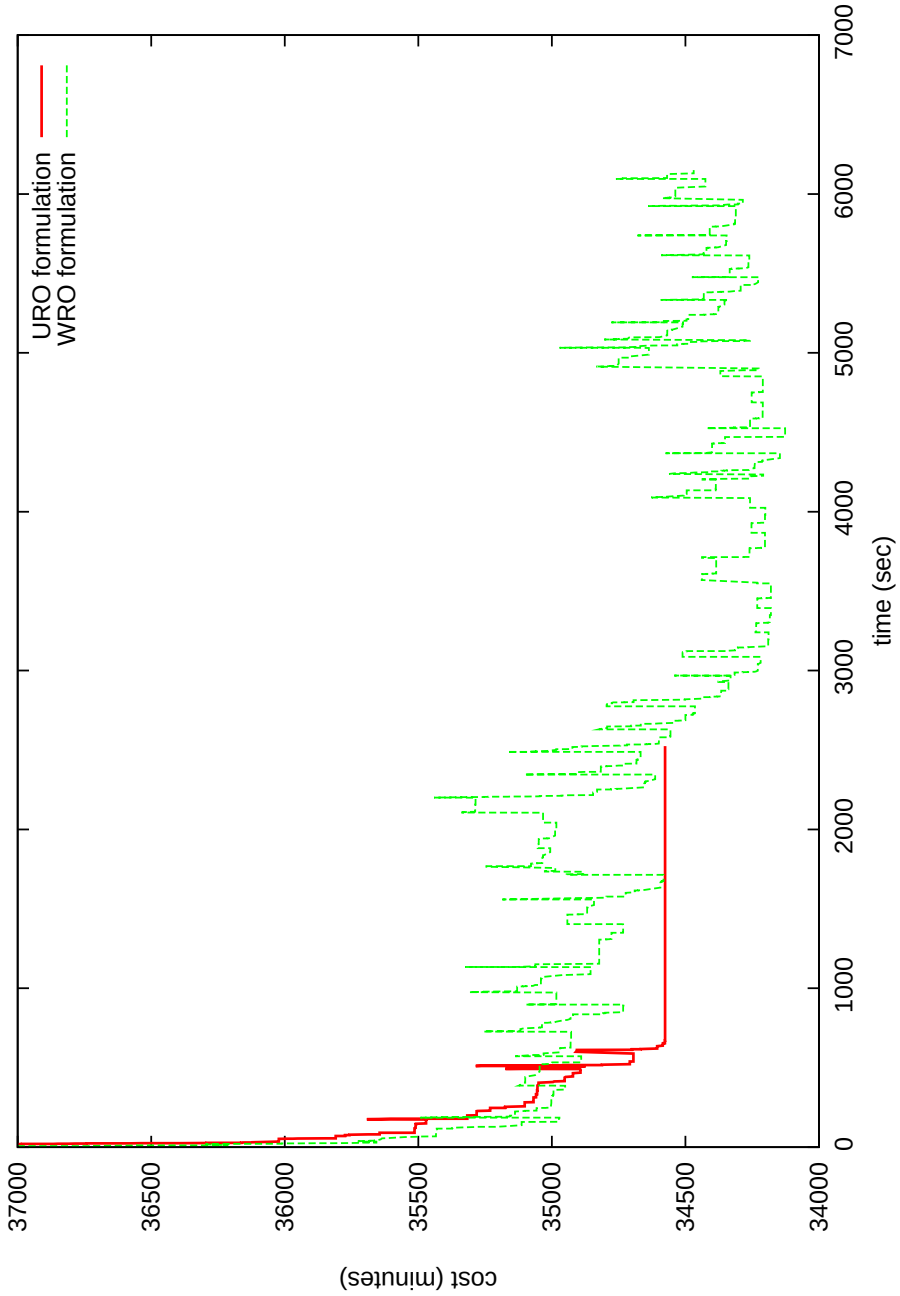

Figure 8.10: Comparison on the execution of the LS on the WRO and URO formulations (Wales dataset). Initial convergence on the URO formulation is faster, but the search gets trapped in a local optimum of higher cost than solutions found on the WRO formulation. Lack of cost-increasing moves in the URO formulation is explained by the inefficiency-correcting being unable to find replacement shifts for the inefficient shifts it chooses.

Chapter 9

Conclusions

In this thesis we study the problem of train driver scheduling with windows of relief opportunities (TDSW). To the best of our knowledge, this problem has only been considered before by Shen [69] and Shen and Kwan [71]. In their work, time windows is not the main focus of the research, and the models and solving tools developed are based on simplified, artificial instances of the problem. In this thesis we concentrate on the time-windows aspect of the problem; also, the research is conscious of the need to make the formulations and solution methods more realistic, producing operational schedules for real-life instances. We try to gain more understanding of the role of WROs in allowing for better solutions to be found; we investigate whether existing approaches to solving the problem on relief-on-arrival formulations can be adapted or extended to exploit WROs, and we propose new methods to solve the problem.

The main contributions of this thesis can roughly be grouped into three categories: those specific to the TDSW problem and understanding of WROs, those that are common to the TDS problem, and those that are general for solving combinatorial optimization problems. In the first group, we believe the main contributions come from:

a) An initial investigation into the impact of considering WROs firstly on problem

formulation and representation, and from there an understanding of the impact on computational approaches for the problem. Our work in Section 2.3 suggests that extending existing formulations that rely on a discrete set of opportunities, e.g. by considering 1-minute expansions of WROs is not viable, mostly because the increase in the number of ROs and the change in the statistical distribution of ROs results in increases in problem sizes that are unmanageable with current approaches.

b) A study on how WROs may allow for more efficient solutions, particularly in relation to scheduling constraints (Chapter 5). It is our belief that experiments in Chapters 5 and 8 clearly show that WROs may play a role in achieving more efficient driver schedules, while at the same time providing evidence that most ROs within WROs may not be instrumental in obtaining those solutions; considering these two observations in combination may lead to better tools for solving the problem.

c) A number of novel proposals (Chapters 4, 5 and 8) that incorporate WROs into both existing and new frameworks, making heavy use of hybridization as a way of exploiting previous research and development. The first proposal in Chapter 5 is able to generate new best-known, operational schedules for real-life instances of the driver scheduling problem. The proposal in Chapter 8 is able to improve on solutions obtained using `PowerSolver` on a URO formulation, and –even when not designed with that intention– it is in some cases also able to produce more efficient solutions than `TrainTRACS` when starting from an empty solution. At the time of writing this thesis, a *beta* trial version of a tool derived from the proposals in Chapter 5 has been made available to some `TrainTRACS` users.

From the point of view of the TDS problem, we provide an investigation into repair heuristics that are based on (re)using the set $\mathfrak{C}$ of valid shifts built during the generation phase of GaS, and in particular its potential use as a powerful way of perturbing solutions, as highlighted in the experiments in Chapter 8, where *exploration* phases are sometimes more instrumental than *standard* phases in improving cost.

In the area of combinatorial optimization, we present a way of allowing infeasible

solutions during the execution of a search, in which infeasible solutions are still costed in a measure that is directly related to the original objective function (Chapters 6 and 8). Although this has been partially studied before, particularly in the area of Genetic Algorithms, work has been mostly oriented to making individual solutions better (e.g. by local search). Instead, we use these results to build a framework in which the capability to work over the space of infeasible solutions is exploited to allow for the design of complex moves that are not subject to generating a feasible solution as a result.

Our ultimate goal in relation to the use of repair costing is to devise local search frameworks where the amount of infeasibility can be decoupled and eventually made orthogonal to the process of comparing candidate solutions during the search – by doing so, we can be much more adventurous in the design of moves and neighbourhoods. Experiments in Chapter 8 suggest that this type of approach is possible; however, there are considerable hurdles to overcome to reach a point where this can be exploited in full. In particular, the aim of making feasibility as orthogonal to costing as possible implies that repair mechanisms should exhibit this property; for example, an 'efficiency gap' in cost between feasible and infeasible solutions (as discussed in Chapter 6) breaks this orthogonality. On the other hand, repairing necessarily has to be a low-order-execution-time operation. Low-order heuristics are usually tied to the problem being solved, which makes this part of the system problem-dependent; for example, the strategy of relying on a pool of partial solutions (e.g. shifts in TDS) may not be applicable to certain optimization problems.

Complete orthogonality cannot be achieved, at least with our proposed repair mechanisms, since e.g. repairing a schedule where all shifts are infeasible is equivalent to solving the full TDSW problem. Therefore, controlling the amount of infeasibility may be needed. Although the degradation in performance of the repair heuristics acts as a self-controlling mechanism on the amount of infeasibility, experiments in Chapter 8 suggest that more explicit control could be beneficial. Overall, we believe that the advantages of successfully implementing a general-purpose scheme along these lines would be such that it definitely merits more research.

9.1 Further Work

Aside from the long-term, optimization-wide goal of encapsulating the distinction between feasible and infeasible solutions out of the search, there are a number of issues that are specific to TDS and TDSW that arise from this thesis, and which are worth exploring. We highlight those we consider most important in the following paragraphs.

Better understanding of the potential of WRO formulations In this thesis we have looked at expanding the driver scheduling model by considering windows of relief opportunities. A valid research question is to understand the limits in efficiency improvements achievable by switching to a WRO formulation. While we have presented instances where we can prove that the expansion leads to better optimal solutions, and we have also conducted experiments that suggest that this is also true for real-life instances of the problem, we believe that to answer the question more fully it is necessary to know how much better solutions to real-life instances can be in theory, that is, decoupled from specific algorithms. In order to give an algorithm-independent answer, it would be necessary to solve a number of real-life problem instances to optimality for both RoA and WRO formulations; however, it is not clear that this is possible with the algorithms currently at hand.

Use of local search as a tool to generate approximations for GaS In Chapter 4 we studied a mechanism involving repeated, alternated calls to a GaS solver and a local search phase. We hinted that the local search phase could be seen as a tool to generate approximated formulations that would lead to increasingly efficient solutions found by the GaS. We believe this is a very promising approach that combines the biggest strengths of both mathematical programming approaches and neighbourhood search mechanisms. However, much more work is needed to integrate these two phases, particularly in terms of enhancing the quality of information fed by the local search to the following GaS phase, and how the GaS uses this information.

General improvements/alternatives to the local search mechanisms Time constraints forbid us from investigating all lines of research. In particular, we would

have liked to tackle the following areas within the local search proposals:

- To show the ability of local search approaches to enforce schedule-level constraints: although it is easy from an implementation point of view to add checks for any given constraint to a solution generated during the search, new constraints would usually result in reduced search-space connectivity, to the point where the search scheme or moves might need to be reconsidered. This is also a valid question for the repair heuristics.

- It would be useful to consider the possibility of using a (long) unattended WRO as if it were an attended WRO, in the sense of drivers covering the full window and handing over inside the window. This would have been a more encompassing model of windows, and might show further improvements in solution quality over RoA models.

- Based on the explosion in the number of legal shifts when considering WROs, we propose a local search scheme where some moves work at a piece-of-work level; a valid research question is whether restricting moves to work at a spell level only would result in more cost-effective algorithms.

Use of WROs to enhance robustness Although we have analyzed some measures of robustness, and adjusted local search proposals to deal with them (Chapter 4), we believe there is much more that could be done in this area. Robustness of a schedule is by nature a statistical measure, since it deals with delays to unforeseen events, which imply a source of randomness. Indicators of this kind are usually measured through simulation; in particular, robustness of a schedule could be measured by 'running' a schedule through a number of delay scenarios, injecting delays (with some randomness) to the operations on each run. However, evaluating each solution through simulation would be prohibitively time-consuming. Hence, the study of low-order functions that reliably estimate robustness is a very important area of research.

Exploiting WROs in the repair heuristic A limitation of the heuristic repair mechanisms used to cost schedules in Chapters 6 and 8 is that, because the

heuristics rely on the pool of valid shifts $\mathfrak{C}$ generated on the RoA formulation, they will effectively not consider WROs when repair-costing. A first idea to solve this is would be to have the repair heuristics work on the pool of valid shifts from the WRO formulation; however, the starting point of this research is the fact that GaS doesn't scale to WRO formulations, mostly because of the explosion in the size of the pool of valid shifts, so this is not a viable option. Some approaches we consider promising are:

- Let the repair heuristics work on a pool obtained from a URO formulation; although experiments in Section 8.3 suggest that working on a URO formulation doesn't translate into better schedules, the proposal here is different in that *standard* phases would still work on a WRO formulation, whereas in Section 8.3 they work over a URO formulation as well.

- Dynamically enlarge the pool of valid shifts used by the repair heuristic during *exploration* phases with any new shifts created during *standard* phases which actively use WROs; careful consideration should still be given in this case to how the size of the pool is kept under control, although we know it will be bounded by the number of solutions generated during the search.

- Have the repair heuristics use a pool of valid *spells*, rather than shifts, since the pool of valid spells is likely to be of a manageable size even in WRO formulations; a drawback of this approach is that shifts would have to be regenerated on each call to the repair heuristics, and having a repair heuristic efficiently generate valid shifts out of these spells may be difficult to achieve.

More comprehensive study of the role of moves in search space connectivity One of the themes of this thesis is how moves and neighbourhoods determine (or restrict) connectivity between solutions in the search space. In this sense, it is useful to see neighbourhoods as restricting the set of feasible solutions that can be reached during the execution of a search. Chapters 6 and 8 are in big part motivated by the need to increase connectivity from that in the original proposals in Chapter 4. The effect of the choice of moves on the resulting search landscape can be very difficult to gauge; for example, the structure of moves designed in Chapter 4 leads

to a very particular restriction in connectivity: because all moves keep or reduce the amount of overcover, solutions in the search cannot have more overcover than the initial solution in the search. This is clearly a restriction, since the optimal solution may well have more overcover than the solution chosen to start the search. While this is (implicitly) addressed by the repair heuristics in Chapter 6, it shows how subtle the effect of moves can be on search space connectivity – hence, more work should be carried in this direction.

Column generation on WRO formulations If the details of constraints are omitted from the description of the problem of train driver scheduling, solving the problem under a WRO formulation looks like the standard situation where column generation may be of use. Unfortunately once constraints are added, the pricing problem becomes NP-hard; even if some recent work claims success in considering more complex constraints/penalties in the context of branch-and-price, it looks like further development is needed before column generation can be considered on full WRO models.

Appendix A

Modelling of WROs

Any algorithm for solving the driver scheduling problem with windows of relief opportunities relies, explicitly or implicitly, on a model for WROs. Given that the driver scheduling problem is already very hard to solve on relief-on-arrival assumptions, special care must be placed when modeling WROs to achieve the right balance between generality and the difficulty in solving the resulting problem instances. Also, solutions allowed in a model must be executable in real life[1].

A.1 Covering of WROs

A feasible schedule has to cover all WROs with drivers. In this sense, a WRO could be seen as a piece of work. However, the similarities end as soon as a relief happens within a WRO at a time other than the start or end of the time window, since pieces of work cannot by definition be covered partially. More generally, there are many ways in which a WRO can be covered by drivers. In this section we examine a number of options regarding covering of WROs, analyzing them in terms of its implications on applicability, schedule robustness, and size of the solution space; we

[1] otherwise there must be a suitable transformation to make them executable, which doesn't affect its perceived cost

will start from the most general, and then propose several constrained versions.

A.1.1 Unconstrained Model

In this case, the only restriction is that the any feasible schedule must cover all WROs completely. Crucially, this formulation allows for drivers who are not assigned to drive any of the pieces of work limiting with the WRO to cover part of the WRO. The model is illustrated in Figure A.1.

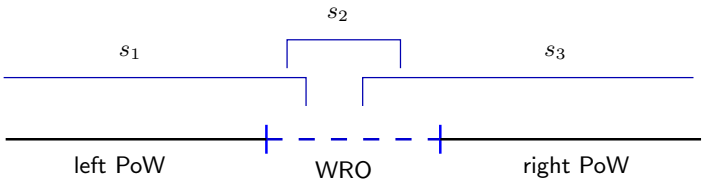

Figure A.1: Unconstrained covering of WROs. Shifts that are not assigned to cover any of the pieces of work limiting with the WRO are allowed to cover part of the WRO.

This model is undesirable from more than one reason. First, from a robustness point of view it seems counterproductive, as it makes the correct execution of the schedule on this WRO to depend on more than two drivers, among others increasing the number of hand-overs. Also, the model would result in an extremely high number of possible ways to cover a WRO. The expected gain in modeling power and efficiency seems more than offset by the loss in schedule robustness and the increase in computational complexity.

A.1.2 Constrained Version I

Our first constraint to the general model is then to disallow drivers who don't cover one of the pieces of work adjacent to the WRO to cover a part of the WRO. A sample allowed way of covering a WRO is presented in Figure A.2.

It is interesting to note that in this model, given any feasible schedule $\mathcal{S}$ for instance $\mathcal{I}$, and a WRO $w \in \mathcal{I}$, it is always possible to select two shifts $s_l, s_r \in \mathcal{S}$ such that w is completely covered by them. This is even when overcovering of work is allowed; one way to select these shifts is to choose s_l to be the shift covering the most of w from those covering the piece to the left of w, and s_r the one covering the

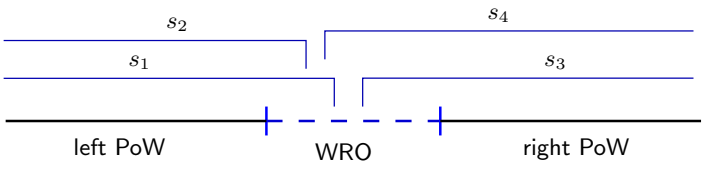

Figure A.2: Covering of WROs – constrained version I. Shifts that are not assigned to cover any of the pieces of work limiting with the WRO are not allowed to cover part of the WRO.

most of w from those covering the piece to the right (in Figure A.2 this would yield $s_l = s_1, s_r = s_4$).

The model is better than the unconstrained model. However, from an algorithmic point of view the number of combinations for covering a WRO w is still very big: for n shifts covering a WRO that is m minutes long (allowing for overcover), there are $O(m^n)$ such combinations. We have shown that in practice covering of w can be assessed by looking at only two shifts from those covering w; therefore, the (very limited) expressiveness of the model gained from allowing each shift to cover a different part of w does not seem to justify the high number of combinations it allows.

A.1.3 Constrained Version II

We now further constrain the model to avoid the problems associated with the previous model. We do so by forcing all shifts that cover the piece to the left of a WRO to cover exactly the same part of the WRO (in effect meaning that they all leave the train at the same time). The same applies for those shifts covering the piece to the right of the WRO. This model is illustrated in Figure A.3.

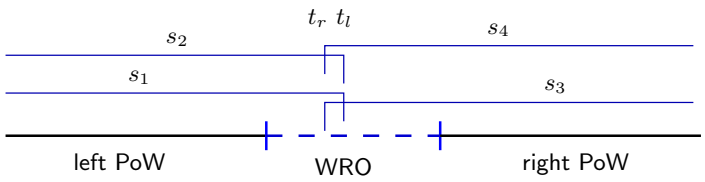

Figure A.3: Covering of WROs – constrained version II. All shifts covering the piece to the left of the WRO leave the train at the same time t_l; all shifts covering the piece to the right board the train at time t_r.

In this model, the way the WRO is covered is determined by two variables: the

time t_l at which all shifts covering the left piece leave the train, and the time $t_r < t_l$ when all shifts covering the right piece board the train. This means that for a WRO w that is m minutes long, there are $O(m^2)$ combinations to cover w when relieving drivers, which is potentially a big decrease in size over the previous model.

It is worth noting that this model still allows to exploit WROs to improve schedule robustness in the way described in section 1.3. Another potentially useful feature of this model is that it is very easy to eliminate overcover while keeping WROs attended, since as long as one leaves one shift covering each side of the WRO one is guaranteed to keep the WRO covered. In the previous model, one would instead have to look at the times at which all shifts leave or board the train before deciding whether a particular shift can be removed from covering the WRO.

A.1.4 Constrained Version III

While the model just introduced seems to strike a good balance between expressive power and computational complexity, there may be situations when analyzing $O(m^2)$ ways of covering a WRO may be too time-consuming or deemed unnecessary. The next natural simplification of the model is to further constrain t_l and t_r, by requiring $t_l = t_r + k$, for a fixed value of k. This reduces the number of ways to cover m-minute long WRO w to $O(m)$. An example is shown in Figure A.4.

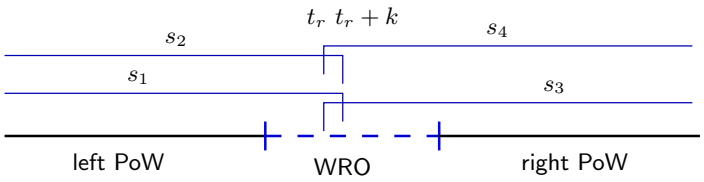

Figure A.4: Covering of WROs – constrained version III. This model is defined by constraining version II by requiring $t_l = t_r + k$.

This further constraining comes at a price in terms of expressive power, since the amount of 'buffering' between shifts on the left and right sides of the WRO is set in advance. This can be partially amended if the scheduler knows how many minutes of buffering he wants at a particular WRO w beforehand, by setting a specific value of k for w. Also, for a m-minute long WRO w, if $k > m$ the constraint $t_l = t_r + k$ would make covering of that WRO impossible. This can be easily fixed by requiring

$t_l = \min(t_r + k, t_e)$, where t_e is the end-time of w. Finally, it is possible to force no buffering by setting $k = 0$; this is shown in Figure A.5.

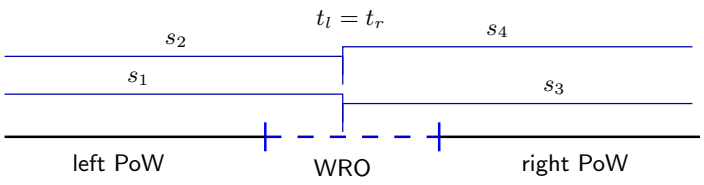

Figure A.5: Covering of WROs – constrained version III, $k = 0$.

Appendix B

Parameter Setup for Experiments in Chapter 8

Figure B.1 summarises the main parameters that can affect the behaviour of the proposal in Chapter 8. For each such parameter, we report the choice taken for the experiment number 2 as described in Section 8.2.4. In the following paragraphs we briefly comment on the choice of values for some parameters, where the motivation is not clear from the description in Chapter 8.

Structuring of *standard* and *exploration* phases. As in the initial proposal in Chapter 4, we use predetermined sets of moves structured as sequences of L_1 moves; in turn, both *exploration* and *standard* phases consist of a series of iterations over those sequences. Iterations within a single *standard* or *exploration* phase provide the search the opportunity to execute certain moves after other types of move have already been made – our experience shows that this allows for a better exploration of the search space. The number of iterations was determined experimentally, by evaluating increasing the number of iterations within a phase up to a point where the marginal improvement in resulting solution costs made extra phases unnecesary or not cost-effective in terms of overall execution times. An intuitive alternative to

allow for different combinations of moves to happen in different orders would be to randomize the selection of the type of move on every move. However, we have used the former mechanism because a non-randomized setup makes it easier to control, analyze and explain the effects of multi-neighbourhood exploration over time.

Number of re-trials when repair-costing a solution. Although experiments in Chapter 6 suggest that the randomized repair heuristics can find better-cost solutions as the number of re-trials is incremented, we consider that in the context of a local search, a similar effect will be achieved even if no re-trials are executed during evaluation of a single solution, since many moves are likely to result in the same effective schedule (after removing infeasible shifts) presented to the repair; hence we set the number of re-trials to one (e.g. no re-trials). Experiments suggest that this does not affect the quality of results obtained significantly, while reducing the execution times considerably.

Global parameters	
termination criterion	total time (7,200 seconds)
cost function	total hours (Equation 2.2)
initial solution	obtained using `TrainTRACS` on a RoA formulation
duration of tabu tenure for attributes derived from inefficiency-correcting move	one *exploration* phase
standard **phase**	
formulation	WRO
move acceptance criterion	non-decreasing solution cost, new solution satisfies all problem constraints
structuring of moves	*standard*-specific predefined sequence of L_1 moves
termination criterion	three iterations over the sequence above
exploration **phase**	
formulation	RoA
move acceptance criterion	non-decreasing solution cost, new solution only required to satisfy 'structural' constraints; no limit on the number of infeasible shifts in the new solution
structuring of moves	*standard*-specific predefined sequence of L_1 moves
termination criterion	two iterations over the sequence above
other settings	active solution is repaired after each iteration of L_1 moves
Repair costing	
heuristic	SR-rLCP-QW (Section 6.6)
number of re-trials per repair costing operation	no re-trials
Inefficiency-correcting move	
inefficiency exploited	high joinup time
selection of inefficient shift	randomized; probability of selection increasing on total shift joinup time

Figure B.1: Summary of parameter setup for the main experiments in Chapter 8.

Appendix C

Notation and Abbreviations

Notation

$\mathfrak{C}$ (candidate)	pool of shifts obtained during a generation phase
$cost(\mathcal{S})$ (schedule), $c(s)$ (shift)	cost function
$\mathcal{I}$	problem instance
p	piece of work
r	RO
$\mathcal{S}$	schedule
s	shift
sp, $s_{k,i}$ (spell i in shift k)	spell
v	vehicle
w	WRO

Abbreviations

CG	Column Generation
GaS	Generate and Select
ILP	Integer Linear Program / Programming
LP	Linear Program / Programming
LS	Local Search
PoW	piece of work
RO	relief opportunity
RoA	relief-on-arrival
TDS	Train Driver Scheduling (problem)
TDSW	Train Driver Scheduling (problem) with Time Windows
TOC	Train Operating Company
WRO	window of relief opportunities

Bibliography

[1] Emile Aarts and Jan K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1997.

[2] Y. Ageeva. Approaches to incorporating robustness into airline scheduling. Master's thesis, Massachusetts Institute of Technology, 2000.

[3] Ravindra K. Ahuja, Jian Liu, James B. Orlin, Jon Goodstein, and Amit Mukherjee. A neighborhood search algorithm for the combined through and fleet assignment model with time windows. *Networks*, 44(2):160–171, 2004.

[4] Uwe Aickelin, Edmund Burke, and Jingpeng Li. Improved squeaky wheel optimisation for driver scheduling. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN IX)*, 2006.

[5] J. Mark Baldwin. A new factor in evolution. *American Naturalist*, 30:441–457, 536–554, 1896.

[6] Nicolas Belanger, Guy Desaulniers, Francois Soumis, and Jacques Desrosiers. Periodic airline fleet assignment with time windows, spacing constraints, and time dependent revenues. *European Journal of Operational Research*, 175(3):1754–1766, December 2006.

[7] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, September 2003.

[8] Ralf Borndörfer, Uwe Schelten, Thomas Schlechte, and Steffen Weider. A column generation approach to airline crew scheduling. In *Operations Research Proceedings 2005*, pages 343–348. 2006.

[9] Edmund K. Burke and Patrick De Causmaecker, editors. *Practice and Theory of Automated Timetabling IV, 4th International Conference, PATAT 2002*, volume 2740 of *Lecture Notes in Computer Science*. Springer, 2003.

[10] Edmund K. Burke and Wilhelm Erben, editors. *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000*, volume 2079 of *Lecture Notes in Computer Science*. Springer, 2001.

[11] Edmund K. Burke and Michael A. Trick, editors. *Practice and Theory of Automated Timetabling V, 5th International Conference, PATAT 2004*, volume 3616 of *Lecture Notes in Computer Science*. Springer, 2005.

[12] L. Cavique, C. Rego, and I. Themido. Subgraph ejection chains and tabu search for the crew scheduling problem. *The Journal of the Operational Research Society*, 50(6):608–616, 1999.

[13] Peter I. Cowling, Graham Kendall, and Eric Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *PATAT '00: Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, pages 176–190, London, UK, 2001. Springer-Verlag.

[14] S. Curtis, B. Smith, and A. Wren. Forming bus driver schedules using constraint programming. In *Proceedings of the First International Conference on the Practical Applications of Constraint Technologies and Logic Programming (PACLP99)*, pages 239–254, 1999.

[15] J. R. Daduna, Isabel Branco, and Jose M. P. Paixao, editors. *Computer-Aided Transit Scheduling*, volume 430 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, 1995.

[16] J.R. Daduna and Anthony Wren, editors. *Computer-aided transit scheduling*, Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, 1988.

[17] Alain Dallaire, Charles Fleurent, and Jean-Marc Rousseau. Dynamic constraint generation in CrewOpt, a column generation approach for transit crew scheduling. Presented at CASPT 2004, San Diego, 2004.

[18] Amal de Silva. Combining constraint programming and linear programming on an example of bus driver scheduling. *Annals of Operations Research*, 108(1):277–291, November 2001.

[19] Department for Transport (DfT), UK. *Stakeholder Briefing Document East Midlands franchise*, October 2006.
http://www.dft.gov.uk/pgr/rail/passenger/franchises/em/stakeholderbriefingdocumente1631.

[20] G. Desaulniers, J. Desrosiers, I. loachim, M. M. Solomon, F. Soumis, and D. Villeneuve. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. G. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, pages 57–93. Kluwer, Norwell, MA, 1998.

[21] Guy Desaulniers, Jacques Desrosiers, Yvan Dumas, Marius M. Solomon, and F. Soumis. Daily aircraft routing and scheduling. *Management Science*, 43(6):841–855, June 1997.

[22] M. Desrochers and J. M. Rousseau, editors. *Computer-Aided Transit Scheduling*, volume 386 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, 1992.

[23] Matthias Ehrgott and David M. Ryan. Constructing robust crew schedules with bicriteria optimization. *Journal of Multi-Criteria Decision Analysis*, 11(3):139–150, 2002.

[24] Thomas Emden-Weinert and Mark Proksch. Best practice simulated annealing for the airline crew scheduling problem. *Journal of Heuristics*, 5(4):419–436, 1999.

[25] Torsten Fahle, Ulrich Junker, Stefan E. Karisch, Niklas Kohl, Meinolf Sellmann, and Bo Vaaben. Constraint programming based column generation for crew assignment. *Journal of Heuristics*, 8(1):59–81, January 2002.

[26] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[27] Matteo Fischetti, Silvano Martello, and Paolo Toth. The fixed job schedule problem with working-time constraints. *Operations Research*, 37(3):395–403, 1989.

[28] Matteo Fischetti, Silvano Martello, and Paolo Toth. Approximation algorithms for fixed job schedule problems. *Operations Research*, 40(Supplement 1):S96–S108, 1992.

[29] L. R. Ford and D. R. Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 50(1):97–101, 1958.

[30] Sarah Fores. *Column Generation Approaches to Bus Driver Scheduling*. PhD thesis, School of Computing, University of Leeds, 2001.

[31] Sarah Fores, Les Proll, and Anthony Wren. Experiences with a flexible driver scheduler. In S. Voss and J. R. Daduna, editors, *Computer-Aided Scheduling of Public Transport, Lecture Notes in Economics and Mathematical Systems 505*, pages 137–152. Springer-Verlag, 2001.

[32] Sarah Fores, Les Proll, and Anthony Wren. TRACS II: a hybrid IP/heuristic driver scheduling system for public transport. *Journal of the Operational Research Society*, 53:1093–1100, 2002.

[33] Richard Freling, Dennis Huisman, and Albert P. M. Wagelmans. Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling*, 6(1):63–85, 2003.

[34] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[35] M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, 1994.

[36] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[37] Fred Glover and Manuel Laguna. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653–684, 2000.

[38] Balaji Gopalakrishnan and Ellis Johnson. Airline crew scheduling: State-of-the-art. *Annals of Operations Research*, 140(1):305–337, November 2005.

[39] K. Haase, G. Desaulniers, and J. Desrosiers. Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Science*, 35(3):286–303, 2001.

[40] P. Hansen and N. Mladenović. Variable neighbourhood search. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, chapter 6, pages 145–184. Springer, 2003.

[41] A. Hertz. Finding a feasible course schedule using tabu search. *Discrete Applied Mathematics*, 35:255–270, 1992.

[42] Mark Hickman, Pitu Mirchandani, and Stefan Voss, editors. *Computer-aided Systems in Public Transport*, volume 600 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, 2008 (to appear).

[43] G. E. Hinton and S. J. Nowlan. How learning can guide evolution. *Complex Systems*, 1:495–502, 1987.

[44] Sin Ho and Michel Gendreau. Path relinking for the vehicle routing problem. *Journal of Heuristics*, 12(1-2):55–72, January 2006.

[45] Dennis Huisman. *Integrated and Dynamic Vehicle and Crew Scheduling*. PhD thesis, Tinbergen Institute, Erasmus University Rotterdam, 2004.

[46] Toshihide Ibaraki, Koji Nonobe, and Mutsunori Yagiura, editors. *Metaheuristics: Progress as Real Problem Solvers (MIC 2003)*, volume 32 of *Operations Research / Computer Science Interfaces Series*. Springer, 2005.

[47] Hisao Ishibuchi, Shiori Kaige, and Kaname Narukawa. Comparison between lamarckian and baldwinian repair on multiobjective 0/1 knapsack problems. In *Evolutionary Multi-Criterion Optimization*, pages 370–385. 2005.

[48] David E. Joslin and David P. Clements. Squeaky wheel optimization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), Madison, WI*, pages 340–346, 1998.

[49] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[50] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[51] Diego Klabjan, Ellis L. Johnson, George L. Nemhauser, Eric Gelman, and Srini Ramaswamy. Airline crew scheduling with time windows and plane-count constraints. *Transportation Science*, 36(3):337–348, 2002.

[52] Ketan Kotecha, Gopi Sanghani, and Nilesh Gambhava. Genetic algorithm for airline crew scheduling problem using cost-based uniform crossover. In *Second Asian Applied Computing Conference, AACC 2004, Lecture Notes in Computer Science*, volume 3285, pages 84–91. Springer, 2004.

[53] Ann S. K. Kwan. *Train Driver Scheduling*. PhD thesis, School of Computing, University of Leeds, 1999.

[54] Ann S. K. Kwan, Margaret E. Parker, Raymond S. K. Kwan, Sarah Fores, Les Proll, and Anthony Wren. Recent advances in TRACS. In *Proceedings of the 9th International Conference on Computer-Aided Scheduling of Public Transport*, 2004.

[55] Raymond S. K. Kwan. Bus and train driver scheduling. In J. Y-T Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 51. CRC Press, 2004.

[56] Raymond S. K. Kwan and Ann S. K. Kwan. Effective search space control for large and/or complex driver scheduling problems. *Annals of Operations Research*, 155(1):417–435, 2007.

[57] Raymond S. K. Kwan, Ann S. K. Kwan, and Anthony Wren. Evolutionary driver scheduling with relief chains. *Evolutionary Computation*, 9(4):445–460, 2001.

[58] David Levine. Application of a hybrid genetic algorithm to airline crew scheduling. *Computers and Operations Research*, 23(6):547–558, 1996.

[59] Jingpeng Li and Raymond S. K. Kwan. A fuzzy genetic algorithm for driver scheduling. *European Journal of Operational Research*, 147(2):334–344, 2003.

[60] Jinpeng Li. *Fuzzy Evolutionary Approaches for Bus and Rail Driver Scheduling*. PhD thesis, School of Computing, University of Leeds, 2002.

[61] Jinpeng Li and Raymond S. K. Kwan. A fuzzy evolutionary approach with taguchi parameter setting for the set covering problem. In D Fogel, editor, *Congress on Evolutionary Computation 2002 Conference*, pages 1203–1208. IEEE Press, 2002.

[62] Marco E. Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

[63] Rob Mills and Richard A. Watson. On crossing fitness valleys with the baldwin effect. In *Tenth International Conference on the Simulation and Synthesis of Living Systems*, pages 493–499, 2006.

[64] Margarida Moz, Ana Respício, and Margarida Vaz Pato. Bi-objective evolutionary heuristics for bus drivers rostering. Presented at CASPT 2006, Leeds, 2006.

[65] Yaghout Nourani and Bjarne Andresen. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41):8373–8385, 1998.

[66] Margaret E. Parker and Barbara M. Smith. Two approaches to computer crew scheduling. In A.Wren, editor, *Computer Scheduling of Public Transport*, pages 193–222. North-Holland, 1981.

[67] Prapa Rattadilok and Raymond S. K. Kwan. Dynamically configured λ-opt heuristics for bus scheduling. Presented at PATAT 2006, Brno, Czech Republic, 2006.

[68] J.M. Rousseau, editor. *Computer Scheduling of Public Transport 2*. North Holland, 1985.

[69] Yindong Shen. *Tabu Search for Bus and Train Driver Scheduling with Time Windows*. PhD thesis, School of Computing, University of Leeds, 2001.

[70] Yindong Shen and Raymond S. K. Kwan. Tabu search for driver scheduling. In S. Voss and J. R. Daduna, editors, *Computer-Aided Scheduling of Public Transport, Lecture Notes in Economics and Mathematical Systems 505*, pages 121–135. Springer-Verlag, 2001.

[71] Yindong Shen and Raymond S. K. Kwan. Tabu search for time windowed public transport driver scheduling. Technical report, School of Computer Studies, University of Leeds, 2002.

[72] Nadia Souai and Jacques Teghem. Hybridizing the genetic algorithm and the simulated annealing for the airline crew rostering problem. Presented at CASPT 2006, Leeds, 2006.

[73] Ingmar Steinzen, Vitali Gintner, and Leena Suhl. Network models for a decomposed pricing problem in crew scheduling. Presented at CASPT 2006, Leeds, 2006.

[74] M. Stojkovic and F. Soumis. An optimization model for the simultaneous operational flight and pilot scheduling problem. *Management Science*, 47(9):1290–1305, 2001.

[75] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[76] Jonathan Thompson and Kathryn Dowsland. General cooling schedules for a simulated annealing based timetabling system. In *Practice and Theory of Automated Timetabling*, pages 345–363. 1996.

[77] S. Voss and J.R. Daduna, editors. *Computer-Aided Scheduling of Public Transport*, volume 505 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, 2001.

[78] Cameron G. Walker, Jody N. Snowdon, and David M. Ryan. Simultaneous disruption recovery of a train timetable and crew roster in real time. *Computers and Operations Research*, 32(8):2077–2094, 2005.

[79] Steffen Weider. *Integration of Vehicle and Duty Scheduling in Public Transport*. PhD thesis, Technische Universität Berlin, 2007.

[80] Darrell L. Whitley, V. Scott Gordon, and Keith E. Mathias. Lamarckian evolution, the baldwin effect and function optimization. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature – PPSN III*, pages 6–15, Berlin, 1994. Springer.

[81] N. H. M. Wilson, editor. *Computer-Aided Transit Scheduling*, volume 471 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, 1999.

[82] Anthony Wren, editor. *Computer Scheduling of Public Transport*, volume 386 of *Second International Workshop on Computer-Aided Scheduling of Public Transport*. North-Holland, 1981.

[83] Anthony Wren, Sarah Fores, Ann S. K. Kwan, Raymond S. K. Kwan, Margaret Parker, and Les Proll. A flexible system for scheduling drivers. *Journal of Scheduling*, 6(5):437–455, 2003.

[84] Anthony Wren and Jean-Marc Rousseau. Bus driver scheduling – an overview. In J.R. Daduna, I. Branco, and J.M.P. Paixao, editors, *Computer-aided Transit Scheduling*, pages 173–187. Springer, 1995.

[85] Shangyao Yan and Chich-Hwang Tseng. A passenger demand model for airline flight scheduling and fleet routing. *Computers and Operations Research*, 29(11):1559–1581, September 2002.

[86] Joyce W. Yen and John R. Birge. A stochastic programming approach to the airline crew scheduling problem. *Transportation Science*, 40(1):3–14, 2006.