A Constraint Programming Pre-Processor for Duty Scheduling

by

Colin J. Layfield

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

The University of Leeds
School of Computing

September 2002

The candidate confirms that the work submitted is his own and the appropriate credit has been given where reference has been made to the work of others.

Abstract

The bus driver scheduling problem involves assigning a set of drivers to cover all available bus work such that every bus is assigned a driver, the number of duties is minimised and each duty conforms to the rules governing them regarding maximum driving time and so on. Generally this problem is solved using mathematical programming methods. The University of Leeds has developed a driver scheduling system, TRACS-II, that solves the bus driver scheduling problem by first generating a large set of potential duties and selecting a subset of these via the associated set covering ILP to form the schedule.

The size of the set of potential duties used by TRACS-II is directly related to the number of *relief opportunities* present in the original bus schedule. Each relief opportunity potentially serves as a handover point between two bus shifts. A bus schedule containing many relief opportunities can have a very large set of potential shifts generated to cover the buswork. The complexity of the ILP is related to the number of relief opportunities present in the bus schedule.

This thesis describes a *pre-processing* stage for the TRACS-II scheduling system. The pre-processor selects potentially useful relief opportunities from bus schedule. The shifts generated by TRACS-II are restricted to the subset of relief opportunities made available to it by the pre-processor. The reduction of the number of relief opportunities serves to reduce the complexity of the resulting set covering problem by reducing the number of variables and constraints in the ILP.

The pre-processor itself uses constraint programming to find several possible ways of selecting relief opportunities from the morning and evening portions of the bus schedule. This is done by exploiting useful properties found in good driver schedules, specifically the chaining together of driver duties such that when one driver takes a break another driver finishing a break continues on the same bus. The pre-processor described has been shown to be effective on a wide variety of schedules in that the minimum number of drivers is almost always used and, in some cases, cheaper schedules can be produced.

Acknowledgements

First and foremost I'd like to offer my thanks to my supervisors Dr. Barbara Smith and Professor Anthony Wren for both their guidance and support.

The interest and support of my friends and family is also appreciated. The support from my parents has also inspired me to be able to finish this piece of work. My partner Stella has also been most helpful and understanding during the writing up phase. Again, a big thank you is in order.

Last but not least, I'd like to acknowledge Dr. Anton Colijn whose belief in me all those years ago has enabled me to experience and do the things I've done in my academic career thus far. Keep writing those funny programs!

Declarations

Some parts of the work presented in this thesis have been published in the following articles:

B M. Smith, C. J. Layfield and A. Wren, *A Constraint Programming Pre-processor for a Bus Driver Scheduling System.* In E. Freuder and R. Wallace, editors, *Constraint Programming and Large Scale Optimization*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, Volume **57**, 2001, pp. 131-148.

C. J. Layfield, B. M. Smith and A. Wren, *Bus relief point selection using constraint programming.* Proceedings of the 1st International Conference on the Practical Applications of Constraint Technologies and Logic Programming, PACLP99, The Practical Application Company, 1999, pp. 537-552.

C. J. Layfield, B. M. Smith and A. Wren, *Relief Opportunity Selection for Bus Driver Scheduling*, Fourth ILOG International Users Meeting, October 5 & 6, Paris, France, 1998.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Both vehicle and driver scheduling problems are commonplace anywhere in the world that public transport is available. A bus company generally has the responsibility of providing bus services for a given area. Most of the routes involved are regularly scheduled; this includes peak-only vehicles which deal with the afternoon and evening rush hours. Occasionally, there are also special routes contracted out to private clients such as schools or factories. The bus routes must be designed to provide the service needed and drivers must be assigned to the relevant vehicles to meet this demand.

A vehicle schedule defines the routes and the stopping points for the vehicles involved as well as the allocation of vehicles to be used on these routes. The stopping points selected include both points for collecting and dropping off passengers as well as changing over drivers; the latter of these referred to as *relief opportunities*. Given a vehicle schedule a problem that must be addressed, and the topic of interest in this thesis, is the *driver* or *duty scheduling* problem. The task of duty scheduling involves assigning drivers (duties) to the bus schedule in such a way as to minimize the number of drivers needed as well as the costs involved (wages) while ensuring any labour rules are not violated (for example maximum driving times).

The privatization of bus and rail services in Great Britain (and elsewhere) has had some

affect on the scheduling aspects mentioned above. In the past it has generally been the case that once a bus route has been scheduled it is used for some period of time before the need to change it should arise. Today, due to competition from rival public transport firms as a result of privatization, schedules are updated more frequently to respond to service changes a competitor may make. This means that driver scheduling problems are faced more frequently than in the past by public transport companies. Having fast effective methods of dealing with them is essential.

1.1 Driver Scheduling

The problem of driver scheduling is computationally challenging. The number of potential duties that can be created given a bus schedule can be very large. From this large set of duties the duty scheduling system must select the duties that will make up the duty schedule. Many computerized methods have been developed to tackle this, as will be discussed in some detail in Chapter 2, but the reasons why this problem is combinatorially expensive remain.

One of the reasons duty scheduling is so difficult is due to the large number of potential duties that can be created from which to select a duty schedule. The size of this large set of potential duties is directly related to the number of relief opportunities present in the bus schedule. More relief opportunities present in the bus schedule creates more ways in which drivers can be relieved from their buses by other drivers, thus the set of potential duties that can be created is larger.

The complexity arising from the number of relief opportunities present in a bus schedule can be reduced. Specifically, this thesis explores the idea of a *pre-processing* stage for a duty scheduling system that was developed at the University of Leeds known as TRACS-II [39]. The pre-processing stage reduces the number of relief opportunities used in the scheduling process. This is accomplished by first examining the morning and evening periods of a bus schedule. Potentially useful relief opportunities are selected and kept whilst leftover

relief opportunities in the morning and evening periods are discarded. This smaller set of relief opportunities is then passed onto the TRACS-II scheduling system to process in order to generate a duty schedule. This has the effect of reducing the number of potential duties that can be used to create a duty schedule. By selecting relief opportunities that are potentially useful the quality of the duty schedule produced need not be compromised.

Before a discussion of the background of the field as well as the pre-processor some basic driver scheduling terminology will be presented. A more complete list of terms and related synonyms can be found in [57].

1.2 Bus Driver and Vehicle Scheduling Terminology

Generally a bus schedule, or a set of *bus workings*, is represented as a set of *running boards*. A running board represents an individual vehicle and consists of various *relief times*. Each relief time is also associated with a place and the pair of them form a *relief opportunity*. These relief opportunities in between the start and finish time of a running board are usually the times it is possible for a bus to switch crews aboard it. An example of a running board can be found in Figure 1.1. In this example the set of relief opportunities is $\{0529, 0720, 0945, 1015, 1310, 1530, 1650, 1905, 2025, 2205\}$. The relief locations are 'D' (depot) and 'L'.

Figure 1.1: Example of a Running Board

The relief opportunities serve to partition the schedule into pieces of *work*. A piece of work is defined as the section of the running board that falls inbetween two relief opportunities. The driver scheduling problem can be thought of as trying to cover the set of all work in the bus schedule such that every piece of work has at least one driver assigned to it.

Bus crews or *driver duties* or just *duties* are assigned to cover the work on these running boards. A bus crew consists of a driver (or, rarely, a conductor as well; hence the term

crew) that has been allocated to cover a set of work present in the bus schedule.

A duty consists of a set of work it is assigned to cover. Generally the set of work is split into distinct halves. Each of these two sets of work are referred to as a *stretch*. The gap inbetween the two stretches of work is referred to as a *mealbreak* as it is in this period of time where the driver usually takes a break of around an hour for a meal. A mealbreak usually has a minimum length, referred to as the *minimum mealbreak length*. Additionally, the locations where a driver starts a mealbreak and finishes the mealbreak may not be the same. There is usually an associated *travel time* between various relief opportunities which is the allowed travelling time between them (not including the minimum mealbreak length). The minimum length of time that must elapse between a driver starting a mealbreak and finishing one is the minimum mealbreak length plus the allocated travel time between the relief opportunities at which the mealbreak will take place. An additional parameter, known as the *maximum mealbreak idle time*, specifies the maximum length of the mealbreak that may be allowed over and above the minimum mealbreak length (plus travel time).

A stretch can sometimes be subdivided as well. Usually a stretch is a continuous set of work on the same vehicle. Sometimes a stretch will consist of two pieces of work separated by a short gap. This gap is known as a *joinup*. A joinup usually represents the driver leaving one vehicle to take over another one with a very short break inbetween (often just the travel time between the two relief opportunities if they are not the same). The two pieces of work would be referred to as *spells*. A stretch can also be a spell if it is a continuous set of work. A duty that consist of two continuous stretches is referred to as a *two part duty*. If one of the two stretches corresponding to a duty is made up of two spells then that duty is referred to as a *three part duty*. This concept can be extended to four or more part duties. Some duties have a large gap inbetween their stretches (a length of several hours). These are referred to as *split duties*.

An example of a two part duty can be found in Figure 1.2. The driver in this case works two stretches. The first stretch starts at 05:29 to 09:05, a mealbreak is then taken from

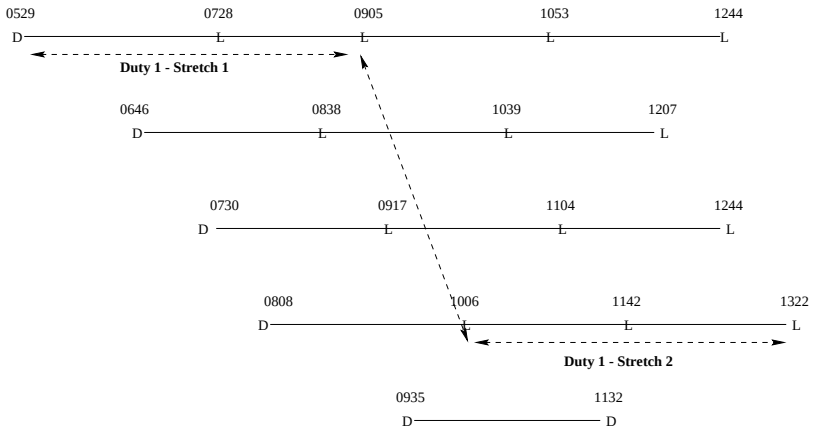

Figure 1.2: Example of Duty

9:05 to 10:06 and then the second stretch goes from 10:06 to 13:22.

When a driver starts and finishes their day they are allocated a set period of time, known as the *sign-on* and *sign-off* times, in order to clock in and out. These times are usually included in the cost calculations for first and last stretch of their duty.

Throughout the schedule drivers need to take their mealbreaks. After a break, the driver is again assigned to work on a vehicle. Often this takes over a bus from another driver who goes on a break. The chaining of such breaks such that one driver starting work after a break takes over on a bus whose current driver is about to start a break is referred to as a *mealbreak chain*. Mealbreak chains have many beneficial properties in a duty schedule and will be discussed in detail in Chapter 5.

The *cost* of a schedule can be defined as the sum of the cost of the individual duties that it contains. The cost of an individual duty is defined as the cost of the time worked. This can further be adjusted by various penalties due to unusual or unpopular features in the duty (for example, unsociable hours). The method of calculating the cost of a duty varies between bus companies.

1.3 Thesis Outline

This thesis describes a pre-processing module for the TRACS-II mathematical programming duty scheduling system. As the complexity of the problem is related to both the number of relief opportunities present in the set of bus workings as well as the rules that govern what makes a legal duty, the goal of the pre-processor is to reduce the number of relief opportunities present in a set of bus workings before TRACS-II is applied to it.

Constraint programming was chosen as the methodology for the pre-processor. The task of selecting relief opportunities involves many constraints between the relief opportunities themselves. The expressiveness of modelling such a problem using constraint programming methodologies was felt to be of benefit in the design of the solution as well as using ILOG-Solver to solve it. The research this thesis describes involves examining the properties of mealbreak chains and trying to exploit any perceived properties they may have with respect to a duty schedule.

The following is an outline of the contents of each chapter.

Chapter 2 - Gives a review of past and present computerised duty scheduling systems employing such techniques as heuristics, mathematical programming and meta-heuristics.

Chapter 3 - Reviews the topic of constraint satisfaction problems. A description of what makes a CSP as well as some of the methods for solving them is described. The ILOG-Solver product that is used in this thesis is described and discussed. Some constraint programming duty scheduling systems are also briefly described.

Chapter 4 - A description of the TRACS-II duty scheduling system.

Chapter 5 - A definition of mealbreak chains is given as well as a description of why they are beneficial. Some previous research regarding how mealbreak chains can be found and a discussion of these is given.

Chapter 6 - The CROSS constraint satisfaction problem is described. This discussion includes a description of the constraints employed and how they interact with one another.

Chapter 7 - The algorithm used for solving the CROSS constraint satisfaction problem is described. How CROSS can be applied to different types of bus schedules is also described.

Chapter 8 - Results generated from the application of CROSS are presented and discussed.

Chapter 9 - Discussion of the overall contribution by CROSS. Suggestions for future research are also given as well as the research contributions of this thesis.

Chapter 2

Review of Driver Scheduling Methods

Over the last thirty years, two classes of approach have been predominantly used to solve the transport driver scheduling problem. These are *heuristic* and *mathematical programming* approaches. In more recent times other *metaheuristic* methods, such as genetic algorithms or ant systems, have been employed to try to tackle this problem. Both the papers [133, 139] give excellent brief overviews of driver scheduling issues and the methods used to solve them over the years.

The earliest successful computer scheduling systems for the driver scheduling problem were developed during the seventies. A series of workshops have been held since, with the purpose of presenting and discussing some of the latest techniques and developments in public transport scheduling with computers [120, 132, 102, 26, 32, 24, 130].

In the sixties and seventies heuristics were, by far, the more popular approach. This was due not so much to their efficiency, but to the available computational power of the day. Mathematical programming algorithms had been developed but it was not feasible to use such methods as the computing capacity available at the time would only have been adequate for very small cases. Watson and later Manington conducted early work into

the suitability of mathematical programming methods for the driver scheduling problem in the seventies [127, 83] but at that time heuristic methods were much more practical.

In most of the successful systems in use today it is a blend of heuristics and mathematical programming methods that are in use.

This chapter will examine some of the past (and present) methods used to solve the driver scheduling problem.

2.1 Heuristic Methods

Heuristics developed for solving transport scheduling problems by computer have existed since the 1950's. The earliest known example of scheduling bus crews by computer was by Cooper, who at the time worked for the City of Oxford Motor Services. This work is referred to in a feasibility study carried out at the University of Leeds [133, 77].

Heuristic systems such as TRACS (which will be described in some detail next) employed various heuristic schedule construction techniques that, essentially, mimic the techniques used by human schedulers. Schedules produced with this method then had refining heuristics applied to them. The refining heuristics usually consisted of routines to exchange work between duties. A review of earlier heuristic methods can be found in [133, 135]. Examples of several heuristic methods follow. It should be noted that all of these heuristic methods have now been abandoned.

2.1.1 TRACS

TRACS (Techniques for Running Automatic Crew Schedules) was a heuristic system created and developed at the University of Leeds during the late sixties and throughout the seventies. Parker and Smith give a description of the heuristics found in TRACS [92]. The approach taken by TRACS was to first create an initial schedule that satisfies all the

labour agreement constraints; then a set of refining heuristics were applied to the schedule generated in order to try to improve its quality (both in terms of number of duties used and cost). A simplified description of the heuristics used in both duty schedule generation and refinement is given next.

TRACS first creates a set of *marked relief opportunities*. This is created from the analysis of both the morning and evening peak periods of the bus schedule. For each morning vehicle the latest relief opportunity on it that can create a legal spell from the vehicle's beginning is marked. An example of a morning portion of a bus schedule's marked relief opportunities (assuming a maximum spell length of four and a half hours) can be found in Figure 2.1. The marked relief opportunities in Figure 2.1 are labelled with an 'X'.

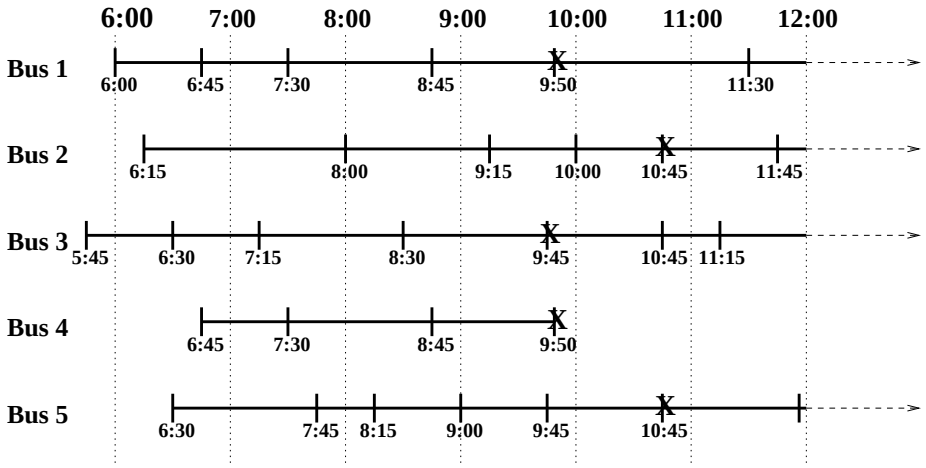

Figure 2.1: TRACS: Marked Relief Times Example

Similarly, in the case of the evening peak, the earliest legal relief opportunities for the start of a spell (marked from the finish of the buses) are marked. The relief opportunities marked in the morning peak are sorted into chronological order. The relief opportunities marked in the evening peak are sorted into reverse chronological order. Some relief opportunities will be marked as *fixed*. This is meant to represent relief opportunities that *must* be used in the schedule. For example, the last relief opportunity on bus four in Figure 2.1 would be a fixed relief opportunity.

The heuristic starts to form duties by examining each marked relief opportunity in chronological order. If the relief opportunity being examined is not fixed then the bus continues

after this relief opportunity and no duty has yet been assigned to work on the bus after this marked time. In this case, the current marked relief opportunity is changed to the earliest possible one on the same bus in order to form a duty whose first stretch of work is on the current bus, up to the given relief opportunity, and whose second stretch of work will take over another bus at, or before, its marked time. Having found a 'good' duty using the earlier marked time the algorithm will progressively move the marked time later until it reaches the original marked time. The best overall duty will be chosen at this point and the relief opportunity used is now *fixed*. If the relief opportunity was already fixed, then only this time is considered. Preference is generally given to taking over a second bus at its marked time. The list of marked times is updated with new marked times due to others being made fixed. Stretches are also left "uncovered" to satisfy targets on the number of split duties that should be generated. A similar method, working backwards, is carried out in the evening working towards the afternoon. Peak work is re-examined after this step, and split duties are created. A second phase is available which deals with any leftover uncovered work. This tries to allocate uncovered work as third portions of duties (if possible) or as single spell duties.

As an example, consider again the schedule represented in Figure 2.1. The earliest marked time is on bus three at 9:45. This marked time is moved to the earlier time of 8:30 (7:15 would be too early). Assume that mealbreaks can be anywhere between 30 and 60 minutes in length. In this case the duty starting on bus three at 5:45 will work until 8:30 and take a mealbreak. This duty will start work again on bus five at 9:00 (giving a 30 minute mealbreak). The marked time on bus three is now fixed at 8:30 and the marked time on bus five is moved to 9:00 and fixed. It should be noted that further marked relief opportunities will be added to buses once a relief opportunity is fixed. For example, bus three, with the now fixed time of 8:30, has a further mark put on it in the early afternoon which will be the latest time a duty joining bus three at 8:30 can be relieved. The earliest marked time is now 9:00 on bus five. This marked time is linked with bus one at 9:50 which happens to be the marked time on bus one. This forms a second duty starting on bus five at 6:30 working until 9:00, taking a 50 minute mealbreak and starting its second

spell at 9:50 on bus one. The time 9:50 on bus one is now fixed and will be considered next. 9:50 on bus one is linked with 10:45 on bus two (since 10:45 is marked) and forms a third duty starting at 6:00 until 9:50 on bus 1 and then works from 10:45 till sometime in the afternoon on bus two. The earliest marked time not yet considered is 9:50 on bus four. The work on bus four will be left uncovered as it starts late enough to form the first half of a split duty. The marked time 10:45 on bus two is considered next. This is linked with 11:45 on bus three which is then marked and fixed forming a duty whose first stretch starts at 6:15 on bus two until 10:45 and then has a second stretch starting at 11:45 that goes into the afternoon. The uncovered work between 8:30 and 11:45 on bus three is then left uncovered to form part of a split or other type of duty. The resulting morning schedule looks like what is found in Figure 2.2.

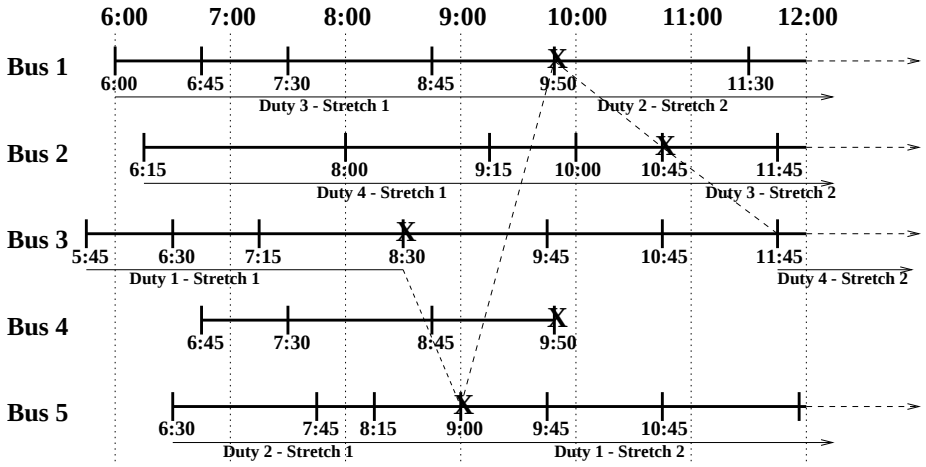

Figure 2.2: TRACS: Morning Schedule Example

Once the initial schedule has been generated using the previously described algorithms a refining stage is applied to try to improve the schedule. These refining steps are broken down into two groups. The first refining stage tries to reduce the number of duties used in the schedule. The second stage concerns itself with reducing the cost of the schedule.

Three routines are used in order to try to reduce the number of duties in the schedule. The first routine, *reduce*, is only invoked if the current schedule has exceeded the target number of duties. Reduce considers every duty in the schedule and an attempt is made to redistribute its work amongst the other duties in the schedule. Another routine, *maxspread*,

tries to decrease the work in duties of low spreadover. This is done by increasing the work in other duties. The gaps created by this process help the *reduce* algorithm outlined previously. The third routine, *only5alter*, considers only the leftover work from the original schedule[1]. This work is, as much as possible, reduced by assigning pieces of it to duties already in the schedule. If this does not reduce all the uncovered work then uncovered stretches are swapped with shorter covered stretches (where legal) and the work in these shorter stretches is examined to see if any of them can be removed (added to another duty). This continues until no further reduction can be achieved.

There are four routines used to try to reduce the cost of a schedule. The first routine, *halves*, creates two lists. Each list corresponds with half a duty from the various types available (early, late and so on). The halves of duties are next matched using a modification of the standard method for solving a transportation problem. The second routine, *recut*, scans the relief opportunities used on each bus. If a reduction in cost can be achieved by moving one of these relief opportunities to an earlier/later time then such a move is made. Another cost reduction routine, *changends*, removes small pieces of work from duties around the middle of the day. An assignment problem is then solved to reallocate these pieces of work in an optimal way. The final routine, *stretchswop*, examines the stretches used in duties, and if removing a stretch from a duty and allocating it to another or, alternatively, swapping stretches between duties lowers the cost further the adjustment is made.

TRACS was demonstrated in the seventies for several UK transit companies with some success. These include Bristol Omnibus Company, Midland Red, Cleveland Transit, West Yorkshire Passenger Transport Executive, Greater Glasgow Passenger Transport Executive and Greater Manchester Passenger Transport Executive. However, Parker and Smith observe that modifying the program to work with different labour agreement rules can be difficult [92].

[1] What TRACS refers to as a type 5 duty, hence the routine name.

2.1.2 RUCUS & RUCUS II

The Run Cutting and Scheduling (RUCUS) system was developed due to sponsorship from the Urban Mass Transportation Administration (UMTA) of the United States Department of Transportation. A *run* is the American term for duty and *run cutting* is the American term for duty scheduling[2]. RUCUS was marketed by the MITRE corporation. RUCUS-I is described in the first transport scheduling conference [120]. The RUCUS-II system being described here is taken from descriptions by Luedtke and Ball, Bodein and Greenberg [79, 4]. RUCUS-II maintained a relational transit system database and provided facilities for generating both vehicle and duty schedules.

The information from the relational transit database was used both for vehicle and duty scheduling. RUCUS-II provided the user with manual aids for vehicle scheduling as well as some automated vehicle scheduling routines as outlined by Luedtke [79]. Duty scheduling was done using heuristics. RUCUS-II used many of the same heuristic procedures used in the original RUCUS system. The heuristics developed in RUCUS and RUCUS-II try to simulate manual techniques for selecting pieces of work and combining them to create duties.

RUCUS's original interface was fairly complex and many parameters needed to be specified and routines needed to be invoked in the schedule generation process. The schedule generation process itself was run in a batch oriented system. RUCUS-II makes some effort to improve interface issues raised by the users of RUCUS. Hildyard and Wallis give a review of experiences using RUCUS and outline some of the improvements they had made [59].

The first step in the RUCUS-II crew scheduling system is to generate an initial duty schedule (this is with virtually no optimization being performed). This is done by specifying time periods and the type of duties to be cut (for example, early duties). This is carried out over all possible time periods by using the three automated duty scheduling heuristic

[2] As observed earlier, a whole plethora of regional transport scheduling terminology exists. Hartley gives a good glossary for such terms [57].

procedures offered by RUCUS-II . RUCUS-II contains a manual element in that at any point the scheduler can change the duty schedule as needed. After this has been finished a module called IMPROVE is invoked to try to improve the schedule generated.

Luedtke provides a description of the heuristic algorithms used in the creation of the initial schedule [79]. As mentioned above, when the various duty scheduling algorithms are invoked a set of parameters are passed that represent constraints such as earliest/latest start/finish time of duties, length of duty, mealbreak length, spreadover and so on. Three heuristic duty scheduling algorithms are outlined:

1. **One-piece duties -** One-piece duties are cut either from the front or the back of running boards. Such work cut from running boards is evaluated against the parameters passed and evaluated by the cost routine to ensure its legality[3].

2. **Two-piece duties using dovetailing technique -** To generate early and late duties a dovetailing technique is applied to the creation of duties from the front or back end of a running board respectively. The vehicle running boards are sorted by earliest pull-out (or pull-in in the case of late duties). This creates a pyramid-like arrangement of running boards. A cutting technique is employed which cascades down the running boards. The first half of a duty is cut and then the other running boards are searched for a suitable second part of the duty. A scanning technique (not elaborated upon in [79]) is used to select several possible relief points to be used in the formation of a two-part duty.

3. **Two-piece duties using forward/reverse splitting -** This method is similar to the dovetailing technique described above. There is no sorting of the blocks of work although each iteration of the duty scheduling process selects the earliest pull-out (or pull-in). Luedtke claims a similar scanning technique, used in the dovetailing technique, is employed in this method to aid in the selection of a subset of relief opportunities used to help evaluate possible mealbreaks for two-part duties.

[3]As an aside, today such duties are generally no longer acceptable.

It is observed that these duty scheduling steps can be applied in varying sequences so that the scheduler can produce different duty schedules. These schedules will have their costs and associated advantages/disadvantages explored and the best one chosen.

Once a duty schedule has been formed, a module called IMPROVE is used to try and improve the duty schedule. RUCUS-II first used the same heuristic based IMPROVE module found in the original RUCUS system. These heuristic methods essentially used a series of shifting and switching steps. The shifting step involved shifting a relief opportunity in a duty to an earlier or later time, potentially reducing the cost of a schedule. The switching step involves switching halves of various two-part duties if this would result in a cheaper schedule. These techniques are similar to the improvement heuristics used in TRACS.

Ball, Bodin and Greenberg discuss an enhancement to the RUCUS-II system that replaces this original version of IMPROVE [4]. Instead of using the heuristic shifting and switching steps, a mathematically based matching approach is used in order to try to get the optimal matching of pieces of work. The drawback to both versions of IMPROVE is that the quality of the final schedule is very dependent on the quality of the original heuristically generated schedule.

It is believed that work on RUCUS-II has been abandoned.

2.1.3 COMPACS

COMPACS was a duty scheduling system that worked interactively with the user in order to produce crew schedules. Wren *et al.* report that it could usually produce schedules with the same number of duties as manually produced ones and occasionally uses an extra duty compared to the manually produced solution [141]. COMPACS was part of the BUSMAN scheduling suite of programs marketed by Wootton Jeffreys [129].

COMPACS was designed to operate in two different ways. One method was to produce an entire duty schedule automatically using heuristics similar to those used in TRACS. The schedule is then created quickly with no interaction from the user. The second

method produced a duty schedule in an interactive fashion. Users could create their own duties to add to the schedule or ask COMPACS to form duties which could then be accepted or rejected as seen fit by the scheduler. At any point the scheduler could have COMPACS finish the partially created schedule automatically. The scheduler also had complete control over the schedule through facilities to edit or delete duties which are already present in the schedule.

Wren *et al.* [141] highlight several inherent advantages interactive systems, such as COMPACS, have over integer programming methods. Flexibility is maintained as to how the schedule is created and modified. Heuristics are generally quick and, thus, cheap to run. The scheduler maintains complete control over the scheduling process and can modify the schedule as seen fit at any point. Finally, such an interactive system had the ability to solve very large problems as it did not have the space, nor computational requirements of mathematical programming methods. With the power of today's modern computers these advantages have become less apparent. Mathematical programming methods will create better schedules than heuristics.

2.1.4 HOT & HOT-II

The HOT (Hamburger Optimierungs Technik) system was originally developed by the Hamburger Hochbahn AG and is now marketed and developed by a former subsidiary, HanseCom. The various components of the system are documented in several papers to be found in the transport scheduling conference proceedings [60, 25, 123]. HOT has been in use in Hamburg, Germany since 1978. Since then it has been used by several other municipal bus operators in Bremen, Cologne, Dresden and Saarbrucken, and Besançon (France).

In addition to generating driver schedules, HOT can schedule buses and create duty rosters. The complete HOT system is composed of five modules to address individual tasks: data management using a relational database management system, sensitivity analysis (for bus scheduling), vehicle scheduling (using a variant of the hungarian algorithm), duty

scheduling and duty rostering. The output of one module is designed to be the input for the next module.

The crew scheduling component of the original HOT system is mathematically specified as an assignment problem (methods to solve such problems can be found in any reasonable operations research text such as [131, 14, 99]) that matches up the pieces of work available in order to form duties. First, one part early duties are created by cutting them out of early vehicles that have relief opportunities late enough to allow this. Leftover pieces of work or uncut vehicles are then split, if necessary, and recombined to form duties. A similar process is carried out for the late duties, except that the blocks are broken backwards in time from the finish of the vehicles to the earliest legal starting relief opportunity. Afternoon duties are created next using the leftover bus work from the late and early duty creation phase. Split duties are created last to cover any uncovered buswork at this point which is generally to be found during the peak periods. An assignment model is used to create the duties in the steps above.

Völker and Schütze document the newer version of HOT, HOT-II [123]. One difference is in the user interface. An effort seems to have been made to allow the user more interactive control in order to modify computer generated solutions manually. There were changes made to the algorithms themselves not documented in [123]. Wren gives a description of some of the algorithms used HOT-II for the system developed for Bremen [134].

HOT-II first splits up the morning periods on buses into pieces of work by selecting relief opportunities (generally not those that are close to the beginning of a bus) from the bus workings. These pieces of work are next sorted into order of starting time. From this point onwards HOT only considers the individual pieces of work in order to form duties. Unassigned pieces of work are processed in chronological order as potential first halves of duties. Every possible duty beginning with the given starting work unit is considered and given a weight. The duty with the lowest weight is chosen. Any work contained in the selected duty is considered assigned to a driver and, thus, is removed from the pool of uncovered work. If work before the second part of a duty can be the beginning of a new

duty it is considered next and the same method described above is applied. The same method described above is also applied to the evening portion of the schedule to produce late duties (forming duties backwards). Uncovered pieces of work are grouped into vehicle workings and they have the relief opportunities on them selected that produce maximum length pieces of work. These long pieces of work are then matched using a variant of the hungarian algorithm. A similar process is used to create split duties.

2.1.5 INTERPLAN

INTERPLAN was an interactive program for mass transit crew scheduling that employs heuristics to find its solutions. INTERPLAN was capable of rostering as well as duty scheduling, although these were carried out in two separate steps. A description of INTERPLAN is given by Mott and Fritsche [89].

For duty scheduling purposes INTERPLAN provided two different computer-aided procedures. One is based on a heuristic and the other is an exact optimization algorithm.

The heuristic procedure is based on the fact that duties in a crew schedule can be classified into different types, each with their own characteristics. These characteristics can be used as parameters in the algorithm to help describe the duties that are to be formed. The parameters consist of such things as the total length of a duty, the total working time, the number of pieces in a duty, the earliest start and/or finish time as well as whether or not the duty is a split duty. These parameters, essentially, define "time-windows" in which different types of duties are generated. It should also be noted that these parameters are not unlike parameters used in the previously described systems.

The user would first define these time-windows described by the different types of duties desired. The heuristic procedure would then check all the possible combinations of the pieces of work and evaluate them. The work checked falls in between the earliest starting and latest finishing times of the type of duty that is being generated. The best duties in each group of potential duties generated would be selected. This would continue until a

given number of duties are generated. It should be noted that using these time-windows also helps to reduce the number of potential duties generated. Once this process has been completed the generated duties would be presented to the scheduler graphically. These duties could then be accepted, deleted or modified as seen fit. Unfortunately this method cannot guarantee any degree of optimality and it is possible that there will be some pieces of work that will be left uncovered.

The second method available in INTERPLAN to tackle the duty generation process is based on a matching algorithm. This procedure only optimizes the pairing of given pieces of work (the algorithm assumes that the running boards have already been partitioned into pieces of work). The quality of the schedule generated in this fashion is very dependent on how the running boards are partitioned into pieces of work.

The two methods can be combined. The heuristic method can be used to generate an initial set of duties. These duties can then be decomposed into the pieces of work they cover and then the matching algorithm can be applied to it.

INTERPLAN has been tried in both Munich and Nuremberg, Germany. Schedules produced by INTERPLAN have been used in practice but Mott and Fritsche do not say how successful they were and there is no further mention of INTERPLAN in later computer-aided transport scheduling conferences.

2.2 Mathematical Programming

Wren and Rousseau observe that most successful driver scheduling packages in use today use a mathematical programming methodology. Many use a model based on set covering but there are others [139].

2.2.1 Set Covering Formulation

The most common mathematical formulation of the driver scheduling problem is as a *set covering problem*. This requires first generating a very large set of potential duties. The selection of a subset of these duties in order to cover the bus schedule at a minimum cost is done by solving the associated set covering problem. This can be expressed as an Integer Linear Programming (ILP) problem, similar to Shepardson's formulation [108], as follows:

$$\text{minimize} \quad \sum_{j=1}^{n} c_j x_j \tag{2.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1, i = 1, 2, \ldots, m \tag{2.2}$$

$$\text{and} \quad x_j \in \{0, 1\}, j = 1, 2, \ldots, n. \tag{2.3}$$

The m constraints are the pieces of work to be covered in the bus schedule. The n variables correspond to the duties in the generated set. The value $a_{ij} = 1$ if the j^{th} duty covers the i^{th} piece of work; c_j is the cost associated with the j^{th} duty. Variable $x_j = 1$ if the j^{th} duty is in the schedule, 0 otherwise.

The objective function (2.1) represents the cost of the schedule and is to be minimized. The inequalities (2.2) state that every piece of work should be covered by *at least* one driver; this problem formulation can, therefore, allow a piece of work to be covered by more than one driver (referred to as *overcover*). If inequality (2.2) is set to a strict equality then this problem is transformed into the *set partitioning problem*. The set partitioning problem is not usually used to model bus or rail driver scheduling. The reason for this is there can be a danger that there will be no solution; that is no subset of shifts will exist that "fit-together" without causing any overcover. Set partitioning, however, is a popular choice for some other crew scheduling problems, for example air crew scheduling.

The set covering problem, in the ILP context given above, is usually solved by first relaxing the constraint that the variables must have integral values. Once a continuous solution has been found a branch-and-bound method is used to transform it to an integral solution.

Examples of driver scheduling systems that use set covering or other mathematical methods will be described next.

2.2.2 IMPACS

IMPACS (Integer Mathematical Programming for Automatic Crew Scheduling) is a driver scheduling software package developed by the Operational Research Unit at the University of Leeds. A more thorough description of the IMPACS system is given by Smith and Wren in [111, 112, 115, 140]. The IMPACS system was installed for London Transport in 1984 and for Greater Manchester Transport in 1985. IMPACS is also part of the BUSMAN system [129, 15].

TRACS-II is the successor system to IMPACS and has been developed since 1994. TRACS-II is described in detail in Chapter 4. Only the basics of the set covering model are described here.

The number of variables in the set covering formulation is equal to the size of the set of potential duties generated. This set can be very large, especially if many relief opportunities are available. To make the set covering problem a practical one, in the sense that it can be solved with the given computer hardware in a reasonable amount of time, often the number of variables and constraints must be reduced. Several heuristic methods have been developed within IMPACS to accomplish this.

Before the large set of potential duties is generated IMPACS inspects the schedule to try to reject some potential relief opportunities. Essentially the maximum spell length is used to mark out the latest time successive spells can finish from the beginning of a vehicle. Similarly, starting from the end of the vehicle, the earliest times a sequence of spells can start are marked. Generally each backward marked time is earlier than the corresponding forward marked time. This forms time-bands and the relief opportunities falling outside these bands are deleted. Further analysis may result in additional relief opportunities being re-instated or deleted. For example a relief opportunity may be re-instated if it is

potentially useful in the sense that it could help in the formation of a particular type of desirable duty.

After the set of potential duties is generated it is examined to see if any duties can potentially be deleted from it if the set is large. Duties that have their work contained within other duties (and are not cheaper) are deleted as well as duties that have their pieces of work covered by a specified number of other duties.

Once a schedule has been produced, swap and switch heuristics (similar to what is found in the TRACS system) are applied to try to improve the solution.

2.2.3 EXPRESS

The EXPRESS system uses a set partitioning model to schedule bus duties as described by Falkner and Ryan [35]. The set partitioning model is solved using the ZIP package which is also used in IMPACS [106, 38].

The EXPRESS system was designed specifically to be used in Christchurch, New Zealand. The overall schedule to be produced is divided into five subproblems. The first of these is a set of contract bus services; this problem is solved manually. Each of the remaining subproblems consists of a subset of the routes. Each subproblem is solved individually and broken down into three stages.

Stage 1 - The first stage involves creating the middle and late duties for the schedule. The early portion of the schedule is removed. With the remaining part of the schedule left intact thousands of potential duties are generated and then the problem is solved using the set partitioning formulation. The mathematical model used for this stage includes extra *mealbreak* constraints to help minimize the number of mealbreaks that are used over certain times.

Stage 2 - Most of the early duties are selected in Stage two. Duties generated in Stage one and any work that is too late to appear in early duties are excluded from the

schedule. The pieces of work that are to be covered by early duties are identified and a reduced set of potential duties is generated and some of these duties are selected using a mathematical model. Pieces of work that are covered by the early duties are then removed from the schedule.

Stage 3 - The final stage creates the remaining early duties and any other duties needed to cover the remaining uncovered work. The halves that will be later matched together to form split duties are also created at this stage.

One disadvantage to this approach is that there is a loss of interaction between the three stages. The EXPRESS system was first used in Christchurch New Zealand in a major re-scheduling project that was completed in July 1990. The system itself is menu-driven and the scheduler often has to make manual adjustments to the final schedule to reduce the cost of drivers and to take into account some soft constraints that are difficult to model.

2.2.4 HASTUS

The original work on HASTUS began in 1974 as a joint research project between the Transportation Research Center of the University of Montréal and the Montréal Urban Community Transit Corporation. The company GIRO was created to deal with the commercial distribution and development of the work. HASTUS is designed to be a complete scheduling package.

HASTUS (as reported in [78, 104, 105, 10, 54]) is an interactive transport scheduling system that is composed of three parts. They are HASTUS-Macro, HASTUS-Micro and HASTUS-Bus. A more recent software module for the HASTUS suite is the CREW-OPT procedure. This will be discussed in 2.2.5.

The HASTUS-Bus component is responsible for creating vehicle schedules given the service requirements. Once the bus schedule has been generated HASTUS-Macro and HASTUS-Micro components can be invoked to create the driver schedule.

HASTUS-Macro is the first phase in the driver schedule creation process. It uses a linear program that tries to match idealized duties with buswork. HASTUS-Macro partitions the vehicles into pieces of work with relief opportunities supplied at pre-determined intervals. A variable set representing idealized possible duties is then generated and a minimum cost solution is found using mathematical programming methods. This solution gives an indication of the types of duties that should be constructed in order to form the final solution.

HASTUS-Macro is often used as a approximation tool useful in evaluating proposed changes to labour agreements as well as examining any "what-if" situations that may arise. HASTUS-Macro is, thus, useful as a tool to measure the economic impact of proposed labour agreement changes as well as a planning tool that can evaluate the impact of the changes of the level of service. Garnier reports that the Paris bus company RATP[4] has used HASTUS-Macro as a tool for this reason [46]. The solution found in HASTUS-Macro can also be used as input into HASTUS-Micro in order to transform this schedule "approximation" into a proper duty roster as described by Lessard, Rousseau and Dupuis [78, 105].

HASTUS-Micro completes the duty generation process. HASTUS-Micro uses the HASTUS-Macro solution to split the vehicles into pieces of work. These pieces of work correspond, as closely as possible, to the work generated in the HASTUS-Macro solution. The vehicles are cut into pieces of work by solving a shortest path problem. This is expressed as finding a path from the starting time to the finishing time in a vehicle using arcs that correspond to pieces of work. A matching algorithm is then used to create duties from the work [10]. A refining heuristic method can be applied afterwards in order to improve the solution slightly and to reduce the number of one piece duties used.

HASTUS has been used or is currently being used in cities throughout the world, including Montréal, Boston, New York, Calgary, Helsinki, Singapore, Stockholm, Barcelona, Nantes, Edinburgh, Sheffield, and Newcastle upon Tyne.

[4]Régie Autonome des Transports Parisiens.

2.2.5 CREW-OPT

GIRO Inc has added CREW-OPT into its HASTUS family of scheduling software. CREW-OPT is a scheduling method that employs a set covering model using a column generation approach [31, 103].

The column generation method is essentially an extension of the set-covering approach to solving the crew-scheduling problem. The initial approach is similar to that used in the set covering problem. A set of "good" duties are generated and the relaxed LP is then solved using the simplex algorithm. Once a continuous solution has been found a subproblem is then formulated that is used to try to generate new variables (duties) that would reduce the cost of the current solution.

The subproblem formulation used in CREW-OPT is modeled as a shortest path problem with resource constraints. It is solved using a dynamic programming algorithm[5]. Desrochers and Soumis [31] give a thorough description of the subproblem formulation. The resource constraints are imposed on the network to ensure that only legal duties are created. These constraints consist of: maximum duration of a spell, minimum and maximum break lengths, maximum spreadover of a duty and the maximum number of pieces allowed in a duty.

The subproblem itself is modeled using a network. Arcs in the network are defined to represent pieces of work, breaks in between pieces of work (mealbreak or joinup), and signing on and signing off activities. By solving this network as a shortest path problem, efficient duties are created and compared with the current continuous solution of the scheduling problem. If any of the duties created using this algorithm have a negative reduced cost[6] then they are included in the LP formulation (a new column (variable) representing this duty is added to the LP). When it becomes impossible to find a duty with a negative reduced cost then the LP solution is optimal.

[5] A dynamic programming approach solves problems by combining the solutions to subproblems. A description of this technique can be found in [20].

[6] That is, they can improve the solution.

Rousseau [103] gives a description of some of the scheduling problems CREW-OPT has been applied to. CREW-OPT has been used in Lyon, Toulouse, Barcelona, Vienna, and East Japan Railways.

2.3 Metaheuristics

There are other approaches that can be employed in order to solve the driver scheduling problem. Many of these fall into the category of *metaheuristics*. The University of Leeds has examined metaheuristic approaches to duty schedule generation as reflected in the transport scheduling conferences. These will be examined next.

2.3.1 Genetic Algorithms

The approach of *genetic algorithms* (GA) involves intelligently exploiting a random search by using an analogy to a natural process. GAs are based on a natural analogy with population genetics and evolution. In nature, usually the stronger creatures in a population will tend to survive whilst the weaker ones perish. This also works for some of their inherited traits. This concept is extended to operational research problems by applying the same principle to populations of potential solutions to a problem.

The interest in this approach began in the seventies when John Holland first published his book *Adaptation in Natural and Artificial Systems* [61]. The basic idea of a GA is to create an initial *population* of *solutions* or *chromosomes* for a given problem. Examples of scheduling problems which have been solved using GAs include university course timetabling problems, professional ice hockey league timetabling problems, examination scheduling problems and vehicle scheduling problems as well as bus driver scheduling [75, 21, 36, 68, 18]. Two standard operators are then applied to this population to create new and, it is hoped, better solutions. *Crossover* is employed to mate two solutions in the hope that good characteristics from both of them will be combined in the offspring.

Mutation is a unary operator that, with a low probability, randomly alters the solutions in such a way that genetic diversity is maintained. The parents used to create offspring are often probabilistically chosen in such a way that the stronger solutions are more likely to participate in reproduction. Similarly the less desirable solutions are more likely to be replaced in the population with the offspring from these parents.

Over time the population will tend to *converge*. That is, the solutions will begin to resemble one another so much that genetic diversity is lost. Mutation often helps to delay convergence somewhat but it is inevitable that convergence will occur given the population sizes used in practical problems. Convergence should be avoided in the early stages of the GA process. Early convergence can lead to a sub-optimal or even a poor solution. Further information on the basics of GAs can be found in any of [85, 100, 28, 6, 7].

Two examples of GAs applied to the driver scheduling problem follow.

2.3.1.1 Greedy Genetic Algorithms and Optimizing Mutations

At the University of Leeds Wren and Wren began work with the application of genetic algorithms with reasonably good results [142]. Clement and Wren continued this work with GAs with somewhat promising results which will be described next [18].

The initial population of solutions (chromosomes) are composed of binary strings of which each bit represents a potential duty in the solution. A bit is set to '1' if that duty is present in the solution, '0' otherwise. Crossover, in the context of this GA, is somewhat unusual. When two chromosomes are mated, a union of all the duties used in both chromosomes is constructed. This is referred to as a *fertilized cover*. Child chromosome(s) can then be created using some form of heuristic that generates a legal schedule from the fertilized cover. This can be seen conceptually in Figure 2.3. Several methods of crossover and mutation are reported giving varying results.

The first method is referred to as the *raw genetic* approach. The crossover can be thought of as a variation of the uniform crossover operator [119]. Child chromosomes are created

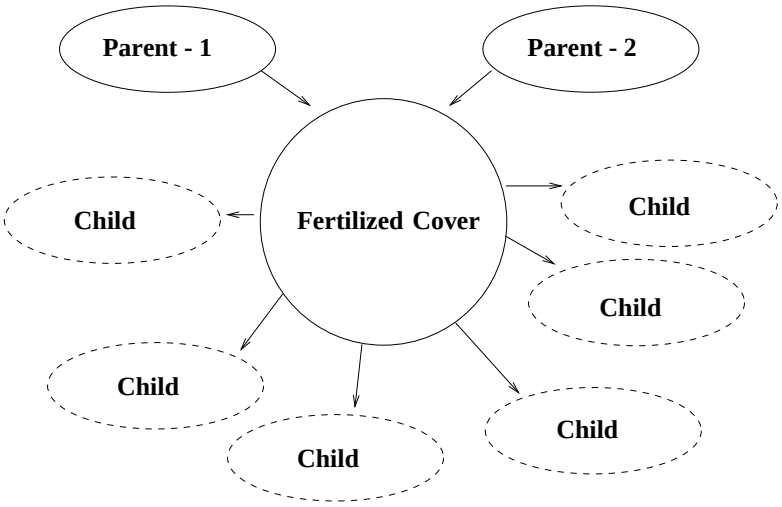

Figure 2.3: Reproduction using Fertilized Cover

from the random selection of bits (potential duties) from the fertilized cover. This takes into account constraints such as ensuring all work is covered by at least one driver. The results from this approach were not entirely satisfactory. One reason may be that the population converges very quickly. This is avoided by introducing population remakes; this entails entering many new random solutions into the population when convergence is detected (while keeping the best solution). When population remakes were introduced the quality of the solutions improved.

A new crossover method is introduced called *greedy crossover*. Given the fertilized cover constructed from the two parents a heuristic derived from greedy algorithm methodology is then used to create a new solution (child). Duties are linked with the pieces of work they cover. Given a list of uncovered work this heuristic examines each piece of work in the order presented and then selects the duty covering the most work, including the piece of work being considered, from the fertilized cover. This is repeated until the entire set of work is covered. Unfortunately this approach only generates one possible child per mating pair of solutions. This problem is solved by permuting the order in which work is considered in the greedy heuristic, which seems to help avoid convergence.

Clement and Wren report that classical bitwise mutation was tried and found to be of little benefit. A mutation operator referred to as an *optimizing mutation* was then developed

which searched the chromosome's duty set in order to try and replace some duties with more efficient ones. Although this procedure is computationally expensive it did seem to improve results.

Unfortunately, none of the results were able to reproduce known optimal solutions to the problems (although, it should be noted, Wren and Wren's original work on this did [142]). The best solutions, using greedy crossover with optimizing mutations, usually had two or three duties more than the known optimal. This is not to say that GAs are not applicable to driver scheduling problems but rather that they need more research. One possible application would be to apply them to large problems where solutions are difficult to find using more conventional mathematical programming techniques.

2.3.1.2 Genetic Algorithms with Embedded Combinatorial Traits

The approach taken by Kwan *et al.* [72] is substantially different from the previous GA examined. It is based on the TRACS-II system developed at the University of Leeds (as will be described in Chapter 4), which uses a set covering model similar to the one described in Section 2.2.1.

TRACS-II initially generates a large set of potential duties that abide by all the constraints of the labour agreement for the schedule in question. The problem is then formulated as a set covering problem and solved with an ILP with the integral constraint on the variables relaxed. Normally at this point the ILP would use a Branch & Bound (B&B) process if the solution is not integer. The GA takes over the schedule generation at this stage instead.

The continuous solution provides two pieces of useful information. The sum of all variables in the continuous solution[7] provides a useful lower bound on the number of duties needed in the solution. The second point is that the continuous solution involves a number of duties that is no more than the total number of work pieces. Through experimentation it

[7]Rounded up if non integral.

has been found that at usually about 75% of the duties in the final integer solution are in the continuous solution. This implies that there are probably subsets of shifts that fit well together (or "well-fitting" shifts) in the continuous solution (each such subset is referred to as a *combinatorial trait*). The goal of the GA is to derive such subsets, supplemented with duties selected from the original large set of potential duties.

The GA chromosome is of a fixed length that is dependent on the number of duties in the continuous solution whose continuous value exceeds[8] p. The chromosome is composed of a bitstring with each bit representing a duty that has been selected from the continuous solution. A value of '1' indicates its presence in the schedule whilst a value of '0' indicates its absence. The bits of the chromosomes are generated randomly. It is unlikely that any of the chromosomes will represent a legal schedule. A *repair heuristic* is applied to the chromosomes to remedy this. This involves potentially considering duties other than those represented in the chromosome (those duties not in the continuous solution with a value $> p$), using a greedy approach. After the repair procedure any redundant duties are removed.

The fitness of a chromosome is represented in terms of the weighted cost of the schedule as well as a heavy penalty for each duty used. If the schedule S to be evaluated has n duties and C_j represents the cost of shift j the cost can be expressed as follows:

$$\text{WeightedCost}(S) = \sum_{j=1}^{n} C_j + 5000n \tag{2.4}$$

The high cost imposed for each duty helps to identify better solutions (in terms of number of duties) from their cost value. The genetic operators employed are classic one-point and uniform crossover as well as standard and aggressive mutation (as described in [72]). These unary operators become more active over the lifetime of the algorithm as they help to enlarge the search space in order to avoid convergence. The parent selection procedure for crossover is probabilistic with the probability being proportional to the parent's fitness. Population sizes used in the GA have varied between 50 and 800.

[8] A good value of p has been found to be 0.2.

The results obtained with this GA have been quite good. The GA was able to match TRACS-II results for several problems whilst coming within one or two duties of the optimum for others. The GA has also solved problems which were too big for TRACS-II. For TRACS-II the problems were decomposed. TRACS-II produced better solutions than manually constructed ones, but the GA did better than TRACS-II. This work is still at an early stage and one of the goals is to identify combinatorial traits within some of the more fit members in the population of solutions in order to exploit them. Four such combinatorial traits were identified, these are: well-fitted duties, relief chains, handover relief and overcover links. Well-fitted duties are defined as duties that simply fit well together. Relief chains, otherwise known as mealbreak chains, are another useful schedule property which will be discussed in detail in Chapter 5. Handover relief traits refers to a break where a driver takes a mealbreak and a new driver takes over. Overcover links are pairs of relief opportunities that are potentially critical to a good solution but the GA overcovers the work represented between them.

The end goal of this research is twofold. First is to identify the combinatorial traits in promising solutions and exploit them during the GA process to find better solutions. The second goal is to use this GA as a replacement for the branch and bound phase of the ILP.

2.3.2 Ant Systems

Ant Systems (AS), as created by Dorigo *et al.* [34], are, like genetic algorithms, based on a natural process. Rather than simulating evolution, ASs simulate activities that take place in an ant colony, specifically their behaviour when searching their environment for food.

In an ant colony it is the job of worker ants to provide for their nest by searching the surrounding area to find, and return with, food. As the worker ants wander about looking for food they leave trails of *pheromones* behind which allows communication with other ants. These trails of pheromones evaporate over time; however, if many ants use the same trail, the scent of pheromones can be quite strong. When an ant is searching for a path

to take, the strength of pheromones on various trails in the area influences its decision on which trail to follow. The stronger the scent of pheromones the more likely that particular path will be chosen. If there is a short path to food it is likely the ants will traverse this path more frequently, thus, the pheromones will evaporate less and the scent will be stronger; this path with the strong pheromones is likely to influence other ants to follow this trail to the nearby food.

An analogy with this natural process is made with the Ant Systems metaheuristic technique. Virtual ants are 'let loose' in a virtual solution space environment and, given a set of rules, it is hoped they will be able to 'trace' reasonable solutions. The strength of the pheromone trail is directly related to the quality of a solution (similar to the GA analogy of a fitness function for a chromosome). Over a period of time it is hoped that a good solution will be found.

2.3.2.1 An Ant System for Bus Driver Scheduling

Forsyth and Wren [42] have applied the AS metaheuristic to the bus driver scheduling problem. The basic idea is to have these virtual ants trace paths through a bus schedule (with the paths representing driver duties) to create driver schedules. Good duties will be used more often by the ants and, thus, have a higher concentration of pheromone trails and are therefore more likely to be chosen in future iterations.

Initially a large set of potential duties is generated. The routine to do this is the same one that is used in TRACS-II as described in Chapter 4. A heuristic devised by Willers [128] is next invoked to produce an initial solution. This covers the work in a greedy fashion by repeatedly selecting duties from the large set generated such that as many pieces of uncovered work as possible remaining in the vehicle schedule are covered. This generally leads to a moderate amount of overcover (and more duties than required) in the initial schedule.

The duties used in this initial schedule are used to calculate the initial *closeness* values.

Closeness values are used to aid the ants when there is little or no pheromone activity in the AS. The purpose of this is, early on in the AS process, to favour duties that are similar to the ones selected in the heuristic solution generated earlier. As time goes on the pheromone trails produced by the ants in the system will take over and their influence will be more dominant than these initial closeness values.

When pheromone trails are updated the values used reflect how much work is uncovered. Duties selected that cover more uncovered work are given larger pheromone values. Once all the work is covered the bias goes towards trying to reduce the number of duties used in the solution.

Unfortunately, there are many parameters to be decided in an AS and Forsyth and Wren recognize that this is a current problem. The current AS being developed is still in the testing and development phase.

2.4 Discussion

Three types of approach to the driver scheduling problem have been examined in some detail. The first of these examined was the use of heuristics to generate driver schedules.

The main reason heuristic systems were developed to produce driver schedules had more to do with the technology of the time rather than lack of understanding of other approaches (specifically mathematical ones). Heuristic approaches, however, do have advantages of their own. One advantage is that heuristics are quick and cheap to run. The computational power needed to support such a system is also an advantage although with todays inexpensive desktop computers this is much less of a factor. Interactive systems that employ heuristics enable the scheduler to have complete control over the scheduling process as the ability to manually modify a solution at almost any point in the solution generation process can be a powerful tool (such as COMPACS).

Disadvantages inherent with a heuristic approach are also present. Many heuristic methods

(such as TRACS, RUCUS, etc) first form a solution to the driver scheduling problem and then try to heuristically improve the solution. The quality of the final duty schedule produced is often very dependent on the quality of the initial solution. Another issue that arose in the development of heuristic systems (as observed by Parker and Smith with the TRACS system [92]) is that modifying them to be used with different labour rules is difficult.

Mathematical methods are, by far, the most common and successful method of producing driver schedules to date. Systems such as TRACS-II, HASTUS and CREW-OPT are in use to this date which demonstrates how effective and popular such methods can be. Most mathematical systems model the problem using a set covering formulation. First a large set of potential duties are generated and then the duties to be used in the final solution are selected by solving the resulting set covering ILP. One important inherent advantage to a technique such as this is that the solution *method* and the *duty generation* procedures are in two distinct steps. One of the most difficult features in driver scheduling is adapting to the different labour agreements and rules/regulations present in each scheduling situation. These constraints can be taken into account during the duty generation process such that only legal duties are generated. The actual solution method can be thought of as a *stand alone* procedure in the scheduling process. The set covering ILP does not have to reflect these driver scheduling constraints; these are taken care of when the set of potential duties is constructed. With heuristic methods, changes to the labour agreement rules often result in major modifications that have to be made to the scheduling code in order to ensure legal duty schedules are produced. With mathematical methods, such as set covering, only the code that generates the initial large set of potential duties must be updated to reflect any such changes.

Disadvantages that can be found when mathematical programming methods are employed become apparent when the problem size is quite large. If a large number of running boards with many relief opportunities is to have a driver schedule produced for them using mathematical programming it can be the case that the potential set of duties generated is *extremely* large. Heuristic procedures, or overly restrictive labour agreement parameters,

must be used in order to trim the size of the set of duties down to something a mathematical program can cope with in a reasonable amount of time. This implies that valid duties will be excluded from this set of potential duties and this can have a negative effect on the quality of the overall solution.

Metaheuristic approaches such as genetic algorithms or ant systems are still fairly new and unproven approaches to solving the driver scheduling problem. The research done thus far with these techniques has, however, borne some promising results. The approach taken by Kwan *et al.* [72, 70] shows how metaheuristics can be combined with a mathematical programming approach and that has had especially promising results showing that, perhaps, for very large and difficult problems some hybrid between the realms of metaheuristics and mathematical programming may be where future research should be done.

Chapter 3

Constraint Programming

3.1 Introduction

This chapter will explore constraint programming (CP) in some detail. A brief survey of the literature will be presented and towards the end of the chapter examples will be given where constraint programming has been used for to help solve driver scheduling problems.

3.2 Constraint Satisfaction Problems (CSP)

Constraint Satisfaction Problems (CSPs) have been a topic of research for many years. CSPs have practical significance in operational research problems including scheduling, timetabling and other combinatorial problems (see the papers by Davis for the description of a exam timetabling system and Le Pape for a historical discussion of some CP systems that have been applied to scheduling problems [27, 76]).

Unfortunately, not many good texts or references exist that give an overview of the CP field. The texts by Tsang [121] and Marriot and Stuckey [84] are widely regarded as excellent references for those interested in CSPs. Baptiste *et al.* have produced a useful book on constraint programming applied to scheduling problems [5].

This section is split into several parts. The first part will explore what makes a CSP. After CPSs have been outlined the properties of variables and constraints will be examined; this includes properties between constrained variables, such as arc consistency and path consistency. Search algorithms will then be described which will include such topics as variable and value ordering. Lastly a description of the ILOG-Solver constraint programming library will be given.

3.2.1 CSP Definition

A Constraint Programming (CP) or Constraint Satisfaction Problem (CSP) problem consists of three parts [121, 113]. They are:

1. A set of *variables* $X = \{x_1, \ldots, x_n\}$,
2. For each variable x_i a finite set D_{x_i} of possible values (its *domain*).
3. A set of *constraints* restricting the values that the variables are allowed to take simultaneously.

A solution to a CSP consists of an assignment to every variable from its domain in such a way that every constraint has been satisfied. Depending on how the problem is defined it may be the case that:

- Any solution may be sufficient.
- All possible solutions to a problem may be required.
- An optimal (or near-optimal) solution may be required. To do this some objective function must be defined in terms of some subset of the variables.

In general, solutions are found to a CSP by systematically assigning to variables values from their domains. Heuristics often guide this search.

A *label* is a variable-value pair representing the assigned value to a variable. For example, if x is assigned the value v, then $< x, v >$ represents assigning the value v to x. Generally this can be written as $x = v$. A *compound label* is the simultaneous assignment of values to a set of variables [121].

A variable that has been assigned a value or labelled is said to be *bound* or *instantiated*. A variable that has not yet been assigned a value is said to be *unbound*.

3.2.2 Constraints

A constraint is usually represented as an expression involving a subset of the variables in the CSP. For example, the following expressions represent constraints on a subset of the integer variables x, y and z:

$$x = y \tag{3.1}$$

$$z \neq 2 \tag{3.2}$$

$$xz \leq 2y \tag{3.3}$$

Alternatively, and more formally, a constraint on a set of variables can be defined as a set of compound labels for those variables [121]. The compound labels define the only assignments to the variables which can be made simultaneously. Often the notation C_S is used to represent a constraint on the set of variables S. Thus we can refer to constraint (3.1) as C_{xy}.

As an example of this definition of a constraint, consider the integer variables x and y. If their domains are:

$$D_x = \{1, 2, 3\}$$

$$D_y = \{1, 3\}$$

then any subset of:

$$\{(< x, 1 >, < y, 1 >), (< x, 1 >, < y, 3 >), (< x, 2 >, < y, 1 >),$$
$$(< x, 2 >, < y, 3 >), (< x, 3 >, < y, 1 >), (< x, 3 >, < y, 3 >)\}$$

would be a legal constraint between x and y. The constraint $x > y$ would be represented by the subset

$$\{(< x, 2 >, < y, 1 >), (< x, 3 >, < y, 1 >)\}.$$

From the definition of a constraint it is clear that a constraint can affect any number of variables from 1 to n (where n is the number of variables in the CSP). The number of variables affected by a constraint is known as the *arity* of the constraint.

Unary constraints only affect one variable. Unary constraints can be used to remove values from a variable's domain (for example constraint (3.2)).

Binary constraints affect two variables (for example constraint (3.1)). When all the constraints in a CSP are binary both the variables and constraints can be represented in a graph. This *constraint graph*, $G = (E, V)$, represents the variables in the CSP as the vertices, V, and the edge set, E, consists of edges joining two vertices if and only if there is a constraint between them.

CSPs not limited to binary and unary constraints are referred to as *general CSPs*. Any constraint of an arity greater than two can be expressed in terms of binary constraints however it may not always be a good idea to do this in practice. Unfortunately, it is not always easy to decide whether or not it is worthwhile transforming a general CSP to a binary CSP. Bacchus and van Beek explore transforming general CSPs into binary CSPs and there are issues to do with the performance of solving such binary CSPs as well as cost overheads involved in transforming them [3]. Stergiou and Walsh also explore this topic in some detail [116]. Both papers conclude that more research needs to be carried out in this area to help determine when transforming a general CSP to a binary CSP might be

advantageous.

Similar to the graph representation of a binary CSP, a general CSP can be represented by a *hypergraph*. A hypergraph is similar to a graph except instead of an edge set a hypergraph has a *hyperedge set*, H. A hyperedge would join a set of vertices if and only if there is a constraint between them [20, 13].

3.2.3 Node Consistency

For any unary constraint a variable is said to be *node consistent* if and only if each value in its domain satisfies any unary constraints placed on it.

Achieving node consistency is a trivial operation. One simply has to ensure that for each variable in the CSP each of its domain values are examined. Any values that do not satisfy the unary constraint(s) on the variable in question are deleted from the domain.

3.2.4 Arc Consistency

For any binary constraint, C_{ij}, between two variables, say x_i and x_j, the arc (x_i, x_j) is said to be *arc consistent* if for every value $a \in D_{x_i}$, there is a value $b \in D_{x_j}$ such that the assignments $x_i = a$ and $x_j = b$ satisfy the constraint C_{ij}. Any value $a \in D_{x_i}$ that violates C_{ij} because no such $b \in D_{x_j}$ exists can be removed from D_{x_i} since it cannot possibly participate in a solution. If every arc in a binary CSP is arc consistent, then the whole problem can be said to be arc consistent.

The arc (x_i, x_j) is *directional*. If arc (x_i, x_j) is arc consistent it is not necessarily the case that (x_j, x_i) is also arc consistent.

Figure 3.1 shows a constraint graph between the variables x and y. The sole constraint between the two variables is:

$$5 - x \quad > \quad y \tag{3.4}$$

Part (a) of Figure 3.1 shows the original domains of both x and y. Part (b) shows the

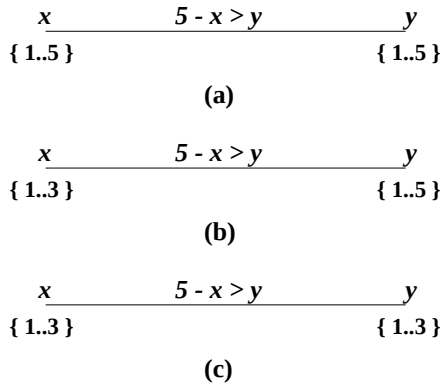

Figure 3.1: Making (x, y) and (y, x) Arc Consistent

new domain of x as (x, y) is made arc consistent. Part (c) shows the new domain of y as (y, x) is also made arc consistent.

3.2.4.1 Achieving Arc Consistency

The number of possible assignments of values to variables which an algorithm solving a CSP might have to consider is the product of the domain sizes of the variables in the problem. Arc consistency can reduce the domain sizes of the variables in a CSP and, thus, help to reduce the number of labels that must be considered in the search for a solution.

Consider another CSP example which will be represented as a search tree showing the search space of the CSP which will be made arc consistent[1]. The search space shows every possible state at which a search could arrive. Again, we will use the integer variables x, y and z with initial domains of:

$$D_x = \{1, \ldots 4\} \tag{3.5}$$

$$D_y = \{1, \ldots 4\} \tag{3.6}$$

$$D_z = \{1, \ldots 4\} \tag{3.7}$$

[1]Each node in a tree can be thought of as a variable and the branches emanating from a node can be though of as values in the variable's domain to which that variable can be potentially bound.

Assume, also, that the following binary constraints are present in this CSP:

$$x - y = 1 \tag{3.8}$$

$$z > x \tag{3.9}$$

$$z - y = 2 \tag{3.10}$$

The original search tree for this CSP, before the CSP is made arc consistent, will look like what is found in Figure 3.2. Note that this search tree assumes the variables x, y and z are considered in order. After this CSP has been made arc consistent it will look like the

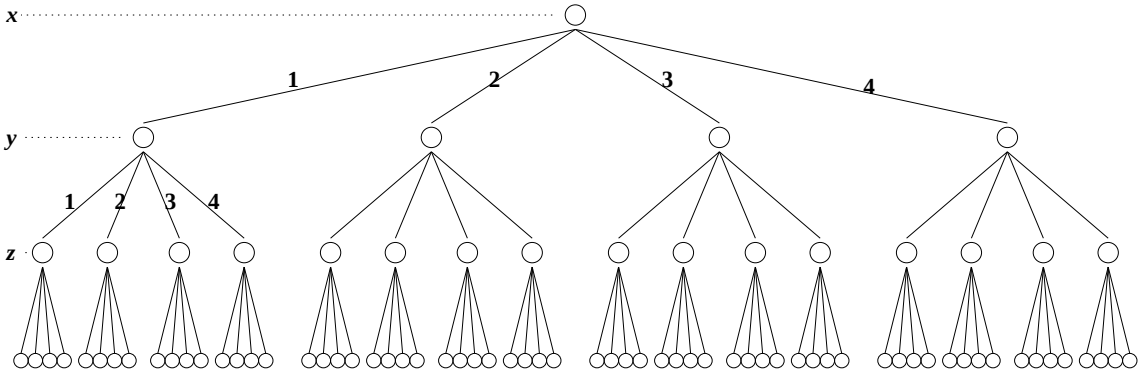

Figure 3.2: Search Tree Example

search tree in Figure 3.3. As can be seen making the problem arc consistent has pruned the search space quite dramatically.

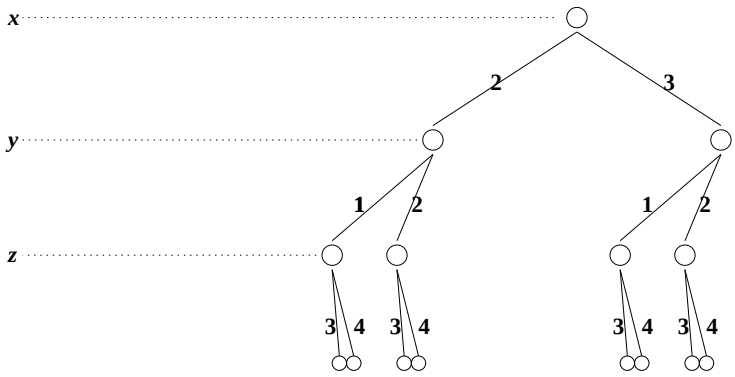

Figure 3.3: x, y and z Made Arc Consistent

Making a CSP problem arc consistent is a relatively cheap operation. Because of this it is often done in the pre-processing stage of a CSP. In ILOG Solver, for example, this is

done as constraints are placed on the CSP (or *posted* as ILOG Solver calls the operation) and also whenever any change is made to the domains of the variables.

Often when a constraint is made arc consistent, it can have a knock-on effect with other constraints. For example if the constraint $x > y$ is made arc consistent and there is another constraint, say $y < z$, then this too must be re-examined to ensure it is arc consistent if the domain of y was modified (if the problem is to stay arc consistent). When a constraint has this knock on effect it is due to the fact that the changes to the variable domains are propagated to other variables via the constraints (also known as *constraint propagation*) . Constraint propagation through arc consistency is powerful as it helps reduce the size of variables' domains throughout the search for a solution.

Several algorithms have been developed to make a CSP arc consistent. Each one is supposedly better than its predecessors in some way (generally with regard to time and space complexity, or perhaps in the way they take advantage of known features of the CSP it is applied to). AC-1 through AC-3 are described by Mackworth[2] [82]. The algorithm AC-3, the best of the three, first creates a queue containing all of the binary constraints (or arcs as it is often convenient to view binary CSPs as graphs) in the problem. An arc is taken from this queue and made arc consistent. If a domain was changed during this consistency procedure then all arcs that might be affected by this (and are not already in the queue) are added to the queue. This process continues until such time as the queue becomes empty. The CSP is then arc consistent. AC-3 has a time[3] complexity of $O(a^3e)$ and a space complexity of $O(e + na)$ where n is the number of variables and e is the number of binary constraints and a is the size of the largest domain.

Mohr and Henderson devised an arc consistency algorithm, AC-4, with the theoretically optimal time complexity of $O(a^2e)$. The price paid is with the space complexity which is $O(a^2e)$ [87]. AC-4 went further than AC-3 in recognizing that values in a CSP *support* other values. For example, if the arc (i, j) is arc consistent with respect to $< i, v >$ and

[2]Actually, Waltz's filtering algorithm [125] is generally credited as being one of the first arc consistency algorithms and Mackworth [82] states that AC-2 follows its spirit the closest.

[3]A detailed analysis of these algorithms can be found in Mackworth and Freuder [80, 81].

$< j, w >$ then the value $w \in D_j$ is said to support $< i, v >$. The extra space required by AC-4 reflects this in that the supporting values are stored and a counter that keeps track of how many supports a value has is also created. This counter is decremented as supports are removed due to constraint propagation. As reflected in the time complexity this reduces the number of consistency checks required. One downside with AC-4 is that, although its optimal time complexity is proven, it usually performs close to its worst case time complexity. AC-3, on average, has a better time complexity. Thus, AC-3 is often used in preference to AC-4 [8, 124].

Further work on arc consistency by van Hentenryck, Devill and Teng has produced a generic arc consistency algorithm, AC-5 [33]. It is generic because it allows consistency checks to be done depending on the type of constraint being checked. This allows the implementation to take advantage of features found in different constraints in order to implement the most efficient consistency check. This is made possible by the queuing strategy used in AC-5. Like AC-3, AC-5 starts by first examining each arc in the CSP and making it arc consistent. Whenever a domain is reduced all arcs that may be affected by this domain reduction are added to a queue. AC-5 also includes the values that have been removed from the variable in question, say Δ, along with the arcs in the queue. When this queue is processed after all of the arcs have been examined the user implemented arc consistency routine receives an element from the queue, say (i, j, v). This represents the arc to be considered, (i, j), as well as the value, v, removed from variable j. The consistency check can then proceed as seen fit. As with AC-3 and AC-4 any further domain reductions result in more elements being added to the queue to be processed. When the queue is empty the CSP is arc consistent.

These user implemented consistency checking routines can also take advantage of how constraints are internally represented. For example the custom arc consistency procedures may perform operations on the domains of variables such as retrieving the minimum and maximum value in the domain. Functionality such as this is more sophisticated than what is required with other arc consistency algorithms and can have a profound impact on the overall time complexity of AC-5 with regards to certain type of constraints. Devill,

van Hentenryck and Teng discuss these special types of constraints and show the time complexity of AC-5 can be reduced to $O(ed)$ in some cases [33].

ILOG-Solver [97] takes advantage of exactly this type of functionality as do other systems such as CHIP[4] (for which AC-5 was designed). More will be said on this topic when ILOG Solver is discussed in Section 3.2.8.

AC-6, as described by Bessiere and Cordier, is another arc consistency algorithm that keeps the optimal worst-case time complexity of AC-4 but has an $O(ea)$ space complexity [8]. AC-6, like AC-4, does keep track of supports for variables. Unlike AC-4, however, only one support per variable is stored rather than all of them. If that support is removed during the execution of the algorithm then another support is searched for and used (if present). Bessiere and Cordier go on to describe some experimental results that show that AC-6 outperforms, in the sense of comparisons made, both AC-3 and AC-4 on their test problems.

Bessiere, Freuder and Regin [9] published AC-7 which tries to reduce the number of constraint checks necessary by *inferring* support for values in a variable's domain. For example, they state that AC-7 takes advantage of the knowledge that support is *bidirectional*. That is if $< i, v >$ supports $< j, w >$ then $< j, w >$ supports $< i, v >$. Bessiere *et al.* further state that, in terms of constraint checks, AC-7 is an improvement over AC-3, AC-4 and AC-6. AC-7 has a time complexity of $O(ed^2)$ and a space complexity of $O(ae)$ (like AC-6).

The AC algorithms discussed so far only consider binary constraints. Generalized arc consistency considers constraints with an arity of more than two. A CSP is *generalized arc consistent* if for any variable in a constraint and value it is assigned there exists legal values for all the other variables in the constraint. Mohr and Massini [88] describe GAC-4, a generalized arc consistency algorithm which is derived from AC-4. The algorithm maintains a queue of labels that have been removed from a domain and are not yet processed. For each label processed, say $< i, a >$, each constraint containing a compound label that

[4]Constraint Handling In Prolog.

contains $< i, a >$ must have that compound label removed. When such a compound label is removed from a constraint, say C, it may be the case that this compound label was the last element in C that was supporting another label. This, now unsupported, label must be added to the queue and processed in turn. Similar to the other AC algorithms, once the queue is empty the CSP is generalized arc consistent.

Mohr and Massini show results that this generic algorithm does work well for some problems. In practice for many problems, such as the global cardinality constraint as described by Regin [101], the number of admissible tuples that need to be processed can be too large to cope with. GAC is used in some CP tools, such as ILOG Solver, in special cases where it is practical to do so. In these cases it is a question of implementation. Solver's usage of GAC will be discussed in Section 3.2.8. Stergiou and Walsh also discuss GAC using a specific example, the all-different constraint [117].

This is a brief illustration of the algorithms available for arc consistency. The number of algorithms designed to make CSPs arc consistent should highlight the importance of arc consistency in CSPs.

3.2.5 Path Consistency

Even when a CSP has been made arc consistent it may be possible to make further deductions from the constraints and, hence, further domain reductions. Arc consistency considers all pairs of variables so the next logical step would be to consider all triples of variables. Consider the constraint network in Figure 3.4 (as taken from [113]). Although this CSP is arc consistent, the variable x can never be assigned the value 2; this would violate the constraint $y + z < 20$. Unfortunately, arc consistency is unable to detect this.

A path (x_i, x_j, x_k) in a constraint graph for a CSP is *path consistent* if and only if for each pair of values $v_i \in D_{x_i}$ and $v_k \in D_{x_k}$ that satisfy constraint C_{ik} there is a value $v_j \in D_{x_j}$ such that $(v_i, v_j) \in C_{ij}$ and $(v_j, v_k) \in C_{jk}$. If no such value v_j exists then (v_i, v_k) should be removed from C_{ik}. This states that there is no value v_j that is consistent with $x_i = v_i$

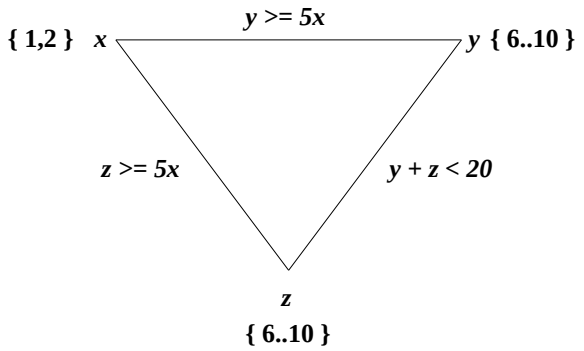

Figure 3.4: An Arc Consistent Constraint Network

and $x_k = v_k$. Removing a value from a constraint, such as (v_i, v_k) from C_{ik}, is known as *tightening* a constraint. A CSP is said to be path consistent if every path of length two in the graph is path consistent [121].

Like arc consistency there are several algorithms to achieve path consistency. Mackworth describes the first two known as PC-1 and PC-2 [82]. PC-2 is the most efficient of the two having a time complexity of $O(a^5n^3)$ and a space complexity of $O(n^3 + n^2a^2)$. PC-3 was devised by Mohr and Henderson but their description contained some minor mistakes [87]. This error was corrected by Han and Lee which produced the algorithm PC-4 [55]. PC-4 has both a time and space complexity of $O(a^3n^3)$. PC-5, as proposed by Singh, has a space complexity of $O(n^3a^2)$ but still has the time complexity of PC-4 [110]. PC-5 uses the same idea exploited by AC-6 in that variable supports are searched for only when required, thus reducing the space complexity of the problem.

The complexity of these algorithms illustrates that path consistency is a fairly expensive operation in comparison to making a problem arc consistent. Tsang suggests that running a path consistency algorithm on a CSP before a search starts may improve the search efficiency and some problems lend themselves to pre-processing more than others [121, 11].

Conventional CP tools don't employ PC. One problem with PC is that considering more than 1 constraint at a time increases the complexity of the problem. There are far more ways of choosing 2 constraints from a set of n constraints than 1. Additionally if a pair of constraints is not PC then you have to forbid the pairs of values of the relevant variables.

This, in effect, adds more binary constraints to the problem. This is less convenient than simple domain reduction which is what you would get from AC.

3.2.6 k Consistency

Freuder [43] has generalised the concept of consistency in CSPs and devised the concept of k consistency. k consistency (taken from [121]) is defined as follows:

> A CSP is said to be k consistent for $k > 1$ if and only if all $(k-1)$-compound labels which satisfy all the relevant constraints can be extended to include any additional variable to form a k compound label that satisfies all the relevant constraints.

This means if a problem is k consistent then if we choose values for any $k-1$ variables that satisfy the constraints amongst them and choose any kth variable, then there will exist a value for the kth variable that satisfies the constraints amongst all k variables. However, a CSP that is k consistent is not necessarily $k-1$ consistent. Additionally a CSP with n variables that is n consistent does not necessarily have a solution.

Cooper [19] has produced an algorithm for achieving k consistency, KS-1, and its complexity is $O(\sum_{i=1}^{k}(\begin{pmatrix} n \\ i \end{pmatrix} a^i))$. For values of k larger than three or four this algorithm becomes very expensive to run.

Now that k consistency has been defined we can see that node consistency is equivalent to 1-consistency, arc consistency is equivalent to 2-consistency and path consistency is equivalent to 3-consistency. Clearly it is possible to consider groups of four or more variables to generate higher orders of consistency but it is questionable whether or not it would be a good idea to do so due to the excessive cost and the benefits in doing so. The benefits gained from making a problem k consistent for larger values of k are not worth the cost (anyway, $k-1$ consistency is not guaranteed). k-consistency is never used in practice. Maintaining node and arc consistency is a cheap and effective operation and is much more

favourable to k consistency. Although k consistency can remove more inconsistent values from a CSP it will not eliminate the need to search for a solution in general.

3.2.7 Search Algorithms

Once a CSP has been defined various algorithms can be applied in order to find a solution(s). A solution to a CSP is defined as an assignment of a value to each variable that does not violate any of the constraints. The search tree for a CSP grows in size exponentially in relation to the number of variables in the problem (although the entire search tree will not be explored). Unless the problem is fairly small then the search for a solution will need some guidance.

3.2.7.1 Backtracking (BT)

The classic algorithm for solving CSPs is backtracking [51]. The BT algorithm instantiates (binds) each variable in turn and builds up a solution sequentially. Variables already bound in a search algorithm are known as *past* variables whilst variables that are unbound are known as *future* variables. The variable currently being considered by a search algorithm is known as the *current* variable.

First, assume the variables being labelled are done so in sequence such that x_1 is the first variable that will be considered, x_2 will be the second and so on until x_n is lastly labelled with a value. When the BT algorithm labels the current variable the assignment is checked against all past variables. If any constraint between the current and past variables is violated the BT algorithm instantiates the current variable to another value in its domain. If all the values of the current variable are tried with no success, then the algorithm *backtracks* to the previously bound variable. This variable is then assigned another value from its domain. If the BT algorithm successfully assigns a value to every variable in the CSP then a solution has been found. The algorithm can either terminate with the one solution or continue to find another solution(s). If there are no solutions the

BT algorithm will exhaust all possible opportunities and terminate.

To show an example of what the search space explored by the BT algorithm would look like the N-Queens problem will be used. The problem is stated as:

> Given an $N \times N$ chessboard the goal is to place N queens on the board such that no queen can attack any other queen.

For this problem to have a solution N must be greater than three; for this example the 4-Queens problem will be used. The CSP can be represented with four integer variables. Each variable represents a row of the chessboard. Each variable's domain, $\{1, \ldots, 4\}$, represents the column a queen can be placed in. The constraints on the problem are that no two queens can be on the same row, column or diagonal on the chessboard.

Figure 3.5 shows what the search space for this problem would look like when the BT algorithm is used. Each row in the 4×4 chessboards corresponds to a variable in the problem and a 'Q' in a particular row corresponds directly to the column to which that row's variable has been bound (a value between one and four). The location in the search tree where the search is unsuccessful and, therefore, must backtrack is marked by a circle with a X in it. The leaf node representing a successful search is marked with a circle. From this figure it can be seen that the search requires a fair amount of work. For example the first subtree (where a queen is placed in the top left corner of the chessboard) contains no valid solution at all. This is not detected because BT only checks constraints between the current variable and the past variables. With the BT algorithm this cannot be detected until the subtree is exhausted.

BT is the least favourable search algorithm as no effort is made to record any information about past failures which may aid the algorithm in backtracking[5]. A variable may be backtracked to and re-labelled repeatedly when it is an incorrect instantiation higher up in the search tree that is causing the failures. If a search algorithm was to incorporate

[5]However, BT is much better than generating all possible solutions and testing each one. When BT was introduced it was hailed as a great advance because of this.

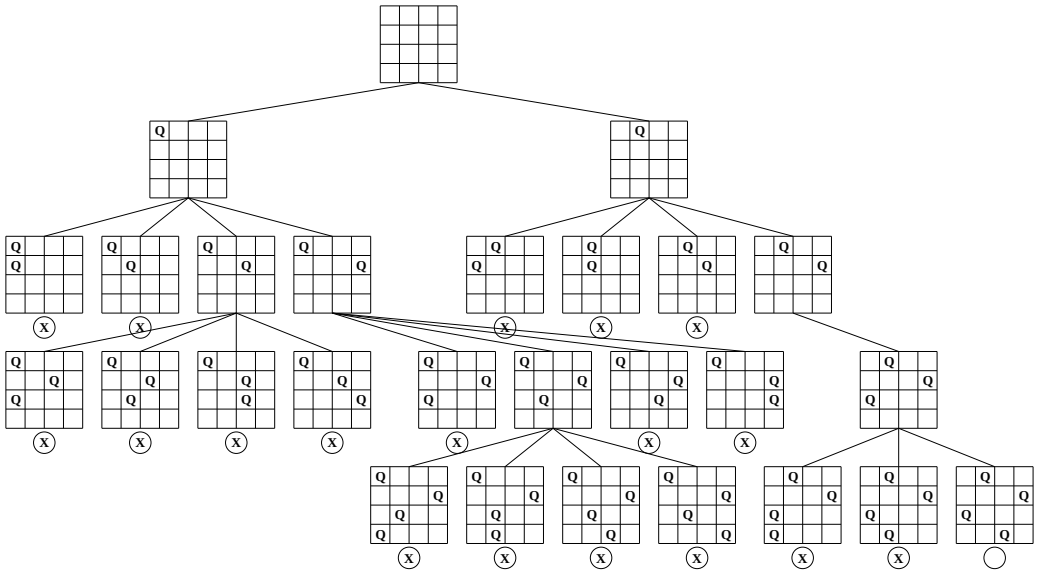

Figure 3.5: 4-Queens Search Using Chronological Backtracking (BT)

information gathered during the search then irrelevant backtrack points may be avoided. Search algorithms that do this will be explored next.

3.2.7.2 Backjumping (BJ)

Gaschnig devised a technique known as Backjumping (BJ) that avoids backtracking to nodes where the reassignment of values to them cannot possibly lead to a solution [49]. The algorithm for BJ is exactly the same as BT except when it backtracks. Recall that backtracking is needed if the current variable has no legal value in its domain. With the BT algorithm, once it has been found that backtracking is necessary, BT will always blindly go to the previous past variable and try to assign it another value. BJ identifies *culprit* labels or bindings which are earlier labellings that have contributed to the failure.

When a variable is instantiated to a value from its domain it is then compared with all the past variables to ensure the instantiation is consistent. If all the values in the current variable's domain are inconsistent with past bindings then backtracking is necessary. In this situation every value in the current domain must be in conflict with at least one past variable. For each value tried in the current variable's domain a note is made of the first

variable that it is in conflict with (the past variables are processed in the order they were bound). These are the culprit bindings. Next, BJ selects the most recent culprit binding to backtrack to as this binding is responsible for making at least one of the current variables domain values inconsistent. All of the assignments back to the selected culprit variable are undone. This culprit variable is assigned another (legal) value from its domain and the algorithm progresses from this point.

If the current variable was successfully bound, but backtracked to at a later point, and the available domain values turn out to be inconsistent, then BJ will backtrack to the immediate past variable. The immediate past variable is backtracked to due to the fact we know that the current variable has had a value successfully assigned to it (otherwise we would not have backtracked to it). Because there is no obvious culprit variable to use BJ will backtrack to the immediate past variable.

Other varieties of BJ have been devised. Dechter's graph-based backumping (GBJ) views the CSP as a constraint graph (for a binary problem) and tries to exploit the topology of the graph. When backtracking is necessary GBJ jumps back to the most recent variable, say x_j, connected to the current variable in the constraint graph, say x_i [29, 30]. If backtracking is needed at x_j then GBJ jumps back to the most recent connected variable to x_i or x_j, and so on. This may not be good as Gaschnig's BJ as no information about *why* a failure has occurred is recorded during the search. As a result of this there is no guarantee that the variable backtracked to was responsible for the domain wipeout forcing the backtrack.

Prosser developed a variety of BJ known as conflict directed backjumping (CBJ) [94]. During the algorithm a *conflict set* is maintained for each variable. The conflict set is updated when the current variable has been instantiated and the past variables are being checked for consistency. Any past variable, say x_j, that makes the current binding, say at variable x_i, inconsistent is added to the conflict set. If backtracking is required then the most recent variable in the conflict set is jumped back to. If the variable jumped back to, say x_j, itself causes backtracking then the conflict set used is the union of x_j's

and x_i's conflict set and the most recent variable in that set is chosen as the variable to backjump to, and so on. This algorithm is similar to graph directed backjumping except it exploits information gathered during the search. The conflict set generated may contain information gained from the search about several variables which neither Gashnig's BJ or GBJ take advantage of.

Out of three CBJ would be the best one to use. CBJ records more relevant information about culprit variables than BJ and the most recent connected variable backtracked to by GBJ is not guaranteed to have contributed to the domain wipeout of the current variable.

3.2.7.3 Backmarking (BM)

Another algorithm that takes advantage of information recorded during the search is Backmarking (BM) (also by Gaschnig [47, 48]). BM reduces the number of redundant consistency checks performed during a search for a solution to a CSP. There are two cases where consistency checks can be considered redundant. One is when we know the checking will succeed and the other is performing checks when we know they will fail.

Like BT and BJ when a variable has been instantiated we check the past variables (starting at the first variable instantiated and working our way down the search tree to just before the current variable) to ensure the assignment is consistent.

Consider the case where the current variable, say x_i, is given the labelling $< x_i, v >$. Next assume during consistency checking that it fails against the past variable x_j. Two useful pieces of information can be deduced from this.

First, we know that if at some point in the future x_i again becomes the current variable and the labelling $< x_i, v >$ is attempted then its consistency check with x_j will fail if x_j has not been re-labelled with a different value. The next piece of information that is gained from this consistency check failure is that if x_i becomes a current variable again and we have backtracked to some variable before x_j, say x_l where $l < j$, then when we try the labelling $< x_i, v >$ we know that no consistency checks are required for all the

variables x_1 to x_{l-1} as they will, again, pass since their labels have not changed since the last time the labelling $< x_i, v >$ was attempted. These two types of savings are referred to as type (a) and (b) savings by Nudel [91].

Two data structures are maintained during the BM search in order to be able to do tests to benefit from these types of savings. The first structure, *mcl* (maximum checking level), is an n by m array where m is the largest domain size of the n variables. When a consistency check is carried out between x_j and $< x_i, v >$ where $j < i$ then $mcl[i, v] = j$ (note that when a consistency check fails no more checks are made for that current variable labelling so $mcl[i, v] = i - 1$ for a labelling that was successful). The second structure, *mbl* (minimum backtracking level), is a one dimensional array of length n. An entry, say $mbl[i]$, records the shallowest variable we have re-labelled in the search tree since the last time x_i was the current variable.

Using these data structures two tests can be made when the labelling $< x_i, v >$ is attempted again (as described by Prosser [93]):

(a) If $mcl[i, v] < mbl[i]$ then we know that a consistency check has failed in the past against some variable x_j where $j < mbl[i]$. Since this variable has not been re-instantiated to another value the consistency test will fail.

(b) If $mcl[i, v] \geq mbl[i]$ then we know that the labelling $< x_i, v >$ has passed consistency checks with variables x_1 to $x_{mbl[i]-1}$. As these test would again succeed we only perform consistency checks between x_i and the variables $x_{mbl[i]}$ to x_{i-1}.

More in depth descriptions of BJ and BM can be found in [49, 47, 48, 30, 93, 56].

The proceeding algorithms, (BT, BJ and BM) are classified as *look-back schemes* as they focus on the current and past variables only. The next algorithms to be examined are classified as *look-ahead schemes* as they explicitly take future variables into consideration whilst binding the current variable [30].

3.2.7.4 Forward Checking (FC)

Haralick and Elliott's forward checking algorithm differs from the BT algorithm in one important aspect [56]. When the current variable is bound to a value all of the future variables in the CSP are checked against this value and inconsistent values are removed (only temporarily), thus pruning the search tree[6]. One significant advantage gained from FC is that if any of the future variables' domains become empty then it is immediately known that the current partial solution is invalid. Therefore either another value should be tried with the current variable (if possible) or the algorithm must backtrack to the previous variable. FC allows backtracking to be initiated much earlier than in the BT search algorithm.

When backtracking, to say previous variable x, the domains of the future variables are restored to what they originally were before x was bound. Figure 3.5 shows what the search space for the 4-Queens problem looks like if the FC algorithm is used. The 'x' values in the figure represent domain values removed due to the forward checking process. As can be seen this is a substantial improvement from the BT algorithm as much less of

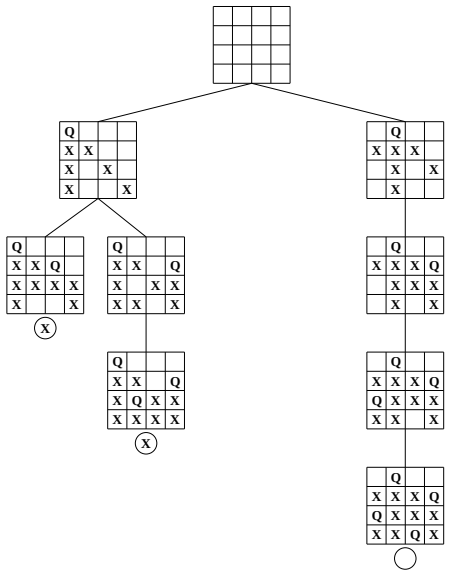

Figure 3.6: 4-Queens Search Using Forward Checking (FC)

the solution space must be searched. However, there is extra overhead in the FC algorithm

[6]Note: This only applies to binary CSPs

from the extra consistency checks performed. Another advantage with FC is that if the CSP is binary it is never necessary to check constraints between the current and past variables. Due to the lookahead a current value is always bound to a legal value with respect to the past variables in the problem. In some cases BT and FC can both do, roughly, the same amount of work. It is generally accepted that with larger problems FC is a better choice than BT.

The FC algorithm has been generalized to handle non binary[7] CSPs [58]. Forward checking only takes place under certain circumstances in the non binary case. A k-ary constraint is said to be *forward checkable* if $k-1$ of its values have already been legally instantiated and the remaining variable is uninstantiated. The current node bound in the FC algorithm may cause some constraints to become forward checkable. For each new forward checkable constraint FC will forward check the remaining unassigned variable.

3.2.7.5 Full Lookahead (AC-L)

The goal of AC-L is to enforce a higher degree of consistency on the problem (thus reducing the number of nodes explored in the search tree). The *Full Lookahead* or *AC-Lookahead* (Arc Consistency Lookahead, (AC-L)) algorithm is identical to the FC lookahead but, additionally, arc consistency is maintained between future variables, thus, potentially removing even more values from the domains of unbound variables [56]. This guarantees that there will be a pair of legal domain elements between every pair of unbound variables[8].

Figure 3.5 shows what the search space for the 4-Queens problem when the AC-L search algorithm is used. Right away AC-L determines that the first subtree (with a queen placed in the upper left hand corner of the board) does not contain a solution as a result of keeping the future variables arc consistent. Upon examination of the second subtree the solution is found almost immediately. Once a queen is placed in the second column of the first row

[7] Bacchus and van Beek [3] compare FC applied to non binary CSPs and FC applied to non binary CSPs whose constraints have been converted to binary ones.

[8] Another variation of the AC-L algorithm in the literature is the Maintaining Arc Consistency (MAC) algorithm. The only difference is that the problem is first made arc consistent before the search is started [107].

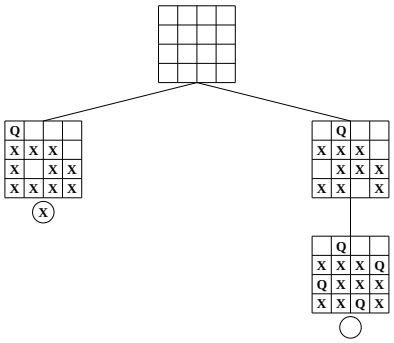

Figure 3.7: 4-Queens Search Using Full Lookahead (AC-L)

the forward checking and arc consistency determine that each of the remaining variables can only be bound to one value, thus finding a solution.

Haralick and Elliott [56] suggest that the extra effort used in an algorithm like AC-L may not justify the large number of consistency tests incurred. Dechter and Frost [30] suggest that more recent work shows that the higher levels of look ahead become more useful when applied to larger and more difficult problems. This enables the detection of an insoluble state earlier in the search process such that backtracking is initialized earlier than in other algorithms such as FC. Sabin and Freuder, in their experiments, report that maintaining full arc consistency during search was more efficient than the limited consistency maintained in algorithms such as FC except on very easy problems [107].

Search algorithms themselves often need heuristics to help guide the search. The next two topics of discussion, *variable* and *value* ordering address this.

3.2.7.6 Variable Ordering

During a search of a CSP's solution space, the search algorithm selects a variable to bind and then binds it. The order in which the variables are selected for binding is referred to as the *variable ordering* of the problem. There are two types of variable ordering:

1. *Static* variable ordering (SVO) - the variable ordering for the problem is decided before the search starts.

2. *Dynamic* variable ordering (DVO) - the variable ordering is dependent on the current state of the search.

Dynamic ordering is not a good idea for all search algorithms. For example if the BT search algorithm is used, since there is no lookahead whatsoever, there is no extra information available to help in aiding the selection of the next variable to be bound to a value. If the FC or AC-L search algorithms are used then it is possible to gain some insight from the domains of future variables and use a heuristic to select the next variable to be bound.

One popular heuristic is known as 'smallest remaining domain' (SRD). Using this heuristic the variable with the smallest remaining domain will be the next variable to be bound. This makes the assumption that any value in the domain is equally likely to appear in a solution; the more values a domain contains, the more likely it is that one of them will be successful.

There are other variations on this theme such as the algorithms described by Brélaz or Frost and Dechter [12, 45]. For many problems, heuristics like SRD work reasonably well. Gent *et al.* have done an empirical study on DVO heuristics such as SRD and others [50].

3.2.7.7 Value Ordering

The order in which values are selected from a domain of a variable when it is being instantiated is referred to as the *value ordering*. The order in which domain values are tried has a direct impact, like variable ordering, on the shape of the search tree.

In the case of value ordering the popular view in the literature is that one would want to select domain values that are *most likely to succeed*. Such a choice should have a minimal impact on future variables and would seem more likely to lead to a solution. This in itself can be difficult. If much is known about the problem being solved then this may offer clues as to which values are more promising than others. If no such information is available it may be possible to see what the state of the future variables would look like by applying

forward checking for each potential value in the domain and base a decision on the result of that (this is referred to as look ahead value ordering (LVO) [44]). Forward checking from each possible value in a domain would be an expensive proposition to say the least and may not be worth doing.

Care should be taken with both variable and value ordering heuristics. Smith [114] has examined a car sequencing problem and has reported that variable ordering heuristics such as SRD and value ordering heuristics similar to most likely to succeed are poor choices for that problem. When deciding on variable and value ordering heuristics for a CSP, as noted by Smith, it is often a case of trial-and-error and insight.

3.2.7.8 Hybrids

Although somewhat beyond the scope of this discussion here, hybrids of the algorithms discussed so far are possible and, indeed, have been tried and experimented with.

Prosser carried out an evaluation of hybrid schemes as reported in [93]. Hybrids examined were forward checking combined with backmarking (FC-BM), conflict directed backjumping (FC-CBJ) and backmarking with conflict directed backjumping (BM-CBJ) [93, 95].

Further examples can be seen in the work of others who have evaluated or developed hybrid schemes [29, 69].

3.2.8 ILOG Solver

The constraint programming software being used for the work described in this thesis is ILOG Solver[9] 3.2 [62, 63]. ILOG Solver is a C++ library providing classes and algorithms to help formulate a constraint programming framework for solving CSPs.

Puget originally developed a LISP-based constraint programming language known as

[9]ILOG hold a annual international users' conference where people can present work done with Solver (including scheduling problems) as well as experiences and techniques that they have found useful. Four such conferences have been held thus far [64, 65, 66, 67].

PECOS [96]. As PECOS was implemented using object classes it seemed natural to create an implementation of a similar library in C++. This was the motivation behind ILOG-Solver.

ILOG-Solver is described by Puget [97]. Implementing the constraint programming language in C++ has two advantages. First, everything in ILOG Solver is an object (variables, constraints and the search algorithms). This makes Solver easily extendible as you can define new classes. The second advantage highlighted by Puget is that these objects can be used for modelling the problem to be solved which gives an advantage from a software engineering standpoint.

Arc consistency is maintained in ILOG-Solver through use of the AC-5 algorithm [33]; ILOG-Solver also employs an algorithm similar to AC-L during search.

Puget and Leconte describe how constraints are represented in Solver [98]. Constraints are objects. Users can create their own custom constraints and define their own rules for propagation. As discussed earlier the algorithm AC-5 is parameterised such that the routines to actually enforce arc consistency are left open to the user. Solver takes advantage of this fact by allowing the user to write their own functions to do this. The Δ values (values removed from a variable's domain during the AC-5 process), as defined in AC-5 [33], are present in Solver and easily accessible to the user specified arc consistency routines. Each Solver constrained variable maintains several lists of constraints. Each list of constraints is related to how the constraint is invoked for the purpose of constraint propagation. There are three such contexts:

1. When the variable is bound.
2. When the variable's domain is changed.
3. When one of the bounds of the variable's range is changed.

This flexibility enables some forms of GAC to be implemented. One form of GAC is implemented by Solver using *bounds consistency*. An example of this would be the constraint

$X + Y < Z$. To ensure the variables X, Y and Z are arc consistent the bounds of the variables can be examined and illegal values would be discarded. Three checks can be made. These are:

$$min(X) + min(Y) < \mathbf{min(Z)} \tag{3.11}$$

$$\mathbf{max(X)} < max(Z) - min(Y) \tag{3.12}$$

$$\mathbf{max(Y)} < max(Z) - min(X) \tag{3.13}$$

where the domain bounds in bold represent the domains that may be reduced. A constraint such as this would be activated when the appropriate bound of a variable is modified, which can be detected by AC-5.

Another example of GAC in Solver can be found in the `IlcAllDiff()` constraint. This constraint can be applied to an array of constrained variables and ensures that none of the variables may be bound to the same value. This constraint would be activated when any variable is bound to a value, say x, and the value x would be taken out of the domains of all remaining unbound variables. A discussion of how constraints are represented in Solver as well as how arc consistency is maintained is discussed further in [98].

At each step during a search for a solution of a CSP with Solver a *choice point* is created. Each choice point has two branches. The first branch binds a variable to some value, say i, and any constraints that can be propagated by this action are activated. The effect of this instantiation on other variables' domains is found and processed via constraint propagation. If a domain wipeout is detected the second branch is tried which removes the value i from the domain of the variable in question and tries another value from its domain (at a new choice point). If the variable assignment was successful then another variable to bind is chosen and another choice point is formed. Choice points can be marked and backtracked to dynamically so Solver is not restricted to chronological backtracking. This allows search techniques such as BJ to be implemented. Additionally the user can write their own routines for both variable and value ordering.

3.2.9 Summary

A CSP consists of three things. A set of variables, a finite domain for each variable and a set of constraints restricting the values that the variables may simultaneously take.

Constraints are defined as a set of compound labels for the variables in the constraint. Alternatively a constraint can be viewed as a subset of the cartesian product of the domains of the variables they involve. Once a CSP has been defined it is useful to make the problem arc consistent such that every pair of variables each have a legal value to which they can be simultaneously bound. Ensuring a CSP is arc consistent helps to reduce the size of domains in the CSP, thus reducing the problem size. Path consistency is also a desirable feature in a CSP but is more expensive and impractical than arc consistency to achieve.

Solving a CSP involves choosing a search algorithm. They are classified into two schemes: look-back and look-ahead. Look-back schemes only focus on the current and past variables whilst look-ahead schemes take future variables into consideration when binding the current variable.

BT is the simplest algorithm to use but is also the most inefficient as it only compares constraints between the current variable and the past variables. BJ and BM are somewhat more sophisticated than BT as they try to exploit knowledge gained in the search up to the current variable. Algorithms such as forward checking or full lookahead search examine future variables and compare them with the current variable's binding to ensure inconsistent values are taken out of their domains. The full lookahead search goes one step further by ensuring all the future variables are arc consistent with one another. Hybrids of the above algorithms have been made with some success.

The order in which the variables are bound as well as the order in which a variable is assigned values from its domain is also important. The smallest remaining domain heuristic for variable ordering is probably one of the most common approaches and works well with many problems. Value ordering is more tricky as it is often difficult to identify which values are more likely to lead to a solution. As is the case with most CSPs the properties

that an individual problem has will have an influence over which search algorithms and variable/value ordering heuristics will be used.

3.3 Scheduling with Constraint Programming

One of the areas where constraint programming has seen some popularity is in the field of scheduling. Le Pape [76] discusses a historical perspective of constraint programming with regards to scheduling systems and tools. In the following sub sections we will examine some scheduling systems that employ constraint programming techniques. Specifically, those systems that apply constraint programming towards bus, rail or air crew scheduling problems.

3.3.1 The COBRA System

The COBRA train driver scheduling system was developed jointly by PA Consulting and COSYTEC. The constraint handling package CHIP was used in the development [58].

Unfortunately there is little published about this system. The only published material the author is aware of is an abstract, some unpublished presentation slides and a brief non technical description published by PA Consulting [16, 17, 1]. The work was carried out on behalf of North Western Trains in the UK.

COBRA appears to split the train schedules up into driving activities (this would correspond to a piece of work). The problem is represented as a directed graph where the set of nodes correspond to the driving activities or resources. The arcs connecting the nodes have a weight corresponding to the cost of the connection between the two activities plus any penalties in order to discourage disadvantageous links. The driver shifts appear to be generated by a shortest path algorithm.

The results given claim that the initial solutions found are better than manually generated

ones. It is unclear how it competes with other systems as no concrete figures are given.

3.3.2 Yunes *et al.*

Yunes *et al.* describe a linear programming (LP) / constraint programming (CP) hybrid applied to a bus driver scheduling problem in the city of Belo Horizonte in Brazil [143, 144].

Given a set of pre-generated shifts the LP portion of the system is applied to solve the associated set partitioning problem. This produces a continuous solution. The CP portion is found in the column generation stage of the LP. Once the continuous solution of the current problem has been found in the LP a CP is invoked which generates new columns (shifts) which are added to the LP. The LP is solved again and then more new shifts are added via the CP. This cycle continues until such point that no shifts that can reduce the overall cost of the schedule can be found. A branch and bound process is then invoked which finds the final solution.

From the LP process the CP is supplied with the values from the dual variables (pieces of work) from the LP process. A vector of constrained variables is initialized where the domain of each variable is a range of integers, each of which represent a piece of work. Five other vectors are created that contain the start time, end time, duration, departure depot and the arrival depot. These vectors are linked to the vector of constrained variables in that when a variable is bound they are updated with the relevant value. Constraints can be placed between the vectors to ensure the duty generated is legal. The vector of variables is then repeatedly solved producing more duties which are then fed back to the LP as new columns. Yunes *et al.* also describe a version of the CP designed to solve the entire driver scheduling problem but found it was only viable for small problems.

This approach differs from systems such as TRACS-II in that all the duties that may be found in the LP are not generated in advance. The CP generates these duties on the fly when the LP has found the optimal continuous solution for the set of duties at hand. The system is also designed to find an optimal solution and examines a much larger proportion

of the solution space than systems such as TRACS-II. As a result this system cannot cope with the size of problems that other systems routinely deal with. The results given involve finding optimum solutions for problems needing 25 and 19 duties respectively.

Pure IP and CP approaches were also tried but found to be unsatisfactory in their results. With the pure IP approach all feasible duties were first generated and used in the associated set partitioning problem. The size of this set of potential duties quickly became too large (millions) to be feasible as no effort was made to restrict its size.

The model used for the pure CP approach is an extended version of the CP solved to create additional duties in the column generation method. Rather than a single vector containing integer constrained variables representing pieces of work, a matrix is created with each row being a representation of a duty. The five vectors containing supplementary information are also represented by matrices with each row corresponding to a row in the constrained variable matrix. By posting similar constraints as those found in the column generation CP approach the CSP is solved and a duty schedule is produced. This worked well only for small schedules. Optimal solutions were only proven for exceptionally small solutions. Yunes *et al.* state that with the huge solution spaces involved the lack of good problem specific knowledge hinders the search. Solving scheduling problems of a moderate to large size, given their complexity, in an optimal fashion is intrinsically difficult for pure CP techniques [144].

3.3.3 Curtis, Smith and Wren

Curtis, Smith and Wren use constraint programming by representing the bus driver scheduling problem as a set partitioning problem [23, 22]. A continuous linear programming (LP) solution from the associated set partitioning problem is used as a guide for variable and value ordering in the CSP. Set partitioning is used rather than set covering due to the fact it is easier to perform constraint propagation (as a piece of work is covered by exactly one driver as opposed to at least one driver).

The constrained variables represent pieces of work that are to be assigned to drivers (P_i where $i \in I$, the set of indices of the pieces of work). The domains of the variables correspond to an index into a set of pre-generated duties (before the CSP is invoked a large number of duties is pre-generated, it is from this set the final partition of duties is to be chosen). Because the problem is modelled as a set partitioning problem useful constraints can be applied to the variables in order to aid propagation. For example if P_i is assigned the value j then all other variables, say P_k where $k \in I \setminus i$, containing duties in their domain that cover any of the work duty j covers are removed. Some pre-processing steps, similar to those used in Müller's approach to set partitioning using constraint programming [90], are discussed that can reduce the size of the search space but these were found to be too memory intensive to use in practice. The value to be optimised is the number of duties used in the final solution.

Rather than solving the problem directly by assigning values to the variables, P_i, an additional set of variables is created using information gained from the continuous solution generated from a linear program (TRACS-II in this case). Specifically, information on relief opportunities (points where drivers may change over) is extracted. A set of variables $R = \{R_k, k = 1, \ldots, r\}$ is created. R_k is defined as an 'active' RO and r is the number of such ROs; each has a binary domain of $\{0, 1\}$ to indicate whether or not the corresponding RO is used. Active ROs represent ROs that need to have a value chosen for them (that is they are in between the start or end of a bus). For every RO variable there is a corresponding variable, P_i, such that R_k is the start of a piece of work i. Constraints are placed on the variables R_k such that if $R_k = 1$ (the RO is used) then the adjacent piece of work must be covered by a different duty ($P_{i-1} \neq P_i$). If $R_k = 0$ then the adjacent pieces of work must be covered by the same duty ($P_{i-1} = P_i$). This proves to be a powerful method for reducing the search space as Curtis *et al.* state that the choice of the piece variables is trivial after all the R_k variables have been assigned values. The R_k variables are ordered on their weighting as extracted from the continuous solution (where the weighting falls in the range $[0 \ldots 1]$). The value ordering tries the value 1 first for R_k variables whose weighting is greater than 0.5 in the continuous solution. Any unbound P_i variables are

dealt with after all the R_k variables have been assigned values.

Curtis *et al.* report that this approach works reasonably well for small problems but not as well as a pure ILP approach such as TRACS-II.

3.3.4 Air Crew Scheduling

The structure of the air crew scheduling problem is similar to the bus and rail driver scheduling problems. Constraint programming techniques are more common in this field due to some important differences between bus/rail driver scheduling and air crew scheduling. Pieces of work in a bus/rail driver scheduling context can be quite short, often measured in minutes. Work pieces in a air crew scheduling problem can be very long as they usually represent the length of an individual flight. From this it follows that air crew scheduling problems generally have fewer pieces of work. Because of this it is easier to generate all (or a good proportion of) of the possible duties that can cover the work in the problem.

Mathematical programming methods are popular within the context of air crew scheduling. Approaches similar to that of TRACS-II seem to be the most common. One important difference is that the scheduling problem is often modelled as a *set partitioning* problem rather than a set covering problem.

The terms used in air crew scheduling are different than those found in the bus and rail driver scheduling world. Indeed, there are differences in bus and rail driver scheduling terminology even in between countries (as highlighted in Hartley's glossary of bus and rail driver scheduling terms [57]). A piece of work is often referred to as a *trip*, *duty* or a *flight leg*. A *pairing* is described as a sequence of trips originating and ending at the same crew base for a single crew which corresponds to a duty in bus/rail terminology[10]. The differences in terminology will be highlighted as they are introduced in the following subsections.

[10] See Antes [2] for a good description of what is involved in the entire airline scheduling process

3.3.4.1 The CREM System

Halatsis *et al.* describe a airline crew scheduling system known as CREM[11] developed at the University of Athens as part of their ESPRIT III Project PARACHUTE (PARAllel Constraint Handling for User TEchnologies) [53]. Olympic Airways (OA) is the targeted end user of this system.

The application is written with ILOG Solver [97]. Halatsis further states that the parallel portion of the method is implemented using a parallel version of ILOG Solver.

The CREM system divides the problem into four subproblems. The first part involves generating duties (*duties*, in the context of this paper, are analogous to *stretches* in our earlier defined bus driver terminology). Air crew scheduling duties can be split into several *sectors* each of which is a flight between two airports. All possible duties having between 1 and i sectors are generated using Solver. Constraints taken into account include maximum flight time, duty time, flight time given type of aircraft, etc.

The second step is combining the duties into *pairings*. A pairing is defined as a sequence of duties, separated by the required rest times, starting and ending at the same home base which may not contain any days off (this would correspond to what we have been referring to as a *duty* in a bus driver schedule). A pairing has a limit on the number of duties it may consist of (say D). A limited number of pairings containing from 1 to D duties is constructed using ILOG Solver. These pairings have various constraints placed on them such as maximum flight time, pairings must start and end at the same home base, etc.

Once a set of pairings has been generated the third step is invoked. This is represented as a set partitioning problem and is modelled as a series of equations which is solved using ILOG Solver. Parallelism is exploited in this process although the paper does not give any details on this.

The last step allocates the pairing selected to individual crew members to form the roster

[11] CREM stands for CREw Management.

for a period of time, generally a month. The pairings are assigned such that the assignment of pairings will be about the same for all crews involved. Crew members are modelled as resources and pairings are viewed as activities that require crew resources.

Halatsis *et al.* only describes a prototype system and no concrete results are given although it is noted that OA have found the results produced to date quite satisfactory.

3.3.4.2 Guerinik and Van Caneghem

Guerinik and Van Caneghem discuss a constraint programming approach used both for bus driver and air crew scheduling [52]. The system described is implemented using both in C and Prolog III.

The driver scheduling problem is modelled as a set partitioning problem. The first step involves generating a set of all potential crews (duties in the bus driver scheduling context). Next a pre-processing stage is invoked. Any crews that are to be selected automatically are flagged and various reductions are made based on an examination of the crews produced. Next, the variables (crews) are assigned values (1 or 0 depending on whether or not they are used) via enumeration until the optimal solution is found. This is obviously an expensive way to solve a problem of anything other than a trivial size. In order to speed up the execution time a mathematical programming method is used in order to aid the enumeration process by intelligently selecting the next node (driver) to include in the solution. A set partitioning LP system is solved using a simplex implementation. The variable with the value closest to one is selected and enumerated next. This system, at the time the paper was published, was in use at a French airline company.

The results achieved with this approach are not as good other pure ILP methods.

3.3.5 Summary

Constraint programming on its own has been found to be not as effective as pure mathematical programming approaches. In most of the systems examined the scheduling problems are modelled as *set partitioning* problems due to the fact this model allows for more effective constraint propagation for the CSP models used. The methods examined above have subtle differences between them but the variables in the CSPs defined are generally either the duties themselves or pieces of work that comprise duties. Some of the previously described approaches employ mathematical programming methods in order to help guide the search. This has been shown to dramatically improve the quality of the solutions that originally relied on pure CP methods but also served to further highlight the advantages of mathematical methods. The role of constraint programming in the context of crew scheduling problems would seem to lie in the realm of hybrid systems where they work in conjunction with mathematical programming methods in order to complement them.

Chapter 4

The TRACS II System

4.1 Introduction

The TRACS-II system was developed at the University of Leeds. It is the successor to the IMPACS system developed earlier (described in section 2.2.2) at the University of Leeds and, like IMPACS, uses a set covering formulation to solve the driver scheduling problem. TRACS-II is used both for bus and rail driver scheduling problems but the work within this thesis is directed towards bus driver scheduling. The features in TRACS-II that are particularly relevant for train driver scheduling will, thus, not be described.

The description of TRACS-II is taken mainly from Fores *et al.* [38, 40, 39, 128], Kwan [70], and the TRACS-II user manual [122]. Fores extended the TRACS-II ILP capabilities by adding a column generation process as an alternative solution technique.

As an aside, it should be noted that the version of TRACS-II that was used at the time the research in this thesis took place was a prototype to the version that is available today. As a result some of the procedures and limitations found in the following description of TRACS-II are no longer present (or have been improved). Appendix E lists the differences between this prototype system and the current state of TRACS-II[1].

[1]The current version of TRACS-II is described in [136, 41].

4.2 The TRACS-II System

The TRACS-II system consists of four steps. Figure 4.1 shows these steps in order. They are as follows:

Step 1 - The first step involves creating a large set of potential duties from the *labour* and *vehicle* information files supplied by the user. The BUILD program carries out this task.

Step 2 - The second step is invoked if the set of potential duties is too large to process in the third step, SCHEDULE. SIEVE reduces the set of potential duties generated in the BUILD process. It is also possible that more than one set of potential duties could have been generated by BUILD for the same problem (perhaps these sets contain different properties that justified their separate generation). A procedure known as MERGE exists to merge together several sets of potential duties.

Step 3 - The third step, SCHEDULE, is where the solution is generated. A set covering ILP approach is used to select the duties to cover the work in the bus schedule. The problem is that of covering all the work whilst attempting to minimize the cost of the schedule (with the first priority being the minimization of the number of drivers used and the second priority being the cost of the individual duties).

Step 4 - The fourth step, DISPLAY, simply translates the solution generated by SCHEDULE into an output that is easy to read.

These four steps will be described in some detail in the following subsections.

4.2.1 Step 1 - The BUILD Process

The first step is to generate a large set of potential duties that have been validated against labour agreement rules. A program called BUILD carries out this step. The input to BUILD consists of two files.

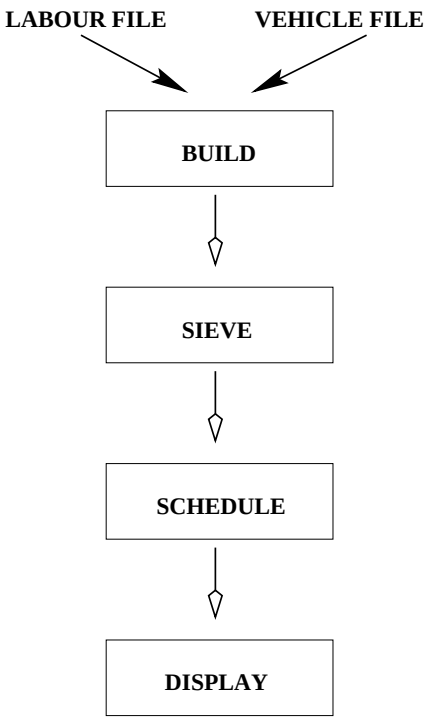

Figure 4.1: The TRACS-II Process

1. Labour Agreement File - This file contains information on rules in the labour agreement as well as restrictions on the type of duties formed in BUILD. Information in this file includes:

   (a) Time ranges for various types of duty (EARLIES, LATES, etc).

   (b) Maximum joinup time.

   (c) Minimum/Maximum length of a spell.

   (d) Minimum first and second stretch lengths.

   (e) Minimum/Maximum mealbreak lengths.

   (f) Earliest/Latest start/finish of a mealbreak.

   (g) Minimum/Maximum cost of 2/3 part duties.

   (h) Earliest/Latest signing on time.

   (i) Earliest/Latest start on bus.

   (j) Earliest/Latest finish on bus.

   (k) Maximum spreadover.

2. Vehicle File - This file contains information on the vehicles for which the duty schedule is to be generated. This includes information such as:

   (a) Relief Opportunities.

   (b) Depots Used.

   (c) Travel time between relief points (for join-ups, mealbreaks).

   (d) Travel time between relief points and the canteen.

   (e) Signon/Signoff times.

BUILD generates a large set of potential legal duties according to the parameters set out in the labour and vehicle data files. BUILD also calculates and records the cost associated with each potential duty generated. The costs include features such as wages and penalties for undesirable traits that may be present in the duties generated. BUILD is capable of building up to four part duties. First, BUILD considers every relief opportunity in the vehicle schedule and generates all possible legal two part duties. Next, if needed, BUILD generates a set of three/four part duties in a similar fashion. Since the number of potential three/four part duties that can be generated from a set of bus workings is quite large, not all three/four part duties will be generated and only a reasonable subset is produced. Three/four part duties are not introduced during the BUILD process if the work they cover is sufficiently covered by previously generated two part duties.

Since not all duties can be realistically considered in any solution process it is possible that the quality of the overall solution will be affected. There is no *guarantee* that the set of duties generated with BUILD will produce an optimal solution. However, it should also be noted that, for the most part, the types of duties that are not generated with TRACS-II are duties that are unlikely to participate in an optimal solution. For example, exceptionally short duties are generally avoided. Fortunately experience with TRACS-II has shown it is a very effective system finding the optimal or near optimal solution on many problems.

4.2.2 Step 2 - The SIEVE Process

In order for an ILP set covering model to be viable, some restriction must be placed on the number of constraints and variables in the problem. The set of potential duties that can be generated can easily number in the millions. Through the selection of sensible parameters in the labour agreement rules, this number can be greatly reduced to tens of thousands or even hundreds. It can still often be the case that too many duties are generated to be considered in the ILP and further reduction is necessary. The SIEVE program can further reduce the size of a generated set of duties from BUILD. SIEVE is an interactive program that uses a variety of heuristics to try to discard duties that are not likely to be necessary for a good solution.

The minimum that SIEVE does is remove identical duties from the set of potential duties generated. This can occur if more than one set of potential duties has been generated (perhaps due to more than one labour agreement file being used) and the sets of potential duties have been merged together.

More advanced options look at all pairs of potential duties. It is determined whether any of these duties are considered redundant. Redundant, in this context, means that a duty is entirely contained within another duty (and, thus, if the duty is discarded the work it covered can still be covered by another duty). An example of this can be found in Figure 4.2. Potential duties B and C in Figure 4.2 can be thought of as redundant compared to potential duty A. Clearly potential duty A covers all the work in duties B and in C. Some redundant duties are cheaper than those containing them; an option allows the user to choose whether to retain such duties.

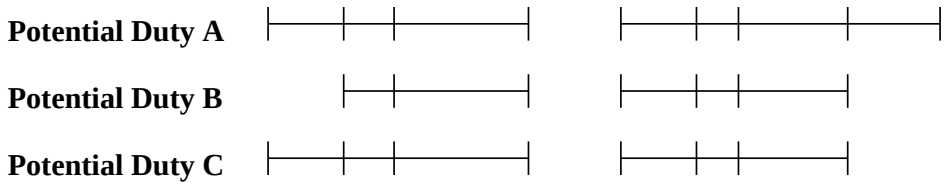

Figure 4.2: Redundant Duty Deletion

If it is still deemed that the set of potential duties is too large, the user has the option of

reducing the number of duties to some desired total. Duties which contain work which is covered by many other potential duties, and which are inefficient, are targeted for deletion in this case.

Before the ILP stage of TRACS-II is entered there is one further duty reduction process used. The program is referred to as the P-Process. The P-process examines the set of potential duties. Any duty that uses a relief point (that is not the beginning or end of a vehicle) that is *not* used by another duty is removed from the set of potential duties. Specifically, it searches for duties that have no other duties sharing its beginning or end relief opportunity for any of its spells (provided, of course, that the work covered by such a duty is sufficiently covered by other duties). This method is repeated until a pass through the set of duties results in no more duties being removed. This helps to reduce the size of the potential duty set further.

4.2.3 Step 3 - The SCHEDULE Process

The SCHEDULE process is where the duty schedule is selected. A ILP set covering process is used, similar to the one used in IMPACS. The description of the mathematical component of TRACS-II is taken from Fores *et al.* and Fores and Proll [40, 39].

4.2.3.1 TRACS-II Mathematical Model

The TRACS-II ILP model is as follows (from [40, 39]):

$$\text{Minimize} \quad \sum_{j=1}^{N} D_j x_j \tag{4.1}$$

$$\text{Subject to} \quad \sum_{j=1}^{N} A_{ij} x_j \geq 1, i = 1, 2, \ldots, m \tag{4.2}$$

$$x_j \in \{0, 1\}, j = 1, 2, \ldots, N.$$

The m constraints are the pieces of work to be covered in the bus schedule. The n variables correspond to the duties in the generated set. Therefore $A_{ij} = 1$ if the j^{th} duty covers the i^{th} piece of work; D_j is the *weighted* cost associated with the j^{th} duty. Variable $x_j = 1$ if the j^{th} duty is in the schedule, 0 otherwise.

The D_j coefficients in objective function (4.1) are defined as:

$$D_j \quad = \quad W_1 + C_j \qquad \text{for } j = 1, \ldots, N \tag{4.3}$$

where

$$W_1 \quad = \quad 1 + \text{sum of } X \text{ largest } C_j \text{ values} \tag{4.4}$$

C_j is the cost of duty j and the W_1 value is known as a *Sherali weight* [109]. This is calculated as 1 plus the cost of the X most expensive duties. X is calculated as the number of shifts in the original solution plus the number of uncovered pieces of work. This combines the objectives of minimizing the number of duties and cost whilst maintaining the priority of minimizing the number of duties as the primary objective. This ensures that the model will find the minimum number of shifts needed (from the given set of potential duties). The Sherali weight was introduced to the model so that only one objective function need be solved. The previous version of SCHEDULE used two objective functions in the ILP stage. The first one minimized the number of duties whilst the second objective function minimized the cost (including a constraint on the number of duties as ascertained from the first objective function). This is beyond the scope of the current discussion but full details on the TRACS-II mathematical programming model can be found in Fores *et al.* [40]. The C_j values combine wage and penalty costs for duty j.

This is a simplified version of the model used. Side constraints can be added to the model restricting the numbers of various types of duty in the schedule (for example, a limit placed on the number of split duties to appear in a solution). There are also situations where some of the constraints in (4.2) are treated as strict equalities. This occurs mostly with

work at the beginning and end of a vehicle.

4.2.3.2 Solution Techniques

There are two solution techniques employed by TRACS-II to solve the previously defined set covering problem. A dual steepest edge method is used for sets containing up to 30,000 duties and column generation, employing a primal steepest edge method, is used for larger sets of duties (up to 100,000 in the prototype used).

The first step before either solution technique is used is to produce an initial solution from the set of potential duties remaining after the SIEVE and P-Process. A heuristic accomplishes this by sequentially selecting the uncovered piece of work that has the least number of available duties to cover it [128]. From these duties, the duty covering the most uncovered work (including the piece of work just selected) is then selected. This is repeated until all the work is covered. This solution then has to be transformed into a basic solution for the ILP model.

In the case of column generation, when the initial heuristic solution is generated, a subset of the available duties is also selected. The ILP problem is then relaxed in that the variables are allowed to take non integer values and is then solved. Once a continuous solution has been found the TRACS-II column generation process looks for a further set of duties, from among those generated by BUILD, to add to the current subset in the continuous solution in order to improve the solution. If such a subset is found the duties are added and a new continuous solution is found. The process is repeated (subsets of duties are added) until no more duties can be found that will improve the continuous solution. An integer solution is then found using a branch and bound process (see section 4.2.3.4).

The column generation technique employed here is an interesting contrast to the column generation technique used by the CREW-OPT system outlined in section 2.2.5. CREW-OPT generates new duties in its mathematical component, using a shortest-path algorithm [31, 103].

Experiments have demonstrated that problems with smaller duty sets can be solved using the first technique quicker than when column generation is used [39].

Complete details of the above techniques can be found in [39, 38].

4.2.3.3 REDUCE

Once a continuous solution has been found using either of the two methods outlined previously, a process known as *REDUCE* can be invoked (this is an optional procedure but is generally recommended).

The REDUCE procedure works on the hypothesis that the continuous solution gives a useful indication of how the vehicle work should be covered in the schedule. This can be exploited in a fashion that reduces the size of the problem solved in the branch and bound phase.

Most of the constraints in the model are concerned with pieces of work. Each piece of work starts and ends at a relief opportunity. Take the vehicle block in Figure 4.3 as an example:

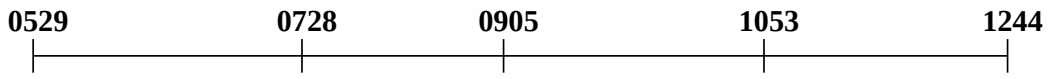

Figure 4.3: Bus Block Before REDUCE Process

The vehicle block in Figure 4.3 would generate four workpiece constraints. The first and last relief opportunities on any block must be used by at least one duty in the continuous solution (assuming all the work is covered). Any relief opportunity not used in the continuous solution separates two pieces of work (which equates to two constraints in the set covering model). Assume the third relief opportunity in the running board represented in Figure 4.3 is not used in the continuous solution. It could be eliminated which would give a running board that looks like the one found in Figure 4.4.

The removal of the unused relief opportunity, as shown in Figure 4.4, will allow the removal

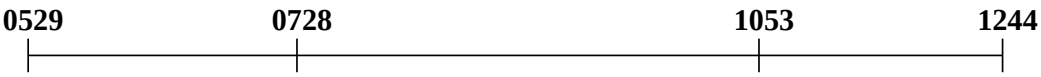

Figure 4.4: Bus Block After REDUCE Process

of one of the two work piece constraints surrounding the removed relief opportunity. In addition, duties using these rejected relief opportunities (these duties are not used in the continuous solution) are themselves removed from the problem (thus, reducing the number of variables in the LP).

Fores *et al.* [40, 39] report that from 13-50% of the total workpiece constraints and 50-90% of the potential duties can be eliminated from the LP before branch and bound with this method. This provides a significant increase in speed.

Smith [112] used a similar method to REDUCE with the IMPACS[2] system. Smith observed that if some variables are removed from the problem then it is impossible for a optimal solution from the remaining variables to be better than that from the full set. However, Smith also reported that the quality of the solutions produced was not found to be consistently better or worse than if the entire set of variables was used.

4.2.3.4 Branch and Bound

Once a continuous solution has been found using either of the methods outlined previously (and, potentially, REDUCE has been called), a branch and bound process is used to find a solution. The branch and bound method has been developed to find an integer solution quickly. Three specialized branching strategies, described in Smith and Wren [115], are used. The sum of the continuous variables in the continuous solution is rounded up to the nearest integer and this value is the target number of duties in a schedule that the branch and bound process will try to find.

The search tree in TRACS-II is limited to 500 nodes. One implication of this is that not every solution is necessarily found and, thus, it is possible the best solution available from

[2]See Section 2.2.2 for a discussion of IMPACS system.

the large set of potential duties may not be found. It is also possible that all 500 nodes are exhausted and *no* solution is found. This does not necessarily mean that there is no solution to be found from the set of potential duties. The branch and bound search may have been unlucky in its search. Regardless, the approach to be taken when this does happen is to either build a new set of duties after altering some of the parameters in the labour file, or, use SIEVE to reduce the set of duties further in the hope that branch and bound will be more successful with this subset of the original set of duties. Alternatively, additional constraints could be added to the ILP placing restrictions on the number of various types of duties that are allowed to participate in a solution.

4.2.4 Step 4 - The DISPLAY Process

The DISPLAY process is the simplest of the four. DISPLAY simply reads the output of the SCHEDULE program and translates that into a human readable duty schedule. This includes a list of the driver duties (times, break lengths, total working time, etc) as well as a complete set of running board diagrams upon which the duties are clearly marked.

4.3 Summary

The TRACS-II system consists of four stages. The first stage, BUILD, generates a large set of potential duties from which the duty schedule will be chosen. These potential duties adhere to the labour agreement rules as specified in the labour file; this ensures all the duties generated are legal for the problem in question.

The second stage involves reducing the size of the potential duties if it is felt they are too large. A program known as SIEVE exists which can do this. At this point it is also possible that more than one set of potential duties has been generated and these sets can be merged together into one set with the MERGE program.

The third step, SCHEDULE, is where the actual duty schedule is generated. SCHEDULE

uses a set covering model to accomplish this. The integer constraints on the set covering ILP are first relaxed and a continuous solution to the LP is found. Next, if selected, a process known as REDUCE is invoked which eliminates unused relief opportunities and unused duties (in the sense they are not used in the continuous solution) from the continuous solution before entering the branch and bound process.

The final stage is known as DISPLAY. DISPLAY simply converts the duty schedule generated by SCHEDULE into a easy to read format for the user.

4.4 Use of TRACS-II

TRACS-II has been used for solving both bus and rail driver scheduling problems. Consultancy work has been carried out consisting of driver scheduling exercises for various train operators throughout the UK with some success. Investigations, detailed by Kwan *et al.* [71], into producing train driver schedules under the different operating conditions, in collaboration with several UK rail operators, found in the UK have been carried out using TRACS-II. TRACS-II produced efficient driver schedules in every case.

TRACS-II has been used for scheduling bus drivers. Wren and Gualda [137] describe the combined use of TRACS-II and BOOST (a bus scheduling system as described by Kwan and Rahin [74]) used in the scheduling of buses and drivers in Brazil with an improvement in efficiency.

Grampian Computers Ltd. had an agreement with the University of Leeds regarding the marketing of TRACS-II. A consultancy exercise with Reading Buses using both TRACS-II and the bus scheduling system BOOST, known as the *Openbus* system from Grampian Computers Ltd., demonstrated that TRACS-II was able to produce more efficient driver schedules than were previously used at Reading Buses. The exercise and experience in installing the system at Reading Buses is described by Wren and Kwan [138].

Kwan *et al.* [73] describe an exercise, for potential suppliers of computer-aided scheduling

systems, carried out for FirstGroup based in Norwich UK. The schedules worked upon were large and difficult in the sense they had unusual constraints. As well as producing driver schedules using FirstGroup's current operating constraints TRACS-II was applied to three 'what if' scenarios. TRACS-II proved it was flexible enough to produce the driver schedules required (better than the previously produced ones where comparisons can be done) and work has now started on the installation of TRACS-II throughout the 26 companies of FirstGroup.

4.4.1 Limitations

Current limitations on the TRACS-II system involve the size of problems that can be solved. As would be expected BUILD generates many more duties for larger problems, especially if they involve three or four part duties. To combat this the duty constraints in the labour file generally have to be tightened such that fewer duties are generated. The current limit for the number of potential duties to be used by SCHEDULE is 100,000. Large problems that use three or four part duties can easily exceed this if the parameters in the labour file are not tightened.

Fine tuning a labour file to generate both a reasonable number of duties as well as a good variety of duties can be difficult and time consuming for large problems. Relief opportunity selection would help to reduce the size of the problem as the removal of relief opportunities would mean that fewer duties will be generated by BUILD.

4.5 Conclusions

TRACS-II has been demonstrated to be a flexible driver scheduling system that can be used in both the bus and rail industry with some success. This flexibility derives from the fact that the operator dependent constraints are dealt with in both the BUILD program and the labour agreement file. If TRACS-II were to be used in a different operating

environment only the first step in the TRACS-II process might[3] need to be modified.

The number of relief opportunities used in a set of bus workings has a direct influence on the number of potential duties generated in the BUILD process. Relief opportunity selection would help to reduce number of duties generated in BUILD and, thus, reduce the size of large problem.

[3]Only if the environment was drastically different, or if some new scheduling rule was introduced which was very different from previous ones.

Chapter 5

Mealbreak Chains

5.1 Introduction

Throughout a duty schedule drivers need to take a breaks between stretches. After such a break the driver is again assigned to work on another vehicle. Often this involves taking over a bus from another driver who then goes on a break. The chaining of such breaks such that one driver starting work after a break relieves another driver who is about to start a break is referred to as a *mealbreak chain*.

The hypothesis used by the relief opportunity selection method developed in this thesis is that mealbreak chains hold a key role in defining a *good* schedule. A good schedule, in this context, is one that uses as few drivers as possible.

This chapter contains a discussion of mealbreak chains. This will include the discussion of topics such as:

1. Properties of mealbreak chains.
2. Peak periods of the schedule.
3. Algorithms to construct mealbreak chains in a bus schedule.

5.2 Properties of Mealbreak Chains

Mealbreak chains are a crucial feature that are present in good schedules. To see how mealbreak chains can improve a duty schedule consider the partial running boards shown in Figure 5.1.

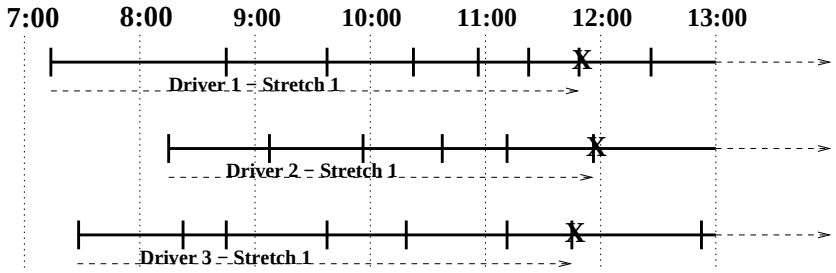

Figure 5.1: No Mealbreak Chain Example

Each relief opportunity in Figure 5.1 marked with an "X" represents the latest possible time a driver can work on the vehicle starting from the beginning. In this example assume the following parameters:

- MAX SPELL LENGTH is set to five hours.
- MINIMUM MEALBREAK LENGTH is set to forty minutes.

All three drivers represented on the vehicles in Figure 5.1 have their latest possible handover times within twenty minutes of each other. This means that if they all work up to the marked time then three new drivers will have to take over their vehicles in order for the original three drivers to take their mealbreaks. This is the worst possible case as one of the most important objectives in the driver scheduling problem is trying to minimize the number of drivers used.

A mealbreak chain will create at most *one additional* driver in the schedule. An example of this can be seen in Figure 5.2 where a sensible mealbreak chain has been formed. After the duty worked on by driver 1 finishes a mealbreak, work is continued on bus 2. This coincides with the time that driver 2 starts a mealbreak. Driver 2's mealbreak finishes at

the same time that driver 3 starts a mealbreak and driver 2, thus, relieves driver 3 on bus 3. The drivers' mealbreaks are chained together and the only new driver required is on bus 1 when driver 1 starts a mealbreak. The mealbreak chain used creates one additional driver instead of three and carries on past noon. If bus 1 in Figure 5.2 were short and ended at the same time that driver 1 takes a mealbreak (in this example) then it would be the case that *no* additional drivers would be required.

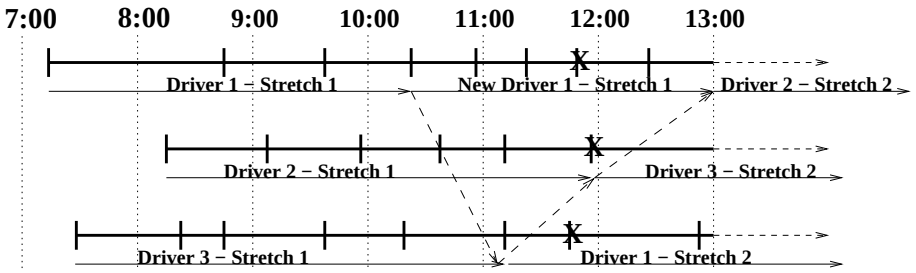

Figure 5.2: Mealbreak Chain Example

On the surface it would seem that it is more efficient to have stretches as long as possible. In practice, as seen with the mealbreak chain example in Figure 5.2, the way in which duties fit together to form mealbreak chains plays a crucial role in reducing the number of duties used in a driver schedule. When drivers take their mealbreaks it is more efficient to relieve them with drivers finishing their mealbreaks rather than starting a new driver. From this it can be deduced that a driver schedule using the minimum number of drivers must have mealbreak chains formed in the schedule in order to limit the number of drivers introduced in the schedule [38].

5.3 Mealbreak Chains and Peak Periods

Some bus workings are *peaked*. That is they have more vehicles on the road during the morning and evening rush hour periods than at any other time during the day. This feature is more common amongst weekday bus workings rather than weekend bus workings.

A *peak* in a schedule is defined as the period of time that has the most vehicles in operation simultaneously. A *peak vehicle* can be defined as a running board that is contained wholly

within the morning (or evening) bus workings and part of it falls entirely within the peak.

Mealbreak chains should be avoided during peak periods in a set of bus workings. If mealbreak chains were to go through the peak period then it is likely that extra drivers will be added to the schedule; it is also possible that there may be more drivers available than there are running boards to work on after the peak.

Although peak periods can occur in both the morning and evening periods in the bus schedule there are some important differences between them; specifically there are differences in the times, with respect to the start and finish of a running board, they occur. The morning peak usually takes place soon after the buses have started the day. In contrast, the evening peak generally occurs several hours before the buses terminate late in the evening.

5.4 Mealbreak Chain Generation – Previous Work

Fores [38] investigated mealbreak chains and methods that can be used to generate them. Fores identified and described three methods for finding mealbreak chains in a bus schedule. These methods are:

1. Assignment Problem.
2. Network Programming.
3. Mathematical Programming.

The methods themselves and the observations made are from Fores's work with mealbreak chains [38].

Before the methods themselves will be described a sample morning schedule will be introduced which will be used as an example in the first two method descriptions. Consider the partial schedule shown in Figure 5.3. For this schedule the following parameter values

will be used:

- MINIMUM SPELL LENGTH is set to 2 hours.
- MAXIMUM SPELL LENGTH is set to $4\frac{1}{2}$ hours.
- MINIMUM MEALBREAK LENGTH is set to 30 minutes.
- MAXIMUM MEALBREAK LENGTH is set to 1 hour 15 minutes.

Legal relief opportunities have been marked with an 'X' in Figure 5.3. The relief opportunities on each vehicle have also been given a letter for identification purposes. The legal relief opportunities are thus:

- Bus 1 : 1A-8:45, 1B-9:45.
- Bus 2 : 2A-9:15, 2B-10:00, 2C-10:45.
- Bus 3 : 3A-8:30, 3B-9:45.
- Bus 4 : 4A-9:45 (end of the vehicle).
- Bus 5 : 5A-9:00, 5B-9:45, 5C-10:45.

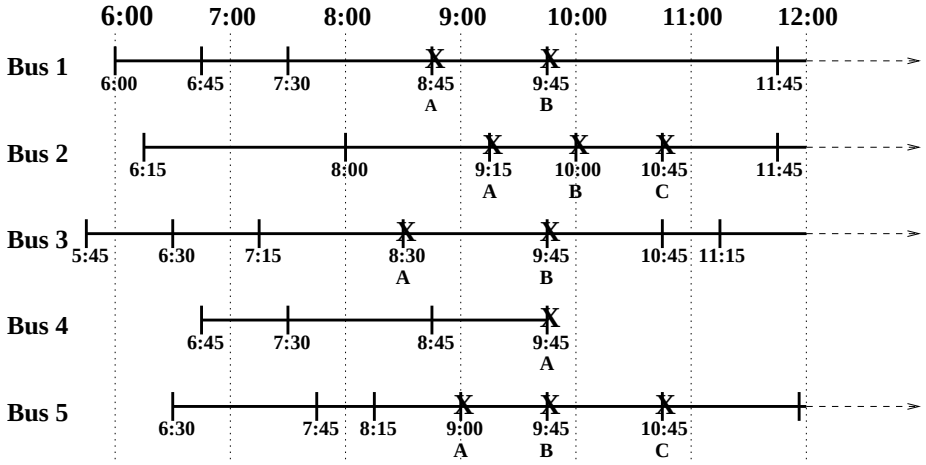

Figure 5.3: Morning Bus Workings Example

A matrix of potential mealbreak links can next be constructed. It will be a n by m matrix where n is the number of relief opportunities that can be the legal start of a mealbreak link

and m is the number of relief opportunities that can be used legally to end a mealbreak link. Each entry, (i, j), will represent the cost of the mealbreak from relief opportunity i to j. The cost is measured in minutes above the minimum mealbreak length used. If it is illegal to have a mealbreak link between relief opportunities i and j a '–' will be present in the table. Such a matrix has been constructed for the bus workings in Figure 5.3 which is shown in Table 5.1. The problem, as formulated by Fores [38], is that of minimizing the total cost of chaining the maximum number of mealbreak links.

	1B	2A	2B	2C	3B	5A	5B	5C
1A	–	***0***	45	–	30	–	30	–
1B	–	–	–	30	–	–	–	30
2A	0	–	–	–	***0***	–	0	–
2B	–	–	–	–	–	–	–	15
3A	45	15	–	–	–	0	45	–
3B	–	–	–	30	–	–	–	***30***
4A	–	–	–	30	–	–	–	–
5A	15	–	30	–	15	–	–	–
5B	–	–	–	30	–	–	–	–

Table 5.1: Matrix of Potential Mealbreak Links

Given the size of Table 5.1 an optimal solution can be determined by examining all possible links. One solution is:

```
1A->2A->3B->5C
3 Links.
30 Minutes idle time.
```

There is a total of three optimal solutions containing 3 links and having a cost of 30 minutes. This would also leave the work on bus 4 to be covered by a split duty as the duty covering this work will not participate in the optimal mealbreak chain solution.

The general solution methods for finding mealbreak chains will be discussed next.

5.4.1 Assignment Method

The goal of the assignment method is to assign as many second stretches of work as possible to drivers who have already completed their mealbreak. For example, in the previous problem there are 9 potential relief opportunities where drivers can start their mealbreaks and 9 potential relief opportunities where drivers can finish their mealbreaks (start a second stretch of work).

Given the cost matrix found in Figure 5.1 a matching algorithm, such as the Hungarian algorithm[1], can be applied. All illegal links can be given some artificially high value to discourage their selection. Fores observed that with using a matching algorithm to solve this problem one relief opportunity would be assigned to one and only one driver; it is therefore possible that some drivers will have to be given invalid links and these would have to be removed from the solution (for example, since the optimal solution for our previous problem only used three links there would have had to have been some illegal links formed if an assignment method was used).

Unfortunately, one problem with this approach is that the rows and columns in the mealbreak link matrix are not independent. For example if relief opportunities 1A and 2A are used (forming the Link `1A->2A` from Table 5.1) then no other relief opportunities present in the assignment problem on buses one and two should be considered any further because the work left between the relief opportunities is too short. As rows and columns are eliminated from the cost matrix, dependent rows and columns will have to be deleted. To complicate things further, the assignment of a relief opportunity in a particular row (or column) can affect other columns (or rows) that represent alternative relief opportunities for that bus. A version of the Hungarian algorithm modified to take these operations into account would be fairly complicated to implement.

Fores identifies a possible way to overcome this drawback. If each vehicle fixes a relief opportunity then the rows and columns are no longer dependent on one another as before.

[1] Which can be found in any reasonable operational research or management science textbook such as [131, 126].

This would mean that a standard matching algorithm could be used. Unfortunately this is a costly way of solving the problem. With the small example in Figure 5.3 there are 36 possible (assuming each vehicle has one relief opportunity fixed) combinations of relief opportunity selections that could be put through a matching algorithm. Each one of the solutions found with the matching algorithm would be a local optimum; this is due to the fact that no effort is made to see how the duties created around the mealbreak chains "fit-together" elsewhere in the schedule. Fores also mentions that swapping relief opportunities and solving each stage with a tree search could work. Heuristics would be needed to decide at which relief opportunity branching should occur. Penalties could be placed on links that are very costly to help avoid choosing them. Unfortunately for a large problem, this could be difficult due to the fact that penalty costs for various relief opportunities are low in comparison to the cost of an actual solution and this would still not take into account the schedule as a whole, only local parts of the schedule [38].

5.4.2 Network Programming Method

Fores also considered solving the problem graphically using a network. To avoid using more than one relief opportunity on a bus to start a second stretch of work Fores introduced one sink node per bus. The beginnings of mealbreaks are represented as source nodes with only one source node per bus. A network flow diagram for the previous example can be found in Figure 5.4.

Each link is given a lower bound of 0 and an upper bound of 1 which indicates whether a given link is used or not. Each link then has its cost assigned to it from an associated cost matrix, such as what is found in Table 5.1.

Unfortunately, this model still suffers from some of the deficiencies found in the assignment approach discussed previously. It is still possible for different relief opportunities on the same bus to be used as the beginning and end of mealbreaks. There are no constraints that would ban certain sink nodes from being used if certain source nodes are used and vice versa.

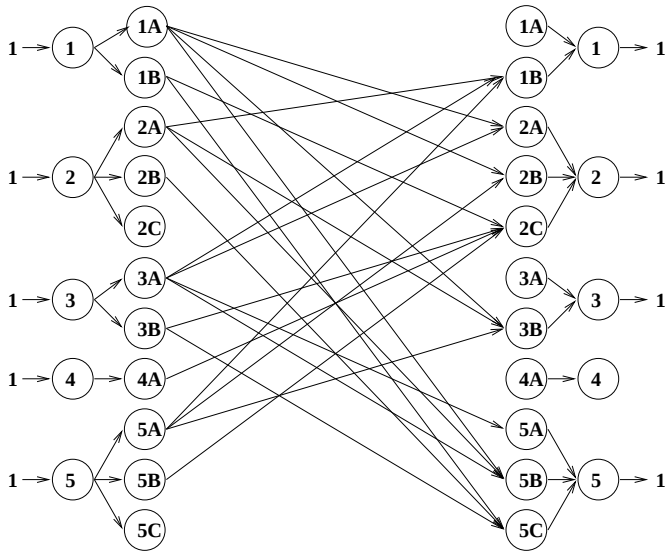

Figure 5.4: Mealbreak Chain Problem Network Flow Diagram

Fores suggests that Lagrangian Relaxation[2] techniques could be used to encompass such constraints. These additional constraints could be modelled in an ILP and used as a penalty on the objective costs if these constraints are violated.

5.4.3 Mathematical Programming Method

Mealbreak chain creation can also be modelled as a mathematical program. Fores investigated this and gives a concrete example. A generalised version of the mathematical programming approach is presented here. Fores split the problem into two separate subproblems.

The first part of the mathematical programming method finds the maximum number of links that can be used forming mealbreak chains. This can be described with the following ILP problem:

$$\text{Maximize} \quad \sum_{i=1}^{n} \sum_{j=1}^{m} u_{ij} l_{ij} \tag{5.1}$$

[2]A good discussion of lagrangian relaxation techniques applied to integer programming problems can be found in an article by Fisher [37].

where

$$u_{ij} = \begin{cases} 0 & \text{if } (i,j) \text{ is an invalid mealbreak link} \\ 1 & \text{otherwise} \end{cases}$$

$$l_{ij} = \begin{cases} 1 & \text{if mealbreak link } (i,j) \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

The variables l_{ij} represent whether or not the mealbreak link between relief opportunities i and j is used. The index i can take values in the range $1, \ldots, n$ and j can have values in the range $1, \ldots, m$; the value n represents the number of legal relief opportunities that can be used as the start of a mealbreak whilst m represents the number of legal relief opportunities that can be used to finish a mealbreak link. The u_{ij} parameter bans invalid links from occurring in a solution.

Several sets of constraints need to be included to ensure the solution is legal. The first set of constraints ensures that a driver only relieves a maximum of one other driver and that a driver is relieved only by another driver who is legally able to do so. These constraints can be expressed as:

$$\sum_{i=1}^{n} l_{ij} \leq 1 \quad \text{for all } j = 1, \ldots, n \tag{5.2}$$

$$\sum_{j=1}^{m} l_{ij} \leq 1 \quad \text{for all } i = 1, \ldots, m \tag{5.3}$$

Conveniently, this turns out to be the same as making sure that only one entry per row or column in the cost matrix is used. Once a relief opportunity has been chosen for a particular bus, the other relief opportunities on the same bus need to be removed as possible handover points. The next set of constraints tackle this by ensuring that only one relief opportunity per bus can have the value of one. Expressed generally these constraints would look like:

$$\sum_{j=k_{n1}}^{k_{n2}} \sum_{i=1}^{n} l_{ij} \leq 1 \quad \text{for all buses } 1, \ldots, k \tag{5.4}$$

$$\sum_{i=p_{n1}}^{p_{n2}} \sum_{j=1}^{m} l_{ij} \quad \leq \quad 1 \quad \text{for all buses } 1, \ldots, p \tag{5.5}$$

where k_{n1} and k_{n2} are the first and last column indices for the relief opportunities on bus k and p_{n1} and p_{n2} are the first and last row indices for the relief opportunities on bus p. The Constraint (5.4) ensures that no columns representing relief opportunities for the same bus has more than one value of one. Constraint (5.5) performs a similar function for the rows.

One more group of constraints must be added to this ILP. It can be the case that some relief opportunities occur in both the rows and the columns in the matrix. Constraints must be added to ensure that more than one relief opportunity is not used on the same vehicle; every combination of relief opportunities that cannot be used on a vehicle must be eliminated to ensure this cannot happen. For each such relief opportunity, row r, along with the other relief opportunities on the same vehicle that can be used as the end of a mealbreak (say columns, $c_1, c_2, \ldots c_x$) the following constraint is added:

$$\sum_{j=1}^{m} l_{rj} + \sum_{j=1}^{x} \sum_{i=1}^{n} l_{ic_j} \quad \leq \quad 1 \tag{5.6}$$

As the formulation is somewhat complicated an example will be given. Using the sample cost matrix in Table 5.1, consider relief opportunity 2B. This relief opportunity is legal to use as both the beginning and end of a mealbreak link and there are other relief opportunities available on bus 2 that can be used as the finish of a mealbreak link (2A and 2C). Constraint (5.6) would be formulated as follows:

$$\sum_{j=1}^{8} l_{4j} + \sum_{j=1}^{2} \sum_{i=1}^{9} l_{ic_j} \quad \leq \quad 1 \quad \text{where } c_1 = 2 \text{ and } c_2 = 4$$

It should be noted at this point that constraints such as (5.6) may not always need to be implemented. For example if three part duties were being constructed it is possible that two relief opportunities close to one another could be used and such a constraint would not be necessary. This is dependent on the labour agreement. However, this will not be considered in this definition.

The solution to this ILP represents the maximum number of mealbreak links, say z, that can be created with the given set of data. The second subproblem is to revise the mathematical program slightly in order to minimize the cost of the mealbreak links. The new objective function is now:

$$\text{Minimize} \qquad \sum_{i=1}^{n}\sum_{j=1}^{m} Cost_{ij} l_{ij} \tag{5.7}$$

The value $Cost_{ij}$ is the cost of link (i, j) in the cost matrix. Rather than using u_{ij} values to ban certain links, high penalties are assigned to them instead which will discourage their use. All of the previous constraints are incorporated in this second model. However, one more constraint is added. This constraint sets the number of links to be used to z. This constraint is:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} l_{ij} \quad = \quad z \tag{5.8}$$

This method will find the optimum chaining of mealbreaks for a given time interval. However, as can be seen from the equations above, a large problem using this method would have many constraints and the computational expense would be a consideration. This problem becomes more apparent if shorter mealbreak lengths are allowed or complicated labour agreements are used thus potentially creating additional complicated constraints [38].

Fores identifies a few other drawbacks to using this method. For example some decision must be made as to which 'time window' optimum mealbreak chains will be created for. A window too large will make the ILP expensive to solve and a window too small may miss vital mealbreak links. Some mealbreak links created may create illegal duties due to suitable relief opportunities not being available later in the day. Heuristics would have to be developed to try to resolve some of these problems.

5.5 Using Generated Mealbreak Chains

Fores carried out some experiments using the mathematical programming model just described to find optimum mealbreak chains on a schedule at intervals throughout the day. The duties created around these mealbreak chains would, thus, form the duty schedule. In the schedule experimented on by Fores, a driver schedule was created that used 20 duties when the mealbreak chains were selected using the mathematical programming method just outlined. IMPACS [111] found a solution using only 18 duties.

A problem with the mathematical programming generated mealbreak chains was that they included many duties that were fairly short and, thus, inefficient. Since the chaining problem solved here maximizes the number of links used it will often use shorter duties to facilitate longer mealbreak chains. This 'greedy' approach did not give particularly encouraging results. Mealbreak chains hold a key role in the creation of good duty schedules but the selection of these 'optimum' mealbreak chains did not help to solve the problem.

5.6 Summary

It was argued that mealbreak chains are an important part of a good duty schedule. This is due to the fact they help reduce the number of duties required in a schedule. However, mealbreak chains are best avoided during the peak periods of a set of bus workings; if possible duties should try to work past the peak. If mealbreak chains do occur during the peak it may be possible that there are more drivers available than there are running boards to work on after the peak.

The work by Fores on mealbreak chains included three methods in which mealbreak chains could be generated. The mealbreak chains found with these methods, however, concentrated on minimizing the idle time of drivers on their mealbreaks and, in the case of the mathematical programming method, tried to maximize the number of mealbreak links used. Unfortunately the schedules created from using such mealbreak chains were not

satisfactory. However, this does not mean that a good duty schedule cannot be found from a sensible selection of mealbreak chains.

The method described in this thesis on mealbreak chain generation uses the properties of mealbreak chains as its very hypothesis. Rather than trying to find mealbreak chains using the minimal amount of idle time the combination of several good mealbreak chains is used.

Chapter 6

The CROSS CSP

6.1 Introduction

CROSS (Constraint programming Relief Opportunity Selection Scheme) has been designed in order to select relief opportunities, from a set of bus workings, that are likely to be useful in the driver scheduling process. The purpose of this chapter is twofold:

1. Outline the problem to be solved.
2. Outline the constraint programming model used.

CROSS will be referred to in several ways throughout the rest of the thesis. CROSS, on its own, refers to the application of the CROSS method from beginning to finish. The CROSS *algorithm* refers to the CSP search algorithm and the CROSS CSP refers to the constraint programming model itself.

Chapter 7 will examine the algorithm used to search for solutions to the CROSS CSP in detail.

6.1.1 Earlier Work

The CSP model used in CROSS has evolved over time. The original model that was used (and eventually evolved into the CROSS model used today) is described for completeness.

Initially the set of bus workings was split into individual pieces of work (governed by the labour agreement rules). Each individual piece of work had a constrained variable associated with it, say W_i where i is the i^{th} piece of work in the bus workings. The domain of each W_i variable would be the set of work pieces that could legally follow workpiece W_i as well as a value indicating that W_i is the end of a duty. Some inspiration for this original model came from the TRACS heuristic described in Section 2.1.1 [92].

Although this model does encompass all eventualities it was found to be fairly clumsy to work with. No real structure is given to the schedule as the variables are processed and odd short pieces of work could be leftover in between stretches formed part way through a solution. Constraints could be created to discourage this from occurring but constraint propagation was not particularly effective.

The model in this chapter is the end result of the evolution of this original approach and it can be seen that these issues have been addressed.

6.2 Problem Definition and Outline of Approach

The goal of this work was to develop a method of analysing the morning and evening periods of a bus schedule in order to decide which relief opportunities may be useful in duty generation. The procedure that was designed can be viewed as a front end, or a pre-processing stage, in the TRACS-II process. This can be seen conceptually in Figure 6.1.

TRACS-II needs both a labour file and a vehicle file provided in order for the large set of potential duties to be generated by BUILD (as described in Chapter 4). CROSS also requires a labour file and vehicle file but for a different purpose. CROSS examines the

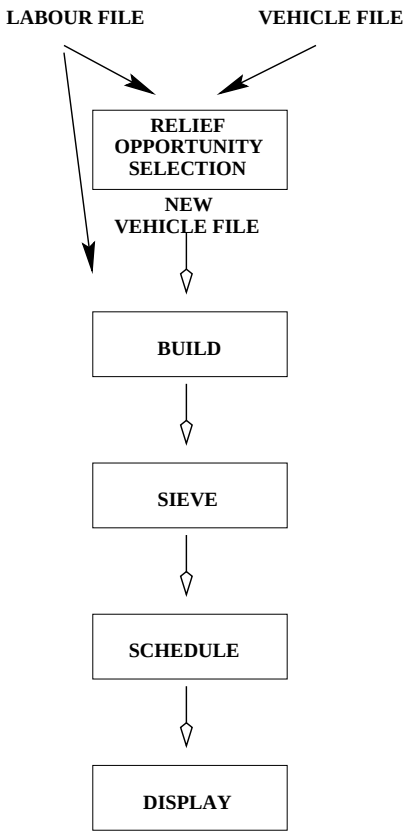

Figure 6.1: TRACS-II with Relief Opportunity Selection

relief opportunities available in the morning or evening period and produces a new vehicle file which contains only the relief opportunities selected by CROSS in the period chosen. This new vehicle file will then be used in the regular TRACS-II process. This has two effects on TRACS-II:

1. It reduces the number of potential duties generated in the BUILD process, thus reducing the number of variables in the resulting ILP.

2. It reduces the number of constraints in the SCHEDULE ILP (due to a reduction in the number of pieces of work to be covered in the bus schedule).

This also has the effect of, in general, reducing the amount of time needed for SCHEDULE to find a solution. The problem that is to be solved is deciding what is the criteria for selecting the relief opportunities to be used.

6.2.1 Hypothesis

As was seen in Chapter 5, mealbreak chains have properties that may be useful to exploit. One of the most important observations made was that *when drivers take their mealbreaks it is more efficient to relieve them with drivers finishing their mealbreaks rather than starting a new driver*. From this observation the hypothesis that is used for the basis of this work is:

> Schedules containing *good mealbreak chains* should require fewer duties than schedules not containing good mealbreak chains.

The issue here is what constitutes a good mealbreak chain. It was seen in Chapter 5 that using what can be thought of as 'optimal' mealbreak chains can have a negative impact on a duty schedule. The mealbreak chains that were generated using Fores's methods were too restrictive; these chains would only allow a few ways of covering the portion of the bus workings to which it was applied. 'Optimal' mealbreak chains do not always lead to satisfactory duty schedules as they are only locally optimal to a small portion of the schedule. Nonetheless the concept of trying to construct mealbreak chains is a useful one. Two additions are needed in order to gain the flexibility to make this a viable approach:

1. Several sets of mealbreak chains should be generated such that the large set of relief opportunities used would enable the scheduling process to have more flexibility towards how the work is to be covered
2. A random element needs to be introduced into the mealbreak chain generation process such that different chains can be created.

CROSS takes these two points into account. Firstly, the CROSS CSP's variable and value ordering heuristics themselves have a random element introduced into them that ensures that each time it is solved it is likely that a different set of relief opportunities (mealbreak chains) will be selected. Due to this if the CROSS CSP is solved several times then a larger

set of relief opportunities will be selected from which TRACS-II can generate a schedule. Solving the CROSS CSP several times to generate several sets of mealbreak chains will enable new mealbreak chains to be implicitly formed from the total set of selected relief opportunities. This added flexibility is important and plays a key role in CROSS.

6.2.2 Schedule Generation using Constraint Programming

The question naturally arises that perhaps generating an entire duty schedule using constraint programming would be a useful approach. This, indeed, has been done as seen earlier with the COBRA system outlined in Section 3.3.1 and others.

Methods such as TRACS-II have an inherent advantage over others. This is due to the separation of the duty generation component from the schedule construction. Recall that with TRACS-II a large set of potential duties is generated initially. This large set is then supplied to the TRACS-II SCHEDULE (ILP) component and a schedule is formed. The SCHEDULE component has no interaction with the generation of duties. All the duty generation issues are addressed in the BUILD component and if a different driver scheduling problem with different labour agreement rules is tackled, only the BUILD component of TRACS-II may need to be modified. The SCHEDULE component remains untouched; the only interaction SCHEDULE has with BUILD is through the large set of duties that BUILD generates. A pure CP approach would not benefit from this separation as the constraints on the duties themselves will be directly connected with the CP model. For different scheduling problems (for example different labour rules) the CP itself may have to undergo serious modification.

Constraint programming has also been applied to solving the associated set covering or set partitioning problems that can be used to model the duty scheduling problem. Curtis, Smith and Wren applied constraint programming to the set partitioning problem and found it worked reasonably well for small schedules but not as well as the pure ILP-approach (such as TRACS-II). Solving the set covering problem using CP techniques was found to not be very practical as it is difficult to model it as a CSP such that useful constraint

propagation can be utilized [23, 22].

6.3 Data Used

The CROSS program uses the same input files that TRACS-II would use on its own. Therefore, like TRACS-II, the input for CROSS consists of a labour file and a vehicle file.

The labour file contains the rules and constraints to be placed on the duties used in a duty schedule. The vehicle file contains most of the information needed in order to formulate a model for mealbreak chain generation. Specifically, the vehicle file consists of:

1. A set of *vehicles* (running boards), $\mathcal{V} = \{V_1, \ldots, V_n\}$
2. Each vehicle, V_i is defined as a set of *relief opportunities* $V_i = \{RO_{(i,1)}, \ldots, RO_{(i,k_{v_i})}\}$ such that $|V_i| = k_{v_i}$.
3. Each relief opportunity consists of a pair[1], (i, j), that represents vehicle i, at time j.

There are two types of relief opportunities, depending on their position on the running board. The first and last relief opportunities on each running board in the morning/evening set of bus workings are referred to as *external relief opportunities*. The relief opportunities in between the external relief opportunities on the same running board are referred to as *internal relief opportunities*.

6.4 The CROSS CSP: Variables and Domains

The CROSS CSP itself was originally devised for the *morning period* of a schedule (and the description of the CROSS CSP reflects this), but can be easily be applied to the evening period of a set of bus workings as will be described in the next chapter. The morning

[1]Relief opportunities are actually defined as triples with the third value representing the relief point (physical location of the stop). The convention used here is to just use the first two values.

period has been defined as all bus workings before noon. What happens after noon is not considered at any point in this process. All relief opportunities until the first one past noon on each bus are considered in the model.

The variables and domains in the CROSS CSP try to model the set of running boards being examined and the relationships between them regarding the structure of mealbreak chains. There are three types of constrained variables used.

6.4.1 Pattern Variables

The first type of constrained variable has to do with the individual buses in the schedule. These are referred to as *Pattern Variables*. The pattern variable for a bus represents which relief opportunities on it are going to be used. Alternatively, this can be viewed as deciding how the vehicle is *partitioned* into pieces of work. The pattern variable for bus i is written as PV_i.

6.4.2 Next Variables

Each individual running board has a set of *relief opportunities*. Each one of these relief opportunities has a *Next* variable associated with it. The *Next* variable for relief opportunity (i, j) (bus i at time j) is written as $Next_{(i,j)}$. The $Next_{(i,j)}$ variable indicates what a driver finishing a piece of work on bus i at time j is going to do next. $Next_{(i,j)}$ could include a relief opportunity representing the finish of a mealbreak started at relief opportunity (i, j) or a value that indicates that the relief opportunity is either not used at all or is the end of a duty.

6.4.3 Prev Variables

Similar to the definition of the $Next_{(i,j)}$ variable, a previous variable is associated with each relief opportunity on each running board. A previous variable for relief opportunity

(i, j) is written as $Prev_{(i,j)}$. The $Prev_{(i,j)}$ variable indicates what a driver starting a piece of work on bus i at time j did previously. This could be a value indicating whether or not this relief opportunity is the start of a duty or a relief opportunity value indicating where a mealbreak started where the relief opportunity (i, j) is the end of that mealbreak or a value indicating that it is not used at all.

6.4.4 PV_i Variable Domains

The example morning schedule that will be used to illustrate the domains of the previously defined types of variables is shown in Figure 6.2.

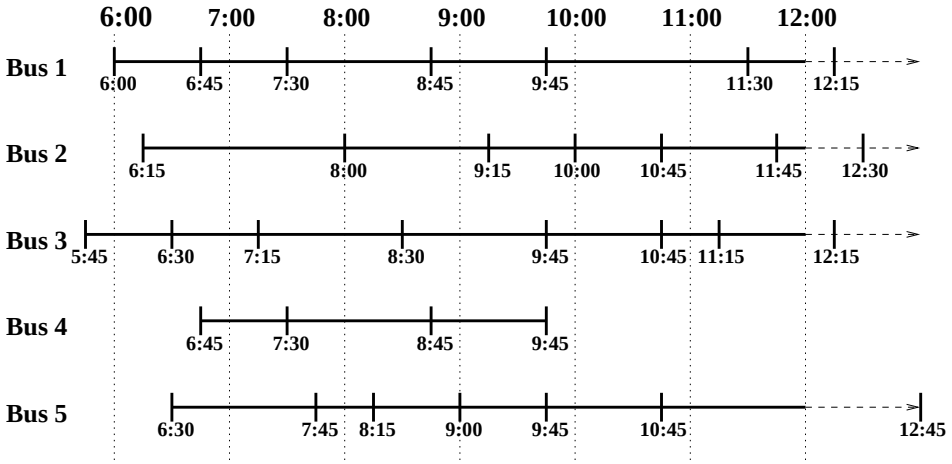

Figure 6.2: Sample Morning Schedule

For the morning schedule represented in Figure 6.2 assume the following parameters:

- MINIMUM SPELL LENGTH is 2 hours.
- MAXIMUM SPELL LENGTH is $4\frac{1}{2}$ hours.
- MINIMUM MEALBREAK LENGTH is 30 minutes.
- MAXIMUM MEALBREAK LENGTH is 1 hour 15 minutes.
- SPLIT DUTIES are allowed.

The domains of the pattern variables in the problem represent all the possible ways in which a vehicle can have its work partitioned. Conceptually each element in a pattern variable's domain is a set of relief opportunities. If the pattern variable is assigned to an element in the domain, the vehicle will be partitioned into pieces of work by the values in that element. Consider the schedule in Figure 6.2; the pattern variable domains for the buses in this schedule are shown in Figure 6.3.

$$
\begin{aligned}
D_{PV_1} &= \{\{0845\}, \{0945\}, \{0845, 1130\}\} \\
D_{PV_2} &= \{\{0915\}, \{1000\}, \{1045\}, \{0915, 1145\}\} \\
D_{PV_3} &= \{\{0830\}, \{0945\}, \{0830, 1045\}, \{0830, 1115\}\} \\
D_{PV_4} &= \{\{-1\}\} \\
D_{PV_5} &= \{\{0900\}, \{0945\}, \{1045\}\}
\end{aligned}
$$

Figure 6.3: Pattern Variable Domains for Sample Morning Schedule

Pattern variable PV_1 has four elements in its domain. The first value, $\{0845\}$, represents partitioning the bus into two pieces of work, the first of which is from 0600 till 0845 and the second piece of work starts at 0845 and carries on until some time past noon. The last value in the domain D_{PV_1} partitions the vehicle into three pieces of work which are 0600-0845, 0845-1130 and 1130 until some time past noon.

The maximum cardinality of a relief opportunity element in the domain of a pattern variable has been set to two. This is done for two reasons.

1. Almost all bus schedules examined used at most two relief opportunities on a single vehicle before noon.

2. This helps to reduce the combinatorial complexity of the problem by avoiding the use of many relief opportunities on a single vehicle.

It should also be observed that as several solutions to the CROSS CSP will be generated it is likely that more than two relief opportunities before noon will be selected.

D_{PV_4} is different from the other pattern variable domains. It consists of a set with the

value '-1'. The '-1' represents a vehicle that is not partitioned. That is, the entire vehicle (or the morning portion thereof) will be treated as a single piece of work.

As a general rule of thumb, any vehicle that is entirely contained in the morning period will have it's pattern variable bound to the value -1. In almost all schedules examined short vehicles such as these are rarely subdivided into workpieces; generally the entire vehicle is worked on by a single driver.

6.4.5 $Next_{(i,j)}$ Variable Domains

The domain of a $Next_{(i,j)}$ variable can consist of several types of values. These values model what a driver does *after* relief opportunity (i, j). The exception to this rule is if the relief opportunity represented by $Next_{(i,j)}$ is not used in the solution at all. The domain for $Next_{(i,j)}$ variables is outlined in Figure 6.4.

$$D_{Next_{(i,j)}} = \begin{cases} 0 & \text{This relief time is not used or a duty ends at this point} \\ RO_{(l,m)} \subset \mathcal{V} & \text{The start of the next spell in the current duty following a mealbreak.} \\ 1201 & \text{Dummy time indicating that the mealbreak after this spell finishes after 1200} \\ 1500 & \text{Dummy time indicating a split duty.} \end{cases}$$

Figure 6.4: $Next_{(i,j)}$ Domain Values

$Next_{(i,j)}$ is only assigned a value of 0 in two cases; either it is not used at all or relief opportunity (i, j) represents the end of a duty (nothing can happen after this point with the current duty). A value representing a relief opportunity, say $RO_{(l,m)}$, is assigned to $Next_{(i,j)}$ if relief opportunity (i, j) is the end of a first spell of work for a duty and $RO_{(l,m)}$ is the relief opportunity where this duty finishes its mealbreak and continues on a new piece of work. Two special values that can occur in the $Next_{(i,j)}$ domains are: 1201, which represents a mealbreak that goes past 1200 noon (since what happens after 1200 is not considered), and 1500, which represents a split duty (which will carry on well past 1200). The 1201 dummy value is only found in $Next_{(i,j)}$ domains if relief opportunity

(i, j) is close enough to noon such that a mealbreak starting at relief opportunity (i, j) can go past noon. The 1500 dummy value, representing a split duty, will only be found in the domain of $Next_{(i,j)}$ variables if split duties are used and the relief opportunity (i, j) represents the legal beginning of a mealbreak for a split duty. Figure 6.5 contains the domains for all of the $Next_{(i,j)}$ variables on bus 1 in Figure 6.2.

$$
\begin{aligned}
D_{Next_{(1,600)}} &= \{0\} \\
D_{Next_{(1,645)}} &= \{0\} \\
D_{Next_{(1,730)}} &= \{0\} \\
D_{Next_{(1,845)}} &= \{(2, 0915), (3, 0945), (5, 0945), (2, 1000), 0, 1500\} \\
D_{Next_{(1,945)}} &= \{(2, 1045), (3, 1045), (5, 1045), 0, 1500\} \\
D_{Next_{(1,1130)}} &= \{0, 1201, 1500\} \\
D_{Next_{(1,1215)}} &= \{0\}
\end{aligned}
$$

Figure 6.5: Next Variable Domains for Sample Morning Schedule Bus 1

The first three $Next_{(i,j)}$ variables in Figure 6.5 represent relief opportunities that are too close to the start of the running board to create a legal spell; therefore the first three $Next_{(i,j)}$ variables are assigned the value 0 as they will not be used. The $Next_{(1,1215)}$ variable is also set to 0 because relief opportunities after 1200 are not considered in the CSP. The first time past 1200 is included on each vehicle such that spells going past 1200 can be formed. The rest of the $Next_{(i,j)}$ variables in Figure 6.5 have domains containing relief opportunities that can form a legal mealbreak link with $Next_{(i,j)}$, the 0 value and the split duty or late mealbreak value (if legal).

6.4.6 $Prev_{(i,j)}$ Variable Domains

$Prev_{(i,j)}$ variable domains represent what a duty was doing *before* it starts work at relief opportunity (i, j). The values these domains can contain are found in Figure 6.6.

A $Prev_{(i,j)}$ variable's domain contains the value 0 when relief opportunity (i, j) is not used. A value of '-1' represents that the relief opportunity (i, j) is the start of a duty. The relief opportunities in the $Prev_{(i,j)}$ variables' domains are the relief opportunities at which a duty may potentially start a mealbreak that terminates, legally, at relief opportunity (i, j).

$$D_{Prev_{(i,j)}} = \begin{cases} 0 & \text{This relief time is not used.} \\ -1 & \text{This is the start of a duty.} \\ RO_{(l,m)} \subset \mathcal{V} & \text{The start of a mealbreak ending at relief opportunity } (i,j). \end{cases}$$

Figure 6.6: $Prev_{(i,j)}$ Domain Values

Figure 6.7 contains the domains for all of the $Prev_{(i,j)}$ variables on bus 1 in Figure 6.2.

$$\begin{aligned} D_{Prev_{(1,600)}} &= \{-1\} \\ D_{Prev_{(1,645)}} &= \{0\} \\ D_{Prev_{(1,730)}} &= \{0\} \\ D_{Prev_{(1,845)}} &= \{0, -1\} \\ D_{Prev_{(1,945)}} &= \{(3,0830), (5,0900), (2,0915), 0, -1\} \\ D_{Prev_{(1,1130)}} &= \{(2,1045), (3,1045), (5,1045), 0, -1\} \\ D_{Prev_{(1,1215)}} &= \{0\} \end{aligned}$$

Figure 6.7: Prev Variable Domains for Sample Morning Schedule Bus 1

The first three $Prev_{(i,j)}$ variables in Figure 6.7 have their values fixed automatically in this problem. The first one at relief opportunity (1,0600) is set to represent the start of a duty since it is impossible for a driver to take over at this relief opportunity following a mealbreak. The next two, relief opportunities (1,0645) and (1,0730), have their associated $Prev_{(i,j)}$ variables set to 0 as they cannot be used due to the fact they would not form a legal spell of work from the beginning of the running board. $Prev_{(1,1215)}$, like its $Next_{(1,1215)}$ counterpart, is set to 0 as it will not participate in the solution. The rest of the domains reflect the possible legal values that are available to them.

6.5 The CROSS CSP: Operational Constraints

The bus driver scheduling problem has many operational constraints. The TRACS-II labour agreement file contains many constraints taken into account as 'duties' are formed in CROSS. These are:

- Minimum/Maximum spell length.
- Sign on and sign off times.
- Travel time to canteen during the mealbreak time.
- Minimum/Maximum mealbreak length.
- Minimum/Maximum length of 1st and 2nd stretches.
- Earliest start on bus.
- Earliest start of a mealbreak.
- Minimum/Maximum cost of duty.
- Maximum spreadover.

Many of these constraints can be taken into account when the variable domains are created. For example, the domains of the PV_i, $Next_{(i,j)}$, and $Prev_{(i,j)}$ variables can be guaranteed to contain only legal values initially (such that $D_{Next_{(i,j)}}$ will not contain a relief opportunity that would form an illegal mealbreak with relief opportunity (i, j)).

It should be noted that the variables essentially trace out two part and the first half of split duties in the bus workings. These are not of great interest as the goal of the CROSS system is to select relief opportunities that may be useful, not to generate duties. However, it is worth noting that only two part and split duties are created. The two part duties, in bus driving terminology, can be thought of as two part early or middle duties; both will be referred to as *regular duties* as the distinction is not important here. *Split* duties, formed during the CROSS process, consist of only one part whilst the second part is assumed to take place sometime after 1200 and is not considered any further.

6.6 The CROSS CSP: Model Constraints

A tool like ILOG Solver is useful if the constraints involved are such that constraint propagation will quickly reduce the size of the solution space to be searched as well as preventing the construction of illegal solutions. The CROSS model takes advantage of this by posting constraints to Solver immediately after the variable domain construction phase as well as while searching for solutions, when variables are bound or domains are altered.

Many of the constraints posted are conditional ones. That is, certain criteria have to be satisfied before the constraint is enforced. ILOG Solver provides functionality to support this in a convenient way as will be seen in the following subsections.

6.6.1 Constraints Involving $Next_{(i,j)}$ and $Prev_{(i,j)}$ Variables

One important fundamental constraint between the $Next_{(i,j)}$ and $Prev_{(i,j)}$ variables in the CSP occurs when a $Next_{(i,j)}$ variable is assigned a mealbreak link with another relief opportunity, say (l, m). For any two relief opportunities (say (i, j) and (l, m) where $j < m$ and $i \neq l$) used in a legal mealbreak link, the following constraint must always hold:

$$Next_{(i,j)} = (l, m) \quad \Rightarrow \quad Prev_{(l,m)} = (i, j) \tag{6.1}$$

Conditional constraints can be posted to ILOG Solver via the `IlcIfThen()` function. The function call would look something like this:

```
IlcIfThen(Next == p, Prev == n);
```

where `Next` represents $Next_{(i,j)}$, `Prev` represents $Prev_{(l,m)}$, `p` represents (l, m) and `n` represents (i, j). This corresponds exactly to Constraint (6.1). This constraint is posted directly to Solver for each possible legal mealbreak link[2].

[2] As an aside, ILOG Solver provides a `IlcInverse()` constraint that is very similar to this. Unfortunately it is not as powerful as the `IlcIfThen()` representation of Constraint (6.3) and Constraint (6.4) is not enforced.

The relationship between $Next_{(i,j)}$ and $Prev_{(l,m)}$ is symmetric in that:

$$Prev_{(l,m)} = (i,j) \quad \Rightarrow \quad Next_{(i,j)} = (l,m) \tag{6.2}$$

is also true. Thus the constraint can be written as:

$$Next_{(i,j)} = (l,m) \quad \Longleftrightarrow \quad Prev_{(l,m)} = (i,j) \tag{6.3}$$

When a $Next_{(i,j)}$ variable is bound to a relief opportunity, say (l,m), the value (l,m) must be taken out of the domains of all other $Next$ variables that contain it. Similarly, when a $Prev_{(l,m)}$ variable is bound to a relief opportunity, say (i,j), then (i,j) also must be removed from the domains of all other $Prev$ variables that contain it. Constraint (6.3) ensures that this happens when $Next_{(i,j)}$ and $Prev_{(l,m)}$ variables are bound. This is automatically carried out by ILOG Solver via constraint propagation. What effectively happens when Constraint (6.3) is posted is that the CSP behaves as if the following constraint is constructed:

$$Next_{(i,j)} \neq (l,m) \quad \Longleftrightarrow \quad Prev_{(l,m)} \neq (i,j) \tag{6.4}$$

Another conditional constraint in the CSP is that if a $Prev_{(i,j)}$ variable is bound to the value '0' then $Next_{(i,j)}$ must also have the value '0' as it cannot be used. There is only one case where this rule does not hold and that is where (i,j) is a relief opportunity at the end of a running board and $j < 1200$ (as it may be possible for a driver to take a break at this point to start a second part of a two part duty elsewhere). The following conditional constraint is posted to ILOG Solver on all relevant $Prev_{(i,j)}$ variables along with the associated $Next_{(i,j)}$ variables:

$$Prev_{(i,j)} = 0 \quad \Rightarrow \quad Next_{(i,j)} = 0 \tag{6.5}$$

Note that Constraint (6.5) is not symmetric because a $Next_{(i,j)}$ variable can also have a value of '0' if it represents the end of a duty.

All of these constraints are posted to Solver after the domains have been constructed. They are fundamental as much of the constraint propagation that takes places during the search for solutions involves them.

6.6.2 Constraints Involving PV_i Variables

There are many conditional constraints that must be enforced between PV_i variables and the $Next_{(i,j)}$ and $Prev_{(i,j)}$ variables in the CROSS CSP. When a PV_i variable is bound it has an immediate impact on the domains of the $Next$ and $Prev$ variables associated with the relief opportunities on running board i.

To see this, consider a pattern variable, say PV_i, bound to a value, say S. It is clear that for all internal relief opportunities (say there are p_i of them) on running board i any relief opportunity that is not contained in the set S must have its $Next_{(i,j)}$ and $Prev_{(i,j)}$ variables bound to '0' as they cannot be used. This constraint can therefore be written as:

$$(t_{i[k]} \notin S) \wedge (PV_i = S) \quad \Rightarrow \quad (Next_{(i,t_{i[k]})} = 0) \wedge (Prev_{(i,t_{i[k]})} = 0) \tag{6.6}$$
$$\text{where} \qquad k = 1, \ldots, p_i \ \text{ and } \ \forall S \in D_{PV_i}$$

where $t_{i[k]}$ represents the time of the k^{th} internal relief opportunity on the running board i and p_i represents the number of internal relief opportunities on running board i. Due to constraint (6.5) and constraint propagation the above constraint can be simplified to:

$$(t_{i[k]} \notin S) \wedge (PV_i = S) \quad \Rightarrow \quad Prev_{(i,t_{i[k]})} = 0 \tag{6.7}$$
$$\text{where} \qquad k = 1, \ldots, p_i \ \text{ and } \ \forall S \in D_{PV_i}$$

Additionally, if the time $t_{i[k]}$ is present in S then $Prev_{(i,t_{i[k]})} \neq 0$. The $Prev_{(i,t_{i[k]})}$ variable, in this context, must take either the value of a relief opportunity (representing the start of a mealbreak link) or the value -1 (representing the beginning of a duty). This is stated

as follows:

$$(t_{i[k]} \in S) \wedge (PV_i = S) \quad \Rightarrow \quad Prev_{(i,t_{i[k]})} \neq 0 \tag{6.8}$$
$$\text{where} \qquad k = 1, \ldots, p_i \quad \text{and} \quad \forall S \in D_{PV_i}$$

It should be noted that it is not necessarily the case that if $(t_{i[k]} \in S) \wedge (PV_i = S)$ that $Next_{(i,t_{i[k]})} \neq 0$. This is because $Next_{(i,t_{i[k]})}$ might represent the end of a driver's duty.

Additionally, since a bound PV_i partitions the vehicle into spells, we can post two more constraints for each such spell. Assuming a spell starts at (i, A) and ends at (i, B) they are as follows:

$$Prev_{(i,A)} = -1 \iff Next_{(i,B)} \neq 0 \tag{6.9}$$
$$Prev_{(i,A)} \neq -1 \iff Next_{(i,B)} = 0 \tag{6.10}$$

This ensures that if $Prev_{(i,A)}$ represents the start of a duty then at the end of the first stretch it will either form a mealbreak link, represent a split duty or have a mealbreak going past noon (Constraint (6.9)). Similarly, if $Prev_{(i,A)}$ is not the beginning of a new duty then relief opportunity (i, B) will have to represent the end of a duty (Constraint (6.10)). As the relief opportunities occurring past noon only have a legal $Next$ domain value of 0 these constraints are not applied to them.

In practice the previous four constraints can be enforced by invoking a routine when PV_i is bound and performing the appropriate checks in order to enforce a unary constraint on the appropriate $Prev$ variables (Constraints (6.7) and (6.8)) or post the appropriate conditional constraints (Constraints 6.9) and (6.10)).

Constraint (6.7) has an effect on other $Next$ and $Prev$ variables in the problem. If relief opportunity (i, j) is not used on running board i because $j \notin S$ then no $Next$ or $Prev$ variables in the model can be assigned the value (i, j). ILOG Solver maintains arc consistency between $Next$ and $Prev$ variables due to Constraint (6.3) which ensures such values are removed from their domains.

When the pattern variable domains are first created, the only constraints taken into account are the minimum and maximum allowed spell lengths. After the pattern variable domains have been generated, a further check is carried out to ensure that the three types of *stretches* that these spells can represent are legal. If a spell cannot represent a legal stretch then constraints must be constructed in order to prevent an illegal stretch from occurring. A spell can be a part of three types of stretch in the CROSS algorithm. The three types of stretches considered are:

1. First stretch of a two part regular duty.
2. Second stretch of a two part regular duty.
3. First stretch of a split duty.

To add stretch constraints to the CSP all possible spells, in conjunction with the pattern variable domain elements which form them, must be considered. Each stretch has to include its own sign on/sign off time as well as its own minimum/maximum length to be legal. If any of the three types of stretches are found to be illegal with the spell being examined then constraints must be placed on the $Next$, $Prev$ and PV_i variables involved in the stretch.

Consider the spell belonging to running board x shown in Figure 6.8. Assume that the spell shown in Figure 6.8 has been created due to the work on the vehicle being partitioned by $S \in D_{PV_x}$. Each pattern variable domain element creates at least one spell on the running board; the general case is being considered here. The spell in Figure 6.8 starts at relief

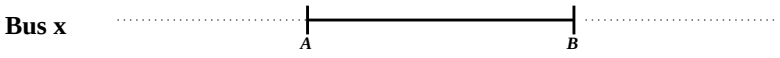

Figure 6.8: Spell on Running Board x

opportunity (x, A) and finishes at relief opportunity (x, B). Each spell must be examined to ensure it forms a legal first stretch of a duty. If not, the following constraints are posted

to Solver:

$$Prev_{(x,A)} = -1 \Rightarrow PV_x \neq S \tag{6.11}$$

$$PV_x = S \Rightarrow Prev_{(x,A)} \neq -1 \tag{6.12}$$

$$PV_x = S \Rightarrow Next_{(x,B)} = 0 \tag{6.13}$$

These constraints ensure that this spell will not be used as a first stretch of a duty. The first constraint ensures that if spell A is the beginning of a duty the domain of PV_x cannot contain S. The second and third constraints ensure that $Prev_{(x,A)}$ cannot be the beginning of a duty and, in fact, must be the second part of a two part duty if $PV_x = S$.

Note that the following is not necessarily true:

$$PV_x \neq S \Rightarrow Prev_{(x,A)} = -1$$

$$Prev_{(x,A)} \neq -1 \Rightarrow PV_x = S$$

It could be the case that relief opportunity (x, A) does not have to be the start of a new duty if PV_x happens to be bound to a value other than S. Similarly if $Prev_{(x,A)}$ is the start of a new duty it may be the case that $PV_x \neq S$.

Every stretch is also examined to ensure it can represent a legal second stretch of a duty. If this stretch is deemed illegal then the following constraints are posted to Solver:

$$Prev_{(x,A)} \neq -1 \Rightarrow PV_x \neq S \tag{6.14}$$

$$PV_x = S \Rightarrow Prev_{(x,A)} = -1 \tag{6.15}$$

$$PV_x = S \Rightarrow Next_{(x,B)} \neq 0 \tag{6.16}$$

The first constraint ensures that if $Prev_{(x,A)}$ does not represent the beginning of a new duty then PV_x cannot take the value S, thus ensuring the spell will not form the second stretch of a regular duty. The second and third constraints ensure that if $PV_x = S$ then $Prev_{(x,A)}$ must represent the start of a new duty; this ensures the spell cannot be used as

a second stretch of a two part duty therefore 0 is taken out of the domain of $Next_{(x,B)}$. As with the constraints preventing a spell being used as a first stretch in a regular duty, these constraints are not symmetric.

The third, and final, stretch test made in the preprocessing stage of the CROSS algorithm involves treating the spell as the first stretch of a split duty (if split duties are allowed). If the stretch is deemed to be illegal for the first stretch of a split duty then the following constraint is posted to ILOG Solver:

$$PV_x = S \quad \Rightarrow \quad Next_{(x,B)} \neq 1500 \tag{6.17}$$

It should be noted at this point that if the first two stretch tests show that the spell in question cannot be a legal part of a duty then S is taken out of the domain of PV_x. The parameter for the minimum split shift length is almost always the same or longer than the minimum first stretch of a two part regular duty. Therefore if the spell in question is not a legal first stretch for a two part regular duty then it is not considered to be suitable for a split duty.

These tests on the stretch lengths are made for every spell induced by every pattern variable element that exists in the CSP except for spells going later than 1200.

Some constraints can be posted after some lookahead has been performed after a variable has been bound. These are discussed next.

6.6.3 Lookahead performed when PV variables are bound

When a pattern variable is bound a routine is invoked that performs a lookahead to ensure that a regular duty with its first stretch on vehicle i (with a spell that finishes before 1200) only has a mealbreak link with a second stretch that creates a duty with a legal cost. Cost, in the CROSS algorithm, is defined as the total time of a duty from sign-on to sign-off. The minimum and maximum cost parameters are defined in the labour file. A general

description of this lookahead follows.

If a stretch of a regular duty starts on running board i at relief opportunity (i, m) and finishes at relief opportunity (i, n) then each domain element representing a mealbreak link (relief opportunity) in $Next_{(i,n)}$ is examined. For each relief opportunity value, say (k, p), in $D_{Next_{(i,n)}}$ the pattern variable domain elements for PV_k are examined. There are two cases: either (k, p) is a relief opportunity at the beginning of running board k or (k, p) is an internal[3] relief opportunity on running board k.

If (k, p) is at the beginning of running board k then each D_{PV_k} element, say S, is examined. For each spell formed from (k, p) to, say (k, q) where $(((k, q) \in S) \vee (S = \{-1\})) \wedge (q > p)$, the cost of the potential duty formed by the two spells (i, m) to (i, n) and (k, p) to (k, q) is calculated. If it is too long or short to form a legal duty (the min/max cost parameter is violated) then the following constraint is posted preventing PV_k from taking the value S should $Next_{(i,n)} = (k, p)$. This is stated as:

$$Next_{(i,n)} = (k, p) \quad \Rightarrow \quad PV_k \neq S \tag{6.18}$$

In the second scenario only D_{PV_k} elements containing p are considered. Spells going past 1200 are not considered in this lookahead.

6.6.4 Lookahead performed when $Next$ variables are bound

If $Next_{(i,j)} = 0$ and PV_i is unbound and either $Prev_{(i,j)}$ has not been bound or $Prev_{(i,j)} \neq 0$ then each element in PV_i, say s, is examined. For each $S \in PV_i$ value where $j \in S$, a spell (i, k) to (i, j) (where k is either the beginning of the running board or in S) is formed. The following constraints are then added to the CSP:

$$Prev_{(i,k)} = -1 \quad \Rightarrow \quad PV_i \neq S \tag{6.19}$$

[3]As defined earlier, an *internal* relief opportunity is one which appears in between the first and last relief opportunity of the morning portion of the running board being considered. An *external* relief opportunity is the first and last relief opportunity on that running board.

$$PV_i = S \quad \Rightarrow \quad Prev_{(i,k)} \neq -1 \tag{6.20}$$

These constraints help to prevent the algorithm from potentially trying to use the spell ending at (i, j) as the first part of a two part duty. At this point it is not clear if (i, j) will be the end of a drivers duty or if relief opportunity (i, j) will be used at all (this depends on the value given to $Prev_{(i,j)}$ if it is not already bound).

6.6.5 Miscellaneous Constraints

When a $Next_{(i,j)}$ variable is bound a routine is invoked to examine the value it has been assigned. If a $Next_{(i,j)}$ variable, where (i, j) is an internal relief opportunity, is not bound to the value 0 then PV_i is tested to see if it has been bound to a value. If PV_i is unbound then it is randomly[4] bound to a value containing j.

When a $Prev_{(i,j)}$ variable is bound it too invokes a routine to examine its value. If $Prev_{(i,j)}$ is bound to a value other than 0 and its pattern variable is unbound then the pattern variable is bound randomly to a value containing j.

The reason for binding the PV_i value at this point is for the constraint propagation such a binding would invoke. This helps to further reduce domains of variables as well as giving structure to the problem (via the partitioning of the vehicles into work).

In both of the routines called when $Prev_{(i,j)}$ and $Next_{(i,j)}$ variables are bound there are special VALID routines that examine the duties, or partial duties, constructed thus far. If there are any unusual constraints that must be taken into account then it can be done at this point to ensure no illegal duties are created.

[4]The method for probabilistically binding pattern variables is described in Section 7.3.8.

6.7 Summary

The CROSS CSP has been described. The CROSS CSP was developed as a front end, or pre-processing stage, in the TRACS-II process. The input for the CROSS CSP is a labour file (containing constraints on the duties themselves) and a vehicle file (containing a description of the set of bus workings). The CROSS process produces a new vehicle file as output having removed some of the relief opportunities that were contained within the original vehicle file. This reduces the number of potential duties generated in the TRACS-II BUILD process and reduces the number of constraints in the TRACS-II ILP.

The motivation behind CROSS is the observation that mealbreak chains have beneficial properties in the context of duty scheduling. Schedules that contain good mealbreak chains should require fewer duties than schedules not containing them.

The CSP itself consists of three sets of constrained variables. The first is a set of *pattern variables*. There is one pattern variable per running board. Pattern variables are used to represent how the work on a running board is partitioned (which relief opportunities are used). Each relief opportunity has an associated $Next$ and $Prev$ variable which determine what a duty does after and before the relief opportunity respectively. By chaining together successive relief opportunities ($Next$ and $Prev$ variables) mealbreak chains can be formed.

The CROSS CSP is designed to form a set of mealbreak chains over the morning period of a set of bus workings, although as described later, it can also be applied to evening portions of a set of bus workings. The algorithm employed for solving the CSP (as described in the next chapter) has a random element introduced into it so that the CROSS CSP is solved several times forming a variety of mealbreak chains over the set of morning bus workings. The union of all the relief opportunities participating in these mealbreak chains is recorded and passed onto the TRACS-II BUILD process via the output vehicle file.

CROSS is implemented using ILOG Solver. The constraints are generally represented as *conditional* constraints. These constraints only take effect when certain criteria are met. Constraints are posted to Solver regarding the interaction between $Next$ and $Prev$

variables which is fundamental to the construction of mealbreak chains in the CROSS CSP. Constraints are posted on PV variables as well as on the interaction between PV and $Next/Prev$ variables, mostly to ensure that illegal duties are not allowed to be formed around the mealbreak chains. Other constraints are posted when PV, $Next$ or $Prev$ variables themselves are bound, again to help ensure only legal duties are traced out around the mealbreak chains formed. Some lookahead is also performed.

Next, in Chapter 7, the search strategy used in solving the CROSS CSP is discussed.

Chapter 7

The CROSS Algorithm

7.1 Introduction

This chapter presents a description of the CROSS algorithm. First an outline of the algorithm is given. This provides an overview as to how the algorithm works and how relief opportunities are selected such that mealbreak chains are formed. The CROSS algorithm itself is then discussed, focusing on the flow of the algorithm. The method by which PV_i, $Next_{(i,j)}$ and $Prev_{(i,j)}$ variables are randomly assigned values will also be presented.

After the algorithm has been discussed the focus will then be placed on other related issues. These include how CROSS can be applied to an evening portion of a set of running boards as well as applying CROSS to select relief opportunities when duties in the TRACS-II schedules may contain three or four parts.

It should be noted that the discussion of the CROSS algorithm is specific to finding a single solution to the CSP. The actual application of the CROSS method will involve finding several solutions, with a random element, to the CROSS CSP. The union of all the relief opportunities selected by CROSS will then be used as the set of relief opportunities selected and supplied to TRACS-II via a new vehicle file. This satisfies the two additions

needed in order to gain the flexibility from a mealbreak chain selection scheme as outlined in Section 6.2.1.

7.1.1 Earlier Work

The search strategy for the original CSP formulation (as described in Section 6.1.1) is similar in spirit to the algorithm that will be described in this chapter. Its main objective is to trace out mealbreak chains via the relief opportunities that they are comprised of.

First the earliest unbound workpiece, W_i, variable was chosen. A value (a pointer to another piece of work) was selected from its domain randomly with a bias towards the adjacent piece of work (if legal); this bias is in order to try and create long rather than short stretches of work. When a mealbreak is formed the same strategy is used in order to form the duties' second stretch of work. The algorithm would then select the workpiece before the second stretch of work just formed in order to form a mealbreak link and the second stretch of the implicitly formed duty is created. This is repeated until the mealbreak chain currently being constructed comes to an end. When a mealbreak chain has come to an end the algorithm restarts with the earliest unbound piece of work. Eventually the entire set of W_i variables are assigned a value.

Unfortunately, without the extensive use of look-aheads, the constraint propagation occurring in this model was not entirely satisfactory and a large amount of backtracking would take place, even on relatively small problems. Some attempt was made to remedy this by modifying the search strategy slightly. For example only a limited number of values were tried from a variable's domain before backtracking was invoked. Additionally a mechanism was implemented whereby if a solution was not found within a set period of time, a variable earlier in the search tree was backtracked to at random in an attempt to avoid the troublesome part of the search space (backtracking techniques such as these were successfully applied in a university course timetabling heuristic [75]).

It was decided that by modelling the problem slightly differently (with more emphasis on

how a running board is to be split up into pieces of work and, thus, on which relief opportunities will be used) the goals of both a better model and better constraint propagation could be realised.

The changing of the problem's variables to represent what happens before and after a relief opportunity ($Next$ and $Prev$ variables) as well as a mechanism to partition a vehicle into pieces of work (PV variables) made the CSP much easier to solve and the constraint propagation much more effective. The model now used also shifts the emphasis onto the relief opportunities themselves rather than the pieces of work.

7.2 Simplified Algorithm Outline

Using the model defined in Chapter 6 the CROSS algorithm selects relief opportunities from the morning period of a set of bus workings by forming mealbreak chains. A simplified version of the CROSS algorithm is displayed in Figure 7.1. The diamond objects on the arrows in the diagram indicate a decision point regarding whether or not the current mealbreak chain currently being traced out is finished or not.

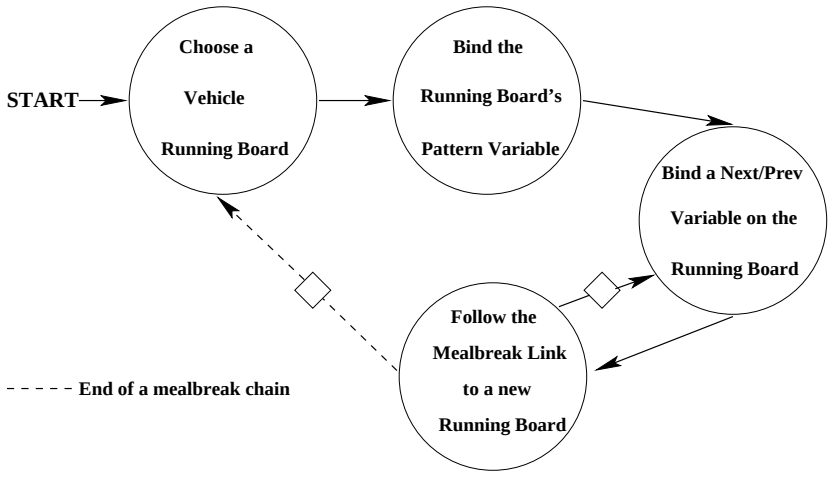

Figure 7.1: Simplified CROSS Algorithm

Figure 7.1 shows that the algorithm starts by selecting a running board. Generally this will be a running board containing a relief opportunity whose $Next_{(i,j)}$ variable is unbound.

The running board selected, say i, will first have its pattern variable, PV_i, bound (if it has not been assigned a value already). The earliest unbound $Next_{(i,j)}$ variable[1] on this running board whose time, j, is selected in PV_i or which occurs at the end of the running board but before 1200 will then be bound, usually forming a mealbreak link or alternatively representing a duty finishing its mealbreak after 1200 or perhaps indicating a split duty. Usually the mealbreak link connects the relief opportunity (i, j) with a relief opportunity on another running board, say relief opportunity (l, m). The running board l is considered next. The $Next_{(l,m)}$ variable is bound, forming another mealbreak link, and so on. This has the effect of tracing out a mealbreak chain through the bus workings as well as forming duties around the chain. This process is repeated until all the variables in the constraint programming model are assigned a value. Many other factors have to be considered whilst the mealbreak chains are being constructed; these will be explored in more detail in this chapter.

7.3 Algorithm

The CROSS algorithm's search strategy itself is split up into four routines that each have a distinct role in the mealbreak chain generation process. The search for mealbreak chains centres on the movement between these routines depending on what needs to be done at the time. Figure 7.2 shows these four routines and the flow of control between them.

The four main routines in the algorithm are as follows:

- **StartNewChain** - Selects a running board, i, with an unbound $Next$ or $Prev$ variable. Binds PV_i if needed.
- **BindFirstVariable** - Binds the earliest $Next$ or $Prev$ variable on a running board.
- **ContinueNextChain** - Deals with newly bound $Next$ variables. Tries to extend current mealbreak chain.

[1] A similar process is followed in the case that the earliest unbound variable on a running board is a $Prev$ variable.

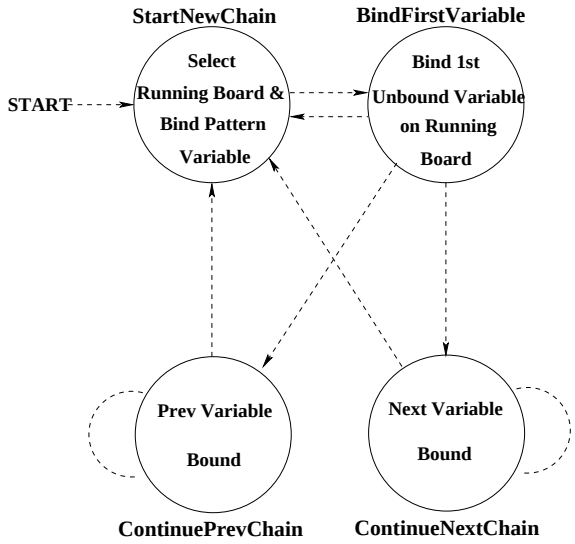

Figure 7.2: CROSS Mealbreak Chain Search Strategy

- **ContinuePrevChain** - Deals with newly bound *Prev* variables. Tries to extend current mealbreak chain.

Some implementation details of the algorithm will be described next. Afterwards the four routines that comprise the search strategy will be explained in some detail.

7.3.1 Implementation

ILOG Solver provides built in functions to solve a CSP. Specifically, it supplies a function called **IlcGenerate()** which is responsible for selecting the next variable to be bound in the algorithm (variable ordering). Another function, **IlcInstantiate()**, binds a constrained variable to a value from its domain (value ordering). These correspond to the constraint programming concepts of variable ordering and value assignment.

The approach taken in the CROSS algorithm uses its own custom-made **Generate()** and **Instantiate()** functions in order to have more control over the variable ordering as well as the value assignments used in the algorithm. The **Generate()** function contains references to the four routines used in the CROSS algorithm; the custom **Instantiate()** function controls how $Next_{(i,j)}$, $Prev_{(i,j)}$ and PV_i variables are bound to values in their

domains.

ILOG Solver provides a useful data type that is used in the CROSS algorithm. Specifically, ILOG Solver provides a *reversible* data type. Such a data type can be assigned values like any other variable in a program. However, when any backtracking takes place the variable's value is restored to the value it had at the point to which the algorithm backtracks. A reversible structure, referred to as the *Status* of the algorithm, is constructed and every time **Generate** is called the running board and current relief opportunity being examined are recorded. This serves two purposes:

1. It informs the next routine in **Generate** to be called what running board and relief opportunity are currently being examined.

2. When backtracking occurs, the *Status* allows the algorithm to revert back to the running board and relief opportunity that was being considered at the point to which the algorithm backtracks. The routine which the algorithm was executing at that point is also recorded.

In the *Status* structure a reversible variable is used to record which of the four routines is to be used next. The **Generate** function uses the value of this in a `switch` statement in order to ensure the correct routine is executed.

The search strategy begins in the **Generate** function and invokes the **StartNewChain** routine. From here we focus on the formation of meal-break chains.

7.3.2 StartNewChain - Starting a new chain

The first step carried out involves selecting a running board on which a mealbreak chain will be started. A list of the earliest twenty relief opportunities with either an unbound $Next_{(i,j)}$ or $Prev_{(i,j)}$ variable is constructed. One of these relief opportunities is selected at random and the running board it is on, say i, will be used.

This random selection of running boards helps to make the solution of the CSP different from run to run. When the same running board was always used to start the search then it was found that many of the solutions looked somewhat similar, especially in problems where the domain size of many of the pattern variables is small. Allowing the algorithm to choose a running board from the twenty earliest relief opportunities ensures that each solution is different.

Next, the pattern variable for running board i, PV_i, is bound (if needed) thus partitioning the work on the running board. This is done in a random fashion which is detailed in Section 7.3.8. The search next moves to **BindFirstVariable** whose goal is to bind the earliest unbound $Next_{(i,j)}$ or $Prev_{(i,j)}$ variable on the running board. The unbound variable will be chosen corresponding to the relief opportunities given in the value assigned to the pattern variable.

7.3.3 BindFirstVariable - Binding the first Unbound Variable on a Running Board

BindFirstVariable searches the running board, say i, selected by **StartNewChain** for the earliest relief opportunity with an unbound $Next_{(i,j)}$ or $Prev_{(i,j)}$ variable. There are two cases to consider:

1. There are no relief opportunities with an unbound $Next_{(i,j)}$ or $Prev_{(i,j)}$ variable on the running board.
2. The earliest relief opportunity, say (i, j), with an unbound variable has either an unbound $Next_{(i,j)}$ or $Prev_{(i,j)}$ variable.

Case 1 If there is no relief opportunity on running board i with an unbound $Next_{(i,j)}$ or $Prev_{(i,j)}$ variable then control is passed back to **StartNewChain** in order to select another running board to try and start a new mealbreak chain.

Case 2 The earliest relief opportunity with an unbound variable has either a $Next_{(i,j)}$ or

$Prev_{(i,j)}$ variable unbound. If a $Next_{(i,j)}$ variable is unbound, then it is bound in a random fashion to form a mealbreak link, if possible, and the search then proceeds to the **ContinueNextChain** routine. If $Next_{(i,j)}$ has already been bound then $Prev_{(i,j)}$ is bound to a value in a random fashion and the search will proceed to the **ContinuePrevChain** routine. The random method by which $Next_{(i,j)}$ and $Prev_{(i,j)}$ variables are bound will be discussed in Section 7.3.9.

It is worthy of note that at the beginning of a mealbreak chain on a running board with an unbound pattern variable, PV_i, the pattern variable is bound *before* a $Next_{(i,j)}$ or $Prev_{(i,j)}$ variable; this is the reason for the distinction between the two routines **StartNewChain** and **BindFirstVariable**. It is important to bind the variables in this order at the beginning of a mealbreak chain in order to ensure all pattern variable domain values may be tried, if necessary, in solving the CSP. To see this consider Figure 7.3. In it assume that $D_{PV_i} = \{\{a\}, \{b\}\}$. Also assume that both $Next_{(i,a)}$ and $Next_{(i,b)}$ are unbound and that $a < b$.

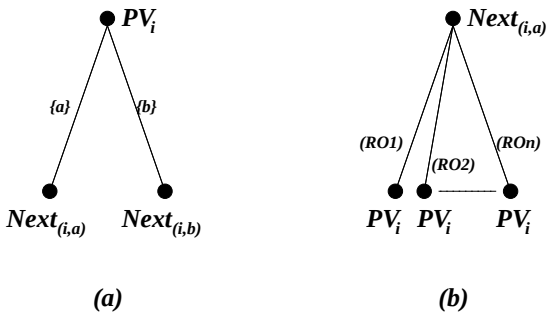

Figure 7.3: Binding PV before $Next$

Figure 7.3 has two parts. Part (a) shows a search tree if the pattern variable is bound before the $Next$ variable. Once the PV_i variable is bound to either $\{a\}$ or $\{b\}$ the $Next_{(i,a)}$ or $Next_{(i,b)}$ variable is bound. If backtracking takes place such that PV_i has to be bound to another value then the other untried $Next$ variable will be bound.

Part (b) of Figure 7.3 shows what happens when $Next_{(i,a)}$ is bound before PV_i. This would be equivalent to **BindFirstVariable** binding a $Next_{(i,j)}$ variable *before* the PV_i variable has been bound. After $Next_{(i,a)}$ has been assigned a value PV_i is bound to some

value containing a. Since only one D_{PV_i} element contains the value a, $PV_i = \{a\}$. If the search backtracks to $Next_{(i,a)}$ it will only assign another relief opportunity value to $Next_{(i,a)}$ and PV_i will be bound with the value $\{a\}$ again. Since **StartNewChain** finds the earliest unbound[2] *Next* (or *Prev*) variable, $Next_{(i,b)}$ will not be chosen at all in this portion of the search tree. However, if the order of variable binding that is shown in part (a) of Figure 7.3 is used then all PV_i domain values will be tried which in turn implies all relief opportunities within its domain values have a chance of being used. When a mealbreak chain is started on a vehicle, i, with an unbound PV_i variable then PV_i *must be bound* before any unbound *Next* or *Prev* variables on running board i.

7.3.4 ContinueNextChain - When a $Next_{(i,j)}$ Variable has been Bound

The role of **ContinueNextChain** is to examine the value of a $Next_{(i,j)}$ variable (usually a mealbreak link) and try to extend the mealbreak chain in which it participates. There are two cases to be considered:

1. $Next_{(i,j)} \in \{1500, 1200, 0\}$.
2. $Next_{(i,j)} = RO_{(l,m)}$ - A relief opportunity representing the finish of a mealbreak link.

Case 1 If the value of the $Next_{(i,j)}$ variable is 1500, 1200 or 0 then the current mealbreak chain being constructed has come to an end and the search moves to **StartNewChain** in order to try and start another mealbreak chain elsewhere.

Case 2 $Next_{(i,j)}$ must be bound to a value representing a relief opportunity, say (l, m). The variable $Next_{(l,m)}$ is examined. If it is already bound then it is assumed that the current mealbreak chain being constructed has joined with the tail end of another mealbreak chain and the search goes to **StartNewChain** in order to try and start a new mealbreak chain elsewhere. If $Next_{(l,m)}$ is unbound then it is randomly bound and the search will stay in **ContinueNextChain** in order to try and extend this

[2] An analogous situation exists for *Prev* variables.

mealbreak chain further. This routine forms mealbreak chains by successively linking together $Next$ variables.

7.3.5 ContinuePrevChain - When a $Prev_{(i,j)}$ Variable has been Bound

This routine examines the value to which a $Prev_{(i,j)}$ variable has been bound. Similar to the **ContinueNextChain** routine, there are two cases to consider:

1. $Prev_{(i,j)} = -1$ - The start of a new duty.
2. $Prev_{(i,j)} = RO_{(l,m)}$ - A relief opportunity representing the start of a mealbreak link where (i, j) is the end of the link.

Case 1 If $Prev_{(i,j)}$ is bound to the value -1 then the current mealbreak chain being constructed has come to an end and the search moves to **StartNewChain** in order to try and start another mealbreak chain elsewhere.

Case 2 If $Prev_{(i,j)}$ is bound to a value representing the beginning of a legal mealbreak link, say (l, m), then the variable $Prev_{(l,m)}$ is examined. If it is already bound then it is assumed that the current mealbreak chain being constructed has joined with the beginning of another mealbreak chain and the search goes to **StartNewChain** in order to try and start a new mealbreak chain elsewhere. If $Prev_{(l,m)}$ is unbound then it is randomly bound and the search stays in **ContinuePrevChain** in order to try and extend this mealbreak chain further. This routine forms mealbreak chains by successively linking together $Prev$ variables.

7.3.6 Backtracking

The default chronological backtracking of ILOG Solver is used. This implies that only reasonably recent variables in the search will be re-examined by backtracking. This might seem to be a disadvantage. However, in practice, CROSS usually finds a solution within

seconds. Additionally a timeout value is used where the search is restarted if a solution has not been found within a pre-determined amount of time. The new search will almost certainly search a different part of the solution space and it is very unusual for two timeouts to occur in a row. This parameter is usually set to 15 seconds.

Backtracking is most commonly caused by $Next$ variables with empty domains as the mealbreak chain creation process focuses mainly on these. The constraints described in the labour file have a direct influence on this, however. For example, if the $Next$ variables have the value 1500 in their domains then backtracking is unlikely to happen due to the fact that this split shift value is always a legal assignment. If split shifts are not allowed or have tight constraints on when they can occur then backtracking due to empty $Next$ variable domains can be more common; this has not been a serious issue in any of the datasets used.

In **Generate** the search spends, roughly, 25% of its time in the **StartNewChain** and **BindFirstVariable** respectively, 40% of its time in **ContinueNextChain** and 10% of its time in **ContinuePrevChain**.

7.3.7 Random Variable Assignment

When a PV_i, $Prev_{(i,j)}$ or a $Next_{(i,j)}$ variable is bound, every value in its domain should have a chance of being used; this would allow any potential mealbreak chain in the set of bus workings to be formed. If value ordering is done in a deterministic fashion then it is possible that potentially useful relief opportunities will be missed out every time CROSS is used (as the CROSS algorithm itself only looks for *a* solution to the CSP). To help avoid this, special random value ordering routines were created to assign values to the PV_i, $Next_{(i,j)}$ and $Prev_{(i,j)}$ variables. The random routines are designed to try to favour introducing features similar to what are found in efficient schedules.

The domains of the PV, $Next$ and $Prev$ variables themselves are sorted in order to give preference to some values over others. This is discussed next.

7.3.8 Random Pattern Variable Assignment

It is a vehicle's pattern variable that decides which of its relief opportunities will be used. An aim of driver scheduling is for each duty to cover as much bus work as possible. In practice this is not always practical as some shorter duties may be required to cover awkward pieces of work. However, a bias towards longer spells was part of the motivation for using a randomised pattern variable assignment strategy. From the combination of different relief opportunity sets provided by several runs of the CROSS algorithm the TRACS-II build process should be able to generate a selection of long and short duties to supply to the SCHEDULE process. For such a strategy to be used each of the pattern variables' domains has to be sorted into order of desirability. After this has been done a method of randomly selecting elements from the domain of a pattern variable must be constructed.

When the pattern variables' domains are sorted there are two cases to be considered; either the schedule is peaked or non-peaked. First, the case of pattern variables on non-peaked sets of bus workings will be considered.

A pattern variable's domain is sorted based on two criteria (in order of preference):

1. Length of first spell from the beginning of the running board as formed by the pattern variable domain element.
2. Length of second spell on the running board as formed by the pattern variable domain element.

The domain elements are sorted in descending order based on the length of the first spell on the running board formed by each domain element. For domain elements that form the same first spell on a running board the length of the second spell they form is compared and, again, longer spell lengths are given preference.

To see how the pattern variable domain element ordering works, consider what the pattern

variable domains from the set of bus workings in Figure 7.4 would look like. Assume the spells formed on this set of bus workings are constrained by the following parameters:

- MINIMUM SPELL LENGTH is set to $1\frac{1}{2}$ hours.
- MAXIMUM SPELL LENGTH is set to 5 hours.

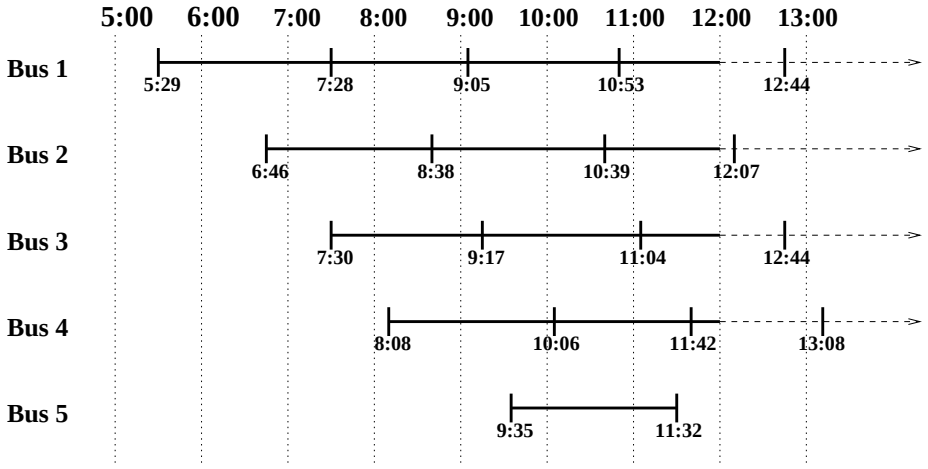

Figure 7.4: Ash Grove Route 11 (London) Morning Bus Workings

$$
\begin{aligned}
D_{PV_1} &= \{\{905\}, \{905, 1053), \{728, 1053\}, \{728, 905\}\} \\
D_{PV_2} &= \{\{1039\}, \{838\}, \{838, 1039\}\} \\
D_{PV_3} &= \{\{1104\}, \{917\}, \{917, 1104\}\} \\
D_{PV_4} &= \{\{-1\}, \{1142\}, \{1006\}, \{1006, 1142\}\} \\
D_{PV_5} &= \{\{-1\}\}
\end{aligned}
$$

Figure 7.5: Ash Grove Non Peaked Bus Workings Sorted Pattern Variables' Domains

If the bus workings in Figure 7.4 were treated as a non-peaked set of bus workings the sorted pattern variables would look like what is found in Figure 7.5. The first element in D_{PV_1} is $\{905\}$ because 905 forms the longest legal spell from the beginning of the running board on bus one. The value $\{905, 1053\}$ is directly after $\{905\}$ because, although both domain elements share the longest first spell on the running board, the domain element $\{905\}$ has a longer second spell than the domain element $\{905, 1053\}$, hence its preferential ordering. The first domain element in D_{PV_4} is $\{-1\}$ as this value uses the entire morning

portion of the running board as a single spell (which, by definition, is the longest). By inspection it can be seen the rest of the domains in Figure 7.5 follow the rules outlined above.

For the CROSS algorithm the definition of a peaked schedule is that 20% of the vehicles in the set of morning running boards are peak vehicles (end before 1200). This is identified during the initialization phase of the algorithm. The bus workings in Figure 7.4 could be considered a peaked set of bus workings.

In the case of a peaked set of bus workings the pattern variable domains are sorted somewhat differently. Before invoking the previous set ordering (the ordering used for non-peaked bus workings) on a pattern variable's domain, the domain is first partitioned into three subsets. These three subsets will each contain domain elements with a particular property. These properties (in order of importance) are:

1. Domain elements whose relief opportunities are all *after* the peak, say subset $S_{(i,1)} \subseteq D_{PV_i}$.
2. Domain elements whose relief opportunities are all *before* the peak or occur on *either side* of the peak, say subset $S_{(i,2)} \subseteq D_{PV_i}$.
3. Domain elements containing at least one relief opportunity that falls *within* the peak, say subset $S_{(i,3)} \subseteq D_{PV_i}$.

Once these three sets have been determined the domain of pattern variable, PV_i, is ordered as follows:

$$PV_i = \{S_{(i,1)}, S_{(i,2)}, S_{(i,3)}\} \tag{7.1}$$

Each of the three subsets in the pattern variable is then ordered by the same criteria that is used for a non-peaked pattern variable domain. Using, again, the set of bus workings in Figure 7.4 as an example and treating the schedule as peaked between the times of 935 and 1132 the pattern variable domains for the vehicles in this schedule will look like what

is found in Figure 7.6.

$$
\begin{aligned}
D_{PV_1} &= \{\{905\},\{728,905\},\{905,1053),\{728,1053\}\} \\
D_{PV_2} &= \{\{838\},\{1039\},\{838,1039\}\} \\
D_{PV_3} &= \{\{917\},\{1104\},\{917,1104\}\} \\
D_{PV_4} &= \{\{-1\},\{1142\},\{1006\},\{1006,1142\}\} \\
D_{PV_5} &= \{\{-1\}\}
\end{aligned}
$$

Figure 7.6: Ash Grove Peaked Bus Workings Sorted Pattern Variables' Domains

Once a pattern variable domain has been sorted, each element in the domain must have an associated probability of being chosen for value ordering purposes. A linear probability distribution was chosen in order to give every element in the domain a reasonable chance of being chosen but, at the same time, providing a bias towards preferred elements. If there are n elements in a pattern variable's domain each element, $e_1, \ldots, e_n$, has a probability $P_1, \ldots, P_n$ of being chosen. The first element in a pattern variable domain of size n will have n times the chance of being selected compared to the last element in the domain (whose probability of being selected is x). The sum of these probabilities will add to one as described by the following equation:

$$nx + (n-1)x + \cdots + 2x + x = 1 \tag{7.2}$$

The terms $nx, (n-1)x, \ldots, 2x, x$ correspond directly to $P_1, P_2, \ldots P_{n-1}, P_n$. This can be simplified and we define P_i as follows:

$$P_i = 2\left[\frac{(n+1-i)}{n(n+1)}\right] \tag{7.3}$$

As the domains of pattern variables vary in size during the CROSS algorithm execution the probabilities of them being selected are adjusted to remain consistent with the current size of the domain.

7.3.9 Probabilistic $Next_{(i,j)}$ and $Prev_{(i,j)}$ Variable Assignment

The $Next_{(i,j)}$ and $Prev_{(i,j)}$ variables' assignments are also important. Ideally mealbreaks should be as short as possible in order minimize the amount of idle time present in mealbreaks, thus, creating more efficient duties. Like the pattern variable domains, $Next_{(i,j)}$ and $Prev_{(i,j)}$ domains are sorted in order of preference with respect to value ordering. The following description specifically addresses the probabilistic binding of $Next_{(i,j)}$ variables but the same method is also used for $Prev_{(i,j)}$ variables.

One of the parameters given in the labour agreement file is the *maximum mealbreak idle time* length allowed. This parameter specifies the maximum amount of idle time allowed in a driver's mealbreak. This does not include, of course, any travel time to and from a relief point. The *travel time* specifies the amount of time allocated for a driver to travel from one relief opportunity to another. This value is taken into account when the $Next_{(i,j)}$ and $Prev_{(i,j)}$ domains are initially generated to ensure all relief opportunities in their domains are legal (and thus will generate legal mealbreak chains[3]). The travel time is taken into account *in addition* to the minimum mealbreak length.

The relief opportunities in the $Next_{(i,j)}$ variable's domain are partitioned into five subsets. Each subset contains relief opportunities within a certain band of mealbreak idle time. These bands of idle time are calculated from the maximum mealbreak parameter divided by this value of five. Thus, if a set of bus workings has a maximum mealbreak value of 50 minutes, the set of relief opportunities in $Next_{(i,j)}$ domains for this problem are partitioned into five sets each of ten minutes in length. The first set would contain relief opportunities forming a mealbreak with 0-10 minutes of idle time with respect to relief opportunity (i, j), the second set will contain relief opportunities that form 11-20 minutes of idle time with respect to relief opportunity (i, j) and so on up until the fifth band which will contain relief opportunities that form 41 to the maximum 50 minutes of allowed idle

[3]For example, if we are generating a domain for $Next_{(i,j)}$ and are currently considering adding relief opportunity (l, m) to the domain, then $m - j \geq$ MinMealbreak + TravelTime$((i, j), (l, m))$. Where MinMealbreak is the minimum mealbreak length and TravelTime is a function that returns the travel time between two relief opportunities.

time.

When a $Next_{(i,j)}$ variable is to be assigned a value from its domain in the CROSS algorithm it is done in a random fashion. Using the probability distribution used for pattern variable binding (represented in Equation (7.3)), except with n set to the value of five, one of the five subsets in the $Next_{(i,j)}$ variable's domain will be chosen randomly. Once a subset has been selected a relief opportunity is taken at random from this subset and assigned to the $Next_{(i,j)}$ variable. If this subset is empty then the next earliest relief opportunity available is chosen. The values representing split duties and mealbreaks going past 1200 are only used once all legal relief opportunities are exhausted as they will not contribute to a mealbreak chain in a meaningful fashion. In the case of $Prev_{(i,j)}$ variables the value -1, representing the start of a new duty, is only tried once all relief opportunities in the domain have been exhausted. By avoiding binding $Prev_{(i,j)}$ to -1 it is hoped this will help avoid the creation of new driver duties.

If the schedule is peaked then there is an additional check made when assigning a $Next_{(i,j)}$ or $Prev_{(i,j)}$ variable in order to avoid selecting relief opportunities during the peak. If the value selected for the $Next_{(i,j)}$ or $Prev_{(i,j)}$ variable is a relief opportunity that falls within the peak then another value is chosen, using the same method, and kept. This additional value ordering attempt at binding the variable makes the selection of a relief opportunity during the peak less likely.

Splitting up the domain of a $Next_{(i,j)}$ or $Prev_{(i,j)}$ variable into five sets may seem odd at first glance. There are, however, good reasons why this was done. To see this consider the case where a large number of running boards are in the problem and/or a liberal maximum mealbreak parameter is used. The cardinality of the domains of $Next_{(i,j)}$ and $Prev_{(i,j)}$ variables can run into the hundreds. Assigning a probability for elements in these large variable domains to be picked using the same distribution that is used for pattern variable value assignment would result in many of the elements being unlikely to be chosen. By statically partitioning the domains this problem is avoided whilst keeping the desired bias towards shorter mealbreaks.

7.3.10 Relief Opportunity Percentage

The CROSS algorithm will be invoked several times such that several solutions to the CROSS CSP are generated. The union of all relief opportunities used in these solutions is calculated and a new vehicle file is produced which uses only these relief opportunities in the morning portion of the bus workings.

A parameter value known as the *Relief Opportunity Percentage* or ROP must be specified. This value specifies the target percentage of relief opportunities which must be selected by the CROSS algorithm. The total number of relief opportunities available for selection is defined as the number of internal relief opportunities represented in the domains of all of the pattern variables. The ROP refers to the percentage of relief opportunities selected from this set. The parameters given in the labour file (such as minimum spell length) will, of course, have a direct impact on the size of this set.

CROSS generates solutions to the CSP until the percentage of relief opportunities used is greater than or equal to the value of the ROP parameter.

7.4 Evening Relief Opportunity Selection

Evening relief opportunity selection is treated in an analogous manner to the morning case. The set of running boards is inverted temporally such that the evening period of bus workings is treated in a similar way to a morning set of bus workings. The running board that finishes at the latest time in the evening is treated as a bus leaving the garage at 4AM and the rest of the running boards have their relief opportunity times changed to reflect this. Thus the *evening period* of a set of bus workings can be defined as the set of bus workings occuring in the eight hour timespan that ends at the latest finish of a bus in the bus workings[4].

[4]For example, if the latest finish of a bus in the bus workings is at 1:00AM, the evening period would be defined as the set of bus workings occuring between 5:00PM and 1:00AM.

After the bus workings have been inverted a new vehicle file is generated using these times. CROSS is then applied to this new data which selects several sets of relief opportunities. The running boards are then inverted back to their original times with any times not selected by CROSS removed.

Using this technique a vehicle file can contain running boards in both the morning and evening periods that have been processed by CROSS.

The afternoon period in a set of bus workings (the buswork that is not processed in either the morning or evening period) is not considered by CROSS. There are two reasons for this. Firstly, when relief opportunities are removed from the morning and evening period this untouched afternoon period of buswork is left intact so that the duties generated by BUILD will have as much flexibility as possible to create duties to cover the afternoon period given the new set of morning and evening relief opportunities in the problem. Secondly, there is, generally speaking, not as many relief opportunities to select from in this afternoon period so the case for relief opportunity selection during this period is not as strong and probably should be avoided.

7.5 Three and Four Part Duties

In many driver schedules, duties with more than two parts are not uncommon. The CROSS algorithm constructs mealbreak chains and, as a result of this, forms two part duties around these chains at the same time. Three and four part duties are never explicitly created, or catered for, in the CROSS algorithm.

This does not mean that three and four part duties cannot be taken into account. Recall that a three or four part duty consists of two stretches of work. Each stretch can contain one or two spells. If a stretch contains two spells they are separated by a joinup. A joinup is a short break in between two spells that is much shorter than a mealbreak. Duty scheduling problems that use three or four part duties will have reasonably short minimum spell length parameter values in the labour file in order for BUILD to be able

to generate such duties. The minimum cost parameter should be reduced in order for CROSS to be able to create short duties that will select relief opportunities likely to be of use to a three/four part duty in BUILD. This is especially critical as it is possible that a three/four part duty may have a very short first spell of work. If the minimum duty cost is not reduced to reflect this then relief opportunities near the beginning of a running board may never be selected and relief opportunities that would be legal in a three/four part duty would be missed altogether.

7.6 Summary

The CROSS algorithm provides a method of selecting relief opportunities from the morning, and evening, periods of a set of bus workings through the solution of the CROSS CSP. Several sets of relief opportunities are selected each resulting from a solution to the CSP model.

The initialisation stage in the CROSS algorithm creates the constrained variables to be solved in the CSP process and assigns them domains. Various constraints are placed on these variables in order to ensure that both legal duties are traced out in the mealbreak chain generation process and constraint propagation takes place when the variables are assigned values. These constraints, as described in Chapter 6, also take into account various relationships that exist between the $Next$, $Prev$ and PV variables that exist due to the model itself.

The CROSS algorithm itself is defined as being split into four routines that are used to help solve the CSP such that two part duties are formed around mealbreak chains. The algorithm starts by selecting a running board from which to start a mealbreak chain and binds the pattern variable on it. The earliest unbound $Next$ or $Prev$ variable is selected. More often than not a $Next$ variable is chosen and a mealbreak chain is started at that point. Successive $Next$ (or $Prev$) variables are bound until a mealbreak chain is completed then the process is started again. Once all the variables in the CSP have been bound to a

value a solution has been found. Chronological backtracking is used when a constrained variable is found to have an empty domain.

One of the key elements in this CROSS algorithm is the way in which PV_i, $Next_{(i,j)}$ and $Prev_{(i,j)}$ variables are randomly assigned values. For pattern variables their domains are sorted sets (with different orderings for peaked and non-peaked sets of bus workings) with a domain element's position in the ordering having a direct link to its probability of being chosen. A linear probability distribution is used such that all the domain elements have a reasonable chance of being selected but there will exist a bias towards favoured elements. Some effort is made, with respect to the random binding of pattern variables, to avoid selecting relief opportunities in the peak.

The $Next_{(i,j)}$ and $Prev_{(i,j)}$ variables are also bound randomly. Their domains are split into five sets; each set contains relief opportunities that fall within a given band of meal-break idle time allowed. The same probability distribution used for binding PV_i variables is used for selecting which of the five sets are used from which a relief opportunity is selected at random and assigned to the $Next_{(i,j)}$ or $Prev_{(i,j)}$ variable in question. A parameter is supplied to CROSS indicating what the minimum percentage of internal relief opportunities should be selected.

The CROSS process can also be used to select relief opportunities from the evening period of a set of bus workings. The schedule is temporally flipped such that the evening period becomes the morning period and is then processed by CROSS. The running boards are then temporally flipped back to their original evening times but with only the relief opportunities selected by CROSS. With this technique vehicle files can be produced that have had their relief opportunities selected in both the morning and evening periods by CROSS.

CROSS is able to select relief opportunities that would be suitable for the generation of potential three or four part duties. This is done by modifying the labour file to reflect the minimum spell lengths and duty costs allowed. These rule modifications combined with the variety of relief opportunities selected by several solutions to the CROSS CSP enable

the TRACS-II BUILD process to generate three/four part duties with some success.

Chapter 8

Results

8.1 Introduction

This chapter presents the results produced using the CROSS pre-processor in conjunction with the TRACS-II system. The pre-processor has been applied to eight data sets. The data sets are real data from several bus companies and have a variety of scheduling conditions.

This chapter is divided into several parts. First, the TRACS-II schedules produced using the original unaltered data are presented. Next, schedules produced by TRACS-II using the CROSS pre-processor are also reported. The effect of the REDUCE process in contrast with the CROSS process is also examined followed by a discussion of the results reported in this chapter.

8.2 Original Solutions

Using the complete set of relief opportunities for each set of data, a duty schedule was generated using TRACS-II. The results generated serve as a baseline in order to see how well the CROSS pre-processing system works with respect to these original solutions.

Table 8.1 shows this set of solutions. Column 1 shows the problem name. Column 2 contains the number of potential duties generated by TRACS-II's BUILD process. Column 3 contains information on the number of parts used in the duties generated by BUILD. Column 4 shows the number of workpiece constraints used in the SCHEDULE ILP; this roughly equates to the number of relief opportunities in the original schedule. The cost of the schedule is displayed in column 5 and takes the form [*hours*].[*minutes*]. The number of duties used to cover all the buswork is in column 6. The last two columns show the time it took BUILD to generate the set of potential duties and the total time SCHEDULE took to produce a solution. The times in these last two columns take the form [*minutes*].[*seconds*]. The second row for each set of data represents the cost and scheduling time for schedule generation using the REDUCE option in TRACS-II.

Problem Name	Potential Duties	Duty Type	Work Pieces	Schedule Cost	Duties Used	BUILD Time	SCHED Time
HE01	43,594	1/2/3/4	158	123.52	14	5.21	11.56
HE01(R)				123.53			3.16
R222	10,088	2	175	223.45	24	0.25	0.57
R222(R)		2		223.45			0.58
EA2	21,664	2	222	223.21	27	1.07	28.04
EA2(R)				223.21			5.34
R207	23,004	2	216	378.43	39	0.22	1.43
R207(R)				379.08			1.33
TRAM	81,928	2	487	406.14	49	2.33	158.30
TRAM(R)				406.06			23.10
R77A	76,906	2	352	490.26	53	2.28	88.31
R77A(R)				493.03			17.40
R61	100,000	2	540	596.42	69	3.05	98.42
R61(R)				597.42			19.57
UMAE	97,429	2/3	873	1171.06	133	76.00	185.18
UMAE(R)				1175.17			140.30

Table 8.1: Original TRACS-II Test Problem Solutions

Examining Table 8.1 one can see that the time taken to generate duty schedules is related to both the size of the schedule (in terms of relief opportunities) and the number of potential duties used in the SCHEDULE ILP. The number of duties generated by BUILD is influenced by both the number of relief opportunities available as well as the types of duties used. For example, the smallest schedule in Table 8.1 is HE01 (Heathrow Express). Although this schedule is quite small the inclusion of 3 and 4 part duties in the BUILD

process greatly increases the number of potential duties that can be generated, which in turn also influences the solution time required by TRACS-II's SCHEDULE component. R222 is a larger problem than HE01 in the sense that its bus workings contain more relief opportunities and running boards but since it only employs two part duties it is easier and quicker to solve.

One of the advantages of a system like TRACS-II (as highlighted in Section 6.2.2) is the ease with which it can be adapted to the different rules and constraints used by various bus companies. This is accomplished by creating a new version of BUILD that ensures that any potential duties generated adhere to the constraints set by the bus company in question. Three versions of BUILD were used for the problems examined in this chapter, each containing slight variations as to how duties are constructed. For example, the EA2 set of bus data represents buses managed by Centre West (London). Driver costs are calculated differently than in the UMAE data set (which belongs to the Reading Buses company[1]) therefore each problem requires its own version of BUILD. The fact that several different versions of BUILD are used also gives an opportunity to see how robust CROSS is regarding different scheduling requirements. Schedules for EA2 and R61 used the Centre West specific version of BUILD. The UMAE schedules were produced using the Reading BUILD program and the remaining data sets employed a generic BUILD program which equates the cost of a duty with its length. The Centre West and Reading Buses BUILD programs calculate the costs slightly differently.

All results reported in this chapter, aside from the UMAE data set, were generated on a Pentium Pro-200Mhz machine running Windows-NT 3.51. As all the results were produced on the same computer, comparisons between BUILD and SCHEDULE times will provide accurate information. The sole exception is that the solutions generated for the UMAE data set were generated on a 333MHZ machine running Windows-98.

The datasets used for this comparison have different characteristics. Specifically, the density of relief opportunities available for selection in the morning/evening periods examined

[1]The experiences of installing the TRACS-II system for Reading Buses can be found in [138].

as well as the number of running boards/relief opportunities present varies between the data sets. Various statistics have been compiled on the data sets and are presented in Table 8.2. The first column contains the problem name. The next two columns contain the number of running boards in the morning and evening periods of the problems examined. This gives an indication of the size of the bus schedule. The next two columns show the morning and evening relief opportunity densities of the schedules involved. The density is calculated as the average number of internal relief opportunities available for CROSS to choose from per running board. The higher the value, the more combinatorially complex the problem is (the larger the set of potential duties). The next two columns contain the total number of internal relief opportunities available for selection in the morning/evening periods; this value gives a rough indication as to how many *Next* and *Prev* variables will be present in the CROSS CSP for the respective morning and evening periods. The final column indicates whether or not the schedule contains a peak (morning or evening).

Problem Name	Morn Running Boards	Eve Running Boards	Morn RO Density	Eve RO Density	Morn RO's	Eve RO's	Peaked?
HE01	5	10	10.8	11.2	54	56	Y
R222	13	13	4.1	4.0	53	44	N
EA2	16	14	5.0	5.3	60	64	N
R207	25	23	2.7	2.6	53	59	Y
TRAM	21	21	6.3	7.6	133	161	N
R77A	30	33	4.3	3.8	86	104	Y
R61	39	39	3.4	4.9	125	117	N
UMAE	80	87	4.8	2.6	325	161	N

Table 8.2: Test Data Properties

It should be noted that the TRACS-II REDUCE process was not used in the majority of the schedules generated using CROSS. The reason for this is that the effect of REDUCE is similar to that of CROSS[2]. With REDUCE not used, the solutions generated will give a clearer picture of effectiveness of the CROSS method on its own. The sole exception to this was the UMAE data set. The UMAE data set is very large and finding schedules using TRACS-II without the REDUCE process can take a considerable amount of time.

[2]REDUCE removes relief opportunities after a continuous solution has been found in the SCHEDULE phase. A description of REDUCE can be found in Section 4.2.3.3.

The results presented in this chapter are grouped into two sections. The first reports results using data sets that contain non peaked data. The second reports results on data that contains peaks (either in the morning or evening).

8.3 Non Peaked Dataset Results

For all of the results, the CROSS pre-processor is applied to each data set in three ways:

1. Apply CROSS to the morning period of a schedule. Generate a duty schedule using the vehicle file generated by CROSS with TRACS-II.
2. Apply CROSS to the evening period of a schedule. Generate a duty schedule using the vehicle file generated by CROSS with TRACS-II.
3. Use the morning and evening relief opportunities selected by CROSS to form a new vehicle file from parts 1 and 2 (above). Use this data with TRACS-II.

This section describes the results for five sets of data. These are: R222, EA2, TRAM, R61, and UMAE. Each of these data sets will be examined in turn.

8.3.1 R222 Results

The results produced from the bus workings R222 (London) can be found in Table 8.3. The sets of potential duties generated for R222 were produced using the generic version of BUILD.

Table 8.3 (and all the remaining tables of results in this chapter) are formatted as follows. Each table is split into four sections: the first contains the original TRACS-II solution with all relief opportunities available to BUILD, the next section contains average results from schedules generated with CROSS applied to the morning period, the third section displays the average results from schedules where CROSS has been applied to the evening period

and the final section shows results where CROSS was applied to both the morning and evening periods. The combined results are produced from the corresponding morning and evening results[3]. Since there is a random element introduced into CROSS regarding the selection of relief opportunities, each run will be different[4]. In the morning/evening and combined sections each row represents the average of five runs of CROSS. The columns in the results tables are organized as follows: column 1 is the average number of duties generated by BUILD; column 2 contains the average number of workpiece constraints used in the SCHEDULE ILP; column 3 contains the target percentage of relief opportunities used. These values are in 5% ranges (for example the value 70% indicates that between 70% and 75% of the legal relief opportunities in the schedule were selected for this averaged set of solutions). Column 4 displays the average percentage of relief opportunities used; column 5 contains the average number of *unions of solutions* that were generated with the CROSS process[5]; columns 6 and 7 show the average cost of the schedule as well as the average number of duties used in the solutions and the final two columns contain the average BUILD time and the average SCHEDULE time for the schedules generated. Values in parentheses in the last column represent the number of schedules that were found to have *integer solutions* in the SCHEDULE ILP (that is, the branch and bound stage was not necessary). Values with an asterisk beside them indicate that one of the runs did not find a solution in the branch and bound phase of the SCHEDULE ILP (but the target number of duties the branch and bound was trying to achieve was the same as the optimal amount used in the TRACS-II original solution).

The R222 set of bus workings is a fairly straightforward non-peaked set of data. The R222 bus workings can be found in Figure 8.1. All thirteen running boards run throughout the day with several of them running into the late evening.

Several observations can be made from the R222 results that, generally, apply to all the results presented in this chapter. These general observations will be highlighted for R222

[3]The first morning CROSS produced vehicle file is merged with the first evening CROSS produced vehicle file to produce the first combined vehicle file and so on.

[4]The complete set of results can be found in Appendices B, C and D.

[5]Recall that the CROSS process repeatedly selects sets of morning/evening relief opportunities until the percentage of relief opportunities used falls into the specified range.

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
Original Result								
10,088	175	–	100%	–	223.45	24.0	0.25	0.57
Morning Results								
5,305	149.4	70%	72.4%	3.2	226.17	24.0	0.15	0.44
5,186	148.0	65%	68.2%	2.2	226.18	24.0	0.14	0.40(1)
4,902	146.2	60%	63.2%	2.0	226.29	24.0	0.13	0.22
4,655	145.0	55%	59.0%	2.0	227.27	24.0	0.24	0.20(1)
Evening Results								
7,463	157.8	70%	72.2%	3.6	224.53	24.0	0.17	0.37(1)
7,199	155.4	65%	66.8%	3.6	225.21	24.0	0.16	0.35
7,087	153.8	60%	63.4%	3.0	224.53	24.0	0.15	1.15
6,684	151.6	55%	58.2%	3.0	225.36	24.0	0.14	0.55(1)
Combined Results								
3,350	130.0	70%	–	–	227.15	24.0	0.08	0.14(1)
3,071	126.0	65%	–	–	227.44	24.0	0.06	0.11(1)
2,779	123.6	60%	–	–	227.04	24.0	0.06	0.10(1)
2,378	118.6	55%	–	–	229.00	24.0	0.05	0.07(1)

Table 8.3: R222 - Morning/Evening/Combined Average Results (2 part duties))

and only additional observations of note will be discussed regarding the other results.

The average time to generate the set of potential duties with BUILD is considerably reduced. Additionally, the average time needed to run the SCHEDULE process is, the vast majority of the time, considerably lower than in the original TRACS-II solution. This is to be expected because since there are fewer relief opportunities present in the vehicle file given to TRACS-II then there will be fewer combinations from which BUILD can generate potential duties. Similarly there will be fewer constraints present in the SCHEDULE ILP thus making it easier to find a solution to the ILP. Results may also vary due to the version of BUILD being used or due to parameter settings made in the TRACS-II labour or vehicle files. The average costs of the schedules is slightly higher than the cost of the original TRACS-II solution (this is common between most of the results except for the data sets employing 3 and/or 4 part duties as will be discussed later). The average costs for the combined results are more expensive than either the morning or evening results. Regardless, these costs would be considered acceptable. The tradeoff here is with the considerable decrease in solution time (the original TRACS-II solution took four times

as long to solve than did the average combined results for the 70-75% range). The most important feature of the R222 data set is that in every case the minimum number of duties was used.

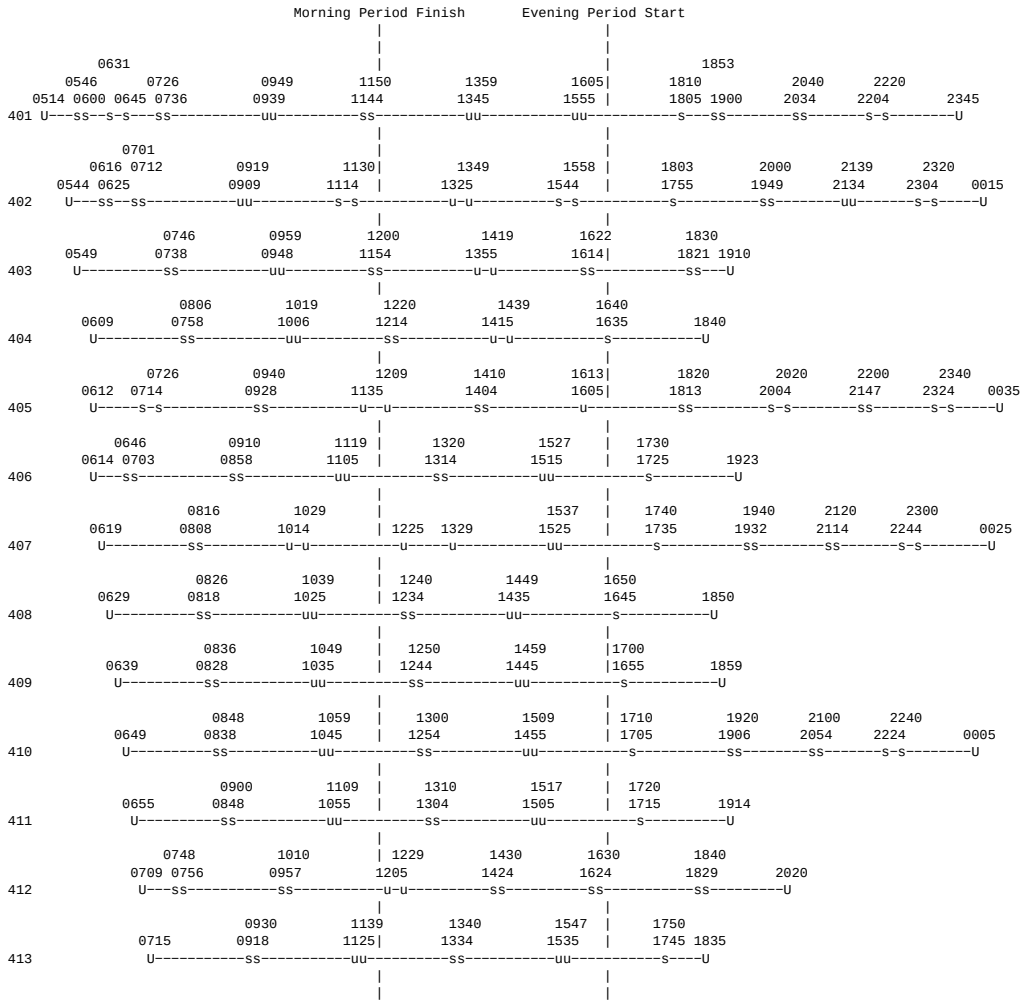

Figure 8.1: R222 Bus Workings

Examining the evening results it can be seen that the average number of potential duties used is considerably larger than in the morning case. The reason for this can be seen from the running boards for R222 as seen in Figure 8.1. The profile of buses in the set of bus workings in a typical weekday schedule (like R222) is different in the morning and evening periods. In peaked schedules the morning peak occurs shortly after the buses have started in the morning but the evening peak occurs several hours before normal bus services finish. In non-peaked schedules, like R222, it can be seen that the morning buses all start within

a period of a few hours. The evening buses can terminate many hours apart from one another. The range where the morning and evening periods for R222 are processed by CROSS are clearly marked with vertical bars through the schedule in Figure 8.1. The number of relief opportunities available for CROSS to select from is fewer in the evening period due to the fact that there are fewer relief opportunities to choose from because there is less work to cover. A result of this fact is that fewer relief opportunities are taken away from the original set of bus workings compared to a morning run of CROSS thus the number of duties created by BUILD is higher. This behaviour can be seen in other results in this chapter. This is a common difference between the morning and evening periods of a set of bus workings.

8.3.2 EA2 Results

The results produced from the bus workings EA2 (London) can be found in Table 8.4. The sets of potential duties generated for EA2 were produced using the Centre West version of BUILD.

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
Original Result								
21,664	222	–	100%	–	223.21	27.0	1.07	28.04
Morning Results								
14,907	210.4	70%	73.4%	4.8	223.21	27.0	0.44	8.13
13,936	208.0	65%	67.6%	4.8	223.25	27.0	0.49	30.27
13,389	204.0	60%	63.0%	4.0	223.25	27.0	0.38	20.43
12,497	202.2	55%	57.8%	3.6	223.22	27.0	0.37	8.57
Evening Results								
15,086	202.4	70%	73.0%	5.4	223.21	27.0	0.48	10.20
13,485	197.6	65%	66.6%	4.8	223.24	27.0	0.45	9.01
12,620	195.0	60%	62.6%	4.2	223.27	27.0	0.43	14.58
12,356	192.4	55%	58.4%	3.4	223.27	27.0	0.40	10.06
Combined Results								
9,854	185.2	70%	–	–	223.25	27.0	0.28	21.04
8,053	178.6	65%	–	–	223.28	27.0	0.22	7.20
7,191	172.6	60%	–	–	223.32	27.0	0.20	3.15
6,378	167.0	55%	–	–	223.40	27.0	0.18	10.46

Table 8.4: EA2 - Morning/Evening/Combined Average Results (2 part duties))

Again, it can be observed that all the results used the optimal number of duties. The average cost in this set of results is much closer to the original cost than in some of the other sets of results. This can be attributed to the CW-BUILD version of BUILD used. Cost is calculated as the length of the stretches worked whereas the cost using the generic version of build is equivalent to the spreadover. The costs using the generic build are more sensitive to different length duties.

8.3.3 TRAM Results

The results produced from the Sheffield supertram data set TRAM can be found in Table 8.5. The sets of potential duties generated for TRAM were produced using the generic version of BUILD.

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
Original Result								
81,928	487	–	100%	–	406.14	49.0	2.33	158.30
Morning Results								
56,114	451.2	70%	71.0%	7.0	405.59	49.0	1.15	105.46
54,460	446.2	65%	67.6%	6.2	406.38	49.2	1.12	145.08
52,246	439.0	60%	63.2%	5.4	406.32	49.0	1.08	58.26
50,415	434.2	55%	59.8%	4.8	406.50	49.0	1.06	86.07
Evening Results								
52,244	442.0	70%	72.4%	8.4	407.28	49.0	1.11	100.48
48,786	433.8	65%	67.8%	6.6	407.04	49.0	1.02	123.17
46,880	427.4	60%	63.6%	5.8	407.09	49.0	0.59	112.28
42,796	419.0	55%	57.6%	5.2	407.25	49.0	0.53	62.53
Combined Results								
32,361	406.4	70%	–	–	407.15	49.0	0.39	101.14
28,247	393.0	65%	–	–	407.18	49.2	0.34	59.53
25,616	379.4	60%	–	–	408.06	49.0	0.29	48.33
21,408	367.2	55%	–	–	408.40	49.0	0.24	20.21

Table 8.5: TRAM - Morning/Evening/Combined Average Results (2 part duties)

This medium sized data set has a structure similar to that found in the R222 data set previously examined. Although there are only twenty one running boards in this data set, it contains the second highest number of morning and evening relief opportunities available for selection by CROSS in all the data examined (as seen in Table 8.2); thus

BUILD generates a substantial number of potential duties.

The results are quite encouraging but require some explanation. The original TRACS-II solution has a cost of 406.14 yet the average cost for the first set of morning solutions is below this figure. This has to do with some limitations in the SCHEDULE program. At the time these experiments were carried out SCHEDULE had a limit of 500 nodes on the branch and bound portion of the ILP process. The original TRACS-II solution exhausted this 500 node limit and the solution with the cost of 406.14 was the best solution found up until that point. Although several of the other solutions generated using the CROSS pre-processor also exhausted the 500 node SCHEDULE branch and bound limit, cheaper solutions were found. This is due to the fact that every solution generated using the CROSS pre-processor will have a different continuous solution. Various heuristics guide the branch and bound search and are influenced by the fractional values in the continuous solution. The complexity of the CROSS pre-processed problems is also reduced so the ILP has fewer variables and constraints to deal with. SCHEDULE was able to occasionally find solutions cheaper than the original TRACS-II solution because its branch and bound search found solutions that the original TRACS-II solutions branch and bound tree did not get to before the 500 node limit was reached. All but one of the schedules produced using CROSS used the minimal number of duties.

8.3.4 R61 Results

The results produced from the bus workings R61 can be found in Table 8.6. The sets of potential duties generated for R61 were produced using the Centre West version of BUILD.

This is the second largest schedule from our data set in terms of the number of running boards contained in the set of bus workings (39). Inspecting the results in Table 8.6 it can be seen that, as in the set of TRAM results, the average cost for the solutions in the first (and second) set of morning results is actually lower than that of the original TRACS-II solution. This is due to the same reason previously discussed with the TRAM dataset

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
Original Result								
100,000	540	–	100%	–	596.42	69.0	3.05	98.42
Morning Results								
52,457	463.8	70%	71.4%	5.4	596.31	69.0	3.33	94.25
50,913	459.8	65%	69.0%	4.6	597.21	69.0	3.24	33.55
46,644	452.6	60%	62.6%	3.6	597.16	69.0	2.59	18.51
44,063	449.6	55%	59.6%	3.0	598.17	69.0	2.44	22.23
Evening Results								
64,977	496.8	70%	72.6%	6.6	596.53	69.0	4.17	118.40
61,355	492.4	65%	68.2%	5.6	597.48	69.0	4.08	44.14
59,342	488.4	60%	64.2%	5.4	597.11	69.0	3.49	35.04
54,715	481.4	55%	57.4%	4.2	598.13	69.0	3.32	31.08
Combined Results								
33,144	420.4	70%	–	–	597.20	69.0	2.01	60.27
29,741	410.2	65%	–	–	597.52	69.0	1.46	12.38
25,487	395.8	60%	–	–	597.56	69.0	1.32	17.00
21,784	384.8	55%	–	–	598.11	69.0	1.17	16.28

Table 8.6: R61 - Morning/Evening/Combined Average Results (2 part duties))

regarding the original TRACS-II solution exhausting all 500 branch and bound nodes. All of the schedules produced using CROSS used the minimum number of duties. One feature in this set of data that stands out more than in others is the significant reduction in the average SCHEDULE time. However, this significant saving is not realised until fewer than 70% of the relief opportunities are selected in both the morning and evening cases.

8.3.5 UMAE Results

CROSS was not designed to directly incorporate three or four part duties into its mealbreak chain generation scheme. Indeed, the CROSS model essentially just traces out two part duties in the data given and keeps track of which relief opportunities are used in their formation. However this does not prevent us from employing CROSS to be used with problems that employ three or four part duties.

The results produced from the bus workings UMAE can be found in Table 8.7. The sets of potential duties generated for UMAE were produced using the Reading version of BUILD.

This set of results was generated using the REDUCE process in order to reduce the running time of SCHEDULE such that experiments could be carried out in a reasonable amount of time.

The asterisks beside the average duties in Table 8.7 indicate that a continuous solution of 133 was found but the branch and bound tree was exhausted. The averages in the relevant rows are for four runs, not five.

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
Original Result								
97,429	873	–	100%	–	1171.06	133.0	76.00	185.15
Morning Results								
71,806	803.8	70%	72.4%	4.0	1162.41	133.0	26.12	57.56
66,258	785.8	65%	66.2%	3.0	1164.10	133.0	22.36	59.24
63,019	778.0	60%	63.6%	3.0	1164.50	133.0	20.48	47.53
53,029	749.6	50%	52.8%	2.2	1165.13	133.0	15.48	34.18
Evening Results								
93,555	858.4	70%	75.2%	4.0	1163.51	133.0	42.24	81.24
89,206	853.6	65%	66.0%	3.2	1165.49	133.0*	38.00	57.27
87,583	842.8	60%	62.0%	3.0	1169.47	133.0	35.12	72.13
84,734	841.0	55%	58.6%	2.8	1168.29	133.0	35.00	66.35
Combined Results								
63,773	787.8	70%	–	–	1164.47	133.0	20.36	42.58
56,178	768.4	65%	–	–	1168.00	133.0	17.48	37.14
50,820	747.8	60%	–	–	1171.54	133.0	15.00	36.45
40,442	711.8	55%	–	–	1171.36	133.0*	10.24	43.49

Table 8.7: UMAE - Morning/Evening/Combined Average Results (2/3 part duties))

The UMAE data set was the largest one considered with CROSS. Additionally it was also the only data set that required a specific change to be made to CROSS in order for it to properly select relief opportunities. The Reading Buses version of the BUILD program only allowed mealbreaks between certain buses (dependent on their bus number). If mealbreaks are restricted in this way then it is clear that CROSS should be modified with this domain specific knowledge otherwise mealbreak chains will be formed that simply cannot take place. Fortunately this is a fairly minor change that needed to affect only the way the domains of $Next$ and $Prev$ variables are initialised. In addition to this the maximum amount of time a driver is allowed on a vehicle during a stretch was varied

between different vehicles. This, again, could be accounted for in CROSS by simply setting the maximum amount of driving time on a vehicle to be the smallest value allowed amongst all of the vehicles. Mealbreak chains will still be formed but the duties will be shorter than might be used in the real world. However, as several sets of relief opportunities are generated by CROSS this will not necessarily detract from the quality of the relief opportunities selected as a whole.

The original solution generated by TRACS-II used a labour file that contained fairly tight constraints. Duties allowed to be generated by BUILD were restricted to ensure they covered a large amount of work thus not exploring the possibility of using shorter[6] duties. The sheer size of the problem to be solved makes this necessary as too many duties will be generated by BUILD if more relaxed constraints were used. Unfortunately this caused a problem for CROSS as it was found that some of the vehicle files generated by CROSS occasionally produced combinations of relief opportunities for which it was impossible for BUILD to generate duties to cover all the work. This is due to the combination of the fact CROSS does not exactly try to produce legal duties while forming mealbreak chains and that the UMAE dataset had some unusual working constraints (like the drivers being restricted to the buses they can work on). However, there is no reason why the labour file parameters to the UMAE data set could not take advantage of the smaller set of relief opportunities and be *slackened*. This would seem to be an advantageous strategy to use for larger schedules because if slacker labour parameters are used then a richer variety of shifts can be generated by BUILD.

The labour file used for CROSS for the UMAE data set had its join-up and minimum duty work content parameters slackened slightly (by about 10%-20%) and the results produced were very encouraging. Again, as seen in Table 8.7, it can be seen that the average cost of the solutions found were less than that of the original solution (bar the final two sets of combined solutions). Additionally the minimum number of duties was used in every solution found using CROSS. The slacker parameter values used are at least partly responsible for these good results.

[6]Possibly less efficient but perhaps better at covering awkward pieces of work.

8.4 Peaked Dataset Results

The next three data sets examined in this section, R207, R77A, and HE01 contain peaks. R207 and R77A have peaks in their morning period whilst HE01 contains its peak in its evening period.

The tables presenting the results of the peaked data sets are slightly different than the ones examined previously. The morning or evening period of work has three separate sets of results generated. Three different ways of calculating the beginning and end times of a peak were investigated.

1. **No Peaks** - CROSS processes the morning set of relief opportunities as normal. No special attention is given to the peak period.
2. **Short Peaks** - The peak period on the vehicle is defined as the block of time in the morning period where the most running boards are active at the same time.
3. **Long Peaks** - Peaks periods are defined as they are in short peaks but are extended by the length of the minimum allowed mealbreak at the beginning and end time of the peak. This extends the peak period by double the minimum mealbreak length.

The combined results section incorporates both the Long Peak results with the morning/evening results (as appropriate).

First, the R207 peaked results will be presented.

8.4.1 R207 Results

The results produced from the bus workings R207 can be found in Table 8.8. The sets of potential duties generated for R207 were produced using the generic version of BUILD.

All of the results generated used the minimum number of duties and the average costs of the schedules are close to the original TRACS-II value. The most important feature to be

found in the morning sets of results is the average cost for the schedules generated. The long peaked set of data produced cheaper results than the short peaked data which, in turn, was cheaper than the results generated when R207 was treated as if it had no peaks. This supports the hypothesis that mealbreak chains are best avoided through the peak period of a set of bus workings.

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
Original Result								
23,004	216	–	100%	–	378.43	39.0	0.22	1.43
Morning Results (With Long Peak)								
12,511	200.4	70%	71.8%	5.4	379.05	39.0	0.11	2.00(1)
11,834	197.4	65%	66.6%	3.4	379.41	39.0	0.10	1.18
11,440	195.6	60%	62.2%	3.6	379.46	39.0	0.10	1.49(1)
10,609	192.6	55%	57.2%	2.4	380.22	39.0	0.10	1.25
Morning Results (With Short Peak)								
13,043	201.6	70%	73.2%	3.6	379.24	39.0	0.12	1.15
12,370	199.0	65%	68.0%	2.4	380.34	39.0	0.11	1.36
11,715	196.0	60%	62.0%	2.4	380.58	39.0	0.10	1.24
11,158	193.4	55%	57.4%	2.0	381.16	39.0	0.10	1.09(1)
Morning Results (With No Peak)								
13,079	201.2	70%	72.6%	3.6	380.06	39.0	0.11	1.24(1)
12,461	198.6	65%	68.0%	2.8	379.50	39.0	0.11	1.12
11,729	196.0	60%	62.0%	2.4	380.24	39.0	0.11	1.26
11,177	193.2	55%	57.4%	2.0	381.26	39.0	0.11	1.31
Evening Results								
16,589	198.6	70%	73.0%	4.2	380.39	39.0	0.14	1.54
16,015	197.0	65%	67.8%	3.0	381.09	39.0	0.14	1.53
15,187	194.2	60%	63.0%	2.6	381.35	39.0	0.14	1.57(1)
14,296	191.2	55%	57.6%	2.0	382.25	39.0	0.13	1.12
Combined Results								
8,631	183.0	70%	–	–	381.00	39.0	0.07	1.13
7,767	178.6	65%	–	–	382.06	39.0	0.07	0.54
7,138	174.0	60%	–	–	382.53	39.0	0.06	0.51
6,077	167.6	55%	–	–	383.55	39.0	0.05	0.26(1)

Table 8.8: R207 - Morning/Evening/Combined Average Results (2 part duties))

8.4.2 R77A Results

The results produced from the bus workings R77A can be found in Table 8.9. The sets of potential duties generated for R77A were produced using the generic version of BUILD.

This second set of peaked data is considerably larger than the R207 data set. Examining the results generated for the morning period it can be seen, as in the previous R207 data set, that when the data is treated as if it contained the longer peak that the average cost of the morning results is cheapest.

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
Original Result								
76,906	352	–	100%	–	490.26	53.0	2.28	88.31
Morning Results (With Long Peak)								
54,558	310.0	70%	74.4%	4.8	492.00	53.0	1.07	53.29
53,039	304.0	65%	67.6%	3.4	494.42	53.0	1.04	57.48
51,201	300.6	60%	63.6%	3.2	498.26	53.0	1.02	55.35
48,992	296.2	55%	58.2%	2.8	499.34	53.0	0.59	49.42
Morning Results (With Short Peak)								
55,249	309.2	70%	73.4%	4.0	497.52	53.0	1.07	55.40
53,585	304.4	65%	67.8%	3.8	494.47	53.0	1.03	57.15
51,241	299.8	60%	62.6%	3.0	498.54	53.0	1.02	56.11
48,551	295.2	55%	57.2%	2.8	497.18	53.0	0.58	55.19
Morning Results (With No Peak)								
54,452	308.2	70%	72.2%	3.4	496.09	53.0	1.05	61.08
52,472	304.4	65%	68.0%	3.0	499.39	53.0	1.03	54.02
50,505	300.8	60%	63.8%	2.8	501.15	53.0	1.01	52.06
47,559	295.4	55%	57.4%	2.0	495.11	53.0*	0.57	53.34
Evening Results								
51,023	324.6	70%	73.6%	5.0	492.30	53.0*	1.01	65.13
49,378	317.8	65%	66.8%	4.0	495.50	53.0	0.56	59.13
47,948	313.2	60%	62.8%	3.4	496.07	53.0	0.56	67.56
47,722	308.6	55%	58.6%	3.0	497.05	53.0	0.54	53.59
Combined Results								
34,835	282.6	70%	–	–	498.08	53.0	0.39	45.36
28,979	269.8	65%	–	–	500.46	53.0	0.32	29.19
26,048	261.8	60%	–	–	508.30	53.0	0.29	32.45
22,763	253.0	55%	–	–	511.57	53.0	0.26	20.50

Table 8.9: R77A - Morning/Evening/Combined Average Results (2 part duties))

The asterisk beside the average number of duties used in the morning periods with no peak indicates that one of the runs did not find a solution in the branch and bound phase within the allotted 500 nodes (but did have a continuous solution indicating the need for 53 duties). Like in the UMAE dataset the average is for four runs, not five.

8.4.3 HE01 Results

The results produced from the Heathrow Express data set (HE01) can be found in Table 8.10. The sets of potential duties generated for HE01 were produced using the generic version of BUILD. HE01 is unique in comparison to the other datasets as it is a small train schedule that is very 'bus like' in structure.

The HE01 data set is one of two that use 3 or 4 part duties in the set of potential duties generated by BUILD (in this case both 3 and 4 part duties are used). Additionally it is the only schedule to contain an evening peak.

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
Original Result								
43,594	158	–	100%	–	123.52	14.0	5.21	11.56
Morning Results								
34,929	143.8	70%	71.6%	8.2	123.34	14.0	4.56	6.36
33,436	140.2	65%	67.4%	7.0	123.27	14.0	4.34	4.51
31,657	137.6	60%	62.2%	6.2	123.12	14.0	4.03	3.17
30,758	135.4	55%	58.0%	5.4	123.17	14.0	4.05	3.45
Evening Results (With Long Peak)								
34,049	142.4	70%	71.8%	10.0	122.16	14.0	4.18	2.49
32,267	139.8	65%	67.6%	8.0	121.48	14.2	4.01	4.48(1)
29,902	136.6	60%	62.2%	6.2	121.58	14.2	3.39	4.05(3)
28,405	133.4	55%	55.8%	5.0	122.48	14.2	3.16	3.09(1)
Evening Results (With Short Peak)								
34,272	142.6	70%	72.2%	9.0	123.09	14.0	4.11	3.08
32,397	140.0	65%	67.8%	7.2	122.48	14.0	3.53	3.00
30,137	136.8	60%	62.4%	5.2	122.49	14.0	3.26	2.22
28,340	133.8	55%	56.6%	5.2	122.33	14.2	3.07	3.46
Evening Results (With No Peak)								
33,279	142.6	70%	71.8%	9.0	123.31	14.0	3.57	4.26
31,079	139.6	65%	67.0%	7.8	122.31	14.0	3.32	2.46
28,956	136.8	60%	62.2%	5.6	122.43	14.0	3.10	3.16
29,027	134.6	55%	58.2%	5.4	123.41	14.2	3.10	2.52
Combined Results								
25,449	127.2	70%	–	–	122.59	14.0	2.50	1.57(2)
22,258	122.0	65%	–	–	122.34	14.2	2.11	1.45(1)
18,334	116.4	60%	–	–	123.23	14.2	1.35	1.46(1)
16,188	110.8	55%	–	–	123.47	14.2	1.19	0.52(3)

Table 8.10: HE01 - Morning/Evening/Combined Average Results (1/2/3/4 part duties))

Perhaps the most remarkable feature of the HE01 set of results is that all of the averaged schedule costs are *less than* the cost of the solution given when all relief opportunities are allowed in the BUILD process. Unlike the UMAE dataset the labour file has not been slackened in any way so the rules governing duty construction for TRACS-II are the same between the original and CROSS pre-processed results.

There are two possible explanations for this. The first has to do with the TRACS-II BUILD process. BUILD, when generating 3 or 4 part duties, does not generate all such duties. Many are heuristically deemed to be unlikely to be useful and discarded. When relief opportunities have been removed from the bus workings some of the previously rejected duties will be included by BUILD. Unfortunately, this does not explain all the HE01 results. Several of the HE01 schedules were examined and only a few of them contained any of these previously rejected duties. Most of the schedules produced were cheaper than the original TRACS-II result regardless as to whether or not these previously discarded duties were present in the solution.

The real reason these cheaper solutions can be generated is due to TRACS-II's SCHEDULE component. The reasons are not dissimilar to the ones presented for the cheaper schedules produced in the TRAM and R61 datasets. There is one difference however; the original TRACS-II solution for HE01 did not exhaust its entire branch and bound tree. Due to heuristics in the brand and bound phase cheaper solutions were pruned from the search space and omitted. The different continuous solutions for the HE01 CROSS results enabled SCHEDULE to find cheaper solutions. The fact a smaller set of constraints/variables were used may have had some influence on this. Regardless, all of the solution costs presented for HE01 are acceptable.

The pattern in the results for the evening peak is similar to the previous two datasets examined. The long peak produced the cheapest solutions but the difference between the short peak and no peak results is very minimal. However, the peak in the HE01 schedule is somewhat different than the others in that the peak vehicles are all very short (20 minutes long) and the peak period itself is only about 5 minutes long making the CROSS produced

vehicles files for the short peaks and non peaked runs very similar. The long mealbreak set of results has a more profound impact producing cheaper schedules like we have seen in the previous two datasets. The 5 minute peak may seen insignificant but the short peak buses are all scheduled around the same time with the temporal overlap occurring in about the middle thus the consideration of the long peak made a difference.

CROSS, when applied to this data set, did not always result in the generation of a duty schedule using the optimal number of duties. This can be attributed to the unusual peak. There are not many sensible ways of covering the short buses. By examining the schedules produced it was observed that the extra duty is a one part duty that takes a vehicle back to the depot at the end of the day (4-5 hrs work) whilst a less efficient duty is formed to cover part of the peak buswork. Table 8.10 provides some evidence to support this as it can be seen that the result set generated for the long peak had the most schedules using 15 duties. CROSS specifically avoided selecting relief opportunities temporally close to these peak buses in the long peak results. This can lead to the selection of relief opportunities that does not enable BUILD to produce duties to cover the peak buses in a sensible way. This accounts for why there are more results generated using 15 instead of 14 duties for the long peak results. The results for the 70-75% range were all safe indicating that with the slightly higher number of relief opportunities selected it is more likely this peak work will be covered in a sensible way while still gaining the benefits from relief opportunity selection.

8.5 The Effect of the REDUCE Process

As outlined in Chapter 4, TRACS-II has an optional procedure called REDUCE. This procedure, when selected, is invoked after a continuous solution to the set covering problem is solved. Any relief opportunities that are not used in the continuous solution are, effectively, removed from the problem. This also has two effects on the branch and bound phase [40]:

1. The number of constraints in the branch and bound phase is reduced. When a relief opportunity is removed from the problem, the two pieces of work are merged into one.

2. Any duties using a relief opportunity that were removed by REDUCE may also be removed this eliminating shift variables from the problem.

In practice, REDUCE removes about 50% of the relief opportunities (merges adjacent pieces of work). Whilst CROSS does not remove nearly as many relief opportunities there does seem to be a reasonable overlap in the relief opportunities removed.

On the surface it may seen that the REDUCE process is superior to the CROSS process. In practice it would probably be ideal to use both methods, especially for larger schedules; this was not done with our results, however, because REDUCE and CROSS perform similar tasks. A clearer picture can be given of the effectiveness of CROSS if REDUCE is not used in the SCHEDULE component (bar the UMAE dataset for time reasons). REDUCE selects relief opportunities to discard from information gained from the continuous solution to the ILP. CROSS has no such knowledge and can't take advantage of this information as it only forms mealbreak chains given a set of bus workings. However, for large schedules like UMAE, it is not possible to generate as rich a variety of schedules as would be possible when the CROSS pre-processor is used (as labour rules can be relaxed when relief opportunities are taken away before the BUILD process). REDUCE is entirely dependent on what duties are generated by BUILD and can only act from within the framework of these duties.

By examining several of the CROSS runs it was found that around 50% of the relief opportunities removed by REDUCE (from the morning/evening periods) from the original unaltered data were also removed by CROSS process (when 70-75% of the relief opportunities are set as the target percentage).

Examining the times and costs for the combined results for the datasets considered (in the 70-75% bracket) we can see that in 4 of the 8 datasets, the schedules produced by

TRACS-II using REDUCE were always cheaper than the average costs of CROSS processed schedules. Like the average schedule costs, in 4 of the 8 datasets the schedules produced by TRACS-II using REDUCE were done so in a shorter period of time than if CROSS were used. Two important observations that can be made about the 4 cases where CROSS produced some cheaper average schedule costs. First, 2 of these 4 cases were datsets involving 3 or 4 part duties. Secondly, the others were with datasets that exhausted the 500 node branch and bound tree in SCHEDULE with the original TRACS-II dataset. This indicates that CROSS can be an effective tool when 3 or 4 part duties are employed or when SCHEDULE exhausts its search tree.

8.6 Summary

This chapter presents the results produced from applying CROSS to 8 different scheduling problems. Several general observations can be made about the results produced using CROSS. In virtually every case it was found that the minimum number of duties were used in the CROSS pre-processed results. Additionally, the average time taken by BUILD as well as SCHEDULE was also considerably less. The average costs of the schedules generated were acceptable and in some cases were found to be cheaper than the original TRACS-II results. In all datasets, except UMAE, the same version of CROSS was used to pre-process the data. Only a minor modification was required for the UMAE dataset. Schedules containing peaks were found to have better solutions produced when the peak period was modelled as a long peak. Overall, the 70%-75% relief opportunity selection range was found to be safe to use in all cases.

Two of the original two part duty TRACS-II solutions exhausted the 500 node branch and bound search in SCHEDULE. It was found that CROSS occasionally produced cheaper schedules for these datasets. This was due to the branch and bound phase searching a different part of the solution space for the CROSS produced results.

CROSS usually produced cheaper solutions when three or four part duties were used. The

UMAE dataset was large and had a fairly restrictive set of labour agreement rules. The approach used with UMAE was to slacken these rules so that a richer variety of duties would be available to SCHEDULE. This helped to enable TRACS-II to produce cheaper schedules for this dataset. The HE01 dataset also produced cheaper solutions even though the labour file was not modified. This was attributed mainly to the branch and bound phase of SCHEDULE. No schedules using two part duties produced cheaper schedules than the original TRACS-II solutions (except those exhausting the 500 node branch and bound limit). As TRACS-II generates many more ways of covering work in schedules using three and four part duties this is not unexpected as it will be more difficult to find the best solution in such cases (as demonstrated with the HE01 results).

It was found that the REDUCE process selected some of the same relief opportunities that CROSS does. REDUCE and CROSS both influence the SCHEDULE ILP in a similar way in that they both reduce the number of constraints and variables found in the set covering ILP. However, REDUCE exploits information gathered from the continuous solution while CROSS selects relief opportunities solely from forming mealbreak chains in the bus workings.

CROSS has been successfully applied to several types of problems of varying sizes. CROSS would probably not be of much benefit for smaller scheduling problems unless they employed 3 or 4 part duties (such as HE01). Small schedules using only 2 part duties can generally have a rich enough set of potential duties generated by BUILD such that a very good solution can be produced in a reasonable amount of time. CROSS would speed up the solution time but the final schedule produced would most likely be slightly more expensive than if the entire set of original relief opportunities was left intact. Additionally, the REDUCE process seems to work just as well in these situations. Large scheduling problems where strict parameters must be enforced on the duties produced by BUILD would be where a pre-processor like CROSS would be most effective. It would enable the users to use less strict parameters regarding the duties formed by BUILD thus enabling a richer variety of potential duties to be produced by BUILD and therefore better solutions may be produced. The UMAE dataset is an excellent example of this.

Chapter 9

Summary and Conclusions

9.1 Introduction

This chapter summarizes what has been presented and reported in this thesis. A discussion on future work that could be undertaken as a result of this work is also given. How one might apply the CROSS system in practice is described and a summary of the research achievements is presented.

9.2 Summary

This thesis has described the bus driver scheduling problem itself and the TRACS-II mathematical programming system designed to solve it. Mathematical programming has been the most successful technique applied to problems such as bus, rail and air duty scheduling. However, some problems in practice can still be problematic due to their size or their labour agreement rules.

The CROSS pre-processor has been described. CROSS helps to reduce the complexity of the problem solved by TRACS-II by trying to select only those relief opportunities that are likely to be beneficial in the scheduling process. The motivation for CROSS comes

from the hypothesis that schedules containing good mealbreak chains should require fewer duties than those not containing good mealbreak chains. Constraint programming was chosen as the methodology to employ with the design and implementation of CROSS. The expressiveness and ability to model the problem was important and made the definition of constraints between the variables involved logical and effective with respect to the constraint propagation gained during the search for a solution to the CSP.

Little research has been carried out on the topic of mealbreak chains themselves. Fores [38] reports her findings on mealbreak chains and their beneficial properties but concluded that it would be extremely unlikely that an automatic system could be built that would create a duty schedule based only on finding mealbreak chains. Earlier heuristic systems, such as TRACS [92], made some effort to incorporate mealbreak chains into the schedules they created but suffered from the fact the optimisation performed was of a local nature and was too inflexible. This was borne in mind when the CROSS CSP was designed and a flexible and generic approach was used.

The CROSS CSP itself was designed to trace out mealbreak chains during the morning period of a set of bus workings. This was also applied to the evening periods. The relief opportunities traced out in the mealbreak chains are selected and the CROSS CSP is solved several times. The union of all of the relief opportunities selected by CROSS is calculated. A new vehicle file is then produced that uses only the selected relief opportunities in its morning and/or evening periods. TRACS-II uses this new vehicle file, with the reduced set of relief opportunities, in order to generate a duty schedule.

Because the CROSS CSP is solved several times there is a good selection of sets of relief opportunities that can be used by TRACS-II. The union of these sets will enable additional mealbreak chains to be formed that were not explicitly selected by CROSS; this is where the solutions of the CROSS CSP gain their flexibility and is why CROSS is effective.

The results reported in this thesis support the hypothesis about the benefits of mealbreak chains and demonstrate that CROSS can be applied to a wide variety of schedules and labour rules with success. The results also demonstrate that avoiding mealbreak chains

around the peak period of a schedule is beneficial.

9.3 Further Work

There are still some areas regarding relief opportunity selection, and further experimentation with CROSS itself, that could be investigated further.

9.3.1 Morning vs Evening Periods

The CROSS system was designed for a morning period of a set of bus workings. Although it was found that CROSS could also be applied to the evening period of a set of bus workings the evening and morning periods in a set of bus workings do tend to have different features. Investigating this relationship further in the context of relief opportunity selection could be beneficial to future work with CROSS.

9.3.2 Duty Properties

Investigation into potentially useful properties of three and four part duties may prove useful. Relief opportunities suitable for three and four part duties are not explicitly created in CROSS but the interaction between several runs of the CROSS algorithm creates suitable relief opportunities to enable BUILD to construct useful duties. CROSS may be able to be extended to select certain relief opportunities that would be of particular use to the construction of three/four part duties in various contexts. For example, a set of short running boards contained in a set of bus workings often, ideally, have their work covered by three or four part duties.

The BUILD component of TRACS-II uses heuristics that discard three/four part duties that are unlikely to be useful. It was found that when CROSS is applied to a set of bus workings that three/four part duties that were not considered in the original problem are

constructed. Some of these duties were actually present in schedules produced that were cheaper than the original TRACS-II produced ones. Although this is probably not the main reason these schedules were cheaper it may be worthwhile to revisit these heuristics used in BUILD in order to try to identify in what context it might be useful to keep some of these duties that would otherwise be discarded.

9.3.3 Variable and Value Ordering

The method by which the random variable and value ordering is carried out is somewhat simplistic but effective. It may be possible to enhance these procedures to be dynamic with respect to the size and shape (and perhaps other features to be found) of the schedule; this could potentially warrant future research.

As the goal of the CROSS CSP is to select relief opportunities by tracing out mealbreak chains, domain specific knowledge, or heuristics, will be required[1].

9.3.4 Other types of Duty Scheduling Problems

The research done in this thesis has been domain specific in that it was only applied to the bus duty scheduling problem and a simple rail problem. It would be useful to see if the techniques and ideas acted upon in this thesis would be applicable to the domain of other types of duty scheduling problems. There would undoubtedly be differences between them that would need to be addressed. For example, in rail driver scheduling problems, the stops on the train can be great distances apart and additional constraints may be present such as ensuring that a driver finishes his duty at the same relief opportunity (or a subset of relief opportunities) that the driver started at. Constraints regarding what trains can be driven by the same driver are also present, so called *traction knowledge*. Some investigation would need to be carried out in order to determine how to model the additional constraints found in such problems.

[1]Variable and Value ordering are very closely related in the context of *Next* and *Prev* variables as the value assigned to a *Next* variable often represents the next variable to be bound.

9.4 Potential Applications of the CROSS System

As reported in Chapter 8, CROSS was found to be reasonably effective with all of the data examined. There were three cases in particular where the CROSS pre-processor was found to enable TRACS-II to produce better solutions than if all the relief opportunities were left intact and, thus, these would be the target scenarios where CROSS would probably be beneficial to apply.

The first is the case where three or four part duties are employed. Since several solutions to the CROSS CSP are generated BUILD is supplied with a rich enough variety of relief opportunities from which three or four part duties can be generated. Additionally, since BUILD creates its set of potential duties from a reduced set of relief opportunities, three and four part duties that would not normally be generated are created and occasionally used.

If a large schedule is being examined it can often be the case that the labour file contains tight constraints in order to strictly limit the number of duties generated. With the application of CROSS the constraints in the labour file can be slackened. This enables BUILD to generate a richer variety of duties than was previously available for the SCHEDULE process. This aids SCHEDULE in finding, potentially, a better solution to the problem.

The third case where CROSS provided better solutions is where the SCHEDULE module exhausted its branch and bound search space and may not have found the best solution available. The reduction of relief opportunities reduces both the number of variables and the number of constraints in the SCHEDULE ILP. This, in turn, reduces the search space that the ILP examines thus enabling SCHEDULE to find different and possibly better solutions due to the fact the associated branch and bound search will, more than likely, search a different part of the solution space.

One additional benefit is CROSS's ability to reduce the complexity of SCHEDULE's ILP and, therefore, reducing the amount of time it takes to find a solution. If a scheduling problem takes too long to solve then using either CROSS or SCHEDULE's REDUCE

feature (or both) should be considered in order to reduce the time it takes SCHEDULE to finish.

In the newer version of TRACS-II the portion of the system that takes the most CPU time is the BUILD component. The application of CROSS would produce immediate speed benefits with the new BUILD from the reduction of relief opportunities from which BUILD can construct duties.

CROSS was designed to be reasonably generic but, as was seen in the UMAE dataset, sometimes modifications may have to be made. The modification made for UMAE was essential in that legal mealbreak chains would not have been formed otherwise. Similar situations may arise with future use due to unforeseen constraints.

The CROSS system should be applicable to virtually any mathematical programming driver scheduling package. CROSS is a pre-processing stage that changes the input data (bus workings); the scheduling system would then take the modified set of bus workings and proceed as usual. This could be extended further to systems that employ column generation where additional duties are generated in the column generation phase (for example in systems like CREW-OPT [31, 103]). CROSS could be employed during this duty generation phase by pre-processing the bus workings before additional duties are constructed (This, indeed, itself could be done in a cyclical fashion where several sets of duties can be created, each from a set of bus workings processed by CROSS).

In many large scheduling problems (bus and rail) sometimes problems have to be *sub-divided*; that is, the problem is so large it must be decomposed into several sub problems. It may be possible that an application of CROSS to the entire original problem can reduce its size sufficiently that sub-division is not necessary.

Although CROSS itself is not entirely suitable for rail scheduling, there are circumstances in rail duty scheduling where such an approach could be beneficial. In some rail problems there are situations where *time-windows* arise. Occasionally when a rail driver finishes a spell of work on a train the train must be attended for some fixed amount of time (perhaps

there may be a short period of time before it heads out again and the train must remain attended whilst it is kept running during this time). This fixed amount of time is referred to as a time window. Essentially the driver can be relieved at any time during this time window. If the time window was for, say 15 minutes, then essentially there are 15 relief opportunities present in this time window from a duty scheduling perspective. Clearly if a schedule has many of these time windows the complexity of the problem is increased by a substantial amount as BUILD will create many similar duties. A procedure such as CROSS would be useful to apply to such a scenario in order to reduce these sets of time windows whilst ensuring legal mealbreak chains (thus duties) can be created using them.

9.5 Research Achievements

The work carried out in this research has expanded the knowledge in the domain of duty scheduling, specifically that of bus driver scheduling and relief opportunity selection. A technique for creating, with a random element, mealbreak chains from portions of a set of bus workings has been presented.

Constraint programming has been shown to be very valuable as an *aid* to constructing good driver schedules even though constraint programming itself has not been very successful at producing duty schedules when used on its own. It may be the case that with other problems where constraint programming on its own is not suitable the technique may be useful as an aid.

The CROSS pre-processor summarises the achievements. We have shown that mealbreak chain selection successfully enables the reduction of the number of relief opportunities that need to be supplied to a mathematical programming system such as TRACS-II [38, 40, 39, 128, 70] without significantly affecting the quality of solutions produced. In some cases it was found that CROSS actually enabled TRACS-II to find solutions of a higher quality than those that have already been identified. Another benefit achieved from relief opportunity selection was the resulting reduction in complexity of the resulting ILP in

TRACS-II. This enables the possibility of tackling larger problems as well as giving the option of slackening labour parameters enabling a richer variety of duties to be produced in the BUILD process for large schedules.

The empirical results produced regarding the selection of relief opportunities in peaked schedules is worthy of note. This clearly shows the link between the relief opportunity selection and peak periods in the schedule.

The CROSS system has been shown to be extremely generic and flexible in that schedules with vastly different constraints and features were all successfully solved using the same CROSS pre-processor before applying TRACS-II (with the exception of the one minor enhancement made for the UMAE dataset). This includes applying CROSS to the evening period of a set of running boards as well as using CROSS to select relief opportunities with sets of running boards whose labour file allows the use of three or four part duties. This is particularly important as three and four part duties are often used in real world driver scheduling problems. Running boards having both their morning and evening periods processed by CROSS were also reported with satisfactory results. Most importantly, in almost every case, the CROSS pre-processor was able to select relief opportunities that enabled TRACS-II to create duty schedules using the minimum number of duties required.

Bibliography

[1] Optimising traincrew allocation can deliver step-change performance improvement. PA Consulting brochure, 1998.

[2] J. Antes. Structuring the Process of Airline Scheduling. In *Operations Research Proceedings, 1997*, Berlin, Heidelberg, 1998.

[3] F. Bacchus and P. van Beek. On the Conversion between Non-Binary and Binary Constraint Satisfaction Problems. In *Proceedings of AAAI-98*, pages 311–318, July 1998.

[4] M. O. Ball, L. D. Bodin, and J. Greenberg. Enhancements to the RUCUS-II Crew Scheduling System. In Rousseau [102], pages 279–293.

[5] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling - Applying Constraint Programming to Scheduling Problems*, volume 39 of *International Series in Operations Research and Management Science*. July 2001.

[6] D. Beasley, D. R. Bull, and R. R. Martin. An Overview of Genetic Algorithms: Part 1, Fundamentals. *University Computing*, 15(2):58–69, 1993.

[7] D. Beasley, D. R. Bull, and R. R. Martin. An Overview of Genetic Algorithms: Part 2, Research Topics. *University Computing*, 15(4):170–181, 1993.

[8] C. Bessiere and M.-O. Cordier. Arc-Consistency and Arc-Consistency Again. In *Proceedings of AAAI-93*, pages 107–113, 1993.

[9] C. Bessiere, E. C. Freuder, and J.-C. Regin. Using constriant metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107:125–148, 1999.

[10] J. Y. Blais and J.-M. Rousseau. Overview of HASTUS Current and Future Versions. In Daduna and Wren [26], pages 175–187.

[11] J. E. Borrett and E. Tsang. Observations on the Usefulness of Arc Consistency Preprocessing. Technical Report CSM-236, University of Essex, March 1995.

[12] D. Brélaz. New Methods to Color the Vertices of a Graph. *Communications of the ACM*, pages 251–256, 1979.

[13] P.J. Cameron. *Combinatorics: Topics, Techniques, Algorithms.* Cambridge University Press, first edition, 1994.

[14] B. Catlow. *Quantitative Methods for Computing Students.* DP Publications, 1993.

[15] M. Chamberlain and A. Wren. Developments and Recent Experience wiht the BUSMAN and BUSMAN II Systems. In Desrochers and Rousseau [32], pages 1–15.

[16] P. Charlier and H. Simonis. A system for train crew scheduling. Abstract in DIMACS Workshop on constraint programming and large scale discrete optimisation, 1998.

[17] P. Charlier and H. Simonis. A system for train crew scheduling. Slides for DIMACS Workshop on constraint programming and large scale discrete optimisation, 1998.

[18] R. Clement and A. Wren. Greedy Genetic Algorithms, Optimizing Mutations and Bus Driver Scheduling. In Daduna et al. [24], pages 213–235.

[19] M.C. Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41:89–95, 1989.

[20] T. H. Corman, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[21] D. Costa. An Evolutionary TABU Search Algorithm and the NHL Scheduling Problem. *INFOR*, 33(3):161–178, August 1995.

[22] S.D. Curits, B. M. Smith, and A. Wren. Forming Bus Driver Scheduling usin Constraint Programming. In *Practical Application of Constraint Technologies and Logic*

Programming (PACLP99), pages 239–254. The Practical Application Company Ltd, 1999.

[23] S. D. Curtis. *Constraint Satisfaction Approaches to Bus Driver Scheduling.* PhD thesis, University of Leeds, Februrary 2000.

[24] J. R. Daduna, I. Branco, and J. M. P. Paixão, editors. *Computer-Aided Transit Scheduling*, Proceedings of the Sixth International Workshop on Computer-Aided Scheduling of Public Transport. Springer-Verlag, 1995.

[25] J. R. Daduna and M. Mojsilovic. Computer-Aided Vehicle and Duty Scheduling Using the HOT Programme System. In Daduna and Wren [26], pages 133–146.

[26] J. R. Daduna and A. Wren, editors. *Computer-Aided Transit Scheduling*, Proceedings of the Fourth International Workshop on Computer-Aided Scheduling of Public Transport. Springer-Verlag, 1988.

[27] P. David. A Constraint-Based Approach for Examination Timetabling Using Local Repair Techniques. In E. Burke and M. Carter, editors, *Practice and Theory of Automated Timetabling*, Proceedings of the Second International Conference on the Practice and Theory of Automated Timetabling, pages 169–186. Springer-Verlag, 1998.

[28] L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[29] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[30] R. Dechter and D. Frost. Backtracking algorithms for constraint satisfaction problems. Technical report, UCI, 1997.

[31] M. Descrochers and F. Soumis. CREW-OPT: Crew Scheduling by Column Generation. In Daduna and Wren [26], pages 83–90.

[32] M. Desrochers and J.-M. Rousseau, editors. *Computer-Aided Transit Scheduling*, Proceedings of the Fifth International Workshop on Computer-Aided Scheduling of Public Transport. Springer-Verlag, 1992.

[33] Y. Deville and P. van Hentenryck. An efficient arc consistency algorithm for a class of CSPs. In *Proceedings International Joint Conference on AI*, 1991.

[34] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26:29–41, 1996.

[35] J.C. Falkner and D.M. Ryan. EXPRESS: Set Partitioning for Bus Crew Scheduling in Christchurch. In Desrochers and Rousseau [32], pages 359–378.

[36] H. Fang. Investigating Genetic Algorithms for Scheduling. Master's thesis, University of Edinburgh, 1992.

[37] M. L. Fisher. The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Science*, 27(1):1–18, January 1981.

[38] S. Fores. *Column Generation Approaches to Bus Driver Scheduling.* PhD thesis, University of Leeds, March 1996.

[39] S. Fores and L. Proll. Driver Scheduling by Integer Linear Programming - The TRACS II Approach. In P Bourne, M Ksouri, and A El Kamel, editors, *Proceedings CESA'98 Computational Engineering in Systems Applications*, volume 3 Symposium on Industrial and Manufacturing Systems, pages 213–218, 1998.

[40] S. Fores, L. Proll, and A. Wren. An Improved ILP System for Driver Scheduling. In Wilson [130], pages 43–61.

[41] S. Fores, L. Proll, and A. Wren. TRACS to Better Schedules. Technical Report 2001.03, School of Computing, University of Leeds, January 2001.

[42] P. Forsyth and A. Wren. An Ant System for Bus Driver Scheduling. In Wilson [130], pages 405–421.

[43] E. C. Freuder. Synthesizing Constraint Expressions. *Communications of the ACM*, 21(11):958–966, 1978.

[44] D. Frost and R. Dechtedr. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 572–578, 1995.

[45] D. Frost and R. Dechter. In search of the best constraint satisfaction search. In *Proceedings AAAI-94*, pages 301–306, 1994.

[46] G. Garnier. MERCATOR and HASTUS-MACRO Computerization and Changing Working Conditions for RATP Bus Drivers. In Rousseau [102], pages 137–144.

[47] J. Gaschnig. A General Backtrack Algorithm that Eliminates Most Redudant Tests. In *Proceedings IJCAI-77*, 1977.

[48] J. Gaschnig. Experimental case studies of backtrack vs. Waltz-type vs. new algorithms for satisficing assigment problems. In *Proceedings Second National Conference of the Canadian Society for Computational Studies of Intelligence*, pages 268–277, 1978.

[49] J. Gaschnig. Performance measurement and analysis of certain search algorithms. Technical Report CMU-CS-79-124, Carnegie-Mellon University, Pittsburg, January 1979.

[50] I. P. Gent, E. MacIntyre, P. Prosser, B. M. Smith, and T. Walsh. An Empirical Study ofc Dynamic Variable Ordering Heuristics for the Constraint Satisfaction Problem. In E. C. Freuder, editor, *Constraint Processing*, LNCS 1118, pages 179–193. Springer-Verlag, 1996.

[51] S. W. Golumb and L. D. Baumert. Backtrack Programming. *Journal of the ACM*, 12:516–524, 1965.

[52] N. Guerinik and M. Van Caneghem. Solving Crew Scheduling Problems by Constraint Programming. In U. Montanari and F. Rossi, editors, *Principles and Practice*

of Constraint Programming, CP '95, First International Conference, CP '95 Cassis, France, Proceedings, pages 481–498. Springer-Verlag, September 1995.

[53] C. Halatsis, P. Stamatopolous, I. Karali, T. Bitsikas, G. Fessakis, A. Schizaz, S. Sfakianakis, C. Fouskakis, T. Koukoumpetsos, and D. Papageorgiou. Crew Scheduling Based on Constraint Programming: The PARACHUTE Experience. In *Proceedings of the 3rd Hellenic-European Conference on Mathematics and Informatics (HERMIS '96)*, pages 424–431, 1996.

[54] N. Hamer and L. Seguin. The HASTUS System: New Algorithms and Modules for the 90s. In Desrochers and Rousseau [32], pages 17–29.

[55] C. Han and C. Lee. Comments on Mohr and Henderson's Path Consistency Algorithm. *Artificial Intelligence*, 36:125–130, 1988.

[56] R. Haralick and G. Elliott. Increasing Tree Search Effciency for Constraint Satsifaction Problems. *Artificial Intelligence*, 14:263–313, 1980.

[57] T. Hartley. A glossary of terms in bus and crew scheduling. In Wren [132], pages 353–359.

[58] P. van Hentenryck. *Constraint Satisfaction in Logic Programming.* Logic Programming Series. MIT Press, 1989.

[59] P. M. Hildyard and H. V. Wallis. Advances in Computer-Assisted Runcutting in North America. In Wren [132], pages 183–192.

[60] J. Hoffstadt. Computerized Vehicle and Driver Scheduling for the Hamburger Hochbahn Aktiengesellschaft. In Wren [132], pages 35–52.

[61] J. H. Holland. *Adaptation in Natural and Artificial Systems.* Massachusetts Institute of Technology Press, 1975. First edition University of Michigan Press.

[62] ILOG. *ILOG Solver - Reference Manual Version 3.2*, July 1996.

[63] ILOG. *ILOG Solver - User Manual Version 3.2*, July 1996.

[64] *ILOG Solver and ILOG Schedule First International Users's Conference*, ILOG Solver and ILOG Schedule First International Users's Conference, July 10-11, Abbaye des Vaux de Cernay, France, 1995.

[65] ILOG Solver and ILOG Schedule Second International Users's Conference, July 9-10, Paris, France, 1996.

[66] ILOG Solver and ILOG Schedule Third International Users's Conference, France, 1997.

[67] Fourth ILOG International Users Meetings, October 5-6, Paris, France, 1998.

[68] S. Kikuchi and M. Arimura. Use of Genetic Algorithms to Schedule the Specialized Transporation Vehicles. In Wilson [130], pages 322–336.

[69] A. Kwan and E. Tsang. Minimal Forward Checking with Backmarking. Technical Report CSM-260, University of Essex, March 1996.

[70] A. S. K. Kwan. *Train driver scheduling.* PhD thesis, University of Leeds, 1999.

[71] A. S. K. Kwan, R. S. K. Kwan, M. E. Parker, and A. Wren. Producing Train Driver Schedules under differing Operating Strategies. In Wilson [130], pages 129–154.

[72] A. S. K. Kwan, R. S. K. Kwan, and A. Wren. Driver scheduling using Genetic Algorithms with embedded combinatorial traits. In Wilson, editor, *Computer-Aided Transit Scheduling*, pages 81–102. Springer-Verlag, 1999.

[73] A. S. K. Kwan, S. K. Kwan, M. E. Parker, and A. Wren. Proving the Versatility of Automatic Driver Scheduling on Difficult Train and Bus Problems. Technical Report 2000.12, University of Leeds, 2000.

[74] R. S. K. Kwan and M. A. Rahin. Object Oriented Bus Vehicle Scheduling – The BOOST System. In Wilson [130], pages 177–191.

[75] C. J. Layfield. Investigations into the Master Timetabling Problem. Master's thesis, University of Calgary, September 1998.

[76] C. Le Pape. Constraint-Based Programming for Scheduling: An Historical Perspective. In *Working notes of the Operations Research Socity Seminar on Constraint Handling Techniques, London, UK*, 1994.

[77] The Solution of Large Scale Scheduling Problems by Computer: An Appreciation of Basic Principles and Future Possibilities. Internal Report, The University of Leeds, 1960.

[78] R. Lessard, J.-M. Rousseau, and D. Dupuis. HASTUS I: A Mathematical Programming Approach to the Bus Driver Scheduling Problem. In Wren [132], pages 255–266.

[79] L. K. Luedtke. RUCUS II: A Review of System Capabilities. In Rousseau [102], pages 61–116.

[80] A. K. Mackworth and E. C. Freuder. The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence*, 25:65–74, 1985.

[81] A. K. Mackworth and E. C. Freuder. The Complexity of Constraint Satisfaction Revisited. *Artificial Intelligence*, 59:57–62, February 1993.

[82] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[83] P. D. Manington. *Mathematical and Heuristic Approaches to Road Transport Scheduling.* PhD thesis, University of Leeds, 1977.

[84] K. Marriot and P. Stuckey. *Programming with Constraints: An Introduction.* The MIT Press, Cambridge, Mass, 1998.

[85] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1992.

[86] M. Minoux. *Mathematical Programming Theory and Algorithms.* Wiley & Sons, 1986.

[87] R. Mohr and T. C. Henderson. Arc and Path Consistency Revisited. *Artificial Intelligence*, 28:225–233, 1986.

[88] R. Mohr and G. Masini. Gold Old Discrete Relaxation. In *Proceedings of ECAI88, Munich*, 1988.

[89] P. Mott and H. Fritsche. INTERPLAN - An Interactive Program System for Crew Scheduling and Rotering of Public Transport. In Daduna and Wren [26], pages 200–211.

[90] T. Müller. Solving set partitioning problems with constraint programming. In *Practical Applications of Constraint Technologies (PACT98)*, pages 313–332. The Practical Application Company Ltd, 1998.

[91] B. Nudel. Constraint Satisfaction Algorithms. *Computational Intelligence*, 5:188–224, 1989.

[92] M. E. Parker and B. M. Smith. Two Approaches to Computer Crew Scheduling. In Wren [132], pages 193–221.

[93] P. Prosser. Forward Checking with Backmarking. Technical Report AISL-48-93, University of Strathclyde, 1993.

[94] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[95] P. Prosser. Forward Checking with Backmarking. In Meyer, editor, *Constraint Processing*, LNCS 923, pages 185–204. Springer-Verlag, 1995.

[96] J.-F. Puget. PECOS A High Level Constraint programming Language. In *Proceedings of SPICIS 92, Singapore*, September 1993.

[97] J.-F. Puget. A C++ Implementation of CLP. In *Proceedings of SPICIS 94, Singapore*, November 1994.

[98] J.-F. Puget and M. Leconte. Beyond the Glass Box: Constraints as Objects. In J. Lloyd, editor, *Logic Programming, Proceedings of the international symposium on logic programming*, pages 513–527, 1995.

[99] Ravindran, Phillips, and Solberg. *Operations Research Principles and Practice*. Wiley & Sons, 1987.

[100] C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. Wiley, 1993.

[101] J-C Regin. Generalized Arc Consistency for Global Cardinality Constraint. In *Proceedings of AAAI/IAAI*, volume 1, pages 209–215, 1996.

[102] J.-M. Rousseau, editor. *Computer Scheduling of Public Transport 2*, Proceedings of the Third International Workshop on Computer-Aided Scheduling of Public Transport. Springer-Verlag, 1985.

[103] J.-M. Rousseau. Results Obtained with Crew-Opt: A Column Generation Method for Transit Crew Scheduling. In Daduna et al. [24], pages 349–358.

[104] J.-M. Rousseau and J.-Y. Blais. HASTUS : An Interactive System for Buses and Crew Scheduling. In Rousseau [102], pages 45–60.

[105] J.-M. Rousseau and R. Lessard. Enhancements to the HASTUS Crew Scheduling Algorithm. In Rousseau [102], pages 295–310.

[106] D.M. Ryan. ZIP - a Zero One Integer Programming Package for Scheduling. Technical Report CSS-85, AERE, Computer Science and Systems Division, Harwell, Oxfordshire, 1980.

[107] D. Sabin and E. C. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proceedings of the European Conference on Artificial Intelligence*, pages 125–129, 1994.

[108] F. Shepardson. Modelling the Bus Crew Scheduling Problem. In Rousseau [102], pages 247–261.

[109] H. D. Sherali. Equivalent weights in lexicographic multi-objective programs. *European Journal of Operations Research*, 18:57–61, 1982.

[110] M. Singh. Path Consistency Revisited. In *IEEE-ICTAI'95*, 1995.

[111] B. M. Smith. *Bus Crew Scheduling using Mathematical Programming.* PhD thesis, University of Leeds, 1986.

[112] B. M. Smith. IMPACS - A Bus Crew Scheduling System using Integer Programming. *Mathematical Programming*, 42:181–187, 1988.

[113] B. M. Smith. A Tutorial on Constraint Programming. Technical Report 95.14, School of Computer Studies, University of Leeds, April 1995.

[114] B. M. Smith. Succeed-first or Fail-first: A Case Study in Variable and Value Ordering Heuristics. In *Proceedings of PACT'97*, pages 321–330, 1997.

[115] B. M. Smith and A. Wren. A Bus Crew Scheduling System using a Set Covering Formulation. *Transportation Research*, 22A:97–108, 1988.

[116] K. Stergiou and T. Walsh. Encodings of Non-Binary Constraint Satisfaction Problems. In *Proceedings of AAAI-99*, 1999.

[117] K. Stergiou and T. Walsh. The Difference All-Difference Makes. In *Procedings of IJCAI-99*, 1999.

[118] D. Stirzaker. *Elementary Probability.* Cambridge University Press, 1994.

[119] G. Syswerda. Uniform Crossover in Genetic Algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–8, 1989.

[120] Preprints of the Workshop on Automated Techniques for Scheduling of Vehicle Operators for Urban Public Transportation Services. Chicago, 1975.

[121] E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[122] University of Leeds. *TRACS-II*, May 1998.

[123] M. Völker and P. Schütze. Recent Developments of HOT II. In Daduna et al. [24], pages 334–348.

[124] R. J. Wallace. Why AC-3 is almost always better than AC-4 for establishing arc consistency in CSPS. In *Proceedings IJCAI-93, Chambéry, France*, 1993.

[125] D. L. Waltz. Generating semantic descriptions from drawings of scenes with shadows. Technical Report AI-271, Massachusetts Institution of Technology, 1972.

[126] C.D.J. Waters. *A Practical Introduction To Management Science.* Addison Wesley, first edition, 1989.

[127] D. J. Watson. *Computational Problems in the Scheduling of Buses and Allocation of their Crews.* PhD thesis, University of Leeds, September 1971.

[128] W. P. Willers. *Improved Algorithms for Bus Crew Scheduling.* PhD thesis, University of Leeds, 1995.

[129] R. H. Williamson. BUSMAN: The United Kingdoms Integrated Approach to Transit Scheduling. In Rousseau [102], pages 1–43.

[130] N. H. M. Wilson, editor. *Computer-Aided Transit Scheduling*, Preprints of the Seventh International Workshop on Computer-Aided Scheduling of Public Transport. Springer-Verlag, 1999.

[131] W. L. Winston. *Operations Research - Applications and Algorithms.* Duxbury Press, third edition, 1993.

[132] A. Wren, editor. *Computer Scheduling of Public Transport*, Proceedings of the Second International Workshop on Computer-Aided Scheduling of Public Transport. North-Holland, 1981.

[133] A. Wren. General Review of the user of Computers in Scheduling Buses and their Crews. In *Computer Scheduling of Public Transport* [132], pages 3–16.

[134] A. Wren. HOT - Hamburger Optimierungs Technik, 1993. Personal communication.

[135] A. Wren. Heuristics Ancient and Modern: Transport Scheduling Through the Ages. *Journal of Heuristics*, 4:87–100, 1998.

[136] A. Wren, S. Fores, A. Kwan, R. Kwan, M. Parker, and L. Proll. A Flexible System for Scheduling Drivers. Technical Report 2002.08, School of Computing, University of Leeds, April 2002.

[137] A. Wren and N. D. F. Gualda. Integrated Scheduling of Buses and Drivers. In Wilson [130], pages 155–176.

[138] A. Wren and R. S. K. Kwan. Installing an Urban Transport Scheduling System. *Journal of Scheduling*, 2:3–17, 1999.

[139] A. Wren and J.-M. Rousseau. Bus Driver Scheduling - An Overview. In Daduna et al. [24], pages 173–187.

[140] A. Wren and B. M. Smith. Experiences with a Crew Scheduling System based on Set Covering. In Daduna and Wren [26], pages 104–118.

[141] A. Wren, B. M. Smith, and A. J. Miller. Complementary Approaches to Crew Scheduling. In Rousseau [102], pages 263–278.

[142] A. Wren and D. Wren. A Genetic Algorithm for Public Transport Driver Scheduling. *Computers and Operations Research*, 22(1):101–110, 1995.

[143] T. H. Yunes, A. V. Moura, and C. C. de Souza. Solving large scale crew scheduling problems with constraint programming and integer programming. Technical Report IC-99-19, Institute of Computing, UNICAMP, August 1999.

[144] T. H. Yunes, A. V. Moura, and C. C. de Souza. A Hybrid Approach for Solving Large Scale Crew Scheduling Problems. In *Proceedings of the Second International Workshop on Practical Aspects of Declarative Languages (PADL'00)*, volume 1753 of *Lecture Notes in Computer Science*, Boston, MA, January 2000. Springer-Verlag.

Appendix A

Sample TRACS-II Data Files

A.1 Sample Labour File (R77A Stockwell Garage)

```
Spreadover Range
A 0000 0840
B 0841 1200
*SELECT
NO
MAX-JOINUP
:35
MIN. LENGTH OF SPELL
:60
LIMIT
:40
MAX. LENGTH OF SPELL
500 500
MIN. LONG BREAK IN SPLIT DUTY
000
MIN. SPREADOVER IN SPLIT DUTY
000
MIN LENGTH OF 1ST STRETCH
100 100
MIN LENGTH OF 2ND STRETCH
100 100
MAX MEALBREAK
0100 0100
EARLIEST START OF MEALBREAK
0000 0000
LATEST START OF MEALBREAK
3600 3600
EARLIEST FINISH OF MEALBREAK
0000 0000
LATEST FINISH OF MEALBREAK
3600 3600
```

```
MIN COST (2-veh)
500 625
MIN COST (3-veh DUTIES)
3600 3600
Min work content (4-veh shifts)
3600 3600
MAX COST
0840 1200
EARLIEST SIGNING ON TIME
0000 0000
EARLIEST START ON BUS
0000  0000
LATEST START ON BUS
3600 3600
EARLIEST FINISH OFF BUS
0000 0000
LATEST FINISH OFF BUS
3600 3600
LATEST SIGNING OFF TIME
3600 3600
MAXIMUM STRETCH LENGTH
510  510
MAXIMUM PLATFORM TIME
0840 1200
MAXIMUM SPREADOVER
0840 1200
MINIMUM PAID DAY
000 000
Fine Cover (**** important, Yes 1, No 0)
0
MAXIMUM NUMBER OF MEALBREAKS
1
MAX LENGTH OF DUTY WITHOUT PNB  MIN WORK CONTENT   Min paid day (1-stretch)
000 100  400
Min work content (3-bus shifts)
3600 3600
Stop program if the number of stretches created is greater than
20000
NO.PNBS LENGTH SO.RANGE   PNB.PERIODS
1        :40     000  840      000 840
1        :240    0841  1200    000 1200
2        :40     000  840      000 840
****
Passenger Travelling? (1 or 0)
0
Depth
9999
```

A.2 Sample Vehicle File (R77A Stockwell Garage)

```
Route 77A Stockwell Garage Monday-Friday
3
1  G  Stockwell Garage (depot)
2  h  VAUXSN
3  h  VAUXSN A
Prep/disp/mob/immob
0
91      0708 1  0806 2  0919 3  1023 2  1116 3  1223 2  1316 3
        1423 2  1512 3  1616 2  1704 3  1814 2  1901 1
92      0652 1  0730 3  0842 2  0929 3  1035 2  1128 3  1235 2
        1328 3  1435 2  1523 3  1626 2  1700 3  1826 2  1940 1
93      0602 1  0710 3  0818 2  0939 3  1047 2  1140 3  1247 2
        1340 3  1447 2  1539 3  1650 2  1812 3  1928 1
94      0645 1  0723 3  0834 2  0953 3  1059 2  1152 3  1259 2
        1352 3  1459 2  1605 3  1730 2  1808 1
95      0704 1  0738 3  0850 2  1006 3  1111 2  1204 3  1311 2
        1404 3  1511 2  1601 3  1710 2  1800 3  1918 2  1959 3
        2101 2  2144 3  2246 2  2328 1
96      0427 1  0505 3  0550 2  0631 3  0731 2  0810 3  0922 2
        1017 3  1123 2  1216 3  1323 2  1416 3  1523 2  1628 3
        1754 2  1913 3  2016 2  2059 3  2201 2  2244 3  2346 2
        2424 1
97      0652 1  0751 2  0840 3  0935 2  1028 3  1135 2  1228 3
        1335 2  1428 3  1535 2  1652 3  1818 2  1851 1
98      0546 1  0653 3  0756 2  0847 3  0947 2  1040 3  1147 2
        1240 3  1347 2  1440 3  1546 2  1641 3  1750 2  1839 1
99      0805 1  0843 3  0959 2  1052 3  1159 2  1252 3  1359 2
        1451 3  1556 2  1632 3  1742 2  1831 3  1926 1
100     0650 1  0720 3  0813 2  0903 3  1009 2  1104 3  1211 2
        1304 3  1411 2  1502 3  1606 2  1724 3  1854 1
101     0749 1  0827 3  0941 2  1022 3  1141 2  1222 3  1341 2
        1422 3  1541 2  1620 3  1746 2  1821 1
102     0457 1  0537 3  0621 2  0702 3  0809 2  0856 3  0956 2
        1034 3  1153 2  1234 3  1353 2  1434 3  1551 2  1708 3
        1834 2  1905 1
103     0731 1  0851 3  1004 2  1046 3  1205 2  1246 3  1402 2
        1446 3  1601 2  1648 3  1758 2  1847 3  1925 2  1955 1
104     0637 1  0746 2  0907 3  1018 2  1058 3  1217 2  1258 3
        1417 2  1456 3  1611 2  1656 3  1807 2  1853 1
105     0522 1  0600 3  0655 2  0742 3  0837 2  0916 3  1029 2
        1110 3  1229 2  1310 3  1429 2  1506 3  1620 2  1740 3
        1857 2  1944 3  2046 2  2129 3  2231 2  2314 3  2421 1
106     0701 1  0802 2  0923 3  1041 2  1122 3  1241 2  1322 3
        1441 2  1517 3  1634 2  1752 3  1844 1
107     0713 1  0759 3  0854 2  0934 3  1053 2  1134 3  1253 2
        1334 3  1453 2  1531 3  1641 2  1717 3  1841 2  1954 3
        2021 2  2054 3  2121 2  2154 3  2221 2  2254 3  2321 2
        2352 1
108     0728 1  0829 2  0948 3  1105 2  1146 3  1305 2  1346 3
        1505 2  1544 3  1706 2  1827 1
```

```
109     0734 1  0823 3  0918 2  0959 3  1117 2  1158 3  1317 2
        1358 3  1517 2  1556 3  1722 2  1842 3  1946 2  2029 3
        2131 2  2214 3  2316 2  2358 1
110     0702 1  0819 2  0930 2  1011 3  1129 2  1210 3  1329 2
        1410 3  1529 2  1624 3  1734 2  1823 3  1932 2  2014 3
        2116 2  2159 3  2301 2  2344 3  2446 1
111     0455 1  0525 3  0606 2  0647 3  0738 2  0831 3  0931 1
1111    1502 1  1551 3  1713 2  1749 3  1847 2  1931 1
112     0518 1  0549 3  0638 2  0716 3  0825 2  0912 1
1112    1504 1  1610 3  1638 2  1728 3  1838 2  1923 3  1951 2
        2024 3  2051 2  2124 3  2151 2  2224 3  2251 2  2324 3
        2356 1
113     0532 1  0611 3  0705 2  0751 3  0846 2  0934 1
1113    1510 1  1616 3  1726 2  1817 3  1906 1
114     0542 1  0622 3  0716 2  0835 3  0956 1
1114    1520 1  1636 3  1802 2  1837 1
115     0559 1  0637 3  0724 2  0815 3  0910 2  0949 1
1115    1522 1  1646 2  1736 3  1849 1
116     0618 1  0726 3  0822 2  0859 3  1017 1
1116    1530 1  1657 2  1733 3  1852 2  1925 1
117     0633 1  0747 3  0858 2  0945 3  1030 1
1117    1533 1  1608 3  1718 2  1805 3  1841 1
118     0647 1  0803 3  0914 2  1030 1
1118    1542 1  1654 2  1744 3  1904 2  1939 3  2006 2  2039 3
        2106 2  2139 3  2206 2  2239 3  2306 2  2337 1
119     0719 1  0755 3  0906 2  0955 3  1026 1
1119    1549 1  1702 2  1752 3  1911 2  1951 1
120     0721 1  0807 3  0902 2  0939 1
1120    1607 1  1738 2  1858 3  2001 2  2044 3  2146 2  2229 3
        2331 2  2413 1
121     1625 1  1712 3  1822 2  1908 3  1937 2  2009 3  2036 2
        2109 3  2136 2  2209 3  2236 2  2309 3  2336 2  2407 1
122     1633 1  1720 3  1830 2  1916 1
123     1639 1  1810 2  1929 3  2031 2  2114 3  2216 2  2259 3
        2401 2  2439 1
Minimum time allowance for meal break
:0   :0   :0
:0   :0   :0
:0   :0   :0
Length of time paid during a meal break
:0   :0   :0
:0   :0   :0
:0   :0   :0
Minimum join-up time
:10  :10  :10
:10  :10  :10
:10  :10  :10
signing on
:10  :10  :10
Signing off
:10  :10  :10
Signing on at depot
:10  :10  :10
Signing off at depot
```

```
:10  :10  :10
Depot with lack of route knowledge
0
0
0
Depot with lack of traction knowledge
0
0
0
Leeway
0
```

Appendix B

Morning Results

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
35,789	144	70%	74%	8	123.37	14	4.44	4.22
36,126	143	70%	72%	7	122.18	14	4.56	2.34
34,146	142	70%	70%	6	124.31	14	5.09	9.52
34,450	143	70%	72%	9	123.47	14	4.58	4.43
34,133	142	70%	70%	11	123.37	14	4.58	6.31
33,568	140	65%	67%	8	125.31	14	4.35	4.29
32,979	140	65%	67%	9	123.52	14	4.30	4.10
32,982	140	65%	67%	7	122.39	14	4.15	3.39
33,911	140	65%	67%	6	122.18	14	4.50	6.23
33,741	141	65%	69%	5	122.54	14	4.40	3.53
31,335	137	60%	61%	8	122.09	14	3.48	2.25
31,474	138	60%	63%	6	123.37	14	3.40	2.26
31,097	138	60%	63%	5	123.53	14	4.10	6.14
31,714	137	60%	61%	7	124.02	14	4.15	3.05
32,666	138	60%	63%	5	122.18	14	4.23	2.16
30,618	136	55%	59%	6	123.08	14	4.05	4.54
30,442	135	55%	57%	5	122.09	14	4.04	3.02
30,441	134	55%	56%	5	124.38	14	4.06	4.25
31,347	136	55%	59%	6	123.40	14	4.03	2.17
30,942	136	55%	59%	5	123.08	14	4.07	4.07

Table B.1: HE01 - Complete Morning Results (2/3/4 pt duties)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
5,316	150	70%	72%	4	225.49	24	0.15	1.32
5,274	149	70%	72%	3	225.59	24	0.15	0.29
5,265	149	70%	72%	3	227.44	24	0.15	0.34
5,261	149	70%	72%	3	225.51	24	0.14	0.34
5,410	150	70%	74%	3	226.10	24	0.17	0.32
5,123	148	65%	67%	2	227.34	24	0.15	0.31
5,190	148	65%	69%	2	225.59	24	0.14	0.57
5,237	148	65%	69%	2	226.22	24	0.14	1.06
5,240	149	65%	69%	2	226.07	24	0.15	*0.17*
5,139	147	65%	67%	3	225.26	24	0.13	0.27
4,951	147	60%	64%	2	225.59	24	0.13	0.19
5,028	146	60%	64%	2	227.31	24	0.14	0.28
4,729	145	60%	62%	2	226.35	24	0.12	0.20
4,932	147	60%	64%	2	226.10	24	0.14	0.18
4,871	146	60%	62%	2	226.08	24	0.14	0.23
4,655	145	55%	59%	2	227.34	24	0.13	0.10
4,602	145	55%	59%	2	228.10	24	0.12	0.16
4,625	145	55%	59%	2	225.42	24	0.12	*0.15*
4,714	145	55%	59%	2	228.09	24	0.14	0.43
4,680	145	55%	59%	2	227.41	24	0.14	0.15

Table B.2: R222 - Complete Morning Results (2 pt duties)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
14,421	208	70%	72%	5	223.21	27	0.40	8.19
14,851	210	70%	72%	5	223.21	27	0.47	5.27
15,429	212	70%	75%	5	223.21	27	0.46	15.11
14,653	210	70%	73%	5	223.21	27	0.43	7.16
15,180	212	70%	75%	4	223.21	27	0.45	4.51
14,072	208	65%	68%	4	223.21	27	0.47	9.16
13,982	208	65%	67%	5	223.21	27	0.52	11.26
13,701	207	65%	67%	5	223.28	27	0.52	5.30
14,225	208	65%	68%	4	223.35	27	0.52	75.13
13,700	209	65%	68%	6	223.21	27	0.40	50.51
13,148	204	60%	62%	4	223.28	27	0.40	4.43
13,077	205	60%	63%	4	223.35	27	0.38	73.52
13,709	205	60%	65%	4	223.21	27	0.37	10.45
13,184	200	60%	62%	4	223.21	27	0.38	8.58
13,826	206	60%	63%	4	223.21	27	0.39	5.17
12,769	202	55%	58%	3	223.21	27	0.35	5.03
12,357	202	55%	58%	3	223.21	27	0.39	21.22
12,380	202	55%	57%	3	223.28	27	0.37	5.37
12,528	201	55%	58%	4	223.21	27	0.37	6.51
12,450	204	55%	58%	4	223.21	27	0.37	5.51

Table B.3: EA2 - Complete Morning Results (2 pt duties)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
12,707	200	70%	71%	7	379.28	39	0.11	1.39
12,250	200	70%	71%	5	379.15	39	0.10	3.04
12,657	201	70%	73%	5	378.43	39	0.12	*0.55*
12,816	201	70%	73%	5	378.08	39	0.11	1.24
12,127	200	70%	71%	5	379.50	39	0.10	3.01
12,326	199	65%	69%	3	379.21	39	0.11	1.09
12,078	197	65%	67%	4	380.12	39	0.10	1.32
11,489	197	65%	65%	4	379.04	39	0.10	1.03
11,287	197	65%	65%	3	379.04	39	0.09	1.15
11,988	198	65%	67%	3	380.44	39	0.10	1.32
11,139	195	60%	62%	3	379.49	39	0.10	*0.44*
11,354	195	60%	63%	4	379.15	39	0.10	1.34
11,646	196	60%	63%	4	380.08	39	0.10	2.56
11,521	196	60%	63%	4	379.33	39	0.11	2.41
11,540	196	60%	63%	3	380.06	39	0.11	1.10
10,896	194	55%	60%	3	380.53	39	0.10	2.06
10,532	192	55%	56%	3	380.00	39	0.10	0.45
10,468	193	55%	58%	2	379.41	39	0.10	0.56
10,790	192	55%	56%	2	380.39	39	0.12	1.36
10,360	192	55%	56%	2	380.36	39	0.10	1.42

Table B.4: R207 - Complete Morning Results (2 pt duties with long peak)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
12,989	202	70%	74%	4	379.48	39	0.12	0.59
13,271	202	70%	74%	4	379.57	39	0.12	2.00
13,261	202	70%	74%	4	379.39	39	0.11	1.22
12,913	201	70%	72%	3	380.34	39	0.13	1.14
12,782	201	70%	72%	3	380.13	39	0.12	1.38
12,177	198	65%	66%	3	379.57	39	0.11	1.09
12,812	200	65%	70%	3	379.52	39	0.13	1.47
12,481	200	65%	70%	2	379.57	39	0.11	1.23
12,093	198	65%	66%	2	380.49	39	0.11	1.41
12,289	199	65%	68%	2	382.14	39	0.11	2.02
11,482	195	60%	60%	3	381.38	39	0.10	1.32
11,926	196	60%	62%	2	381.11	39	0.10	2.01
11,649	197	60%	64%	3	381.04	39	0.10	1.09
11,699	196	60%	62%	2	380.00	39	0.11	1.15
11,818	196	60%	62%	2	380.58	39	0.11	1.02
11,050	194	55%	58%	2	381.41	39	0.10	1.54
11,360	194	55%	58%	2	380.59	39	0.11	*0.41*
11,256	193	55%	58%	2	380.07	39	0.11	1.27
11,033	193	55%	57%	2	382.22	39	0.09	0.54
11,093	193	55%	57%	2	381.13	39	0.09	0.49

Table B.5: R207 - Complete Morning Results (2 pt duties with short peak)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
13,434	202	70%	75%	4	379.33	39	0.11	1.58
13,314	201	70%	72%	3	380.02	39	0.12	*1.00*
12,879	201	70%	72%	3	380.08	39	0.11	1.42
13,229	201	70%	72%	3	381.00	39	0.12	1.03
12,538	201	70%	72%	4	379.46	39	0.11	1.17
12,581	200	65%	70%	3	380.29	39	0.11	0.49
12,316	197	65%	66%	3	380.18	39	0.11	1.25
12,910	199	65%	70%	3	379.16	39	0.11	1.08
12,425	199	65%	68%	2	379.26	39	0.11	1.27
12,072	198	65%	66%	3	379.41	39	0.11	1.13
11,779	196	60%	62%	2	380.43	39	0.11	1.49
12,142	197	60%	64%	3	380.11	39	0.11	1.23
11,199	195	60%	60%	2	380.01	39	0.10	0.50
11,692	195	60%	60%	2	381.06	39	0.11	1.25
11,833	197	60%	64%	3	379.59	39	0.11	1.42
11,342	194	55%	58%	2	380.16	39	0.11	1.22
10,832	193	55%	57%	2	382.02	39	0.11	2.31
11,544	194	55%	58%	2	382.47	39	0.11	1.07
11,325	192	55%	57%	2	380.14	39	0.11	1.03
10,844	193	55%	57%	2	381.50	39	0.10	1.31

Table B.6: R207 - Complete Morning Results (2 pt duties with no peak)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
56,577	449	70%	71%	7	405.55	49	1.25	39.54
55,978	449	70%	71%	6	407.17	49	1.12	114.53
56,212	453	70%	71%	8	405.54	49	1.12	179.46
55,691	452	70%	71%	8	405.29	49	1.12	156.15
56,116	453	70%	71%	6	405.20	49	1.13	48.00
54,270	445	65%	68%	6	406.24	49	1.12	145.17
53,895	448	65%	69%	7	405.33	49	1.12	131.40
54,447	447	65%	68%	6	406.16	49	1.12	168.39
54,857	447	65%	68%	6	408.17	50	1.12	175.52
54,833	444	65%	65%	6	406.39	49	1.10	104.14
52,080	437	60%	63%	6	405.43	49	1.09	21.27
51,617	439	60%	62%	5	407.35	49	1.07	25.26
53,366	442	60%	65%	6	406.06	49	1.08	118.21
53,248	440	60%	65%	5	406.36	49	1.09	97.09
50,920	437	60%	61%	5	406.42	49	1.05	29.45
50,076	432	55%	59%	5	406.01	49	1.05	122.29
47,831	431	55%	56%	4	406.46	49	1.03	90.08
52,195	439	55%	62%	5	406.04	49	1.10	42.57
50,076	435	55%	59%	5	408.02	49	1.04	17.38
51,897	434	55%	63%	5	407.19	49	1.08	157.23

Table B.7: TRAM - Complete Morning Results (2 pt duties)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
55,640	309	70%	73%	5	490.59	53	1.05	55.48
56,918	310	70%	74%	5	492.13	53	1.08	60.05
54,711	307	70%	71%	4	491.21	53	1.05	42.34
58,473	313	70%	78%	5	491.19	53	1.10	53.37
57,049	311	70%	76%	5	494.10	53	1.08	55.23
53,350	305	65%	69%	4	493.46	53	1.04	64.39
52,692	302	65%	65%	3	489.36	53	1.03	47.21
53,912	306	65%	70%	3	492.53	53	1.04	53.52
53,699	305	65%	69%	3	498.35	53	1.05	71.21
51,542	302	65%	65%	4	498.42	53	1.02	51.46
50,968	300	60%	63%	3	501.16	53	1.00	50.29
50,947	301	60%	64%	3	499.14	53	1.01	56.39
52,177	302	60%	65%	4	494.36	53	1.03	53.05
50,678	299	60%	62%	3	498.12	53	1.04	67.17
51,237	301	60%	64%	3	498.53	53	1.01	50.27
49,250	297	55%	59%	2	499.20	53	0.59	49.11
49,421	296	55%	58%	3	494.55	53	1.01	55.35
48,525	295	55%	57%	3	493.46	53	0.58	44.00
49,765	297	55%	59%	3	506.54	53	1.02	54.50
48,001	296	55%	58%	3	502.53	53	0.57	44.52

Table B.8: R77A - Complete Morning Results (2 pt duties with long peak)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
54,682	309	70%	73%	4	491.50	53	1.08	51.57
54,318	308	70%	72%	4	500.51	53	1.07	64.18
55,605	308	70%	72%	4	490.36	53	1.06	52.34
57,654	314	70%	79%	4	496.13	53	1.12	65.26
53,984	307	70%	71%	4	509.48	53	1.04	44.04
54,486	306	65%	70%	3	490.45	53	1.04	61.10
52,431	303	65%	66%	4	493.02	53	1.02	53.19
53,939	304	65%	67%	4	490.45	53	1.04	49.57
53,972	305	65%	69%	4	504.21	53	1.03	75.34
53,095	304	65%	67%	4	495.02	53	1.04	46.16
50,556	299	60%	62%	3	499.09	53	1.04	61.41
50,283	298	60%	60%	3	500.24	53	1.00	49.53
52,178	301	60%	64%	3	491.36	53	1.03	67.01
51,241	300	60%	63%	3	507.08	53	1.02	53.00
51,948	301	60%	64%	3	496.15	53	1.03	49.21
48,689	296	55%	57%	2	494.25	53	0.58	78.46
48,440	295	55%	58%	3	503.25	53	0.57	49.32
47,949	294	55%	56%	2	497.21	53	0.56	45.10
49,563	296	55%	58%	3	495.47	53	1.00	47.03
48,116	295	55%	57%	3	495.31	53	0.59	56.04

Table B.9: R77A - Complete Morning Results (2 pt duties with short peak)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
54,142	308	70%	72%	3	493.52	53	1.05	55.40
53,998	308	70%	72%	3	494.20	53	1.04	63.30
53,546	307	70%	71%	4	501.52	53	1.04	62.58
54,895	308	70%	72%	4	497.34	53	1.05	62.03
55,680	310	70%	74%	3	493.08	53	1.06	61.30
52,522	305	65%	69%	3	497.32	53	1.04	73.01
52,698	304	65%	67%	3	498.04	53	1.04	57.15
52,431	306	65%	70%	3	499.40	53	1.02	37.30
51,712	302	65%	65%	3	501.18	53	1.01	40.35
52,996	305	65%	69%	3	501.42	53	1.04	61.47
50,642	301	60%	64%	3	501.36	53	1.00	52.56
50,479	301	60%	64%	2	502.43	53	1.00	54.47
50,957	301	60%	64%	3	497.42	53	1.00	41.18
50,421	301	60%	64%	3	504.09	53	1.00	47.08
50,027	300	60%	63%	3	500.04	53	1.05	64.20
46,361	294	55%	56%	2	489.05	53	0.57	46.45
47,281	295	55%	57%	2	490.12	53	0.56	49.19
47,488	296	55%	58%	2	NS	NS	0.56	
47,986	296	55%	58%	2	498.09	53	0.57	51.17
48,677	296	55%	58%	2	503.16	53	0.58	62.55

Table B.10: R77A - Complete Morning Results (2 pt duties with no peak)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
52,648	464	70%	72%	5	596.50	69	3.29	91.14
52,670	465	70%	72%	6	596.25	69	3.32	100.01
53,868	468	70%	73%	5	596.15	69	3.42	69.22
51,928	460	70%	70%	6	596.22	69	3.34	93.36
51,171	462	70%	70%	5	596.44	69	3.30	117.54
51,591	459	65%	69%	5	596.48	69	3.28	90.32
50,617	460	65%	69%	4	597.18	69	3.25	18.51
47,398	454	65%	66%	4	597.37	69	3.11	19.55
51,180	457	65%	67%	5	597.30	69	3.26	19.29
53,780	469	65%	74%	5	597.30	69	3.32	20.46
49,194	456	60%	66%	4	597.06	69	3.02	23.12
44,438	451	60%	61%	3	596.58	69	2.57	18.36
45,953	451	60%	61%	3	597.41	69	2.50	12.24
47,907	454	60%	64%	4	597.07	69	3.07	14.32
45,728	451	60%	61%	4	597.27	69	3.01	25.31
42,487	447	55%	58%	3	599.26	69	2.51	31.21
46,078	454	55%	63%	3	598.47	69	3.09	24.02
46,702	452	55%	62%	3	598.21	69	3.02	26.54
42,335	447	55%	57%	3	596.57	69	2.49	15.27
42,712	448	55%	58%	3	597.56	69	1.49	14.13

Table B.11: R61 - Complete Morning Results (2 pt duties)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
68,675	807	70%	71%	4	1162.26	133	26.00	39.23
69,967	807	70%	72%	4	1161.37	133	25.00	53.26
74,734	809	70%	75%	4	1161.27	133	27.00	43.39
72,835	796	70%	72%	4	1161.59	133	26.00	68.33
72,819	800	70%	72%	4	1165.56	133	27.00	84.37
67,902	786	65%	66%	3	1163.46	133	23.00	74.27
62,263	782	65%	65%	3	1167.31	133	20.00	32.21
66,430	784	65%	65%	3	1163.47	133	22.00	27.53
68,595	790	65%	67%	3	1164.09	133	26.00	77.41
66,102	787	65%	68%	3	1161.38	133	22.00	58.28
64,553	780	60%	63%	3	1165.59	133	22.00	69.35
65,722	780	60%	63%	3	1169.35	133	20.00	79.18
62,771	777	60%	65%	3	1162.09	133	22.00	30.25
63,643	782	60%	65%	3	1162.26	133	22.00	34.11
58,408	771	60%	62%	3	1164.00	133	18.00	25.56
56,733	767	50%	59%	3	1161.32	133	17.00	24.46
49,813	743	50%	51%	2	1169.25	133	15.00	55.36
52,574	748	50%	52%	2	1164.17	133	15.00	23.53
53,677	748	50%	51%	2	1165.19	133	16.00	21.35
52,347	742	50%	51%	2	1165.30	133	16.00	45.41

Table B.12: UMAE - Complete Morning Results (2/3 pt duties)

Appendix C

Evening Results

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
33,340	142	70%	71%	10	122.47	14	4.00	2.49
34,898	143	70%	73%	9	125.01	14	4.19	4.16
33,916	142	70%	71%	8	122.39	14	4.05	2.25
34,961	144	70%	75%	9	123.54	14	4.20	3.59
34,244	142	70%	71%	9	121.26	14	4.12	2.11
32,615	140	65%	68%	7	121.26	14	4.03	2.32
32,620	140	65%	68%	9	122.46	14	3.51	2.26
32,150	141	65%	69%	7	122.47	14	3.53	2.10
33,235	140	65%	68%	6	125.01	14	4.02	5.19
31,360	139	65%	66%	7	122.01	14	3.37	2.32
30,260	137	60%	63%	5	121.26	14	3.31	1.49
30,707	138	60%	64%	5	126.03	14	3.32	2.47
30,084	137	60%	63%	6	121.42	14	3.24	2.15
28,665	136	60%	61%	4	121.26	14	3.05	1.58
30,971	136	60%	61%	6	123.31	14	3.38	3.03
27,982	134	55%	57%	5	122.47	14	3.02	1.53
27,271	133	55%	55%	5	124.31	14	2.55	4.12
29,928	135	55%	59%	5	122.53	15	3.25	7.49
27,894	133	55%	55%	5	121.26	14	3.05	2.17
28,624	134	55%	57%	6	121.08	15	3.08	2.41

Table C.1: HE01 - Complete Evening Results (1/2/3/4 pt duties with short peak)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
32,627	142	70%	71%	8	123.03	14	3.57	2.50
35,573	144	70%	75%	9	121.48	14	4.38	2.34
33,757	142	70%	71%	11	121.48	14	4.11	3.32
34,731	142	70%	71%	12	122.57	14	4.28	2.49
33,555	142	70%	71%	10	121.42	14	4.16	2.22
31,967	140	65%	68%	8	122.41	14	3.47	3.40
33,256	141	65%	70%	9	121.42	14	4.10	2.13
30,333	139	65%	66%	7	121.47	15	3.33	13.26
33,285	140	65%	68%	8	121.26	14	4.11	2.21
32,492	139	65%	66%	8	121.26	14	4.24	*2.21*
30,214	136	60%	61%	6	122.43	15	3.45	7.57
29,744	137	60%	63%	7	121.26	14	3.40	*4.07*
29,888	137	60%	63%	6	122.47	14	3.32	2.07
30,897	137	60%	63%	6	121.26	14	3.55	*3.45*
28,934	136	60%	61%	6	121.26	14	3.23	*2.32*
28,347	133	55%	55%	5	121.26	14	3.11	*1.55*
27,908	133	55%	55%	6	124.03	14	3.05	2.08
29,372	134	55%	57%	4	122.00	15	3.33	7.00
27,901	134	55%	57%	5	124.46	14	3.10	2.49
28,497	133	55%	55%	5	121.47	14	3.20	1.53

Table C.2: HE01 - Complete Evening Results (1/2/3/4 pt duties with long peak)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
32,303	142	70%	71%	8	123.23	14	3.45	3.00
34,369	144	70%	75%	9	122.18	14	4.12	2.52
33,779	143	70%	71%	8	123.17	14	4.04	3.57
32,828	142	70%	71%	10	124.18	14	3.53	5.10
33,116	142	70%	71%	10	125.20	14	3.55	7.15
31,587	141	65%	69%	8	123.54	14	3.38	2.38
31,361	139	65%	66%	8	121.26	14	3.39	2.59
31,331	140	65%	68%	8	121.26	14	3.34	3.38
31,432	139	65%	66%	7	122.18	14	3.37	2.37
29,685	139	65%	66%	8	123.32	14	3.16	2.01
29,526	138	60%	64%	5	122.18	14	3.18	2.45
28,746	136	60%	61%	6	122.47	14	3.09	3.33
28,621	138	60%	64%	6	124.29	14	3.03	3.11
28,832	136	60%	61%	6	124.07	14	3.10	4.38
29,054	136	60%	61%	5	122.54	14	3.11	2.14
29,511	134	55%	57%	5	123.00	14	3.08	3.10
28,545	135	55%	59%	6	123.19	14	3.08	2.31
28,752	134	55%	57%	6	122.25	14	3.11	1.56
29,062	135	55%	59%	5	122.47	14	3.10	1.49
29,267	135	55%	59%	5	121.52	15	3.17	4.58

Table C.3: HE01 - Complete Evening Results (1/2/3/4 pt duties with no peak)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
7,534	157	70%	70%	3	223.58	24	0.17	0.35
7,454	159	70%	75%	5	226.31	24	0.17	0.54
7,564	158	70%	73%	3	225.53	24	0.17	0.40
7,429	158	70%	73%	4	224.12	24	0.17	*0.27*
7,332	157	70%	70%	3	223.51	24	0.17	0.27
7,282	155	65%	66%	3	223.45	24	0.15	0.27
7,049	156	65%	68%	3	223.45	24	0.15	0.27
7,449	156	65%	68%	5	227.48	24	0.17	0.56
7,189	155	65%	66%	4	225.16	24	0.17	0.34
7,028	155	65%	66%	3	226.12	24	0.16	0.29
7,238	154	60%	64%	3	226.03	24	0.16	1.03
7,144	154	60%	64%	3	225.47	24	0.16	1.55
7,145	154	60%	64%	3	223.45	24	0.15	0.29
6,843	153	60%	61%	3	223.56	24	0.15	0.25
7,063	154	60%	64%	3	224.56	24	0.15	2.25
6,401	151	55%	57%	3	224.12	24	0.14	0.30
6,836	152	55%	59%	3	226.57	24	0.14	2.01
6,706	152	55%	59%	3	225.46	24	0.15	0.36
6,760	152	55%	59%	3	226.17	24	0.14	0.58
6,716	151	55%	57%	3	224.48	24	0.14	*0.32*

Table C.4: R222 - Complete Evening Results (2 pt duties)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
15,672	204	70%	75%	6	223.21	27	0.45	8.50
14,197	199	70%	70%	5	223.21	27	0.52	7.54
15,895	204	70%	75%	5	223.21	27	0.45	19.57
14,836	203	70%	73%	5	223.21	27	0.45	9.51
14,830	202	70%	72%	6	223.21	27	0.55	5.09
13,491	199	65%	67%	5	223.21	27	0.41	9.00
13,762	200	65%	67%	4	223.28	27	0.46	4.38
13,239	196	65%	66%	5	223.21	27	0.40	3.55
13,563	196	65%	66%	4	223.21	27	0.58	10.33
13,370	197	65%	67%	6	223.28	27	0.40	17.01
12,400	194	60%	63%	4	223.21	27	0.37	12.00
12,762	195	60%	64%	5	223.28	27	0.50	20.32
12,610	193	60%	61%	4	223.28	27	0.37	12.37
12,716	197	60%	64%	4	223.28	27	0.37	21.01
12,614	196	60%	61%	4	223.28	27	0.52	8.41
12,682	192	55%	60%	3	223.28	27	0.49	8.17
12,787	192	55%	60%	4	223.21	27	0.38	9.10
11,903	191	55%	58%	4	223.28	27	0.45	5.05
12,593	195	55%	58%	3	223.21	27	0.36	1.53
11,806	192	55%	56%	3	223.38	27	0.34	26.04

Table C.5: EA2 - Complete Evening Results (2 pt duties)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
14,729	192	70%	72%	4	381.08	39	0.14	1.15
17,862	202	70%	76%	5	380.59	39	0.15	1.35
16,711	199	70%	71%	4	380.21	39	0.14	3.32
16,780	200	70%	73%	4	380.51	39	0.14	1.35
16,861	200	70%	73%	4	379.56	39	0.15	1.32
16,060	197	65%	68%	3	380.11	39	0.14	1.51
16,172	198	65%	69%	3	382.32	39	0.14	1.24
15,252	196	65%	66%	3	380.21	39	0.14	0.59
16,681	197	65%	68%	3	381.15	39	0.14	2.07
15,909	197	65%	68%	3	381.27	39	0.14	3.06
14,960	193	60%	61%	2	382.37	39	0.14	4.50
15,399	194	60%	63%	3	384.47	39	0.14	1.01
14,958	194	60%	63%	2	378.43	39	0.14	*0.49*
15,091	195	60%	64%	3	380.53	39	0.14	1.38
15,526	195	60%	64%	3	380.53	39	0.14	1.28
13,854	190	55%	56%	2	385.02	39	0.13	1.04
13,824	190	55%	56%	2	385.15	39	0.13	0.51
15,105	192	55%	59%	2	380.18	39	0.13	1.26
13,970	192	55%	58%	2	380.20	39	0.13	1.27
14,729	192	55%	59%	2	381.08	39	0.14	1.12

Table C.6: R207 - Complete Evening Results (2 pt duties)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
50,095	439	70%	70%	7	406.49	49	1.06	115.37
53,744	443	70%	73%	8	407.55	49	1.20	73.00
53,141	443	70%	73%	9	407.29	49	1.15	87.33
52,577	443	70%	73%	8	408.11	49	1.07	139.03
51,665	442	70%	73%	10	406.55	49	1.05	98.46
49,680	433	65%	69%	7	407.08	49	1.03	117.45
47,225	432	65%	66%	6	408.57	49	0.59	99.55
51,815	437	65%	71%	7	406.15	49	1.07	151.34
46,764	432	65%	65%	6	406.30	49	0.58	112.03
48,450	435	65%	68%	7	406.31	49	1.02	135.06
47,709	426	60%	64%	6	406.08	49	0.59	156.43
43,557	424	60%	60%	5	406.50	49	0.55	48.31
49,234	432	60%	66%	6	406.56	49	1.02	111.58
45,774	425	60%	63%	6	407.18	49	0.59	114.31
48,125	430	60%	65%	6	408.32	49	1.01	130.35
42,751	419	55%	58%	5	407.33	49	0.52	79.06
42,736	418	55%	57%	5	407.38	49	0.53	95.25
43,762	420	55%	58%	6	407.50	49	0.55	40.10
42,618	415	55%	57%	5	407.12	49	0.52	11.51
42,112	423	55%	58%	5	406.51	49	0.52	104.35

Table C.7: TRAM - Complete Evening Results (2 pt duties)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
54,031	327	70%	76%	6	494.10	53	1.00	68.16
53,398	325	70%	74%	5	NS	53*	1.02	73.27
51,701	322	70%	71%	4	493.54	53	0.58	66.05
53,438	326	70%	75%	5	490.37	53	0.59	58.29
52,549	323	70%	72%	5	491.20	53	1.04	59.50
50,067	319	65%	68%	5	498.05	53	0.56	54.16
47,699	316	65%	65%	3	495.42	53	0.58	51.34
49,393	318	65%	67%	4	494.55	53	0.54	61.05
49,223	317	65%	66%	4	495.40	53	0.57	71.24
50,509	319	65%	68%	4	494.48	53	0.56	57.44
47,389	311	60%	61%	3	498.01	53	0.57	65.45
47,626	313	60%	63%	3	496.14	53	0.52	45.29
48,554	315	60%	64%	4	495.26	53	0.58	66.16
48,745	315	60%	64%	4	495.31	53	0.54	60.09
47,426	312	60%	62%	3	495.24	53	0.59	52.02
47,075	310	55%	60%	3	494.43	53	0.56	50.02
45,699	310	55%	60%	3	493.48	53	0.56	56.45
45,168	309	55%	59%	3	500.58	53	0.53	57.10
45,141	308	55%	58%	3	496.17	53	0.55	57.07
45,527	306	55%	56%	3	499.38	53	0.52	48.50

Table C.8: R77A - Complete Evening Results

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
69,787	501	70%	76%	8	596.49	69	4.36	136.00
61,378	492	70%	70%	7	596.42	69	4.11	113.40
63,184	498	70%	72%	6	597.09	69	4.16	80.38
64,447	497	70%	74%	6	596.44	69	4.12	136.49
66,088	496	70%	71%	6	597.01	69	4.12	126.17
60,150	491	65%	68%	5	597.51	69	4.06	71.14
60,383	493	65%	68%	5	598.51	69	4.22	42.10
62,001	489	65%	66%	5	596.42	69	3.58	33.09
63,292	494	65%	70%	7	597.50	69	4.11	33.49
60,951	495	65%	69%	6	597.46	69	4.03	40.49
60,630	490	60%	64%	6	598.37	69	3.55	42.33
57,704	485	60%	62%	5	596.38	69	3.50	29.18
58,656	484	60%	62%	5	596.57	69	3.51	28.26
58,431	491	60%	65%	5	596.33	69	3.34	26.03
61,287	492	60%	68%	6	597.10	69	3.56	49.01
53,641	484	55%	59%	5	597.27	69	3.21	28.20
54,251	480	55%	56%	4	598.00	69	3.33	23.23
55,022	480	55%	56%	4	599.16	69	3.35	52.57
56,821	485	55%	60%	4	599.16	69	3.40	24.46
53,842	478	55%	56%	4	597.08	69	3.29	26.12

Table C.9: R61 - Complete Evening Results (2 pt duties)

Potential Duties	Work Pieces	T%	A%	#U	Sched Cost	Duties Used	BUILD Time	SCHED Time
92,114	850	70%	76%	4	1164.23	133	37.00	57.30
92,905	857	70%	73%	4	1162.13	133	43.00	104.53
94,096	867	70%	77%	4	1168.11	133	44.00	56.41
96,317	866	70%	76%	4	1162.09	133	45.00	96.33
92,341	852	70%	74%	4	1162.21	133	43.00	91.21
89,570	854	65%	65%	3	1165.00	133	37.00	60.21
89,929	858	65%	70%	4	1169.51	133	37.00	76.20
87,620	858	65%	65%	3	NS	NS	38.00	128.12
91,158	848	65%	65%	3	1160.29	133	41.00	53.11
87,751	850	65%	65%	3	1167.57	133	37.00	57.56
90,161	846	60%	63%	3	1166.16	133	36.00	52.37
91,517	853	60%	64%	3	1175.47	133	37.00	109.57
86,856	832	60%	61%	3	1166.48	133	35.00	62.12
82,994	845	60%	61%	3	1171.46	133	33.00	72.14
86,389	838	60%	61%	3	1168.19	133	35.00	64.00
85,992	848	55%	60%	3	1164.59	133	34.00	53.45
83,892	836	55%	57%	2	1176.13	133	36.00	123.43
86,076	840	55%	60%	3	1166.32	133	35.00	50.59
85,123	843	55%	58%	3	1168.56	133	34.00	50.15
82,588	838	55%	58%	3	1165.43	133	36.00	54.11

Table C.10: UMAE - Complete Evening Results (2/3 pt duties)

Appendix D

Combined Results

Potential Duties	Work Pieces	T%	Sched Cost	Duties Used	BUILD Time	SCHED Time
24,933	128	70%	124.01	14	3.01	1.54
28,138	129	70%	121.48	14	3.24	*2.06*
24,466	126	70%	123.32	14	2.30	2.11
25,569	127	70%	123.08	14	2.41	1.51
24,142	126	70%	122.25	14	2.34	*1.43*
22,084	122	65%	124.04	14	2.08	1.35
22,665	123	65%	123.52	14	2.20	1.24
20,188	121	65%	121.44	15	1.45	3.14
23,678	122	65%	121.26	14	2.28	*1.19*
22,677	122	65%	121.42	14	2.12	1.12
18,677	115	60%	122.07	15	1.46	1.38
18,241	117	60%	123.10	14	1.33	1.31
18,017	117	60%	126.01	14	1.28	2.29
18,241	117	60%	123.10	14	1.34	1.40
18,492	116	60%	121.26	14	1.33	*1.34*
16,027	111	55%	122.41	14	1.18	*1.42*
15,331	110	55%	124.19	14	1.13	0.50
16,659	110	55%	122.13	15	1.26	1.32
16,533	112	55%	126.39	14	1.18	*0.49*
16,391	111	55%	123.02	14	1.19	*0.44*

Table D.1: HE01 - Combined Morning/Evening Results (2/3/4 pt duties)

Potential Duties	Work Pieces	T%	Sched Cost	Duties Used	BUILD Time	SCHED Time
3,412	129	70%	225.59	24	0.08	0.24
3,286	131	70%	227.42	24	0.08	0.09
3,344	130	70%	229.31	24	0.08	*0.09*
3,326	130	70%	226.51	24	0.08	0.15
3,380	130	70%	226.10	24	0.09	0.13
3,053	125	65%	227.45	24	0.06	0.07
3,024	127	65%	225.55	24	0.07	0.19
3,208	127	65%	228.40	24	0.07	0.09
3,086	126	65%	228.19	24	0.06	*0.08*
2,984	125	65%	227.59	24	0.06	0.12
2,869	123	60%	227.06	24	0.06	*0.07*
2,951	123	60%	228.23	24	0.06	0.18
2,635	122	60%	226.35	24	0.06	0.09
2,684	122	60%	226.39	24	0.05	0.07
2,755	122	60%	226.36	24	0.06	0.10
2,266	118	55%	228.42	24	0.04	0.07
2,352	119	55%	229.38	24	0.06	0.07
2,370	119	55%	228.23	24	0.05	0.07
2,475	119	55%	229.41	24	0.06	0.08
2,428	118	55%	228.36	24	0.05	*0.07*

Table D.2: R222 - Combined Morning/Evening Results (2 pt duties)

Potential Duties	Work Pieces	T%	Sched Cost	Duties Used	BUILD Time	SCHED Time
9,930	184	70%	223.21	27	0.29	2.21
9,178	183	70%	223.28	27	0.26	58.22
10,738	188	70%	223.28	27	0.30	25.54
9,557	185	70%	223.21	27	0.27	3.04
9,865	186	70%	223.28	27	0.28	15.39
8,139	179	65%	223.21	27	0.22	6.30
8,266	179	65%	223.28	27	0.22	3.52
7,658	177	65%	223.28	27	0.21	1.00
8,282	178	65%	223.35	27	0.22	22.52
7,919	180	65%	223.28	27	0.22	2.27
6,858	172	60%	223.35	27	0.17	4.02
7,116	174	60%	223.42	27	0.20	1.52
7,402	173	60%	223.28	27	0.22	1.52
7,159	171	60%	223.28	27	0.21	3.12
7,420	173	60%	223.28	27	0.21	5.16
6,755	168	55%	223.49	27	0.17	38.51
6,543	168	55%	223.30	27	0.19	3.58
6,045	167	55%	223.42	27	0.18	3.50
6,569	165	55%	223.36	27	0.19	1.40
5,976	167	55%	223.42	27	0.17	5.29

Table D.3: EA2 - Combined Morning/Evening Results (2 pt duties)

Potential Duties	Work Pieces	T%	Sched Cost	Duties Used	BUILD Time	SCHED Time
7,725	176	70%	381.26	39	0.06	0.29
9,170	186	70%	380.59	39	0.08	1.07
8,820	184	70%	380.24	39	0.08	1.04
8,883	185	70%	381.25	39	0.07	2.25
8,555	184	70%	380.48	39	0.08	0.58
8,192	180	65%	380.41	39	0.07	1.02
8,059	179	65%	383.38	39	0.08	0.34
7,155	177	65%	380.59	39	0.06	1.03
7,602	178	65%	381.22	39	0.07	1.00
7,826	179	65%	383.50	39	0.07	0.49
6,888	172	60%	383.52	39	0.06	0.30
7,150	174	60%	386.39	39	0.06	0.40
7,084	174	60%	380.08	39	0.06	0.29
7,145	175	60%	381.23	39	0.06	1.24
7,422	175	60%	382.23	39	0.06	1.13
6,064	168	55%	386.43	39	0.05	0.21
5,672	166	55%	387.28	39	0.05	*0.18*
6,388	169	55%	381.23	39	0.05	0.37
6,084	167	55%	381.53	39	0.05	0.32
6,179	168	55%	382.07	39	0.05	0.22

Table D.4: R207 - Combined Morning/Evening Results (2 pt duties)

Potential Duties	Work Pieces	T%	Sched Cost	Duties Used	BUILD Time	SCHED Time
30,964	402	70%	407.41	49	0.37	11.16
33,315	405	70%	407.41	49	0.39	121.46
32,872	409	70%	406.46	49	0.41	111.26
32,404	408	70%	407.33	49	0.40	146.02
32,248	408	70%	406.35	49	0.37	115.42
29,364	391	65%	407.00	49	0.33	45.58
26,694	393	65%	406.44	49	0.34	92.30
30,597	397	65%	406.47	49	0.38	113.36
27,299	392	65%	408.50	50	0.31	37.57
27,281	392	65%	407.07	49	0.32	9.23
26,048	376	60%	406.42	49	0.27	64.46
22,628	376	60%	409.36	49	0.27	52.24
28,075	387	60%	406.58	49	0.32	45.42
25,869	378	60%	408.27	49	0.30	76.27
25,460	380	60%	408.45	49	0.29	3.24
21,572	364	55%	407.20	49	0.25	8.53
19,858	364	55%	408.52	49	0.20	6.29
22,736	372	55%	409.27	49	0.25	43.37
20,960	363	55%	409.57	49	0.24	34.39
21,914	373	55%	407.44	49	0.25	8.08

Table D.5: TRAM - Combined Morning/Evening Results (2 pt duties)

Potential Duties	Work Pieces	T%	Sched Cost	Duties Used	BUILD Time	SCHED Time
35,109	284	70%	498.19	53	0.39	52.18
35,240	283	70%	510.31	53	0.39	58.03
32,119	277	70%	499.25	53	0.36	38.28
36,911	287	70%	491.11	53	0.41	59.22
34,794	282	70%	491.13	53	0.38	19.51
27,181	272	65%	498.19	53	0.30	32.02
29,999	266	65%	496.33	53	0.35	42.56
29,731	272	65%	495.59	53	0.33	10.13
29,110	270	65%	518.34	53	0.33	43.55
28,873	269	65%	494.25	53	0.31	17.31
25,247	259	60%	509.40	53	0.28	36.10
25,682	262	60%	516.46	53	0.29	31.13
27,548	265	60%	499.35	53	0.30	38.46
26,112	262	60%	505.51	53	0.29	31.16
25,650	261	60%	510.39	53	0.28	26.20
23,794	255	55%	522.51	53	0.28	41.07
23,306	254	55%	502.23	53	0.25	13.53
22,296	253	55%	510.25	53	0.26	28.16
22,874	253	55%	502.17	53	0.25	13.50
21,544	250	55%	521.51	53	0.25	7.04

Table D.6: R77A - Combined Morning/Evening Results (2 pt duties)

Potential Duties	Work Pieces	T%	Sched Cost	Duties Used	BUILD Time	SCHED Time
35,782	426	70%	596.38	69	2.06	87.09
31,391	417	70%	596.40	69	1.56	53.12
33,266	424	70%	597.21	69	1.59	74.15
32,686	417	70%	598.33	69	2.02	17.34
32,597	418	70%	597.28	69	2.01	70.07
29,754	411	65%	598.57	69	1.52	10.27
29,818	412	65%	596.59	69	1.48	11.43
28,495	402	65%	597.37	69	1.31	10.35
30,831	411	65%	597.33	69	1.50	12.27
29,807	415	65%	598.14	69	1.47	17.57
26,776	401	60%	598.38	69	1.39	9.51
23,993	391	60%	597.24	69	1.27	7.30
25,776	390	60%	598.01	69	1.34	10.39
26,950	404	60%	597.52	69	1.35	14.39
23,939	393	60%	597.44	69	1.25	42.19
21,290	389	55%	599.18	69	1.17	9.49
21,090	381	55%	597.41	69	1.11	6.03
22,215	383	55%	598.46	69	1.17	46.59
22,670	387	55%	597.53	69	1.25	7.03
21,657	384	55%	597.19	69	1.17	12.25

Table D.7: R61 - Combined Morning/Evening Results (2 pt duties)

Potential Duties	Work Pieces	T%	Sched Cost	Duties Used	BUILD Time	SCHED Time
61,820	785	70%	1164.44	133	18.00	34.35
59,879	788	70%	1162.13	133	20.00	42.09
66,653	803	70%	1168.24	133	23.00	37.55
66,769	787	70%	1162.02	133	21.00	30.04
63,742	776	70%	1166.31	133	21.00	70.06
56,922	762	65%	1170.11	133	16.00	61.27
52,030	765	65%	1170.09	133	15.00	30.44
60,030	789	65%	1169.46	133	21.00	38.04
58,150	761	65%	1162.32	133	20.00	26.33
53,759	765	65%	1167.24	133	17.00	29.22
53,468	750	60%	1169.01	133	16.00	27.22
55,653	756	60%	1182.09	133	17.00	74.51
50,217	745	60%	1164.50	133	15.00	23.19
48,696	751	60%	1171.25	133	14.00	34.58
46,065	737	60%	1172.04	133	13.00	23.19
44,501	735	55%	1168.15	133	12.00	23.26
36,857	702	55%	–	133*	9.00	72.46
40,834	708	55%	1170.50	133	11.00	17.27
41,221	708	55%	1174.30	133	11.00	43.31
38,799	706	55%	1172.48	133	9.00	61.53

Table D.8: UMAE - Combined Morning/Evening Results (2/3 pt duties)

Appendix E

Differences Between New and Old TRACS-II

As mentioned in Section 4 the version of TRACS-II used for the research carried out in this thesis was a prototype to the version of TRACS-II available today.
There are several differences between the two versions. This section will list the differences with regards to the components present in TRACS-II (it should be observed that the basic system is the same). The current state of the art TRACS-II is described in [136, 41].

BUILD - The limit on the number of potential duties built is now much higher. In fact [136] reports that BUILD has generated 1,500,000 duties in one problem (although this is not typical).

SIEVE - As BUILD can product twice as many duties as before, SIEVE is no longer used to trim the size of the set of duties generated. SIEVE is used either to prepare for MERGE or, if needed, to cut down the set of duties generated by BUILD to speed up the SCHEDULE process.

MERGE - Merge can now, effectively, cope with any number of duties.

P-Process - No longer used.

SCHEDULE - SCHEDULE can deal with 200,000+ potential duties now (the previous limit was 100,000). The search tree in SCHEDULE is no longer limited to 500 nodes.

Index