MODELLING AND OPTIMISATION OF RAILWAY CREW SCHEDULING

Rosmalina Hanafi

BEng (Mechanical Engineering)
MEng (Industrial Engineering)

A thesis submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy

Principal Supervisor: **Professor Erhan Kozan**

Decision Science Discipline, Science and Engineering Faculty
Queensland University of Technology
Brisbane, Australia
2014

KEYWORDS

Combinatorial Optimisation, Constraint Programming, Constructive Heuristics, Hybrid Algorithms, Mathematical Programming, Metaheuristics, Railway Crew Scheduling, Simulated Annealing, Tabu Search.

ABSTRACT

Crew Scheduling is one stage of operational planning in transportation systems to which mathematical and algorithmic optimisation techniques can be applied. Crew Scheduling Problem (CSP) in the transportation industry represents a computationally difficult combinatorial optimisation problem. The large number of tasks (trips) to include and the complicated operational and contractual requirements are the main reasons for the complexity of the problem. Nevertheless, solving CSP has been one of the most important focuses of the transportation industry because it affects the company's profitability and its service quality. An optimal crew schedule is essential to ensure efficient and reliable operations of transportation services. Furthermore, the cyclic nature of the crew scheduling application makes the CSP a good candidate for optimisation. A small improvement to the crew schedules can lead to accumulated savings that produce large annual cost savings. The difficulty of solving CSP yet its enormous practical significance, have led to a large number of proposed solution techniques. However, unlike CSP in other modes of transportation such as airline and bus which have been intensively studied, railway CSP is less cited in literature. Railway crew scheduling is domain specific and there has been no developed solving method which has been applied universally. Models and algorithms are designed mainly for a specific case and may not readily be applied in different applications.

Railway CSP is the process of allocating train services to the crew duties based on the published train timetable while satisfying operational and contractual requirements. The problem is restricted by many constraints and it belongs to the class of NP-hard (nondeterministic polynomial-time hard). CSP is more frequently formulated mathematically, as either set covering problem or set partitioning problem, and then solved analytically or approximately. Even though some studies have been done on CSP using a wide variety of solution techniques, the problem is still difficult to solve. CSP involves real-life constraints which are difficult to handle, such as crew breaks, elapsed time and the requirement to return the crews to their home depots at the end of their duty. Furthermore, an optimisation model should be well designed by which all the relevant parameters of the problem can be incorporated. One way of dealing with the problem is to develop a specific model that is capable of incorporating important features of the problem and can be solved using a wide range of methods.

This research has developed and analysed two railway crew scheduling models. The first is mathematical programming (MP)-based model which is formulated as a mixed integer programming (MIP) while the second is constraint programming (CP)-based model. The objective of the optimisation models is to minimise the number of crew duties by reducing idle transition times. Duties are generated by arranging scheduled trips over a set of duties and sequentially ordering the set of trips within each of duties. The integration of relief opportunities period (ROP) into models would enable the train crew to be relieved at any relief point (RP) within the interval of ROP.

Existing models and algorithms usually only consider relieving a crew at the beginning of the interval of relief opportunities which may be impractical. The inclusion of the ROP into models has not been studied in depth. Allowing the train crew to be relieved at any RP during the ROP will provide better representation of real-world conditions and improve the robustness of the schedule.

Due to the combinatorial nature of the CSP, heuristic methods are the most promising approach for solving the problem. The main limitation with many of conventional heuristic algorithms is their difficulty to escape from locally optimal solutions. The search is usually conducted from a single point in the solution space, then continues until there is no possible improvement. This type of local search method can easily get trapped in local optima. In an attempt to address this problem, several metaheuristics approaches have emerged for solutions to combinatorial complex problems such as the simulated annealing (SA) and tabu search (TS). SA and TS have been applied successfully to solve many combinatorial optimisation problems in various practical settings. Despite the potential application of metaheuristics to solve combinatorial optimisation problems, very few attempts have been made to tackle crew scheduling related problems in the literature applying metaheusristics.

SA and TS algorithms have been utilised in this study to improve solutions and to derive near-optimal solutions. Initial solution for railway CSP is generated by a constructive heuristic (CH) and then it is improved by a hybrid constructive heuristic SA (HCHSA) algorithm. Both the CH and the HCHSA algorithms produced acceptable solutions, although the produced solutions are not guaranteed to be an optimal solution. The HCHSA algorithm significantly improves the solution produced by the CH. The HCHSA algorithm increases the average driving time by 3.06% and decreases the average excess cost by 3.35%. Overall, the HCHSA algorithm increases the total crew working time and reduces the number of crew duties for all datasets. As the number of crew duties corresponds to the number of crew needed, significant savings can be gained on the annual cost of crew related expenses. The solutions produced by the HCHTS algorithm, which was composed of a three-phase heuristic, indicate that the proposed algorithm is able to generate near-optimal feasible solutions within an acceptable computational time. This is indicated by the average Q value which is fairly close to zero. An aggregation procedure has a significant effect in reducing the problem size such that the proposed TS-based algorithms are able to handle large-sized railway CSP and solve it within an acceptable computational time. The neighbourhood structure also contributes to the effectiveness of the search process. The solutions obtained by the hybrid CP and SA algorithm (HCPSA) and the hybrid TS and SA algorithm (HTSSA) also give an indication of the effectiveness of hybridisation of an exact method and metaheuristics as well as hybridisation of metaheuristics (TS and SA) to produce good acceptable solutions in a reasonable computational time for large-sized instances.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CP	Constraint Programming
CSP	Crew Scheduling Problem
HCHSA	Hybrid Constructive Heuristic Simulated Annealing
HCPSA	Hybrid Constraint Programming Simulated Annealing
HD	Home Depot
HTSSA	Hybrid Tabu Search Simulated Annealing
IP	Integer Programming
LP	Linear Programming
MB	Meal Break
MIP	Mixed Integer Programming
MP	Mathematical Programming
ROP	Relief Opportunities Period
RP	Relief Point
SA	Simulated Annealing
SCP	Set Covering Problem
SPP	Set Partitioning Problem
TS	Tabu Search

PUBLICATIONS ARISING FROM THIS RESEARCH

Published Journal Papers

Hanafi, R., and Kozan, E. (2014). A hybrid constructive heuristic and simulated annealing for railway crew scheduling. *Computers & Industrial Engineering*, *70*, 11-19. Available online 16 January 2014, ISSN 0360-8352, doi: http://dx.doi.org/10.1016/j.cie.2014.01.002

Hanafi, R., and Kozan, E. (2012). A constraint programming approach for railway crew scheduling. *Annals of Operations Research*. (under review)

Hanafi, R., and Kozan, E. (2014). Multi depots railway crew scheduling: model and algorithms. *International Journal of Production Economics*. (under review)

Refereed Conference Paper

Hanafi, R., and Kozan, E. (2012). A flexible optimisation model for the railway crew scheduling problem. In Günther, H. O., Kim, K. H., Kopfer, H. (Eds.), International Conference on Logistics and Maritime Systems (pp. 159–169). Bremen, Germany.

STATEMENT OF ORIGINAL AUTHORSHIP

The work contained in this thesis has not been previously submitted to meet requirements for an award at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signature:

Date: 20 October 2014

ACKNOWLEDGEMENTS

I would like to acknowledge and sincerely thank my Principal Supervisor Professor Erhan Kozan for his continuous support and encouragement throughout my study. His expertise in his field has helped me to keep making progress and enabled me to get through all stages of my research journey. My appreciation is extended to the members of research committee, Professor Paul Hyland, Professor Vo Anh, and Adjunct Associate Professor Paul Corry for review, comments, and suggestions.

I thank the Service Planning Manager Mr Graeme Sang, the Allocation Coordinator Mr Scott Brown, and the Vocational Support Unit Manager Mr Eric Whittington at Queensland Rail (QR), Australia for the visit and for providing me with supported materials.

I am thankful to many people in ISS QUT International Team E for various help during my study, especially to Elkin Dario Giraldo, Zia Song, Thara Jayaratne, and Christine Wang.

I thank all members of the Decision Science Discipline Group at QUT, to make my PhD study within Decision Science Discipline Group is a good experience.

I thank my fellow research students Hazreen Harith, Maryam Shirmohammadi, Candra Dharmayanti, Diah Parami Dewi, and Ellya Zulaikha, for discussions and friendship. I enjoyed the interactions with which have enriched my social life during my PhD study in the last few years.

I thank my friends Mel, Gina, Ben, Phoenix, Iris, Kath, Cansu, Matt, Astrid, Muthu, Steve, Amber, and Sam, for all the fun we have had and for making my stay in Brisbane much more pleasurable.

My sincere thanks go to my beloved family members, for all of their love, great support, and unending encouragement. Their support enabled me to cope with challenges during my study. Without them, this journey would not have been possible.

Chapter 1: **Introduction**

This chapter presents an overview and a brief explanation of research background. The context of the research and the crew scheduling related issues are briefly explained. The chapter also outlines the structure of the thesis.

1.1 RESEARCH BACKGROUND

The transportation industry is a capital-intensive sector which involves a large number of resources. Optimum utilisation of the available resources has been one of the main targets of the transportation industry, in striving to reduce operational costs with a better service quality level. Operations crew is one of the resources that has a great impact on the operational performance of the transportation system. Effective management of this resource can considerably reduce operational costs and increase the efficiency of the entire systems. The crew scheduling problem (CSP) in the transportation industry is a typical optimisation problem concerned with finding the optimum arrangement of a set of activities while subjected to specified constraints. An activity in this regard is the assignment of a crew to a set of scheduled trips in such a way that the generated crew duties conform to the predefined work rules and regulations. Optimisation criterion can be considered as either minimising a time (cost)-based objective or maximising resource utilisation-based objective.

CSP falls into the category of combinatorial complex optimisation problem. As the computation progresses, the number of potential solutions is sequentially compounded leading to a large number of choices. CSP has been proved to be NP-hard (Fischetti et al. 1987, 1989), for which no known method is able to obtain an optimal solution in polynomial time. Because of its computational complexity and application potential, CSP has remained one of challenging optimisation problems.

Crew scheduling is one stage of planning and scheduling problem that can be modelled and solved using mathematical optimisation techniques. However, the process of crew scheduling at large transportation organisations is very complex. This is because of the large number of tasks (trips) to cover and the complex operational and contractual requirements involved.

The study on CSP mainly relied on the traditional set covering and set partitioning formulations. Various solution techniques have been offered to solve these models. In both the set covering and the set partitioning formulations, the decision variable is a binary integer variable which represents whether or not a duty (roundtrip, pairing) is selected as work for a crew member. The constraint in the set covering problem consists of a matrix of binary values, which defines that each piece of work is covered by a duty at least once. Each column represents one possible roundtrip or work to be performed by an individual crew member over a defined period of time. The set partitioning problem is similar to the set covering problem, but for the set partitioning formulation the constraint becomes equal to one, meaning that each task is covered exactly once. The main difficulty in applying the exact methods to solve CSP is that in determining all possible solutions. For CSP with a large number of trips, there can be an unmanageably large number of possible roundtrips. As a consequence, the problem becomes a time-consuming process of enumerating all the possible roundtrips. For this reason, there is a requirement of large-scale solution techniques such as column generation-based methods or sub problem optimisations. The concept of column generation is to solve a sequence of reduced problems (master problem) in which each reduced problem contains a small fraction of the set of variables (columns). Bengtsson et al. (2007) formulated a general crew pairing problem with the objective function being to minimise the cost of selected pairing and the cost of violating soft constraints. The research combines resource constraints, k-shortest path enumeration, and label merging techniques and shows that a column generation approach is able to heuristically solve large and highly complex railway pairing problems in a reasonable time. Given the size and complexity of the railway operation, the researchers indicate the necessity of combined optimisation techniques. Nishi et al. (2011) proposed a column generation with dual inequality for railway crew scheduling. Computational results have shown that the proposed technique can accelerate the convergence of conventional column generation for a large data set application. Yan and Tu (2002), however, stated that column generation-based methods could be inefficient because when the crew scheduling is formulated as a traditional set covering problem, the obtained optimal solutions could be non-integer solutions. Other techniques should then be incorporated to refine the non-integer solutions. Bangert (2012) has also noted that the method of enumeration is not realistic when the number of options is too large and cannot be practically listed. De Leone et al. (2011) proposed a mathematical model

to solve a CSP. Since their proposed model can only handle small- to medium-sized problems, a greedy randomised adaptive search procedure has then been offered to solve large instances.

Due to the combinatorial nature of CSP, heuristic methods are the most promising approach for solving the problem. There has been an emerging approach toward the heuristics and metaheuristics search methods for solutions to combinatorial complex problems. Some of these approaches are inspired by natural phenomena. They are usually not sensitive to initial solutions and enable the application of parallel processing. Additionally, since the process of searching a solution is not limited to a certain domain in the solution space, the chance of being trapped into local optima is much less. Simulated Annealing (SA) and Tabu Search (TS) are such methods and they have become promising search techniques to find solutions for combinatorial optimisation problems.

1.2 CONTEXT OF THE RESEARCH

Crew planning and scheduling in railway transport are highly complicated problems because of the size of the instances and the type and number of involved constraints. CSP is concerned with finding an optimal way of allocating crews to perform their duties in such a way to cover all travelling tasks in a published timetable. Due to its computational complexity, CSP is still a significant topic of research. Many researchers and practitioners have devoted considerable effort to solve this problem. However, the majority of the work to date in this area has come from European, American, and Asian researches. Very limited research has been done to examine CSP in the Australian railway industry. Furthermore, the existing models and algorithms for the railway CSP are usually designed for a specific application area. As the operating procedures and regulations vary between railway operators, the policies and workplace agreements specifically dictate the condition of a problem might not be applicable to other situations.

To address the deficiency, this research studies CSP and develops an analytical model for railway CSP. Based on this research problem, this research addresses several questions;

i. What are the existing modelling and solution approaches for generic CSP and railway CSP?
ii. What are the real-life constraints that need to be included in developing a model for railway CSP?
iii. What is the suitable modelling approach for crew scheduling in the railway industry?
iv. How this complex problem can be solved in a reasonable time frame.

The research objectives are established based on the research questions to drive the research process. The aim of this research then are;

i. To investigate the existing approaches in modelling and solving generic CSP and railway CSP.
ii. To identify real-life constraints which directly influence the development of a model for railway CSP and integrate them into the developed model.
iii. To design and develop an analytical model and algorithms for railway CSP.
iv. Design innovative algorithms to solve the model developed for this study.

The research was based on a combined research strategy of literature review, analysis, and discussions to explore relevant information. The overall research consists of three main stages, namely problem identification; model development and solutions; and sensitivity analysis and implementation recommendations.

At the first stage, problem identification was initiated through an extensive literature review on the area of CSP in the transportation systems in general. After reviewing the existing literature, the scope of the problem was narrowed down to railway CSP. The objectives of this stage were to have a better understanding of the topic to be addressed; to identify relevant information; to identify the gap in the research area and the position of research in the context of existing researches; and to identify models and algorithms that have been used to solve the generic CSP and specifically railway CSP. Furthermore, identification of problem was required to clarify the need for research which leads to the research questions and research objectives. The literature review process was conducted throughout the research period to reflect and accommodate new information. A brief summary of the literature review is presented in Chapter 2.

The second stage is the model development and solutions. Modelling and optimisation of railway CSP has been be approached through the application of mathematical programming (MP), constraint programming (CP) methods, and heuristics/metaheuristics techniques. A mathematical model is a mathematical representation of a real problem. The model should express important features of the problem under study in the form of mathematical functions of decision variables, and express the relationships among them using appropriate equations or inequalities. Real-world problems are usually too complex to capture all details. A model is usually designed by simplifying features but it still provides a sufficiently precise representation of the main problem characteristics such that the solutions obtained remain valid to the problem under study to an acceptable degree of approximation. Therefore, developing a mathematical model may involve making approximations and adjustments, and sometimes ignoring or adjusting features which are difficult to represent mathematically.

In this research, railway CSP was formulated based on MP, as a Mixed Integer Programming (MIP), and CP. The choice to model railway CSP as MP and CP was motivated by the fact that they allow complex constraints to be incorporated. Once the problem has been formulated then a solution has to be found using the models. Optimisation programming language such as Xpress-Optimizer (FICO) and CPLEX Optimization Studio (IBM ILOG) softwares are used for solving the models. Constructive heuristics and metaheuristics techniques have also been applied in this research to improve solutions and computational performances.

The last stage is sensitivity analysis and implementation recommendations stage. It includes validation and verification of model as well as a comprehensive sensitivity analysis. The solution obtained was refined for practical feasibility. If necessary, modifications should be carried out in the models and the process in this stage may be repeated as needed.

1.3 SIGNIFICANCE OF THE RESEARCH

The research presented in this thesis focuses on developing models and algorithms for railway CSP. The proposed mathematical model was formulated based on data provided by Queensland Rail (QR), Australia combined with analysis and information retrieved from literature. Some preliminary solutions can be obtained for

railway CSP by solving the models using standard optimisation programming languages such as Xpress-Optimizer (FICO) and CPLEX Optimization Studio (IBM ILOG). However, because of the large number of decision variables and constraints, e.g. a set of crew home depots, a set of relief points, a set of scheduled trips with predetermined starting and ending times at each station, crew breaks, elapsed time and the requirement to return the crews to their home depots at the end of their duty, analytical solution is difficult to obtain especially for large–sized instances. Therefore, metaheuristics techniques such as the Tabu Search (TS) and Simulated Annealing (SA) were applied to solve the problem. Initial solution for both the SA and TS was generated by applying constructive heuristics. For relatively smaller data sets, the underlying problem can be solved to optimality analytically by the proposed MP and CP models, while for larger data sets, it can be solved approximately by the proposed hybrid heuristics/metaheuristics-based algorithms.

This research is significant in developing new models and solution methods for crew scheduling in the railway industry. The proposed models and algorithms consider challenging practical situations for a number of reasons. The model is general and flexible enough to be adapted to different locations and modes of transportation. The optimisation models incorporate a complex set of railway crew scheduling constraints and can be easily adapted to include additional constraints. The optimisation models include a specific real-life constraint in which a crew can be relieved during the interval of relief opportunities. Existing models and algorithms usually only consider relieving a crew at the beginning of the interval of relief opportunities which may be impractical. Allowing the train crew to be relieved at any relief point (RP) during the relief opportunities period (ROP) will provide a better representation of real-world conditions and improve the robustness of the schedule. Several appropriate techniques have been used to solve the models and algorithms, with flexibility in terms of efficiency and scalability. In addition, the proposed models can be solved by a wide range of techniques.

1.4 THESIS OUTLINE

This thesis consists of six chapters. Following this introductory chapter, Chapter 2 gives an overview of CSP along with the models and algorithms which have been used to solve the problem. The objective of this chapter is to acquire an

understanding of the previous researches that have been conducted in this area. It presents the application of mathematical programming approaches, exact algorithms, as well as heuristics and metaheuristics techniques for solving the problem. The literature review process has been conducted throughout the research period to reflect and accommodate new publications. Chapter 3 presents a brief description of the railway CSP and the detailed formulation of railway crew scheduling model. Chapter 4 presents the formulation of the CP model along with its solution techniques. Chapter 5 presents heuristics and metaheuristics techniques to solve the problem. It gives detailed explanation of the proposed constructive heuristics, the hybrid constructive heuristics and metaheuristics, as well as the hybrid metaheuristics method with the computational experiments on each solution approach. Chapter 6 concludes this thesis with some recommendations for further study.

Chapter 2: **Literature Review**

2.1 PROBLEM BACKGROUND AND DEFINITION

The Crew Scheduling Problem (CSP) is a well-known combinatorial optimisation in the transportation systems such as railways, airlines, public mass transit, and buses. CSP is the construction of a minimum cost set of duties for crew members in such a way that each task is covered and all restrictions imposed by governmental regulations, union enterprise agreements, and company policies are satisfied. A task or trip is the part of work to be assigned to one crew. A duty (shift) is a sequence of individual trips that return the crew to its starting point. CSP is one of the most important planning and scheduling problems that can be modelled and solved using mathematical optimisation techniques. The process of crew scheduling in the transportation industry however, is very complex. The large number of tasks (trips) to include and the complicated operational and contractual requirements are the main reasons for the complexity of the problem. Practical work schedules have to be produced which must satisfy a large set of constraints, and might also take into account preferences of individual crew members.

Many factors need to be identified when developing a crew scheduling model. A proper description of the processes involved is necessary before building a suitable model to clearly understand the implication of all related activities. CSP requires definition of the work to be performed in a given planning horizon. While there has been significant work in the area of CSP in general, very few optimisation models have been formulated to solve railway CSP. The developed models and algorithms are mainly designed for a particular condition and might not be readily adapted to another situation.

Railway crew scheduling can be described as follows. There is a railway network where passenger trains travel from one station to the next station according to a published train schedule. There are depots in the railway network to which sets of crew members are allocated. Crew members are required to perform a set of activities to meet the planned schedule. The problem is to construct work schedules for crew members located in the depots such that they comply with the predefined work

practices and regulations. The problem can be approached by identifying the two sub problems of crew scheduling and crew rostering, which may be modelled sequentially. Both crew scheduling and crew rostering problems require finding minimum cost sequences through a given set of tasks. Analysing crew scheduling will be based on depots as personnel bases where crew members are positioned.

The research on CSP has mainly focused on the mass transit and airline industry. The airline CSP, in particular, has received extensive interest due to the high crew related expenses encountered by airlines. Airline CSP has been studied, for example, in (Arabeyre et al. 1969; Barnhart et al. 1995; Chu et al. 1997; Yan and Chang, 2002; Goumopoulos and Housos, 2004; Gopalakrishnan and Johnson, 2005). CSP in railway industry is similar to the CSP in airline industry in terms of time horizons as they may contain short-haul and long-haul trips. CSP in both airlines and railways applications also involve a large number of tasks and a complex set of constraints. Railway CSP, however, is even more complex than in airline because problem instances are much larger than in the airline CSP (Caprara et al. 1998). Moreover, real-world constraints are more difficult to handle such as crew breaks, elapsed time, and other restrictions that must be considered. Also, there is a special constraint that might influence the modelling phase, such as a crew has to return to his/her home depot at the end of his/her daily duty or return to different home depots after a certain period of time.

Early study on CSP for railway application can be found in Tykulsker et al. (1985). This work constructs rail crew schedules and rosters. An enumeration approach, controlled by user-parameters, is used to construct a set of feasible crew duties. This set is reduced with the help of heuristic procedures. A set covering problem is used to select the best from the remaining set of possible crew duties. This program was developed for and implemented at New Jersey Transit Corporation.

Morgado and Martins (1992) also presented early work on the crew scheduling application, ESCALAS for the Portuguese Railways. The system uses a graphic, highly intuitive interface, and allows four different modes of operation, manual operation, semiautomatic operation, fully automatic operation and mixed mode. The main purpose of the development of ESCALAS was to create a decision support system in the area of human resource management. A ‘what-if’ scenario can be conducted through modification of parameters and rules in this system, allowing the verification of consequences of changing labour rules or changing the structure of the

network. The system can generate alternative schedules using different scheduling criteria and enable the evaluation of the cost of the solution to be considered.

2.2 MATHEMATICAL PROGRAMMING APPROACH

Planning and scheduling problems encountered in the transportation industry can be formulated as linear program or linear integer program, or general mathematical program. The most widely used method of solving CSP in the literature has been through modelling CSP as a set covering problem (SCP) or set partitioning problem (SPP). The formulations consist of a binary integer variable that represents whether or not a roundtrip is selected as a work for a crew member. The constraints consist of a matrix of binary values that indicate if a duty *j* covers a task *i*. A row in this 0-1 matrix shows which duties cover a single task. A column corresponds to one possible roundtrip for an individual crew member over a defined time horizon. The constraint in the SCP consists of a matrix of binary values, which defines that each piece of work is covered by a duty at least once. This implies that deadheading is allowed that is the crew can travel as a passenger for repositioning.

The SCP formulation of the CSP is as follows:

$$\textit{Min} \quad \sum_{j \in D} c_j x_j$$

$$\textit{Subject to} \quad \sum_{j \in D} a_{ij} x_j \geq 1 \qquad \forall\, i \in T,$$

$$x_j \in \{0,1\}, \qquad \forall\, j \in D$$

Where: c_j = cost of pairing

$$a_{ij} = \begin{cases} 1 & \text{pairing } j \text{ covers work requirement } i \\ 0 & \text{otherwise} \end{cases}$$

T is the set of trips,

D is the set of feasible duties.

The SPP models the CSP as a problem of finding a minimum cost crew roundtrips that covers each task exactly once. The SPP formulation is as follows:

$$\textit{Min} \quad \sum_{j \in D} c_j x_j$$

$$\textit{Subject to} \quad \sum_{j \in D} a_{ij} x_j = 1 \qquad \forall\, i \in T,$$

$$x_j \in \{0,1\}, \qquad \forall\, j \in D$$

Where: c_j = cost of pairing

$$a_{ij} = \begin{cases} 1 & \text{pairing } j \text{ covers work requirement } i \\ 0 & \text{otherwise} \end{cases}$$

T is the set of trips,

D is the set of feasible duties.

Caprara et al. (1997) formulated crew scheduling for the Italian railways as a set covering problem, using an approach applied for airline crew scheduling. Rail crew scheduling and rail crew rostering are solved by finding a minimum cost sequences of trips and duties, respectively. The crew scheduling phase is formulated as a set covering problem with variables are associated with the circuits of the graph, and solved using an iterative Lagrangean heuristic procedure. According to Caprara et al. (1997), considerable savings can be found through a clever sequencing of the duties obtained in the crew scheduling phase. Therefore, the objective of this phase has to take into account the characteristics of the duties selected and their implication in the subsequent rostering phase. The crew rostering phase is modelled as an integer linear program, with variables are associated with the edges of the graph, which is solved by Lagrangean relaxation and a heuristic technique. Caprara et al. (1997) have noted that the choice of a suitable model and algorithm strongly depends on the particular structure of the problem in hand. The algorithm developed in this research is capable of providing near-optimal solutions to the crew scheduling and crew rostering problems within limited computational time.

A further work by Caprara et al. (1999) provided a thorough overview of the types of crew planning problems that a typical European railway company has to carry

out where track management and service operations are split. This study emphasises the particular experiences of the Italian state railways, which have generated and applied a computerised crew planning system called ALPI, with optimisation based approaches to crew planning problem. The solution of this crew planning system is obtained by decomposing it into three phases, namely pairing generation, pairing optimisation and rostering optimisation. Given a planned timetable for the train services, crew planning is concerned with building the work schedules of train crew (drivers and conductors). The pairing generation phase requires the determination of a set of feasible pairings from the given timetabled trips. A pairing is a trip sequence starting and ending at the same depot, and subject to the specified constraints. The pairing generation phase implemented a depth-first enumerative algorithm and backtracks when infeasibilities are detected. In the pairing optimisation phase an iterative Lagrangean relaxation heuristic is applied for solving a set covering model. The approach consists of three main steps. The first step is aimed at finding a near-optimal Lagrangian multiplier vector. The next step uses the retrieved information provided from the vector and sequences the pairings of the incumbent best solution. The last step is to select a subset of pairings with a high probability of being in an optimal solution. The three-step procedure is iterated and after each application of the three-step procedure, a refining method is applied to improve the solution. This research, however, does not mention how this refining method works. A constructive heuristic procedure then creates one feasible roster at a time by choosing in turn the pairings to be sequenced consecutively. This experiment achieves promising results and points out one possible improvement that could be gained by using a feed-backing mechanism between phase two and three. The use of the set covering model in this study implies that it allows coverage of work requirements more than once, meaning that deadheading is allowed. This study, however, does not show how to handle this situation.

The exact SCP algorithms proposed in the literature can solve instances with up to a few hundred trips and a few thousand duties (Caprara et al. 1997). When dealing with larger problems, one has to adopt heuristic algorithms. Research by Caprara et al. (2001) used constructive heuristics with relaxation techniques to solve the CSP. This research divided the CSP for an Italian railway into three parts: pairing generation, pairing optimisation, and roster optimisation. The problem is one of generating cost

efficient rosters that cover all timetabled train trips. A depth-first branch and bound method is first employed to enumerate all feasible pairings for all depots. Heuristics are used to reduce the feasible pairing set. The study experimentally shows that solution quality can be substantially improved if the pairing optimisation and roster optimisation phases of the process can be iterated on through a feedback mechanism.

Freling et al. (2004) also formulated both crew scheduling and crew rostering problems as an SPP model. They presented a decision support system for airline and railway crew scheduling. The focus of their study is on how to apply a branch-and-price algorithm for a practical application. Both crew scheduling and crew rostering problem are formulated as a SPP model. The crew scheduling sequences trips into duties, and the crew rostering assigns duties for individual crew members. The objectives are to minimise the number of uncovered tasks and minimise the total cost of the duties or rosters selected in the solution. A column generation method is utilised, dealing with a large number of feasible duties or rosters which correlates to a large number of columns. In this research, nodes can be selected by a depth-first-search, best-first-search, or combination of these two. The computational results give an interesting comparison obtained with the approach in which crew scheduling was carried out before crew rostering, and an approach in which these two planning problems were solved in an integrated method.

Kroon and Fischetti (2000) described the intelligent information systems TURNI that are used by the Dutch railway operator NS Reizigers for supporting the planning processes of scheduling train drivers and guards efficiently. The primary model of the TURNI system is a set covering model with a number of additional constraints and is solved by applying dynamic column generation techniques, Lagrangean relaxation and effective heuristics. This study illustrates the use of the software and the underlying model by the Noord-Oost case, which involves the scheduling of train drivers and guards for four interconnected intercity lines of NS Reizigers. The Noord-Oost case was carried out with the aim of obtaining an efficient schedule for the drivers and guards, with a high robustness with regard to the transfer of delays of trains. Apart from the advantage of the powerful algorithm, the TURNI system's drawback is that its user-system interface is relatively simple and that its data handling facilities are limited. Therefore, the findings from this study suggest integrating with the CREW, a commercially available system that uses techniques originating from artificial

intelligence which was customised to the situation at NS Reizigers. This integration could provide an intelligent information system for supporting the scheduling of train drivers and guards.

Subsequent study by Kroon and Fischetti (2001) described the use of a set covering model with additional constraints for scheduling train drivers and conductors with the objective of obtaining an improved quality and punctuality in the train services provided to the customers. This project generates efficient and acceptable schedules for the drivers and conductors, with a high robustness with respect to the transfer of delays of trains. The set covering model is solved by dynamic column generation techniques, Lagrangean relaxation and constructive heuristics. The experiment successfully solved instances with up to about 2500 trips which could be handled effectively within a computational time of about one hour. A promising result was also found in a number of experiments with much larger instances. Given the complexity of the practical crew scheduling problems, this research suggests that the most important part of an intelligent system to provide a practical solution of the problems should be a powerful algorithm that does not only consider the feasibility of the individual duties, but also the feasibility of the whole schedule.

The main difficulty with the SPP and SCP formulations of the CSP is that in determining all possible roundtrips. In the SPP and SCP formulations, the matrix of constraints contains columns for every possible roundtrip. For a CSP with a large number of trips, it will produce an extremely large number of columns hence, an unmanageably large number of possible roundtrips. Because of the large number of decision variables associated with combinatorial explosion of the problem, there is a requirement of large-scale optimisation techniques such as column generation. Therefore, for a very large CSP, column generation-based techniques are employed to solve the SCP or SPP formulations.

The concept of column generation is to solve a sequence of reduced problems (master problem) in which each reduced problem contains a small fraction of the set of variables (columns). When a reduced problem is solved, a new set of columns (sub-problem) is obtained by using dual information of the solution. The sub-problem or an auxiliary problem is usually formulated as a restricted shortest path problem. The restricted shortest path problem however, is difficult to solve and it also needs other

optimisation methods such as dynamic programming algorithms or branch-and-bound methods.

Research by Freling et al. (2001) presented a heuristic algorithm for scheduling train crews at Dutch Railways (NS). The railway crew scheduling problem is formulated as a set covering model with side constraints. The side constraints correspond to the high level constraints dealing with sets of duties. Medium level constraints deal with the construction of paths during the column generation procedure, while low level constraints deal with the construction of the network. This research uses a column generation approach to solve the LP relaxation of the IP formulation and a branch-and-price heuristic to find integer solutions. New columns are generated implicitly using a dynamic programming algorithm. Several acceleration techniques are applied to speed up the algorithm in order to solve a larger-scale train crews scheduling. The researchers, however, claimed that as the approach is general in nature, it can be applied to the CSP in different contexts. Although the results are very promising, care is needed when drawing conclusions based on one instance only. Therefore, a further research effort is necessary to test the algorithm and its variations on other instances as well.

Alfieri et al. (2007) presented the case of scheduling train drivers on a railway sub network. Train driver scheduling involves the construction of feasible duties from a set of trips to be serviced by a number of train drivers. Each duty consists of a sequence of trips to be carried out by a single train driver on a single day. The duties should be such that each trip is covered by at least one duty, each duty satisfies feasibility constraints, and additional constraints involving the complete schedule are satisfied while one or several objectives are met. This research also uses a set covering problem based on an implicit column generation solution approach and focuses on minimising the number of duties and on maximising the robustness of the obtained schedule for outside disruptions. A heuristic procedure is presented to find an initial feasible solution together with a heuristic branch-and-price algorithm based on a dynamic programming algorithm for the pricing-out of columns. This approach is tested on the timetable of the intercity trains of NS Reizigers, the largest Dutch operator of passenger trains. Although the proposed approaches are considered acceptable, the findings suggest an improvement in the algorithm by studying further several issues such as how many columns are to be added in each iteration of the

pricing algorithm for speeding up the convergence of the algorithm, and how to find the best stop criterion for terminating the column generation in a certain node.

Another research which studied large scale crew scheduling problems arising at the Dutch railway operator, Netherlands Railways (NS) was conducted by Abbink et al. (2007). They presented several methods of partitioning large instances into several smaller ones. These approaches are used to create a weekly crew schedule for drivers and conductors. The four different partitioning methods are weekday partitioning, geographical partitioning, line based partitioning, and column information partitioning. These smaller instances are then solved with the commercially available crew scheduling algorithm TURNI. These partitioning methods are then compared with each other. It is reported that all methods significantly improve the solution. The mathematical model for the CSP containing two days without tasks overnight is as follows. T^1 and T^2 are the set of tasks for day 1 and day 2, respectively. D^1 and D^2 denote the set of duties for these days. The subset D_i^1 (D_i^2) of D^1 (D^2) consists of the set of duties containing task i. The binary decision variables x_j (and y_j) indicate whether duty $j \in D^1$ (D^2) is included in the solution or not. Every duty j has positive costs c_j. Moreover, S is the set of additional constraints and l_s and u_s are the lower and upper bound for constraint $s \in S$. Finally, v_j^s (and w_j^s) are the weight of duty $j \in D^1$ (D^2) for constraints s. Then the CSP formulation is as follows.

$$\textit{Min} \quad \sum_{j \in D^1} c_j x_j + \sum_{j \in D^2} c_j y_j \tag{1}$$

$$\textit{Subject to} \quad \sum_{j \in D_i^1} x_j \geq 1 \qquad \forall i \in T^1 \tag{2}$$

$$\sum_{j \in D_i^2} y_j \geq 1 \qquad \forall i \in T^2 \tag{3}$$

$$l_s \leq \sum_{j \in D^1} v_j^s x_j + \sum_{j \in D^2} w_j^s y_j \leq u_s \qquad \forall s \in S \tag{4}$$

$$x_j \in \{0,1\} \qquad \forall j \in D^1 \tag{5}$$

$$y_j \in \{0,1\} \qquad \forall j \in D^2 \tag{6}$$

Equation (1) is the objective function, which states that the sum of the duty cost is minimised. Constraints (2) and (3) guarantee that for each task i, at least one duty

that contains this task is selected. Note that only duties of day 1 (2) can contain tasks of day 1 (2). It sometimes may be better to perform a task more than once. If, for example, the number of tasks going out of a crew base differs from the number of tasks going into the crew base in one day, over-covering is necessary. Moreover, even if over-covering is unnecessary, it may be cheaper to allow over-covering. Constraint (4) is an additional constraint. For example, a crew base for which the total number of duties on both days is limited to 50. Then, $l_s = 0$ and $u_s = 50$ and v_j^s (w_j^s) = 1 for all duties belonging to this base and v_j^s (w_j^s) = 0 for all other duties. For some additional constraints it is allowed to violate the constraint at the cost of a penalty. The last two sets of constraints (5, 6) indicate that the decision variables are binary. Even though they were able to solve the problem, this approach could not model the problem completely.

Bengtsson et al. (2007) also studied the crew pairing problem at the large European railway, Deutsche Bahn. A mathematical formulation of the general crew pairing problem is presented with the objective is to minimise the cost of selected pairing and the cost of violating soft constraints.

Kwan (2010) discussed a case study of an automatic optimising train crew scheduling system, TrainTRACS. The optimisation technique of TrainTRACS is formulated mathematically as an integer linear program (ILP) based on set covering. Train crew scheduling is partitioned into segments, which are permuted and recombined with breaks and other crew activities to form crew shifts. The objective function is to minimise the total cost and the total number of crew shifts, subject to the constraints that each train work piece has to be covered by at least one shift, and that using a fraction of a shift is not allowed. The mathematical model is as follows.

$$\textit{Min} \quad w_1 \sum_{j=1}^{n} c_j x_j + w_2 \sum_{j=1}^{n} x_j \qquad (1)$$

$$\textit{Subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad i = 1, 2, ..., m \qquad (2)$$

$$x_j = 0 \text{ or } 1 \qquad j = 1, 2, ..., n \qquad (3)$$

Where n is the number of candidate shifts, m is the number of work pieces, c_j is the cost of shift *j*,

$$x_j = \begin{cases} 1 & \text{if shift } j \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

$$a_{ij} = \begin{cases} 1 & \text{if work piece } i \text{ is covered by shift } j \\ 0 & \text{otherwise} \end{cases}$$

w_1 and w_2 are weight constants.

The ILP is solved in two stages. The first stage ignores the integer constraints. The relaxed LP is solved using the revised simplex method with column generation. The second stage uses the branch-and-bound method to search for an all integer solution. The optimal continuous solution from the first stage is utilised in setting the target integer number of shifts to be used and making branching decision in the second stage. Given a set of train unit diagram, a solution of crew schedule consists of a set of legal shifts. The construction of a shift must conform to the general working condition of the crew. Within a shift, a crew normally takes a meal break and the shift usually contains two or more spells of work. A spell is a period of time a crew works continuously on a train. Because of the train crew scheduling is a very hard combinatorial optimisation problem TrainTRACS takes many years of incremental research and development and is currently mainly used by UK's train operating companies.

Lezaun et al. (2007) described an applied study conducted by the regional rail passenger carrier EuskoTren (Spain), on how to assign drivers' annual workload equally. The allocation is subject to the working conditions' requirements and the preferences of employees. To meet such requirements the company assigns a higher work-load to those drivers willing to take it, counting the hours as overtime if the driver has already been assigned full complement of hours, or else hires additional drivers. The proposed solution is obtained in four related steps, at each of which a binary programming problem is solved using commercial software. Step one builds five lists of weekly multi shift patterns that contain all the shifts in the week. Step two involves the partially rotating annual assignment of patterns to drivers, while step three includes the extraction of shifts by reducing services on public holidays. The last step incorporates the duration in hours into the shifts already assigned. The achieved solution is able to assign all drivers a similar number of morning, evening, and night

shifts with Sunday off nearly the same number of days and hours per year. Although the obtained result in this study is satisfactory, Lezaun et al. (2007) still have suggested a possibility for improvement.

Beasley and Cao (1996) described a tree search algorithm for solving a generic crew scheduling problem. The crew scheduling problem is defined as a problem of assigning K crews to N tasks with fixed start and finish times such that each crew does not exceed the limit of the total working time. To provide a lower bound, a Lagrangean relaxation of a zero-one integer programming formulation is applied and it is improved by sub gradient optimisation. A tree search (branch-and-bound) procedure is then applied to find optimal solution of the crew scheduling problem. Computational results indicate that the proposed algorithm can solve relatively large-sized problems.

A further work by Beasley and Cao (1998) studied a dynamic programming based algorithm for solving a generic crew scheduling problem. A Lagrangean based penalty procedure is used to derive a lower bound and sub gradient optimisation which is then used to maximize the lower bound obtained from the previous step. Computational results show that the developed algorithm can solve large problems optimally.

Chu and Chan (1998) presented the problem of crew scheduling for the Hong Kong Light Rail Transit (LRT), a rail transit division of Kowloon-Canton Railway Corporation. The project aims at automating the complex schedule construction, adopting an optimisation modelling approach, and amenable for decomposition into separate solution stages. The resulting crew schedule has been constructed iteratively in a reasonable computational time. Although some issues in the LRT system are not incorporated in this produced software, Chu and Chan (1998) have stated that it can be used as a good starting point for further crew schedule constructions, leading to a continuous improvement on the LRT's scheduling system.

Ernst et al. (2001) presented the crew scheduling problem faced by Australian railways and developed an optimisation model that constructs crew parings and rosters. Given a rail network with a number of depots and train trips, this study first distinguishes between the planning problem, which is one of deciding the total number of crews and their distribution across the network. The problem is formulated as an integrated model to generate cyclic rosters and non-cyclic rosters in which under-coverage and over-coverage of specific duties is allowed. Even though they have

included several constraints, they still have solved a relaxed method of the problem and suggest the need for a better method in their conclusions.

Caprara et al. (1998) proposed a general model of crew rostering problems (CRP) for airline or railway applications. The CRP objective is for finding a feasible set of rosters, covering all the duties and minimising the total number of weeks in the rosters. The global number of crews required every day to cover all duties is equal to k times the total number of weeks. Thus the minimisation of the number of weeks implies the minimisation of the global number of crews required. Integer Linear Programming (ILP) model for CRP is based on the graph-theoretical formulation which is relaxed in a Lagrangean way and obtains the objective function. A constructive heuristic then uses the information obtained from the solution of the previous relaxed problem to constructs one roster at a time. The heuristic chooses in turn the duties to be sequenced consecutively in the roster. When a roster has been completed, all the duties it contains are removed from the problem. The process is repeated on the remaining duties until all duties have been sequenced. The approach has been applied on to the real-world railway CRP proposed by Ferrovie dello Stato SpA within the FARO competition. The computational results from this railway application involving up to 1000 duties shows that the proposed approach achieves lower and upper bound values that are typically very close, within a short computational time.

Sodhi and Norris (2004) presented a general modelling approach to crew rostering at the London Underground. The approach decomposes the overall crew rostering problem into stages, which are solved with a general mixed integer linear programming (MILP) solver, graph-theoretic, manual approaches, and allowing general solution techniques to be applied at each stage. The primary objective is to maximise the weighted sum of regular weekends, of pairs of consecutive days off not including weekends, and of long weekends. The secondary objective is to minimise the violation of soft constraints for the specific duty assignment. Despite the ability of this computer-assisted approach to solve crew rostering for the London Underground, Sodhi and Norris (2004) have noted that the shortcoming of this approach is its inability to handle constraints that cannot be captured at the node or arc level.

2.3 NETWORK FLOW APPROACH

The network flow approach has been used in several studies on the CSP. An attempt towards this approach was proposed by Vaidyanathan et al. (2007), who described a network flow-based approach to solve the railroad CSP arising on North American railroads. The CSP is formulated as an integer program on a space-time network, enforcing the first-in-first-out requirement by including side constraints where the objective is to minimise the total cost of crew wages, the cost of deadheading, the cost of crew detentions, and the cost of train delays. The network is created in such a way to accommodate all Federal Railway Administration (FRA) regulations and trade union work rules. The computational results in this research show that the perturbation method outperforms the other approaches in terms of solution time and solution quality. Yan and Tu (2002) introduced a network model to solve an airline cabin crew scheduling. The network simplex method is applied to solve the problem. A flow decomposition algorithm is then applied to get the pairing from the integer solutions. The network flow approach however, is difficulty to apply in highly constrained practical-sized optimisation problems. Therefore, this approach has been successful for small- to moderately-sized real-world problems.

2.4 METAHEURISTICS APPROACH

Metaheuristics have become a popular approach in tackling the complexity of practical scheduling problems. Metaheuristics are typically high-level heuristic strategies which guide the search to avoid being trapped in local optima. The main limitation with many of conventional heuristic algorithms is their difficulty to escape from locally optimal solutions. The search is usually conducted from a single point in the solution space and continuously searches for better and improved solutions until there is no possible improvement. In an attempt to deal with this problem, several metaheuristics approaches have emerged such as Simulated Annealing (SA), Tabu Search (TS), and Genetic Algorithms (GAs).

SA derives from physical science, notably the second law of thermodynamics. The method is motivated by an analogy to the physical process of annealing, where the temperature of a material is reduced to achieve its thermal equilibrium (Kirkpatrick et al. 1983). This principle is applied in combinatorial optimisation problems to

minimise the objective function value. The advantage of this technique is that it can avoid local optima by allowing the acceptance of non-improving solutions occasionally in the hope that a better solution may be found later on.

SA has been applied to the CSP by a number of researchers. Emden and Proksch (1999) solved an airline CSP using a SA approach. The result indicates that the SA produces good quality solutions, but requires longer processing times than simpler heuristics. Lucic and Teodovoric (1999) also applied a SA approach to solve a multi-objective CSP for airline pilots. In general, SA based solution approaches to the CSPs have produced acceptable near-optimal solutions but have not been shown to be as effective as other methods.

TS has also been used to solve the CSP. TS is a higher level heuristic introduced by Glover (1977, 1986). TS is an iterative improvement procedure and it can be combined with other search techniques to make the search more efficient. TS avoids becoming trapped in local optima by exploiting memory and data structures that prevent cycling and induces the exploration of new regions. One of the attempts towards this direction was proposed by Cavique et al. (1999), who solved a CSP for Lisbon Underground. A heuristic search is proposed to minimise the number of necessary duties for a determined planning period. The methods based on the TS obtain better results in terms of solution quality even though require longer computational times.

Shen and Kwan (2001) used TS in transit CSP, but produced solutions that were substandard to other methods. They, however, show that refining a TS procedure has the potential to produce better solutions. Chew et al. (2001) presented a report on an optimisation-based approach to develop a computerised train-operator scheduling system that has been implemented at Singapore Mass Rapid Transit (SMRT). The approach includes a combination of optimisation techniques, a bipartite matching algorithm and a TS algorithm. The objective is to minimise the system wide crew-related costs and to address concern with respect to the number of split duties. The developed system improves and automates the current manual scheduling process at SMRT and produces applicable schedules in comparison with the manual process. Cabrera and Rubio (2009) conducted a research using the TS approach for solving a CSP. The objective is to assign a subset of the duties to each crew in such a way to minimise cost and no trip is left unassigned by considering a number of constraints.

The problem is retrieved from a railway public company in Valparaiso, Chile called Metro Regional Valparaiso (MERVAL), which served from Valparaiso to Limache. The findings from this study show that the hybrid algorithm produces very good solutions in adequate computational time. They have suggested hybridisation of other different solution techniques to improve resource allocation in the transportation industry.

Elizondo et al. (2010) proposed methods to address operation management problems which emerge in underground passenger transport. The problem is to generate duty and identify an optimal trip set that the conductors should complete in one workday. The objective of the problem is to minimise the number of conductors required and minimise total idle time between trips. The problem is modelled and solved using a constructive hybrid approach. The obtained results are compared with two alternative approaches, based on TS and a greedy method. The TS technique provides better results in terms of idle time than both the hybrid and the greedy methods. Although their proposed methods address the two objectives separately, they have suggested solving the problem as a multi-objective optimisation.

As was explained earlier, many of the traditional heuristic algorithms use iterative improvement techniques where the search is started at a single point in the search space. During a single iteration, a new point is selected from the current point. If the new point provides a better solution, then the new point becomes the current point. It is clear that such a local search only exploits the best solution for possible improvement. Random search such as SA, on the other hand, explores the search space ignoring the exploitation of the promising regions of the space. Genetic Algorithms (GAs) emerged as a powerful technique by performing these two objectives, exploiting the best solutions and exploring the search space. As Michalewicz (1996) has noted, GAs are a class of general purpose (domain independent) search methods that strike a remarkable balance between exploration and exploitation of the search space.

GAs have been successfully applied to optimisation problems for planning and scheduling in the transportation contexts. GA-based approaches have been used in several crew scheduling related problems and have produced good quality solutions, but cannot guarantee optimal solutions. Research which was conducted by Lee and Chen (2003) found that a GA approach provides a flexible structure for driver

scheduling problems with multiple objectives and various constraints. This research applies a mathematical programming approach and a GA approach for solving a driver scheduling problem at Taiwan Railway administration (TRA). The problem consists of generating feasible duties, creating a schedule of duties for a depot, and circulating duties in the schedule into a roster for each driver. Some heuristics methods are developed and used for generating pairing. A bi-objective SCP is then used in the pairing optimisation phase with the primary objective and the secondary objective are to minimise the total number of duties and to minimise the compensation for the selected duties, respectively. A commercial software LINDO is used to solve the linear integer programming. The outputs of the pairing optimisation are passed into the rostering phase to generate individual rosters for a depot. The GA uses a binary coding scheme for the SCP while for the rostering problem uses non-binary coding scheme. The study indicates that the solution obtained by GA in general gives better results for the TRA and provides a flexible structure for driver scheduling problem with multiple objectives and various constraints.

Park and Ryu (2006) proposed a GA to solve the pairing optimisation for subway CSP. The pairing optimisation is modelled by using a maximal covering problem (MCP) and is solved by applying the developed GA. The GA employs greedy heuristics in crossover and mutation operators to improve the efficiency of the search. A new chromosome structure incorporates unexpressed genes as a way of preserving diversity of population. While the genes in both expressed and unexpressed parts evolve, only the genes in the expressed part are used when an individual is evaluated. Experiments with large real-world data have shown that the GA outperforms other level search algorithms such as SA and TS.

Metaheuristics have become a popular approach in tackling the complexity of practical optimisation problems such as the CSP. Although metaheuristics cannot guarantee optimality of their solutions, they have shown a very good performance in solving real-world optimisation problems. Metaheuristics represent a general type of solution method that illustrates the interaction between local improvement procedures and higher level strategies to facilitate the algorithm for both escaping local optima and exhaustively searching a feasible region. By applying metaheuristics, a good feasible solution for a large number of input data can usually be obtained in a reasonable amount of computational time.

2.5 CONCLUSION

In this chapter, a survey of the CSP has been presented. The objective of this chapter is to acquire an understanding of the previous researches that have been conducted in this area. The literature review process has been conducted throughout the research period to reflect and accommodate new publications. The review has revealed that very few studies have been conducted on the practical crew scheduling in the railway industry. The existing approaches in solving the CSP are classified according to the exact mathematical models and algorithms, heuristics and metaheuristics. As the CSP belongs to the class of NP-hard, the efforts have been directed to the use of metaheuristic algorithms which are capable of producing good feasible solutions within reasonable computational time.

Chapter 3: **Railway Crew Scheduling Model**

3.1 SCHEDULING PROBLEM

Scheduling is the process of allocating resources to activities over time. In a typical scheduling problem, resources are scarce and constrained in various ways. Most practical scheduling problems belong to a special class of NP-hard problems for which no polynomial time algorithm has been found. The algorithms in this class normally have exponential time behaviour, and hence there is no fast solution method exists for the problems yet. This means that, in the case of large scheduling problems, no optimal solutions can be found in a reasonable computational time. Therefore, we have to be satisfied with a feasible schedule rather than optimal schedule for a given problem.

Real-world optimisation problems are usually too complex to capture all details. A model is usually a simplification that provides a sufficiently precise representation of the main features such that the solutions obtained to the problem under study still remain valid to an acceptable degree of approximation. Therefore, developing a mathematical model usually involves making approximations and adjustments, and sometimes ignoring or relaxing features which are difficult to formulate.

Because of the complexity of the scheduling problems, several different solution approaches have been offered such as through the application of mathematical models and optimisation methods, simulation techniques, artificial intelligence techniques, expert systems techniques, and metaheuristics techniques.

CSP involves real-life constraints which are difficult to handle, such as crew breaks, elapsed time and the requirement to return the crews to their home depots at the end of their duty. Furthermore, an optimisation model should be well designed such that all relevant parameters related to the problem can be incorporated. One way of dealing with the problem is to invent a specific model representation that is capable of incorporating important features of the problem, and it can be solved using a wide range of methods. This research develops a mathematical optimisation model for

railway CSP. The objective of the model is to minimise the number of crew duties by minimising total idle transition times. The optimisation model incorporates a complex set of railway crew scheduling constraints encountered in real-life situation. The integration of relief opportunities period (ROP) into the model, in particular, offers flexibility in where and when crew can be relieved. This will enhance the robustness of the schedule and provide a better representation of real railway crew scheduling conditions. Although the optimisation model presented in this chapter, was designed in the context of railway CSP, it is general and can be easily adapted to different locations and modes of transportation. Furthermore, the proposed mathematical model in this research can be extended to the integration of vehicle and CSPs with ROP.

3.2 CREW SCHEDULING PROBLEM (CSP)

The generic CSP is the construction of a minimum cost work schedule for crew members in such a way that all restrictions imposed by governmental regulations, union enterprise agreements, and company specific rules are satisfied. The CSP includes two sequential and interconnected sub problems, the crew pairing problem and the crew rostering problem. The crew pairing problem, which is the focus of this study, is the assignment of crew members to scheduled tasks (trips) that have to be serviced subject to operational and contractual requirements. The assignment of crews seeks to find a minimum cost sequences of a given set of trips. The sequence of trips to be carried out by one crew is called a duty (pairing, roundtrip) and it is usually last for one day (a working day of a crew). Then the subsequent problem, the crew rostering problem, is the arrangement of the generated duties into sequence of duties to be performed by an individual crew member over a defined period of time.

3.3 RAILWAY CREW SCHEDULING PROBLEM

Railway CSP involves a rail network where trains travel along specified train lines from one station to a subsequent station according to the published train schedule. There are a number of depots in the railway network to which each crew member is positioned. Crew members are responsible for performing a given set of activities to meet the train schedule. The railway crew scheduling under study consists of a set of crew home depots (HDs), a set of relief points (RPs), a set of scheduled train trips with

fixed starting and ending times at each station. The problem is to construct minimum cost crew duties based on the train timetable while satisfying operational and contractual requirements.

The crew in this case is the train crew which consists of a train driver and a conductor, and they are considered as a team. A crew typically operates a train starting from a HD, travelling from one station to the next, taking a break at a specified location (relief point) within a specified time (relief time), and then operates another train back and terminate at the same HD. The railway CSP in this context is to specify the sequence of trips to be performed by the crew.

The passenger railway operator in this study was Queensland Rail (QR) Australia. The railway operator offers regular train service on specified lines in the network. A line is characterized by a departure station and an arrival station with a number of intermediate stations. When the line is served by a single crew, the sequence of trips can be treated as an individual trip. Examples of such lines are the Ferny Grove (FYG) line and the Beenleigh (BNH) line. The FYG line mainly serves the train trip from FNY to Park Road station. However, there are also scheduled trips from FNY to other terminal stations such as Cleveland (CVN), Corinda (CQD), and Bowen Hills (BHI). The same situation also occurs on the other lines in the QR network. The QR network is shown in Figure 3.1.

Figure 3.1 Map of QR Network.
Source: www.queenslandrail.com.au

The train crew scheduling in this study has the following inputs. A set of HDs and a set of RPs, a set of scheduled trips with fixed starting and ending times, and predetermined driving times between all pair of stations. A train crew duty (shift) contains a meal break (MB) which starts and ends within a specific period as determined in the union collective agreement. The MB begins after the completion of the third hour and finishes before the completion of the sixth hour, relative to the start of the duty (shift). For example, crews sign on at 08:00 and sign off at 16:00, then the earliest MB will be at 11:00 to 11:30 and the latest MB will be at 13:29 to 13:59. Figure 3.2 illustrates two possible alternatives of crew relief. The first condition is when 0.5 h MB occurs at the earliest time and the second condition is when the MB occurs at the latest time. A sequence of trips in a duty (shift) is shown in Figure 3.3. As can be seen from this figure, a MB divides a duty into two partial duties with different durations. The 1st part of a duty is the period from the start of a duty to the start of the MB, whereas the 2nd part of a duty is the period from the end of the MB to the end of the duty. Every single horizontal blue bar in Figure 3.3 represents a trip. Transition time or turnaround time within a partial duty is the time incurred between trips which may include the time required by the crew to move from one end to the other end of the train at platform and drives the train away in the opposite direction. Transition time in a duty (shift) is the time period between trips of different partial duties (MB) which includes crew relieving related activities.

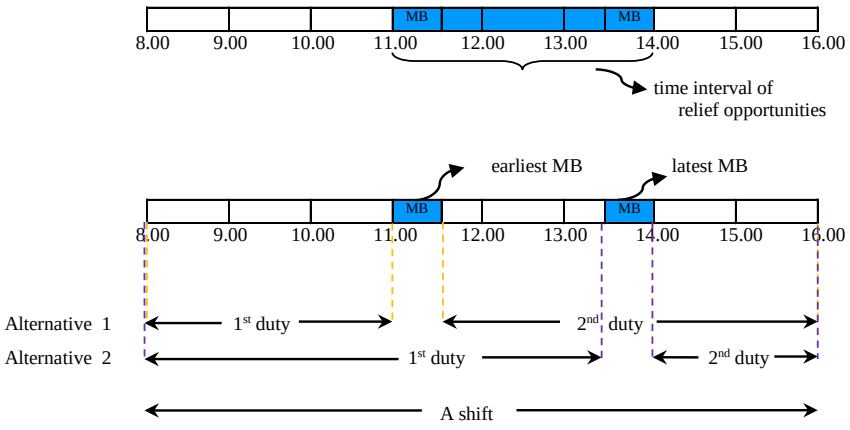

Figure 3.2 Two possible combination of partial duties in the shift.

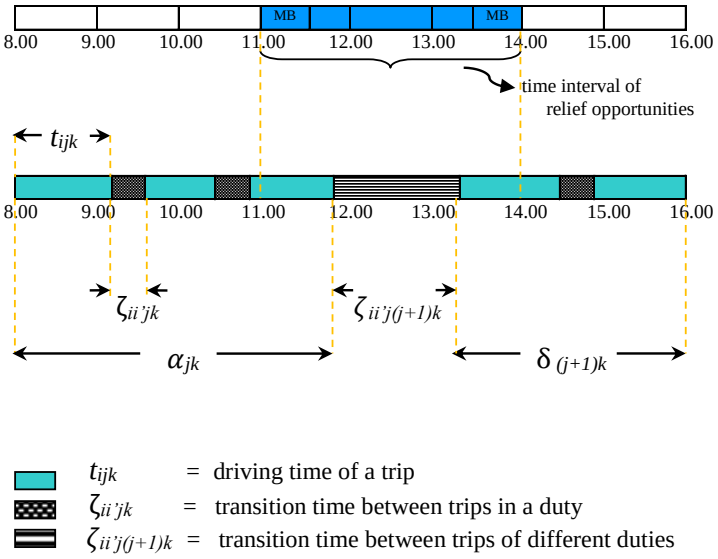

Figure 3.3 A sample of the sequence of trips in the shift.

The railway network involves interconnected segments of train tracks. Each segment of train journeys consists of a sequence of trips that must be serviced. Figure 3.4 illustrates an example of a train timetable. The route of trains can be traced by straightening the traveling path of trains in the train timetable. Each trip in the timetable must be serviced by a train. The railway CSP is to specify the sequence of trips to be performed by the crew. A train journey begins and ends at a crew HD, and can feasibly be serviced by a single crew.

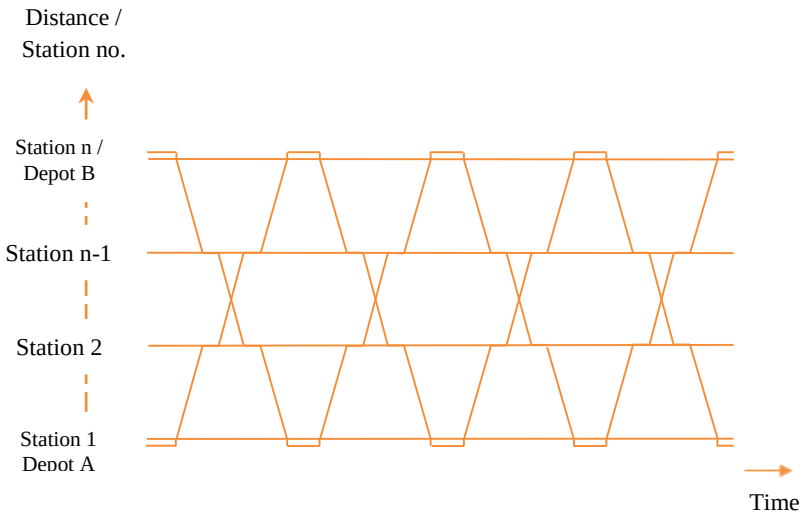

Figure 3.4 An example of a train timetable.

A train service is the overall journey accomplished by a vehicle from the time it begins at its first station until it arrives at its last station. A vehicle block specifies the sequence of trips made by a train during a service work day. It contains pieces of segments in which crew relief may be done at the both end of each segment. Each crew belongs to one crew base (HD) and the crew has to start (sign on) and end (sign off) his/her duty (daily work shift) at the same crew depot (HD). The spread time is the time elapsed between the crew sign on and the crew sign off in a duty. The time interval between the earliest break and the latest break corresponds to the transition period between two consecutive pieces of duty, and is defined as a relief opportunities period (ROP). The ROP is a period of time within which a train crew is allowed to be relieved. Any RP can be chosen for crew relief within the two limits of the ROP. The set of

crew HDs is a subset of the set of RPs. This transition period includes the time spent for taking a meal and other crew relieving related activities such as handing over a train to (from) another train crew. An example of vehicle blocks and a crew duty with ROP is shown in Figure 3.5.

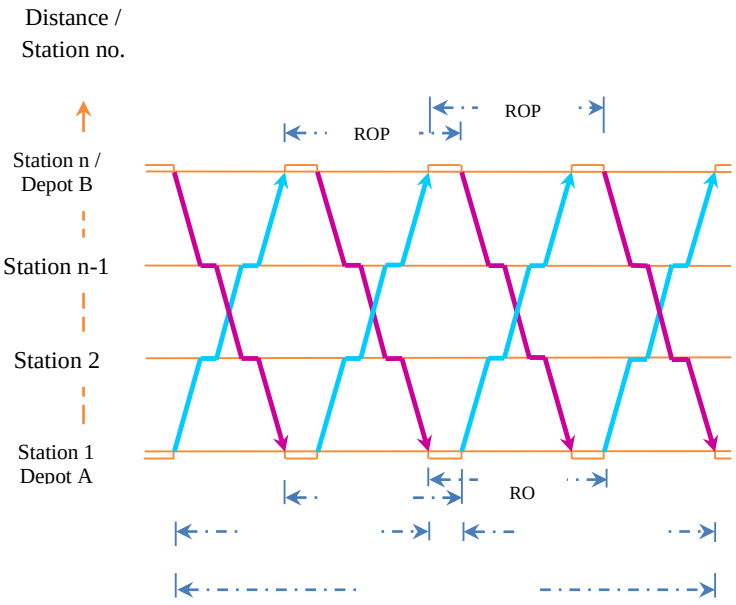

Figure 3.5 An example of vehicle blocks with ROP.

The path of a train is indicated by the blue lines and the purple lines, as shown in the diagram of Figure 3.5. The blue lines show the movement of a train from the Station 1 (Depot A) to the terminal at Station *n*, with transition times (short dwell times) at each station. Crew arriving at stop *n* can be relieved at this point and take a MB at an away depot (Depot B). The relieved crew may then continue with another vehicle block passing through the same terminal station or RP. Alternatively, the crew may return directly along the route in the opposite direction (the purple lines) and take a MB at the HD (Depot A). When more trips are considered, the network becomes denser and more paths need to be evaluated. A duty covers a set of consecutive trip segments in a block. The 1st part of a duty (duty stretch) is the period from the start of a duty to the start of the MB, whereas the 2nd part of a duty is the period from the end

of the MB to the end of the duty. Transition time (idle interval) between two consecutive trips in each partial duty is the time incurred between the departure time of the next trip and the arrival time of the previous trip.

The railway transportation industry imposes a complex set of operational and contractual requirements correspond to the work regulations for the crew. For safety reasons, for example, there is a restriction on the length of continuous driving time. A crew will be required to take a break when the total continuous driving time on the same vehicle has reached a maximum limit. In the formation of duties, crew schedule should satisfy several constraints corresponding to work load regulations. There are predetermined maximum and minimum durations of a duty. A minimum of 0.5 h for a MB is required in a duty (shift). A crew takes a break only at a RP and the changeover of trains is at the same RP.

3.3.1 Terminology

As there is no uniform terminology for the CSP in the literature, the following terms are used in this study to clearly define the problem.

- *Trip* is the movement of a train from one station to the next station at specified times according to the train timetable. Each trip is characterised by a train, a departure time, a departure station, an arrival time, and an arrival station.
- *Trip segment* (or *segment* for short), is part of a vehicle block contains a sequence of trips in which crew relief can be performed at both ends of each segment.
- *Crew* is the train crew which consists of a train driver and a conductor and they are considered as a team.
- *Vehicle block* is the sequence of trips assigned to the same vehicle during a service workday. It contains pieces of segments in which crew relief may be performed at both ends of each segment.
- *Duty* (*shift*) is the work to be carried out by a train crew in one day. It begins and ends at the same crew depot. In a duty, there are two partial duties and a MB in between. Partial duty is the sequence of trips serviced by a train crew on the same vehicle without a long rest. The 1st part of a duty (duty stretch) is the period from the start of a duty to the start of the MB, whereas the 2nd part of a duty is the period from the end of the MB to the end of the duty.

- *Transition time* (*idle interval*) is the time incurred between two consecutive trips in the same partial duty. It is the difference between the departure time of the next trip and the arrival time of the previous trip.
- *Driving period* is the total time spent by a train crew driving trips in the duty including transition times (short dwell times) between consecutive trips.
- *Home depot* (*HD*) is the work location at which a train crew is positioned to begin and end his duty.
- *Relief opportunities period* (*ROP*) is the time period within which a train crew has a chance to be relieved at a specified location.
- *Relief point* (*RP*) is the location at which a train crew can be relieved.
- *Working time* is the total duration of train crew working from the start to the end of the duty, performing both driving activities and non-driving activities.
- *Changeover* is the relieving of a train crew and taking over the responsibility for the train by another crew.

3.3.2 Input Data

Train timetables generally show all train movements along a particular route. Each entry of the timetable specifies element of the sequence of trips assigned to a particular train with time and location of train stops. The timetable lists the route to follow by trains through the rail network with the times and locations of scheduled train services. The input data comes in the form of a two dimensional table where each row represents one trip. A sample train schedule with 12 trips is given in Table 3.1. The data are retrieved from the operational environment of lines in the railway network. By considering this initial segment, we derived artificial data as problem instances.

Table 3.1 A sample train schedule with 12 trips.

Train ID	Departure Station (*ds*)	Departure Time (*dt*) (hh:mm)	Arrival Station (*as*)	Arrival Time (*at*) (hh:mm)
L001	A	05:00	B	05:24
L002	A	05:30	B	05:54
L003	A	06:00	B	06:24
L001	B	05:25	C	06:35
L002	B	05:55	C	07:03
L003	B	06:25	C	07:34
L201	C	04:28	B	05:41
L202	C	04:59	B	06:08
L203	C	05:27	B	06:37
L201	B	05:42	A	06:08
L202	B	06:08	A	06:38
L203	B	06:38	A	07:02

3.3.3 Operational and Contractual Requirements

The railway transportation industry imposes a complex set of operational and contractual requirements correspond to the work regulations for the crew. The regulations are defined in the enterprise agreement and in the company policies and procedures. For a safety reason, for example, there is a restriction on the length of continuous driving time. A crew will be required to take a break when the total continuous driving time on the same vehicle has reached a maximum limit. Feasible crew duties should satisfy several constraints corresponding to work load regulations. The formation of a feasible crew schedule is restricted by a complex set of rules and regulations. The rules and regulations may vary between railway operators. The commonly applied rules for railway CSP considered in the proposed model are given as follows:

- The minimum and maximum duration of working time;
- The maximum spread time (elapsed time). The spread time is the time elapsed between sign-on and sign-off in the duty (shift);
- The maximum continuous driving time. A crew will be required to take a break when the continuous driving time has reached a certain limit;

- There must be a MB of at least 0.5 h between the third and the sixth hour relative to the start of the duty (shift). A MB is required in between consecutive partial duties;
- A crew has to start and end (sign-on and sign-off) his/her daily shift at the same depot;
- The crews take a break only at a RP and the changeover of trains is at the same RP.

A crew schedule should also take into account extra time needed by the train crew to perform other functions associated with crewing of a train. These include preparing a train at the first start of a run, handing over (taking over) a train to (from) another train crew, secure a train at the end of its run and the time spent by the crew walking to (from) the signing-off (on) point from (to) the train.

3.4 DEVELOPMENT OF MATHEMATICAL MODEL

The scheduling problem is modelled as equations written in a set of algebraic notations. Solving the problem would then require solving the equations that represent the problem. Mathematical programming has three main components, namely a set of decision variables, constraints over these variables and an objective function to be optimised. The objective function represents the goal of the problem in terms of decision variables. It is a function of the variables which is used to navigate and select among possible solutions. Constraints put a limit on the outcome of finding a solution. The decision variables are the unknown values or decisions that are to be optimised. In this case, the basic assumption is that the problem is linear. In mathematical terminology, the objective is a linear function and the constraints are linear equations and inequalities. Such a problem is called a linear program and the process of modelling and solving this problem is called linear programming. Mixed Integer Programming (MIP) consists of constraints and an objective function where decision variables may have either discrete or continuous domains. Thus, in MIP some of decision variables required to have integer values.

Railway crew scheduling is considered as a highly constrained problem. The problem of forming a minimum cost set of duties which cover all the trips in a timetable is a difficult combinatorial optimization problem. A mathematical modelling approach for handling the problem is presented in this section. It generates a number

of duties and computes the time of each duty. The crew scheduling problem in this context is to specify the sequences of trips to be performed by the crews. The goal is to determine a schedule which includes the details of the sequence of trips to be performed by a crew in one duty (shift). The shift is divided into two partial duties with different durations.

The train crew scheduling requires scheduling all trips in a published train timetable into a set of train crew duties such that the crews perform feasible sequence of trips with minimum cost. Railway CSP has the following inputs. A set of crew HDs and a set of RPs, a set of scheduled train trips with fixed starting and ending times, and predetermined driving times between all pair of stations. Based on the descriptions presented in Chapter 3, railway CSP is formulated mathematically as a mixed integer programming (MIP) and this is presented in the following section.

3.5 RAILWAY CREW SCHEDULING MODEL

This section presents a mathematical model for railway crew scheduling. The optimisation model integrates the two phases of pairing generation and pairing optimisation by simultaneously sequencing trips into feasible duties and minimising total elapsed time of any duties. The optimisation model incorporates commonly encountered real-life railway crew scheduling constraints, particularly the inclusion of the time interval of relief opportunities. Existing models usually only consider relieving crew at the beginning of the interval of relief opportunities, which may be impractical. Allowing the train crew to be relieved at any relief point within the interval of relief opportunities offers flexibility. This will improve the robustness of the schedule and provide a better representation of real-world conditions. Computational results obtained from randomly generated instances indicate that the optimisation model can produce feasible railway crew schedules within a reasonable computational time.

The following notations are used through the description of the model.

3.5.1 Notations

Indices

$i,\ i'$	train trip
$j,\ j'$	duty
$k,\ k'$	shift
ohd	originate at crew HD
thd	terminate at crew HD
orp	originate at RP
trp	terminate at RP
ots	originate and terminate at any station

Sets

I	set of all trips
I_{ohd}	set of trips that originate at crew HD ($I_{ohd} \subseteq I$)
I_{thd}	set of trips that terminate at crew HD ($I_{thd} \subseteq I$)
I_{orp}	set of trips that originate at RP ($I_{orp} \subseteq I$)
I_{trp}	set of trips that terminate at RP ($I_{trp} \subseteq I$)
I_{ots}	set of trips that can be sequential in the same duty ($I_{ots} \subseteq I$)
J	set of duties
J_i	set of duties which can contain trip i ($J_i \subseteq J$)
K	set of shifts
K_j	set of shifts for duty j ($K_j \subseteq K$)

Parameters

t_{ijk}	driving time of trip i in duty j of shift k
$\zeta_{ii'jk}$	transition time from trip i to trip i' in the j^{th} duty of shift k
$\zeta_{ii'j(j+1)k}$	transition time from trip i of the 1st duty to the trip i' of the 2nd duty of shift k
$\zeta_{ii'k(k+1)}$	transition time from trip i of shift k to trip i' of the next shift
α_{jk}	minimum duration of 1st part of a duty in shift k
α'_{jk}	maximum duration of 1st part of a duty in shift k
$\delta'_{(j+1)k}$	minimum duration of 2nd part of a duty in shift k
$\delta_{(j+1)k}$	maximum duration of 2nd part of a duty in shift k
dt_i	departure time of trip i

at_i	arrival time of trip i
ds_i	departure station of trip i
as_i	arrival station of trip i
Wt_{max}	normal working time per shift
Wt_{min}	minimum working time allowed per shift
Wt_k	actual driving time in shift k
St_k	spread time of shift k
St_{max}	maximum spread time allowed per shift

Variables

v_{ijk}	$\in \{0,1\}$	binary variable for assignment of trip i in duty j of shift k
w_{ijk}	$\in \{0,1\}$	binary variable for assignment of i as the first trip in duty j of shift k
$x_{ii'jk}$	$\in \{0,1\}$	binary variable denotes that the assignment of i is followed by i' in duty j of shift k
y_{ijk}	$\in \{0,1\}$	binary variable denotes that the assignment of i as the last trip in duty j of shift k
$z_{ii'j(j+1)k}$	$\in \{0,1\}$	binary transition variable denotes that the assignment of i at the end of a partial duty j to be followed by i' at the beginning of the subsequent partial duty of shift k
$z_{ii'k(k+1)}$	$\in \{0,1\}$	binary transition variable denotes that the assignment of i at the end of the duty (shift) to be followed by i' at the beginning of the subsequent duty (shift)
U	$\in \{0,1\}$	binary variable
σ_{ijk}	$\in \mathbb{R}$	starting time of trip i in duty j of shift k
ϑ_{ijk}	$\in \mathbb{R}$	completion time of trip i in duty j of shift k

3.5.2 Objective Function

The objective function is designed to minimise the total number of duties by minimising idle transition times. The idle transition times includes the idle intervals between trips and an idle transition during a MB. The function consists of driving period and non-driving period.

$$\text{Min} \left(\sum_{j \in J_i} \sum_{i \in I_k} t_{ijk} v_{ijk} + \sum_{j \in J_i} \sum_{i,i' \in I_k} \zeta_{ii'jk} x_{ii'jk} + \sum_{i,i' \in I_k} \zeta_{ii'j(j+1)k} z_{ii'j(j+1)k} \right) \quad (1)$$

$$\forall\ i, i' \in I, i \neq i',\ j \in J_i,,\ k \in K_j$$

3.5.3 Constaints

Equation (2) is the trip assignment. It enforces every trip *i* to be allocated in exactly one duty *j* of shift *k*.

$$\sum_{k \in K_j} \sum_{j \in J_i} v_{ijk} = 1 \qquad \forall\ i \in I \quad (2)$$

This equation implies that no deadheading is allowed. A crew has to wait for the next trip at a RP and the changeover of trains is at the same RP. When the assignment of trip *i* is followed by trip i' in the same duty, a sequence of the trips is enforced via constraint (3). Trips *i* and trip i' are consecutive only in the case that the binary variable $x_{ii'jk} = 1$. Similarly, constraint (4) denotes that the assignment of trip *i* in duty *j* is followed by trip i' at the next duty j'. The transition variable $z_{ii'j(j+1)k}$ is activated when both v_{ijk} and $v_{i'(j+1)k}$ are equal to one. As a result, one transition from trip *i* to trip i' occurs at the end of any partial duty if and only if trip i' is assigned in the subsequent partial duty.

$$x_{ii'jk} \geq v_{ijk} + v_{i'jk} - 1 \qquad \forall\ i, i' \in I_k, i \neq i',\ j \in J,\ k \in K_j \quad (3)$$

$$z_{ii'j(j+1)k} \geq v_{ijk} + v_{i'(j+1)k} - 1 \qquad \forall\ i, i' \in I_k, i \neq i',\ j \in J,\ k \in K_j \quad (4)$$

Constraint (5a) ensures that no overlap is allowed. The start time of trip i' in any duty require the completion of the previous trip. Constraint (5b) and constraint (5c) are included to ensure a connectivity of the trip sequences.

$$at_{ijk} + \zeta_{ii'jk}\, x_{ii'jk} \leq dt_{i'jk} \qquad \forall\ i, i' \in I_k,\ i \neq i',\ j \in J,\ k \in K_j \qquad (5a)$$

$$as_{ijk}\, v_{ijk} = ds_{i'jk}\, v_{ijk} \qquad \forall\ i, i' \in I_k,\ i \neq i',\ j \in J,\ k \in K_j \qquad (5b)$$

$$as_{ijk}\, v_{ijk} = ds_{i'(j+1)k}\, w_{ijk} \qquad \forall\ i, i' \in I_k,\ i \neq i',\ j \in J,\ k \in K_j \qquad (5c)$$

Constraint (6a) and constraint (6b) denote the relation between the start and completion times in a duty. The completion time of the last trip in a duty is greater than or equal to the start time of the first trip plus the total driving time and the total transition time in the duty.

$$at_{ijk} \geq dt_{ijk} + \sum_{i \in I_k} t_{ijk}\, v_{ijk} \qquad \forall\ j \in J,\ k \in K_j \qquad (6a)$$

$$\vartheta_{ijk} \geq \sigma_{ijk} + \sum_{i \in I_k} t_{ijk}\, v_{ijk} + \sum_{i,i' \in I_k} \zeta_{ii'jk}\, x_{ii'jk} \qquad \forall\ j \in J,\ k \in K_j \qquad (6b)$$

Constraint (7a), along with constraint (7b), constraint (8a), and constraint (8b), indicate that the total continuous driving time in the 1st part of a duty should be greater than or equal to the minimum allowable duration of the 1st part of a duty in shift k (α_{jk}) and the total continuous driving time of the 2nd part of a duty should be less than or equal to the maximum duration of the 2nd part of a duty in shift k ($\delta_{(j+1)k}$). Otherwise, the total continuous driving time of the 1st part of a duty should be less than or equal to the maximum duration of the 1st part of a duty in shift k (α'_{jk}) and the total continuous driving time of the 2nd part of a duty should be greater than or equal to the minimum duration of the 2nd part of a duty in shift k ($\delta'_{(j+1)k}$). The set of constraints satisfy a condition in which a train crew takes a MB at the earliest or latest times or in any time between the two limits (Figure 3.6).

$$\sum_{i \in I_k} t_{ijk}\, v_{ijk} + \sum_{i,i' \in I_k} \zeta_{ii'jk}\, x_{ii'jk} \geq \alpha_{jk}\, U \qquad \forall\ j \in J,\ k \in K_j \qquad (7a)$$

$$\sum_{i \in I_k} t_{ijk}\, v_{ijk} + \sum_{i,i' \in I_k} \zeta_{ii'jk}\, x_{ii'jk} \leq \alpha'_{jk}\,(1 - U) \qquad \forall\ j \in J,\ k \in K_j \qquad (7b)$$

$$\sum_{i \in I_k} t_{i(j+1)k}\, v_{i(j+1)k} + \sum_{i,i' \in I_k} \zeta_{ii'(j+1)k}\, x_{ii'(j+1)k} \leq \delta_{(j+1)k}\, U \qquad \forall\ j \in J,\ k \in K_j \qquad (8a)$$

$$\sum_{i \in I_k} t_{i(j+1)k}\, v_{i(j+1)k} + \sum_{i,i' \in I_k} \zeta_{ii'(j+1)k}\, x_{ii'(j+1)k} \geq \delta'_{(j+1)k}\,(1 - U) \qquad \forall\ j \in J,\ k \in K_j \qquad (8b)$$

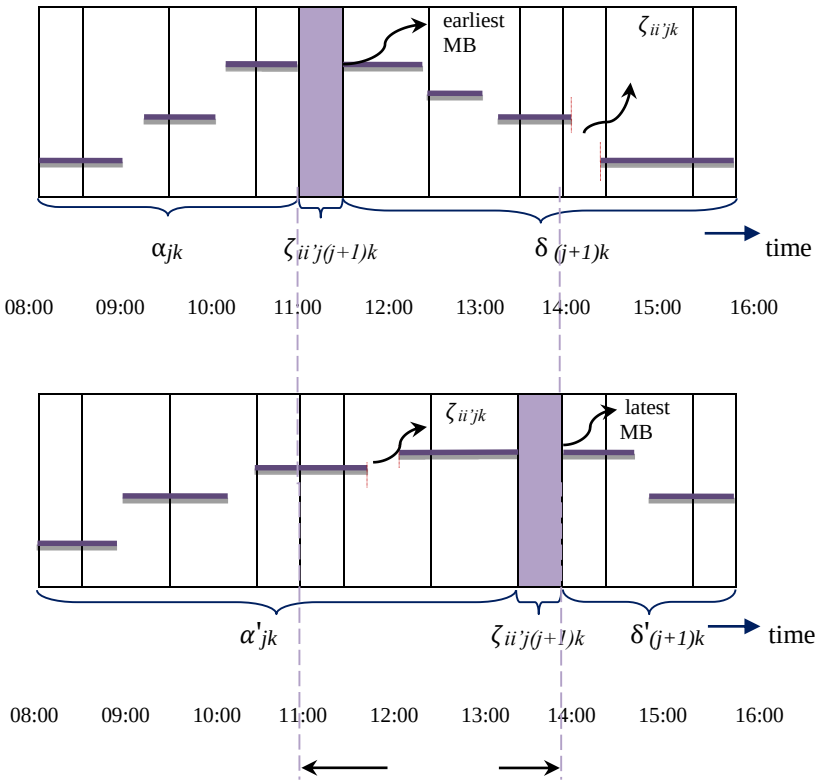

Figure 3.6 Two combinations of partial duties in the shift.

Equation (9a) calculates the total actual driving time in shift k (Wt_k), which is equal to the total working time of all partial duties in the shift. Constraint (9b) states that the total actual driving time within the shift must not exceed the upper bound (Wt_{max}) and the lower bound (Wt_{min}).

$$\sum_{j \in J_i} \sum_{i \in I_k} t_{ijk}\, v_{ijk} \;+\; \sum_{j \in J_i} \sum_{i,i' \in I_k} \zeta_{ii'jk}\; x_{ii'jk} \;\leq\; Wt_k \qquad \forall\; j \in J,\; k \in K_j \qquad (9a)$$

$$Wt_{min} \;\leq\; Wt_k \;\leq\; Wt_{max} \qquad (9b)$$

Constraint (10a) restricts the spread time of a shift from exceeding the maximum allowed total spread time. Spread time of a shift (St_k) is equal to the total working time plus the transition time between each partial duty (MB). The relation between the last trip i in partial duty j of shift k and the start of trip $i + 1$ in the next partial duty $j + 1$ of shift k is given by equation (10b).

$$\sum_{j \in J_i} \sum_{i \in I_k} t_{ijk}\, v_{ijk} \;+\; \sum_{j \in J_i} \sum_{i,i' \in I_k} \zeta_{ii'jk}\; x_{ii'jk} \;+\; \sum_{i,i' \in I_k} \zeta_{ii'j(j+1)k}\; z_{ii'j(j+1)k} \;\leq\; St_{max} \qquad (10a)$$
$$\forall\; j \in J, k \in K_j$$

$$\sum \zeta_{ii'jk}\, x_{ii'jk} \;+\; \sum_{i \in I_k} t_{ijk}\, v_{ijk} \;+\; \sum \zeta_{ii'(j+1)k}\; x_{ii'(j+1)k} \;=\; \sum \zeta_{i(i+1)(j+1)k}\; x_{i(i+1)(j+1)k}$$
$$+ \sum_{i \in I_k} t_{i(j+1)k}\, v_{i(j+1)k} \;+\; \sum \zeta_{i(i+1)jk}\, x_{i(i+1)jk} \qquad (10b)$$
$$\forall\; i, i' \in I_k, i \neq i', j \in J,\; k \in K_j$$

Considering that each duty consists of at least one trip, only one trip can be the first or the last one in each duty. Equation (11) expresses the requirement that the first trip in the 1st part of a duty which is also the first trip of the corresponding duty (shift) should originate from a HD. Equation (12) states that the last trip in a duty should terminate at a HD or at a RP. Equation (13) ensures that each trip, except the first trip, is assigned after another trip. Similarly, equation (14) ensures that each trip, except the last trip, is assigned before another trip. Equation (15) expresses that for each trip which terminated the 1st part of a duty, there is a transition time (MB) from this trip to the first trip in the subsequent part of a duty. Similarly, equation (16) expresses that for each trip which originated a duty, there is a transition time from the last trip of the previous duty to the current duty. Equation (17) ensures that for each trip which terminated a duty (shift), there is a transition time from this trip to the first trip of the next duty (sign off to sign on). Similarly, equation (18) ensures that for each trip which originated a duty (shift), there is a transition time from the last trip of the previous duty

(shift) to the first trip of the current duty (shift). Constraints (19) and (20) are the variable restrictions for 7 binary variables and 2 real variables, respectively.

$$\sum_{i \in I_{ohd}} w_{ijk} = 1 \qquad \forall\ j \in J,\ k \in K_j \tag{11}$$

$$\sum_{i \in I_{thd} \cup I_{trp}} y_{ijk} = 1 \qquad \forall\ j \in J,\ k \in K_j \tag{12}$$

$$\sum_{i \in I_{ots},\, i \neq i'} x_{ii'jk} = v_{i'jk} - w_{ijk} \qquad \forall\ i' \in I_{ots},\ j \in J, k \in K_j \tag{13}$$

$$\sum_{i' \in I_{ots},\, i' \neq i} x_{ii'jk} = v_{ijk} - y_{ijk} \qquad \forall\ i \in I_{ots},\ j \in J, k \in K_j \tag{14}$$

$$\sum_{i \in I_{thd} \cup I_{trp}} z_{ii'j(j+1)k} = w_{i'(j+1)k} \qquad \forall\ i' \in I_{ohd} \cup I_{orp},\ j \in J, k \in K_j \tag{15}$$

$$\sum_{i' \in I_{ohd} \cup I_{orp}} z_{ii'j(j+1)k} = y_{ijk} \qquad \forall\ i \in I_{thd} \cup I_{trp},\ j \in J, k \in K_j \tag{16}$$

$$\sum_{i \in I_{thd} \cup I_{trp}} z_{ii'k(k+1)} = w_{i'j(k+1)} \qquad \forall\ i' \in I_{ohd},\ j \in J, k \in K_j \tag{17}$$

$$\sum_{i' \in I_{ohd} \cup I_{orp}} z_{ii'k(k+1)} = y_{ijk} \qquad \forall\ i \in I_{thd} \cup I_{trp},\ j \in J,\ k \in K_j \tag{18}$$

$$\begin{aligned} &v_{ijk} \in \{0,1\},\ w_{ijk} \in \{0,1\},\\ &x_{ii'jk} \in \{0,1\},\ y_{ijk} \in \{0,1\},\\ &z_{ii'j(j+1)k} \in \{0,1\},\\ &z_{ii'k(k+1)} \in \{0,1\},\\ &U \in \{0,1\} \end{aligned} \tag{19}$$

$$\begin{aligned} &\sigma_{ijk} \in \mathbb{R},\\ &\vartheta_{ijk} \in \mathbb{R} \end{aligned} \tag{20}$$

Generally, these constraints can be divided into four groups. The first group focuses on the scheduling and sequencing trips (Constraints 2 – 6); the second group addresses the duty restrictions (Constraints 7 – 10); the third group determines the assignment and sequencing of trips in a duty (Constraints 11 – 18), and the remaining group restricts the value of variables (Constraints 19 – 20).

3.6 MP SOLUTION TECHNIQUE

Real-life optimisation applications involve various complex constraints that are difficult to find satisfactory solutions. An optimisation approach works by exploring the search space to find the optimum solution according to an objective function while satisfying given constraints. It determines a value of cost to minimise or of profit to maximise. The exact solution approaches refer to the methods which can obtain optimal solution and prove its optimality. Some well-known exact methods for solving mathematical formulations are branch-and-bound, branch-and-cut, branch-and-price, and dynamic programming. MIP solvers, such as Xpress-Optimization (FICO) and CPLEX (ILOG, Inc.), employ branch-and-cut which is a combination of branch-and-bound and cutting-plane techniques.

Branch-and-bound is a classic method for solving the IP. In branch-and-bound, the problem is decomposed recursively into a disjunction of smaller sub-problems by a tree search. This decomposition creates further branch nodes and stop decomposing when the node is either pruned (reach feasibility or optimality) or a leaf node is reached (have assigned value to all variables). Thus, these smaller subsets were evaluated until best solution is found.

The first node of the branch-and-bound search tree can contains the relaxed linear programming solution and has two designated bound, an upper bound (UB) and a lower bound (LB). The optimal integer solution will be between these two bounds. An optimal integer solution is reached when a feasible integer solution is achieved at a node that has an upper bound greater than or equal to the upper bound at any other ending node.

The branch and bound method can be used for mixed integer problems, except only variables with integer restrictions are rounded down to achieve the initial lower bound and only integer variables are branched on. When determining which variable to branch from, the greatest fractional part is selected from among only those variables that must be integer. The optimal solution is reached when a feasible solution is generated at a node that has integer values for those variables requiring integers and that has reached the maximum upper bound of all ending nodes (Taylor, 2009). MIPs apply relaxation methods. Relaxation of a MIP is a strategy used such that (a) any solution to the MIP corresponds to a feasible solution to the relaxed problem, and (b)

each solution to the MIP has an objective function value greater than or equal to that of the corresponding solution to the relaxed problem. MIP techniques examine a subset of possible solutions and do not explicitly examine every possible combination of discrete solutions and use optimisation theory to prove that no other solution can be better than the best one found. This type of technique is known as implicit enumeration (Smith and Taskın, 2007).

Optimal solutions can be obtained for CSP by solving the model using standard optimisation programming languages such as Xpress-Optimizer (FICO) and CPLEX Optimization Studio (IBM ILOG). Xpress-Optimization uses branch-and-bound algorithm to solve MIP. This was used to obtain the solutions and includes classes of cutting-planes which are generated during optimisation. Using the Xpress-Optimization solver, the results indicate that it is difficult to solve a practical optimisation problem using pure MIP method particularly for large-sized instances. This is because the practical optimisation problem involves a large number of variables and constraints. The standard branch and bound technique employed by the Xpress-Optimization begins by solving the linear programming relaxation which is obtained by removing some restrictions in the mixed integer program. The number of feasible duties will increase with the number of trips included in the problem as well as the increase of runtimes. The overall computational results of the mathematical model by Xpress-Optimization are given in Appendix A of this thesis.

3.7 CONCLUSION

Railway CSP represents a computationally difficult problem because of the size of the instances and the complex structure of operational constraints. In this chapter, an alternative mathematical model for railway CSP has been presented. The optimisation model integrates the two phases of pairing generation and pairing optimisation by simultaneously sequencing trips into feasible duties and minimising total elapsed time of any duty. Crew scheduling constraints in which the crews have to return to their home depot at the end of the shift are included in the model. The flexibility of this model comes in the inclusion of the time interval of relief opportunities, allowing the crew to be relieved during the ROP. The MIP model involves binary variables which determine whether a trip is assigned in a duty or not, whether it is the first trip, followed by the next trip or it is the last trip. The number of these variables will increase significantly with the number of trips included in the

problem. Optimisation Programming Language (ILOG OPL Studio) software was used to obtain the solutions. Modelling with ILOG OPL Studio was done by firstly declaring of data and variables. Then was followed by defining the objective function and constraints. The overall results suggest that improvement should be made in terms of search strategy and the time consumed by the methods to obtain final results. From a practical viewpoint, the proposed model and its solution technique can be integrated with other search techniques to find better solutions.

Chapter 4: **Constraint Programming**

This chapter describes Constraint Programming (CP) with the solution techniques applied to solve the problem. CP is a structured technique with natural way in expressing optimisation problems by means of variables and constraints through the application of constraint propagation mechanism based on the tree search structure for the solving process.

4.1 CONSTRAINT PROGRAMMING (CP)

CP converged from different areas of optimisation, computer science and artificial intelligence (AI) (Hooker, 2002). It incorporates both a modelling and a problem solving paradigm. CP is a structured computer programming technique that provides a natural way of expressing a wide variety of algorithmic problems.

CP is an effective technique for solving large combinatorial optimisation problems and has proven a success in various application areas. The strength of CP is due to the clear separation between model and solver. The problem is stated declaratively in terms of variables and constraints and modelling the relations of the entities in the problem is engaged in. This model is then passed to a constraint solver, which will return a solution to the problem.

Several operations research (OR) techniques have been applied to tackle the crew planning and scheduling problems. The OR applications incorporate various kinds of enumeration strategies that are usually embedded in complex computer programs. Barták et al. (2010) noted that constraint satisfaction offers a very good framework for integrating OR techniques in more general AI solving algorithms. The primary technology for this integration is based on the concept of *global constraints*. Global constraints accommodate efficient algorithms to solve well defined sub-problems while they still can be combined with other constraints for modelling the side features of the problem. Global constraints coupled with sophisticated search techniques are the main strength behind the success of constraint-based scheduling. Rodosek et al. (1999) have introduced a hybrid algorithm to reduce the solution space by integrating constraint logic programming and mixed integer

programming using both the local constraint propagation and the global constraint propagation. Milano and Wallace (2010) presented an important survey on how CP can be applied to exploit linear programming within various hybrid algorithms and can enhance Lagrangian relaxation, Benders decomposition and column generation techniques. Gualandi and Malucelli (2013) reviewed the applications of the CP-based column generation framework to solve several complex real-life optimisation problems.

Railway crew scheduling represents a very complex problem due to the presence of conflicting constraints that have to be satisfied and the huge search space that has to be explored. The straightforward approach is used to be applied to this problem is *generate and test* approach. Thus a complete roundtrip is generated and then tested its feasibility. CP in contrasts applies a different method of computation, which is the *constraint and generates* one. When a solution is found, its objective value is stored. A new constraint is subsequently added to the problem while imposing the value of the objective function to be better than the best previously found.

A combinatorial optimisation problem in CP is modelled as a set of variables, the objects the problem deals with, and a set of constraints representing the relationships among the objects. A CP system implements these variables and constraints and provides a solution procedure of assigning variables subject to all the constraints. Thus the solution to CP is an assignment to each variable a value from its domain such that all the problem constraints are satisfied. The objective is to find the minimum solution with complete assignment of values to the variables.

Even though some work has been done on the CSP using a wide variety of solution techniques, the problem is still hard to solve. CP is one of the techniques that has drawn increased attention in recent years and has been successfully applied to the scheduling problems (Lustig and Puget, 2001). Sellmann et al. (2002) applied CP based column generation framework and CP based heuristic tree search. The objective of their study is to assign lines of work to a set of crew members and minimise the cost of that assignment. The researchers have shown how CP can be incorporated to overcome typical deficiencies of OR approach. Silva (2001) combined CP and linear programming (LP) for solving bus driver scheduling problem. This study found that CP can handle complex real life constraints easily and constraint propagation can improve the efficiency of generating duties.

Despite the fact that CP is an emergent and promising approach for solving large combinatorial optimisation problems, very little research has applied this method to the CSP. This study presents a CP formulation and solution methodology to solve the railway CSP. The CP formulation incorporates commonly encountered real-life railway crew scheduling constraints. A computational experiment was carried out on randomly generated problem instances which are based on the data from Queensland Rail (QR), Australia.

The approach to solve the railway CSP follows several stages. Firstly, the train segment is divided into trips for each depot. A duty (roundtrip, pairing) is defined as a sequence of trips which can be assigned to a crew that starts and ends at the same crew depot. The overall modelling approach can be seen in Figure 4.1. The optimisation model sequences the trips into feasible duties and minimises the total elapsed time of the duties (shifts). A number of feasible duties will be generated for a depot. The best subset will then be selected from all the generated duties based on the minimum cost of duties.

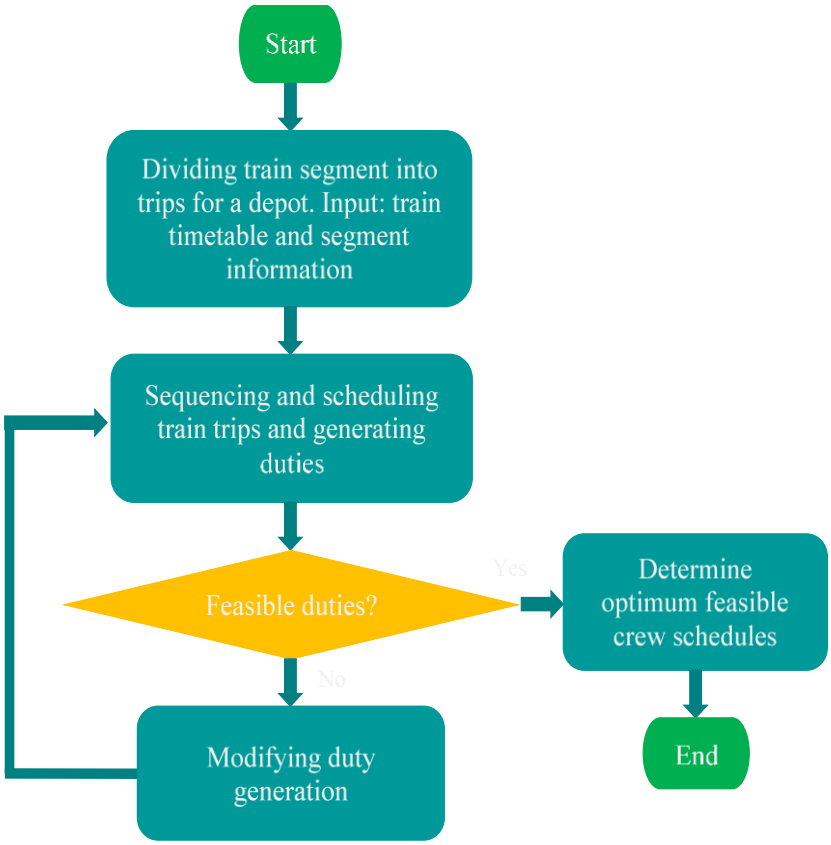

Figure 4.1 Modelling approach.

4.1.1 Constraint Propagation and Search

The process of inference is called constraint propagation. Constraint propagation is the main inference method in constraint programming systems. It is an efficient inference mechanism using concurrently working propagators that accumulates information in a constraint group. It infers that certain values cannot be part of certain variable domains anymore if they violate some constraint. The entities that perform constraint propagation are called propagators. A propagator removes values from variable domains that cannot be part of any solution of its constraint. Constraint distribution divides the problem into corresponding cases when constraint propagation cannot proceed any further. By iterating propagation and distribution, propagation will ultimately resolve a solution to the problem.

Many algorithms for solving CP systematically search through the possible assignments of values to variables. Such algorithms are guaranteed to find a solution, if one exists, or to prove that the problem is insolvable. Therefore, the systematic search algorithms are complete. The main disadvantage of these algorithms is that they take a very long time to perform it. There are two main classes of systematic search algorithms. The first one is the algorithms that search the space of complete assignments, i.e., the assignments of all variables, till they find the complete assignment that satisfies all the constraints, and the second one is the algorithms that extend a partial consistent assignment to a complete assignment that satisfies all the constraints.

4.1.2 Tree Structure

Trees are a type of nonlinear structure where the data are organised such that items of information are related by branches. Trees can describe the relationship among their elements or objects and represent them naturally. Relationships such as one to one and one to many are among those that can always be found in real life, and can be easily described by trees. The ability of trees to organise data and represent them naturally is useful in solving a wide variety of algorithmic problems.

Horowitz and Sahni (1987) define a tree as a finite set of nodes in which there is a specially designated node called the root, and the remaining nodes are partitioned into $n \geq 0$ disjoint sets $T_1,\ldots..,T_n$ where $T_1,\ldots..,T_n$ are called the subtrees of the root. The node is the item of information contained in the tree. A sample tree is given in Figure 4.2. This tree has 13 nodes, each item of data being a single letter for simplicity.

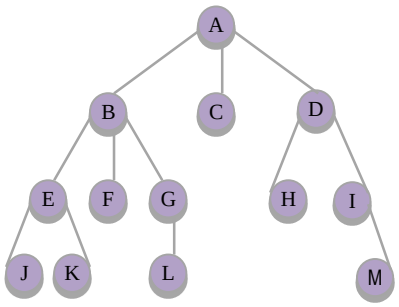

Figure 4.2 A sample tree with 13 nodes.

The number of subtrees of a node is called its degree. The degree of A is 3, of C is 0, of D is 2, etc. Nodes that have degree zero are called terminal nodes or external nodes, i.e. C, F, H, J, K, L, M. The other nodes are referred to as non-terminal nodes or internal nodes. The degree of a tree is the maximum degree of a node in the tree. The tree in Figure 4.2 has degree 3.

The relationship between a node and its successors is described as a parent – child relationship. The predecessor of a node X is said to be the node X's parent and a successor of a node X is said to be the node X's child. Thus, considering the tree in Figure 4.2, the children of B are E, F, and G and the parent of B is A. Every node that is not a terminal node has at least one child. The nodes of the same parent are said to be siblings.

Each node in a tree is at a certain level in that tree. The root node is at level one, and if a node is at level l, then its children are at level l + 1. Figure 4.3 shows the levels of nodes in the tree. The height or depth of a tree is defined as the maximum level associated with any of its nodes. Figure 4.3 contains 10 nodes and has a height of 4.

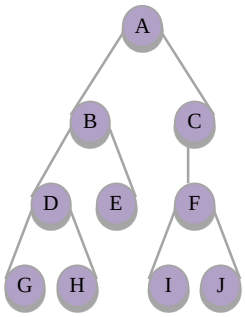

Figure 4.3 A sample tree with 4 levels.

A binary tree is characterized by the fact that each node can have at most two branches, each sub-tree is identified as being either the left or right sub-tree of its parent, and a binary tree may be empty (Stubbs and Webre, 1993). Horowitz and Sahni (1987) define a binary tree as a finite set of nodes which is either empty or consists of a root and two disjoint binary trees called the left sub-tree and the right sub-tree. Figure 4.4 shows two samples of binary trees with 9 nodes and a height of 4.

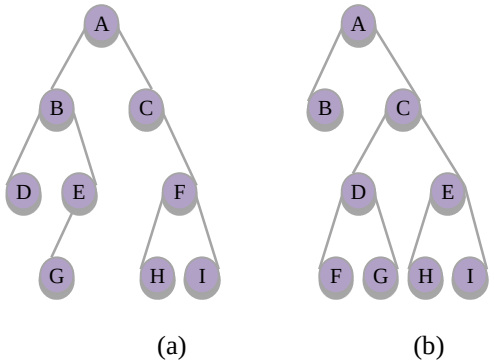

Figure 4.4 Binary trees.
(a). A binary tree with 9 nodes
(b). A complete binary tree with 9 nodes

A binary tree is drawn with the root at the top and with the left and right children always positioned to the left and right of their parent, respectively. An external node has no children and an internal node has at most two children. If every internal node in a binary tree has nonempty left and right sub-trees, the tree is termed a complete binary tree (Figure 4.4 (b)). A complete binary tree of depth d is a binary tree having $2^d - 1$ nodes. For example, a complete binary tree of depth 4 as shown in Figure 4.5

has a number of $2^4 - 1 = 15$ nodes, which is the maximum number of nodes this binary tree can have.

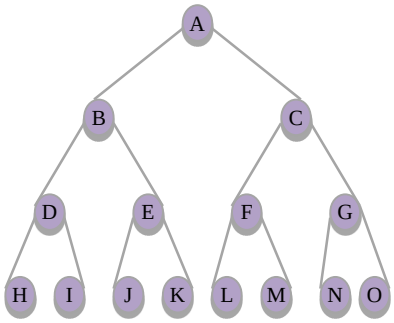

Figure 4.5 A complete binary tree of depth 4.

In a complete binary tree, each node is either an internal node with two nonempty left and right subtrees, or an external node having no children nodes. In such a binary tree, the number of external nodes always exceeds the number of internal nodes by one. This relationship can be termed as $n_1 = n_0 + 1$ where n_1 is the number of external nodes and n_0 is the number of internal nodes (Horowitz and Sahni, 1987).

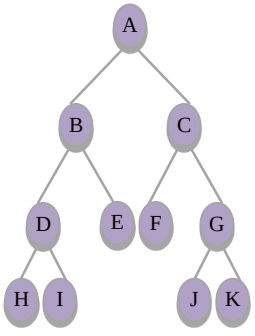

Figure 4.6 The sequence in which the node of the tree are visited for preorder (ABDHIECFGJK), inorder (HDIBEAFCJGK) and postorder (HIDEBFJKGCA).

Postorder traversal is commonly referred to as Polish expression and has been widely used as representation of the slicing tree for many algorithmic problems (Figure 4.6). A tree search starts at the root and explores nodes from this point, searching for a goal node that satisfies certain conditions. For some problems, any goal node is acceptable but for other problems, a minimum-depth goal node is required or a goal node nearest to the root.

Depth-first search (DFS) is an algorithm for traversing or searching a tree, tree structure, or graph. It starts at the root and explores a path all the way to a branch before **backtracking** and exploring another path. DFS is an uninformed search that progresses by expanding the first child node of the search tree that appears and thus going further down until a goal node is found, or until it reaches a node that has no children. Then the search backtracks, returning to the most recent node it hasn't finished exploring. DFS is mainly used to find any solution when cost is not an issue.

DFS follows the following rules:

1. Select an unvisited node x, visit it, and treat as the current node.
2. Find an unvisited neighbor of the current node, visit it, and make it the new current node.
3. If the current node has no unvisited neighbors, backtrack to its parent, and make that parent the new current node.
4. Repeat steps 3 and 4 until no more nodes can be visited.
5. If there are still unvisited nodes, repeat from step 1.

DFS is not good for tall trees when it is possible to over commit to a bad path early. It may miss a complete solution because it focused on checking the first partial path and did not test the others in the queue.

Breadth-first search (BFS) is one of the simplest algorithms for searching a graph. It explores nodes nearest the root before exploring nodes further away. BFS is an uninformed search method that aims to expand and examine all nodes or combination of sequences by systematically searching through every solution. It exhaustively searches the entire graph or sequence without considering the goal until it finds it. BFS is mainly used to find a solution at minimum distance from the root of a search tree.

BFS follows the following rules:

1. Select an unvisited node x, visit it, treat as the root in a BFS tree being formed. This level is called the current level.
2. From each node z in the current level, in the order in which the level nodes were visited, visit all the unvisited neighbors of z. The newly visited nodes from this level form a new level that becomes the next current level.
3. Repeat step 2 until no more nodes can be visited.
4. If there are still unvisited nodes, repeat from Step 1.

BFS is not good for fat trees where the nodes have high branching factors. It may also be a bad choice when several partial paths lead to the same node several levels down (bottleneck). It is not always fast but it may never miss a complete solution.

Backtracking is the most common algorithm for performing systematic search. It is a methodical way of trying out various sequences of decisions to find a feasible solution. Backtracking algorithm is a recursive method of building up feasible solutions to a combinatorial optimisation problem one step at a time (Kreher and Stinson, 1999). Backtracking comprises of doing a DFS of the state space tree, checking whether each node is promising and if the node is non-promising backtracking to the parent node.

Backtracking is an algorithm design technique for solving problems in which the number of choices increases at least exponentially with their initial size. This approach makes it possible to solve many large instances of NP-hard problems in an acceptable computational time.

4.2 CONSTRAINT PROGRAMMING MODEL

The CP formulation has the objective of minimising the completion time of the crew duties that consists of continuous driving time, transition time between trips, and transition times between trips of different duties. In this section, model subscripts, sets, parameters and model variables are presented followed by problem constraints and their explanations in detail.

Indices

$i,\ i'$	Index for trip
$j\ \ j'$	Index for duty
$k\ \ k'$	Index for shift

Sets

I	Set of all trips
J	Set of all duties
J_i	Set of duties which can contain trip i ($J_i \subseteq J$)
K	Set of all shifts
K_j	Set of shifts for duty j ($K_j \subseteq K$)

Parameters

α_{jk}	minimum duration of 1st part of a duty in the shift k
α'_{jk}	maximum duration of 1st part of a duty in shift k
$\delta'_{(j+1)k}$	minimum duration of 2nd part of a duty in shift k
$\delta_{(j+1)k}$	maximum duration of 2nd part of a duty in shift k
$\zeta_{ii'jk}$	transition time of trip i in the same partial duty j
$\zeta_{ii'j(j+1)k}$	transition time of trip i of different partial duties j
dt_i	departure time of trip i
at_i	arrival time of trip i
ds_i	departure station of trip i
as_i	arrival station of trip i
t_i	traveling time of trip i
Tr_{max}	maximum transition time in the same partial duty
To_{max}	maximum transition time between partial duties (ROP)
Tr_{min}	minimum transition time in the same partial duty
To_{min}	minimum transition time between partial duties (ROP)
Wd_{max}	maximum continuous driving time allowed per duty
Wt_{max}	normal working time per shift
Wt_{min}	minimum working time allowed per shift
St_k	spread time of shift k
St_{max}	maximum spread time allowed per shift

Variables

x_{ijk} $\in\{0,1\}$ binary variable for the assignment of trip i to duty j of shift k

$y_{ii'jk}$ $\in\{0,1\}$ binary variable that denotes whether a transition time occurs from trip i to trip i' in duty j of shift k

$z_{ii'j(j+1)k}$ $\in\{0,1\}$ binary variable that denotes whether a transition time appears between duties in shift k

σ_{ijk} $\in \mathbb{R}$ departure time of trip i in duty j of shift k

Objective function

The objective is to minimise the total working time of the crew ($C_{\max}$).

$$\text{Min } C_{\max} \qquad (1)$$

The maximum completion time should be greater than or equal to the arrival time of all the last trips of all duties. Thus, completion of all traveling tasks is given by expression (2).

$$C_{\max} \geq \sigma_{ijk} + \sum_{j \in J_i} \sum_{i \in I_k} t_i x_{ijk} + \sum_{j \in J_i} \sum_{i,i' \in I_k} \zeta_{ii'jk} y_{ii'jk} + \sum_{i,i' \in I_k} \zeta_{ii'j(j+1)k} z_{ii'j(j+1)k} \quad \forall i, i' \in I, i \neq i', j \in J_i,, k \in K_j \qquad (2)$$

Both discrete variable and continuous variable are used in this model. There are three discrete decision variables that are; the decision about whether to assign trip i in duty j of shift k (x_{ijk}); the decision about whether a transition time occurs in the duty ($y_{ii'jk}$); and the decision about whether a transition time between duties appears in the shift ($z_{ii'j(j+1)k}$). The continuous variable is the departure time of trip i in duty j of shift k (σ_{ijk}).

The decision variable trip i (x_{ijk}) is declared as an activity. An activity in the schedule is considered as a non-pre-emptive in which the activity cannot be interrupted. Each trip must be serviced without interruption from its departure time to its arrival time. The amount of time to be used by all the trips is unknown value in a given time interval. This means that the time capacity required by all the trips in a duty can vary over time, provided that the duty duration is not exceeded. In this crew

scheduling problem, the complexity increase as the duty duration is not constant over time but ranging between two predefined limits.

Various related constraints for scheduling and sequencing trips to complete a duty are included in the model. When an activity is declared, the equation (3) is automatically included in the system.

$$dt_i + t_i = at_i \qquad \forall\ i \in I \qquad (3)$$

A chain of activities can be enforced by using the following precedence constraint.

$$\begin{aligned} & at_i + \zeta_{ii'jk} \leq dt_{i'} \\ \Longrightarrow\ & \zeta_{ii'jk} = dt_{i'} - at_i \qquad \forall\ i, i' \in I, i \neq i',\ j \in J_i,\ k \in K_j \end{aligned} \qquad (4)$$

When trips are allocated on the same duty j, the sequence of these trips can be enforced by using the following expression (5). This expression implies that when trip i and trip i' are allocated in duty j, transition time occurs either from trip i to trip i' or from trip i' to trip i.

$$\bigvee_{j \in J_i} x_{ijk} \ \wedge \ \bigvee_{j \in J_i} x_{i'jk} \ \Longrightarrow\ y_{ii'jk} \vee y_{i'ijk} \qquad \forall\ i, i' \in I, i \neq i', j \in J_i, k \in K_j \qquad (5)$$

The duties can be regarded as unary resources in which two trips assigned in the same duty cannot overlap in time. The set J_i represents the set of duties that can contain trip i. The trip i can be either starts before trip i' or starts after trip i'. All trips that will be in the same duty are related by constraint (6) and constraint (7).

$$\left(\sigma_{ijk} + t_i\, x_{ijk} + \zeta_{ii'jk}\, y_{ii'jk} \leq \sigma_{i'jk} \right) \qquad \forall\ i, i' \in I, i \neq i', j \in J_i, k \in K_j \qquad (6)$$

$$\vee$$

$$\left(\sigma_{i'jk} + t_{i'}\, x_{i'jk} + \zeta_{ii'jk}\, y_{ii'jk} \leq \sigma_{ijk} \right) \qquad \forall\ i, i' \in I, i \neq i', j \in J_i, k \in K_j \qquad (7)$$

The start time of trip i' in any duty requires the completion of the previous trip i. Similarly, the start time of the 2nd part of a duty at every shift k requires the completion of the 1st duty. This can be stated by the following expression (8) and expression (9).

$$\left(\sigma_{ijk} + \sum_{i \in I_k} t_i \; x_{ijk} + \sum_{i,i' \in I_k} \zeta_{ii'jk} \; y_{ii'jk} + \zeta_{ii'j(j+1)k} \; z_{ii'j(j+1)k} \leq \sigma_{i'jk} \right)$$

$$\forall \; i, i' \in I, i \neq i', j \in J_i, k \in K_j \qquad (8)$$

$$\vee$$

$$\left(\sigma_{i'jk} + \sum_{i' \in I_k} t_{i'} \; x_{i'jk} + \sum_{i,i' \in I_k} \zeta_{ii'jk} \; y_{ii'jk} + \zeta_{ii'j(j+1)k} \; z_{ii'j(j+1)k} \leq \sigma_{ijk} \right)$$

$$\forall \; i, i' \in I, i \neq i', j \in J_i, k \in K_j \qquad (9)$$

Constraint (10) is included to ensure connectivity of the trip sequence in a duty. Whereas constraint (11) and constraint (12) ensure that both the transition time between trips in the sequence of the same duty and the transition time between trips of different duties do not exceed the maximum and minimum allowed transition times. Constraint (13) ensures that the allowed maximum continuous driving time is not violated in each duty.

$$\left(\sigma_{ijk} + t_i \; x_{ijk} \leq \sigma_{i'jk} \right) \wedge \left(ds_i = as_i \right)$$

$$\forall \; i, i' \in I, i \neq i', j \in J_i, k \in K_j \qquad (10)$$

$$\left(\sum_{i,i' \in I_k} \zeta_{ii'jk} \; y_{ii'jk} \leq Tr_{max} \right) \wedge \left(\sum_{i,i' \in I_k} \zeta_{ii'jk} \; y_{ii'jk} \geq Tr_{min} \right)$$

$$\forall \; i, i' \in I, i \neq i', j \in J_i, k \in K_j \qquad (11)$$

$$\left(\sum_{i,i' \in I_k} \zeta_{ii'j(j+1)k} \; z_{ii'j(j+1)k} \leq To_{max} \right) \wedge \left(\sum_{i,i' \in I_k} \zeta_{ii'j(j+1)k} \; z_{ii'j(j+1)k} \geq To_{min} \right)$$

$$\forall \; i, i' \in I, i \neq i', j \in J_i, k \in K_j \qquad (12)$$

$$\left(\sum_{i \in I_k} t_i \; x_{ijk} \leq Wd_{max} \right) \wedge \left(\sum_{i \in I_k} t_i \; x_{i(j+1)k} \leq Wd_{max} \right)$$

$$\forall \; i \in I, \; j \in J_i, k \in K_j \qquad (13)$$

The set of constraints (14), (15), (16), and (17) indicates that the total continuous driving time in the 1st part of a duty should be greater than or equal to the minimum duration of the 1st part of a duty in shift k (α_{jk}) and the total continuous driving time of the 2nd part of a duty should be less than or equal to the maximum duration of the 2nd duty in shift k ($\delta_{(j+1)k}$). Otherwise, the total continuous driving time of the 1st part of a duty should be less than or equal to the maximum duration of the 1st part of a duty in shift k (α'_{jk}) and the total continuous driving time of the 2nd part of a duty should be greater than or equal to the minimum duration of the 2nd duty in shift k ($\delta'_{(j+1)k}$) (see Figure 3.6). This set of constraints enforces to satisfy a condition of either the crew takes a MB at the earliest time or the crew takes a MB at the latest time. This set of constraints also accommodates conditions in which the crew takes a MB between the earliest break time and the latest break time in the shift.

$$\left(\sum_{i \in I_k} t_i \; x_{ijk} + \sum_{i,i' \in I_k} \zeta_{ii'jk} \; y_{ii'jk} \geq \alpha_{jk}\right) \wedge \quad \forall \; i, i' \in I, i \neq i', j \in J_i, k \in K_j \tag{14}$$

$$\left(\sum_{i \in I_k} t_i \; x_{i(j+1)k} + \sum_{i,i' \in I_k} \zeta_{ii'(j+1)k} \; y_{ii'(j+1)k} \leq \delta_{(j+1)k}\right) \vee \quad \forall \; i, i' \in I, i \neq i', j \in J_i, k \in K_j \tag{15}$$

$$\left(\sum_{i \in I_k} t_i \; x_{ijk} + \sum_{i,i' \in I_k} \zeta_{ii'jk} \; y_{ii'jk} \geq \alpha'_{jk}\right) \wedge \quad \forall \; i, i' \in I, i \neq i', j \in J_i, k \in K_j \tag{16}$$

$$\left(\sum_{i \in I_k} t_i \; x_{i(j+1)k} + \sum_{i,i' \in I_k} \zeta_{ii'(j+1)k} \; y_{ii'(j+1)k} \leq \delta'_{(j+1)k}\right) \quad \forall \; i, i' \in I, i \neq i', j \in J_i, k \in K_j \tag{17}$$

Constraint (18) and constraint (19) calculate the total actual driving time in shift k (Wt_k) which is equal to the total working time of all duties in the shift. The total actual driving time within this shift must not exceed the upper bound (Wt_{max}) and lower bound (Wt_{min}).

$$\left(\sum_{j \in J_i} \sum_{i \in I_k} t_i \; x_{ijk} + \sum_{j \in J_i} \sum_{i,i' \in I_k} \zeta_{ii'jk} \; y_{ii'jk} \leq Wt_{max}\right) \quad \forall \; i, i' \in I, i \neq i', j \in J_i, k \in K_j \tag{18}$$

$$\wedge$$

$$\left(\sum_{j \in J_i} \sum_{i \in I_k} t_i \; x_{ijk} + \sum_{j \in J_i} \sum_{i,i' \in I_k} \zeta_{ii'jk} \; y_{ii'jk} \geq Wt_{min}\right) \quad \forall \; i, i' \in I, i \neq i', j \in J_i, k \in K_j \tag{19}$$

Constraint (20) restricts the spread time of a shift from exceeding the maximum allowed total spread time. The spread time of a shift (St_k) is equal to the total working time plus the transition time between duties (ROs). The third term in the left hand side of this equation is the transition time between duties in a shift. This transition time is the time required by the crew to take a MB during the period of ROs in the shift. Constraint (21) and constraint (22) are the variable restrictions for 3 binary variables and 1 real variable, respectively.

$$\sum_{j \in J_i} \sum_{i \in I_k} t_i \; x_{ijk} + \sum_{j \in J_i} \sum_{i,i' \in I_k} \zeta_{ii'jk} \; y_{ii'jk} + \sum_{i,i' \in I_k} \zeta_{ii'j(j+1)k} \; z_{ii'j(j+1)k} \leq St_{max} \quad \forall \; i, i' \in I, i \neq i', j \in J_i, k \in K_j \tag{20}$$

$$x_{ijk} \in \{0,1\}, \; y_{ii'jk} \in \{0,1\}, \; z_{ii'j(j+1)k} \in \{0,1\} \tag{21}$$

$$\sigma_{ijk} \in \mathbb{R} \tag{22}$$

4.2.1 Solution Techniques

The CP problem is characterised by a set of decision variables and their domains, and a set of constraints involving these variables. The set of variables is denoted by X and each variable $x_i \in X$, has a finite domain of possible values that the variable can take. A set of constraints C restricts the values the variables can simultaneously acquire. The problem definition is completed by a branching strategy which is also known as enumeration or search strategy.

Searching for solutions is performed by means of a tree search algorithm. Although the propagation engine is effective, the number of nodes may still be too large and leads to a very long optimisation process and extremely large memory usage. To avoid such tendency, the search may be limited in a number of ways. The first one is by setting a maximum computational time. The search process will terminate as soon as the computational time exceeds this limit. The second is by setting a maximum number of nodes to explore. The search process will terminate when the number of explored nodes exceeds this threshold. The third is by setting a maximum depth for the search tree to explore. The branch-and-bound will only explore nodes of the search tree up to this maximum depth. The forth is by setting the number of solutions. The search process will terminate when this acceptable number of solution is reached. The last is by setting a maximum number of backtracks. When exploring the search tree, the algorithm may backtrack or it returns to the parent node of the current node to explore a new branch. Thus the search process will terminate when the number of backtracks is reached. Optimisation programming language such as ILOG OPL Studio allows a number of instructions for the search procedure. Instructions such as 'timeLimit' limits the CPU time in seconds, 'firstSolution(n)' returns the first n solutions only, and instruction 'failLimit' limit the number of failures. In this study, we apply the 'timeLimit' instruction to terminate the search according to the given time limit.

A search strategy defines the traversal of the search tree. The test problems were solved with different search mechanisms of depth first search (DFS) and best first search (BFS). The DFS is similar to the BFS from the algorithmic point of view. The DFS starts at the root node and the search proceeds by moving downward to its first descendant. This process backtracks to the parent node when a leaf is reached and then continues to its next descendant, if it exists. The DFS has an advantage that it needs

small memory, while the BFS can improve the algorithm's convergence speed. We applied the DFS because it exhaustively explores the search tree and handles memory management better. However, if maximum depth is very large, the DFS may take very long to find a solution or will not discover it at all.

Given a set of travelling tasks (trips) from a train timetable, the variables can be declared which correspond to the sequence of trips covered by one duty. In addition to this, the constraints that dictate the feasibility of the duty can be further stated such as the time required between trips and maximum continuous driving time allowed. A search procedure will then be carried out to generate potential trip sequences. The optimised solution is a schedule which satisfies all constraints and minimises the objective function or equivalently the highest possible crew utilisation.

4.2.2 Computational Results

The CP model was formulated using ILOG OPL Development Studio 6.3. This commercial package can help with modelling and integrating CP components and mathematical programming. The data contains information about the train identification with its departure time, its departure station, its arrival time and its arrival station. The input data are retrieved from the operational environment of one line in the railway network. By considering this initial data, we derived several test data as problem instances.

This railway crew scheduling can be regarded as a resource allocation problem, where each activity has to be assigned to a resource. A natural way in modelling such a problem would be to have one decision variable for each activity and then define the set of activities assigned to each resource and ensure that the connectivity constraints such as the minimum time required between trips are not violated.

The scheduler module of OPL Studio provides several ways of representing the types of resource. A global constraint represents that a value equal one when activity trip *i* has been assigned to the resource identified as duty *j* which is element of the set of alternative resources *J*. Any given activity requires a predefined length of slot of capacity *j*. Given a set of duties *J* with given time capacity, a set of trips with known travelling time and transition times between trips, the scheduling problem then decide which trips to assign to a duty in such a way that all related constraints are satisfied. Alternatively, the decision variables can be modelled using arrays of *interval* variables.

This variable represents the trips and captures the departure time, arrival time and travelling time of each trip if it is present in the schedule. The length of this interval is equal to its arrival time minus its departure time or equivalently the distance between the start time of the earliest scheduled trip of the duty and the end time of the latest scheduled trip of the duty.

In disjunctive scheduling, each resource can execute at most one activity at a time. To prevent the trips within a duty which may overlap each other in time when they are assigned to the same resource, we eliminated the value of resource that has been assigned to those activities from the domains of all activities. To be categorised as feasible, a duty has to satisfy many constraints. For example, the departure time of the trip i plus duration of the driving time is less than or equal to the departure time of the next trip i'. The arrival station of the previous trip i must be the same as the departure station of the next trip i'. Every duty (shift) should start and end at the same depot and the total working time should be less than or equal to the maximum working time allowed. The feasible duties compose a schedule and for a schedule to be acceptable it has to minimise total working time of the crew. By imposing constraints for the actual driving time and the spread time allowed on the model, the optimisation model eventually minimises the variation of spread times from the regular crew working time. The total working time of the crew consists of driving period and non-driving period. The non-driving periods or idle time of the crew corresponds to the transition times in this model. Ultimately, a feasible schedule is any acceptable schedule with the highest utilisation of crew to increase productivity.

The CP model was tested using generated random instances by assuming the time interval between trips is 10 minutes (120 trips per line with 2 lines). Normal daily working time was equal to 8 hours and maximum spread time allowed was set to be 12 hours. The minimum length and maximum length of working periods of the 1st duty were set to be 3 hours and 5.5 hours, respectively. Whereas the minimum length and maximum length of working periods of the 2nd duty were set to be 2 hours and 4.5 hours, respectively. The computational results obtained from all the generated problem instances are summarised in Table 4.1.

Table 4.1 Computational results of CP with randomly generated problem instances.

Search	**DFS**				
Instance	**No. of Trips**	**Feasible Duties (FDs)**	**Avg. Traveling Time (min)**	**Avg. Transition Time (MB)**	**Driving Time (%)**
DT-01	25	6	469	30	0.869
DT-02	45	11	480	30	0.888
DT-03	65	17	478	30	0.885
DT-04	85	24	487	30	0.902
DT-05	105	35	477	30	0.883
DT-06	125	41	479	30	0.887
DT-07	145	48	465	40	0.861
DT-08	165	52	483	35	0.894
DT-19	185	63	488	45	0.904
DT-10	205	69	485	35	0.898
DT-11	225	74	485	35	0.898
Search	**BFS**				
Instance	**No. of Trips**	**Feasible Duties (FDs)**	**Avg. Traveling Time (min)**	**Avg. Transition Time (MB)**	**Driving Time (%)**
DT-01	25	6	467	30	0.865
DT-02	45	11	479	30	0.888
DT-03	65	17	479	30	0.887
DT-04	85	24	488	30	0.898
DT-05	105	35	477	40	0.887
DT-06	125	42	477	35	0.887
DT-07	145	47	480	37	0.888
DT-08	165	50	488	35	0.904
DT-19	185	63	480	40	0.888
DT-10	205	70	485	40	0.898
DT-11	225	74	478	35	0.885

The size of the problem increases considerably with the increase of the number of trips included. Several different data sets were considered in this study with the number of feasible duties (FDs) ranging from 6 to 74. The number of FDs will increase with the number of trips included in the problem as indicated by the CP model. Driving time of the crew is the ratio between the total working time and the total spread time in the shift. This driving time was used to measure the performance of the obtained schedules (productivity rate). By analysing the results, we noted that the productivity rate of the crew increase slightly with the size of the instances with some fluctuations. The crew driving time percentage for each data instance is shown in Figure 4.7.

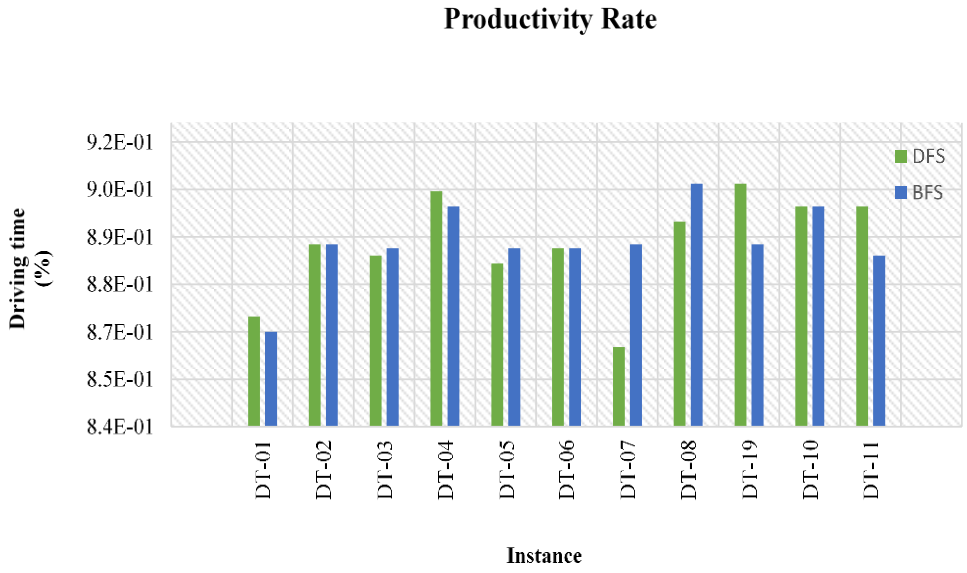

Figure 4.7 Driving time of the crew based on the search methods.

The smallest problem size considered had 272 variables and 3034 constraints and the largest problem size had 1118 variables and 12284 constraints. As can be seen in Table 4.2, the DFS approach required slightly more computational time than the BFS method. Computational results have shown that in most cases, there was no significant improvement after one hour of running time. Therefore, a time limit of 60 minutes was imposed for each data set.

Table 4.2 Computational performances based on the search methods.

Search	DFS			BFS		
Instance	**CPU time***	**Variables**	**Constraints**	**CPU time***	**Variables**	**Constraints**
DT-01	1.08	272	3034	1.05	299	3106
DT-02	3.25	488	5367	3.16	386	5354
DT-03	17.34	822	8736	15.12	500	8921
DT-04	73.05	978	10715	66.14	755	10722
DT-05	90.02	979	10799	90.02	823	11343
DT-06	119.12	996	11288	103.51	890	11616
DT-07	147.44	1022	12126	145.09	1107	12201
DT-08	188.17	1024	12132	180.04	1108	12239
DT-19	222.36	1031	12180	204.96	1108	12255
DT-10	255.38	1033	12186	239.58	1116	12282
DT-11	297.87	1035	12217	269.60	1118	12284

* In seconds using Intel Core 2 Duo 1.96 GHz Processor with 3.46 GB of RAM.

The experimental results showed that the CP model produced optimal solutions for most problem instances. The average relative deviation of all datasets from the optimum is less than 5%. However, when more trips were included in the problem, both the DFS and the BFS methods failed to achieve optimality.

The model sometimes failed to schedule a few trips but the overall results indicate that the model can produce feasible schedules within a reasonable computational time. The results obtained indicate that the CP approach is very sensitive to the increase of the problem size. Also, by using commercial CP software with a default search strategy sometimes lead to unpredictable results. It can be noted that the embedded basic search strategies is not sufficient especially for solving practical large-sized problems. The CP model has to be coupled with suitable user defined search techniques which instantiate the variables to improve the performance of the method. Figure 4.8 shows that the number of variables and constraints of the problem have a positive correlation with the increase of the problem size.

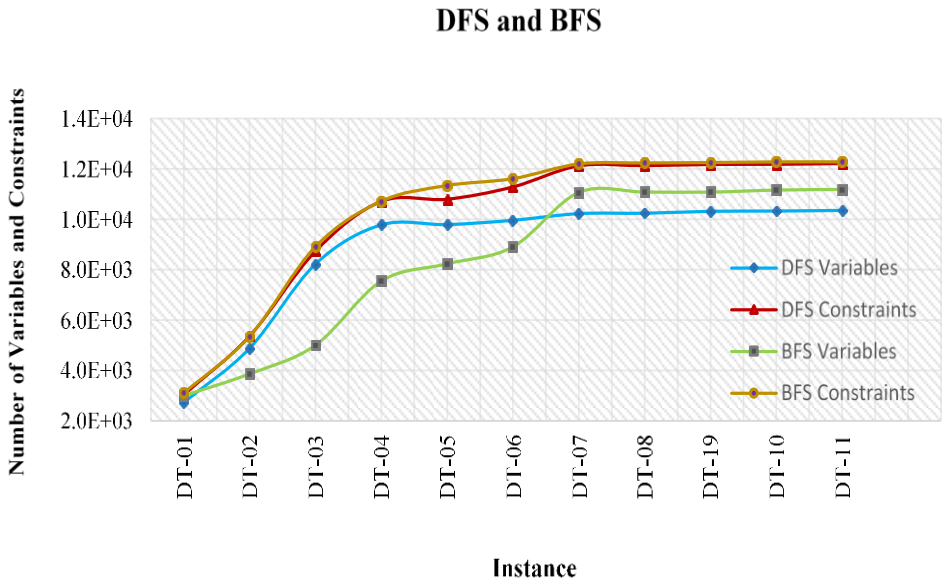

Figure 4.8 Variables and constrains of the search methods.

4.3 CONCLUSION

This chapter presents a CP-based model and its solution approach to solve railway CSP. The objective of the model is to minimise the total working time by minimising variation of spread times from the regular crew working time. The best solution obtained by the DFS or the BFS methods is used as the base in determining the relative deviation of solutions from the optimal. In a few cases, both the DFS and the BFS failed to reach optimality when solving small-sized datasets, but the relative deviation is very small and can still be considered as acceptable solutions. The CP formulation, however, is more natural in representing the problem and requires much fewer variables and constraints than MIP-based methodologies. This is due to the global constraint that capable of representing complex relationships between variables which in turn provides effective domain reduction. Using the CP technique, the model provides acceptable results in all test datasets. The overall results indicate that the CP model can produce feasible railway crew schedules within a reasonable computational time.

Chapter 5: **Metaheuristics**

The CSP falls into the category of combinatorial complex optimisation problem. As the computation progresses, the number of potential solutions is sequentially compounded leading to a large number of possibilities. Due to the combinatorial nature of the CSP, heuristic algorithms are the most promising approach for solving the problem. The main limitation with many of conventional heuristic algorithms is their difficulty to escape from locally optimal solutions. The search is usually conducted from a single point in the solution space and continuously searches for improved solutions until there is no possible improvement. This local search method makes the search easily trapped in local optima. In an attempt to deal with this problem, several metaheuristics approaches have emerged for solutions to combinatorial complex problems such as simulated annealing (SA), tabu search (TS), genetic algorithms (GAs), ant colony optimisation (ACO), and particle swarm optimisation (PSO). These approaches are not sensitive to initial solutions and allow the application of parallel processing.

5.1 OPTIMISATION PROBLEM

Practical optimisation problems involve a high complexity and require extensive computational times because of the number of potential solutions. The approximate methods are generally used to resolve this class of problems. These methods are based on an iterative exploration of the search space to find a good quality solution in reasonable computational times. Figure 5.1 illustrates the global minimum and the local minimum in the solution space. The approximate methods, among others are the neighbourhood methods, such as Local Search, Simulated Annealing (SA), and Tabu Search (TS).

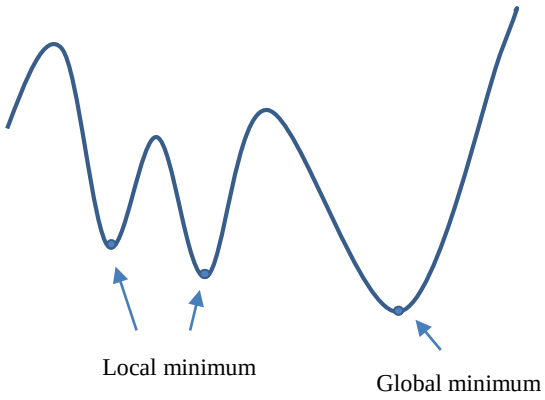

Figure 5.1 Global optimum of a minimisation criterion in the solution space.

5.1.1 Combinatorial Optimisation Problem

Combinatorial optimisation can be regarded as optimising a linear function based on other linear functions over a finite set of possible solutions. Combinatorial optimisation is the discipline of decision making in case of discrete alternatives and can be regarded as optimising a linear function based on other linear functions over a finite or infinite number of possible solutions (Aarts & Lenstra, 2003). A combinatorial optimisation problem $P = (s, f)$ can be defined by;

- a set of variables $X = \{x_1,..., x_n\};\}$
- variable domains $d_1, ...,d_n$;
- constraints among variables;
- an objective function f to be maximised or minimised;

The set of all possible solutions is $S = \{s = \{(x_1,v_1),.....,(x_n,v_n)\} \mid v_i \in d_i$, s satisfies all the constraints$\}$. Each element of the set S can be seen as a candidate solution. S is called a search space or solution space. The solution of a combinatorial optimisation problem is the element of S, $s^* \in S$, with optimum objective function value. Objective function f is a function that measures the value of each solution in S. In the case of minimising objective function value, $f(s^*) \leq f(s), \forall s \in S$. s^* is called a globally optimal solution of (S, f) and the set $S^* \subseteq S$ is called the set of globally optimal solutions.

The difficulty of solving combinatorial optimisation problems yet their great practical importance, have led to a large number of proposed solution techniques. The existing solution methods for solving combinatorial optimisation problems can be classified as either exact or approximate algorithms. Exact algorithms can guarantee

to find an optimal solution (if there is one exists) or prove that no feasible solution exists. When optimal solutions cannot be computed efficiently in practice, it is usual to find a good trade-off between optimality and efficiency.

5.1.2 NP-Complete Problems and Combinatorial Explosion

A decision problem can be expressed as a problem that requires a 'yes' or 'no' answer. The class P refers to the set of all decision problems for which polynomial time algorithms exist. The class of P is the class of problem solvable in polynomial time and it has at least one algorithm to solve it. A hard or intractable problem requires solution time which is an exponential function of its problem size. The class NP (nondeterministic polynomial-time) is a class of problems in which its solution can only be guessed and evaluated in polynomial time with no known rule to make such guess, hence non-deterministic. This class includes most combinatorial optimisation problems and all problems that are in P, $P \subseteq NP$. Within the class NP there are the NP-complete problems. The NP-complete problems are by definition the hardest problems in the class NP. If one NP-complete problem can be proved to be solvable in polynomial time, then each problem in NP can be solved in polynomial time, thus P = NP would follow.

Computational complexity is the measurement of how much resource is required to solve the problem. When an algorithm is designed to solve a particular problem, it is important to know how much time and space an implementation will consume. Mathematical methods can often be applied to predict how much time and space required by an algorithm before it is implemented in the form of a computer program. Having understood that will help save work in order to test its performance and to decide what kind of solution technique is suitable. If the problem is hard, finding exact optimal solution may be impractical and therefore have to resort to an approximate solution obtained by heuristics.

Since most practical combinatorial optimisation problems are NP-hard, it cannot guarantee to obtain an optimum solution. In spite of the numerous studies on this type of problems, no efficient algorithm had been found yet for solving the problems. Therefore, the use of heuristics and/or metaheuristics is very promising and is completely justified in obtaining approximate solutions when solving such difficult problems.

5.2 CONSTRUCTIVE HEURISTICS

Constructive heuristics build a solution by iteratively add solution components until a feasible solution has been found. Neighbourhood search methods are iterative procedures in which a neighbourhood $\mathcal{N}(s)$ is defined for each feasible solution, and the next solution is searched among the solutions in $\mathcal{N}(s)$. The neighbourhood of a solution is obtained by moving each pair of consecutive or non-consecutive positions of the sequence that represents the solution. Local search explore the search space by moving from solutions to its neighbouring solutions in the hope improving the value of the objective function. Local searches are based on the definition of a set $\mathcal{N}(s)$ of solutions in the neighbourhood of any solution s.

5.2.1 Initial Solution by Constructive Heuristic (CH)

Local search explores the neighbourhood $\mathcal{N}(s)$ of a current solution iteratively and finds a better solution $s^b \in \mathcal{N}(s)$ according to some criteria. The initial solution is constructed by means of a constructive heuristic from an ordered list of trips with their attributes to form crew duties. We break down this phase into two sub-phases. The first is the initialising phase that includes listing all vehicle blocks in ascending order of start time, $v_b = \{v_{b1}, v_{b2},\ldots, v_{bn}\}$; and grouping them based on the length of run, $l_r = \{l_{r1}, l_{r2},\ldots, l_{rn}\}$. Cutting vehicle blocks into trip segments is also performed in this phase $t_s = \{t_{s1}, t_{s2},\ldots, t_{sn}\}$. Some vehicle blocks may have sufficient length to be divided into two straight runs that are approximately equal to the length of regular working hours (8 h) each. Other vehicle blocks may be divided into one straight run of 8 h with a piece left over. The remaining of the vehicle blocks do not need to be divided as they have sufficient length to form one straight run with no pieces left over. The second is combining phase which is joining trip segments by progressively selecting uncovered trip segments from a block to create feasible duties. The CH method is described in a pseudo code form in Algorithm 5.1. It is desirable to construct feasible schedules that will minimise idle transition times and maximise the length of the route per cycle time. The cycle time is the time spent to drive a round trip plus idle intervals on a route.

The pseudo code of the CH algorithm is as follows:

Algorithm 5.1: Generating initial solutions

```
1   procedure Constructive_Heuristics()
2   Input all relevant data: trip list, vehicle blocks, parameters, and constraints;
3   Output: initial solutions S^0;
4   begin Initialisation ()
5       // first phase
6       i ← 1 to I (I total number of trips) ∀ i ∈ v_b;
7       v_b ← {v_b1, v_b2, v_b3, …, v_bn} in ascending starting time order, ∀ i ∈ v_b;
8       l_r ← {l_r1, l_r2,…, l_rn} ∀ i ∈ v_b;
9       t_s ← {t_s1, t_s2,…, t_sn} ∀ i ∈ v_b;
10      // second phase
11      ζ_n ← 1 to 𝒩 (𝒩 number of trip segments);
12      list trip segments sequentially;
13      S^0 = ∅ ;
14      ζ_n ≠ ∅ ;
15      while (ζ_n ≤ t_sn − 1) do
16          allocate trip segments into time slots based on the starting time of the trip;
17          S^0 ← S^0 ∪ {ζ_i};
18          ζ_n ← ζ_n \ {ζ_i};
19          determine possible trip segment combinations;
20      end while
21      S^0 ← S^0 + 1;
22  return (S^0);
23  end
```

5.3 SIMULATED ANNEALING (SA)

SA is a variant of local (neighbourhood) search which is based on an analog of cooling solid material. The material is heated past its melting point, then it cooled back slowly until it crystallizes into a solid state (low-energy state). This process is known as annealing. At high temperatures, the atoms in the material have high energies and more freedom to arrange themselves. When the temperature is reduced, the atom energies decrease. The structural properties of the cooled solid depend on the rate of cooling. Metropolis et al. (1953) simulated the change in energy of the system as it cools, until it converges to a steady frozen state.

This basic concept of SA derived from the analogy with the thermodynamic annealing process and it is widely used for solving combinatorial optimisation problems. At temperature *T*, the moves are accepted based on probability *P* and this probability is compared with a randomly generated number between (0, 1). *T* represents control parameter in the heuristic. SA algorithm generates a perturbation and calculates the resulting energy change, $\Delta E = f(S'_{max}) - f(S^c_{max})$ which represents a change in objective function value. If energy has decreased then the system moves to the new state, otherwise the new state is accepted with the probability. Higher energy state solutions are accepted if the calculated probability is higher than the randomly generated number.

This technique can be applied to minimization problems based on consecutive update steps where the update step length is proportional to parameters which can play the role of a temperature. Acceptance probability is dependent on the control parameter, *T* and magnitude of increase in objective value, ΔE. When ΔE is small, acceptance probability is high. When *T* is high, acceptance probability is high. As *T* decreases, acceptance probability decreases. The process of moving from one state to the next is repeated for a number of iterations at the current temperature, then the temperature is decreased and the same process is repeated until the system freezes into a steady state. In terms of search methods, SA is a stochastic local search method. It always accepts a selected better local solution and it allows accepting a worse local solution to avoid getting stuck at a local optimum, with a probability which is gradually decreased as the algorithm proceeds.

The following notations are used through the description of the SA and HCHSA algorithms.

S	: set of feasible solutions
$N(s)$	: set of neighbourhood solutions
s'	: generated solution (sample solution from neighbourhood) $s' \in S$
s^c	: current solution
s^b	: best solution found
$f(S'_{max})$	: function value of neighbourhood solution
$f(S^c_{max})$	: function value of current solution
$f(S^b_{max})$	: function value of best solution
T_0	: initial temperature
T^c	: current temperature
R	: uniformly distributed random number between 0 and 1
α	: cooling rate
i_{max}	: maximum iteration

SA can be described in the pseudo code as follows:

Algorithm 5.2: Simulated Annealing

```
1   procedure Simulated_Annealing()
2   Input: initial schedules;
3   Output: best solutions s^b;
4   begin Simulated_Annealing()
5       define_neighbourhood_structure();
6       s^c ← get_initial_solution (S) ;
7       s^b ← s^c;
8       T ← initial_temperature();
9       while (¬ stopping_criterion)
10          search_neighbourhood ← true;
11          while (search_neighbourhood)
12              s' ← sample_solution_from_neighbourhood(N(s));
13              if ( f(s') > f(s) ) then
14                  P_accept ← exp ( f(s) – f(s') / T ) ;
15              else
16                  P_accept ← 1;
17              end if
18              if (random() < P_accept ) then
19                  s^c ← s' ;
20                  search_neighbourhood ← false;
21              else
22                  search_neighbourhood ← searching_current_neighbourhood();
23              end if
24              if (s^c < s^b ) then
25                  s^b ← s^c;
26              end if
27          end while
```

```
28          T ← update_temperature(T);
29      end while
30  return s^b;
31  end
```

5.3.1 Solution Improvement by Hybrid Constructive Heuristic Simulated Annealing (HCHSA)

SA is motivated by an analogy to the physical process of annealing, where the temperature of a material is reduced to achieve its thermal equilibrium (Kirkpatrick et al. 1983). This principle is applied in combinatorial optimisation problems to optimise the objective function value. The advantage of this technique is that it can avoid local optima by occasionally allowing the acceptance of non-improving solutions in the hope that a better solution may be found later on.

We utilize the SA metaheuristic to improve solution and to derive a near-optimal solution. The design of SA algorithm to solve the railway crew scheduling generally consists of four components, an objective function (analogue of energy) to be optimised; the neighbourhood structure that defines how to efficiently generate random solutions from neighbourhood; an acceptance criterion that is a criterion for accepting or rejecting a new generated solution; and a cooling schedule. Implementation details of the proposed HCHSA algorithm are given as follows:

a) *Initial sequence*. An initial schedule is obtained from the best schedule returned by the CH algorithm. This schedule is assumed as the current solution. A set of scheduled trips, $\mathcal{I} = \{i_1, i_2, i_3, \ldots\ldots, i_n\}$, that need to be serviced during a defined period of time is identified by its departure station, departure time, arrival station, arrival time, represented by vector ds_i, dt_i, as_i, at_i, respectively. The algorithm sorts an array $\mathcal{I} = \{i(0), \ldots, i(n-1)\}$ of n trips in increasing order of departure time. Every iteration removes an element from the input data, inserting it into the correct position and simultaneously moves the data in the already-sorted list, until no input elements remain. Initialising can be considered as the process of queuing all the trips in the right order to their assigned duties. The constraints considered in this case are connectivity restrictions, traveling times and transition times.

b) *Neighbourhood structure*. The neighbourhood structure defines a method of generating alternative solution from a current solution. We generate the

neighbourhood using swap and insert mechanisms. The neighbourhood structure proposed in Elizondo et al. (2010) is adapted for our problem. Two different duties s_x and s_y, with the number of trips v and w, respectively, are selected and denoted as, $s_x = \{i_{x1}, i_{x2}, \dots, i_{xm}, i_{x,m+1}, \dots, i_{xv}\}$ and $s_y = \{i_{y1}, i_{y2}, \dots, i_{yn}, i_{y,n+1}, \dots, i_{yw}\}$. The swap operation is performed on the selected duties by exchanging the position of trip segments (t_s) between two blocks. The swap operation is only performed on duties with trip segments originate and terminate at the same crew depot. The insert operation is performed by moving one trip segment to another duty. This operation is only applied to the trip segments that arrive and depart from stations with a local connection.

c) *Acceptance criterion*. Given the initial configuration, a small perturbation is performed by exchanging a piece of the trip between two duties and moving a piece of the trip within one duty to another. The change in the objective function value is then calculated. If it gives a better solution, the new solution is accepted. Otherwise it still has a chance to be accepted with a particular condition that is the value of the function $f(\Delta E) = e^{-\Delta E/T}$ is greater than a randomly generated value between 0 and 1. When the solution is accepted, the current neighbourhood configuration is updated as the algorithm proceeds.

d) *Cooling schedule*. An annealing or a cooling schedule consists of (i) the initial value of temperature parameter T_0, (ii) the cooling factor (a method of gradually decreasing the value of T^c), (iii) the number of iterations to be performed at each T^c before it is decreased, and (iv) the stopping criterion to terminate the algorithm.

The overall method of the proposed HCHSA algorithm is captured in the pseudo-code form in Algorithm 5.1 and Algorithm 5.3.

Algorithm 5.3: Simulated Annealing

```
1   procedure Simulated_Annealing()
2   Input : initial solutions S^0;
3   Output: best found solutions (duties);
4   begin Simulated_Annealing()
5   // initialisation step
6       select an initial solution and set it as the current solution;
7       s^c ← s ∈ S;
8       calculate S^c_max;
9       s^b ← s^c;
10      f(S^b_max) ← f(S^c_max);
```

```
    select an initial temperature, T0;
    T^c ← T0;
    select maximum iterations i_max;
    select temperature reduction function, α (cooling rate);
    initialise step counter i ← 0;
    define neighbourhood structure();
// iterative step
    while (i < i_max and T^c > T0 )
        search neighbourhood;
        generate solution from neighbourhood, s ∈ 𝒩(s);
        s´ ← s ∈ 𝒩(s);
        evaluate sample solution from neighbourhood;
        ΔE = f(S'_max) − f(S^c_max)
        if ΔE < 0 then
            s^c ← s´;
            if f(S^c_max) < f(S^b_max) then
                f(S^b_max) ← f(S^c_max);
            end if
        else
            generate random number ℛ ~ (0,1);
            P_accept = e^(−ΔE/T)
        if ℛ < P_accept ) then
            s^c ← s´ ;
            f(S^c_max) ← f(S'_max) ;
        end if
        T^c ← α T^c ;
        i ← i + 1;
        end if
        update temperature T;
        T^c ← T^c (i) ;
        return (s^b , f(S^b_max));
    end while
end
```

5.4 TABU SEARCH (TS)

TS is a higher level heuristic originally introduced by Glover (1986). TS has been successful in solving many combinatorial optimisation problems in various practical settings. The fundamental idea of TS is the use of search history to guide the search process while escaping local optima. It occasionally accepts non-improved solutions to prevent returning to recently visited solutions and caught up in cycles. By directing the search away from local optima, other region in the search space can be explored. This is accomplished by maintaining a search history of recently visited candidate solutions stored in a tabu list such that the algorithm prevents the search reconsidering those candidate solutions for a certain number of iterations. The recorded search history is expressed by a list of prohibitive moves. A corresponding

move or transition from a current solution to another solution is acceptable if it is not tabu or if a certain aspiration criterion is satisfied. When a move is accepted, then it becomes tabu in the next iterations. This means that this move is forbidden unless an aspiration level is fulfilled. The aspiration criterion is a measure for accepting tabu moves and it may be applied to allow all moves that lead to a neighbour with a better objective function value than the previously obtained solutions. Whenever a new candidate solution is adopted, it goes in the tabu list and the selected neighbouring solution replaces the current solution. The neighbourhood $\mathcal{N}(s^c)$ is modified to $\mathcal{N}(\mathcal{H}, s^c)$ and the recorded search history $\mathcal{H}$ is dynamically updated as the algorithm proceeds. If the tabu list exceeds the tabu list length, the oldest candidate solution is removed and it is no longer tabu to reconsider. After evaluating all neighbours, the one with the best objective function value is selected from amongst those that satisfy the aspiration level. When termination criteria are met, the algorithm is stopped. Otherwise the tabu list is updated and the process continues.

The notations that are used through the description of the TS algorithm are given as follows.

S	: the set of feasible solutions
s^i	: the initial solution
s^c	: the current solution, $s^c \in S$
s^b	: the best solution found
$f(s)$	: the objective function of solution s
$\mathcal{N}(s)$	: the set of neighbourhood of $s \in S$
$\mathcal{N}_t(s)$	: neighbour solution that do not violate tabu condition
$\mathcal{N}_a(s)$	: neighbour solution that meet aspiration criteria
TL	: tabu list
AC	: aspiration criterion

TS can be described in the pseudo code as follows:

Algorithm 5.4: Tabu Search
1 **procedure** Tabu_Search()
2 Input: related data;
3 Output: best found solutions;
4 **begin** Tabu_Search()
5 $s^c \leftarrow$ generate initial solution ($s \in S$);
6 $s^b \leftarrow s^c$;
7 set $AC = f(s^b)$
8 initialize_tabu list $T_L() \leftarrow \emptyset$;
9 **while** (¬ stopping criterion)
10 determine neighbourhood of current solution s^c;
11 $\mathcal{N}_t(s) \leftarrow (\mathcal{N}(s))$ select best non-tabu solution;
12 $\mathcal{N}_a(s) \leftarrow (\mathcal{N}(s))$ select solution that meet aspiration criteria;
13 $s^c \leftarrow$ get best current solution $(\mathcal{N}_t(s) \cup \mathcal{N}_a(s)) \setminus T_L()$;
14 **if** $(s^c < s^b)$ **then**
15 $s^b \leftarrow s^c$;
16 **end if**
17 update $T_L()$;
18 **end while**
19 **return** s^b;
20 **end**

5.4.1 Proposed Hybrid Constructive Heuristic and Tabu Search (HCHTS)

A combinatorial optimisation problem with a minimisation objective is a problem that requires an optimal solution $s^* \in S$ such that $f(s^*) \leq f(s)\ \forall s \in S$, where S is the finite set of all possible solutions. The railway CSP in this study is described as follows. There is a set of scheduled trips, $\{i_1, i_2, \ldots, i_n\}$, $\forall\ i_n \in I_{(n)}$. A trip is identified by the train, its departure station, its departure time, its arrival station, and its arrival time, represented by a quintuple vector t, ds_i, dt_i, as_i, at_i, respectively. There is a set of vehicle blocks, $\{v_{b1}, v_{b2}, \ldots, v_{bn}\}$, $\forall\ v_{bn} \in V_{(b)}$. There is a set of all trip segments, $\{t_{s1}, t_{s2}, \ldots, t_{sn}\}$, $\forall\ t_{sn} \in T_{s(n)}$ and a set of trip segment in line ℓ in the network, denoted by $T_{s(l)}$. A graph $G = (V, A)$, where $V = \{v_{0\,x,y}, v_{1\,x,y}, v_{2\,x,y}, \ldots, v_{n\,x,y}\} \subseteq$ HDs ∪ RPs is the vertex set. $A = \{(v_{i\,x,y}, v_{j\,x,y}): v_{i\,x,y}, v_{j\,x,y} \in V,\ i \neq j\}$ is the arc set, and d_{ij} represents the travel distance in time for each (v_i, v_j). Note that the subscript x and y are used to refer to the time and location of the node, respectively. We first apply aggregation procedure to combine trips between two adjacent RPs into a trip segment. The aggregation procedure is intended to reduce the problem size. The scheduled trips to be aggregated should be based on the same vehicle block and train line. The aggregation procedure is described in the pseudo code below.

Algorithm 5.5: Aggregation Method

```
1   procedure Aggregation_Method()
2   Input: a set of nodes v x,y ∈ V x,y ;
3   Output: a set of aggregated nodes v 'x,y ∈ V 'x,y ;
4   begin Aggregation_Method()
5       {vi' x,y, vj' x,y} ← ∅;
6       dij ← 0;
7       ω ← 0 (max continuous driving time);
8     repeat
9       for {vi x,y, vj x,y} ∈ V x,y do
10        sorts the nodes in increasing order of departure time;
11        if (vi x,y, vj x,y) ∈ HDs ∪ RPs and (vi x,y, vj x,y) ∈ V(b) then
12          if (vi x,y, vj x,y) ∈ Ts(n) and tij ≤ ω then
13            (i'x,y, j'x,y) ← (ix,y, jx,y);
14            dij ← dij;
15            ω ← dij;
16            V x,y ← V x,y \ (ix,y, jx,y);
17          end if
18        end if
19          {vi' x,y, vj' x,y} ← (i'x,y, j'x,y) ;
20          V 'x,y ← {vi'x,y, vj'x,y};
21      end for
22    until V x,y = ∅
23  return V 'x,y ;
24  end
```

Figure 5.2 shows a subset of the pattern of train movements from a real train timetable. The horizontal red line is a point of reference for inbound and outbound trains. Inbound services are trains heading towards this point, while outbound services are the trains moving away from this point. The time flows from left to right along the horizontal axis.

Figure 5.2 A subset of the pattern of train movements on lines of the rail network.

The problem is in a symmetric form in which the forward and reverse directions of crew movements are indistinguishable. If a crew travels from, point *A* (HD), for example, it first goes along segment ($A_{x,y}$, $a^{+}_{1\,x,y}$) on line ℓ_1 and reach point *X* (Figure 5.3). This route is recorded in the tabu list, the traveling time of this move is also recorded and the immediate predecessor *i*' and successor *i*" are identified. When the travel reaches point *X*, there are several possible routes that can be pursued for continuation. The transition time from point *X* to each of the possible route is analysed and checked whether it satisfies the connectivity constraints. Every time the travel reaches a location which serves as a RP, the total traveling time is calculated to check the possibility for crew relief and to ensure no violation of the allowed maximum continuous traveling time. A crew may get on and off at a RP or at a terminal location along the route. The reverse order of crew movement is not allowed unless the trip sequence has reached a RP or the trip has no successor. Now, at point *X*, the travel can continue outbound either along (X, $b^{-}_{1\,x,y}$) or ($X_{x,y}$, $a^{-}_{1\,x,y}$). Suppose the route (X, $a^{-}_{1\,x,y}$) is selected, the tabu list now is $TL = \{(A_{x,y}, a^{+}_{1\,x,y}) \rightarrow (a^{+}_{1\,x,y}, X_{x,y}) \rightarrow (X_{x,y}, a^{-}_{1x,y}) \rightarrow (a^{-}_{1x,y}, a^{-}_{1(x,y)})\}$. A part of the flexible tabu list length is used as the ongoing tabu active status.

Figure 5.3 Schematic illustration of crew movements in the rail network.

At every point, there are several possible paths to take for a crew to complete the roundtrip. 1). Continue with the same train on the same line (orange line). 2). Continue with a different train on the same line (purple dotted line). 3). Continue with a different train on the same line in opposite direction (blue dotted line). 4). Continue with a different train on a different line in the network (invisible in Figure 5.3). The sequence of trip is progressively constructed in forward order. The goal is to construct minimal cost roundtrips for the crews to cover every trip possible without violating the given constraints. We use a crew roundtrip construction heuristic to generate an initial solution s^c as the starting point of the search. The produced solutions are then refined by the TS-based algorithm. The overall procedure is outlined in the pseudo code below in Algorithm 5.6.

Algorithm 5.6: Tabu Search

```
1   procedure Tabu_Search()
2   Input: trip list, vehicle blocks, parameters, and constraints;
3   Output: best found solutions;
4   begin Tabu_Search()
5      s^c ← generate initial solution (s ∈ S^0);
6      s^b ← s^c;
7      set no. of iterations ← 0;
8      set the aspiration function AC = f(s^b);
9      initialise_tabu list T_L() ← ∅;
10     while ( ¬ stopping criterion)
11          candidate solution list ψ ← ∅;
12          determine neighbourhood of current solution s^c;
13          for s^c ∈ N(s)
14              for each trip segment t_sn ∈ T_s(n) select randomly one of the move m;
15                  select best non-tabu move m ∈ M such that m ∉ T_L();
16                  apply transition from s to s' → N(s) = {s´| = s ⊕ m, m ∈ M};
17                  t_s ← t_sn \{T_s(n)};
18                  include attributes of recent move m into the tabu list T_L();
19                  ψ ← ψ ∪ s^c;
20              end for
21          end for
22          evaluate the current solution s^c from candidate solution list ψ;
23          s^c ← get best current solution {N_nt(s) ∪ N_ac(s)} \ T_L();
24          if f(s^c) < f(s^b) then
25              set s^b ← s^c;
26          end if
27          update T_L();
28     end while
29  return s^b;
30  end
```

The proposed TS based algorithm for solving railway CSP generally consists of four main elements; a solution representation; the neighbourhood structure; an aspiration criterion; and a stopping criterion. Implementation details of the proposed algorithm are as follows:

a) *Solution representation and evaluation*. A feasible solution consists of a number of trips, idle intervals between trips in the partial duty, and a transition period in between two partial duties for a crew MB represented by t_{jk}^{i}, $\zeta_{jk}^{ii'}$, and $\zeta_{jj'k}^{ii'}$ respectively, while satisfying the given constraints, i.e Eqs. (2) – (20).

b) *Neighbourhood structure*. A neighbourhood search method $\mathcal{N}(s)$ is defined to allow iterative search for the next solution amongst the possible solutions in $\mathcal{N}(s)$. A possible solution in neighbourhood $\mathcal{N}(s)$ can be selected by applying a move.

Thus, a move m to solution s is a transition from a current solution to its neighbouring solution, where m is the member of a set of moves $\mathcal{M}$. Given a solution (t_{jk}^{i} , $\zeta_{jk}^{ii'}$, $\zeta_{jj'k}^{ii'}$), the neighbourhood is defined by a swap and insert mechanism such that, $\mathcal{N}(s) = \{s' | = s \oplus m, m \in \mathcal{M}\}$. A swap-based neighbourhood operation is performed on the selected partial duties by exchanging the position of trip segments. A swap operation exchanges the values of $t_{jk}^{i} = 1$ and $t_{jk}^{i'} = 0$ to $t_{jk}^{i} = 0$ and $t_{jk}^{i'} = 1$, where $i \neq i'$. If the move m was applied to the solution s, then it is evaluated by $\Delta(s, m) = f(s \oplus m) - f(s)$. The swap operation is only performed on duties with trip segments originating and terminating at the same crew depot. The insert operation is performed by moving one trip segment to another duty. This operation is only applied to trip segments that arrive and depart from stations with a local connection.

c) *Aspiration criterion*. The objective function $f(s)$ is used as an aspiration level for non-tabu solution. When the transition is within the tabu restriction and it is a non-improving move, then the optimal solution is reselected as this phase move and the move is recorded in the tabu list. If the move is outside of the tabu restriction or if it belongs to the tabu restriction but satisfy the aspiration criterion, then this move is recorded in the tabu list and the tabu status is adjusted accordingly. Thus a tabu move may be accepted if it produces a solution with a better function value than the best current function value. The best found solution $f(s^b)$ is then updated.

d) *Stopping criterion*. The stopping criterion is defined to terminate the TS. The number of sequential iterations without improved objective function value is used as a termination criterion. Figure 5.4 shows the flow chart of TS algorithm.

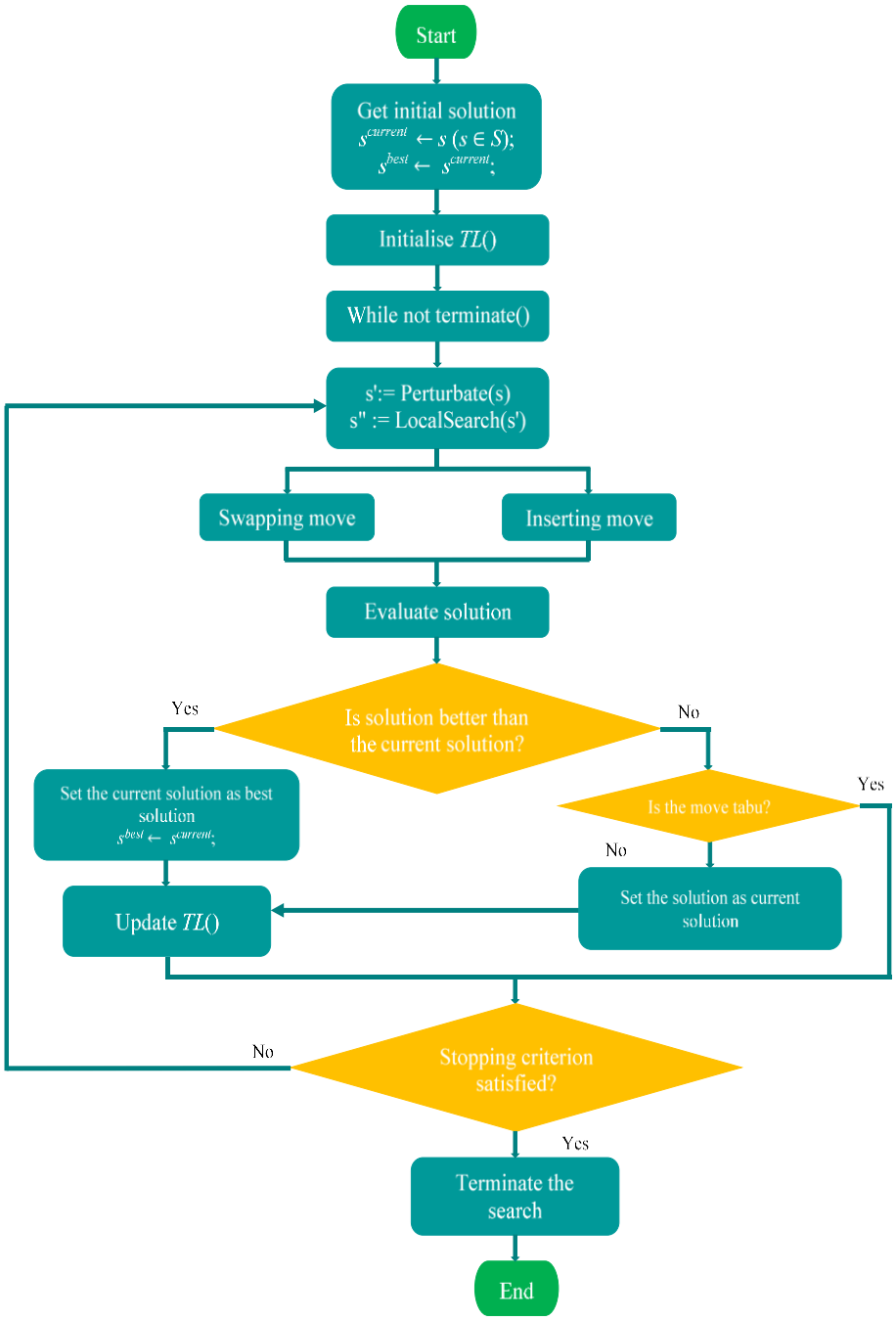

Figure 5.4 Flow chart of TS-based algorithm.

5.5 HYBRID CONSTRAINT PROGRAMMING AND SIMULATED ANNEALING (HCPSA)

Hybridising CP and SA is based on the idea of combining the strength of CP method and metaheuristics to derive better solutions for the problem under study. By hybridising with exact techniques such as CP, the performance of the algorithm can be improved by reducing computational time, improving the effectiveness of the search, and limiting the search space, in such a way that it leads the search to the promising region. The solution method consists of a two-phase algorithm. The first phase is the construction of an initial solution by CP. The obtained initial solution is then improved by SA metaheuristic. The overall procedure for generating initial solution by CP is given in the pseudo code as below.

Algorithm 5.7: Initial solution construction by CP

```
1   procedure CP()
2   Input: trip list, vehicle blocks, parameters, and constraints;
3   Output: best initial solutions;
4   begin CP()
5      set s^b ← s^c;
6      while i ≤ number of trip do
7           select a trip segment t_s ← t_sn \{T_s(n)};
8           evaluate the selected variable;
9           if t_s is feasible then
10             i ← i + 1;
11          else
12             select a trip segment t_s ← t_sn \{T_s(n)};
13          end if
14          if current function value f(s^c) < best function value f(s^b) then
15             set s^b ← s^c;
16          end if
17     end while
18  return s^b;
19  end
```

5.6 HYBRID TABU SEARCH AND SIMULATED ANNEALING (HTSSA)

The proposed hybrid constructive heuristic and TS algorithm seems to have comparatively larger computational time. It can be noted that the initial feasible solution generated for the TS algorithm affects the successful implementation of the algorithm. The possibility of cycling is still remain as the tabu list uses deterministic means. Elements stored in the tabu list depend on a tabu tenure which defines the tabu list size and how long to use a memory. However, the aspiration criteria can deal with this possibility and prevent the search from getting trapped into a local optima. This feature allows the search to check condition of acceptance and to override the tabu status on a candidate solution. It provides a means of backtracking of recent solutions, leading to a different path towards an improved solution. To exploit and combine the advantages of individual metaheuristic, a hybrid optimisation method which combine TS and SA is proposed. More specifically, the initial feasible solution obtained from the constructive heuristic is used as input for the HTSSA algorithm.

The following notations are used through the description of the HTSSA algorithm.

S	: set of feasible solutions
$N(s)$	: set of neighbourhood solutions
s'	: generated solution from neighbourhood $s' \in S$
s^c	: current solution
s^b	: best solution found
$f(s')$	: function value of neighbourhood solution
$f(s^c)$	: function value of current solution
$f(s^b)$	: function value of best solution
T_0	: initial temperature
T^c	: current temperature
R	: uniformly distributed random number between 0 and 1
α	: cooling rate
i_{max}	: maximum iteration

Algorithm 5.8: Hybrid Tabu Search Simulated Annealing (HTSSA)

procedure HTSSA()
Input: initial solutions;
Output: best found solutions;
begin HTSSA()
  $s^c \leftarrow$ get the initial solution ($s \in S^0$);
  $s^b \leftarrow s^c$;
  $f(s^b) \leftarrow f(s^c)$;
  select an initial temperature, T_0;
  $T^c \leftarrow T_0$;
  select temperature reduction function, α (cooling rate);
  initialise step counter $i \leftarrow 0$;
  set the aspiration function $AC = f(s^b)$;
  initialise_tabu list $T_L() \leftarrow \emptyset$;
  while (¬ stopping criterion)
    candidate solution list $\psi \leftarrow \emptyset$;
    determine neighbourhood of current solution s^c;
    for $s^c \in \mathcal{N}(s)$
      for each trip segment $t_{sn} \in T_{s(n)}$ select randomly one of the move m;
        select best non-tabu move $m \in \mathcal{M}$ such that $m \notin T_L()$;
        apply transition from s *to* $s' \rightarrow \mathcal{N}(s) = \{s' \mid = s \oplus m, m \in \mathcal{M}\}$;
        $t_s \leftarrow t_{sn} \setminus \{T_{s(n)}\}$;
        include attributes of recent move m into the tabu list $T_L()$;
        $\psi \leftarrow \psi \cup s^c$;
      end
    end
    evaluate the current solution s^c from candidate solution list ψ;
    $s^c \leftarrow$ get best current solution $\{\mathcal{N}_t(s) \cup \mathcal{N}_a(s)\} \setminus T_L()$;
    if $f(s^c) < f(s^b)$ **then**
      set $s^b \leftarrow s^c$;
    else
      generate random number $\mathcal{R} \sim (0,1)$;
      $P_{accept} = e^{-\Delta E/T}$
      if $\mathcal{R} < P_{accept}$) **then**
        $s^c \leftarrow s'$;
        $f(s^c) \leftarrow f(s')$;
      end if
    end if
    update $T_L()$;
    $T^c \leftarrow \alpha T^c$;
    $i \leftarrow i + 1$;
    update temperature T;
    $T^c \leftarrow T^c(i)$;
  end while
return (s^b, $f(s^b)$);
end

5.7 COMPUTATIONAL EXPERIMENTS

To evaluate the scheduling methods presented in Chapter 5 of this thesis, benchmark instances were randomly generated for the problem with 24-h scheduling horizons. A sample train schedule with 12 trips is given in Table 3.1 in Chapter 3. The railway crew scheduling in this study is to create a feasible set of crew duties to cover a given set of trips. A feasible crew duty (shift) includes one or two partial duties, a period of MB, idle transition times, and the sign-on and sign-off activities. The accumulated time represents the crew total working time.

5.7.1 Constructive Heuristic (CH)

The exact solution of the mathematical model was obtained using Xpress-Optimizer (FICO) algorithms for mixed integer problems. 3 HDs and 5 RPs were considered with the number of trips varied between 25 and 120 trips for instances solved by Xpress-Optimizer and the HCHSA algorithm. Whereas the number of trips varied between 258 and 732 trips for instances solved by the CH and HCHSA algorithms. All trips in a day were divided into four different intervals, 05.00 – 08.59; 09.00 – 12.59; 13.00 – 16.59; and 17.00 – 22.59. Normal daily working time was fixed to 8 h and maximum spread time allowed was 12 h. The minimum and maximum lengths of working periods of the 1st part of a duty were 3 h and 5.5 h, respectively. Whereas the minimum and maximum lengths of working periods of the 2nd part of a duty were set to 2 h and 4.5 h, respectively. The length of the ROP was 2.5 h within which a MB of minimum 0.5 h is required between the third and the sixth hours of an 8 h duty. There was a time allowance of about 10 min for signing-on or signing-off when a crew starts or ends his duty at a HD. Table 5.1 presents computational results, i.e. the number of feasible duties, the total objective value, driving time, excess cost, and run time in seconds.

Table 5.1 Computational results of the Constructive Heuristic (CH).

Instance	# Trips	# Duties	Objective Value	Total Elapsed Time (min)	Driving Time (%)	Excess Cost (%)	CPU Time (sec)	Average Working Time (min)
DH-01	258	112	54882	60376	90.90	10.01	012	490.02
DH-02	350	121	60257	66935	90.02	11.08	020	497.98
DH-03	425	139	68943	78522	87.80	13.89	028	495.79
DH-04	455	156	79095	91125	86.80	15.21	043	507.02
DH-05	470	153	85526	96218	88.89	12.5	057	555.36
DH-06	485	160	80480	89425	90.00	11.11	067	503.00
DH-07	550	157	84309	95807	88.00	13.64	089	537.00
DH-08	582	162	82135	94413	87.00	14.95	112	507.01
DH-19	607	169	82810	97538	84.90	17.79	124	490.00
DH-10	625	172	84795	97712	86.78	15.23	146	492.90
DH-11	660	176	87120	102294	85.17	17.42	341	495.00
DH-12	732	182	89874	105704	85.02	17.61	595	493.81

All small-sized instances were solved to optimality by Xpress-Optimizer. As can be seen from Table 5.2, the computational time increases significantly as the size of the instance become larger. The largest instance was solved by Xpress-Optimizer with a reasonable computational time.

Table 5.2 Computational results of the Mathematical Programming (MP) and the Hybrid Constructive Heuristic Simulated Annealing (HCHSA) algorithm.

Instances	No. of Trips	Feasible Duties (FDs)	Objective value (Ext)	Driving Time (%)	CPU time (sec) (Ext)	Objective value (HCHSA)	CPU time (sec) (HCHSA)	RPD (%)
DM-01	25	6	2944	0.95	0.42	2944	0.0001	0
	45	11	5407	0.93	0.88	5407	0.0001	0
	65	16	7851	0.93	1.09	7851	0.0002	0
DM-02	70	23	11288	0.94	1.73	11288	0.0004	0
	90	28	13710	0.91	3.50	13710	0.0006	0
	105	33	16205	0.92	9.24	16205	0.0019	0
DM-03	110	35	17166	0.87	121.51	17166	0.0033	0
	115	36	17685	0.82	2262.04	17685	0.0085	0
	120	38	18601	0.80	15927.33	18601	0.0127	0

5.7.2 Simulated Annealing (SA) and Hybrid Constructive Heuristic Simulated Annealing (HCHSA)

The HCHSA algorithm was able to solve all small-sized problems with computational time less than one second. We used relative percentage deviation as a performance measure to further evaluate the obtained solutions for the small-sized problems. This was calculated by the following equation;

$$\text{RPD } (\%) = \{[S_{(Alg)} - S_{(Ext)}] / S_{(Ext)}\} \times 100\%$$

where $S_{(Alg)}$ is the objective value of the solution obtained by the HCHSA algorithm and $S_{(Ext)}$ is the objective value of the optimal solution given by the Xpress Optimizer.

The proposed algorithms were implemented in Microsoft Visual C# and run on an Intel Core 2 Duo 1.96 GHz Processor with 3.46 GB of RAM under Microsoft Windows XP operating system. The computational results obtained from both the CH and the HCHSA algorithms are summarised in Table 5.1 and Table 5.3, respectively. The number of trips per duty varies because of the length of a trip also varies. On average, the number of trips in each duty varies from 3 to 6 trips. The problem of smaller size corresponds to a higher percentage of driving time with less computational times. This is because smaller sized problems can be better optimised due to a more exhaustive search. For larger problems, long idle transition times remain high as indicated by a lower percentage of driving time. This is due to the fact that services at different times of the day have different frequency. Early morning and late afternoon hours have higher service frequency than that of the middle day. It seems that the more the trips we include the higher probability of the delay.

The driving time is used to measure the performance of the obtained schedules (productivity rate). Driving time of the crew is the ratio between the total working time (W*t*) and the total spread time or elapsed time (E*t*) in a duty. Excess cost (E*c*) was calculated as follows. $\text{E}c\ (\%) = \{(\text{E}t - \text{W}t) / \text{W}t\} \times 100\%$. Both the CH and the HCHSA algorithms produced acceptable solutions, although the produced solutions are not guaranteed to be an optimal solution. The CH algorithm was sometimes unable to include some trip segments and left them out unscheduled. In all cases, the HCHSA algorithm was able to produce better solutions than the CH in terms of the solution quality and the runtime. As can be seen from Table 5.3, the HCHSA algorithm significantly improves the solution produced by the CH. The HCHSA algorithm increases the average driving time by 3.06% and decreases the average excess cost by

3.35%. Furthermore, the number of leftovers in the HCHSA is smaller than that in the CH. Overall, the HCHSA algorithm increases the total crew working time and reduces the number of crew duties for all datasets. As the number of crew duties corresponds to the number of crew needed, significant savings can be gained on the annual cost of crew related expenses.

Table 5.3 Computational results of the Hybrid Constructive Heuristic and Simulated Annealing (HCHSA) algorithm.

Instance	# Trips	# Duties	Objective Value	Total Elapsed Time (min)	Driving Time (%)	Excess Cost (%)	CPU Time (sec)	Average Working Time (min)
DH-01	258	105	51345	54045	95.00	5.26	007	489.00
DH-02	350	117	58146	62526	92.99	7.53	014	496.97
DH-03	425	133	65825	72338	91.00	9.89	018	494.92
DH-04	455	142	71722	79693	90.00	11.11	032	505.08
DH-05	470	145	80762	87805	91.98	8.72	048	556.98
DH-06	485	157	78657	85476	92.02	8.67	055	501.00
DH-07	550	155	83120	92352	90.00	11.11	064	536.26
DH-08	582	162	81840	92007	88.95	12.42	093	505.19
DH-19	607	167	81679	93884	87.00	14.94	116	489.10
DH-10	625	172	84965	96404	88.13	13.46	130	493.98
DH-11	660	174	86371	97999	88.13	13.46	336	496.39
DH-12	732	178	88255	100285	88.00	13.63	582	495.81

To measure the quality of solutions obtained by the algorithms, the upper and lower bound values were calculated as follows. Q = (objective value – LB) / (UB – LB) where $0 \leq Q \leq 1$ (Burdett and Kozan, 2010). The equation describes approximately the quality of the solution in the search space. If Q value is close to zero then the obtained solution is near to the optimal solution. The Q values of solutions obtained by both the CH and HCHSA algorithms can be seen in the chart of Figure 5.5. The Q value shown in Figure 5.5 is enough to validate the quality of the proposed algorithms.

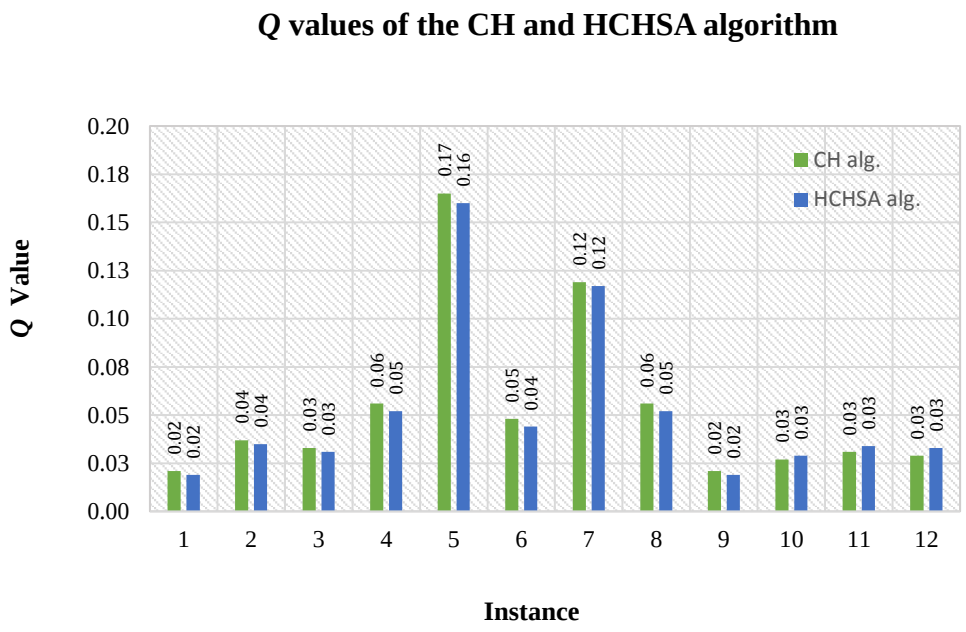

Figure 5.5 *Q* values of the CH and HCHSA solutions.

Figure 5.6 shows *Q* values of the SA solutions. The average *Q* values of all instances is 0.063 which indicates that the SA algorithm can produce good quality feasible solutions for all datasets.

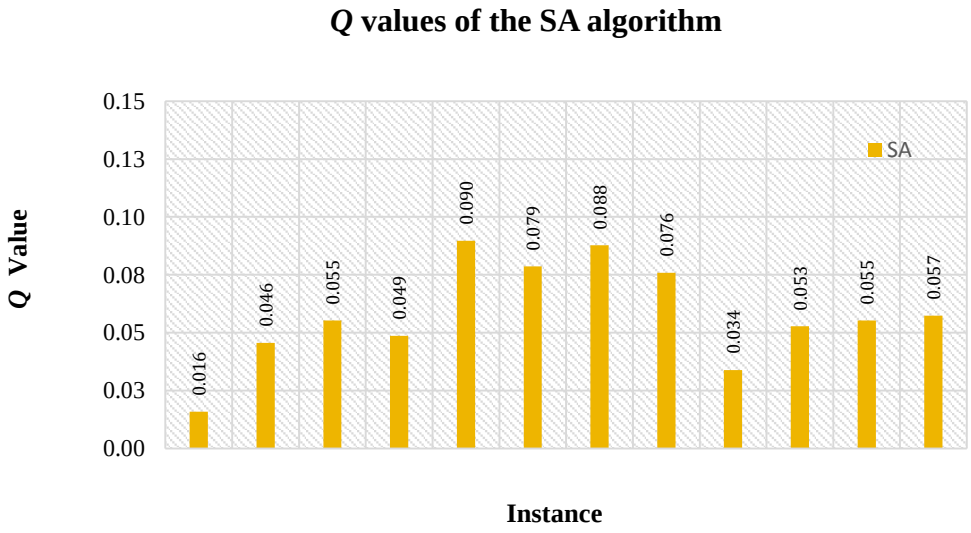

Figure 5.6 *Q* values of the SA solutions.

With regard to the SA algorithm, the initial temperature T_0 was set to be large to allow the search exploring some areas of the solution space with low level quality solutions hence, accepting a worse solution at the beginning of the search. A large neighbourhood is more attractive because it tends to find much better solution in one local search step than in the small neighbourhoods. However, a large neighbourhood is associated with long search process to come up with a better neighbouring solution. Large neighbourhood may also lead to a potential bottleneck for local search when searching a better neighbouring solution. The temperature was updated in each iteration by applying geometric cooling schedule where $T^c = \alpha T^c$. We applied a cooling factor of 0.93, 0.95, and 0.97 as the temperature reduction should be controlled by a constant cooling factor with the value approximately close to one (Kirkpatrick, 1983). Altering this parameter however, did not have a significant effect on the final objective value after 500 iterations. Computational results of SA algorithm are given in Table 5.4 below.

Table 5.4 Computational results of the Simulated Annealing (SA) algorithm.

Instance	# Trips	# Duties	Objective Value	Total Elapsed Time (min)	Driving Time (%)	Excess Cost (%)	CPU Time (sec)	Average Working Time (min)
DH-01	258	110	55172	59850	92.184	8.48	009	501.57
DH-02	350	119	61114	68355	89.407	11.85	020	513.56
DH-03	425	137	70717	79412	89.051	12.30	024	516.18
DH-04	455	155	77571	87633	88.517	12.97	039	500.45
DH-05	470	150	76987	86979	88.512	12.98	055	513.24
DH-06	485	152	78992	89273	88.484	13.01	062	519.68
DH-07	550	151	81167	91750	88.466	13.04	079	537.53
DH-08	582	165	82195	93585	87.830	13.86	105	498.15
DH-19	607	168	84531	96252	87.822	13.87	120	503.16
DH-10	625	171	84946	101361	83.805	19.32	141	496.76
DH-11	660	173	84355	101855	82.819	20.75	336	487.60
DH-12	732	180	86621	107197	80.805	23.75	585	481.23

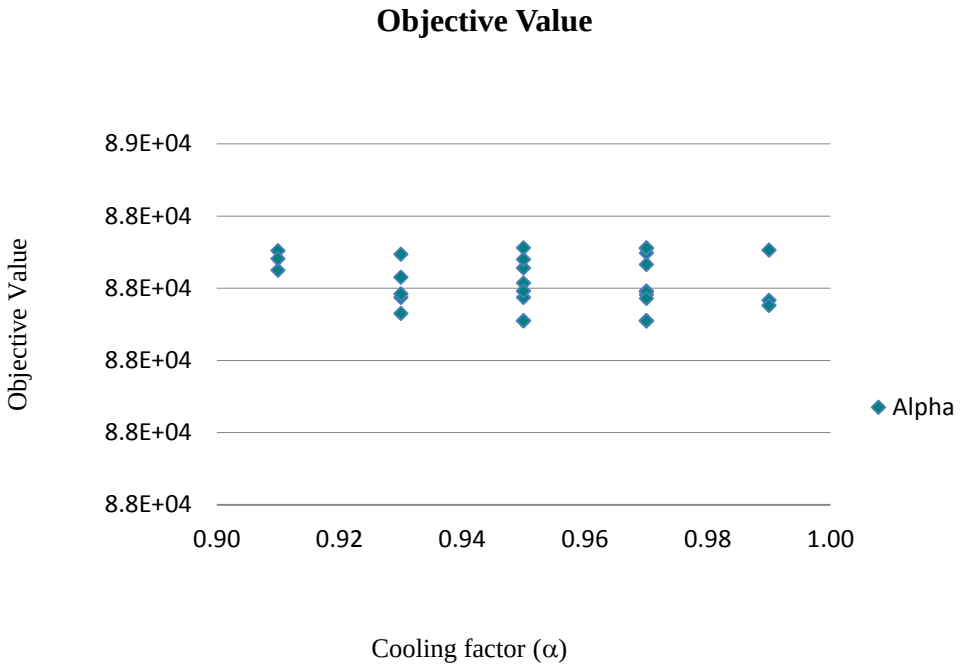

Figure 5.7 Objective values by varying cooling factor (α).

Sensitivity analysis was performed by changing each parameter while keeping the others fixed for multiple runs. Figure 5.7 shows the objective function values obtained by varying cooling factor of the SA. As can be seen from this figure, there is no significant effect of cooling factor on the improvement of the objective function value. It does show however, that on average, better results can be obtained with cooling factor (α) between 0.95 and 0.97.

5.7.3 Hybrid Constructive Heuristic and Tabu Search (HCHTS)

Constructive heuristics look for solutions in a reasonable computational time, even though there is no guarantee of reaching optimality. The same applies to the metaheuristics, by which a certain trade-off of local search and randomisation is applied. A random search provides a method of escaping local optima by moving away wandering other region in the solution space. In this way, metaheuristics become suitable for searching the global optimum.

A computational experiment was conducted to solve randomly generated benchmark instances with 24-h scheduling horizons. A feasible crew duty (shift) represents an accumulated total working time for the crew consisting of one or two partial duties, a period of MB, idle transition times, and the sign-on and sign-off

activities. Normal daily working time was equivalent to 8 h and maximum spread time allowed was set at 12 h. We used the same parameter settings with the one applied by CH and SA algorithm. The minimum and maximum lengths of working periods for the 1st part of a duty were set to 3 h and 5.5 h, respectively. The minimum and maximums lengths for the 2nd part of a duty were set to 2 h and 4.5 h, respectively. The length of the ROP is 2.5 h within which a minimum 0.5 h MB is required between the third and the sixth hours of the crew working period. There is a time allowance of about 10 min for sign-on or sign-off when a crew starts or ends his duty at a HD. The proposed mathematical model was solved by Xpress-Optimizer (FICO) algorithms for mixed integer problems to obtain exact solution and this solution was used as a reference for the TS metaheuristic approach. We considered 3 HDs and 5 RPs. The number of trips varied between 25 and 120 for instances solved by Xpress-Optimizer. For larger datasets in which Xpress-Optimizer was unable to find optimal solutions, the numbers of trips varied between 250 and 750 and were approximately solved by the proposed TS-based algorithm. Although Xpress-Optimizer was capable of finding optimal solutions for smaller problem instances, it was unable to solve considerably larger test instances examined in this study. We used relative percentage deviation as a performance measure to further evaluate the solutions for the small-sized problems. The relative percentage deviation indicates that an average gap between the optimal solutions obtained by Xpress Optimizer and the proposed algorithm for small-sized problems up to 200 trips is less than 3.5%.

The proposed algorithm was implemented in Microsoft Visual C# and computational experiments were executed on an Intel Core 2 Duo running at 1.96 GHz Processor with 3.46 GB of RAM. In this study, the multi depots CSP concerns constructing a set of crew roundtrips such that each crew roundtrip starts and ends at the same crew HD. Each trip is covered once and the total crew working time does not exceed the maximum allowable limit. It is desirable to construct feasible schedules that will minimise the time gap between trips (idle transition times) and maximise the length of the route per cycle time. A crew cycle time is the time spent to drive a round trip plus idle intervals while on a route. For this reason, frequent relieving should be avoided and a crew needs to work on the same train for as long as possible. Figure 5.8, Figure 5.9, and Figure 5.10 show a subset of train schedule on lines in the rail network.

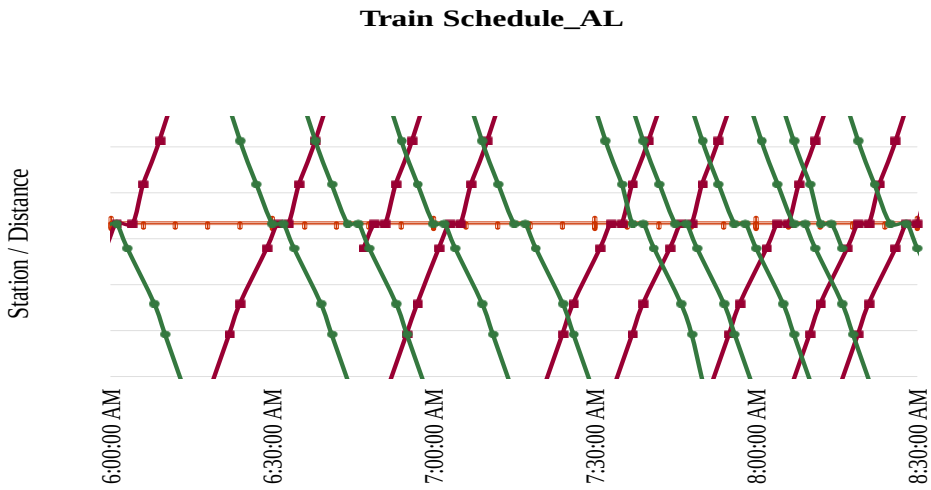

Figure 5.8 A subset of the train schedule on lines in the rail network.

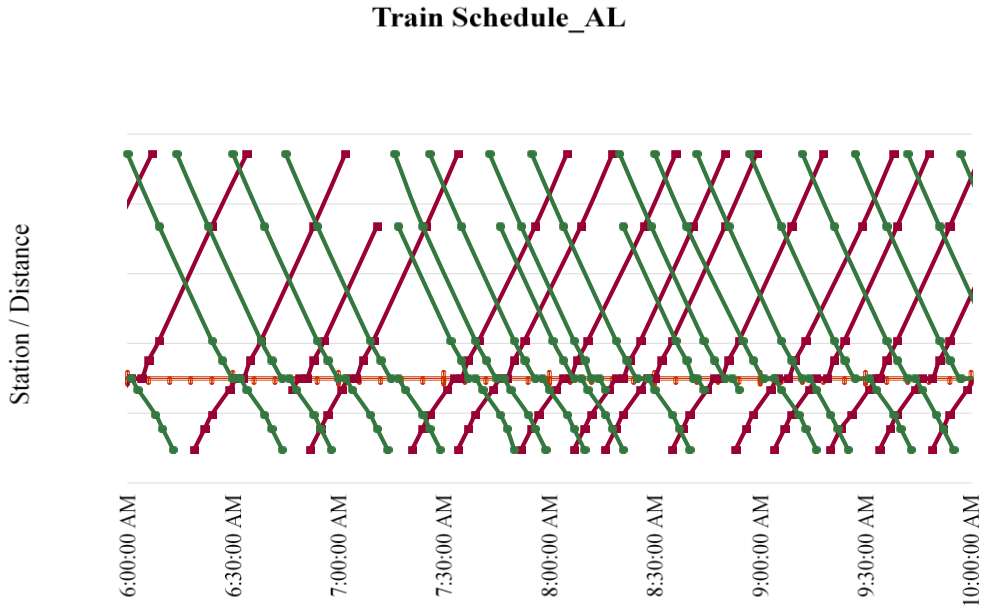

Figure 5.9 A subset of the train schedule on lines in the rail network.

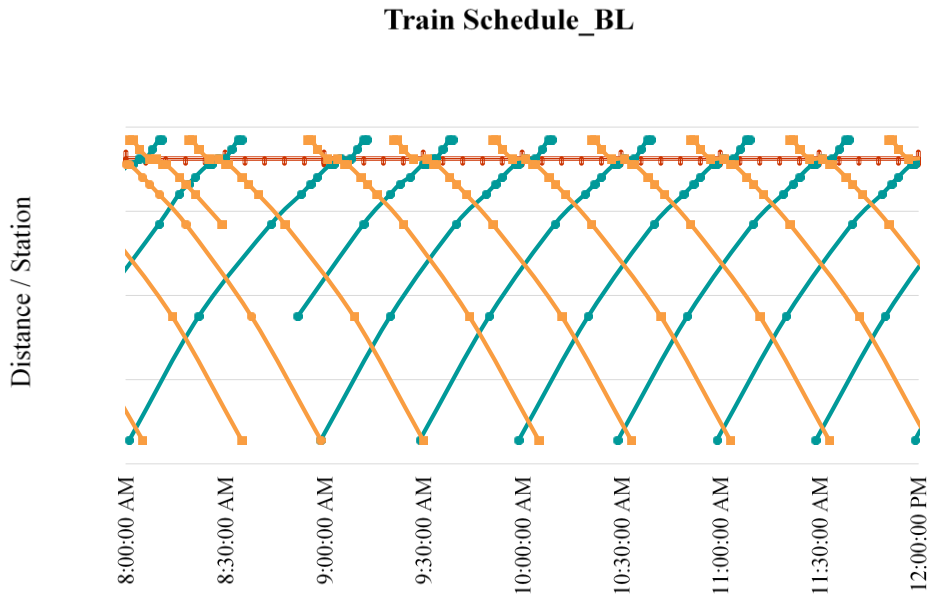

Figure 5.10 A subset of the train schedule on lines in the rail network.

Figure 5.11 shows an illustrative example of a train crew schedule. Each distinctly coloured line in the figure represents a crew roundtrip. As seen in the figure, the train crew TCr_001, for example, departs from a crew HD A at 05:00 and travels for 3 hours and 18 minutes, takes a MB for 39 minutes, from 08:18 to 08:57 at HD A, and terminates at 12:35 at the same crew HD A. The total length of the crew duty is 7 hours 35 minutes. The average duty period of all crew roundtrips is 8 hour 5 minutes. The time gap between the arrival of the previous trip and the departure of the next trip should not exceed the minimum transition time allowed.

Two partial duties are merged by connecting the selected RP locations of trip segments where a train crew takes a MB. It may not always be possible to combine two partial duties to form a crew duty with a time length of approximately equal to the regular working hours of 8 h. Therefore, a balance is needed by considering the crew's working hour guarantee and overtime. If eight hour is set as the working hour guarantee, then the crew will be paid for 8 h even if they work less than 8 h. While if the crew work time exceeds 8 h, they will be paid extra.

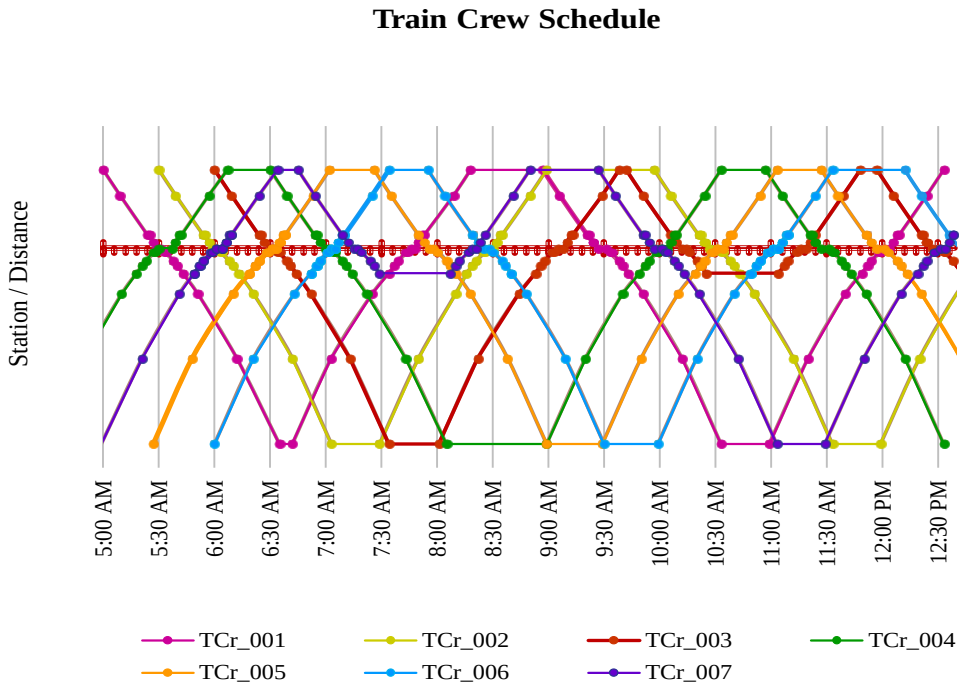

Figure 5.11 An example of generated crew roundtrips.

Each distinctly coloured line represents a crew roundtrip. As seen in the figure, the train crew TCr_001 departs from a crew HD A at 05:00 and travels for 3 hours and 18 minutes, takes a MB for 39 minutes, from 08:18 to 08:57 at HD A, and terminates at 12:35 at the same crew HD A. The total length of the crew duty is 7 hours 35 minutes.

TCr_002 departs from a crew HD A at 05:30 and travels for 3 hours and 29 minutes, takes a MB for 58 minutes, from 08:59 to 09:57 at HD A, and terminates at 13:48 at the same crew HD A. The total length of the crew duty is 8 hours 18 minutes.

TCr_003 departs from a crew HD A at 06:00 and travels for 4 hours and 25 minutes, takes a MB for 39 minutes, from 10:25 to 11:04 at away depot, and terminates at 13:33 at the same crew HD A. The total length of the crew duty is 7 hours 33 minutes.

TCr_004 departs from a crew HD B at 04:28 and travels for 3 hours and 37 minutes, takes a MB for 54 minutes, from 08:05 to 08:59 at HD B, and terminates at 12:33 at the same crew HD B. The total length of the crew duty is 8 hours 5 minutes.

TCr_005 departs from a crew HD B at 05:27 and travels for 3 hours and 32 minutes, takes a MB for 30 minutes, from 08:59 to 09:29 at HD B, and terminates at 13:43 at the same crew HD B. The total length of the crew duty is 8 hours 16 minutes.

TCr_006 departs from a crew HD B at 06:00 and travels for 5 hours and 33 minutes, takes a late MB for 39 minutes, from 11:33 to 12:12 at HD B, and terminates at 13:42 at the same crew HD B. The total length of the crew duty is 7 hours 42 minutes.

TCr_007 departs from a crew HD B at 04:59 and travels for 3 hours and 51 minutes, takes a late MB for 37 minutes, from 08:50 to 09:27 at away depot (HD A), and terminates at 14:03 at the same crew HD B. The total length of the crew duty is 9 hours 4 minutes. The average duty period of all crew roundtrips shown in Figure 5.12 is 8 hour 5 minutes.

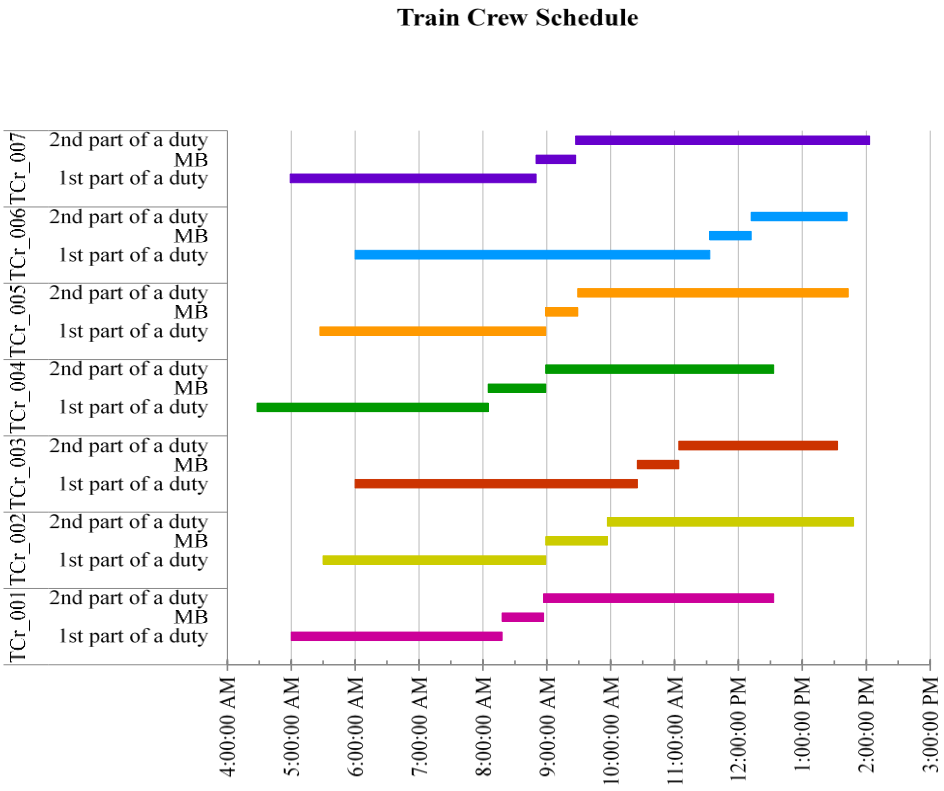

Figure 5.12 An example Gantt chart of the train crew schedule.

Table 5.5 shows the computational results, i.e. the number of crew roundtrips (duties), objective function values, the driving times, excess cost, and the run times. The performance of the obtained schedules (productivity rate) was measured by the driving time percentage. The driving time of the crew is the ratio between the total working time (W*t*) and the total spread time or elapsed time (E*t*) in a duty. Excess cost (E*c*) was calculated as follows. E*c* (%) = {(E*t* – W*t*) / W*t*} × 100%.

Table 5.5 Computational results of the Hybrid Constructive Heuristic and Tabu Search (HCHTS) algorithm.

Instance	# Trips	# Duties	Objective Value	Total Elapsed Time (min)	Driving Time (%)	Excess Cost (%)	CPU Time (sec)	Average Working Time (min)
DH-01	258	99	52256	55527	94.10	6.26	009	527.84
DH-02	350	115	62569	67861	92.20	8.46	019	544.08
DH-03	425	130	65697	73569	89.30	11.98	024	505.36
DH-04	455	140	67994	78244	86.90	15.07	038	485.67
DH-05	470	147	76185	85707	88.89	12.50	052	518.27
DH-06	485	155	80251	88191	91.00	9.89	061	517.75
DH-07	550	152	81667	91887	88.88	12.51	077	537.28
DH-08	582	165	82175	94802	86.68	15.37	108	498.03
DH-19	607	169	81804	94139	86.90	15.08	121	484.05
DH-10	625	171	85014	95521	89.00	12.36	142	497.16
DH-11	660	172	83032	97011	85.59	16.84	340	482.74
DH-12	732	175	86430	100711	85.82	16.52	590	493.89

Average crew working time for small-sized instances is 515.74 min, medium-sized instances is 517.83 min, and large-sized instances is 489.46 min. It seems that for small- to medium-sized instances, the average crew working time is slightly higher than the average crew working time for large-sized problems (Figure 5.13). This is because long idle transition times occur when more trips were included as a result of congestions. This observation is also confirmed by a lower driving time percentage of large-sized problems compared to the driving times percentage of small-sized to medium-sized instances. Driving time percentage for small-sized instance, medium-sized instance and large-sized instances are 90.63, 88.88, and 86.83, respectively.

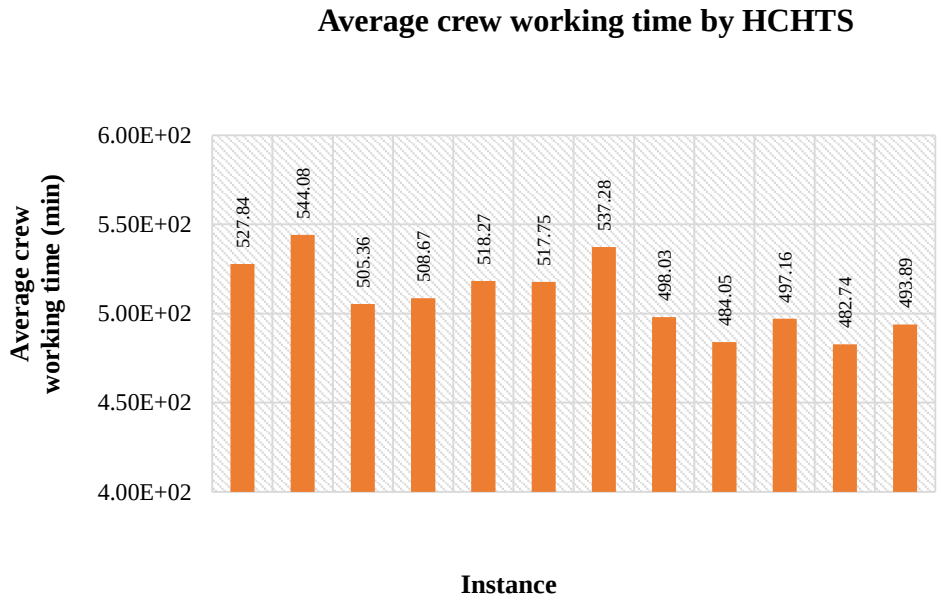

Figure 5.13 Average crew working time of all datasets.

The performance of HCHTS algorithm was evaluated by measuring the average computational time and the best solution found. The overall solution of all datasets can be obtained in less than 10 min of computational time. The algorithm is quite robust as indicated by the small coefficient of variation from average. The length of tabu list seems contributes an important effect in the search process. With the increasing number of iterations, the solution is improved. The number of sequential iterations without improved objective function value is used as the termination criterion. The best found solution was compared with the upper and lower bound values to measure the quality of solutions obtained by the algorithm. This was calculated as follows. Q = (objective value – LB) / (UB – LB) where $0 \leq Q \leq 1$ (Burdett and Kozan, 2010). The Q value approximately indicates the quality of the solution in the search space. The obtained solution is near to the optimal when the Q value is close to zero. The Q value is used to validate the quality of solutions obtained by the proposed algorithm. The Q values of solutions obtained by the proposed algorithm can be seen in the chart of Figure 5.14. On average, the Q values of solutions by HCHTS algorithm is 0.068.

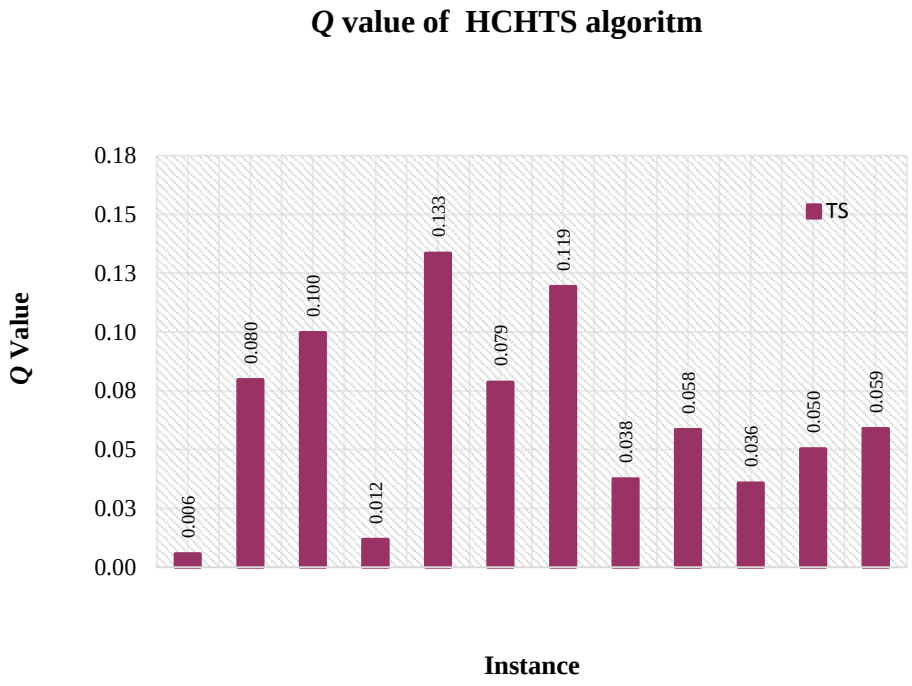

Figure 5.14 *Q* values of the HCHTS algorithm.

5.7.4 Hybrid Constraint Programming and Simulated Annealing (HCPSA)

Computational experiments were conducted based on the same generated benchmark instances to make the performance of each approach are comparable. Computational results of the hybrid CP and SA are presented in Table 5.6. Overall results indicate that the proposed approach yields good acceptable solution. The average *Q* values of solutions by HCPSA algorithm shown in Figure 5.15 is 0.067.

Table 5.6 Computational results of the Hybrid Constraint Programming and Simulated Annealing (HCPSA) algorithm.

Instance	# Trips	# Duties	Objective Value	Total Elapsed Time (min)	Driving Time (%)	Excess Cost (%)	CPU Time (sec)	Average Working Time (min)
DH-01	258	107	52275	55619	93.99	6.40	007	488.55
DH-02	350	114	62540	67146	93.14	7.37	016	548.60
DH-03	425	128	65478	73146	89.52	11.71	022	511.55
DH-04	455	138	67966	77826	87.33	14.51	037	492.51
DH-05	470	146	76177	85686	88.90	12.48	051	521.76
DH-06	485	154	80248	87683	91.52	9.27	060	521.09
DH-07	550	152	81667	91887	88.88	12.51	075	537.28
DH-08	582	165	82175	94802	86.68	15.37	108	498.03
DH-19	607	169	81804	94139	86.90	15.08	121	484.05
DH-10	625	171	85014	95521	89.00	12.36	142	497.16
DH-11	660	172	83032	97011	85.59	16.84	340	482.74
DH-12	732	175	86433	100711	85.82	16.52	590	493.89

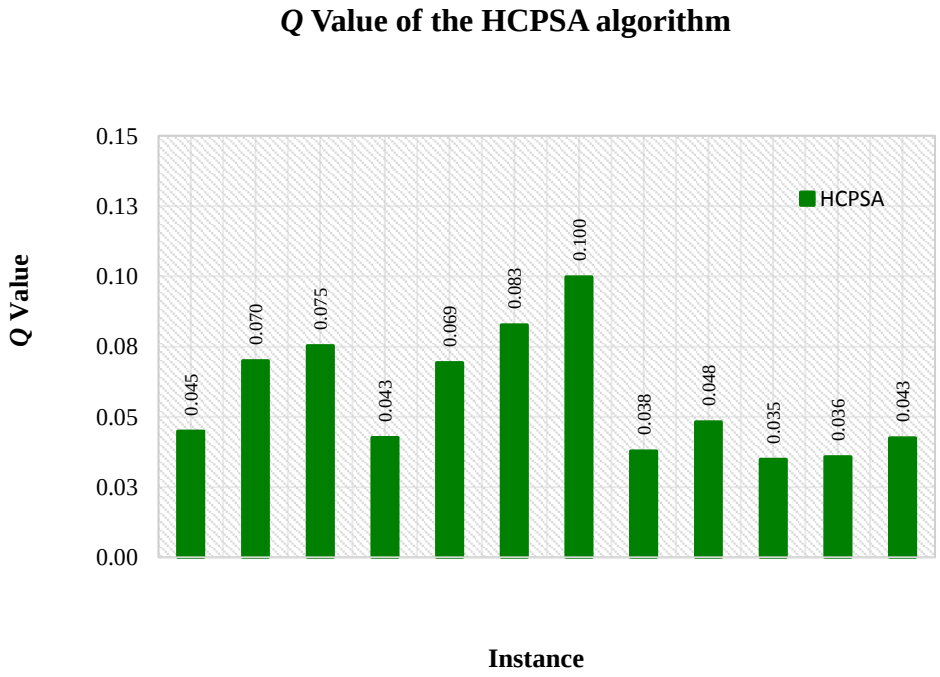

Figure 5.15 *Q* values of the HCPSA solutions.

5.7.5 Hybrid Tabu Search and Simulated Annealing (HTSSA)

The performance of HTSSA algorithm was evaluated by measuring the average computational time and the best solution found. The performance of the HTSSA algorithm was tested ten times on each randomly generated datasets. The parameters examined for TS algorithm were the tabu list size and the number of iterations. The obtained results indicate that smaller tabu list sizes more likely to get trapped and find it difficult to escape from a certain point in the solution space. On the other hand, larger tabu list sizes seem do not give a thorough search for obtaining improved neighbourhood solution. The overall solution of all datasets can be obtained in less than 10 min of computational time. The algorithm is quite robust as indicated by the small coefficient of variation from average. The length of tabu list seems contributes an important effect in the search process.

Table 5.7 Computational results of the Hybrid Tabu Search and Simulated Annealing (HTSSA) algorithm.

Instance	# Trips	# Duties	Objective Value	Total Elapsed Time (min)	Driving Time (%)	Excess Cost (%)	CPU Time (sec)	Average Working Time (min)
DH-01	258	97	51703	54742	94.45	5.88	007	533.02
DH-02	350	113	61880	67103	92.22	8.44	013	547.61
DH-03	425	130	65337	73152	89.32	11.96	022	502.59
DH-04	455	138	67533	77486	87.15	14.74	034	489.37
DH-05	470	145	75208	84638	88.86	12.54	052	518.68
DH-06	485	155	80106	88044	90.98	9.91	060	516.81
DH-07	550	151	81450	91605	88.91	12.47	073	539.40
DH-08	582	165	82136	94761	86.68	15.37	110	497.79
DH-19	607	167	80832	93045	86.87	15.11	125	484.02
DH-10	625	168	83523	93792	89.05	12.29	141	497.16
DH-11	660	171	82549	96426	85.61	16.81	336	482.74
DH-12	732	173	85442	99524	85.85	16.48	587	493.88

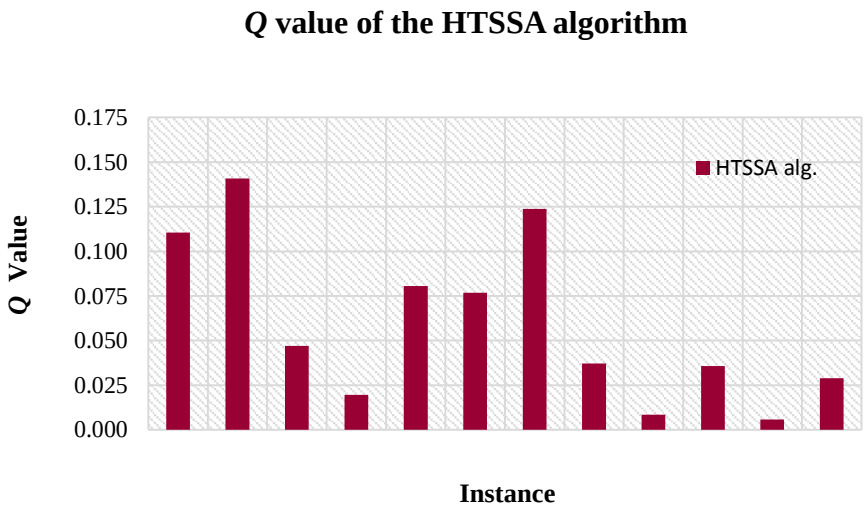

Figure 5.16 *Q* values of the HTSSA solutions.

An aggregation procedure has a significant effect in reducing the problem size such that the proposed algorithm is able to handle large-sized railway CSP and solve it within an acceptable computational time. The neighbourhood structure also contributes to the effectiveness of the search process. *Q* values of the HTSSA solutions indicate that the algorithm can obtain good quality solution for all datasets.

5.8 CONVERGENCE ANALYSIS OF THE PROPOSED HEURISTICS / METAHEURISTICS

This section investigates the convergence of the heuristics and metaheuristics applied in this study. A stopping criterion should be defined to terminate the algorithm. The stopping criterion can be a maximum number of iterations, such as after 1000 iterations. The iteration limits can be determined in an experimental way and it usually depends on the size and structure of the problem. Another termination criterion is a maximum allowed execution time, such as after 5 minutes computational time. A good trade-off between setting computational time early or late to stop should be applied, because if the algorithm terminates too late, it wastes the computational time. Conversely, if the algorithm terminates too early, the optimum solution may not be reached yet. Additionally, the number of sequential iterations without improvement in the objective function value can be used as a termination criterion. This criterion is more frequently used and it was also used as a stopping criterion to solve the problem under study. Standard deviation was also calculated to know how much variation exists from the average. High standard deviation indicates that the data points are spread out over a large range of values. Low standard deviation indicates that the data points tend to be very close to the mean.

Figure 5.17 shows a line chart of the objective function values vs the number of iterations of the local search heuristic and metaheuristics. This is a two dimensional graph in which the number of iterations is plotted on the X-axis and the objection function value is plotted on the Y-axis.

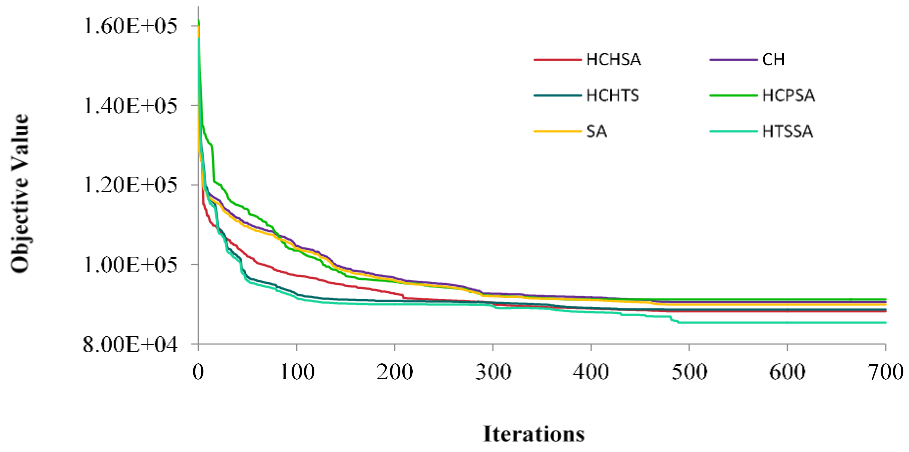

Figure 5.17 Objective value improvement with the number of iterations.

As can be seen from this figure, in general, the algorithms improved the solutions with the increasing number of iterations and they significantly improved the obtained solutions within the first 200 iterations. The minimisation of the objective function values then tends to be stagnant at about 500 iterations and thereafter the solutions show no sign of advancement and algorithms indicate a tendency of convergence. The number of sequential iterations without improved objective function value was used as the termination criterion. To prove convergence of the proposed algorithm, standard deviation was also calculated to shows the extent of variation from the average. It is observed that the standard deviation tends to decrease as the number of iterations $i \rightarrow \infty$, which means that the data points are getting closer to the mean as the number of iterations increases. It can be noted that in general the hybrid TS and SA algorithm (HTSSA) outperforms all other heuristics/metaheuristics shown in the figure. All algorithms give a similar trend. Overall, the HTSSA gives the best result with average crew working time is 8.477 h.

5.9 CONCLUSION

Heuristics and metaheuristics-based algorithms as well as hybrid heuristics/metaheuristics to solve railway CSP have been presented in this chapter. The objective of the algorithms is to find minimum time crew roundtrips by minimising idle interval between trips and an idle transition between partial duties. Computational

results indicated that the proposed algorithms are able to generate feasible near-optimal solutions within an acceptable computational time, as indicated by the average Q values of all methods which are fairly close to zero. An aggregation procedure inserted in the proposed algorithms has a significant effect in reducing the problem size such that the algorithms are able to handle large-sized railway CSP and solve it within a reasonable computational time. The neighbourhood structure also contributes to the effectiveness of the search process. The optimisation approaches incorporate a complex set of railway crew scheduling constraints and can be easily adjusted to include additional constraints. A specific constraint in which a crew can be relieved during the interval of ROP gives more flexibility on crew schedule. In that way, the proposed methods will enable crew members to flexibly choose the relief time and location in their duties. The proposed algorithms produced good acceptable solutions and can solve medium- to large-sized instances under study. The proposed optimisation methods for solving railway CSP can be improved to develop a decision support system to solve real-world railway CSP. Furthermore, the model and algorithms proposed in this study can be extended to the integration of vehicle and CSPs with ROP.

Chapter 6: **Conclusions and Recommendations**

This study has developed models and algorithms to solve the problem. The need for studying railway CSP has been highlighted in chapter 1. The general research methodology has also been explained in chapter 1. A survey on the existing modelling and solution approaches for CSP has indicated that there is limited study on CSP, particularly on the railway CSP. The study on CSP has relied mainly on the existing models such as the SCP and SPP formulations and then solves the problem analytically or approximately, as indicated in chapter 2. This suggests the need for improving the existing models by integrating real-life crew scheduling constraints into a developed model. The railway CSP models developed in this study include a specific crew scheduling constraint in the models such that they enable the crew to be relieved during the ROP. Integrating this specific constraint in the proposed models will enhance the robustness of the schedule and provide a better representation of real railway crew scheduling conditions. The detailed formulation of the MP-based model and the CP-based model with its solution technique are presented in chapter 3 and chapter 4, respectively. Chapter 5, presents metaheuristics techniques to solve railway CSP. Constructive heuristics, hybrid SA- and TS-based algorithms were designed to improve both the solutions and the computational performances. This chapter also includes the computational results and analysis of the proposed algorithms. Subsequently, the chapter 6 provides the conclusions and recommendations for further study.

6.1 CONCLUSIONS

In this thesis, models and algorithms for railway CSP have been presented. Railway CSP is the process of allocating train services to the crew duties based on the published train timetable while satisfying operational and contractual requirements. The problem is restricted by many constraints and it belongs to the class of NP-hard. The objective of the models and algorithms is to minimise the number of crew duties by minimising total idle transition times. The idle transition times includes idle intervals between trips and an idle interval between partial duties. These unproductive

parts of a crew duty (shift) contribute the most to the optimisation potential of the crew scheduling. The mathematical model includes the interval of relief opportunities, allowing a train crew to be relieved at any RP during the ROP. Small-sized to medium-sized instances can be solved to optimality analytically by applying the developed mathematical model. CP based approach to solve railway crew scheduling is able to obtain best solutions through the DFS and BFS methods. In a few cases, both the DFS and BFS failed to reach optimality. However, the relative deviation is very small and thereby can still be considered as acceptable solutions. CP formulation is more natural in representing the problem and requires much fewer variables and constraints than the MIP-based methodologies. This is due to the global constraint that capable of representing complex relationships between variables which in turn provides effective domain reduction. Using CP technique, the model provides acceptable results in all test datasets. However, the problem in its entirety is complex because of the presence of conflicting constraints. The overall results indicate that the CP model can produce feasible railway crew schedules of small- to medium-sized instances within a reasonable computational time.

Railway CSP is mathematically intractable due to the number of possible trip combinations and the complexity of the involved constraints. To handle the difficulty due to the combinatorial explosion of the problem, hybrid constructive heuristics and metaheuristics are proposed to solve the problem. The overall results indicate that the proposed algorithms can produce near-optimal railway crew schedules of large-sized datasets within an acceptable computational time. This study also shows the effectiveness of the hybridization of local search constructive heuristics and metaheuristics in solving a highly constrained combinatorial optimisation problem.

HCHTS algorithm has been presented in this thesis to solve multi depots railway CSP. The objective of the algorithm is to find minimum time crew roundtrips by minimising idle interval between trips and an idle transition between partial duties. Small-sized instances are solved to optimality by applying the proposed mathematical model. The obtained results are then compared against the solution produced by the HCHTS algorithm which was composed of a three-phase heuristic. Computational results indicated that the proposed algorithm is able to generate near-optimal feasible solutions within an acceptable computational time, as indicated by the average Q values which is fairly close to zero. An aggregation procedure has a significant effect

in reducing the problem size such that the proposed algorithm is able to handle large-sized railway CSP and solve it within an acceptable computational time. The neighbourhood structure also contributes to the effectiveness of the search process. With regard to the HCPSA and HTSSA algorithms, the results obtained indicate that the proposed algorithms can obtain good acceptable solutions within a reasonable computational time. Overall, the HTSSA algorithm gives the best objective function value with less computational times.

The developed optimisation models and algorithms involve a complex set of railway crew scheduling constraints and they can be easily adapted to include additional constraints. The models and algorithms incorporate a specific constraint in which a crew can be relieved during the interval of relief opportunities period. In that way, the proposed models and algorithms will enable crews to flexibly choose the relief time and location in their duties. Although the models and algorithms are presented in the context of railway CSP, they are general and flexible enough to be adapted to different locations and modes of transportation. In addition, the proposed models can be solved using a wide range of techniques.

6.2 RECOMMENDATIONS FOR FURTHER RESEARCH

This study has raised a number of issues which might be interesting topics for further work. These are summarised as follows.

- **Investigation of railway crew scheduling models.** In this research, two alternative models based on MP and CP for railway crew scheduling have been developed. It would be valuable for further investigation by analysing and comparing their performance. Further research to solve these models should focus on improving search techniques and integrating these models with other solution methods to combine the strength of the techniques. Although we have developed models for railway CSP, these models can be applied to other modes of transportation. The proposed models deal with the construction of duties (shifts) with one period of ROs (straight runs). These models can also be easily extended to model a situation in which a duty may contain more than two pieces of work (split runs). Furthermore, the models can be applied to the integration of vehicle and crew scheduling problems with ROP.

The complexity of railway CSP might hinder in capturing all details, thereby it might be interesting to develop a simulation model to analyse the real operations and evaluate the effect of constraints on the performance of the crew schedules. A simulation model can measure the impact of crew utilisation under different scenarios, for example, by varying cost parameters such as crew operating cost, deadhead cost, and train operating cost.

- **Investigation of metaheuristics.** SA and TS-based algorithm have been applied to solve railway CSP in this research. Local search heuristics for generating initial solutions for the SA and TS are worthwhile for further investigation. It might also be interesting to analyse the performance of the SA and TS using different neighbourhood structures. Even though there have been numerous studies conducted to determine the optimum parameter setting of SA and TS applications, more experiments are needed to identify suitable parameters of SA and TS to solve crew scheduling related problems.

  Other metaheuristics that are suitable for solving combinatorial optimisation problems should be further applied on the crew scheduling related problems. These metaheuristics include genetic algorithms, ant colony optimisations, particle swarm optimisations, and bee algorithms.

  There are many aspects of railway CSP that need to be incorporated in future research such as train delay, deadheading, and crew balance at each HD. Moreover, it would be interesting to consider the dynamic crew scheduling (Huisman and Wagelmans, 2006), as this challenging problem might provide better solutions than the static one. Although a large body of research has been made in the area of generic CSP, there are still many aspects of the CSP that need to be further studied and investigated. Models and algorithms which are able to incorporate many aspects of the challenging practical situations and advanced solution techniques are inevitable.

Because many interesting issues have not been investigated, the findings obtained in this study should provide a reference for further work.

References

Aarts, E and Lenstra, J K. (2003). Local Search in Combinatorial Optimization, Princeton.

Abbink, E., Fischetti, M., Kroon, L.G., Timmer, G., & Vromans, M.J.C.M. (2004). Reinventing crew scheduling at Netherlands Railways. *Tech. rept. ERS-2004-046-LIS. Erasmus Research Institute of Management*, Erasmus University Rotterdam, the Netherlands.

Abbink, E., Van't Wout, J., Huisman, D. (2007). Solving Large Scale Crew Scheduling Problems by using Iterative Partitioning. *Schloss Dagstuhl - Leibniz-Zentrum für Informatik, External Workshops. ATMOS 2007 - 7th Workshop on Algorithmic Methods and Models for Optimization of Railways*.

Alfieri A, Kroon L, Velde S. (2007). Personnel scheduling in a complex logistic system: a railway application case. *Journal of Intelligent Manufacturing, 18*(2), 223–232.

Arabeyre, J., J. Fearnley, F. C. Steiger, and W. Teather. (1969). The Airline Crew Scheduling Problem: A Survey. *Transportation Science, 3,* 140–163.

Azadeh A, Farahani M H, Eivazy H, Nazari-Shirkouhi S, Asadipour G. (2013). A hybrid meta-heuristic algorithm for optimization of crew scheduling. *Applied Soft Computing, 13*(1), 158–164.

Bangert, Patrick (2012). *Optimization for Industrial Problems*. Springer – Verlag, Berlin Heidelberg.

Barnhart, C., L. Hatay, et al. (1995). Deadhead Selection for the Long-Haul Crew Pairing Problem. *Operations Research*, *43*(3): 491-499.

Barták, R., Salido, M., Rossi, F. (2010). Constraint satisfaction techniques in planning and scheduling. *Journal of Intelligent Manufacturing* 21, 5–15.

Bartodziej P, Derigs U, Malcherek D, Vogel, U. (2009). Models and algorithms for solving combined vehicle and crew scheduling problems with rest constraints: an application to road feeder service planning in air cargo transportation. *OR Spectrum, 31*(2), 405–429.

Beasley J. E. and B. Cao. (1998). A Dynamic Programming Based Algorithm for Crew Scheduling. *Computers and Operations Research, 25*(7-8), 567–582.

Beasley, J. E. and B. Cao. (1996). A tree search algorithm for the crew scheduling problem. *European Journal of Operational Research* 94(3) 517-526.

Beck J C, Feng T K, Watson J P. (2011). Combining Constraint Programming and Local Search for Job-Shop Scheduling. *INFORMS Journal on Computing, 23*(1), 1–14.

Bengtsson L, Galia R, Gustafsson T, Hjorring C, Kohl N. (*2007*). Railway Crew Pairing Optimization. Algorithmic Methods for Railway Optimization. *Lecture Notes in Computer Science (LNCS), 4359*, 126–144. Springer.

Benhamou, Frédéric., Jussien, Narendra., O'Sullivan, Barry A. (2010). Trends in Constraint Programming. Wiley-ISTE.

Bianco, L., M. Bielli, et al. (1992). A heuristic procedure for the crew rostering problem. *European Journal of Operational Research, 58*(2): 272-283.

Bodin, L., B. Golden, A.A. Assad, and M. Ball. (1983). Routing and scheduling of vehicles and crews: The state of the art. *Computer and Operations Research*, *10*, 63-211.

Boschetti M A, Mingozzi A, Ricciardelli S. (2004). An exact algorithm for the simplified multiple depot crew scheduling problem. *Annals of Operations Research, 127*(1-4), 177–201.

Burdett R L, and Kozan E. (2010). A disjunctive graph model and framework for constructing new train schedules. *European Journal of Operational Research, 200*(1), 85–98.

Cabrera, G. and Rubio L. J. M. (2009). Hybrid Algorithm of Tabu Search and Integer Programming for the Railway Crew Scheduling Problem. *Second Asia-Pacific Conference on Computational Intelligence and Industrial Applications*.

Caprara A, Fischetti M, Toth P, Vigo D, and Guida P. (1997). Algorithms for Railway Crew Management. *Mathematical Programming, 79*(1–3), 125–141.

Caprara, A., L. Kroon, et al. (2007). Chapter 3 Passenger Railway Optimization. *Handbooks in Operations Research and Management Science*, Elsevier. Volume 14: 129-187.

Caprara, A., M. Fischetti, P. Guida, P. Toth, and D. Vigo. (1999). Solution of Large-Scale Railway Crew Planning Problems: The Italian Experience. In N. Wilson (ed.), Computer-Aided Transit Scheduling, *Lecture Notes in Economics and Mathematical Systems*, Vol. 430, pp. 1–18. Berlin: Springer.

Caprara, A., M. Monaci, and P. Toth. (2001). A Global Method for Crew Planning in Railway Applications. In S. Voss and J. Daduna (eds.), *Computer-Aided Scheduling of Public Transport, Lecture Notes in Economics and Mathematical Systems*, *505*, 17–36. Springer.

Caprara, A., P. Toth, et al. (1998). Modeling and Solving the Crew Rostering Problem. *Operations Research*, *46*(6): 820-830.

Caserta, M and Voß, S. (2010). Metaheuristics: Intelligent Problem Solving. *Matheuristics, Annals of Information Systems,* 10, 1-38.

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

Chew K L, Pang J, Liu Q Z, Ou J H, Teo C P. (2001). An optimization based approach to the train operator scheduling problem at Singapore MRT. *Annals of Operations Research*, *108*(1-4), 111–122.

Chu SCK, Chan ECH. (1998). Crew scheduling of light rail transit in Hong Kong: From modeling to implementation. *Computers and Operations Research*, *25*(11), 887–894.

Chu SCK. (2007). Generating, scheduling and rostering of shift crew-duties: Applications at the Hong Kong International Airport. *European Journal of Operational Research, 177*(3), 1764–1778.

Chu, H. D., E. Gelman, et al. (1997). Solving large scale crew scheduling problems. *European Journal of Operational Research, 97*(2), 260-268.

Corry P, and Kozan E. (2008). Optimised loading patterns for intermodal trains. *OR Spectrum*, *30*(4), 721–750.

Claessens, M. T., N. M. van Dijk, et al. (1998). Cost optimal allocation of rail passenger lines. *European Journal of Operational Research*, *110*(3), 474-489.

Clement, R. and A. Wren. (1995). Greedy Genetic Algorithms, Optimizing Mutations and Bus Driver Scheduling. *Computer-Aided Transit Scheduling, Lecture Notes in Economics and Mathematical Systems*, 430, 213–235.

Crawford, B., C. Castro, et al. (2006). A Constructive Hybrid Algorithm for Crew Pairing Optimization. *Artificial Intelligence*: *Methodology, Systems, and Applications*. J. Euzenat and J. Domingue, Springer Berlin / Heidelberg. 4183, 45-55.

De Leone R, Festa P, Marchitto E. (2011). A Bus Driver Scheduling Problem: a new mathematical model and a GRASP approximate solution. *Journal of Heuristics, 17*(4), 441–466.

Deng G F, Lin W T. (2011). Ant colony optimization-based algorithm for airline crew scheduling problem. *Expert Systems with Applications, 38*(5), 5787–5793.

Desaulniers, G., J. Desrosiers, et al. (1997). Crew pairing at Air France. *European Journal of Operational Research, 97*(2): 245-259.

Desrochers, M. and F. Soumis. (1989). A Column Generation Approach to the Urban Transit Crew Scheduling Problem. *Transportation Science, 23*(1), 1–13.

Diana C. Flórez, Jose L. Walteros, Miguel A. Vargas, Andrés L. Medaglia, Nubia Velasco (2009). A Mathematical Programming Approach to Airline Crew Pairing Optimization. Centro para la Optimización y Probabilidad Aplicada (COPA). Departamento de Ingeniería Industrial, Universidad de los Andes.

Dias T G, de Sousa J P, and Cunha J F. (2002). Genetic Algorithms for the Bus Driver Scheduling Problem: A Case Study. *Journal of the Operational Research Society, 53*(3), 324–335.

Ehrgott, M and Gandibleux, X. (2008). Hybrid Metaheuristics for Multi-objective Combinatorial Optimization. *Hybrid Metaheuristics, Studies in Computational Intelligence*, 114, 221-259.

Elizondo R, Parada V, Pradenas L, and Artigues C. (2010). An evolutionary and constructive approach to a crew scheduling problem in underground passenger transport. *Journal of Heuristics, 16*(4), 575–591.

Elmi A, Solimanpur M, Topaloglu S, and Elmi A. (2011). A simulated annealing algorithm for the job shop cell scheduling problem with intercellular moves and reentrant parts. *Computers and Industrial Engineering, 61*(1), 171–178.

Emden-Weinert T, and Proksch M. (1999). Best practice simulated annealing for the airline crew scheduling problem. *Journal of Heuristics, 5*(4), 419–436.

Ernst, A. T., H. Jiang, et al. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, *153*(1): 3–27.

Ernst, A. T., Jiang, H., Krishnamoorthy, M., Owens, B. and Sier, D. (2004). An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research*, *127*(1–4), 21–144.

Ernst, A., H. Jiang, M. Krishnamoorthy, H. Nott and D. Sier (1999). An optimisation approach to train crew rostering. *Proc. 15th National ASOR Conference (Gold Coast, Australia)*, 437–453.

Ernst, A., H. Jiang, M. Krishnamoorthy, H. Nott and D. Sier (2001). An Integrated Optimization Model for Train Crew Management. *Annals of Operations Research*, Volume 108, Numbers 1-4.

Fischetti M, Martello S, and Toth P. (1987). The Fixed Job Schedule Problem with Spread-Time Constraints. Operations Research, 35(6), 849–858.

Fischetti M, Martello S, and Toth P. (1989). The Fixed Job Schedule Problem with Working-Time Constraints. Operations Research, 37(3), 395–403.

Focacci F, Lodi A, Milano M. (2002). Mathematical Programming Techniques in Constraint Programming: A Short Overview. *Journal of Heuristics*, *8*(1), 7–17.

Freling R, Lentink R. M, and Wagelmans A.P.M. (2004). A Decision Support System for Crew Planning in Passenger Transportation Using a Flexible Branch-and-Price Algorithm. *Annals of Operations Research, 127*, 203–222.

Freling, R., R. Lentink, and M. Odijk. (2001). Scheduling Train Crews: A Case Study for the Dutch Railways. In S. Voss and J. Daduna (eds.), *Computer-Aided Scheduling of Public Transport, Lecture Notes in Economics and Mathematical Systems*, 505, 153–166. Springer.

Fuzhang, W., W. Haixing, et al. (2006). Modeling and Solving for Railway Crew Scheduling Problem. *The Sixth World Congress on Intelligent Control and Automation (WCICA)*.

Giachetti R E, Damodaran P, Mestry S, Prada C. (2013). Optimization-based decision support system for crew scheduling in the cruise industry. *Computers and Industrial Engineering*, *64*(1), 500–510.

Glover F. (1977). Heuristic for integer programming using surrogate constraints. *Decision Sciences*, *8*(1), 156–166.

Glover F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, *13*(5), 533–549.

Glover, F. and Greenberg, H.J. (1989). New approaches for heuristic search: a bilateral linkage with artificial intelligence. *European Journal of Operational Research*, 39, 119-130.

Gomes, M., L. Cavique, et al. (2006). The crew timetabling problem: An extension of the crew scheduling problem. *Annals of Operations Research*, *144*(1): 111-132.

Gopalakrishnan, B. and E. Johnson. (2005). Airline Crew Scheduling: State-of-the-Art. *Annals of Operations Research*, *140*(1), 305-337.

Goumopoulos C, and Housos, E. (2004). Efficient trip generation with a rule modeling system for crew scheduling problems. *Journal of Systems and Software*, *69*(1–2), 43–56.

Gualandi S, and Malucelli F. (2013). Constraint Programming-based Column Generation. *Annals of Operations Research*, *204*(1), 11–32.

Guillermo, C. G. and M. R. L. Jose (2009). Hybrid algorithm of Tabu Search and Integer Programming for the railway crew scheduling problem. *Asia-Pacific Conference on Computational Intelligence and Industrial Applications (PACIIA)*.

Gutjahr, W J. (2010). Convergence Analysis of Metaheuristics. *Matheuristics, Annals of Information Systems*, 10, 159-187.

Hanafi, R. (2000). Design of a Genetic Algorithm to Solve Facility Layout Problem. Master Thesis. *Swinburne University of Technology*. Melbourne, Australia.

Helena R L, Jose P P, and Portugal R. (2001). Multiobjective metaheuristics for the bus-driver scheduling problem. *Transportation Science, 35*(3), 331–343.

Hoffman, K. and M. Padberg. (1993). Solving Airline Crew Scheduling Problems by Branch-and-Cut. *Management Science*, *39*(6), 657–682.

Hooker, J. N. (2002). Logic, Optimization, and Constraint Programming. *INFORMS Journal on Computing* 14(4), 295–321.

Horowitz, E and Sahni, S. (1987). *Fundamentals of Data Structures in Pascal*, 2nd ed., Computer Science Press.

Huisman, D,. and Wagelmans, A. P.M. (2006). A solution approach for dynamic vehicle and crew scheduling. *European Journal of Operational Research, 172*(2), 453–471.

Irina Dumitrescu and Thomas Stützle. (2010). Usage of Exact Algorithms to Enhance Stochastic Local Search Algorithms. Matheuristics: Hybridizing Metaheuristics and Mathematical Programming. *Annals of Information Systems*, 103-134. Springer.

Johnson, E. L., G. L. Nemhauser, et al. (2000). Progress in Linear Programming-Based Algorithms for Integer Programming: An Exposition. *INFORMS Journal On Computing*, *12*(1): 2-23.

Kelbel J, and Hanzálek Z. (2011). Solving production scheduling with earliness/tardiness penalties by constraint programming. *Journal of Intelligent Manufacturing*, *22*(4), 553–562.

Kim H-J, and Hooker J N. (2002). Solving Fixed-Charge Network Flow Problems with a Hybrid Optimization and Constraint Programming Approach. *Annals of Operations Research*, *115*(1-4), 95–124.

Kirkpatrick S, Gelatt C. D, and Vecchi M.P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.

Kornilakis, H. and P. Stamatopoulos (2002). Crew Pairing Optimization with Genetic Algorithms. *Methods and Applications of Artificial Intelligence*. I. Vlahavas and C. Spyropoulos, Springer Berlin / Heidelberg. 2308: 752-752.

Kozan E, and Casey B. (2007). Alternative algorithms for the optimization of a simulation model of a multimodal container terminal. *Journal of the Operational Research Society*, *58*(9), 1203–1213.

Kreher, D L and Stinson D R. (1999). *Combinatorial algorithms: generation, enumeration, and search*. Boca Raton, Fla., London: CRC Press.

Kroon L, and Fischetti M. (2001). Crew scheduling for Netherlands Railways Destination: Customer. S. Voß, J. R. Daduna, eds. Computer-Aided Scheduling of Public Transport. *Lecture Notes in Economics and Mathematical Systems*, *505*, 181–201, Springer, Berlin.

Kwan R. (2010). Case studies of successful train crew scheduling optimisation. *Journal of Scheduling*, *14*(5), 423–434.

Kwan, A., R. Kwan, and A. Wren. (1999). Driver Scheduling Using Genetic Algorithms with Embedded Combinatorial Traits. Computer-Aided Scheduling of Public Transport, *Lecture Notes in Economics and Mathematical Systems, 421*, 81-102.

Kwan, R. and A. Wren. (1996). Hybrid Genetic Algorithms for Bus Driver Scheduling. *Advanced Methods in Transportation Analysis*, 609-619.

Laplagne I, Kwan RSK, Kwan ASK. (2009). Critical time windowed train driver relief opportunities. *Public Transport, 1*(1), 73–85.

Laurent B, and Hao Jin-Kao. (2007). Simultaneous vehicle and driver scheduling: A case study in a limousine rental company. *Computers and Industrial Engineering, 53*(3), 542–558.

Lee, C. and Chen, C. (2003). Scheduling of Train Driver for Taiwan Railway Administration. *Journal of the Eastern Asia Society for Transportation Studies*, Vol.5.

Levine, D. (1996). Application of a Hybrid Genetic Algorithm to Airline Crew Scheduling. *Computers and Operations Research, 23*(6), 547–558.

Lezaun M., G. Perez, et al. (2007). Rostering in a rail passenger carrier. *Journal of Scheduling 10*(4-5), 245–254.

Lezaun, M., G. Perez, et al. (2010). Staff rostering for the station personnel of a railway company. *Journal of the Operational Research Society, 61*(7): 1104–1111.

Li J. (2005). A Self-Adjusting Algorithm for Driver Scheduling. *Journal of Heuristics, 11*(4), 351–367.

Lučic P and Teodorovic D. (1999). Simulated annealing for the multi-objective aircrew rostering problem, *Transportation Research Part A, 33*(1), 19–45.

Lustig and Puget. (2001). Program does not equal program. *Interfaces* 31, 29–53.

Marsten, R. and F. Shepardson. (1981). Exact Solution of Crew Scheduling Problems Using the Set Partitioning Model: Recent Successful Applications. *Networks, 11*, 165–177.

McAloon K, and Tretkoff C. (1997). Logic, modeling, and programming. *Annals of Operations Research, 71*, 335–372.

Mellouli, T. (2001). A Network Flow Approach to Crew Scheduling Based on an Analogy to a Vehicle Maintenance Routing Problem. Computer-Aided Scheduling of Public Transport, *Lecture Notes in Economics and Mathematical Systems*, 505, 91–120.

Metropolis, Nicholas., Rosenbluth, Arianna W., Rosenbluth, Marshall N., Teller, Augusta H., Teller, Edward. (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics, 21*(6), 1087.

Michalewicz, Z. (1996). Genetic Algorithms + Data Structures = Evolution Programs. 3rd ed., Springer – Verlag.

Milano M, and Wallace M. (2010). Integrating Operations Research in Constraint Programming. *Annals of Operations Research, 175*(1), 37–76.

Morgado E. M, and Martins J. P. (1992). Scheduling and managing crew in the Portuguese railways. *Expert Systems with Applications, 5*(3–4), 301–321.

Morgado, E. M. and Martins, J. P. (1998). CREWS_NS - Scheduling train crews in the Netherlands. *AI Magazine, 19*(1), 25-38.

Natalia J. R, David M. R, (2010). The train driver recovery problem - A set partitioning based model and solution method. *Computers & Operations Research, 37*(5), 845-856.

Niemelä, I. (1999). Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence, 25*(3-4), 241–273.

Nishi T, Muroi Y, and Inuiguchi M. (2011). Column generation with dual inequalities for railway crew scheduling problems. *Public Transport, 3*(1), 25–42.

Panayiotis Alefragis, Peter Sanders, Tuomo Takkula and Dag Wedelin (2000). Parallel Integer Optimization for Crew Scheduling. *Annals of Operations Research, 99*(1-4).

Park T and Ryu K R. (2006). Crew pairing optimization by a genetic algorithm with unexpressed genes. *Journal of Intelligent Manufacturing, 17*(4), 375–383.

Pinedo, M L. (2005). *Planning and Scheduling in Manufacturing Services*. Springer.

Rodosek R, Wallace M G, Hajian M T. (1999). A new approach to integrating mixed integer programming and constraint logic programming, *Annals of Operations Research, 86*, 63–87.

Şahin G, and Yüceoğlu B. (2011). Tactical crew planning in railways. *Transportation Research Part E: Logistics and Transportation Review, 47*(6), 1221–1243.

Salido, Miguel A., Garrido A., Barták, R. (2008). Introduction: Special issue on constraint satisfaction techniques for planning and scheduling problems. *Engineering Applications of Artificial Intelligence, 21*(5), 679–682.

Sarker, R. A., and Charles S. N. (2008). Optimization modelling: a practical introduction. CRC Press, Taylor & Francis Group.

Sellmann, M., Zervoudakis, K., Fahle, T. (2002). Crew Assignment via Constraint Programming: Integrating Column Generation and Heuristic Tree Search. *Annals of Operations Research, 115*(1), 207–225.

Shen, Y. and R. Kwan. (2001). Tabu Search for Driver Scheduling. Computer-Aided Scheduling of Public Transport, *Lecture Notes in Economics and Mathematical Systems, 505*, 121–136.

Silva, A. D. (2001). Combining Constraint Programming and Linear Programming on an Example of Bus Driver Scheduling. *Annals of Operations Research, 108*, 277–291

Smith J. C., and Taskın Z. C. (2007). A Tutorial Guide to Mixed-Integer Programming Models and Solution Techniques. *Department of Industrial and Systems Engineering*. University of Florida.

Smith, B. and A. Wren. (1988). A Bus Crew Scheduling System Using a Set Covering Formulation. *Transportation Research, Part A: General, 22A*(2), 97–108.

Sodhi, M. S. and S. Norris (2004). A Flexible, Fast, and Optimal Modeling Approach Applied to Crew Rostering at London Underground. *Annals of Operations Research* 127(1), 259-281.

Souai, N. and J. Teghem (2009). Genetic algorithm based approach for the integrated airline crew-pairing and rostering problem. *European Journal of Operational Research, 199*(3): 674-683.

Stubbs, D.F and Webre, N.W. (1993). *Data Structures with Abstract Data types and Ada*. PWS Publishing Company.

Taylor, B. W. (2009). Integer Programming: The Branch and Bound Method. *Introduction to Management Science*, 10th ed., Prentice Hall.

Tomii, N., Tashiro, Y., Tanabe, N., Hirai, C., and Muraki, K., (2005). Train Operation Rescheduling Algorithm Based On Passenger Satisfaction. *Quarterly Report of Railway Technical Research Institute*, *46*(3), 167-172.

Trick M A. (2003). A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints. *Annals of Operations Research*, *118*(1-4), 73–84.

Tykulsker, R., K. O'Neil, A. Ceder, and Y. Sheffi. (1985). A Commuter Railway Crew Assignment/Work Rules Model. In J. Rousseau (ed.), *Computer Scheduling of Public Transport*, *2*, 233–246. Elsevier Science. University Press, New Jersey.

Vaidyanathan B, Jha K.C, Ahuja R.K. (2007). Multi-Commodity Network Flow Approach to the Railroad Crew Scheduling Problem. *IBM Journal of Research and Development 51*, 325–344.

Van Hentenryck, P. (1999). *The OPL Optimisation Programming Language*. The MIT Press.

Wren, A. and D. Wren. (1995). A Genetic Algorithm for Public Transport Driver Scheduling. *Computers and Operations Research*, *22*(1), 101–110.

Wren, A., S. Fores, et al. (2003). A Flexible System for Scheduling Drivers. *Journal of Scheduling*, *6*(5): 437-455.

Xhafa, F. And Abraham, A. (2008). Metaheuristics for Scheduling in Industrial and Manufacturing Applications. Springer-Verlag.

Yan S, Tu Y. P. (2002). A network model for airline cabin crew scheduling. *European Journal of Operational Research*, *140*(3), 531–540

Yan, S. and J.-C. Chang (2002). Airline cockpit crew scheduling. *European Journal of Operational Research*, *136*(3): 501-511.

Zäpfel, G. and M. Bögl (2008). Multi-period vehicle routing and crew scheduling with outsourcing options. *International Journal of Production Economics*, *113*(2): 980-996.

Zäpfel, Günther (2010). Metaheuristic Search Concepts: A Tutorial with Applications to Production and Logistics. New York: Springer.

Zeghal, F. M. and M. Minoux (2006). Modeling and solving a Crew Assignment Problem in air transportation. *European Journal of Operational Research*, *175*(1): 187-209.

Zhou Feng and Xu Ruihua. (2010). An optimal crew scheduling model for urban rail transit. *Computer Science and Information Technology* (*ICCSIT*), 3rd IEEE International Conference on, 6, 113–116.

Appendices

APPENDIX A–1
Computational Results of Mathematical Programming (MP)

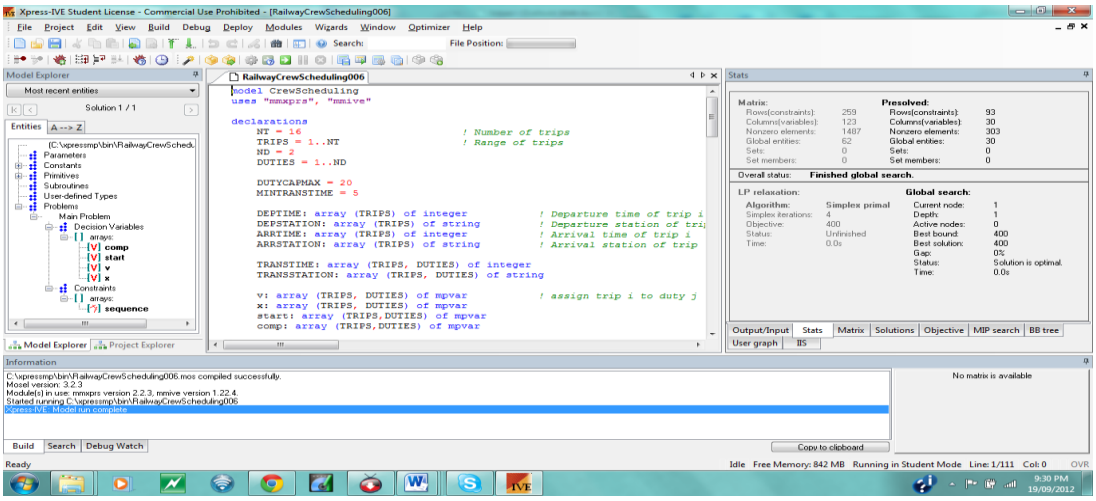

Figure A. 1 The number of variables and constraints of a small-sized instance solved by the mathematical model.
Screen shot of the Xpress Optimizer (FICO) solution on a small-sized instance.

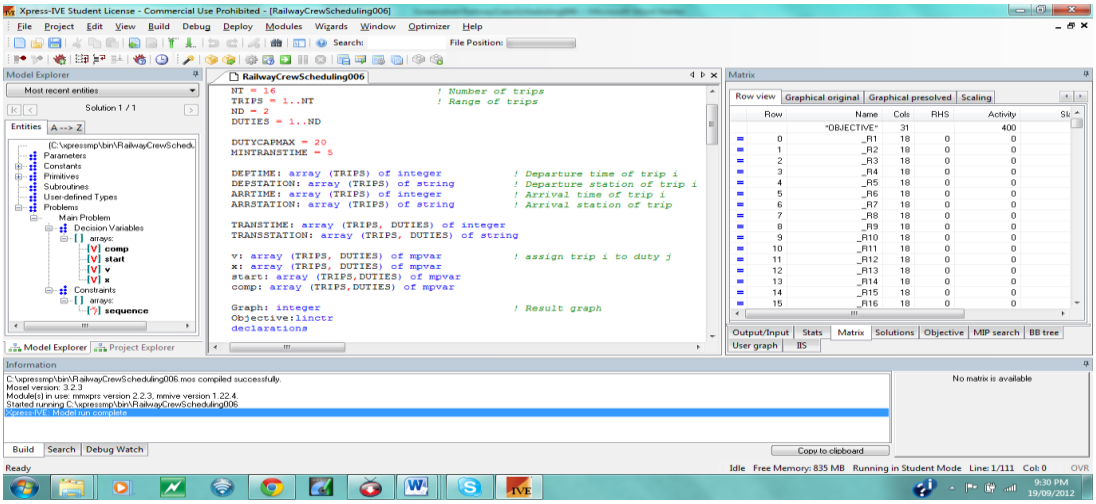

Figure A. 2 Matrix (row view) of a solution by mathematical model.
Screen shot of the Xpress Optimizer (FICO) solution on a small-sized instance.

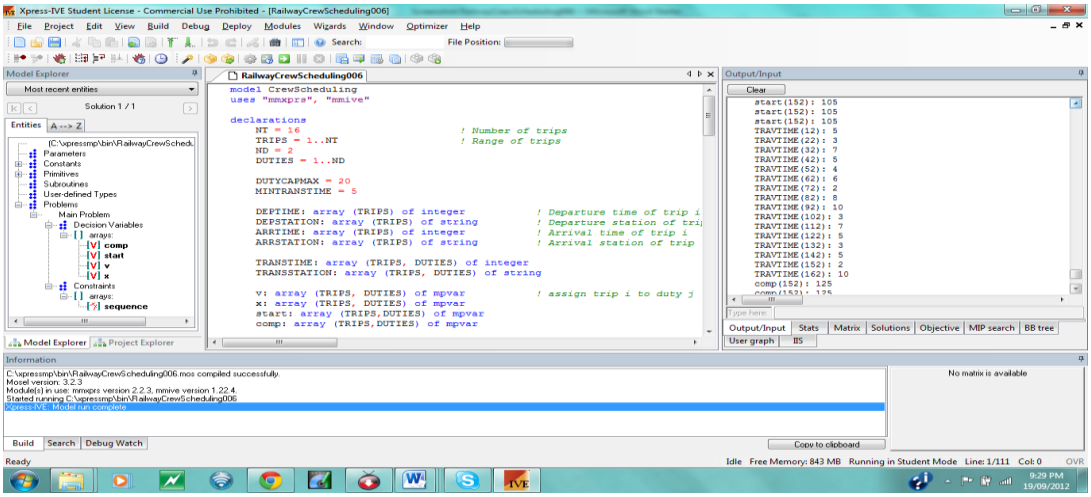

Figure A. 3 Output/input of the mathematical model.
Screen shot of the Xpress Optimizer (FICO) solution on a small-sized instance.

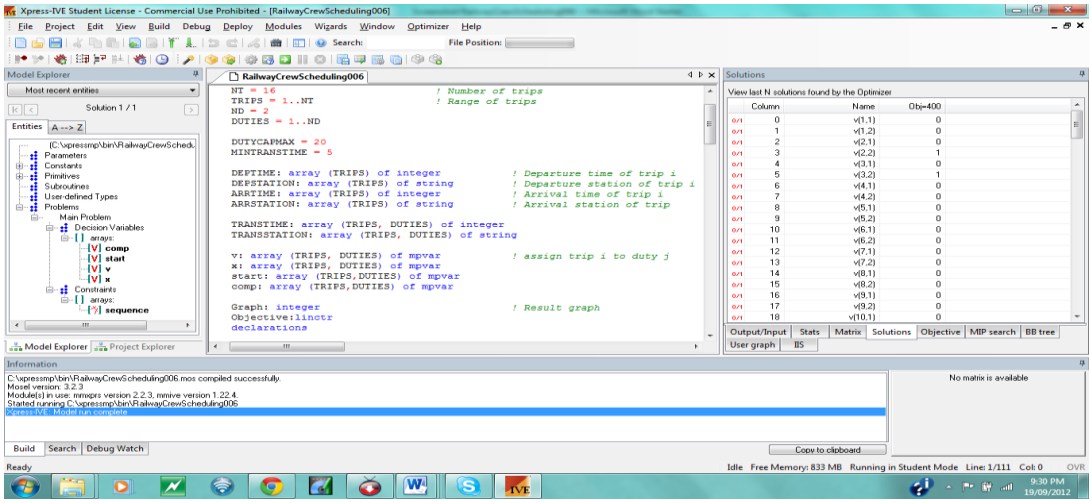

Figure A. 4 Last *n* solutions found by the optimizer.
Screen shot of the Xpress Optimizer (FICO) solution on a small-sized instance.

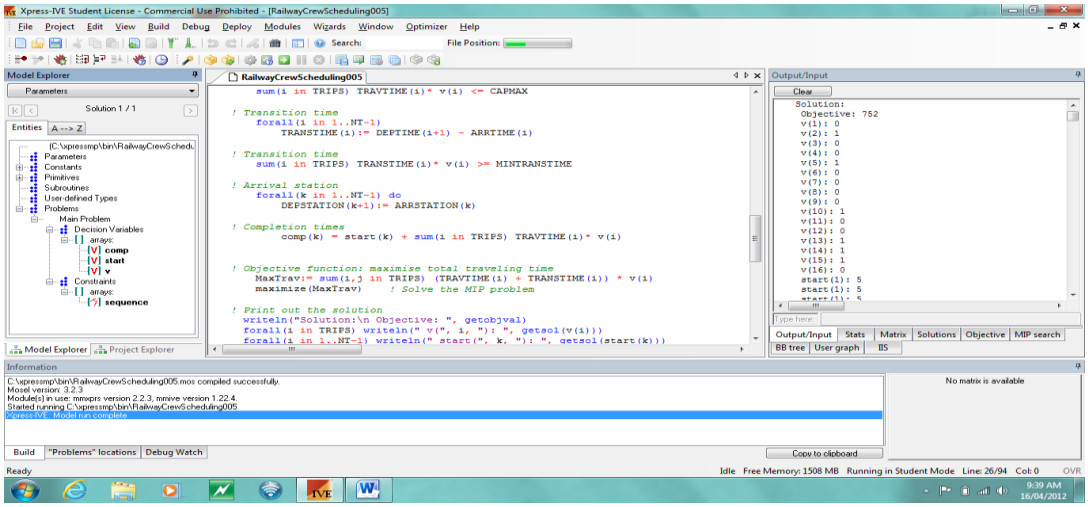

Figure A. 5 The objective function model and the objective value.
Screen shot of the Xpress Optimizer (FICO) solution on a small-sized instance.

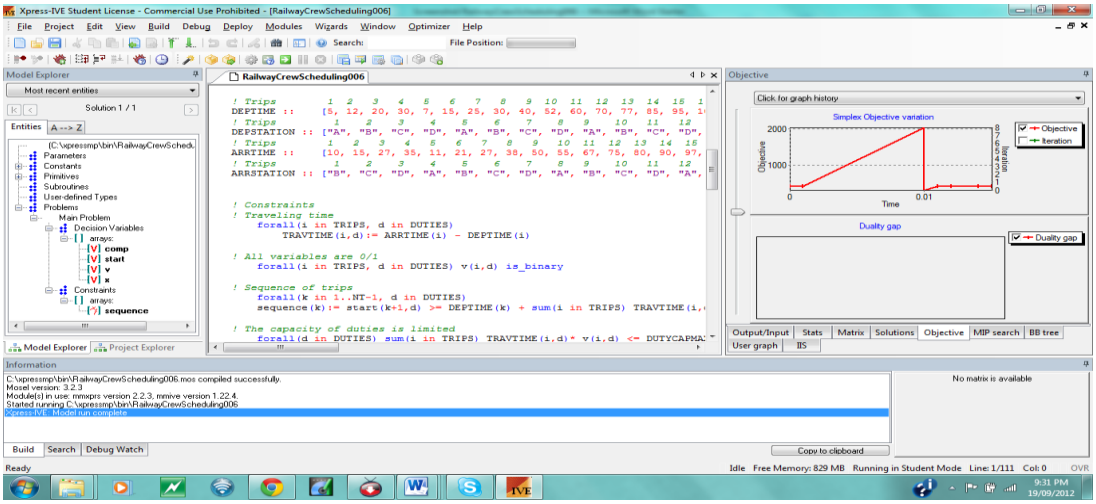

Figure A. 6 The objective and iteration.
Screen shot of the Xpress Optimizer (FICO) solution on a small-sized instance.

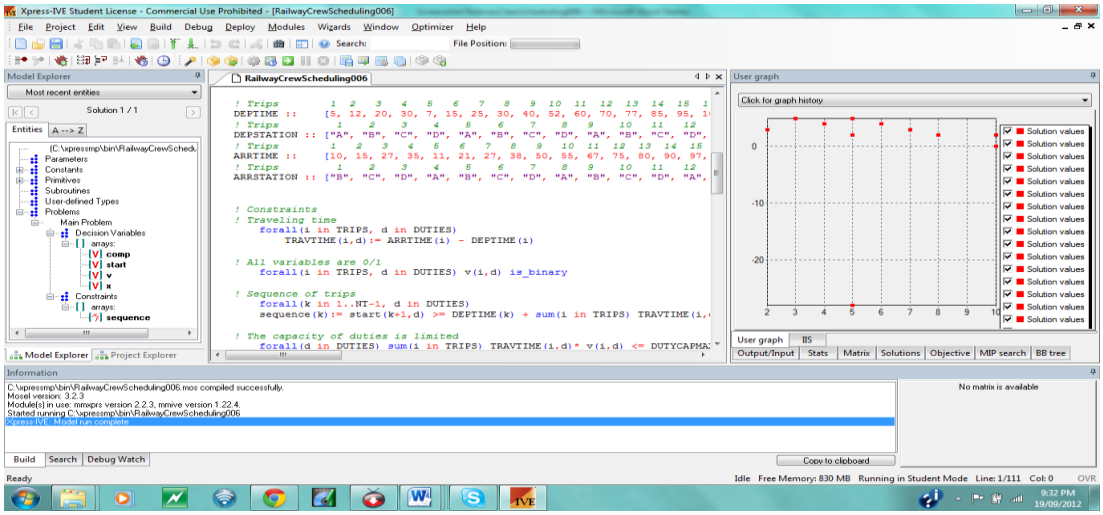

Figure A. 7 The graph history of solution values.
Screen shot of the Xpress Optimizer (FICO) solution on a small-sized instance.

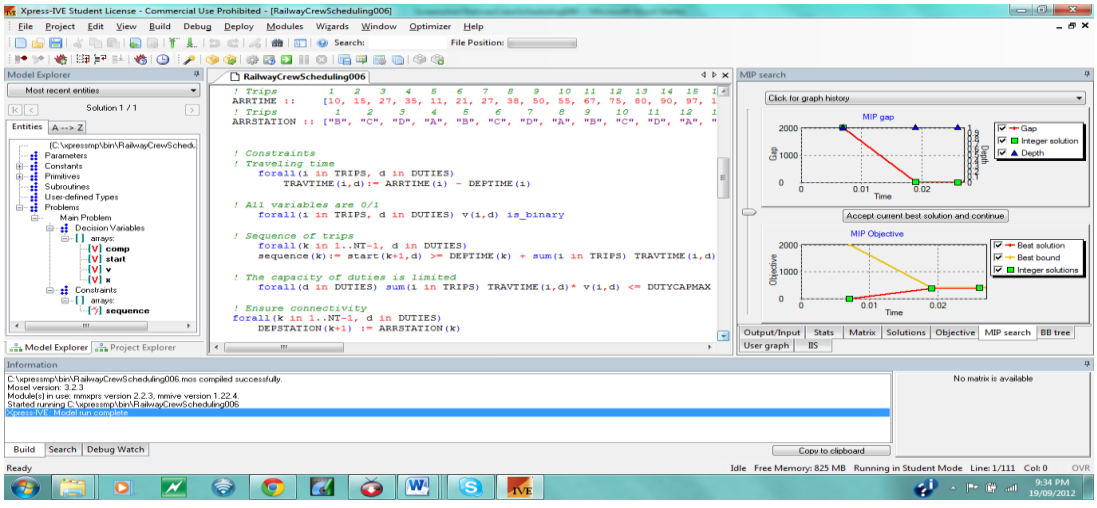

Figure A. 8 The MIP gap and objective.
Screen shot of the Xpress Optimizer (FICO) solution on small-sized instances.

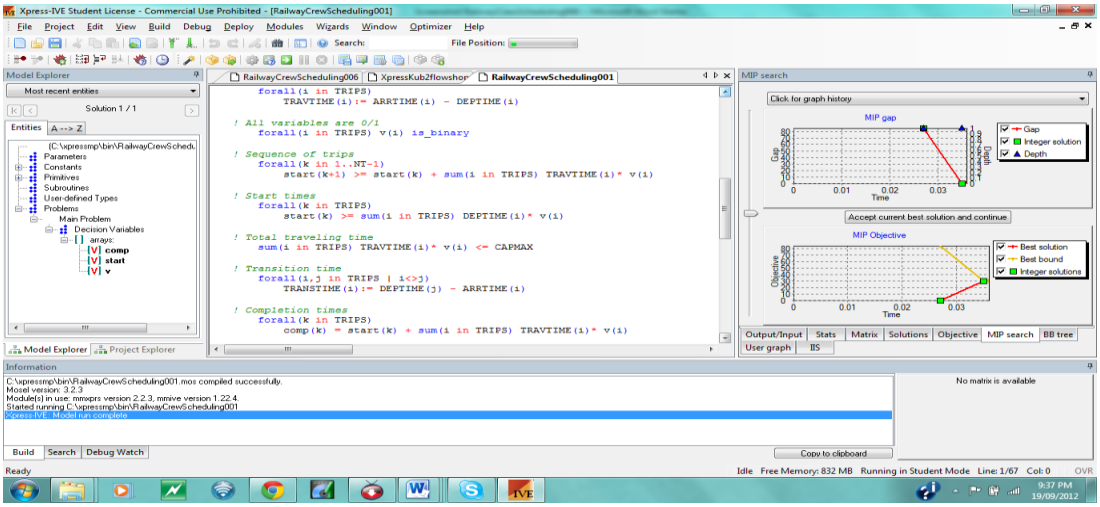

Figure A. 9 The MIP gap and objective.
Screen shot of the Xpress Optimizer (FICO) solution on small-sized instances.

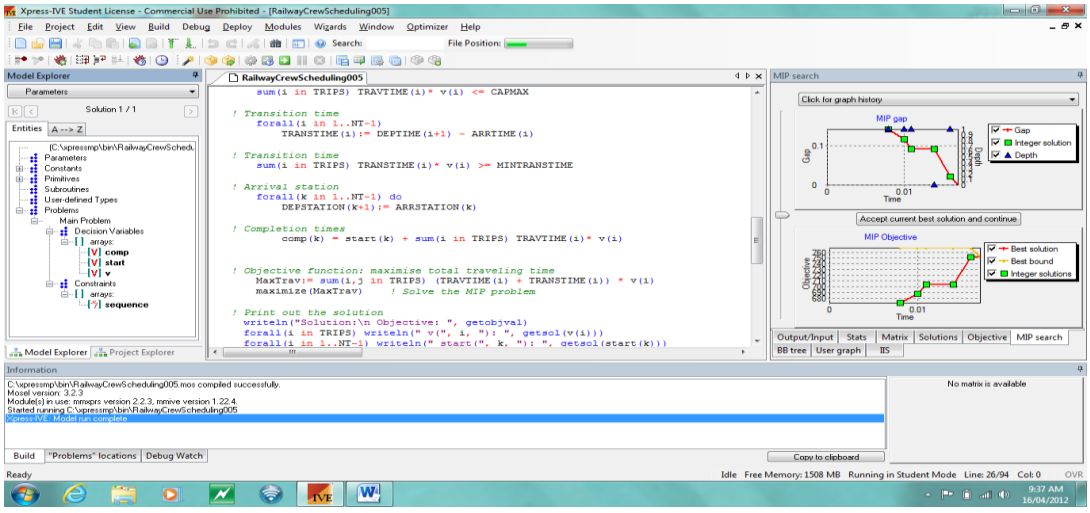

Figure A. 10 The MIP gap and objective.
Screen shot of the Xpress Optimizer (FICO) solution on small-sized instances.

APPENDIX B–1
CPLEX Constraint Programming (CP) Model

```
/*********************************************
 * OPL 6.3 Model
 * Author: n7269102
 * Creation Date: 06/09/2011 at 12:05:40 PM
 *********************************************/


 using CP;

 /*********
 /* Data */
 /********/

 tuple Trip {
      key string tripID;                    // trip (train ID)
      string depSta;                        // departure station
      int depTime;                          // departure time
      string arrSta;                        // arrival station
      int arrTime;                          // arrival time
      int traTime;                          // travelingTime
 };

 {Trip} Trips = ...;                        // list of trips

 int nbCrews = ...;
 int nbmaxShifts = ...;
 int minInterShiftRest = ...;
 int maxIntraShiftWork = ...;
 int maxShiftDuration = ...;

 range Crews = 1 .. nbCrews;
 range Shifts = 1 .. nbmaxShifts;

 tuple Alternative {
      Trip trip;
      int crew;
      int shift;
 };

 {Alternative} Alternatives = {<t,c,s> | t in Trips, c in Crews, s
in Shifts};


 /**********************/
 /* Decision Variables */
 /**********************/

 dvar interval trip [t in Trips] optional in t.depTime .. t.arrTime
size t.traTime;
 dvar interval alt [a in Alternatives] optional;
 dvar interval shift [c in Crews][s in Shifts] optional in
0..maxShiftDuration;
```

```
 /**********************/
 /* Objective Function */
 /**********************/

 // Minimize total number of duties (shifts)

 minimize sum (c in Crews, s in Shifts) presenceOf (shift[c][s]);

 subject to {

 forall (t in Trips)
      alternative (trip[t], all(a in Alternatives: a.trip==t)
alt[a]);


 /***************/
 /* Constraints */
 /***************/

 // station (***) sequence
      forall (t in Trips){
      forall (s in Trips: s.depSta==t.arrSta) {
      endBeforeStart (trip[t], trip[s]);
      presenceOf (trip[s]) => presenceOf (trip[t]);}}

 // time (***) sequence
 forall (t in Trips){
      forall (s in Trips: s.depTime >= t.arrTime) {
      endBeforeStart (trip[t], trip[s]);
      presenceOf (trip[s]) => presenceOf (trip[t]);}}

 // shift sequence for each crew
 forall (c in Crews, s in 1..nbmaxShifts-1)
      endBeforeStart (shift[c][s], shift[c][s+1],
minInterShiftRest);


 // shift optionality chain for each crew
 forall (c in Crews, s in 1..nbmaxShifts-1)
      presenceOf (shift[c][s+1]) => presenceOf (shift[c][s]);

 forall (c in Crews, s in Shifts){
 // shift spanning interval
      span (shift[c][s], all(a in Alternatives: a.crew==c &&
a.shift==s) alt[a]);
 // max intra shift work constraint
      sum (a in Alternatives: a.crew==c && a.shift==s) lengthOf
(alt[a]) <= maxIntraShiftWork;
 // crew unary capacity during shift
      noOverlap (all(a in Alternatives: a.crew==c && a.shift==s)
alt[a]);}

 }
```

APPENDIX B–2
CPLEX Data Example

```
/*********************************************
 * OPL 6.3 Data
 * Author: n7269102
 * Creation Date: 06/09/2011 at 12:05:40 PM
 *********************************************/

 nbCrews = 35;
 nbmaxShifts = 3;
 minInterShiftRest = 30;
 maxIntraShiftWork = 240;
 maxShiftDuration = 720;
 Trips = {
 < F06, FYG, 300, BNH, 395, 95 >,
 < F14, FYG, 330, BNH, 423, 93 >,
 < F18, FYG, 360, BNH, 454, 94 >,
 < F26, FYG, 390, BNH, 485, 95 >,
 < F28, FYG, 426, BNH, 515, 89 >,
 < F30, FYG, 446, BNH, 539, 93 >,
 < F32, FYG, 475, BNH, 570, 95 >,
 < F34, FYG, 510, BNH, 605, 95 >,
 < F29, BNH, 520, FYG, 620, 100 >,
 < F35, BNH, 650, FYG, 750, 100 >,
 < F37, BNH, 760, FYG, 860, 100 >,
 < F38, BNH, 865, FYG, 950, 85  >,
 < B06, FYG, 300, BHI, 324, 24 >,
 < B14, FYG, 330, BHI, 354, 24 >,
 < B18, FYG, 360, BHI, 384, 24 >,
 < B26, FYG, 390, BHI, 414, 24 >,
 < B28, FYG, 426, BHI, 446, 20 >,
 < B30, FYG, 446, BHI, 470, 24 >,
 < B32, FYG, 475, BHI, 499, 24 >,
 < B06a, BHI, 325, BNH, 395, 70 >,
 < B14a, BHI, 355, BNH, 423, 68 >,
 < B18a, BHI, 385, BNH, 454, 69 >,
 < B26a, BHI, 415, BNH, 485, 70 >,
 < B28a, BHI, 447, BNH, 515, 68 >,
 < B30a, BHI, 471, BNH, 539, 68 >,
 < B32a, BHI, 500, BNH, 570, 70 >,
 < E07, BNH, 268, BHI, 341, 73 >,
 < E11, BNH, 299, BHI, 368, 69 >,
 < E13, BNH, 327, BHI, 397, 70 >,
 < E19, BNH, 358, BHI, 428, 70 >,
 < E25, BNH, 389, BHI, 459, 70 >,
 < E29, BNH, 402, BHI, 472, 70 >,
 < E31, BNH, 419, BHI, 490, 71 >,
 < E35, BNH, 449, BHI, 514, 65 >,
 < E07a, BHI, 342, FYG, 368, 26 >,
 < E11a, BHI, 369, FYG, 398, 30 >,
 < E13a, BHI, 398, FYG, 422, 24 >,
 < E19a, BHI, 429, FYG, 454, 25 >,
 < E25a, BHI, 460, FYG, 485, 25 >,
 < E29a, BHI, 473, FYG, 498, 25 >,
 < E31a, BHI, 491, FYG, 518, 27 >,
 < E35a, BHI, 515, FYG, 539, 24 >,
 < E37a, BHI, 553, FYG, 578, 25 >,
 < E41a, BHI, 609, FYG, 635, 26 >,

 };
```

APPENDIX C–1
Simulated Annealing Selected C# Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.OleDb;
using System.Data;
using System.Data.Odbc;

namespace SchedulingProject
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(" ");

// initialise parameters
            int depTime = 0;
            int arrTime = 0;
            int travTime = 0;
            int dutyCap = 0;
            int MaxDutyCapAllowed = 540;
            int dutyMin = 180;
            int transTime = 5;

// SA variables for temperature and cooling rate

            int iteration = 0;
            int maxIteration = 1000000;
            double currentTemp = 10000.0;
            double alpha = 0.975;

            Console.WriteLine("\nInitial State:");
            Display(state);
            Console.WriteLine("Initial Energy: " + energy.ToString("F2"));
            Console.WriteLine("\nSimulated Annealing ");
            Console.WriteLine("Init Temp = " + currentTemp.ToString("F1")+
"\n");

// Provider=Microsoft.Jet.OLEDB.4.0;Data Source="C:\Documents and
Settings\n7269102\My Documents\TimeTable\SchedulingDB.mdb"
// define the connection string

            string strAccessConn = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\\Documents and Settings\\n7269102\\My Documents\\
TimeTable\\SchedulingDB.mdb";

            OleDbConnection myAccessConn = new OleDbConnection(strAccessConn);
            myAccessConn.Open();
// open the connection
            Console.WriteLine("database connected");

// accessing Table
            string strAccessSelectTable2a = "SELECT * FROM Iots1 ";
```

```
            OleDbCommand myAccessCommandTable2a = new
OleDbCommand(strAccessSelectTable2a, myAccessConn);

// execute the command, get the result in datareader

            OleDbDataReader myDataReaderTable2a =
             myAccessCommandTable2a.ExecuteReader();
            string[,] table2adata = new string[50, 6];
            int row = 0;
            int coloumn = 0;
            while (myDataReaderTable2a.Read())
            {

            table2adata[row, coloumn] = myDataReaderTable2a.GetString(0);
table2adata[row, coloumn + 1] = myDataReaderTable2a.GetString(1);
table2adata[row, coloumn + 2] = myDataReaderTable2a[2].ToString();
table2adata[row, coloumn + 3] = myDataReaderTable2a.GetString(3);
                table2adata[row, coloumn + 4] =
myDataReaderTable2a[4].ToString();
                table2adata[row, coloumn + 5] =
myDataReaderTable2a[5].ToString();

                row = row + 1;
                coloumn = 0;
            }
            string strAccessSelectTable2b = "SELECT * FROM Iots2 ";
            OleDbCommand myAccessCommandTable2b = new
             OleDbCommand(strAccessSelectTable2b, myAccessConn);
            OleDbDataReader myDataReaderTable2b =

// execute this command, get the result in datareader

myAccessCommandTable2b.ExecuteReader();
            string[,] table2bdata = new string[50, 6];
            int row2b = 0;
            int coloumn2b = 0;
            while (myDataReaderTable2b.Read())
            {
                table2bdata[row2b, coloumn2b] =
myDataReaderTable2b.GetString(0);
                table2bdata[row2b, coloumn2b + 1] =
myDataReaderTable2b.GetString(1);
                table2bdata[row2b, coloumn2b + 2] =
myDataReaderTable2b[2].ToString();
                table2bdata[row2b, coloumn2b + 3] =
myDataReaderTable2b.GetString(3);
                table2bdata[row2b, coloumn2b + 4] =
myDataReaderTable2b[4].ToString();
                table2bdata[row2b, coloumn2b + 5] =
myDataReaderTable2b[5].ToString();
                row2b = row2b + 1;
                coloumn2b = 0;
            }

            string strAccessSelectTable3= "SELECT * FROM Ithd ";
            OleDbCommand myAccessCommandTable3= new
OleDbCommand(strAccessSelectTable3 ,myAccessConn);
            OleDbDataReader myDataReaderTable3 =
myAccessCommandTable3.ExecuteReader();
            string[,] table3data = new string[50, 6];
            int row3 = 0;
            int coloumn3 = 0;
```

```
            while (myDataReaderTable3.Read())
            {
                table3data[row3, coloumn3] = myDataReaderTable3.GetString(0);
                table3data[row3, coloumn3 + 1] =
myDataReaderTable3.GetString(1);
                table3data[row3, coloumn3 + 2] =
myDataReaderTable3[2].ToString();
                table3data[row3, coloumn3 + 3] =
myDataReaderTable3.GetString(3);
                table3data[row3, coloumn3 + 4] =
myDataReaderTable3[4].ToString();
                table3data[row3, coloumn3 + 5] =
myDataReaderTable3[5].ToString();

                row3 = row3 + 1;
                coloumn3 = 0;
            }


            string strAccessSelect = "SELECT * FROM Iohd order by dT asc";
            DataSet myDataSet = new DataSet();

            OleDbCommand myAccessCommand = new OleDbCommand(strAccessSelect,
myAccessConn);
            OleDbDataReader myDataReader = myAccessCommand.ExecuteReader();
            int rowtable2a = 0;
            int rowtable2b = 0;
            int rowtable3 = 0;

            while (myDataReader.Read())
            {
                while (dutyCapacity <= dutyMax)
                {
                    string baseidValue = myDataReader["ID"].ToString();

                    string baseDepartStationValue =
myDataReader["dS"].ToString();
                    string baseArriveStationValue =
myDataReader["aS"].ToString();
                    int baseDepartTimeValue =
int.Parse(myDataReader["dT"].ToString());

                    int baseArriveTimeValue =
int.Parse(myDataReader["aT"].ToString());

                    travTime = baseArriveTimeValue - baseDepartTimeValue;
                    duty = travTime;
                    dutyCap = duty + dutyCap;

                    Console.Write("Base Value of Travelling Time" +
travellingTime); Console.Write("Base Value of Duty Capacity" + dutyCap);
                    Console.WriteLine();


                    while (!table2adata[rowtable2a,
1].Equals(baseArriveStationValue) && !((int.Parse(table2adata[rowtable2a, 2])
- baseArriveTimeValue) <= transitionTime))
                    {
                        rowtable2a = rowtable2a + 1;
                    }
```

```
                    while (!table2bdata[rowtable2b,
1].Equals(table2adata[rowtable2a, 1]))
                    {
                        rowtable2b = rowtable2b + 1;
                    }

                    while (!table3data[rowtable3,
1].Equals(table2bdata[rowtable2b, 1]))
                    {
                        rowtable3 = rowtable3 + 1;
                    }

                    rowtable2a = rowtable2a + 1;
                    rowtable2b = rowtable2b + 1;
                    rowtable3 = rowtable3 + 1;
public Cap()
            {
                trips = new List<Trip>();
            }
            void AddTrip(Trip i)
            {
                if ((TotalTravTime + i.TravTime) < MaxDutyCapAllowed)
                    trips.Add(i);
            }
            public void Calculate(List<Trip> trips)
            {
                foreach (Trip i in Sort(trips))
                {
                    AddTrip(i);
                }
            }
            List<Trip> Sort(List<Trip> inputTrips)
            {
                List<Trip> choosenTrips = new List<Trip>();
                for (int i = 0; i < inputTrips.Count; i++)
                {
                    int j = -1;
                    if (i == 0)
                    {
                        choosenTrips.Add(inputTrips[i]);
                    }
                    if (i > 0)
                    {
                        if (!Recursive(inputTrips, choosenTrips, i,
choosenTrips.Count - 1, false, ref j))
                        {
                            choosenTrips.Add(inputTrips[i]);
                        }
                    }
                }
                return choosenTrips;
            }

            bool Recursive(List<Trip> capTrips, List<Trip> choosenTrips, int
i, int lastBound, bool dec, ref int indxToAdd)
            {
                if (!(lastBound < 0))
                {
                    if (capTrips[i].ResultWV <
choosenTrips[lastBound].ResultWV)
                    {
                        indxToAdd = lastBound;
```

```
                    }
                    return Recursive(capTrips, choosenTrips, i, lastBound - 1,
true, ref indxToAdd);
                }
                if (indxToAdd > -1)
                {
                    choosenTrips.Insert(indxToAdd, capTrips[i]);
                    return true;
                }
                return false;
            }
            #region IEnumerable<Trip> Members
            IEnumerator<Trip> IEnumerable<Trip>.GetEnumerator()
            {
                foreach (Trip i in trips)
                    yield return i;
            }
            #endregion

            #region IEnumerable Members
            System.Collections.IEnumerator
System.Collections.IEnumerable.GetEnumerator()
            {
                return trips.GetEnumerator();
            }

            #endregion

            public int TotalTravTime
            {
                get
                {
                    var sum = 0;
                    foreach (Trip i in this)
                    {
                        sum += i.TravTime;
                    }
                    return sum;
                }
            }

            public class Trip
            {
                public string TripID { get; set; } public int TravTime { get;
set; } public int Value { get; set; } public int Result { get { return
TravTime-Value; } }

                public override string ToString()
                {
                    return "TripID : " + TripID + "  TravelingTime : " +
TravTime +    "   TransTime : "    + Value;
                }
            }
        }

    class Program
    {

        static void Main(string[] args)
        {

            List<Cap.Trip> capTrips = new List<Cap.Trip>();
```

```
            Cap b = new Cap();
            b.Calculate(capTrips);
            b.All(x => { Console.WriteLine(x); return true; });
            Console.WriteLine(b.Sum(x ===> x.TravTime));
            Console.ReadKey();
        }

    }


                random = new Random(0);
                int numCrews = 35;
                int numTrips = 500;
                double[][]Datasets = myDatasets(numCrews, numTrips);
                int[] state = RandomState(Datasets);
                double energy = Energy(state, Datasets);
                int[] bestState = state;
                double bestEnergy = energy;
                double adjacentEnergy;
                int[] adjacentState;


                while (iteration < maxIteration && currentTemp > 0.0001)
                {
                    adjacentState = AdjacentState(state, Datasets);
                    adjacentEnergy = Energy(adjacentState, Datasets);
                    if (adjacentEnergy < bestEnergy)
                    {
                        bestState = adjacentState;
                        bestEnergy = adjacentEnergy;
                        Console.WriteLine("New best solution found:");
                        Display(bestState);
                        Console.WriteLine("Energy = " +
bestEnergy.ToString("F2") + "\n");
                    }

                    double p = random.NextDouble();
                    if (AcceptanceProb(energy, adjacentEnergy, currentTemp)>p)
                    {
                        state = adjacentState;
                        energy = adjacentEnergy;
                    }
                    currentTemp = currentTemp * alpha;
                    ++iteration;
                }

                while (iteration < maxIteration && currentTemp > 0.0001)
                {

                Console.Write("Temperature reached ");
                Console.WriteLine("at iteration " + iteration);
                Console.WriteLine("Simulated Annealing complete");
                Console.WriteLine("\n------------------------ ");
                Console.WriteLine("\nBest found solution: ");
                Display(bestState);
                Console.WriteLine("Best energy = " + bestEnergy.ToString("F2")
+ "\n");
                Interpret(bestState, Datasets);
                Console.WriteLine("\nEnd Simulated Annealing \n");
                Console.WriteLine("\n------------------------ ");
                Console.ReadLine();
```

```
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                Console.ReadLine();
                Console.ReadLine();
                Console.ReadLine();
            }
        }


        static double[][] myDatasets(int numCrews, int numTrips)
        {
            double[][] result = new double[numTrips][];
            for (int c = 0; c < result.Length; ++c) result[c] = new
double[numTrips];


        }

        static int[] RandomState(double[][]Datasets)
        {
            int numCrews = Datasets.Length;
            int numTrips = Datasets[0].Length;
            int[] state = new int[numTrips];
            for (int t = 0; t < numTrips; ++t)
            {
                int c = random.Next(0, numCrews);
                while (Datasets[c][t] == 0.0)
                {
                    ++c;
                    if (c > numCrews - 1) c = 0;
                }
                state[t] = c;
            }
            return state;
        }
        static int[] AdjacentState(int[] currentState, double[][]Datasets)
        {

            int numCrews = Datasets.Length;
            int numTrips = Datasets[0].Length;
            int[] state = new int[numTrips];
            int trip = random.Next(0, numTrips);
            int crew = random.Next(0, numCrews);

            while (Datasets[crew][trip] == 0.0)
            {
                ++crew; if (crew > numCrews - 1) crew = 0;
            }
            currentState.CopyTo(state, 0);
            state[trip] = crew;
            return state;
        }

        static double Energy(int[] state, double[][] Datasets)
        {
            double result = 0.0;
            for (int t = 0; t < state.Length; ++t)
            {
                int crew = state[t];
                double time = Datasets[crew][t];
```

```
                result += time;
            }

            int numCrews = Datasets.Length;
            int[] numJobs = new int[numCrews];
            for (int t = 0; t < state.Length; ++t)
            {
                int crew = state[t];
                ++numJobs[crew];
                if (numJobs[crew] > 1) result += 7.30;
            } return result;
        }

        static double AcceptanceProb(double energy, double adjacentEnergy,
             double currentTemp)
        {
            if (adjacentEnergy < energy)
                return 1.0;
            else
                return Math.Exp((energy - adjacentEnergy) / currentTemp);
        }


        static void Display(double[][] matrix)
        {
            for (int i = 0; i < mat.Length; ++i)
            {
                for (int j = 0; j < mat [i].Length; ++j)
                Console.Write(mat [i][j].ToString("F2") + " ");
                Console.WriteLine("");
                Console.WriteLine(" ---------------------------------- ");
                Console.WriteLine(" ---------------------------------- ");

            }
        }
        static void Display(int[] vector)
        {
            for (int i = 0; i < vector.Length; ++i)
            Console.Write(vector[i] + " ");
            Console.WriteLine("");
        }
        static void Interpret(int[] state, double[][] Datasets)
        {
            for (int t = 0; t < state.Length; ++t)
            {
                int c = state[t];
                Console.Write("Trip [" + t + "] assigned to crew ");
                Console.WriteLine(c + ", " + Datasets[c][t].ToString("F2"));
                Console.WriteLine(" ---------------------------------- ");
                Console.WriteLine(" ---------------------------------- ");

            }
        }
    }
}
```