Constraint-Based Local Search for Container Rail Scheduling

By

Nakorn Indra-Payoong

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

THE UNIVERSITY OF LEEDS
SCHOOL OF COMPUTING

February 2005

Acknowledgements

I owe a great debt of gratitude to my supervisors, Dr. Raymond R.S. Kwan and Dr. Les Proll for their constant guidance and supervision as well as greatly assisting in various other ways.

I would like to thank all my friends in Leeds for their various kinds of help.

Finally, I will never forget to thank my wife, Kanchana, who has given me infinite love and always cheered me up during the research, and thanks very much to all members of my family in my homeland which have transmitted their encouragement across continents.

Abstract

A traditional container rail service is based on regular timetables. This causes a risk that some customers may turn away if their preferred itinerary is not attainable and the take-up of some services in a fixed schedule may be low and therefore not profitable. To increase railway's profitability and competitiveness, a demand responsive schedule would be advantageous. The decision support model and algorithms for producing a schedule in advance of the weekly operation is the main subject of this thesis.

The container rail scheduling problem is modelled as a constraint satisfaction problem in which the rail business criteria and operational constraints are represented as soft and hard constraints respectively. A constraint-based local search algorithm is developed to solve problems of realistic size. The algorithm includes strategies for accepting non-improving moves and randomised selection of violated constraints and variables to explore. These strategies aim to achieve diversified exploration of the search space. Different measures of the constraint violation are also used to drive the search to promising solution regions.

A predictive choice model is introduced for search intensification to improve further the quality of solutions for the problem. With sufficient trial history, the model will predict a good choice of value for a variable. The variable will be fixed at its predicted value for a dynamically determined number of trials. At this point, the propagation of consistency between the variables is enforced, leading to intensified exploration of the search space. Experimental results, based on real data from the Royal State Railway of Thailand, have shown good computational performance of the approach and suggested benefits can be achieved for both the rail carrier and its customers.

Finally, the proposed algorithm for rail scheduling has been adapted to solve the generalised assignment problem, a well-known hard combinatorial optimisation problem. The experimental results have shown that the proposed method can obtain high quality solutions that are as good as or close to the solutions obtained from the existing methods, but with using significantly less computational time. This suggests that generalising the method may be a promising approach for other combinatorial problems in which all decision variables in the model are binary and where quick and high quality solutions are desirable.

Declarations

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

Indra-Payoong, N., Kwan, R.S.K. and Proll, L.G. "Rail container service planning: a constraint-based approach," (to appear in) Journal of Scheduling.

Indra-Payoong, N., Kwan, R.S.K. and Proll, L.G. (2004) "An adaptively relaxed constraint satisfaction approach for a demand responsive freight rail timetabling problem," (presented at) The 5th International Conference on Practice and Theory of Automated Timetabling, August 18-20, Pittsburgh, USA.

Indra-Payoong, N., Kwan, R.S.K. and Proll, L.G. (2004) "Demand responsive scheduling of rail container traffic," Proceedings of the 10th World Conference on Transport Research.

Indra-Payoong, N., Kwan, R.S.K. and Proll, L.G. (2003) "A randomised algorithm with prediction of rail container service planning" (invited talk at) Scheduling Workshop on Application of Constraint Programming, 9 -10 September, University of Huddersfield.

Indra-Payoong, N., Kwan, R.S.K. and Proll, L.G. (2003) "Constraint-based local search for rail container service planning," in: Kendall G, Burke E & Petrovic S (Eds), Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications, vol. 2, pp. 660 - 677.

Indra-Payoong, N., Kwan, R.S.K. and Proll, L.G. (2002) "Strategic scheduling of containerised rail freight using constraint programming," (presented at) ORS Local Search Workshop, 16-17 April, London.

Contents

List of Tables

List of Figures

Chapter One

Introduction

1.1 Problem background

In Thailand, ports are linked with inland by state-owned rail, which is mainly single-tracked and used by both passengers and freight. We consider the problem of the eastern-line container rail service in Thailand which serves container traffic between Bangkok and the eastern region. In 2000, it was roughly estimated that more than a million containers a year were moved between these areas, of which 30 percent was carried by rail transport (National Economic and Social Development Board, 2000). In a later year, because of traffic congestion and environmental concerns in Bangkok, the capital city of the country, the Thai government decided to limit the number of containers via Bangkok port to one million containers per year. As a result, Laem Chabang port, located in the eastern region, will have to serve increasing container flow between that region and Bangkok.

How to maintain and increase profitability is a major concern for the eastern rail line in order to stay competitive with the growing number of trucking companies as providing container transport service becomes a lucrative business. In general, the rail carrier can increase

profitability in two ways: 1) the railway could generate the same or similar amount of revenue with lower operating costs, 2) the railway may satisfy more customer demand in order to maintain the current level of revenue or receive additional profit. The question may then be how to run the rail business as effectively as possible.

1.1.1 Overview of freight rail planning process

The transportation of rail freight is a complex domain, with several processes and levels of decision, where investments are capital-intensive and usually require long-term strategic plans. In addition, the transportation of rail freight has to adapt to rapidly changing political, social, and economic environments. In general, freight rail planning involves five main processes: path formulation, fleet assignment, schedule production, crew scheduling and fleet repositioning. Figure 1.1 presents the freight rail planning process.

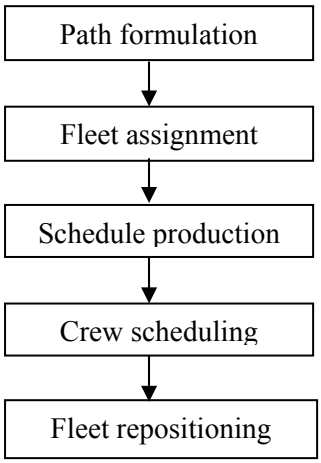

Figure 1.1: Freight rail planning process

Path formulation. The first step in the planning process is path formulation, as shown in Figure 1.2. The formulation process computes container flows in the network using the shortest path or minimum generalised cost, based on the historical demand data. The path formulation is generally not changed frequently, and this step hardly occurs in practice. Note that in case the railway is privatised, routes are usually fixed by contract and therefore this step is performed only at the strategic level and not at the operational level.

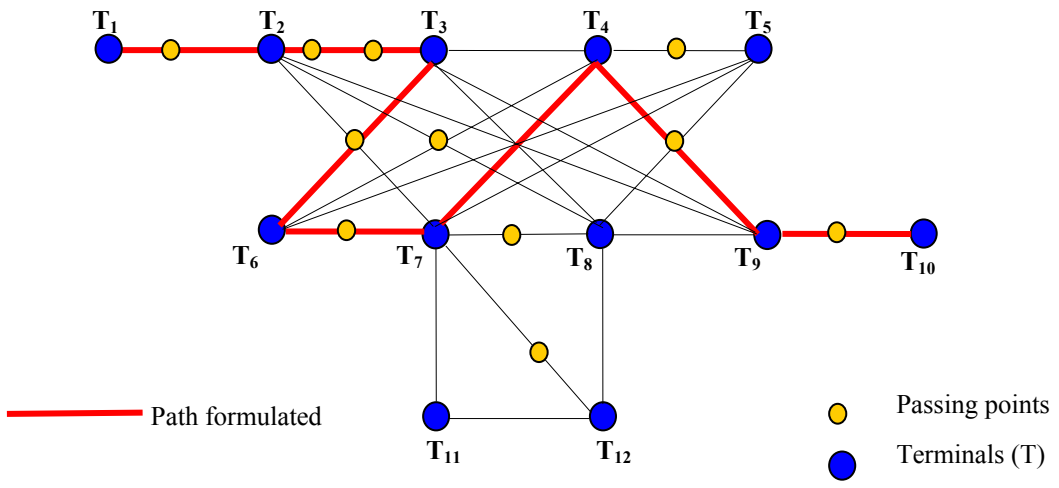

Figure 1.2: An example of path formulation

Fleet assignment. The second step in the planning process is fleet assignment. The goal of this step is to allocate the available fleet (locomotives and wagons) to service pairs so that the capacity matches the average transport demand at both ends. For instance, at a terminal, the fleet cycle starts when customers request services and then compatible locomotive and wagons are grouped and moved to a loading point. Once the containers have been loaded, the train formation will then be adjusted to the requirements at the destination. At this point, the fleet is available with more or less the same capacity for a new shipment in the reverse

direction and the cycle may repeat. Fleet assignment involves huge capital investment, which is done infrequently. If the rail carrier assigns a fleet to a particular origin-destination (OD) pair it wants to serve, the commitment is relatively long term. Therefore, most of the time, the remaining steps in the planning process use a fixed given fleet. Note that in contrast to road and air transportation, a rail carrier uses fixed tracks. Changes to the service network cannot be done easily because they require huge capital investment and involve a number of operational constraints, such as track availability, handling equipment, customs procedures and so forth.

Schedule production. Schedule production typically starts several months before a schedule goes into operation. In this process, significant amounts of time and resources are used to produce a profitable schedule. In general, the scheduling process begins with an existing schedule, which will then be developed or changed, based on historical transport demand data. The aim of this step is to determine, under operational constraints, the appropriate service frequency throughout the month or several months. This step is the most difficult. If the services are too frequent, the rail carrier bears high operational costs. If the services are too infrequent, some potential customers may turn away, resulting in lost revenue.

Crew scheduling. Crew scheduling is concerned with the development of duty schedules for crews, in order to cover a given train schedule. This step involves the short-term (typically one day or one week) tactical scheduling of crew, with the aim of developing a set of duties that will be performed by each crew to cover adequately the train schedule. The objective is to find the minimum cost assignment of crews and attendants to service lines, subject to some restrictions. For instance, train drivers are qualified for certain locomotive classes and service lines; the schedules must satisfy restrictions on maximum working hours and so on.

Fleet repositioning. The last step of rail planning process is fleet repositioning. The imbalance between demand and supply is reflected in the fact that at any point in time there are terminals with a surplus of locomotives and wagons of a certain type, whilst some other terminals show a shortage. Moving empty locomotives and wagons does not directly contribute to the profit of rail carrier but it is essential to its continuing operations. Nowadays, the rail industry has a pooling agreement to consolidate its resources. Under this agreement, the rail carriers agree to pool the locomotives and wagons of each type, so that the empty locomotives and wagons would be redirected to other destinations, rather than being sent back to the point of origin. The operation faced by each rail carrier in the pool is to determine the fleet size it needs to acquire for any given period, and to determine the apportionment of responsibility for the capital investment amongst the participating rail carriers.

In this research we address an issue in schedule production, that of constructing profitable schedules for the rail service.

1.1.2 Container rail scheduling problem

Typically the transportation of rail containers involves an international context. Freight forwarders or multimodal transport operators, who act on behalf of their customers, regularly book those services in advance to make sure that the shipment is delivered to their customers within expected time.

In general, shipping lines cannot provide frequent sailing services. They often provide a weekly service or more. This is because container ships are big (greater than 8,000 twenty-

equivalent unit (TEU) container) in order to achieve economy of scale; thereby significant cost reduction can be obtained. Once containers arrive at the seaport, there is a need to move them to their final customers, which can basically be done either by rail, via intermediate terminals, or by truck direct to the final destinations (see Figure 1.3). Shippers would like to minimise the lead-time of the transport chain.

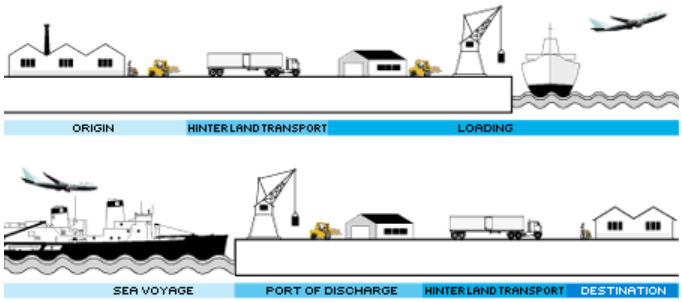

Figure 1.3: A typical container transport

At present, the eastern line container rail carrier provides a weekly fixed schedule, in which a certain number of train services are provided in fixed departure timeslots. This creates a risk that take-up of some services in a fixed schedule may be low and not profitable, and some customers may turn away if their requirements are not satisfied. In order to create a profitable schedule, a container rail carrier needs to engage in a decision-making process with multiple business criteria and numerous constraints, which is challenging. Further descriptions of the problem are given in Chapter 3.

1.2 Research proposal

Generating a good schedule is of utmost importance to a container rail business because rail's profitability is heavily influenced by its service offerings. The container rail scheduling problem has attracted much research interest over several decades; from the view points of both problem modelling and solution techniques. Although a great deal of effort has been made in this area, there are still many aspects that need to be further investigated and improved. A model that incorporates challenging practical situations and advanced solution techniques is significant. The research proposal may be divided into two principal areas:

Application domain. A rail carrier's profitability is influenced by the railway's ability to construct schedules for which supply matches customer demands. The need for responsive flexible schedules is obvious not only because there is a risk that some potential customers may turn away if a desirable schedule is not available, but also because the take-up of some services in a fixed schedule may be low and not profitable. We propose an optimisation model that quantifies customer satisfaction, which is maximised as one of the rail business criteria. This framework is a necessary tool for supporting decision-makers, through which a rail carrier can measure how well their customers are satisfied and the implications of satisfying these customers in terms of cost.

Solution approach. The container rail scheduling problem is complex and large and we need a method that can solve the problem effectively. Two goals in designing a solution method are:

1. It provides good quality of solution within reasonable time.
2. It is simple to implement and convenient to use.

We attempt to achieve these goals by investigating and extending existing methods and developing several new techniques to improve their performance. We propose a constraint-based local search algorithm incorporating a predictive choice model for solving the container rail scheduling problem. The solution algorithm is simple and convenient to use, whilst providing good quality of solution.

1.3 Thesis outline

Following this introductory chapter, Chapter 2 outlines frameworks for combinatorial optimisation. Chapter 3 describes the container rail scheduling model. Chapter 4 and 5 present the research on the investigation and construction of an efficient solution method for the container rail scheduling problem. Chapter 6 adapts this method for another combinatorial optimisation problem, the generalised assignment problem. Conclusions drawn from the research are given in Chapter 7.

In Chapter 3, the container rail scheduling problem is modelled as a constraint satisfaction problem. The rail business criteria and operational requirements are considered as soft and hard constraints respectively. A demand responsive scheduling model is proposed in which service supply matches or responds to customer demands and optimises on booking preference whilst satisfying railway operations.

Chapter 4 describes the constraint-based local search algorithm for solving the container rail scheduling problem. The constraint-based local search starts with random initial assignments

and uses a simple variable flip as a structure of local move. When all variables in the model are assigned a value, the total hard violation is calculated; a quantified measure of the violation is used to evaluate local moves. Different measures of the violation are also used in order to drive the search to the promising regions of the search space.

Chapter 5 develops a novel predictive choice model to improve the solution obtained by constraint-based local search alone. The predictive choice model is based on discrete choice theory and the random utility concept. Learning from search history, the model will predict a good choice of value for a variable. The variable will be fixed at its preferred value for a number of iterations determined by the magnitude of the preference measure. At this point, the propagation of consistency between variables is enforced, leading to intensified exploration of the search space.

Chapter 6 demonstrates the application of the proposed algorithm to the generalised assignment problem (GAP). A set of diversified feasible solutions to GAP is obtained by the constraint-based local search. The predictive choice model learns from the search history and predicts good assignments of jobs to agents. The search focuses more intensively on regions which promise to find better solutions. The performance of the algorithm is evaluated with two different benchmark problem sets.

Finally, Chapter 7 gives the conclusions, discusses the achievement of this research and suggests some future work.

Chapter Two

Literature review

2.1 Introduction

This chapter outlines the basic principles of the optimisation frameworks which could be applied to our container rail scheduling problem. Further discussion on related work will also be given in later chapters as appropriate. The optimisation frameworks discussed in this chapter are: integer programming, finite domain constraint programming, and local search. Integer and constraint programming can be considered as general-purpose optimisation methods, whereas local search may be viewed as an approach that can be tailored to many different combinatorial optimisation problems by adapting its simple conceptual components to the respective problem context.

The local search method is attractive, and often used, when proven optimal solutions may take too long to find. The effective performance of local search mainly depends on two strategies: diversification and intensification. Diversification drives the search to explore new regions so that the search space is fully covered, intensification focuses the search more

intensively on regions previously found to be good or promising to find an optimal solution. Good interplay between the diversification and intensification strategies is the critical issue in the design of local search methods.

This chapter is organised as follows: Section 2.2 outlines the frameworks for combinatorial optimisation. Section 2.3 reviews solution methods for rail scheduling. Section 2.4 reviews local search methods for constraint satisfaction problems, especially for those that lead to the development of our proposed method, which will be used in later chapters. Section 2.5 describes move strategies for local search and introduces concepts for the adaptive control used in local search. Finally conclusions are given in Section 2.6.

2.2 Frameworks for combinatorial optimisation

Many problems arising from diverse areas can be considered as combinatorial optimisation problems. Combinatorial optimisation problems are concerned with the efficient use or allocation of limited resources to meet desired criteria. In this section, we outline the frameworks for combinatorial optimisation problems which are related and applied to our rail scheduling problem (Chapter 3) as well as the generalised assignment problem which will be described in Chapter 6. Three optimisation frameworks will be discussed: integer programming (IP), constraint programming (CP), and local search. These frameworks are well established, comprising a variety of techniques, and many successful applications have been reported.

2.2.1 Integer programming

Combinatorial optimisation problems are considered as integer programming problems when the decision variables in the model are required to be integers. IP is used in practice for solving many industrial problems, for example in transportation and manufacturing: airline crew scheduling, vehicle routing, production planning, etc (Nemhauser and Wolsey, 1988).

IP branch-and-bound is concerned with finding optimal solutions to the IP problem. A general integer linear programming formulation is defined as:

$$z_{IP} = \min\{cx \quad : \quad x \in S\} \tag{2.1}$$

where: $S = \{Ax \geq b \quad : \quad x \geq 0\}$, and $x \in$ Integer

In branch-and-bound method, the original problem is divided into sub-problems, and sub-problems are created by restricting the range of the integer variables. For binary variables, there are only two possible restrictions, i.e. setting the variable to 0 or 1. Lower bounds are provided by the linear-programming relaxation to the problem, i.e. keep the objective function and all constraints, but relax the integrality restrictions to derive a linear programme. If the optimal solution to a relaxed sub-problem is integral, it is a feasible solution to the problem; and the optimal value can be used to terminate searches of sub-problems whose lower bound is higher. Many issues need to be considered to develop efficient branch-and-bound methods, such as the selection of branching variables and the node to develop next. In other words, strategies to explore the search tree need to be defined. Efficient branch-and-bound implementations may further add valid inequalities (cuts), which

are inferred from specific classes of constraints implicitly present in the original constraints (Mitchell, 2000). Using these components, in addition to efficient algorithms for solving and re-solving LP relaxations, IP branch-and-bound provides a general and efficient technique for many combinatorial optimisation problems. A variety of efficient commercial branch-and-bound solvers are available on the market, e.g. CPLEX, LINDO, XPRESSMP, MINTO (Nemhauser and Wolsey, 1988).

2.2.2 Constraint programming

Constraint programming or finite domain constraint programming (CP) has attracted much attention amongst researchers from many areas because of its potential for solving hard combinatorial optimisation problems. Real-life problems tend to have a large number of constraints, which may be hard or soft. Hard constraints require that any solution will never violate the constraints, whereas soft constraints are more flexible, constraint violation is tolerated but attracts a penalty. Naturally, combinatorial optimisation problems can be thought of as constraint satisfaction problems (CSP). A CSP is typically defined in terms of (1) a set of variables, each ranging over a finite discrete domain of values, (2) a set of constraints, which are relations over subsets of the variable domains. The problem is to assign values to all variables from their domains, subject to the constraints (Tsang, 1993).

When combinatorial problems are solved by CP, the constraint store stores information on the constrained variables in the form of the set of possible values that a variable can take. This set is called the current domain of the variable. Computation starts with an initial domain for each variable as given in the CSP-model. Some constraints can be directly entered in the constraint store by strengthening the constraint on a variable, e.g. the

constraint $x \neq y$ can be expressed in the constraint store by removing the current value of y from the domain of x.

Another component in CP is called propagators. Each propagator observes the variables given by the corresponding constraint in the problem. Whenever possible, it strengthens the constraint store with respect to the variables by excluding values from their domains according to the corresponding constraint, e.g. a propagator of constraint $x \leq y$ observes the upper bound and lower bounds of the domains of x and y. A possible strengthening consists of removing all values from the domain of x that are greater than the upper bound of the domain of y. The process of propagation continues until no propagator can further strengthen the constraint store, i.e. the constraint store is said to be stable. However, variables in many CSP problems typically cannot be reduced to a singleton domain. Therefore, the constraint store does not represent a solution and search becomes necessary.

Search for CSP solutions is implemented by choice points. A choice point generates a branching constraint c. From the current stable constraint store cs, two new constraint stores are created by adding c and $\neg c$ to cs respectively. Typically, the new constraint stores are not stable, and c and $\neg c$ trigger some propagators in their respective new stores. After stability is reached again, the branching process is continued recursively on both sides until the resulting store is either consistent or represents a solution to the problem. CP is best considered as a software framework for combining software components to achieve problem-specific tree search solvers. These components can be organised into three parts (Marriott and Stuckey, 1998):

1. Propagation: implements individual constraints by describing how the constraints can be employed to strengthen the constraint store.
2. Branching: selects branching constraints at each node of the search tree after all propagation has been done. Branching strategies define the size and shape of the search tree.
3. Exploration: describes which part of a given search tree is explored and in which order.

CP has seen much success in a variety of application domains, e.g. planning and scheduling. Various techniques have been integrated into constraint programming, propagators and branching strategies to make the solving algorithm powerful (Prosser, 1993; Jussien and Lhomme, 2002). Example of general CP solvers are Oz, CHIP, ECLiPSe, and ILOG (Marriott and Stuckey, 1998).

2.2.3 Local search

Many combinatorial optimisation problems are NP-hard, i.e. may not be solved within polynomial computation time (Nemhauser and Wolsey, 1988). This implies that proven optimal solutions may take too long to find, at least for large instances. However, sub-optimal solutions are sometimes easy to find. Therefore, there is much interest in local search that can find good solutions with reasonable times. Local search methods have successfully been applied to many combinatorial optimisation problems. Local search can be described in terms of several basic components: a cost function of a solution to the problem, a neighbourhood function that defines the possible moves in the search space, and a control strategy according to which the moves are performed.

- Cost function: a combinatorial problem is defined by the set of feasible solutions and a cost (fitness) function that maps each solution to a quantified cost. The search algorithm is to find an optimal feasible solution, i.e. a feasible solution that optimises the cost function.
- Neighbourhood function: local search proceeds by making moves from one solution point to another. The set of points includes feasible solutions, but may also include infeasible solutions. Given a combinatorial problem, the neighbourhood function is defined by mapping from the set of points to its neighbours, i.e. the subsets of the set of points. A solution is locally optimal with respect to a neighbourhood function if its cost is not worse than the cost of each of its neighbours.
- Control strategy: defines how the search space is explored. For instance, a basic control strategy of local search is iterative improvement, i.e. one starts with an initial solution and searches its neighbourhood for a solution of lower cost. If such a solution is found, the current solution is replaced and the search continues. Otherwise, the algorithm returns the current solution, which is locally optimal.

A main problem of local search is local optima, i.e. points in the search space where no neighbour improves over the current point, but which may be far from the global optima. Many strategies have been proposed to overcome this problem. In many cases, non-improving local moves are accepted based on a probabilistic rule or based on the history of the search (Aarts et al, 1997).

2.3 Solution methods for railway scheduling

The solution methods for railway scheduling may be classified into three groups: heuristic

methods, mathematical programming methods, and meta-heuristic methods.

2.3.1 Heuristic methods

The early age of solving railway scheduling problems only relied on heuristic methods. Most heuristics at that time were similar to the methods used by manual schedulers. The refinement of the service plan (schedule) was made complementarily between a planner and a computer.

Crainic et al. (1984) proposed a tactical model for rail service planning. They decomposed the planning model into routing and scheduling models. A local search heuristic was used to solve each two sub-models of the problem in succession and to obtain a good feasible solution offering a rough framework for producing a rail schedule. Haghani (1989) used a heuristic decomposition technique for railway scheduling. The heuristic based on a special structure was used to solve the problem within a small network. However, his approach failed to solve larger problem instances.

Gualda and Murgel (2000) considered the train formation problem. The objectives are to maximise revenue from the transport of cargoes, to safeguard the relative priorities of cargoes, and supply the services with the minimum total operating cost under operational constraints. The heuristic begins with the formulation of direct trains that travel loaded from origin to destination and come back empty. This solution is then submitted to a refinement procedure to combine trains and minimise the movement of empty wagons, and the algorithm seeks a better use of the rolling stock. The heuristic incorporates a shortest path algorithm and a strategy based on the knapsack problem.

The heuristics used for railway scheduling were heavily problem-specific. A heuristic which works for one problem cannot be used to solve a different one, or cannot easily be adapted to new problem conditions. Purely heuristic methods for railway scheduling rarely appear nowadays. Often they are now used to gear up mathematical programming methods into a more flexible and general problem solvers.

2.3.2 Mathematical programming methods

Mathematical programming (MP) has been well known and developed in the operations research society for several decades. Most railway scheduling problems have been modelled based on mathematical formulations.

Keaton (1989) and Keaton (1992) used the Lagrangian relaxation method to simplify the rail routing and scheduling problem. He incorporated the train capacity, travel time and demand flow constraints into the objective function with Lagrangian multipliers. Relaxing the constraints allows him to decompose the problem into separable train demand flow problems. By relaxing the train capacity constraint, the demand flow problem can be viewed as a collection of shortest-path problems, one for each origin - destination pair. Using a dual adjustment approach, he arrived at an infeasible lower bound to the problems. Afterwards, he used a simple heuristic to obtain a feasible solution.

Schrijver (1993) considered the problem of minimising the number of train units of different types for an hourly train line in the Netherlands, given that the passenger's seat demand and train capacity constraint must be satisfied. The restriction on the transition between two compositions on two consecutive trips is that the required train units must be available at the

right time and the right station. Coupling and uncoupling restrictions related to the feasibility of shunting movements are ignored. He proposed an algorithm based on graph theory and integer programming. The algorithm concerns the circulation of different type of train units, which can be linked more together. It can be described as a multicommodity flow problem, and is solved using ideas from polyhedral combinatorics.

Newton et al (1998) considered the freight rail blocking plan problem. The objective is to choose the blocks to be built at each cargo yard and to assign sequences of blocks to deliver each shipment to minimise total mileage, handling cost, and delay costs. They developed a column generation approach in which attractive paths for each shipment are generated by solving a shortest path problem. They also disaggregated some of the constraints in the model to provide a tighter lower bound.

Newman and Yano (2000) considered the trains and containers scheduling problem. The objective is to minimise total operating costs, whilst meeting on-time delivery requirements. They formulated the problem as an integer programme. A decomposition procedure to find near-optimal solutions and a method to provide relatively tight bounds on the objective function values were proposed. Yano and Newman (2001) considered the container rail scheduling problem with due dates and dynamic arrivals. The objective is to minimise the sum of transportation and holding costs. They introduced a definition of a regeneration state, which derived from a strong characterisation of the shipment schedule within the regeneration interval properties of an optimal solution. The optimal assignment of customer orders to trains can then be found by solving a linear programme.

Kraft (2002) considered the shipment routing problem. He formulated the problem as a multi-commodity network flow problem, where each shipment is treated as a separate commodity. A Lagrangian heuristic was used to obtain a primal feasible solution by ranking all flows based on priority. Then the algorithm sequentially assigns flows on a shortest path based on adjusted link costs. A primal feasible solution was used to validate the quality of the dual prices by establishing their prices leading to a tight upper bound on the objective function.

MP incorporating heuristics is often used for many practical railway scheduling. Heuristics are used to enhance MP to obtain the optimal solution or near-optimal solution in a viable time. Most of MP relies on bound strategies, e.g. linear relaxation, linear duality, and Lagrangian relaxation. A good bound helps limit the size of the search. However, the heuristics and bound strategies depend on the presence of special structures in the model; the adaptation of which for new practical aspects might be non-trivial.

2.3.3 Meta-heuristic methods

Meta-heuristics are widely used to solve important practical combinatorial optimisation problems. Basically, a meta-heuristic is a top-level strategy that guides an underlying heuristic solving a given problem. That is, a meta-heuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate iteratively a complete (or incomplete) single solution or a collection of solutions. The subordinate heuristics are e.g. high- (or low-) level procedures, simple local search, or just a construction method. Meta-heuristics may use learning

strategies to structure information in order to find optimal or near-optimal solution effectively (Osman and Kelly, 1996; Glover and Laguna, 1997).

2.3.3.1 Simulated annealing

Simulated annealing is a meta-heuristic technique for combinatorial optimisation problems which is designed as a simple and robust algorithm (Kirkpatrick, 1984). The term simulated annealing derives from the physical process of heating and cooling a substance to obtain a strong crystalline structure. A simulated annealing algorithm repeats an iterative procedure that looks for better solutions, whilst offering the possibility of accepting, in a controlled manner, worse solutions. This second feature allows the algorithm to escape from the local optima.

Huntley et al (1995) used simulated annealing to solve a railway scheduling problem at the CSX transportation company. They used a perturbation move operator that inserts or deletes a stop from the route and adjusts the departure times of the trains. The computational results showed that the algorithm was useful for analysing a variety of scenarios, and producing train schedules having similar properties to those of solutions in use by the CTX company, but with a smaller cost.

Brucker et al. (1999) used simulated annealing for freight rail routing. They defined neighbourhoods using the ideas from the network simplex method for min-cost flow problems. Afterwards, they proposed a two-phase local search method based on simulated annealing which executes a series of local search applications to single commodity problems. In the first phase, the algorithm tries to cover a large part of the search space and to identify

a good solution. In the second phase, the algorithm starts with the best solution found in the first phase and tries to improve this solution. They applied the two phases several times (multiple restarts).

2.3.3.2 Genetic algorithm

A genetic algorithm is a heuristic search algorithm premised on the evolutionary ideas of natural selection and genetics (Holland, 1975). The algorithm starts with a set, called a population, of solutions (represented by chromosomes). Solutions from one generation are taken and used to form a new population. Solutions which are then selected to form new solutions (offspring) are selected according to their fitness; the more suitable they are the more chances they have to reproduce.

Salim and Cai (1997) used a genetic algorithm to schedule rail freight transportation. The algorithm begins with randomly generating the initial population, and then finds the arrival time and departure time of each train at every loop by using stopping and starting matrix schedules and evaluates the cost of the population. Afterwards, the algorithm performs a crossover operation on the randomly chosen individuals to yield two new strings and replace the duplicates in the population with the newly formed individuals. The algorithm terminates if the best individual in the population has not changed for a predefined number of iterations.

Arshad et al (1998) used a genetic algorithm combined with constraint programming for container transport chain scheduling. The objective function is to minimise the empty containers between terminals, depots, and clients under operational constraints. Constraint

programming was used to compute feasible solutions on a subset of search space. A genetic algorithm was used to explore the space formed by solutions provided by CP, and to perform optimisation. The feasibility of the solutions was defined intrinsic to the chosen representation and integrated within the creation of the chromosomes in the different steps (initialisation, crossover, and mutation), and within the fitness.

2.3.3.3 Tabu search

Tabu search is a heuristic method proposed by Glover (1986) for solving combinatorial optimisation problems. Tabu search allows acceptance of non-improved solutions in order to avoid being trapped in local optima. To prevent going back to recently visited solutions, a memory scheme is used to record the moves made in the recent past of the search. This recorded search history is usually represented by a tabu list of moves, which are forbidden for a certain number of iterations.

Marin and Salmeron (1996) used tabu search to plan freight rail services. The algorithm was based on the decomposition of the planning model in two problems; routing for the freight cars and grouping of cars in the trains. Heuristic routing and sequential loading algorithms were proposed. In tabu search, recency-based memory with frequency was used to prevent the search going back to recently visited solutions. They also compared tabu search with simulated annealing and descent methods. The comparison amongst these methods was made with the help of statistical analysis. They assumed the hypothesis that the distribution of local minima can be represented by the Weilbull distribution in order to obtain an approach to the global minimum and a confidence interval. The global minimum estimation was used to compare the heuristic methods.

A combination of genetic algorithm and tabu search was used by Gorman (1998) to solve the rail scheduling problem. In GA, the population was formed by all possible train schedules. Every time an individual schedule is generated, its fitness (total operating cost) is evaluated. Mutations are obtained by either adding or deleting a train, or by shifting a train to an earlier or a later time in the schedule. To improve the performance of the genetic algorithm, each solution is cloned and modified with a tabu search algorithm, thus simulating the use of knowledge based mutation operators. However, implementing random starting solutions and simple tabu moves still suffers from misdirected search.

Much attention has been focused on meta-heuristic methods as conceptually simple, domain-independent frameworks for solving railway scheduling problems. However, classical meta-heuristics, applied totally independently of problem domain knowledge, rarely work well for real industrial problems. Often meta-heuristics are enhanced by incorporating intensive domain-knowledge, and good solutions may be obtained by fastidious tuning of various parameters. The meta-heuristics then lose their appeal as general solution approaches and quickly become algorithms highly specialised for the given problem. Meta-heuristics may be hybridised to be more effective. However, the resulting algorithms would be complex, and they often still have to exploit domain knowledge to be effective.

2.4 Local search for constraint satisfaction problems

The satisfiable problem in propositional logic (SAT) is to decide whether a given Boolean formula is satisfiable and was the first problem proved to be NP-complete (Cook, 1971). To explain the satisfiable problem, the following terms are given:

- A literal is a propositional variable or its negation, e.g. x or $\neg x$
- A clause is a disjunction of literals, e.g. $(\neg x \vee y \vee z)$
- A formula in conjunctive normal form (CNF) is a conjunction of disjunctions, e.g. $(\neg x \vee y \vee z) \wedge (\neg a \vee b \vee \neg c) \wedge \ldots$

The goal of the SAT problem is to find an assignment of values to variables, if one exists, where all clauses are satisfied or to prove it is unsatisfiable if no valid assignment exists. MAX-SAT and weighted MAX-SAT are the optimisation variants of SAT. Given a set of clauses, MAX-SAT is the problem to find a variable assignment that maximises the number of satisfied clauses. In weighted MAX-SAT, a weight is assigned to each clause and the goal is to maximise the weight of the satisfied clauses. Alternatively the goal could be defined as to minimise the weight of the unsatisfied clauses. In MAX-SAT and weighted MAX-SAT, all clauses need not be satisfied and it may be considered as the unsatisfiable problem. Note also that in case the weights are not specified, MAX-SAT can be called unweighted MAX-SAT and therefore, when the term MAX-SAT is used in general, it refers to the general form of the problem including clause weights.

The SAT problem can be viewed as a 0-1 integer constraint problem, i.e. a Boolean clause (disjunction of literals) is translated into an arithmetic constraint. For instance, the clause $(x \vee y)$ would be translated to $x + y = 1$. For a constraint satisfaction problem (CSP), if the variable domain is Boolean and the constraints are expressed in conjunctive normal form, then the CSP is equivalent to the SAT problem; in other words, CSP is a generalisation of SAT in two aspects which are the domains of variables and the arithmetic constraints. There are many local search methods for solving a constraint satisfaction problem. An overview of these methods is given in Hoos and Stutzle (2004).

Hill-climbing is the local search method introduced in the past decades for a hard combinatorial problem (Nilsson, 1980). It starts from a randomly generated assignment of variables. At each step, it changes the value of some variables in such a way that the resulting assignment satisfies more constraints. If a strict local minimum is reached then the algorithm restarts at another randomly generated point. The algorithm stops when all constraints are satisfied, or the computational resource is exhausted. However, the hill-climbing algorithm has to explore all neighbours of the current state in choosing the move. To avoid this problem, heuristics are introduced as described next.

2.4.1 Min-conflict heuristic

The min-conflict heuristic has been introduced as a method for solving constraint satisfaction problems (Minton et al, 1992). This heuristic chooses randomly a variable in a violated constraint, and then picks a variable value which minimises the number of violated constraints. If no such a value exists, it picks randomly one value that does not increase the number of violated constraints (the current value of the variable is picked only if all the other values increase the number of violated constraints). The min-conflict heuristic allows sideway moves, i.e. the current solution is allowed to move to another solution with the same solution cost. This lets the procedure traverse plateaus in the solution landscape. By doing this, the search algorithm can find its way off the plateau and continue the gradient descent. The min-conflict heuristic is briefly outlined as follows:

Given: a set of variables, a set of constraints, and an assignment of a value for each variable; two variables conflict if they both occur in a constraint which is violated at the current point.

Procedure: select a variable that is in conflict, and assign it a value that minimises the number of conflicts.

Empirical tests obtained from Minton et al (1992) using the min-conflict heuristic for hill climbing showed that the heuristic obtained similar results to an existing neural network method. The results also showed that the local search min-conflict heuristic works well on some problems, e.g. the *n*-queens problem, graph colouring problem and the real world problem of scheduling the Hubble space telescope. However, the min-conflict heuristic can easily be trapped in local minima.

Since the min-conflict heuristic alone cannot overcome the problem of local minima, several techniques have been introduced to solve this problem. These methods often diversify the search and can be categorised into two types: 1) methods that add randomness, such as noise, using random walks (Selman and Kautz, 1993), simulated annealing and 2) methods that restructure the neighbourhood, that is the search is not allowed to move to some points for a number of iterations, resulting in a smaller neighbourhood size, e.g. Tabu search (Glover and Laguna, 1997).

2.4.2 GSAT

Local search for the SAT problem became popular when Selman at al (1992) introduced GSAT. The procedure of classical GSAT is shown in Figure 2.1.

From Figure 2.1, GSAT searches for a satisfying variable assignment A for a set of clauses C. Local moves are flips of variables which are chosen by *select-variable* according to a

randomised greedy strategy, i.e. choosing a variable that leads to the largest increase in the total number of satisfied clauses. The parameter *Maxflips* is used to determine the frequency of restarts that helps GSAT overcome local minima. The process continues until the parameter *Maxtries* is reached. Most local search algorithms for SAT (and also MAX-SAT) follow the procedure below and thus have a simple structure of the algorithm.

```
proc GSAT
  Input clauses C, Maxflips, and Maxtries
  Output a satisfying total assignment of C, if found
  for   i := 1 to Maxtries do
            A := random truth assignment
        for   j := 1 to Maxflips do
                  if A satisfies C then return A
                      P := select-variable (C, A)
                      A := A with P flipped
        end
  end
  return "No satisfying assignment found"
end
```

Figure 2.1: Pseudo code for GSAT procedure (Selman et al, 1992)

Although the parameter *Maxflips* helps GSAT overcome local minima, it does not completely eliminate this problem; because the algorithm can still become stuck on a plateau (a set of neighbouring states each with an equal number of unsatified clauses). Therefore, it is useful to employ mechanisms that escape from local minima or plateaus by making uphill

moves (flips that increase the number of unsatified clauses). The mechanism is for example GSAT with walk (Selman et al, 1994). The principle of GSAT with walk is outlined as: Given a random number r ($0 \leq r \leq 1$) and a fixed probability p. With probability p ($r \leq p$), the algorithm randomly picks a variable appearing in some unsatisfied clauses and flips its truth assignment. With probability 1- p ($r > p$), the algorithm randomly picks a variable from the list of variables that gives the largest decrease in the total number of unsatisfied clauses.

2.4.3 WSAT

WSAT (or Walk SAT) is based on ideas first published by Selman et al (1994) and it was later formally defined as a local search for SAT by McAllester et al (1997). WSAT makes flips by first randomly picking a clause that is not satisfied by the current assignment, and then picking (either at random or according to a greedy heuristic) a variable within that clause to flip. Therefore, whilst GSAT with walk can be viewed as adding "walk" to a greedy algorithm, WSAT can be viewed as adding greediness as a heuristic to random walk. There is a subtle difference between GSAT with walk and WSAT in the probability that a variable is chosen to be flipped. GSAT with walk maintains a list (without duplicates) of the variables that appear in unsatisfied clauses, and randomly picks a variable from that list; thus, every variable that appears in an unsatisfied clause is chosen with equal probability. WSAT employs the two-step random process described above (first randomly picking an unsatisfied clause, and then picking a variable) that favours variables that appear in many unsatisfied clauses.

The main difference between local search algorithms for SAT and MAX-SAT is the strategy to select the variable to be flipped. This strategy is a key feature of SAT local search algorithms and is of critical importance for their performance.

2.4.4 Complex neighbourhoods

Most local search algorithms for SAT and MAX-SAT rely on the 1-flip neighbourhood. One exception is the 2 and 3-flip neighbourhood local search algorithm for MAX-SAT proposed by Yagiura and Ibaraki (1999). Since the computational time to examine the effects of 2 and 3-flips is high, they use a special data structure to speed-up as much as possible the neighbourhood evaluation. In addition, they propose restrictions to both neighbourhoods that allow pruning non-improving moves from the neighbourhood.

A different extension is the multi-flip approach by Strohmaier (1998) for SAT, where several independent flips, that is, only variables are flipped that do not occur in the same clause, are executed in parallel. The advantage of the approach is that the effect of independent flips is the sum of the single flips. The independent set of flips is determined by a neural network type architecture. In a similar idea, Roli (2001), and Roli and Blum (2001) proposed to perform flips in parallel without taking into account possible interactions amongst the variables. They divided the variables into k subsets of equal cardinality and, for each of the k sets, flip the variable having the highest score in the set; the evaluation of a variable flip is done as if all the other variables did not change.

2.4.5 Meta-heuristics for MAX-SAT

In this section, we review the meta-heuristics that are intended to solve the MAX-SAT problem. The following meta-heuristics use a 1-flip neighbourhood.

Battiti and Protasi (1997) proposed a history-based heuristic (reactive search) for MAX-SAT. The main idea is to have a two-phase approach consisting of a simple GSAT and a tabu search phase. The tabu search phase is run for a specified number of iterations. After the tabu search stops, GSAT is executed until the search is trapped in a local optimum and then the Hamming distance to the starting point is measured. Based on the resulting distance, the tabu tenure is adjusted and again the tabu phase is initiated.

Resende et al (1997) applied a greedy randomised adaptive search procedure (GRASP) to solve MAX-SAT problems. GRASP consists of two phases: construction and local search. The construction phase builds good feasible solutions (a set of satisfied clauses), whose neighbourhood is investigated until a local optimum is found during the local phase. The best total weight of satisfied clauses is kept as the result. Pardalos et al (1996) proposed a parallel GRASP for MAX-SAT problems. Each GRASP iteration is regarded as a search in some region of the feasible space and a number of processors perform searching in parallel. When the specified number of iterations has been reached, each processor gives the best solution found. The best solution amongst all processors is then identified and used as the solution of the problem.

Mills and Tsang (2000) proposed guided local search (GLS) for solving SAT and MAX-SAT problems. GLS uses a cost function including a set of penalty terms to guide the local search.

Each time local search gets trapped in a local optimum, the penalties are updated and local search is called again to maximise the modified cost function.

Variable neighbourhood search (VNS) was recently applied to the MAX-SAT problem (Hansen et al, 2000). VNS combines local search with systematic changes of neighbourhood in the descent and escapes from local optimum phases. The search explores increasingly far neighbourhoods of the current solution, and allows the exchange of the current best solution for a new one if and only if a better one has been found. Therefore, the favourable characteristics of the current solution are kept and used to obtain a promising neighbourhood, from which a further local search is performed.

2.5 Move strategies for local search

The fundamental principle of local search is to exploit the interplay between the diversification and intensification strategies, where diversification drives the search to explore new regions, and the intensification focuses more intensively on regions previously found to be good or promising to find an optimal solution.

2.5.1 Diversification

One of the main problems of all methods based on local search approaches is that there tend to be numerous local points in the search space, i.e. the local search algorithms tend to spend most, if not all, of their time in a restricted portion of the search space. The negative result of this fact is that, although good solutions may be obtained, one may fail to explore the most

promising regions of the search space and thus end up with solutions that are still pretty far from optimal.

Diversification is one of the strategies that try to reduce this problem. This can be done by forcing the search into previously unexplored areas of the search space. Search diversification may be based on history of the search, e.g. frequency- based memory in tabu search (Glover and Laguna, 1997) in which the algorithm records the total number of iterations since the beginning of the search for which various solution components have been present in the current solution or have been involved in the chosen moves. In cases where it is possible to identify promising regions of the search space, the search history can be refined to track the number of iterations spent in different regions.

Diversification techniques may be classified into three groups. The first, called restart diversification, involves assigning all variable values or forcing a few rarely used components in the current solution or the best known solution and restarting the search from this point. This technique is used, for example, in multi-restarts in GRASP (Feo and Resende, 1995). The second technique integrates the diversification procedure directly into the regular searching process. This is achieved by perturbing or biasing the evaluation of possible moves by adding to the objective a small term related to a component of search history. Examples of this technique are long-term memory in tabu search (Glover and Laguna, 1997), perturbation and bias sampling in iterated local search (Lourenco et al, 2002). The last diversification techniques are heuristic methods that use multi-neighbourhood structures. For a given combinatorial optimisation problem, several neighbourhood structures may be used to diversify the search space and enable a

convergence of the search space. A decision on which neighbourhood is to be chosen in what sequence during the search is an important strategy.

The diversification strategy in SAT local search is often achieved by some noise strategies, e.g. the random walk (Selman and Kautz, 1993; Selman et al, 1994), that randomly picks a value of some variables.

Ensuring proper search diversification is a critical issue in the design of local search methods. It should be addressed with care fairly early in the design phase and revised if the results obtained do not reach expectations.

2.5.2 Intensification

The purpose of the intensification strategy is to focus the search on promising regions of the search space in order to make sure that the best solutions in these regions are found. However, intensive investigation on the search space is computationally expensive; very often the local search algorithm would stop the normal searching process to perform an intensification phase from time to time.

Search intensification is used in many local search implementations, but it is not always necessary (Gendreau, 2002). This is because there are many cases where the search performed by the normal searching process is thorough (good) enough. Therefore, there is no need to spend time exploring more intensively the portions of the search space so that the computational effort is spent more effectively. For example, in tabu search, intensification can be carried out by giving a high priority to the solutions which have common features

with the current solution. This can be done with the introduction of an additional term in the objective function; this term will penalise solutions distant from the present one. This is done during a few iterations and after this it may be useful to explore another region so that the diversification strategy will be used next.

The intensification strategy in local search for CSP has rarely been addressed. The intensification is carried out by consistency techniques but mostly embedded in systematic tree search algorithms, e.g. backtracking algorithm (Kondrak and Beek, 1997). At each tree node, consistency is enforced with respect to the current variable assignments. As further assignments are made, the problem is divided into sub-problems since more of the original variables have fixed assignments within which consistency is enforced. Backtracking is called when any variable domain becomes empty as a result of consistency enforcing based on the current assignments, i.e. the sub-problem is inconsistent given these assignments. As consistency enforcing is also computationally costly, there is always a debate on the trade-off between how much consistency is maintained during the search and the quality of the solution.

2.5.3 Move acceptance criteria

In many combinatorial optimisation problems, constraints often restrict the searching process too much and can lead to low quality of solutions. This occurs, for example, in our rail scheduling problem and many other problems where the resource capacity is too tight to allow assigning demands (resource consumed) effectively between resources. In such cases, allowing non-improving moves (or relaxing constraints) is an attractive strategy. This is

because it creates a larger search space that can be explored with simple structures of local move.

Move acceptance can be carried out by several strategies. The simplest strategy is to always allow improving and non-improving moves. The second simple one can be implemented by dropping selected constraints from the search space definition and adding to the objective weighted penalties for constraint violations. However, this raises the issue of finding correct weights. An interesting way of tackling this problem may use self-adjusting penalties (Frank, 1997), i.e. penalty weights are adjusted dynamically on the basis of the recent history of the search: weights are increased if only infeasible solutions were encountered in the last few iterations, and decreased if all recent solutions were feasible. Penalty weights can also be modified systematically to drive the search to cross the feasibility boundary of the search space and thus induce diversification.

Another strategy is probabilistic move acceptance as used, for example, in simulated annealing. At the beginning of the search, a high probability of accepting any local moves is used whether the algorithm improves the solution or not. At some later iterations, the process is done with respect to a probabilistic acceptance function based on parameter called a temperature. The temperature is decremented until it is small and therefore only few non-improving moves are accepted. The way temperature is controlled is referred to as the cooling schedule.

2.5.4 Adaptive control

An adaptive mechanism is always an attractive feature for a better control of a local search algorithm. For example algorithms using adaptive mechanisms are: adaptive tabu search

(Glover and Laguna, 1997), adaptive simulated annealing (Lester, 1996), variable neighbourhood search (Hansen and Mladenovi, 2001), adaptive noise for SAT local search (Hoos, 2002), etc. Adaptive techniques may range from a simple automated tuning parameter to a complex learning mechanism. The formulation and application of the adaptive techniques are also very different, depending on where the techniques are used and the state of the search.

These adaptive algorithms do not depend on specific designs of heuristics and do not need many tuning parameters. Instead they learn from the history of the search in order to control the search adaptively. For example, Horvitz et al (2002) proposed a Bayesian learning technique for solving hard CSP and SAT problems. The algorithm explicitly learns from the search history and predicts the runtime for restarting policies in randomised search.

In our research a predictive choice learning model is proposed in order to inform the algorithm when the search space needs to be explored intensively and in which regions. Further discussion on this technique is given in Section 5.2.4 of Chapter 5.

2.6 Conclusions

This chapter outlines some frameworks for combinatorial optimisation, which can be mainly categorised into three groups: integer programming, finite domain constraint programming, and local search. Integer and constraint programming can be considered as general-purpose optimisation methods, whereas local search may be viewed as an approach that can be tailored to many different combinatorial optimisation problems by adapting its simple conceptual components to the respective problem context.For large-scale combinatorial

optimisation problems, local search may be used to find good solutions in reasonable time. Local search can be described in terms of basic components: a cost function of a solution to the problem, a neighbourhood function that defines the possible moves in the search space, and control strategy according to which the local moves are performed. The fundamental principle of local search is to exploit the interplay between diversification and intensification. Move acceptance criteria are also of importance for the performance of local search, in particular when simple structures of local move are used.

The research presented in this thesis proposes an effective solving algorithm for the container rail scheduling problem based on ideas discussed in this chapter, primarily those of local search. The container rail scheduling problem is formulated in Chapter 3 as a CSP. The solution method and its extension will be discussed in Chapter 4 and Chapter 5.

Chapter Three

Modelling the container rail scheduling problem

3.1 Introduction

There are many frameworks to represent and describe a container rail scheduling problem (Crainic and Laporte, 1997; Cordeau et al, 1998). However, what we are interested in is not only the framework for the representation of the problem but also an effective way to solve the problem. As the container rail scheduling problem is typically complex and large, a potential computational difficulty arises in solving such a problem. Therefore, research in this area needs to consider both how to model and to solve the problem.

Many real life problems could naturally be represented by constraints and the satisfaction of these constraints provides a solution for the presented problem. In our container rail scheduling problem, train capacity, service restrictions, and some customer requirements are modelled by hard constraints, whilst the objectives: minimum number of trains, maximum customer satisfaction, and minimum timeslot operating costs are modelled by soft

constraints. We can therefore formulate the container rail scheduling problem as a constraint satisfaction problem. Then, we present a constraint-based local search algorithm to solve this class of constraint satisfaction problem (described in Chapter 4).

This Chapter is organised as follows: Section 3.2 describes the problem's characteristics and assumptions. Section 3.3 describes the techniques to quantify the customer satisfaction on a rail schedule. Section 3.4 presents a formulation of the container rail scheduling problem. A generalised cost function is presented in Section 3.5, and finally conclusions are given in Section 3.6.

3.2 Problem description and assumptions

In the past, the transportation of rail freight was considered not to be an efficient mode of transport, particularly in terms of physical accessibility and cargo handling. Since the advent of containerisation in the mid 1940s, rail carriers have gained higher profitability by tailoring containerised freight and have become more competitive with other inland transport providers.

Container rail service differs from conventional freight rail in several important aspects. Because of the high costs of container handling equipment, container rail networks have relatively few and widely spaced terminals. Networks with around ten terminals are common and the network flows are relatively simple, as illustrated in Figure 3.1. A typical container makes few or no stops and may be transferred between trains only up to a few times on its journey. In addition, small lot sizes of shipment, frequent shipment, and demand for flexible service are important characteristics in the transportation of rail containers.

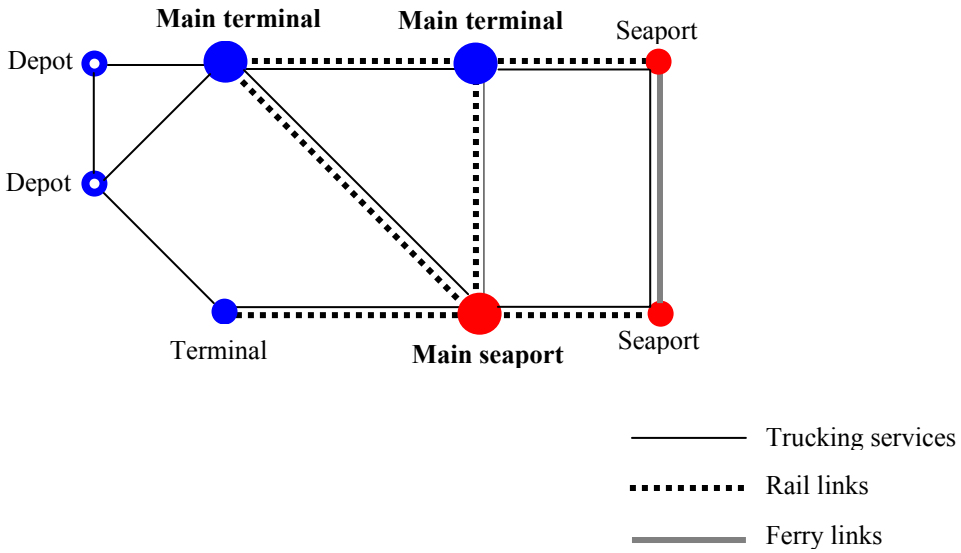

Figure 3.1: A typical container terminal network

It is also noted that container rail services are independent of one another in the sense that demands for a container movement in a specific route do not interact with the demands in any other routes (services). In addition, complex networks are not practical for customs procedures as containerised cargoes, in general, are moved within an international context.

Even though container traffic has increased, the increase in market share of rail transport, particularly in short-haul and medium-haul, has not been successfully achieved. Therefore, there have been efforts to investigate the factors influencing modal choice. The results have shown that the frequency and reliability of service are the main factors influencing shippers' decisions on the choice of transport mode (Indra-Payoong et al, 1998).

A rail carrier's profitability is heavily influenced by the railway's ability to construct schedules for which supply matches customer demands. For the transportation of containerised freight, shippers (customers) can often choose between rail and truck. A need

for responsive flexible schedules may become obvious not only because there is a risk that some potential customers may turn away if the customer's preferred itinerary is not attainable, but also because the take-up of some services in a fixed schedule may be low and therefore not profitable. In order to construct a demand responsive schedule, a rail carrier needs to engage in a decision-making process with multiple business criteria and a number of operational constraints, which is very challenging. There is a large body of literature on freight rail scheduling, using diverse modelling structures. A recent survey by Cordeau et al (1998) suggests most of them cater for fixed schedules. However, our proposed model incorporates challenging practical situations which involve:

1. Non-uniform arrivals with distinct target times, i.e. not all containers are available at the beginning of the planning time horizon and must be treated as distinct customer bookings.
2. A demand responsive service providing the flexible schedules
3. A probabilistic decrease in customer satisfaction with deviation from target time

A few attempts have been made to generate flexible train schedules, which may be categorised into two types according to how the overall demand is met. Huntley et al. (1995), Gorman (1998), and Arshad et al. (2000) aggregate customer demands with minimum operating costs through flexible scheduling. They do not propose to meet individual demands. Newman and Yano (2000), Yano and Newman (2001), and Kraft (2002) share the same spirit of our study by being responsive to individual demand. Their models satisfy the operational constraints fully for each customer. In contrast, our framework models customer satisfaction, computed from preferred and alternative booking time ranges, which is considered as one of the rail business criteria. Therefore, some customers might not be given

their most preferred booking time range. This framework is a natural one for supporting decisions as a rail carrier can measure how well their customers are satisfied and the implications of satisfying these customers in terms of cost.

We consider the container rail service from a container seaport to an inland container depot (ICD) in which the weekly schedule is provided and revised every week. Once containers arrive at the seaport, they can be transported to their final destinations by rail or truck via an inland container depot, or directly by truck. This study assumes an advance booking scheme as illustrated in Figure 3.2. It also assumes that all containers are homogeneous in terms of their physical dimensions, and they will be loaded on trains ready for any scheduled departure times. Note that we consider a standard container, which is measured in Twenty-Equivalent Unit (TEU).

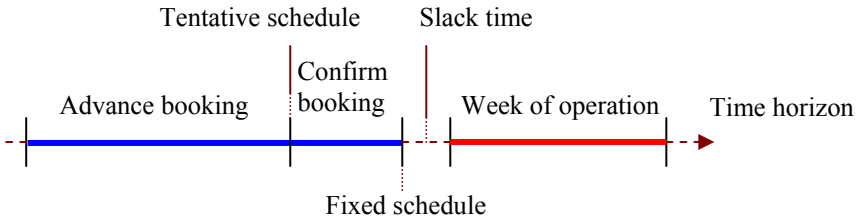

Figure 3.2: A short-term advance booking scheme

The day is divided into hourly slots for booking and scheduling. Customers are requested to state a preferred booking time range (or an earliest booking time range) in advance. A number of alternative booking time ranges for each shipment may be specified, which might be judged from experience or estimated by the customer's delay time functions. These alternatives not only help a rail carrier consolidate customer demands to a particular train

service with minimum total costs, but also provide flexible departure times for the customer's transport planning strategy.

A preferred departure time range and each alternative booking time range may cover a few hours, which is illustrated in Table 3.1. This happens in practice because the service time needed to move containers from the loading point of a containership to the train container platform may vary. In addition, customers may have to allow more time for unexpected delays.

Departure Timeslot	Shipping companies (customers)				
	Evergreen	Mearsk	P&O Nedlloyd	…	Mitsui OSK
.			P		
Sat: 0900		P	A_1		
Sat: 1000	P	P	A_1		
Sun: 1500		A_1	A_2		P
Sun: 1600	A_1	A_1	A_2		P
.	.	.	.	…	.
Wed: 1100			A_3		A_1
Wed: 1200	A_2		A_3		A_1
Wed: 1300	A_2				A_1

Table 3.1: An example of potential departure time ranges for customer

In Table 3.1, P is a preferred departure time range, A_1 is the first alternative departure time range, A_2 and A_3 are the second and the third alternative departure time range respectively.

Amongst these alternative departure time ranges, a customer has a more satisfaction with a departure time in A_1 than in A_2 and with a departure in A_2 than in A_3. That is, if one customer's alternative departure time range is not possible, the next alternative departure time range is accepted but with less satisfaction. Blanks " " denote infeasible departure times for the customer as container handling services may not be available either at the terminal of departure or destination terminal, or at both ends.

It is noted that there may be some customers that book the container rail service close to the end of a week; therefore their alternative booking time range may fall into the following week. The proposed model only takes the booking time ranges for those customers which fall within the schedulable week and the other alternatives are not considered directly.

3.3 Customer satisfaction

In a highly competitive market, assessing customer satisfaction with the transport service is of great importance to a container rail carrier. A rail carrier could take advantage of a knowledge of customer satisfaction to improve its service and to strengthen its competitive position with respect to the other transport services. A rail carrier could increase the quality of service and market share by tailoring a service that satisfies individual customers. The rail schedule may be just one of the mode-choice decision factors including cost, travel time, reliability, safety, and so forth. As customers have different demands, it is hard to find a single definition of what a good quality of service is. For example, some customers may be willing to tolerate a delayed service in return for sufficiently low total shipping costs.

3.3.1 Rail schedule factor

We investigate customer satisfaction with respect to the rail schedule factor. To acquire the customer satisfaction data, face-to-face interviews were carried out by the author. The outline of the interview is tabulated in Table 3.2.

This survey includes 184 customers currently using both rail and trucking services or using only rail but with the potential to switch their shipment to truck in the future. The containerised cargo is classified into four categories as follows:

1. Cargo type I (perishable consumer goods): food and beverages, dairy products, fruits and vegetables (24 customers)
2. Cargo type II (durable consumer goods): household products, and furniture (52 customers)
3. Cargo type III (intermediate products and raw materials): textile fibres, tobacco leaves, paper and paperboard, chemicals (67 customers)
4. Cargo type IV (capital goods and others): iron and steel, metal manufacture, non-electrical machinery and parts, construction materials (41 customers)

Shipping information
▪ Type of company (shipping line, freight forwarder, MTO, .etc) ▪ Type of container and commodity value per ton ▪ Container density measured ▪ Shelf life of the commodity in days ▪ Annual container volume shipped ▪ Period of advance booking regularly used.
Modal characteristics* (shipping time)
▪ Arrival time at container port ▪ Discharging time at container port ▪ Waiting time at the discharging point ▪ Haulage time from the discharging point to the main terminal ▪ Waiting time at port terminal ▪ Loading time at train/truck terminal ▪ Travel time
Modal characteristics* (shipping cost)
▪ Freight rate (TEU-ton-km) and commodity rate factor ▪ Terminal storage cost at port terminal/shipside (TEU-ton/day) (day = a consecutive 24-hour period) ▪ Free time storage period at port terminal (days) ▪ Reduction rate if containers are moved from terminal/shipside within (day-percent) ▪ Terminal handling charge per TEU-ton ▪ Overhead cost for waiting time at port terminal/shipside (TEU-ton/day)
Bookings
▪ Preferred train departure time range ▪ Alternative departure time range I ▪ Alternative departure time range II ▪ Others

* Rail and truck are the two modes surveyed

Table 3.2: The outline of the survey interview

3.3.2 Satisfaction

Understanding and quantifying customer satisfaction benefits a rail carrier. The customer satisfaction is linked to the probability that a customer will select rail or truck for a mode of transport. If customer satisfaction with the rail service decreases, the probability of customer choosing rail will also decrease and this will reduce the demand for the rail transport in the future.

To quantify customer satisfaction, customer satisfaction functions are developed. These use customer characteristics, shipping information and modal characteristics as primary input data. Total shipping costs associated with movement by modes are expressed as a percentage of commodity market price or value of containerised cargo, expressed in price per ton. Average relative shipping costs of the containerised cargo from survey data (see Appendix A) and the market price are summarised in Table 3.3. The market price for each cargo type is estimated by the Department of Business Economics, Ministry of Commerce, Thailand (Ministry of Commerce, 2002).

We assume that all customers know a full set of shipping costs and can justify the modal preferences on a basis of accurately measured and understood costs. The freight rate may be adjusted by the relative costs that a customer may be willing to pay to receive superior service. For example, some customers may have higher satisfaction using a trucking service even if the explicit freight rate is higher; speed and reliability of the service may be particularly important if the containerised cargo has a short shelf life.

Cargo types/cost ($\times 10^3$ Baht /ton)	Cost /unit price		Market price	Modal cost (%)		
	Truck	Rail		Truck C_T	Rail C_R	ΔC C_T - C_R
Freight rate						
Type I	2.21	1.55	25.00	8.84	6.20	2.64
Type II	6.71	2.96	68.00	9.87	4.35	5.52
Type III	10.45	7.56	87.20	11.98	8.67	3.31
Type IV	0.95	0.21	13.00	7.30	1.62	5.68
Terminal handling charge						
Type I	0.28	0.51	25.00	1.12	2.04	-0.92
Type II	0.57	1.04	68.00	0.84	1.53	-0.69
Type III	1.18	2.06	87.20	1.35	2.36	-1.01
Type IV	0.03	0.08	13.00	0.23	0.61	-0.38
Terminal storage charges						
(Within free time storage)	0	0		0	0	0
Overhead cost						
(Within free time storage)	0	0		0	0	0
Total shipping costs						
Type I	2.49	2.06	25.00	9.96	8.24	1.72
Type II	7.28	4.00	68.00	10.70	5.88	4.82
Type III	11.63	9.62	87.20	13.34	11.03	2.31
Type IV	0.98	0.29	13.00	7.54	2.23	5.31

Table 3.3: Average modal cost for each transport mode

To determine customer satisfaction between modes, we assume that the difference between modal cost percentages, i.e. $\Delta C = C_T - C_R$, follows a normal distribution. The customer satisfaction is then derived from cumulative probability density functions (Appendix A). The customer satisfaction functions for the containerised cargoes are shown in Figure 3.3 - 3.6.

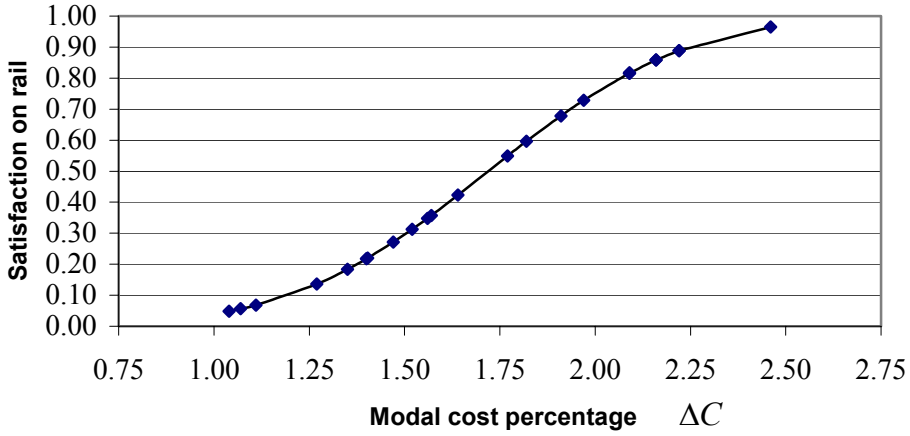

Figure 3.3: Customer satisfaction function of cargo type I

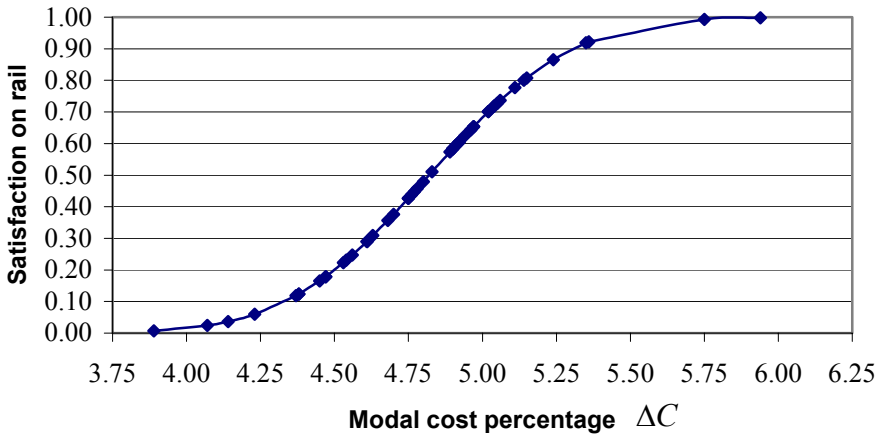

Figure 3.4: Customer satisfaction function of cargo type II

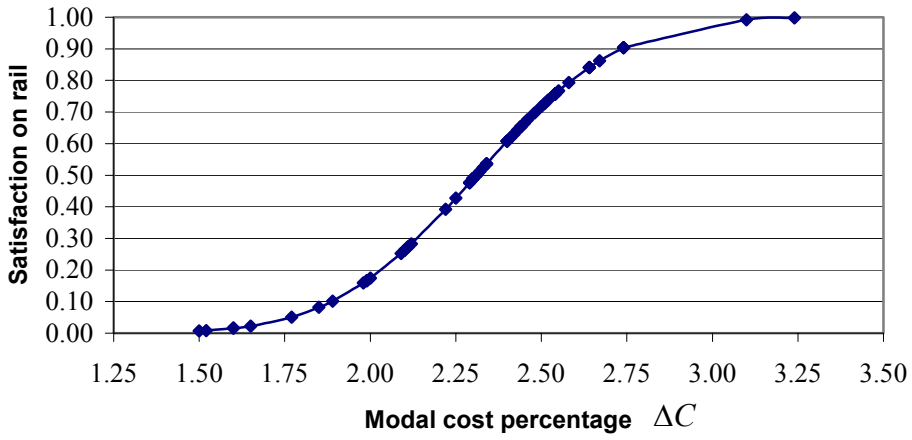

Figure 3.5: Customer satisfaction function for cargo type III

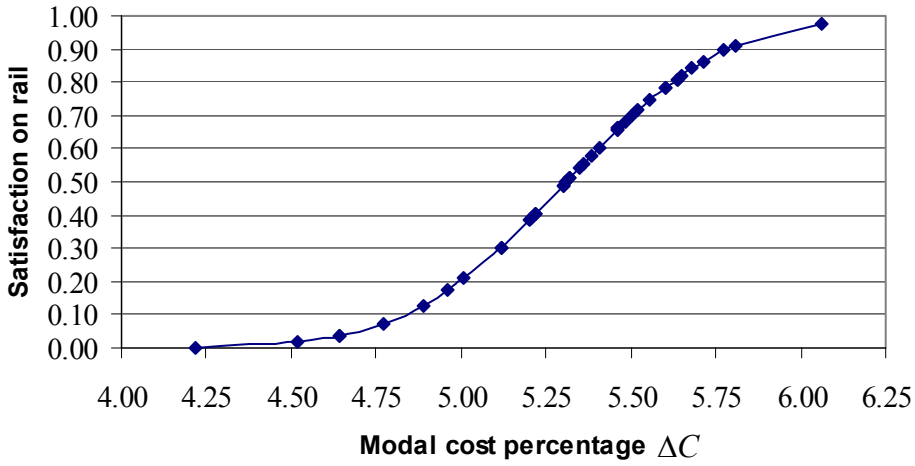

Figure 3.6: Customer satisfaction function of cargo type IV

From Figure 3.3 - 3.6, if there is no difference between the modal cost percentages, i.e. $\Delta C = 0$, customers tend to state their satisfaction on the service between rail and truck equally. A cargo that has a low value of ΔC has a high sensitivity in the total shipping costs. For instance, an arithmetic mean of $\Delta C = 1.72$ for customers shipping cargo I (see Appendix A);

when the transport of containers are delayed by rail, it will result in an increase in the total shipping costs. For customers shipping this type of cargo even a small cost increase can lower their satisfaction using the container rail service quite substantially. This is due to a high sensitivity in the total shipping costs. A probability of 0.5 in the customer satisfaction function indicates the lowest satisfaction level for the container rail service. If the satisfaction is below this level, customers may turn to use a trucking service instead; otherwise, they would tolerate the rail service. Nowadays, a rail carrier would try to keep the customer satisfaction above this level.

Once the satisfaction function has been developed, a customer's satisfaction measure can be obtained from the modal satisfaction probability. This probability could also be used to predict the market share between transport modes and to test the modal sensitivity when the rail schedule is changed.

The customer satisfaction measure is a probability of choosing the rail service and all customers have a satisfaction ranging from 0 to 1. Note that all customers currently using container rail service may already hold a certain level of satisfaction regardless of taking the quality of the rail schedule into account. Once the rail carrier has been chosen as a choice of transport mode and later the schedule is delayed, customers incur additional total shipping costs, i.e. terminal storage and overhead costs involved at the seaport. This would result in a decrease in customer satisfaction. An example of the calculation of customer satisfaction is shown in Table 3.4.

Shipping data	**Unit**	**Value**		
Cargo type	Type	IV		
Ship arrival time	Day: Time	Mon: 0900		
Discharging time	Hour	4		
Free time storage allowance	Day	3		
Reduction rate on terminal handling charges				
- Scheme I	Day - %	1 - 25%		
- Scheme II	Day - %	2 - 20%		
- Scheme III	Day - %	3 - 10%		
Bookings	Day: Time			
- Preferred time range (P)		Mon: 1500-1700		
- Alternative I (A_1)		Tue: 0900-1600		
- Alternative II (A_2)		Wed: 0900-1600		
- Alternative III (A_3)		Thu: 1600-2200		
Modal cost (%)	P	A_1	A_2	A_3
- Truck (C_T)	9.51	9.51	9.51	9.51
- Rail (C_R)	3.78	3.84	4.09	4.33
- ΔC	5.73	5.67	5.42	5.18
Satisfaction (w)	0.86	0.75	0.60	0.37

Table 3.4: Customer satisfaction

From Table 3.4, the customer has the modal cost percentages for truck C_T and rail C_R at the preferred booking time range equal to 9.51 and 3.78 (the first reduction rate on terminal handling charges is applied). Since the trucking service is always available, we use the same C_T for A_1, A_2, and A_3 for a comparison with C_R. C_R for A_1 and A_2 takes the reduction

rate 20% and 10% respectively. For A_3 no reduction is applied and the terminal storage charge is imposed. After getting the difference between modal cost percentages $\Delta C = C_T - C_R$, we can obtain the customer satisfaction by applying the value of ΔC into the customer satisfaction function (Figure 3.6), e.g. $\Delta C = 5.73$, we get the satisfaction $w = 0.86$. Alternatively, the satisfaction w can be obtained by the following function:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{3.1}$$

$$w = \int_{-\infty}^{\Delta C} f(x)dx \tag{3.2}$$

where: $f(x)$ is the normal probability density function that ΔC takes the value x, μ and σ are the mean and standard deviation of the total ΔC (Appendix A).

3.4 Problem formulation

There are often different ways of representing the same combinatorial optimisation problem and this should provide some advantages in developing solution procedures for such a problem. Since obtaining an optimal solution to a large-scale combinatorial optimisation problem in a reasonable amount of computer time may well depend on the way the problem is modelled, we need to consider both how to model and to solve the problem.

3.4.1 Integer programming formulation

Many combinatorial optimisation problems can be formulated as the problems in integer programming in which all decision variables are required to take integral values. We first model the demand-responsive container rail scheduling problem as an integer programme. We consider the day divided into hourly slots for weekly booking and scheduling. The following notation will be used:

Sets:

T : set of schedulable timeslots t

M : set of customers j

S_j : set of potential booking timeslots for customer j

C_t : set of potential customers for departure timeslot t

R : set of service restrictions for departure timeslots

Decision variables:

x_t : 1, if a train departs in timeslot t, 0 otherwise

y_{tj} : 1, if customer j is served by the train departing in timeslot t, 0 otherwise

Parameters:

FC is a fixed cost of running a train

w_{tj} : customer j satisfaction in departure timeslot t

N_j : demand of customer j (number of containers)

r_t : train congestion cost in departure timeslot t

g_t : staff cost in departure timeslot t

P_1 : minimum train loading (number of containers)

P_2 : capacity of a train (number of containers)

The IP formulation of the container rail scheduling problem is:

$$\text{Minimise} \quad FC\sum_{t\in T} x_t + \sum_{t\in T}\left(\sum_{j\in C_t} w_{tj} N_j y_{tj}\right) + \sum_{t\in T}(r_t + g_t)x_t \tag{3.3}$$

$$\text{Subject to} \quad \sum_{t\in S_j} y_{tj} = 1; \quad \forall j \in M \tag{3.4}$$

$$\sum_{j\in M} N_j y_{tj} \geq P_1; \quad \forall t \in T \tag{3.5}$$

$$\sum_{j\in M} N_j y_{tj} \leq P_2; \quad \forall t \in T \tag{3.6}$$

$$x_t \geq y_{tj}; \quad \forall j \in C_t, \forall t \in T \tag{3.7}$$

$$x_t = 0; \quad \forall t \in R \tag{3.8}$$

$$x_t, y_{tj} = \{0,1\}; \quad \forall t \in T, \forall j \in M \tag{3.9}$$

The objective is to minimise the generalised cost representing the operating costs and the virtual loss of future revenue. The first term in the objective function aims to minimise the number of trains on a weekly basis. The fewer trains, the greater reduction on operating costs a rail carrier can expect. The second term is to maximise the total customer satisfaction using values from a customer satisfaction functions (Figure 3.3 - 3.6). Each customer holds the

highest satisfaction at a preferred booking time range, the satisfaction then decreases probabilistically to the lowest satisfaction at the last alternative booking time range, i.e. departure later than the preferred booking time range would cause a decrease in the future demand, and the rail carrier is expected to take a loss in future revenue. For the evaluation of a schedule, the probability of customer satisfaction is then multiplied by demand N_j. The last term in the objective function aims to minimise the timeslot-operating cost. A rail carrier is likely to incur additional costs in operating a demand responsive schedule, in which departure times may vary from week to week. This may include train congestion cost and staff cost. The train congestion cost reflects an incremental delay resulting from interference between trains in a traffic stream. The rail carrier calculates the marginal delay caused by an additional train entering a particular set of departure timeslots, taking into account the speed-flow relationship of each track segment. The over-time cost for crew and ground staff would also be paid when evening and night trains are requested.

Constraints (3.4) ensure that no customer will be left uncovered and each customer can only be served by one train. These hard constraints are crucial for a rail carrier. This is because it does not make good business sense in a competitive market to offer a demand responsive service, requiring customers to state both their preferred, and a set of alternative, time ranges regarding their practical container operations (either at the terminal of departure or destination terminal, or at both ends) and business strategies and then to decline their business. In addition, customers' shipment cannot be split in multiple trains. This is because the destination terminal (inland container depot: ICD) located in the heart of the capital city is relatively small and all arrival containers must be stacked in the designated area. Expensive equipment is required to move the containers from the train to the ICD and later onto transport to the final destination. Multiple shipments are likely to result in container

stacks having to be "shuffled" to assemble a particular customer's load; thereby imposing substantial container handling cost.

Constraints (3.5) ensure that the demand assigned to a departure time slot must not be less than the minimum train loading P_1. Setting a minimum train loading ensures satisfactory revenue for a rail carrier and spreads out the capacity utilisation on train services. The carrier may want to set the minimum train loading as high as possible, ideally equal to the capacity of a train. Section 4.4 explains how sensible values for P_1 can be determined.

Constraints (3.6) ensure that the demand assigned to a departure time slot must not exceed the capacity of a train. These are hard operational constraints; if load exceeds the capacity, running the train can damage the locomotive engine and railway infrastructure, e.g. tracks. Note that in case of a single customer's demand being more than the capacity of a train, we allow splitting this demand over multiple trains and treat the demand as different sub-customers; however, this particular case rarely occurs in practice.

Constraints (3.7) ensure that if timeslot t is selected for customer j, a train does depart at that timeslot. On the other hand, if departure timeslot t is not selected for customer j, a train may or may not run at that time.

Constraints (3.8) are a set of banned departure times. The restrictions may be pre-specified so that a railway planner schedules trains to achieve a desirable headway or to avoid congestion at the container terminal.

Constraints (3.9) require that all decision variables in the model are binary.

The container rail scheduling problem was initially solved by an integer programming branch and bound method (CPLEX Solver, ILOG 2002). We failed to find an optimal solution within the time limit 12 hours, particularly when the minimum train loading P_1 is close to the capacity of a train. Since all decision variables in our container rail scheduling model are binary (constraints 3.9) and fractional solutions to linear relaxations are meaningless, tight bounds on the objective function value cannot be obtained and used to reduce the size of the search space.

Note that the matrix of coefficients in (3.4) consists of only zeroes and ones and that the constraints are equations. Thus there is a relaxation of the container rail scheduling problem to a set partitioning problem (SPP). Formally, the SPP is the problem of partitioning the rows i ($i = 1,...,M$) of a zero-one matrix (a_{ij}) by distinct subsets of the columns j ($j = 1,\ldots,N$) at minimal cost. Defining $x_j = 1$ if column j (with cost $c_j > 0$) is in the solution and $x_j =$ 0 otherwise. The SPP is

$$\text{Minimise} \quad \sum_{j=1}^{N} c_j x_j \tag{3.10}$$

$$\text{Subject to} \quad \sum_{j=1}^{N} a_{ij} x_j = 1; \qquad i = 1,2,3,...,M \tag{3.11}$$

$$x_j \in \{0,1\}; \qquad j = 1,2,3,...,N \tag{3.12}$$

Constraints (3.11) ensure that each row is covered (served) by exactly one column and (3.12) are the integrality constraints. The SPP is known to be *NP*-hard (Balas and Padberg, 1976); hence the container rail scheduling problem is *NP*-hard. In addition, the container rail scheduling problem incorporates other constraints that require the consistency between

different sets of decision variables which will make it more complex than SPP. Many current approaches for *NP*-hard problems focus on finding good solutions within a reasonable time using various local search heuristic methods.

3.4.2 Constraint-based modelling

Since the container rail scheduling problem is *NP*-hard, this implies that proven optimal solutions may take too long to find, at least for large instances. This leads us to the design of a local search that can find good solutions within reasonable run-time on a standard personal computer. As discussed earlier, what we are interested in is not only the framework for the representation of the problem but also an effective way to solve the problem. Therefore, we need to consider both how to model and to solve the problem. We model the container rail scheduling problem as a constraint satisfaction problem (CSP) and then introduce a constraint-based local search method for solving it (described in Chapter 4).

The principal difference between branch and bound based IP and CSP approaches to solving combinatorial optimisation problems is that:

- in IP, the integrality restrictions on the variables are relaxed, the search space is the continuous feasible region and the search is guided primarily by the objective function;
- in CSP, some or all of the constraints are relaxed but the integrality restrictions are enforced, the search space is simply defined by the domains of the variables and the search is guided by the need to gain feasibility.

Both approaches have met with success in a variety of applications, neither of the approaches being able to claim general superiority over the other. It is known that branch and bound searches can be very lengthy if integer feasibility is hard to achieve or if the optimal solution to the continuous relaxation is not a good guide to good quality solutions of the IP (Darby-Dowman and Little, 1998; Brailsford et al,1999). As noted in Section 3.4.1, this appears to be the case for the container rail scheduling problem.

A further important difference is that in IP, all constraints are global constraints, i.e. are always enforced, whereas constraint-based approaches can handle constraints locally, i.e. can decide which constraints to impose at various stages of the search. This allows different constraints to be given different 'priorities', which allows problem-specific knowledge to be exploited to guide the search. Constraints can also be represented more compactly in constraint-based approaches as there is no need to make them linear. Regarding the container rail scheduling problem, a constraint-based local method allows the search to move between feasible and infeasible regions of the search space in a simple and flexible way. In addition, each term in the objective function can be treated separately. This reduces the complexity in the design and evaluation of local moves.

Any local search method can be applied to CSP and is then sometimes called "a constraint-based local search method". In Chapter 2, we reviewed some local search methods (heuristic and meta-heuristic methods) for the railway scheduling problem and CSP. Many local search methods are conceptually simple, domain-independent frameworks; however these methods, applied totally independently of problem specific knowledge, rarely work well for real industrial problems. They are often enhanced by incorporating intensive domain-knowledge

and use complex local moves. Thus, they lose their appeal as simple and general solution methods and quickly become algorithms highly specialised for the given problem.

Our constraint-based local search (CLS) for the container rail scheduling problem is inspired by local search for the satisfiability (SAT) problem. An attractive framework of SAT local search is that the structure of the local move is simple and this may be appropriate for our problem. The development of CLS is described in detail through Chapters 4 and 5.

In a CSP, operational requirements are represented as hard constraints whilst optimisation criteria are handled by transforming them into soft constraints. This is achieved by expressing each criterion as an inequality against a bound on its ideal optimal value. As a result, such soft constraints are rarely satisfied.

A feasible solution for a CSP representation of the problem is an assignment to all decision variables in the model that satisfies all hard constraints, whereas an optimal solution is a feasible solution with the minimum total soft constraint violation (Winston, 1994; Walser 1999; Henz et al, 2000; Lau et al, 2001). For simplicity, we assume that the violation v_i of constraint *i* is linear and is defined as follows:

$$\sum_{j \in N} a_{ij} x_j \leq b_i \Rightarrow v_i = \max\left(0, \sum_{j \in N} a_{ij} x_j - b_i\right) \tag{3.10}$$

where a_{ij} are coefficients, b_i is a bound, x_j are constrained variables. Note that violations for other types of linear and non-linear constraints can be defined in an analogous way. Further, other ways of defining v_i are possible.

In this study, (3.10) is only used for soft constraints, it is later modified for hard constraints as described in Section 4.3.

3.4.2.1 Soft constraints

In the container rail scheduling model, the soft constraints are minimum number of trains, maximum customer satisfaction, minimum timeslot-operating cost.

Number of trains. This constraint aims to minimise the number of trains. The number of trains constraint is defined as:

$$\sum_{t \in T} x_t \leq \theta \tag{3.10}$$

where: θ is a lower bound on the number of trains, i.e. $\left\lceil (\sum_{j \in M} N_j) / P_2 \right\rceil$.

The violation of the number of trains constraint s_1 is

$$s_1 = \max\left(0, \sum_{t \in T} x_t - \theta\right) \tag{3.11}$$

Customer satisfaction. This constraint aims to maximise the total customer satisfaction. The customer satisfaction constraint is defined as:

$$\sum_{t \in T} \left(\sum_{j \in C_t} w_{tj} N_j y_{tj} \right) \geq \Omega \tag{3.12}$$

where: Ω is an upper bound on customer satisfaction, i.e. $\sum_{j \in C_t} W_j N_j$; W_j is the maximum satisfaction on a preferred booking time range for customer j .

The calculation of upper bound on customer satisfaction Ω is illustrated in Table 3.5.

Timeslot	Customer satisfaction							Sum
(t)	C_1	C_2	C_3	C_4	C_5	C_6	C_7	(Ω)
1		0.90			0.70			
2	0.95	0.90	0.76	0.85	0.70	0.84		
3		0.90	0.76	0.85				
4	0.95				0.70	0.63		
5		0.70	0.60			0.63		
6		0.70	0.60	0.60	0.50	0.63	0.91	
7	0.90		0.60		0.50		0.91	
N_j	10	20	15	8	5	25	16	
W_j	0.95	0.90	0.76	0.85	0.70	0.84	0.91	
$W_j \times N_j$	9.50	18.00	11.40	6.80	3.50	21.00	14.56	84.76

Table 3.5: The input data and the calculation of the upper bound Ω

The violation of the customer satisfaction constraint s_2 is

$$s_2 = \max\left(0, \Omega - \sum_{t \in T}\left(\sum_{j \in C_t} w_{tj} N_j y_{tj}\right)\right) \tag{3.13}$$

Timeslot-operating cost. This constraint aims to minimise the timeslot-operating cost. The timeslot-operating cost constraint is defined as:

$$\sum_{t \in T} (r_t + g_t) x_t \leq (\lambda + \delta) \tag{3.14}$$

where: $(\lambda + \delta)$ is a lower bound on the timeslot-operating cost, $\lambda = \sum_{t \in T_a} r_t$; T_a is the set of θ least train congestion costs, $\delta = \sum_{t \in T_b} g_t$; T_b is the set of θ least staff costs, θ is a lower bound on the number of trains.

The calculation of the upper bound on the timeslot-operating cost $(\lambda + \delta)$ is illustrated in Table 3.6. In this example, the lower bound on the number of trains θ = 3, the unit cost $\times 10^3$ Baht.

Timeslot (t)	Congestion cost (r_t)	Staff cost (g_t)	Timeslot cost ($r_t + g_t$)	Sum ($\lambda + \delta$)
1	0.28	3.51	3.79	
2	0.28	3.51	3.79	
3	0.76	1.95	2.71	
4	0.76	1.95	2.71	
5	1.34	3.12	4.46	
6	1.34	3.12	4.46	
7	1.34	3.12	4.46	
$\Sigma\ \theta$ least cost	1.32	7.02		8.34

Table 3.6: The input data and the calculation of the upper bound $(\lambda + \delta)$

The violation of the timeslot-operating cost constraint s_3 is

$$s_3 = \max\left(0, \sum_{t \in T} (r_t + g_t) x_t - (\lambda + \delta)\right) \tag{3.15}$$

Given soft constraint violations for the number of trains, customer satisfaction and timeslot-operating cost, the generalised cost for a rail carrier GC can be obtained as:

$$GC = FC(\theta + s_1) + FRs_2 + (\lambda + \delta + s_3) \tag{3.16}$$

where: FC is a fixed cost of running a train, FR is a freight rate per demand unit (ton-container), λ, δ are defined in Section 3.4.2.1; s_1, s_2, and s_3 are soft constraint violations for (3.11), (3.13), and (3.15) respectively.

3.4.2.2 Hard constraints

In the container rail scheduling model, hard constraints are coverage constraint, train capacity constraint, consistency constraint, covering constraint, and service restriction constraint. These constraints are sometimes called operational or required constraints and must be satisfied to ensure safety of service and practical operations. These constraints have been discussed in Section 3.4.1; however, some of these will be modified for a more efficient solving algorithm (described in Section 4.3.1).

3.4.2.3 Implied constraints

The soft and hard constraints completely reflect the requisite relationships between all the variables in the model, i.e. the operational requirements and business criteria. Implied constraints, derivable from the above constraints, may be added to the model. Whilst implied constraints do not affect the set of feasible solutions to the model, they may have computational advantage in the solution algorithm as they reduce the size of the search space (Proll and Smith, 1998; Smith et al, 2000). The covering constraint is one such constraint, being implied by (3.4).

Covering constraint. A covering constraint can be thought of as a set covering problem in which the constraint is satisfied if there is at least one departure timeslot x_t serving customer

j. This constraint favours a combination of selected departure timeslots that covers all customers. The covering constraint is defined as:

$$\sum_{t \in S_j} x_t \geq 1; \quad \forall j \tag{3.14}$$

3.5 Conclusions

In this Chapter, the container rail scheduling problem is modelled as a constraint satisfaction problem. We have presented a demand responsive scheduling model, in which service supply matches or responds to customer demands and optimises on booking preference whilst satisfying hard constraints. The advance booking scheme is assumed in order to help a rail carrier consolidate the demands with minimum total costs and provide flexible bookings for the customer's transport planning strategy.

A constraint-based modelling framework is used in which rail business criteria and operational requirements are formulated as soft and hard constraints respectively. The criteria are handled by transforming them into soft constraints, which is achieved by expressing each criterion as an inequality against a bound on its ideal optimal value.

The customer satisfaction with a rail schedule is quantified. It is computed from preferred and alternative departure time ranges in which the satisfaction decreases with deviation from target time in a probabilistic scale. The customer satisfaction is then maximised as one of the rail business criteria. Hence some customers might not be given their most preferred time ranges' bookings. This framework is to support decision-makers in which a rail carrier can

measure how well their customers are satisfied and the implications of satisfying these customers in terms of cost.

In Chapter 4, a constraint-based local search method is presented to solve the container rail scheduling problem.

Chapter Four

Constraint-based local search for the container rail scheduling problem

---

4.1 Introduction

As discussed in Chapter 3, the container rail scheduling problem has been modelled as a constraint satisfaction problem. A feasible solution for a CSP is an assignment to all constrained variables in the model that satisfies all hard constraints, and an optimal solution is a feasible solution with the minimum total soft constraint violation.

As the container rail scheduling problem is *NP*-hard, a potential computational difficulty arises in solving such a problem. In this chapter, we present a constraint-based local search algorithm to find a good solution within reasonable computational time. The algorithm uses a simple variable flip as a structure of local move. When all variables in the model are assigned a value, the total hard violation is calculated; a quantified measure of the violation

is used to evaluate local moves. A measure of constraint violation is used to drive the search to the promising regions of the search space.

This chapter is organised as follows. Section 4.2 describes procedure of the constraint-based local search algorithm. Section 4.3 describes how a constraint violation scheme is used to improve the quality of the container rail schedule. Section 4.4 describes the minimum train loading as an adaptive lower bound strategy. Computational results are shown in Section 4.5 and finally conclusions are given in section 4.6.

4.2 A constraint-based local search algorithm

The constraint-based local search algorithm (CLS) is inspired by local search for the satisfiability (SAT) problem. An attractive framework of SAT local search is that the structure of the local move is simple. There are some similarities between the ideas used in CLS and SAT local search. Both methods use a simple variable flip as a structure of local move and employ a randomised strategy for the selection of constraints and variables to explore. However, many aspects of CLS are different from SAT local search. These are described in the following sections.

4.2.1 The main loop

For the container rail scheduling problem, we first apply a simple pre-processing procedure to get rid of the model variables x_t and y_{tj} if there is no customer demand on timeslot t, the total demand on timeslot t is less than the minimum train loading P_1, or there is a service restriction banning x_t. After pre-processing constraints (3.8) can be removed as well.

Points in the search space correspond to a complete assignment of 0 or 1 to all decision variables. The search space is explored by a sequence of simple randomised moves which are influenced by the violated constraints at the current point.

CLS starts with an initial random assignment, in which some hard constraints in the model can be violated. In the iteration loop, the algorithm always selects a violated hard constraint at random, e.g. an assigned train timeslot for which the demands exceed train capacity or an assigned timeslot that is not consistent with a customer's booking preferences.

Having selected a violated hard constraint, the algorithm randomly selects one variable in that constraint and another variable, either from the violated hard constraint or from the search space. Then two flip trials are performed in which the current value of the variable is changed to its complementary binary value. Suppose that V_i takes the value v_i at the start of the iteration so that the current solution $A = (v_1, v_2, ..., v_m \mid h)$, where m is the total number of variables and h is the total violation of all hard constraints. Suppose further that V_1, V_2 are chosen and that their flipped values are $\bar{v}_1, \bar{v}_2$ respectively. We then look at the assignments $A_1 = (\bar{v}_1, v_2, ..., v_m \mid h_1)$, $A_2 = (v_1, \bar{v}_2, ..., v_m \mid h_2)$ and select the alternative with the smaller total hard violation. This alternative becomes the new current point. It is noted that CLS selects the best alternative (A_1 or A_2) for a new current solution even if it is worse than A. The aim is to allow diversity in the search so that the search space may be fully explored. CLS terminates when a feasible solution A is found or when no improvement to the best total hard constraint violation found has been achieved for a specified number of iterations Z. The procedure of the basic CLS is outlined in Figure 4.1.

```
proc CLS
    A ← initial random assignment
    h ← initial hard constraint violation
    if h = 0 then output A , exit CLS
    terminate ← false
    try ← 0
    while not terminate do
        C := select-violated-hard-constraint (A)
        P := select-two-variables (C, A)
        A1 , A2 := flip (A, P)
        try ← try +1
        if (h1 < h2) then (A ← A1)
            if h1 < h then h ← h1
                           try ← 0
        else (A ← A2)
            if h2 < h then h ← h2
                           try ← 0
        end if
        if h = 0 then output A , terminate ← true
        if try = Z then terminate ← true
    end while
end proc
```

Figure 4.1: The basic CLS procedure

From Figure 4.1, in *select-violated-hard-constraint*, CLS randomly selects a constraint that is not satisfied by the current assignment, and then in *select-two-variables*, CLS randomly

selects one variable in that constraint and another variable, either from the violated hard constraint or from the search space to flip.

The principal goal of CLS is to find a complete assignment of values to variables which is consistent with all the hard constraints. This task is eased if partial consistency is maintained throughout the search. The approach taken in applying CLS to the container rail scheduling problem is to decompose it into a series of similar subproblems, each with the number, but not timing, of trains fixed in relation to the minimum train loading parameter P_1 and maintaining the coverage constraints (3.4). Thus consistency is maintained within each set of variables but enforced across different sets.

When CLS finds a feasible solution A, the refined-improvement procedure is called in an attempt to reduce the soft violation for the customer satisfaction constraint s_2 ((3.13) in Section 3.4.2.1). The main concept of the refined-improvement procedure is to search more intensively on A by fixing the number and timing of trains. This is because it is expensive to maintain the consistency between the timeslot variables x_t and the customer's booking variables y_{tj} if the train timetable is not fixed. In this procedure only feasible improving moves are allowed. Here, the refined-improvement does not try to reduce the soft violation for the timeslot-operating cost constraint s_3 ((3.15) in Section 3.4.2.1) for the same reason as above; in addition, the variation in timeslot-operating cost is small. However, in Section 4.3.2, the artificial soft violation S^* is introduced in an attempt to reduce the total soft violation whilst all hard constraints are still to be fully satisfied by CLS.

The refined-improvement procedure is:

Step 1 Record the soft violation s_2 for A.

For each customer:

Step 2 Order the alternative, active timeslots for this customer in increasing order of satisfaction cost for this customer.

For each possible timeslot:

Step 3 Swap this customer with a customer currently served in this timeslot.

Step 4 If the new assignment is feasible and reduces the soft violation s_2, replace A, s_2. Repeat step 2 for the next customer, if any exist. If the new assignment is not feasible or is feasible but does not reduce s_2, repeat step 3 with the next customer if any exist served in this timeslot.

After the refined improvement is performed, the algorithm reduces the soft violation cost s_1 ((3.11) in Section 3.4.2.1) by removing one train out of the current feasible solution. As a result, the problem is more constrained and the current solution becomes infeasible. The value of stopping criterion parameter Z is refreshed and CLS is called again to find a new feasible solution for the problem.

4.2.2 Violated constraint selection

From the procedure of CLS in Figure 4.1, the remaining degrees of freedom in designing the search strategy are how to select a violated constraint and which variables to flip. In CLS, it is hard to find good strategies for the selection of a violated hard constraint and of variables

which have a strong impact on performance of the algorithm. Different selections of a violated hard constraint have been investigated for SAT local search (McAllester, 1997 and Walser, 1999). For instance, choosing the violated constraint with maximum or minimum constraint violation; however none have been shown to improve over random selection. Therefore, the question remaining is how to select good variables to flip.

4.2.3 Variable selection

Once a violated hard constraint has been chosen, CLS selects two variables in order to perform trial flips. This is different from GSAT in which only one variable is chosen to flip in favour of less computation. Random selection is used to achieve a diversified exploration of the search space.

CLS could also choose only one variable to flip in order to reduce the computational cost of each iteration. On the other hand, it could select more than two variables to improve the performance of the local move. However, iterations become more computationally expensive as the number of variables selected increases. Therefore a compromise of selecting two variables is made. In addition, the two-variables selection scheme is theoretically and computationally suitable for the predictive choice model (described in Chapter 5) which involves building a joint probability distribution of the non-deterministic interaction between two variables in the combinatorial optimisation model in order to predict a good choice of value for a variable.

For the container rail scheduling model, there are two sets of binary variables: a timeslot variable x_t represents whether to operate a train service or not in each potential timeslot, and

customer's booking variable y_{tj} represents whether a customer is served in each potential timeslot. In this case, by randomly choosing one variable from the violated constraint and another from the search space, diversified exploration of the search space may not be achieved because the number of y_{tj} variables is much greater than the x_t variables. Therefore, alternative selection schemes are introduced as follows:

- Scheme 1: Randomly select any two variables in the violated constraint.
- Scheme 2: Randomly select one variable from the y_{tj} set and one x_t variable from the violated constraint.
- Scheme 3: Randomly select one y_{tj} variable from the violated constraint and another variable (x_t or y_{tj}) from the search space.
- Scheme 4: Randomly select one x_t variable from the violated constraint and one x_t variable from the search space

CLS selects one of these schemes at random so that a wide exploration of the search space can be achieved. It is noted also that not all these variable selection schemes are applicable at all times during the search process because of maintaining consistency within each set of variables, or across the two sets of variables (described in Chapter 5).

4.3 Violation strategy

In SAT local search and its variants, the number of violated constraints (unsatisfied clauses) is used to evaluate local moves without accounting for how severely individual constraints are violated. In CLS, a quantified measure of the constraint violation is used to evaluate local

moves. In this case, the violated constraints may be assigned different degrees of constraint violation. This leads to a framework to improve the performance of the solving algorithm. The constraints can be weighted according to their relative importance in order to allow the search to give priority to satisfying some subsets of the constraints.

For container rail scheduling, soft and hard constraints in the model are treated separately. The hard constraints play the major role in guiding the search. When all hard constraints are satisfied, the soft constraint violations are calculated and used as a measure of the quality of the solution. Nevertheless, whilst the hard constraints have not yet been fully satisfied, our scheme incorporates an artificial constraint, and its weighted violation measure is designed to exert some influence over the search process based on an estimation of the soft constraint violation (discussed in Section 4.3.2).

4.3.1 Hard violation

The principal goal of CLS is to find a feasible solution to the problem, i.e. solution points at which the total violation of the hard constraints is zero. To improve the performance of the solving algorithm using a violation strategy, two basic questions arise:

1. If the same measure of the constraint violation should be used within each set of hard constraints.
2. If the same measure of the constraint violation should be used across all sets of hard constraints

Using different measures of the violation affects the performance of the algorithm because the algorithm gives priority to satisfying some subsets of constraints. For our container rail scheduling model, all sets of hard constraints use measures of violation weighted according to some heuristic rules.

For the train capacity constraints, any number of containers in a potential timeslot exceeding a train capacity is penalised with the same violation h_m. Here, the violation penalty for exceeding a train capacity does not depend on the amount of overcapacity. The violation for overcapacity timeslots represents the number of violated capacity constraints (or the number of over-loaded trains) and CLS tries to eliminate this violation. However, this penalty measure will be tested when the computational experiments are given in Section 4.5. The violation penalty for a set of capacity constraints is defined as:

$$x_t\left(\sum_{j\in M} N_j y_{tj}\right)\begin{cases} \leq P_2, \text{violation} = 0 \\ \\ > P_2, \text{violation} = h_m \end{cases} ;\forall t \qquad (4.1)$$

where: h_m is a violation penalty for a capacity constraint, P_2 is the capacity of a train.

It is noted that the variable x_t is introduced into the capacity constraint. This is because for CLS, a measure of hard constraint violation should be quantified so that it can be used to sensibly evaluate local moves. For instance, in (4.1) if x_t is excluded, an over-loaded train (overcapacity timeslot) will be penalised with the amount of violation even if a train does not depart at timeslot t, which is non-meaningful. Penalties for assigning customers for timeslots without an assigned train are added via the consistency constraint (3.7). To do so

via (3.6) as well would effectively double count this violation. In addition, with the presence of x_t the algorithm needs not calculate train capacity usage when x_t = 0, reducing computational time.

For the consistency constraints, the algorithm assigns the violation penalty h_c if a train does not depart at the timeslot but there are some customers assigned to that timeslot. The violation penalty for a set of consistency constraints is defined as:

$$\sum_{j \in C_t} y_{tj} \begin{cases} \leq x_t |C_t|, \text{ violation} = 0 \\ \\ > x_t |\mathrm{C_t}|, \text{ violation} = h_c \end{cases} ; \forall t \tag{4.2}$$

where: h_c is a violation penalty for a consistency constraint, C_t is number of potential customers for timeslot t.

For the covering constraints, the algorithm allocates a penalty if the assigned trains do not serve all customer demands. In other words, the covering constraints favour a combination of selected timeslots that covers all customers' bookings. The violation penalty within a set of covering constraints uses the same quantification, which is defined as:

$$\sum_{t \in S_j} x_t \begin{cases} \geq 1, \text{violation} = 0 \\ \\ = 0, \text{violation} = h_s \end{cases} , \forall j \tag{4.3}$$

where: h_s is a violation penalty for a covering constraint.

Covering constraints do not affect the operational requirements for a rail carrier, and their removal does not change a feasible solution. However, their presence may improve the performance of the algorithm as it guides the search to the feasible solutions. A quantified measure of the violation h_s should be set higher than those of h_m and h_c so that the search gives priority to satisfying a set of covering constraints.

Therefore, the total hard violation is the sum of the violation penalties for train capacity, timeslot consistency, and covering constraints, which can be written as:

$$h = H_m + H_c + H_s \tag{4.4}$$

where: h is the total hard violation; H_m, H_c, and H_s are the total hard violations for train capacity, timeslot consistency, and covering constraints respectively.

4.3.2 Artificial soft violation

The artificial soft violation S^* has been introduced so that the search considers an estimation of total soft constraint violation, while some hard constraints are still to be fully satisfied. The artificial soft violation is regarded as if it were a hard violation until the capacity, consistency, and covering constraints have all been satisfied, then the artificial violation is set to zero. The algorithm assigns a violation penalty s_t^* if a timeslot t is selected as a train departure time. This can be written as:

$$x_t \begin{cases} = 1, \text{violation} = s_t^* \\ \\ = 0, \text{violation} = 0 \end{cases} , \forall t \qquad (4.5)$$

Note that, S^* is the sum of these violations.

In contrast to (4.1) – (4.3) which imply a fixed violation penalty for each member of the associated set of constraints, a violation penalty for the artificial soft violation s_t^* varies from timeslot to timeslot. The artificial soft violation penalty depends on the possibility of assigning a particular timeslot on a train schedule with a minimum generalised cost.

An attempt to derive the artificial soft violation in monetary units by trading off between the business criteria is not possible. This is because a train schedule is not a single timeslot, but is a set of the timeslots. Therefore, considering only a single timeslot separately from the others cannot represent a total cost for the rail carrier. However, as in practice, some business criteria play more an important role than others, the relative weights for the criteria could be applied.

A rail carrier may assign a relative weight to the violation costs of number of trains N_t, customer satisfaction Q_t, and timeslot-operating cost E_t constraints in a selected timeslot t. With equal violation costs for N_t, Q_t, and E_t the violation cost with the lower weight will give a smaller artificial soft violation s_t^*.

In practice, given the relative weights 0.2, 0.5 and 0.3 by the Royal State Railway of Thailand, s_t^* is therefore obtained as:

$$s_t^* = 0.2N_t + 0.5Q_t + 0.3E_t; \qquad \forall t \tag{4.6}$$

where: s_t^* is the artificial soft violation if timeslot t is chosen (i.e. x_t = 1).

4.3.2.1 The violation cost N_t

We first assume that the higher the number of potential customers in timeslot t, the more likely it is that assigning a train to that timeslot would lead to the minimum number of trains used. However, it is also necessary to consider the distribution of customer shipment size. Although there are a large number of potential customers in a timeslot, each customer shipment may be large. Therefore, such a timeslot could allow only a few customers to be served on a train so giving a high violation cost (or a priority) to this timeslot is no longer reasonable. N_t is defined by:

$$a_t = \frac{\mu_t + \sigma_t}{|C_t|} \qquad ;\forall t \tag{4.7}$$

$$n_t = \frac{a_t}{\sum_{t=1}^{T} a_t} \qquad ;\forall t \tag{4.8}$$

$$N_t = \frac{n_t}{n_{(\max)}} \times 100 \qquad ;\forall t \tag{4.9}$$

where: $n_{(\max)} = \max\{n_t : t = 1,2,3,...,T)$, μ_t is the mean of customer shipment size in timeslot t, σ_t is the standard deviation of the customer shipment size in timeslot t, $|C_t|$ is the number of customers in timeslot t.

The rationale of the formula for N_t is that it aims to tackle the variation of customer shipment size in the potential booking timeslot and then to provide the estimated chance for that timeslot to be used. The variation of customer shipment size is simply handled using the sum of μ_t and σ_t (4.7) as a demand threshold for the shipment size. The remaining concepts are just to reflect that the higher the number of potential customers in timeslot t, the more likely it is that assigning a train to that timeslot would lead to the minimum number of trains used. The calculation of N_t is illustrated as in Table 4.1.

Timeslot	Customer shipment size (containers)					Mean	STD	a_t	n_t	N_t
(t)	C_1	C_2	C_3	C_4	C_5	μ_t	σ_t			
1		20			5	12.50	10.61	11.55	0.24	100
2	10	20	15	8	5	11.60	5.94	3.51	0.07	30
3		20	15	8		14.33	6.03	6.79	0.14	59
4	10				5	7.50	3.54	5.52	0.12	48
5		20	15			17.50	3.54	10.52	0.22	91
6		20	15	8	5	12.00	6.78	4.70	0.10	41
7	10		15		5	10.00	5.00	5.00	0.11	43
Sum								47.58		

Table 4.1: An example for the calculation of violation cost N_t

Table 4.1 shows that when timeslot 1 is selected (x_1 = 1), the algorithm assigns the highest violation cost for that timeslot = 100. The more customers in a timeslot, the lower the violation cost in general. When timeslots serve an equal number of customers (e.g. timeslot 3 and 7). the lower violation cost is assigned to the timeslot with a more even spread of demand, here timeslot 7.

4.3.2.2 The violation cost Q_t

Although the virtual loss of future revenue in the generalised cost function could represent customer satisfaction in terms of cost, it is an indirect cost. In practice, the indirect cost is not obvious for rail expenditure as it affects the long-term financial plan. Therefore, in a competitive transport market, a direct cost that affects the short-term cash flow is regarded as more important. The satisfaction of customer j in timeslot t is w_{tj} ($0 \le w_{tj} \le 1$), and ($w_{tj} \times 100$) is the satisfaction score. W_t is a total customer satisfaction score in timeslot t, i.e. $W_t = \sum_{j \in C_t} \left(w_{tj} \times 100 \right)$. The higher the value of W_t, the more likely it is that the timeslot t would be used. The violation cost Q_t is defined as:

$$b_t = \frac{\sum_{t=1}^{T} W_t}{W_t} \quad ;\forall t \tag{4.10}$$

$$q_t = \frac{b_t}{\sum_{t=1}^{T} b_t} \quad ;\forall t \tag{4.11}$$

$$Q_t = \frac{q_t}{q_{(\max)}} \times 100 \quad ;\forall t \tag{4.12}$$

where: $q_{(\max)} = \max\{q_t : t = 1,2,3,...,T)$. The calculation of Q_t is illustrated as follows:

From Table 4.2, Q_t is not only affected by the customer satisfaction score but also is implicitly dependent on the number of potential customers using the timeslot, i.e. the more customers in the timeslot, the lower would be the violation cost Q_t. When timeslots have the same number of customers (e.g. timeslot 3 and 7), the algorithm assigns a lower violation cost to timeslot 3 because it has a higher value of W_t.

Timeslot	Satisfaction score					W_t	b_t	q_t	Q_t
(t)	C_1	C_2	C_3	C_4	C_5				
1		90			70	160	9.76	0.18	81
2	95	90	76	85	70	416	3.75	0.07	31
3		90	76	85		251	6.22	0.11	52
4	95				70	165	9.47	0.17	79
5		70	60			130	12.02	0.22	100
6		70	60	60	50	240	6.51	0.12	54
7	90		60		50	200	7.81	0.14	65
Sum						1562	55.54		

Table 4.2: An example for the calculation of violation cost Q_t

4.3.2.3 The violation cost E_t

A rail carrier may have different operating costs for different timeslots. The operating costs comprise train congestion cost and staff cost. Although a train schedule is a set of timeslots, we could consider E_t for the operating costs of each timeslot directly. This is because the operating cost is a cost unit and does not affect the number of timeslots in the optimal train schedule. E_t is defined by:

$$e_t = \frac{U_t}{\sum_{t=1}^{T} U_t} \quad ;\forall t \tag{4.13}$$

$$E_t = \frac{e_t}{e_{(\max)}} \times 100 \quad ;\forall t \tag{4.14}$$

where: $e_{(\max)} = \max\{e_t : t = 1,2,3,...,T)$, E_t is a violation cost for the operating costs in timeslot t, U_t is the operating cost in timeslot t (U_t includes train congestion cost and staff cost).

In (4.13), e_t is derived from the proportion of the operating costs in timeslot t to the total timeslot operating cost. E_t in (4.14) reflects that the lower the operating costs for the timeslot, the higher the estimated chance that the timeslot would lead to a schedule with the minimum generalised cost, and the value of E_t is shown in a percentage scale. The calculation of the violation cost E_t is illustrated as follows:

Timeslot (t)	U_t ($\times 10^3$ Baht)	e_t	E_t
1	4.46	0.16	92
2	2.71	0.10	56
3	2.71	0.10	56
4	3.36	0.12	69
5	4.85	0.17	100
6	4.85	0.17	100
7	4.85	0.17	100
Sum	27.79		

Table 4.3: An example for the calculation of the violation cost E_t

Table 4.3 shows that a rail carrier incurs operating costs that vary from timeslot to timeslot. Choosing a timeslot that has high operating costs would be penalised with high violation E_t.

From (4.4) and (4.5), the total constraint violation h is the sum of total hard violation (H_m, H_c, H_s) and the weighted total artificial soft violation S^*:

$$h = \left(H_m + H_c + H_s\right) + \left(S^* \times \alpha\right) \tag{4.15}$$

The parameter α represents the violation penalty of one unit of artificial soft violation. This parameter can be adjusted empirically in order to balance the trade-off between the artificial

soft violation S^* and the hard violations (H_m, H_c, H_s), i.e. when α is increased, the search algorithm treats S^* more importantly relative to H_m, H_c, and H_s.

Now, CLS uses this total constraint violation h to evaluate local moves. The modified procedure of CLS for the container rail scheduling problem is given as:

proc CLS
  $A \leftarrow$ initial random assignment
  $h \leftarrow$ initial total constraint violation
  if (H_m, H_c, H_s) = 0 **then** $S^* \leftarrow 0$, output A, exit CLS
  terminate $\leftarrow$ false
  try $\leftarrow$ 0
  while not terminate **do**

  { **the same as the basic CLS in Figure 4.1**

    if (H_m, H_c, H_s) = 0 **then** $S^* \leftarrow 0$, output A, terminate $\leftarrow$ true
    if try = Z **then** terminate $\leftarrow$ true
  end while
end proc

Figure 4.2: The modified CLS procedure

There are two differences between the basic CLS (Figure 4.1) and the modified CLS for the container rail scheduling problem (Figure 4.2): the quantified measure of local moves and the stopping criterion. In Figure 4.2, h is the sum of (H_m, H_c, H_s) and S^*; the algorithm stops when ($H_m + H_c + H_s$) = 0 (i.e. a feasible solution A is found) or when no improvement to the best total constraint violation found has been achieved for Z iterations. At this time, S^* is discarded and is no longer used. The algorithm continues with the same procedure as described in Section 4.2.1.

4.4 Minimum train loading

The constraint-based local search assigns a fixed number of trains according to the number of trains expected, which is derived from the minimum train loading. In other words, a fixed number of timeslots used is maintained during the search process, which can be written as:

$$\sum_{t \in T} x_t = T_{\text{exp}} \tag{4.16}$$

where: T_{exp} is the number of trains expected.

Setting a minimum train loading ensures satisfactory revenue for a rail carrier and spreads out the capacity utilisation on train services. The carrier may want to set the minimum train loading as high as possible, ideally equal to the capacity of a train. Note that the minimum train loading is directly related to T_{exp}, which is defined as:

$$T_{\exp} = \left\lceil \sum_{j \in M} N_j / P_1 \right\rceil \tag{4.17}$$

where: N_j is the demand of customer j, P_1 is the minimum train loading used (3.5).

Apart from ensuring satisfactory revenue, minimum train loading is a key factor in the performance of the search algorithm. The higher the minimum train loading, the more constrained the problem is and hence the number of feasible solutions decreases. Using a high minimum train loading allows the algorithm to focus on satisfying the hard constraints more than the soft constraints. In addition, it increases the usefulness of the predictive choice model, i.e. the variables in the container rail scheduling model would take more consistent values in all the feasible solutions during the search (described in Chapter 5).

However, it would be very hard to prove whether there exists a feasible solution to the problem constrained by a high minimum train loading. If we could prove the existence of a feasible solution for the highest possible minimum train loading, it implies that the solution is approximately optimal. A good setting of the minimum train loading helps limit the size of the search space. Although a few techniques for proving the existence of feasibility have been proposed (Hansen, 1992; Wolfe, 1994; Kearfott, 1998), implementations of these techniques for practical problems have not yet been achieved. In this research, the minimum train loading is derived from some heuristic rules.

Suppose that the customer's booking data is given as in Table 4.4.

Day	Customers	Containers	Shipment size	
			Mean	Std.
MON	15	228	15.20	8.06
TUE	18	295	16.39	4.30
WED	27	389	14.41	5.60
THU	32	554	17.31	7.04
FRI	18	329	18.28	5.09
SAT	11	243	22.09	7.50
SUN	23	430	18.70	7.09
			μ_g = 17.48	σ_g = 6.38

Table 4.4: An example – customer's booking data

From Table 4.4 each schedulable day has an average customer demand (containers) and its standard deviation (e.g. Mon: Mean = 15.20 and Std. = 8.06). Note that in Table 4.4, the total demand $\sum_{j \in M} N_j$ is not the sum of the demand in all days (i.e. 228+295+389+…+430). This is because a customer demand is assigned to a set of the preferred and alternative booking timeslots in which a train schedule has not yet been provided.

However, we need the average size of customer demand in a scheduling week, here μ_g = 17.48, in order to estimate roughly how many customers (demand units) can be served by one train (note that a customer demand cannot be split in multiple trains as described in Section 3.4.1), and therefore how many trains are required to serve all customer demands.

We also take the customer demand variation into account. By assuming customer demand is normally distributed, we could say that in most cases customer demand is less than the demand threshold ($\mu_g + \sigma_g$). If we use this threshold as the average customer demand, we could obtain a number of trains needed to serve all customer demands and, from (4.17), a feasible minimum train loading can be derived. An initial value of P_1 is defined as follows:

$$T^* = \frac{P_2}{\mu_g + \sigma_g} \tag{4.18}$$

$$P_1 = \left\lceil \frac{\sum_{j \in M} N_j}{M / T^*} \right\rceil \tag{4.19}$$

where: P_2 is the capacity of a train, N_j is the demand of customer j, M is the total number of customers.

In (4.18), T^* is used to express the proportion of the train capacity to the demand threshold ($\mu_g + \sigma_g$). In (4.19), the proportion of the total demand to the estimated number of trains (M / T^*) gives the initial minimum train loading P_1. The calculation of P_1 is illustrated as follows: From Table 4.4, suppose that M = 70, total demands $\sum_{j \in M} N_j$ = 1370, a train capacity P_2 = 68, we will obtain the minimum train loading P_1 = 56, and by substituting P_1 into (4.17), we can obtain the initial number of trains expected T_{exp} = 25. However, we note that this is a very rough estimate used simply to set the initial value for the number of trains.

Whenever all hard constraints are satisfied (a feasible train schedule is obtained), the minimum train loading is increased by removing one train from the current state of the feasible solution, i.e. $T_{\exp} = T_{\exp} - 1$, and CLS attempts to find a new feasible schedule. However if no feasible solution is found, $T_{\exp}$ is increased by 1.

4.5 Computational results

The container rail scheduling model is tested on two sets of four successive weeks data from the eastern-line container service, the Royal State Railway of Thailand (SRT), involving 184 shipping companies. Each train has a capacity of 68 standard containers. The problem instances are summarised in Table 4.5. In this table, θ denotes a lower bound on number of trains. Note that a customer may require several different container rail services per week and we consider these demands as arising from different customers.

Test case	Customers	Containers	θ	SRT's schedules		Supply - Demand	
				Trains	Capacity	Capacity	Trains
W1	134	2907	43	57	3876	969	14
W2	116	2316	35	42	2856	540	7
W3	84	1370	21	28	1907	537	7
W4	109	2625	37	50	3400	775	13
W5	225	4115	61	73	4964	816	12
W6	198	3350	50	59	4012	612	9
W7	126	2542	38	49	3332	748	11
W8	286	4731	70	86	5848	1088	16

Table 4.5: Problem instances

From the constraint model's point of view, Table 4.6 shows the size of the problem instances in terms of the number of the constrained variables (timeslot variable x_t and customer's booking variable y_{tj}) and operational constraints (train capacity, coverage, and timeslot consistency constraints). Note that those x_t without any demands and y_{tj} corresponding to timeslots that customer j does not prefer are discarded and not counted.

Test case	Model variables		Constraints
	x_t	y_{tj}	
W1	152	1588	438
W2	168	1270	452
W3	125	967	334
W4	131	1362	371
W5	168	1415	561
W6	149	1970	496
W7	132	1197	390
W8	168	2534	622

Table 4.6: Problem size

From Table 4.6, all test cases have a large number of binary decision variables. This shows that in the worst case, there would be 2^n possible solutions; where n is the number of the model variables. Good short cuts may be obtained to drive the search to the good solutions, or to reduce the size of the search space. However, for the container rail scheduling problem, the size of the search space cannot be reduced easily because the consistency and capacity constraints are enforced (Section 3.4.2).

In this section, we present the computational results of the implemented framework of the constraint-based local search algorithm (CLS), and compare results between the demand responsive scheduling model we propose and current practice. Each test case is run ten times on a PC-Pentium 2.4 GHz using different random number seeds at the beginning of each run.

Now, we want to find a good value of the stopping criterion parameter. The concept is that the algorithm should not spend more computational time than necessary; in contrast, more time should be given if solution tends to be improved further. We varied this parameter value from 500 to 5000 and observed that the value = 2000 is a reasonable setting as solutions do not tend to be improved after this value.

We set violation penalties for capacity, consistency, and covering constraints (h_m, h_c, h_s) = 1, 1, 100 respectively, and the violation parameter α is set to 0.15. The sensitivity for these parameters will be tested when the computational results are given.

In SRT's schedules, the rail business criteria are represented by the operating costs OC. However, in CLS's schedules, the criteria are expressed in terms of the generalised cost. Although the criteria for the CLS 's schedules are converted into a single generalised cost for more evaluation, each component of the generalised cost can be shown explicitly. Tables 4.7 - 4.8 compare the model results with current practice. In these tables, GC is the generalised cost, NC is the number of trains cost, TC is the timeslot-operating cost, VC is the virtual loss of future revenue, and the operating costs $OC = NC + TC$. All costs are shown in a unit of ($\times 10^6$) Baht.

Test case	SRT		CLS					
	Trains	*OC*	Trains			*GC*		Time (Sec)
			Min	Avg.	Std.	Avg.	Std.	
W1	57	5.17	49	50.87	1.39	5.47	0.42	387
W2	42	3.74	37	38.68	1.40	3.95	0.21	168
W3	28	2.19	24	24.74	0.81	2.06	0.18	89
W4	50	4.48	41	42.21	1.55	4.48	0.26	169
W5	73	6.49	68	69.44	1.57	7.01	0.18	930
W6	59	5.25	56	58.80	2.06	6.18	0.75	655
W7	49	4.66	44	45.11	1.19	4.85	0.39	297
W8	86	7.65	79	80.64	1.71	8.06	0.37	1980

Table 4.7: The schedules obtained by CLS

Test case	SRT	CLS				*OC*
	OC	*GC*	*NC*	*TC*	*VC*	Reduction (%)
W1	5.17	5.47	4.34	0.17	0.97	12.86
W2	3.74	3.95	3.49	0.11	0.36	3.88
W3	2.19	2.06	1.79	0.07	0.29	15.30
W4	4.48	4.48	3.61	0.14	0.72	16.24
W5	6.49	7.01	5.61	0.23	1.17	10.02
W6	5.25	6.18	4.76	0.20	1.22	5.52
W7	4.66	4.85	4.17	0.15	0.53	7.40
W8	7.65	8.06	6.29	0.22	1.55	14.90

Table 4.8: Operating cost comparison: SRT vs CLS

Tables 4.7 and 4.8 show, in all test cases, there are some reductions in the number of trains and operating costs. In some test cases the reductions in operating cost are not considerable; this is because in practice the SRT's schedule is not fixed at the same service level everyday. The rail carrier may cut down the number of train services with short notice if the train supply is a lot higher than the customer demand. This is done by delaying some customer's departure times according to its demand consolidation strategy.

However, the proposed model maximises customer satisfaction, in other words, minimises the virtual loss of future revenue within a generalised cost framework. Therefore, the schedule obtained by CLS could reflect a high degree of customer satisfaction with the minimum rail operating costs through a demand responsive schedule.

From a computational viewpoint, we perform further experiments to test the performance of CLS using different search strategies and to test the sensitivity of parameters in the algorithm.

4.5.1 Experiment I

We first test the performance of CLS without the refined improvement procedure. When a feasible solution is found by CLS, the algorithm continues to reduce the soft violation for the number of trains constraint by removing one train from the current state of the feasible solution. In addition, with the artificial soft constraint violation S^* in (4.15), the algorithm reduces the soft constraint violation, whilst some hard constraints are still to be fully satisfied. Therefore, it seems that CLS without the refined improvement may be good enough and if so we could use the simpler algorithm and reduce computational effort. In this

experiment, each test case is run ten times, the parameters in CLS are the same as in the previous experiment. The computational results are shown in Table 4.9.

Test Case	Trains		Cost				Time (Sec)
	Min	Avg.	*GC*	*NC*	*TC*	*VC*	
W1	49	51.20	5.63	4.34	0.23	1.06	256
W2	37	38.90	4.19	3.49	0.15	0.55	121
W3	24	24.60	2.24	1.79	0.11	0.34	62
W4	41	42.90	4.75	3.61	0.24	0.90	115
W5	68	69.70	7.71	5.61	0.27	2.18	739
W6	56	59.20	6.48	4.76	0.28	1.44	428
W7	44	45.90	5.05	4.17	0.24	0.64	205
W8	80	81.70	9.02	6.36	0.34	2.32	1320

Table 4.9: CLS without the refined improvement procedure

From Table 4.9, not surprisingly we can obtain the number of trains as good as the one in Table 4.7 and the computational time is better in all test cases. However, in Table 4.9 the generalised cost GC and its components (NC, TC, and VC) are worse than the results obtained from CLS with the refined improvement. On average, GC is 7.16% more than GC in Table 4.8. Although the fact that the difference is not substantial, it could benefit the rail carrier in the long run. In addition, given a fixed number of trains, the rail carrier can assess how well their customers are satisfied in terms of VC; the schedule with more trains but smaller VC may sometimes be chosen depending upon the railway's strategies.

4.5.2 Experiment II

In this experiment, we test the sensitivity of the violation penalty parameters (h_m, h_c, h_s) for capacity, consistency, and covering constraints (described in Section 4.3). Each test case is run five times, h_m, h_c, h_s are varied whilst the remaining parameters in the algorithm are fixed. The results are shown in Table 4.10, in which " - " denotes that CLS failed to find a feasible solution.

Test case	CLS schedule 's cost (GC)							
	(1,1,0)	(1,10,0)	(10,10,0)	(100,1,0)	(1,1,10)	(1,1,100)	(1,1,200)	(1,1,500)
W1	5.53	5.68	5.60	5.87	5.29	5.22	5.33	5.91
W2	4.76	4.92	4.76	5.23	4.31	3.99	4.23	5.41
W3	2.47	2.66	2.54	3.05	2.13	2.18	2.22	2.77
W4	4.66	4.80	4.62	5.10	4.60	4.41	4.62	-
W5	7.80	7.92	7.64	-	7.84	7.23	7.64	7.58
W6	6.79	6.66	6.29	6.94	6.57	6.24	6.25	6.87
W7	4.86	5.07	4.98	5.01	4.77	4.71	4.75	5.24
W8	8.79	9.23	9.23	-	8.84	8.38	8.52	-

Table 4.10: Different measures of the violation penalties

From Table 4.10, using the same value of the capacity and consistency violations (h_m, h_c = 1) produces better quality schedules than does using different values. This indicates that h_m and h_c are not sensitive and shows the robustness of the algorithm, which does not require

special tuning of parameters for h_m and h_c. The violation strategy with covering violation ($h_s > 0$) shows superior quality of schedules. The values of h_s have a computational advantage as it may influence the search for the feasible schedules. However, choosing too large a value for h_s can decrease its solution quality. The experiments indicate that $h_s = 100$ is a reasonable setting.

4.5.3 Experiment III

Another experiment, which is closely related to the experiment in Section 4.5.2, is carried out. We test the sensitivity of the violation penalty parameter h_m but, in this experiment, the violation penalty for exceeding train capacity is given by the amount of overcapacity h'_m, i.e. from (4.1), $h'_m = \left(\sum_{j \in M} N_j y_{tj} \right) - P_2$. In this experiment, each test case is run ten times; the violation penalties are $h_c = 1$ and $h_s = 100$ and the remaining parameters in the algorithm are fixed. The results are shown in Table 4.11.

Test case	Trains			*GC*	
	Min	Avg.	Std.	Avg.	Std.
W1	50	52.26	1.58	5.62	0.48
W2	39	41.12	1.67	4.80	0.25
W3	24	25.95	1.13	2.66	0.25
W4	42	44.38	1.75	4.71	0.29
W5	70	72.58	1.81	7.63	0.21
W6	58	60.75	1.76	6.49	0.64
W7	44	45.58	1.39	4.90	0.46
W8	82	83.54	1.41	8.85	0.31

Table 4.11: Parameter h'_m given by an amount of overcapacity

From Table 4.11, the results obtained by CLS using h'_m are worse than the results in Table 4.7 in which the violation penalty for exceeding the train capacity is set equally, i.e. $h_m = 1$. This may be because, given a fixed number of trains, it may be better if CLS tries to reduce the number of over-capacity trains, focusing on satisfying the constraint, rather than considering the number of over-capacity containers on an assigned train. For instance, the larger violation penalty may account for a smaller number of over-loaded trains because a customer shipment is considered as one unit and cannot be split over multiple trains. However, in CLS, different measures of violation penalty across sets of hard constraints are used so that the algorithm gives priority to satisfying some subsets of constraints.

4.5.4 Experiment IV

This experiment is to test the sensitivity of the violation parameter α. This parameter controls the trade-off between the hard violations H_m, H_c, H_s and the artificial soft violation S^* in (4.15). The sensitivity test of α is shown in Table 4.12. In this experiment, each test case is run five times by varying α. The violation parameters h_m, h_c, h_s are fixed at 1, 1, 100 respectively.

Test case	CLS schedule s' cost (GC)						
	α = 0	0.05	0.15	0.25	0.35	0.50	0.75
W1	5.30	5.30	5.19	5.32	5.22	5.40	5.77
W2	4.33	4.23	4.02	4.31	4.22	4.42	5.16
W3	2.22	2.20	2.22	2.04	2.20	2.47	2.92
W4	4.70	4.50	4.41	4.13	4.80	4.75	5.10
W5	7.68	7.69	7.28	7.84	7.68	7.80	8.14
W6	6.36	6.45	6.04	6.11	6.04	6.54	6.90
W7	5.04	4.94	4.74	4.92	4.71	5.14	5.34
W8	9.12	8.92	8.75	8.68	8.78	8.74	9.06

Table 4.12: Sensitivity analysis of the timeslot violation parameter

Table 4.12 shows the effect of introducing the artificial soft violation S^*. With the existence of S^*, i.e. $\alpha > 0$, we tend to obtain a slightly better quality of solution in terms of the

generalised cost and α = 0.15 seems to be the best choice in general. The results also show that a high value of α provides a low quality solution as the search is most likely dominated by the artificial soft violation S^* whilst the hard violations H_m, H_c, H_s have not been eliminated.

It is noted that if the artificial soft violation S^* does not bring a significant improvement in the schedule, it may not be worthwhile to use it. However, the ideas of how S^* is obtained and used could be adapted and applied to other combinatorial optimisation problems, especially problems that have many feasible solutions and for which an effective bound on the objective function could not be achieved. In Chapter 6, we apply this idea to the generalised assignment problem.

4.5.5 Experiment V

We carry out an experiment to test the performance of CLS using different acceptance strategies of local moves. CLS always accepts both improving and non-improving moves, and CLS incorporating a simple simulated annealing method (CLS+SA) always accepts an improving move and accepts a non-improving move only with probability P_{sa} (Kirkpatrick, 1984):

$$P_{sa} = \exp\frac{-\Delta}{T} \tag{4.20}$$

where: Δ is the change in an non-improving move, and T is the temperature

A simple form of simulated annealing (SA) was used in this experiment composing of four components: initial temperature T_0, temperature length, cooling schedule and stopping criterion. To ensure a fair comparison between CLS and CLS + SA, we use approximately the same number of iterations for CLS + SA as CLS for each problem. A simple geometric cooling schedule was used, with the temperature being multiplied by a constant factor (or cooling rate) α at every iteration n, i.e. $T_{n+1} = \alpha T_n$. Some experiments were carried out for each problem in order to find good values of initial temperature T_0 and final temperature T_f and then the cooling rate α can be calculated directly from T_0 and T_f. The values of parameters T_0, T_f, and the number of iterations selected are shown in Table 4.13.

Test Case	Control parameters			
	T_0	α	T_f	Iteration
W1	5	0.9999870	1	118742
W2	8	0.9999740	1	78588
W3	9	0.9999300	1	30827
W4	8	0.9999650	1	58530
W5	5	0.9999956	1.2	322115
W6	5	0.9999930	1	226866
W7	6	0.9999830	1	102869
W8	5	0.9999981	1.5	634790

Table 4.13: Control parameters for SA

To perform the experiment using CLS+SA, each test case is run ten times, the parameters in CLS are fixed. The computational results are shown in Table 4.14.

Test Case	Trains			*GC*	
	Min	Avg.	Std.	Avg.	Std.
W1	50	51.23	1.87	5.41	0.48
W2	39	41.20	2.18	4.37	0.63
W3	25	26.43	0.72	2.36	0.14
W4	42	43.10	0.93	4.57	0.36
W5	70	72.14	2.03	7.51	0.85
W6	58	60.25	1.94	6.45	0.75
W7	45	46.50	1.84	4.80	0.66
W8	81	82.10	1.66	8.55	0.90

Table 4.14: Computational results CLS+SA

Table 4.9 shows that the results obtained from CLS+SA are relatively good; however they are worse than the results obtained from CLS alone (Table 4.7). This may be because in CLS+SA, the search may get trapped in poor quality states, in particular when the temperature is small. Assuming convergence to the optimal solution in infinite time; SA may cause problems because it uses approximately the same number of iterations as CLS; this could also imply that SA needs to be run a lot longer. In CLS the search is more diversified as improving and non-improving moves are always accepted.

4.6 Conclusions

The constraint-based local search algorithm, CLS, is presented in this chapter. The algorithm is inspired by SAT local search and is applied to the container rail scheduling problem. The ability to find a good operational train schedule along with satisfying customer demand leads to some reductions in the operating costs, and also enhances the level of customer service through the demand responsive planning model.

CLS starts with random initial assignments and uses a simple variable flip as a structure of local move. The algorithm is easy to implement and convenient to use. A violation strategy is introduced in order to drive the search to the promising regions of the search space, different measures of constraint violation allow the search to give priority to satisfying some subsets of the constraints. The artificial soft violation is introduced so that the algorithm evaluates the quality of the soft violations implicitly whilst satisfying the hard constraints; thereby it helps to reduce significantly computational effort. In CLS, diversified exploration of the search space is achieved by three features:

1. Acceptance criterion of local moves: CLS always accepts both improving and non-improving moves to gain a diversified exploration of the search space and to overcome local minima.
2. Randomised selection: a violated constraint and the variables to flip are selected in diversified sequences, which carries little computational cost.
3. Consistency enforcement: consistency is only maintained within each set of variables in the model independently (Sections 3.4.2.1 and 4.4). At some later

iterations, the consistency will be enforced by the predictive choice model (described in Chapter 5).

In the next chapter, we will present the construction and use of the predictive choice model as the search intensification strategy. CLS will be incorporated with the predictive choice model so that the algorithm can move around the solution space more effectively, leading to better quality solutions.

Chapter Five

Search intensification using the predictive choice model

5.1 Introduction

As discussed in Chapter 4, the constraint-based local search uses a simple variable flip as the local move. When all variables in the model are assigned a value, the total constraint violation is calculated. A quantified measure of the violation is used to evaluate local moves. Using different measures of the constraint violation, the algorithm gives priority to satisfying some subsets of constraints and thereby it guides the search to promising solution regions. In this Chapter a predictive choice model is developed to improve further the solutions to the problem of container rail scheduling. The predictive model is based on discrete choice theory and the random utility concept. When sufficient trial history has been collected for a variable, it is analysed to infer a good value for the variable. The variable is then fixed at this value for a number of iterations, determined in a probabilistic manner. At this point, consistency between variables is enforced, leading to intensified exploration of the search space.

This chapter is organised as follows. Section 5.2 describes the motivation for using the predictive choice model. Section 5.3 describes the development of the predictive choice model. Section 5.4 describes how the predictive choice model is used to improve the solutions of the container rail scheduling problem. The computational results obtained by constraint-based local search incorporating the predictive choice model are shown in Section 5.5. Finally conclusions are given in Section 5.6.

5.2 Motivation for the predictive choice model

For the purpose of search intensification, local search algorithms may try to fix a value for some variables for a certain number of iterations, depending on the search history. To do this we need a quantified assessment of each variable, called its "preference measure", with regard to its likely value in an optimal solution. The preference measure may also help in deciding how long a variable should remain fixed. Fixing may be done either deterministically or non-deterministically, using some heuristic rules. The simplest way would be, at some predetermined iteration, to observe which value (0 or 1) was chosen more often for a particular variable in the search so far. The preference measure for this value would be the proportion of time the variable was chosen to have that value. Then we could fix that variable at its preferred value for a number of iterations proportional to the preference measure.

If the proportion of time a binary variable was chosen to have one of its values is much higher the proportion of time the other value was chosen, we could have some confidence in the value at which we fix the variable, Conversely, if the proportions are close in value, we then lose some confidence in which value is to be chosen. Therefore, it may be helpful to

incorporate the amount by which one value is preferable to the other into the preference measure. This preference measure could be analysed by statistical methods so as to increase the confidence in choosing a value for a variable; in this research, we propose such an approach, the predictive choice model.

5. 3 Predictive choice model

The first development of choice models was in the area of psychology (Luce and Suppes, 1965). The development of these models arose from the need to explain the inconsistencies of human choice behaviour, in particular consumer choice in marketing research and mode choice in transportation research. If it were possible to specify the causes of these inconsistencies, a deterministic choice model could be easily developed.

These causes, however, are usually unknown or known but very hard to measure. In general, these inconsistencies are taken into account as non-deterministic or random behaviour. Therefore, the choice behaviour could only be modelled in a probabilistic way because of an inability to understand fully and to measure all the relevant factors that affect the choice decision.

5.3.1 Choice decision

Deciding on a choice of value for a variable is not obviously similar to the consumer choice decision. However, we could set up the search algorithm to behave like the consumer behaviour in choice selection. That is, we consider the behavioural inconsistencies of the algorithm in making a choice of good value for a variable.

For general combinatorial problems, a particular variable may take several different values across the set of feasible solutions. Thus it is hard to predict a consistently good value for the variable during the search. However, when the problem is severely constrained and has few feasible solutions, it may well be that some variables would take a more consistent value in all the feasible solutions during the search. For the container rail scheduling problem, the problem is more severely constrained by setting a high value of minimum train loading (as described in Chapter 4).

Once a variable has been selected, the algorithm has to choose a value for it. The concept is to choose a good value for a variable, e.g. the one that is likely to lead to a smaller total constraint violation in a complete assignment. In our constraint-based search algorithm, two variables are considered at each flip trial. The first variable is randomly chosen from those appearing in a violated constraint and considered as a variable of interest, the second variable is randomly selected either from that violated constraint or from the search space and is to provide a basis for comparison with the variable of interest.

Clearly, the interdependency of the variables implies that the effect of the variable value chosen for any particular variable in isolation is uncertain. Flipping the first variable might result in a reduction in total constraint violation. However, it might be that flipping the second variable would result in even more reduction in the violation. In this case, the flipped value of the first variable is not accepted.

In Table 5.1, the variable of interest is X_1 and the compared variable is X_j, the two variables are trial flipped in their values; the violations associated with their possible values are recorded and compared. In this Table, h is the total violation (in 4.15), X_1^* is the value

of X_1 chosen in the flip trial. Note that only h_1, h_1', and X_1^* are recorded for the violation history of X_1.

Flip trial	Variable of interest X_1				j	Compared variable X_j				X_1^*
	Current		Flipped			Current		Flipped		
	Val	h_1	Val	h_1'		Val	h_2	Val	h_2'	
1	1	26	0	22	15	1	26	0	36	0
2	1	20	0	12	9	0	20	1	6	1
3	0	5	1	2	2	0	5	1	7	1
4	1	15	0	14	30	0	15	1	10	1
.										
N	0	46	1	53	8	0	46	1	31	0

Table 5.1: Violation history

In flip trial 1 the selected variables are X_1 (current value 1) and, separately, X_{15} (current value 1). The current assignment has violation = 26. Flipping X_1, with X_{15} fixed at 1, gives violation = 22; flipping X_{15}, with X_1 fixed at 1, gives violation = 36. Hence in this trial the algorithm records X_1 = 0 as the better value. At some later iteration the algorithm chooses to flip X_1 again, this time (flip trial 2) with compared variable X_9. Flipping X_1, with X_9 fixed at 0, gives violation = 12; flipping X_9, with X_1 fixed at 1, gives violation = 6. Although flipping X_1 to 0 gives a better violation than the current assignment, in this flip trial the algorithm records X_1 = 1 as the better value as there is an assignment with X_1 = 1 which gives an even better violation. If we view the results of these flip trials as a random

sample of the set of all assignments, we can build up a predictive model to capture the inconsistency in the choice selection and to predict what would be a 'good' value for X_1.

5.3.2 Proportional method

The proportional method is based on a probabilistic mechanism in the sense that the algorithm may select the current value of a variable even though flipping that variable to the other value gives a lower violation.

The proportional method is straightforward. The choice selection is only affected by the number of occurrences of choice values in X^*, i.e. the constraint violation is not considered. The proportional method is defined as:

$$P_0 = \frac{\aleph_0^*}{N} \tag{5.1}$$

where: P_0 is a probability for the algorithm choosing value 0, $\aleph_0^*$ is the number of occurrences in X^* choosing value 0, N is the number of flip trials.

To investigate the accuracy of forecast of the proportional method, the container rail scheduling problem is scaled down. The optimal solution and their assigned values are known by running an integer programming branch and bound search to completion (ILOG, 2002). Then, we run the CLS to collect the violation history with flip trials N = 20. The experimental results are given in Table 5.2, in which $\overline{h}_0$ and $\overline{h}_1$ are the average total violations when a variable is assigned a value 0 and 1 respectively, $\aleph^*$ is the number of

occurrences of choice values in X^*, X^* is the value of variable X chosen in flip trial, Φ is the value of the corresponding variable in a known optimal solution.

Var no.	Violation		$\aleph^*$		Probability		Φ
	$\bar{h}_0$	$\bar{h}_1$	0	1	P_0	P_1	
1	25.10	28.05	7	13	0.35	0.65	0
2	22.45	27.55	17	3	0.85	0.15	0
3	32.30	27.70	11	9	0.55	0.45	1
4	24.35	20.40	4	16	0.20	0.80	1
5	24.80	23.55	11	9	0.55	0.45	0
6	20.65	18.35	5	15	0.25	0.75	1
7	21.60	19.25	13	7	0.65	0.35	1
8	26.20	22.60	12	8	0.60	0.40	1
9	21.85	20.30	11	9	0.55	0.45	0
10	24.90	20.95	12	8	0.60	0.40	0
11	25.00	20.25	1	19	0.05	0.95	1
12	29.15	23.85	13	7	0.65	0.35	1
13	20.35	26.25	18	2	0.90	0.10	0
14	25.25	23.65	11	9	0.55	0.45	1
15	23.65	25.40	6	14	0.30	0.70	0
16	26.90	21.45	2	18	0.10	0.90	1
17	28.70	27.10	3	17	0.25	0.85	1
18	26.15	27.65	13	7	0.65	0.35	0
19	23.20	18.55	4	16	0.20	0.80	1
20	24.70	23.95	14	6	0.70	0.30	1

Table 5.2: Value choice prediction by proportional method

Table 5.2 shows that the proportional method predicts a wrong optimal value for 8 of the 20 variables. We observe that, in these cases, although the number of occurrences of the choice values favours a particular value, the average violation, $\overline{h}$, does not. It would seem that a more complex predictor that accounts for both factors would be useful. Note that where there is a substantially higher number of choices of a particular value, the choice value is generally associated with a lower value of of $\overline{h}$, e.g. variables 2, 4, 11, 16, 19. There are also cases, e.g. variable 9, for which both the number of occurrences and $\overline{h}$ indicate the wrong choice of value. We introduce an additional method, the logit method, which might be more satisfactory in considering both factors. The selection of whether the proportional method or the logit method is to be applied is controlled by a decision parameter D (%). For example, if D is set to 70, the logit method is called when the proportion of any one value in X^* is less than 70%; otherwise, the proportional method is called instead.

5.3.3 Logit method

The logit method is based on the random utility concept (Ben-Akiva and Lerman, 1985). Choosing a good value for a variable in each flip trial is considered as a non-deterministic task of the search algorithm. In this research, the algorithm is designed to select a choice of value for a variable that has a maximum utility (or minimum disutility).

However, the utility is not known by the algorithm with certainty and is defined as U_0 and U_1; where U_0 and U_1 are utilities for the algorithm choosing value 0 and 1 respectively.

For each flip trial, the algorithm selects value 0 when flipping a variable to 0 is preferred to 1. This can be written as follows.

$$0 \succ 1 \Rightarrow U_0 > U_1 \tag{5.2}$$

From this point, the probability for the algorithm choosing value 0 is equal to the probability that the utility of choosing value 0, U_0, is greater than the utility of choosing value 1, U_1. This can be written as follows:

$$P_0 = \text{Prob}[U_0 > U_1] \tag{5.3}$$

where: P_0 is a probability for the algorithm choosing value 0.

When an occurrence of any choice value X^* is not obviously dominating, the logit method is used. The logit method is the predictive choice model with an assumed probability distribution of the random utility. The search algorithm selects a value 0 when the utility $U_0 > U_1$, and selects a value 1 otherwise.

To derive a logit model, we require an assumption about the joint probability distribution of the utilities U_1 and U_0. In this research, the difference between the utilities, i.e. $U' = U_1 - U_0$, is used. We consider U' as a random sample of the set of all assignments for a variable. From the central limit theorem "whenever a random sample of size *n* is taken from any distribution with mean μ and variance σ^2, the sample would be approximately normally distributed" (Trotter, 1959). We observe a real distribution of U' by running the

constraint-based local search algorithm to collect the violation history with 50 flip trials (samples). The total constraint violation h is used as a measure of the utilities. To make a fair assumption whether U' fits any specific probability distribution, we first take a look at the histogram and probability density trace, which is illustrated in Figure 5.1 and 5.2 for a typical case.

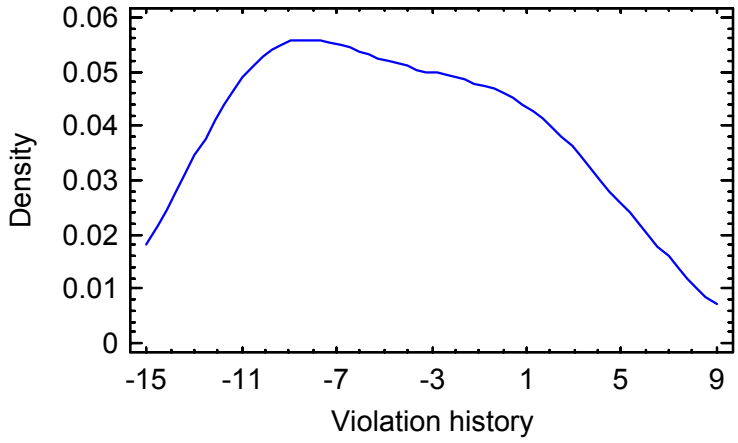

Figure 5.1: Probability density trace

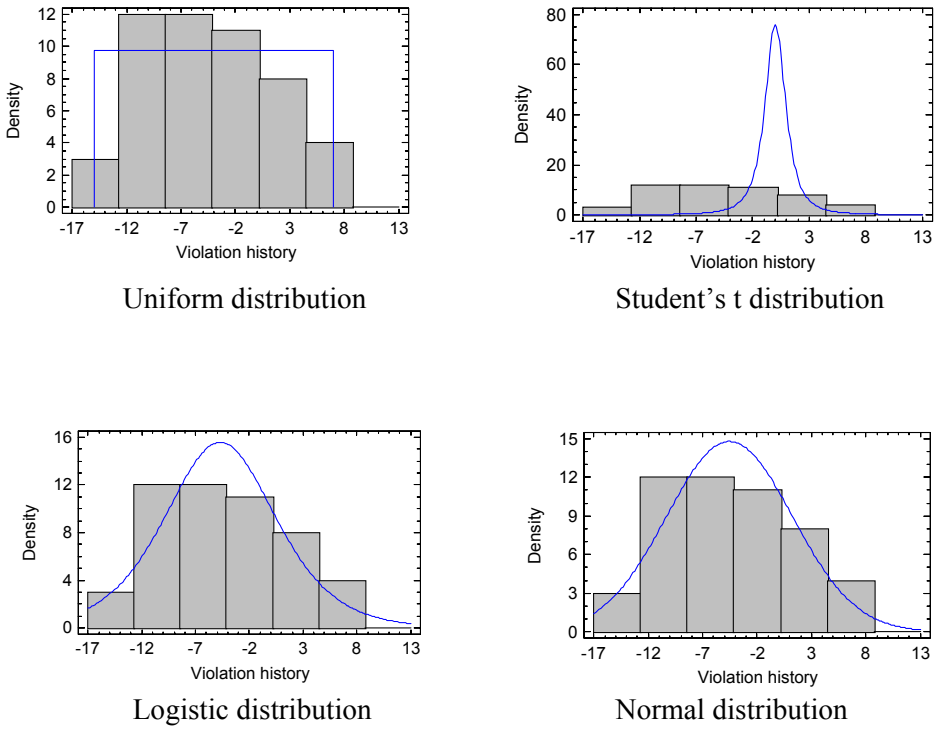

Figure 5.2: Histograms and possible probability distributions

From Figure 5.1 and 5.2, it appears that the logistic and normal distributions are the most likely candidates to represent the probability distribution of U'. Since the choice function assuming the normal distribution is expressed in terms of an integral (Ben-Akiva and Lerman, 1985), it requires a significant computational effort to calculate the probabilities in the choice function. Therefore, the choice model based on a logistic distribution is considered because the logistic distribution is an approximation of the normal law (Kallenberg, 1997). As shown in Figure 5.2, there is not a substantial difference between the normal and logistic distribution.

The Kolmogorov-Smirnov (K-S) test (Chakravart et al, 1967) is carried out to test whether U' comes from a population with a logistic distribution. The test statistic D measures the largest absolute difference between a theoretical logistic distribution and the observed U' distribution. The test statistic D is defined as:

$$D = \max_{1 \le i \le N} \left| F(x_i) - \frac{i}{N} \right| \tag{5.4}$$

where: $F(x_i)$ is the theoretical cumulative distribution of the logistic distribution, x_i is ordered U' from smallest to largest, and N is the number of flip trials.

The null hypothesis H_0 is defined as follows:

H_0: There is no difference between the distribution of U' and a theoretical logistic distribution

H_0 is rejected if the test statistic D is greater than the K-S critical value (for example, from the statistic table with sample size n = 50 and significance level α = 0.05 based on a logistic distribution, the K- S critical value = 0.1250). Table 5.3 illustrates the Kolmogorov-Smirnov test for probability distribution of U' for typical cases. The statistical software, SAS, was used to obtain the statistic value D (SAS, 2002).

Var no.	1	2	3	4	5	6	7	8	9	10
D	0.107	0.112	0.120	0.094	0.042	0.167	0.085	0.148	0.018	0.099
Var no.	11	12	13	14	15	16	17	18	19	20
D	0.069	0.094	0.083	0.091	0.073	0.110	0.073	0.081	0.098	0.180
Var no.	21	22	23	24	25	26	27	28	29	30
D	0.067	0.123	0.080	0.177	0.069	0.074	0.083	0.073	0.128	0.126

Table 5.3: The K-S test for probability distribution of U'

The results from Table 5.3 show that in general, the distribution of U' appears to be logistic because the test statistic *D* of a large majority of the variables is less than the K- S critical value 0.1250. Therefore, it may be reasonable to assume that U' is logistically distributed and to derive the predictive choice model from a logistic probability density function. The choice model assuming a logistic distribution is obtained as follows (Ben-Akiva and Lerman, 1985):

$$P_0 = \frac{e^{U_0}}{e^{U_0} + e^{U_1}} \tag{5.5}$$

where: P_0 is a probability for the algorithm choosing value 0

5.3.3.1 Utility function

For any flip trial, the utility U may be characterised by many factors. In this research, the utility is only determined by the total constraint violation *h*. This is because it can easily be measured by the algorithm and gives a reasonable hypothesis to the choice selection. In other words, we would like to use a function of utility for which it is computationally easy to estimate the unknown parameters.

We define a function that is linear in parameters. A choice specific parameter is introduced so that one alternative is preferred to the other when the total violation is not given, i.e. the choice decision may be explained by other factors. The utility functions for U_0 and U_1 are defined as:

$$U_0 = \beta_1 + \beta_2 h_0 \tag{5.6}$$

$$U_1 = \beta_2 h_1 \tag{5.7}$$

where: β_1 is a choice specific parameter, β_2 is a constraint violation parameter, h_0 and h_1 are the total violations when a binary variable is assigned a value 0 and 1 respectively.

5.3.3.2 Likelihood estimation

The parameters β_1 and β_2 can be estimated by several multivariate methods, such as maximum likelihood method, discriminant analysis, least square estimation, etc. (Johnson and Wichern, 1996). Among these methods, the maximum likelihood method is most popular. The aim of maximum likelihood method is to estimate the parameter values that

make the observed data a good fit to the likelihood function. The likelihood function over the N flip trials is a product of individual likelihoods, which is written as:

$$L(\beta_1, \beta_2) = \prod_{n=1}^{N} P_{0,n}{}^{y_{0,n}} P_{1,n}{}^{y_{1,n}} \tag{5.8}$$

where: L is a likelihood function, $P_{0,n}$ and $P_{1,n}$ are probabilities for the algorithm choosing value 0 and 1 in flip trial *n*, $y_{0,n}$ = 1 if the algorithm selects a value 0 in flip trial *n*; otherwise = 0, and $y_{1,n}$ = 1 if the algorithm selects a value 1 in flip trail *n* ; otherwise = 0.

We simplify and transform the likelihood function L to L^*

$$L^*(\beta_1, \beta_2) = \sum_{n=1}^{N} \left[y_{0,n} \log P_{0,n} + y_{1,n} \log P_{1,n} \right] \tag{5.9}$$

We then solve for the maximum of the log likelihood function L^* by differentiating it with respect to the parameters β_1 and β_2, and setting the partial derivatives equal to zero:

$$\frac{\partial L^*}{\partial \beta_k} = \sum_{n=1}^{N} \left\{ y_{0,n} \frac{\partial P_{0,n} / \partial \beta_k}{P_{0,n}} + y_{1,n} \frac{\partial P_{1,n} / \partial \beta_k}{P_{1,n}} \right\} = 0 \tag{5.10}$$

where: k = 1, 2

The maximum likelihood estimation for utility's parameter values in the predictive choice model is illustrated as follows. In Table 5.4, h is the total constraint violation, X^* is the value of variable X chosen in the flip trial.

Flip trial	Variable of interest (X)				X^*
	Current		Flipped		
	Value	h	Value	h'	
1	1	26	0	22	0
2	1	20	0	12	1
3	0	5	1	2	1
4	1	15	0	14	1

Table 5.4: The input data for maximum likelihood estimation

From (5.9), we get

$$L^*(\beta_k) = \{\log P_{0,1}\} + \{\log(1 - P_{0,2})\} + \{\log(1 - P_{0,3})\} + \{\log(1 - P_{0,4})\}$$

From the logit model assuming the logistic distribution:

$$L^*(\beta_k) = \left\{\log \frac{e^{\beta_1 + (\beta_2 \times h_{0,1})}}{e^{\beta_1 + (\beta_2 \times h_{0,1})} + e^{\beta_2 \times h_{1,1}}}\right\} + \left\{\log\left[1 - \frac{e^{\beta_1 + (\beta_2 \times h_{0,2})}}{e^{\beta_1 + (\beta_2 \times h_{0,2})} + e^{\beta_2 \times h_{1,2}}}\right]\right\}$$

$$+ \left\{\log\left[1 - \frac{e^{\beta_1 + (\beta_2 \times h_{0,3})}}{e^{\beta_1 + (\beta_2 \times h_{0,3})} + e^{\beta_2 \times h_{1,3}}}\right]\right\} + \left\{\log\left[1 - \frac{e^{\beta_1 + (\beta_2 \times h_{0,4})}}{e^{\beta_1 + (\beta_2 \times h_{0,4})} + e^{\beta_2 \times h_{1,4}}}\right]\right\}$$

where: $h_{0,n}$ and $h_{1,n}$ are the total violations when a decision variable is assigned a value 0 and 1 in flip trial *n*.

Substituting $h_{0,n}$ and $h_{1,n}$ from the Table 5.4

$$L^*(\beta_k) = \left\{\log \frac{e^{\beta_1 + 22\beta_2}}{e^{\beta_1 + 22\beta_2} + e^{26\beta_2}}\right\} + \left\{\log\left[1 - \frac{e^{\beta_1 + 12\beta_2}}{e^{\beta_1 + 12\beta_2} + e^{20\beta_2}}\right]\right\}$$

$$+ \left\{\log\left[1 - \frac{e^{\beta_1 + 5\beta_2}}{e^{\beta_1 + 5\beta_2} + e^{2\beta_2}}\right]\right\} + \left\{\log\left[1 - \frac{e^{\beta_1 + 14\beta_2}}{e^{\beta_1 + 14\beta_2} + e^{15\beta_2}}\right]\right\}$$

Then we obtain the values for β_k which maximise $L^*(\beta_k)$ via:

$$\frac{\partial L^*(\beta_k)}{\partial \beta_k} = 0$$

To find the maximum likelihood estimate, we have to solve a system of two non-linear equations in two unknowns. The parameter values can be obtained by using non-linear unconstrained optimisation methods, such as Newton-Raphson method (Kallenberg, 1997). An example of using the logit model for predicting a value of variable *X* with flip trials N = 20 is shown in Table 5.5 in which h_0 and h_1 are the total violations when a variable is assigned a value 0 and 1 respectively, X^* is the value of variable X chosen in the flip trial. To find the estimates of the parameter that best fits the observed violation history data, the statistical package SAS (SAS, 2002) based on the Newton-Raphson method is used. For this data set, the maximum likelihood method gives the choice specific parameter β_1 = 0.035 and the violation parameter β_2 = - 0.361.

Flip trial	Violation		X^*	Utility		Probability	
	h_0	h_1		U_0	U_1	P_0	P_1
1	20	18	1	-7.185	-6.498	0.33	0.67
2	13	8	1	-4.658	-2.888	0.15	0.85
3	16	15	0	-5.415	-5.741	0.58	0.42
4	13	8	1	-4.658	-2.888	0.15	0.85
5	11	12	0	-3.936	-4.332	0.60	0.40
6	9	11	1	-3.971	-3.214	0.32	0.68
7	19	18	1	-6.824	-6.498	0.42	0.58
8	13	16	1	-4.658	-5.776	0.75	0.25
9	33	29	1	-11.878	-10.469	0.20	0.80
10	15	8	1	-5.380	-2.888	0.08	0.92
11	20	18	0	-7.185	-6.498	0.33	0.67
12	13	8	1	-4.658	-2.888	0.15	0.85
13	15	16	1	-5.380	-5.776	0.60	0.40
14	13	8	1	-4.658	-2.888	0.15	0.85
15	11	12	0	-3.936	-4.332	0.60	0.40
16	9	11	0	-3.971	-3.214	0.32	0.68
17	19	18	1	-6.824	-6.498	0.42	0.58
18	13	16	1	-4.658	-5.776	0.75	0.25
19	33	29	1	-11.878	-10.469	0.20	0.80
20	15	8	1	-5.380	-2.888	0.08	0.92

Table 5.5: Probabilities of a value choice selection in flip trials

5.3.3.3 Aggregate prediction

Up to this point, we have focused on a model that predicts a choice of values for a variable in each flip trial *n*. However, the predictions for an individual flip trial may not reliably help the algorithm make a decision on what a good value for a variable would be. Instead, we are interested in an aggregate quantity, i.e. a prediction for the value choice based on a set of trials. We use the arithmetic mean of the total violation to represent the aggregate violation of N flip trials, which can be written as:

$$\bar{h}_0 = \sum_{n=1}^{N} \frac{h_{0,n}}{N} \text{ , and } \bar{h}_1 = \sum_{n=1}^{N} \frac{h_{1,n}}{N} \tag{5.11}$$

where: $\bar{h}_0$ and $\bar{h}_1$ are the average total violations when a variable is assigned a value 0 and 1 respectively.

To investigate the accuracy of forecast of the logit method, the container rail scheduling problem is scaled down. The optimal solution and their assigned values are known by running an integer programming branch and bound search to completion (ILOG, 2002). Then, we run CLS to collect the violation history with flip trials N = 20. The experimental results are given in Table 5.6, $\aleph^*$ is the number of occurrences of choice values in X^*, X^* is the value of variable X chosen in flip trial, Φ is the value of the corresponding variable in a known optimal solution. P_0 is calculated via (5.5) – (5.7).

Var no.	Violation		$\aleph^*$		Probability		Φ
	$\bar{h}_0$	$\bar{h}_1$	0	1	P_0	P_1	
1	28.45	19.65	3	17	0.15	0.85	1
2	24.60	23.90	9	11	0.51	0.49	0
3	23.30	26.25	15	5	0.75	0.25	0
4	23.45	24.50	11	9	0.52	0.48	1
5	23.05	25.60	18	2	0.90	0.10	0
6	12.30	18.50	18	2	1.00	0.00	0
7	9.95	21.00	12	8	0.71	0.29	0
8	11.70	13.70	8	12	0.46	0.54	1
9	24.10	18.85	14	6	0.30	0.70	0
10	21.95	25.05	14	6	0.70	0.30	0
11	26.70	20.55	6	14	0.30	0.70	1
12	26.80	27.15	15	5	0.75	0.25	0
13	10.55	18.25	20	0	1.00	0.00	0
14	15.50	14.45	11	9	0.56	0.44	1
15	10.15	17.40	20	0	1.00	0.00	0
16	21.85	16.30	3	17	0.15	0.85	1
17	20.95	24.90	14	6	0.70	0.30	0
18	20.25	25.00	18	2	0.90	0.20	0
19	25.15	24.85	12	8	0.51	0.49	1
20	26.75	26.60	14	6	0.61	0.39	0

Table 5.6: Value choice prediction by logit method

The percent correctly predicted for a set of variables, PC, is calculated as follows:

$$PC = \frac{t_c}{T} \times 100 \tag{5.12}$$

where: t_c is the number of variables correctly predicted, T is total number of predicted variables.

Table 5.6 shows that our predicted value for each binary variable is relatively close to its known optimal value with PC = 85%. For variables 4, 9, and 19, the choice model predicts a wrong optimal value. However, PC is sufficiently high to allow us to have some confidence in the logit method.

5.3.3.4 Simplified estimation

Until now, the parameters β_1 and β_2 have been estimated by the maximum likelihood method. Unfortunately, this method requires a significant computational effort and needs to be applied many times during the search.

In this section, we introduce a simplified estimation of the utility's parameter values. Since the number of occurrences of choice values in X^* will be treated separately by the proportional method, the logit method only accounts for the total constraint violation. However, in the logit method, we may have to set the choice specific parameter β_1 to a small value, e.g. β_1 = 0.05, so that the utility of one alternative is preferred to the other. This

is because an equal utility lies outside the assumption of the choice theory (Ben-Akiva and Lerman, 1985).

Instead of considering $\bar{h}_0$ and $\bar{h}_1$ separately, the absolute difference between $\bar{h}_0$ and $\bar{h}_1$, $\Delta\bar{h}$ is used in order to characterise the value choice selection. $\Delta\bar{h}$ is defined as follows:

$$\Delta\bar{h} = \left|\frac{\bar{h}_0 - \bar{h}_1}{\bar{h}_0 + \bar{h}_1}\right| \tag{5.13}$$

where: $\bar{h}_0$ and $\bar{h}_1$ are the average total violations when a variable is trial flipped or assigned a value 0 and 1 respectively.

From (5.13), when the value of $\Delta\bar{h}$ is large, the probabilities of two alternatives (value 0 and 1) would be significantly different, and when $\Delta\bar{h}$ = 0, the probabilities of the two alternatives would tend to be equal. $\Delta\bar{h}$ is shown in a proportional scale so that the formulation could be generalised for the problem in which the total violation and the number of flip trials can be varied. Then we present a simplified estimation of the violation parameter β_2 as follows:

$$\beta_2 = -\Delta\bar{h} \tag{5.14}$$

Now finding the utility's parameter values only requires a very little computational effort. It is noted that in this study β_1 could either be set to any small positive or negative value,

whilst β_2 is always set to be negative because the constraint violation represents disutility, i.e. value 0 is preferred to 1 when $\overline{h}_0$ is less than $\overline{h}_1$.

Table 5.7 show the results obtained by the predictive choice model using the simplified estimation for logit method. The experiment uses the same test data as shown in Table 5.6. For all test variables, the number of flip trials N = 20, the choice specific parameter β_1 is set to 0.05, and the decision parameter D is set to 70 (more experimental results are shown in Appendix B)

Table 5.7 shows that the predicted value for the test variable is close to its known optimal value with PC = 75%. This PC value is sufficiently high to allow us to have some confidence in the predictive choice model, and indicates that the result obtained by the simplified estimation for logit method serves almost as well as the result from the more complicated mathematical one used in Table 5.6.

In Tables 5.6 – 5.7 (and Appendix B), we observe that the predictions when P_0 is between 0.45 – 0.55 are not very accurate. Therefore, the predictive choice model discards those predictions in order to increase the accuracy of forecast for all variables.

Var no.	Violation		$\aleph^*$		Probability		Φ
	$\bar{h}_0$	$\bar{h}_1$	0	1	P_0	P_1	
1	14.35	16.15	7	13	0.54	0.46	1
2	11.80	22.25	17	3	0.85	0.15	0
3	10.75	21.00	9	11	0.97	0.03	0
4	19.85	16.75	17	3	0.85	0.15	0
5	12.60	19.35	15	5	0.75	0.25	0
6	12.30	18.50	18	2	0.90	0.10	0
7	9.95	21.00	12	8	0.98	0.02	0
8	11.70	13.70	8	12	0.55	0.45	1
9	8.25	19.00	11	9	0.99	0.01	0
10	9.90	20.65	12	8	0.98	0.02	0
11	9.60	15.50	19	1	0.95	0.05	0
12	14.70	13.05	8	12	0.49	0.51	1
13	10.55	18.25	20	0	1.00	0.00	0
14	15.50	14.45	11	9	0.50	0.50	1
15	10.15	17.40	20	0	1.00	0.00	0
16	15.20	12.25	2	18	0.10	0.90	1
17	14.30	12.41	12	8	0.48	0.52	0
18	12.20	12.10	14	6	0.70	0.30	0
19	13.40	11.50	6	14	0.30	0.70	1
20	12.98	14.40	11	9	0.53	0.47	1

Table 5.7: Value choice prediction by the simplified estimation for logit method.

As an outcome of the predictive choice model is a probability of choosing value for a variable, the variable will be fixed at its predicted value for a number of iterations

determined by the magnitude of the probability. During these iterations other variables may become fixed. When the fixing iteration for a variable is reached, it is freed and its violation history is refreshed. Before we describe how the predictive choice model is used to improve the container rail schedule obtained by CLS, related work on the construction and use of probabilistic models in the search algorithm will be discussed.

5. 3.3.5 Related work

The predictive choice model has some general similarities with both adaptive techniques used in local search and population-based search techniques. For example, algorithms based on these techniques are: adaptive Tabu search (Glover and Laguna, 1997), variable neighbourhood search (Hansen and Mladenovic, 2001), ant colony optimisation (Blum et al, 2001), estimation of distribution algorithms (EDAs) (Pelikan et al, 1999), etc. These algorithms do not rely on problem specific designs of heuristics and do not need many tuning parameters. Instead they build probabilistic models to keep fixed some variables during the search or to predict the movements of populations in a probabilistic way. Such probabilistic models attempt to draw inferences specific to the complex combinatorial problem being solved and therefore may be regarded as adaptive processes that learn domain knowledge implicitly.

In most local search algorithms based on probabilistic models, the problem specific interactions amongst the decision variables in the combinatorial optimisation model are kept in mind implicitly, whereas in our predictive choice learning algorithm, as well as EDAs, the interactions are treated explicitly through the probability distribution associated with the variables selected at each sample. In EDAs, often incorporated in genetic algorithm

(Goldberg, 1989), a probabilistic model for selecting promising solutions is constructed, based on the observed probability distribution. The new solutions are generated by the probabilistic model. The observed distributions of the interactions amongst variables are estimated and are often displayed as complex chains or networks.

We consider the deterministic and non-deterministic (random) interactions amongst the variables separately. The logistic probability distribution of the non-deterministic behaviour in choosing a good value for a variable is assumed in order to derive a specific probabilistic model; thereby it makes our model easy to develop and convenient to use. The model learns from the search history, the outcome in terms of a probability is used to influence the search.

In general, the probability distribution in EDAs is categorised into three types according to the interactions amongst the variables (Larraaga and Lozano, 2002). These are: no interactions, pairwise interactions, and multivariate interactions. No interactions assumes that all variables in a problem are independent, i.e. the search algorithm only looks at the values of each variable regardless of the remaining variables. The second type assumes that the variables in the combinatorial optimisation model are largely independent except for some pairwise interactions. In this type, the joint probability distribution of the interactions between pairwise variables is constructed. The last is the most complex and requires significant computational effort, but in return for a potentially better result. It assumes multivariate interactions amongst the variables. The variables may be divided into a number of independent clusters, and the conditional probabilities are estimated within each cluster. As the search history for our predictive choice model is based on a two-variables selection scheme, it is closely related to the pairwise interactions of EDAs. In our model, the joint

probability distribution of the interaction between variables is estimated by observing the constraint violations resulting from assigning values for two variables.

In addition, the application of our predictive choice model and EDAs is different. In EDAs, the model is used to predict the movements of the solution (populations) in the search space. For instance, when applied to genetic algorithms (Pelikan et al, 1999), EDA generates new solutions by sampling the constructed probability distribution of the promising solutions (i.e. EDAs do not use crossover and mutation). In our application, the probabilistic model is used to predict a choice of good value for an individual binary variable. With sufficient trial history, the model predicts likely optimal values for variables.

The next section describes how the predictive choice model is used to improve the solution of the container rail scheduling problem.

5.4 CLS incorporating the predictive choice model

In our algorithm, the predictive choice model is used as the search intensification strategy in order to improve on solutions using CLS alone. As discussed in Section 5.3.1, after some iterations, the search history is analysed, some variables may appear to have high preference measure of having certain good values, and they will be fixed for a number of iterations. At that point, consistency between variables is enforced. When the fixing iteration limit is reached, the variable is freed, together with those other variables which were fixed by consistency enforcing. The procedure of CLS incorporating the predictive choice model is outlined as follows:

Step 1: The algorithm starts using CLS (as in Chapter 4) to perform the normal searching process and to collect the violation history.

Step 2: For each variable separately: when it has been flip-tested N times, the algorithm calculates the preferred value and proportional preference measure P; where $P = \max(P_0, P_1)$. If P is greater than or equal to the decision parameter D, it uses this measure, otherwise it recalculates P using the logit method. Now if P is less than the prediction error parameter E, there is no preferred value and no fixing is done. Otherwise the variable is fixed at its preferred value for a number of iterations equal to $P \times F$; where F is a predetermined number of fixing iterations. However, if the number of fixed variables becomes greater than the limit on the maximum number of fixing variables, then a variable may become unfixed before this number of iterations. In this case the variable chosen to be unfixed early is the one which has been fixed for the longest. This limit controls how many variables can be fixed and is necessary to allow some flexibility in the search.

Step 3: Fixing a variable may imply that other variables should be fixed. At this point the algorithm enforces the consistency between variables. See the cases in Sections 5.4.1 – 5.4.2 for how this should be done.

Step 4: When a variable becomes unfixed, it stays unfixed until it has been flip-tested a further N times and the process above is repeated using these N observations. Other variables which were fixed at the same time as this

variable in order to maintain consistency are also unfixed at the same time as this variable.

Step 4: The algorithm stops when the iteration limit Z is reached.

In the container rail scheduling model, the timeslot and customer's booking variables (x_t and y_{tj}) are chosen for flip trials, but propagation of consistency may not be carried out fully all the time. At the beginning, the propagation of consistency is only maintained within each of the set x_t and y_{tj}, but not across the two sets of variables, in order to promote wider exploration of the search space. However, after a specified number of iterations, the trial history is analysed. As a result, some variables will be fixed at the preferred value given by the predictive choice model for a number of iterations. At this point, consistency between timeslots and customer's booking variables is enforced, leading to intensified exploration of the neighbouring search space.

5.4.1 Timeslot enforcing

When the trial history is analysed and the timeslot variable x_t is predicted to have a particular value, the variable will be fixed at that value for the number of iterations determined by the magnitude of the preference measure. In this case, all customers' booking variables y_{tj} associated with timeslot x_t are assigned values consistent with the preferred value of x_t. For example, if a preferred value of x_t = 0 (timeslot t is not selected), no customer's booking will be assigned to that timeslot, i.e. y_{tj} = 0 for all potential customers

in that timeslot. On the other hand, if the preferred value of x_t = 1 (timeslot t is selected), any customer's booking may or may not be assigned to that timeslot.

As the timeslot variable x_t may take a current value 0 or 1 and its preferred value can either be 0 or 1 during the search processes, we categorise the timeslot consistency enforcing into four cases. To simplify our discussion, the following will be used.

States of variables:

N_f : number of iterations to be fixed (for each variable)

(N_f = probability P × number of fixing iterations F)

N_x : number of fixing x_t variables

N_0 : number of y_{tj} variables fixed at 0

N_1 : number of y_{tj} variables fixed at 1

Parameters:

F : number of fixing iterations

$nbFixX$: limit on the max. length of the fixing list, X , for x_t variables

$nbFixY_0$: limit on the max. length of the fixing list, Y_0 , for y_{tj} variables fixed at 0

$nbFixY_1$: limit on the max. length of the fixing list, Y_1 , for y_{tj} variables fixed at 1

The steps of propagation of consistency for x_t are given as follows:

Let K_j be the timeslot chosen for customer j.

U_j be the set of alternative timeslots for customer j not used, i.e. $U_j = S_j / \{K_j\}$;

where S_j is the set of potential booking timeslots for customer j

---

Case 1: current value of $x_t = 0$, preferred value for $x_t = 0$

Step 1 If $N_x \leq nbFixX$, fix x_t at 0 for N_f iterations, add x_t to the end of the fixing list X, $N_x \leftarrow N_x + 1$.

Otherwise release the first x variable in X, $N_x \leftarrow N_x - 1$, reorder X and repeat step 1.

Step 2 For all j: if $t = K_j$, flip y_{tj} to 0, select randomly $t' \in U_j$,

$U_j \leftarrow (U_j \setminus \{t'\}) \cup \{t\}$, $y_{t'j} \leftarrow 1$,

fix y_{pj} at 0; $\forall p \in U_j$ for N_f iterations, $N_0 \leftarrow N_0 + |U_j|$.

Step 3 If total hard violation = 0, exit with a feasible solution.

Step 4 Release any x variable in X if its fixing iteration limit has been reached and reorder X.

Case 2: current value of x_t = 0, preferred value for x_t = 1

Step 1 If $N_x \leq nbFixX$, flip x_t to 1, select randomly an assigned timeslot t', $t' \neq t$, $x_{t'} \leftarrow 0$,

fix x_t at 1 for N_f iterations, add x_t to the end of X, $N_x \leftarrow N_x + 1$.

Otherwise release the first x variable in X, $N_x \leftarrow N_x - 1$, reorder X and repeat step 1.

Step 2 If total hard violation = 0, exit with a feasible solution.

Step 3 Release any x variable in X if its fixing iteration limit has been reached and reorder X.

Case 3: current value of x_t = 1, preferred value for x_t = 0

Step 1 If $N_x \leq nbFixX$, flip x_t to 0, select randomly an unassigned timeslot t', $t' \neq t$, $x_{t'} \leftarrow 1$,

fix x_t at 0 for N_f iterations, add x_t to the end of X, $N_x \leftarrow N_x + 1$.

Otherwise release the first x variable in X, $N_x \leftarrow N_x - 1$, reorder X and repeat step 1.

Step 2 For all j: if $t = K_j$, flip y_{tj} to 0, select randomly $t' \in U_j$, $t' \neq t$

$U_j \leftarrow (U_j \setminus \{t'\}) \cup \{t\}$, $y_{t'j} \leftarrow 1$,

fix $y_{pj} = 0 \ \forall p \in U_j$ for N_f iterations, $N_0 \leftarrow N_0 + |U_j|$.

Step 3 If total hard violation = 0, exit with a feasible solution.

Step 4 Release any x variable in X if its fixing iteration limit has been reached and reorder X.

Case 4: current value of x_t = 1, preferred value for x_t = 1

Step 1 If $N_x \leq nbFixX$, fix x_t at 1 for N_f iterations, add x_t to the end of X, $N_x \leftarrow N_x + 1$.

Otherwise release the first x variable in X, $N_x \leftarrow N_x - 1$, reorder X and repeat step 1.

Step 2 Release any x variable in X if its fixing iteration limit has been reached and reorder X.

5.4.2 Customer's bookings enforcing

For the customer's bookings enforcing, the booking variable y_{tj} may also take a current value 0 or 1 and its preferred value can either be 0 or 1 during the search. It is noted that consistency within y_{tj} is maintained during the search by the coverage constraint, i.e. a customer's booking can only be assigned into one timeslot. However, it may or may not be consistent with a set of selected timeslot variables. To better control the customer's booking enforcing, we categorise it into two groups: local and global fix.

5.4.2.1 Local fix

The local fix is a process of preventing the algorithm selecting a customer's booking in timeslot t that may not lead to a feasible solution, i.e. fixing y_{tj} at 0 for a number of iterations. Although locally fixing y_{tj} at 0 will not enforce the consistency between x_t and y_{tj}, the number of potential booking timeslots for each customer is reduced quickly. The local fix is categorised into two cases:

Case 1: current value of y_{tj} = 0, preferred value for y_{tj} = 0

Step 1 If $N_0 \leq nbFixY_0$, fix y_{tj} at 0 for N_f iterations; add y_{tj} to the end of the fixing list Y_0, $N_0 \leftarrow N_0 + 1$.

Otherwise release the first y variable in Y_0, $N_0 \leftarrow N_0 - 1$, reorder Y_0 and repeat step 1.

Step 2 Release any y variable in Y_0 if its fixing iteration limit has been reached and reorder Y_0.

Case 2: current value of y_{tj} = 1, preferred value for y_{tj} = 0

Step 1 If $N_0 \leq nbFixY_0$, flip y_{tj} to 0, select randomly $t' \in U_j$, $t' \neq t$,

$U_j \leftarrow (U_j \setminus \{t'\}) \cup \{t\}$, $y_{t'j} \leftarrow 1$,

fix y_{tj} at 0 for N_f iterations, add y_{tj} to the end of Y_0, $N_0 \leftarrow N_0 + 1$. Otherwise release the first y variable in Y_0, $N_0 \leftarrow N_0 - 1$, reorder Y_0 and repeat step 1.

Step 2 If total hard violation = 0, exit with a feasible solution.

Step 3 Release any y variable in Y_0 if its fixing iteration limit has been reached and reorder Y_0.

5.4.2.2 Global fix

The global fix provides a strong propagation of consistency between customer's booking and timeslot variables. When the global fix is called (i.e. a preferred value of $y_{tj} = 1$), one potential timeslot t is assigned to customer j. A train will run at that time in order to serve the demand for customer *j*, this prevents the algorithm selecting the remaining potential timeslots for that customer. The global fix is also categorised into two cases:

Case 1: current value of y_{tj} = 0, preferred value for y_{tj} = 1

Step 1 If $N_1 \leq nbFixY_1$, flip y_{tj} to 1,

$y_{t'j} \leftarrow 0$; $t' = K_j$,

fix y_{tj} at 1 for N_f iterations, add y_{tj} to the end of Y_1,

$N_1 \leftarrow N_1 + 1$.

Otherwise release the first y variable in Y_1, $N_1 \leftarrow N_1$ - 1, reorder Y_1 and repeat step 1.

Step 2 If x_t = 1, fix x_t at 1 for N_f iterations, add x_t to the end of X,

$N_x \leftarrow N_x + 1$.

Otherwise flip x_t to 1, select randomly an assigned timeslot $t', t' \neq t$,

$x_{t'} \leftarrow 0$, fix x_t at 1 for N_f iterations, add x_t to the end of X,

$N_x \leftarrow N_x + 1$.

Step 3 If total hard violation = 0, exit with a feasible solution.

Step 4 Release any y variable in Y_1 if its fixing iteration limit has been reached and reorder Y_1.

Case 2: current value of y_{tj} = 1, preferred value for y_{tj} = 1

Step 1 If $N_1 \leq nbFixY_1$, fix y_{tj} at 1 for N_f iterations, add y_{tj} to the end of Y_1, $N_1 \leftarrow N_1 + 1$.

Otherwise release the first y variable in Y_1, $N_1 \leftarrow N_1$ - 1, reorder Y_1 and repeat step 1.

Step 2 If x_t = 1, fix x_t at 1 for N_f iterations, add x_t to the end of X, $N_x \leftarrow N_x + 1$.

Otherwise flip x_t to 1, select randomly an assigned timeslot $t', t' \neq t$, $x_{t'} \leftarrow 0$, fix x_t at 1 for N_f iterations, add x_t to the end of X, $N_x \leftarrow N_x + 1$.

Step 3 If total hard violation = 0, exit with a feasible solution.

Step 4 Release any y variable in Y_1 if its fixing iteration limit has been reached, and reorder Y_1.

5.5 Computational results

In this section, we demonstrate the performance of the constraint-based local search incorporating the predictive choice model (PCM), and compare the results with constraint-based local search (CLS) alone. The results are shown in terms of number of trains, and the

generalised cost. The generalised cost includes the operating costs and the virtual revenue loss representing customer satisfaction on the given schedules.

Eight weeks data from a case study are run on the same PC-Pentium 2.4 GHz. Each test case is run ten times using different random numbers at the beginning of each run. Before running computational experiments, we need to find good values for parameters here. After some initial experimentation, the values of the parameters were chosen to be: the number of flip trials N = 20, decision parameter D = 75, the prediction error parameter E = 55%, number of fixing iterations F = 100, and the limits on the number of fixed variables, $nbFixX$, $nbFixY_1$, $nbFixY_0$ = 50, 50, 200 respectively.

The parameters $nbFixX$, $nbFixY_1$, $nbFixY_0$ govern the maximum number of variables which are allowed to be fixed in the search and are necessary to allow some flexibility in the search. These parameters are closely related to F ; if F is not too large, the limit parameters may never be reached and thus they become unnecessary. Otherwise, the limit parameters could help the algorithm spread out the number of fixed variables to different sets of variables so that the diversified intensification of the search space may be achieved. In this experiment, we set $nbFixX$, $nbFixY_1$, $nbFixY_0$ = 50, 50, and 200. For $nbFixX$ and $nbFixY_1$ the values are estimated by 1/3 to 1/4 of the schedulable timeslots x_t and customers M respectively, and $nbFixY_0$ is about 1/5 to 1/10 of y_{tj}. Once the values of these parameters have been chosen, we may then only need a good value of F .

We tried different values of the stopping criterion parameter Z , ranging from 500 to 5000 iterations for all test cases, and observed that Z = 2000 iterations is large enough as

solutions do not tend to be improved after this limit. Note that we do not claim here that these values of the parameters above are the best values, but some care was taken. In addition, we will test the sensitivity of the parameters D, E, N and F after the computational results are given. The results for the test cases are shown in Table 5.8. The cost is a generalised cost ($\times 10^6$ Baht).

Test case	CLS				PCM					
	Trains		*GC*	Time	Trains			*GC*		Time
	Min	Avg.	Avg.	(Sec)	Min	Avg.	Std.	Avg.	Std.	(Sec)
W1	49	50.87	5.47	387	47	48.15	1.82	4.66	0.53	257
W2	37	38.68	3.95	168	36	37.18	1.17	3.37	0.24	178
W3	24	24.74	2.06	89	23	23.75	0.56	1.86	0.10	144
W4	41	42.21	4.48	169	39	41.66	1.72	4.04	0.62	219
W5	68	69.44	7.01	930	64	66.09	2.25	6.55	0.98	837
W6	56	58.80	6.18	655	53	54.93	1.06	5.39	0.49	506
W7	44	45.11	4.85	297	43	43.51	0.70	4.35	0.29	236
W8	79	80.64	8.06	1980	75	77.94	1.75	7.87	0.91	1080

Table 5.8: Results obtained by CLS and PCM

Table 5.8 shows that the results of PCM are better than those of CLS in terms of the number of trains and the generalised cost GC. Although the PCM learning from the search history implies a computational overhead over CLS, it is offset against a lower run-time required to find good schedules, in particular for large test cases.

To test the sensitivity of parameters in the algorithm, four additional experiments are carried out as follows.

5.5.1 Decision parameter D

We first test different values for the decision parameter D. This parameter is especially important since the algorithm is switching between the proportional and logit methods. We need to examine whether switching is of value. Although the results in Table 5.2 showed the proportional method alone predicts a wrong optimal value for many variables and suggests the use of logit method, we may also want to find a good setting of *D*. In this experiment, each test case is run five times, D is varied and the remaining parameters in the algorithm are fixed. The results are shown in Table 5.9.

Test	Trains (Avg.)				Cost (Avg.)			
Case	D=55	70	85	100	D=55	70	85	100
W1	53	49	51	51	6.01	5.02	5.65	5.68
W2	42	38	40	41	4.46	3.74	4.15	4.21
W3	26	24	24	25	2.88	2.51	2.34	2.69
W4	45	42	43	43	5.11	4.30	4.49	4.49
W5	73	68	67	71	7.42	7.47	6.71	6.92
W6	61	54	56	59	6.19	5.74	5.87	5.95
W7	46	44	43	44	4.71	4.33	4.19	4.51
W8	84	79	81	82	8.43	7.71	7.90	8.26

Table 5.9: Sensitivity analysis of the parameter D

Table 5.9 shows that in the extreme cases D = 55 (always use proportional method) and D = 100 (always use logit method) the algorithm finds lower quality results. When both cases are compared, the logit method provides better results than the proportional method and in some of the test cases it is as good as the results obtained from the combination of the two methods (i.e. when D = 70 and D = 85). This may be because logit method considers the amount by which one value is preferable to the other in the preference measure which tends to be a main factor in value choice selection, but when an occurrence of any choice value X^* is obviously dominating, the logit method becomes less effective as the measure of preferred value tends to be compensated with the measure of the other value in a choice function. The results suggest that choosing D between 70 – 85 is reasonable as the algorithm performs well in all test cases.

5.5.2 Prediction error parameter E

This experiment is to test the prediction error parameter E. As in our experiments (in Tables 5.6-5.7 and Appendix B) the predictions when P_0 is between 0.45-0.55 are not very accurate. In such cases, if $P_0 < E\%$ there is no preferred value and no fixing is done. The parameter E = 50% means that there is always a preferred value for a variable to be fixed. In this experiment, each test case is run five times, E is varied and the remaining parameters in the algorithm are fixed. The results are shown in Table 5.10.

Test Case	Trains (Avg.)				Cost (Avg.)			
	E =50	55	60	65	E =50	55	60	65
W1	51	50	50	51	5.16	4.95	5.05	5.23
W2	37	37	38	39	3.97	3.76	4.02	4.25
W3	25	24	25	26	2.50	2.42	2.61	2.72
W4	42	42	42	43	4.19	4.24	4.35	4.41
W5	67	67	69	69	6.76	6.63	7.31	7.84
W6	57	58	59	60	6.08	6.15	6.42	6.53
W7	44	44	44	45	4.46	4.55	4.60	5.02
W8	79	78	80	81	7.62	7.53	8.07	9.19

Table 5.10: Sensitivity analysis of the parameter E

Table 5.10 shows that setting E greater than 60 gives a relatively low quality of results, but it still gets slightly better results compared with the results obtained by using CLS alone, where no variable fixing is done. The smaller values of E (E = 50 and E = 55) show some improvement of solutions and choosing E = 55 is slightly better than E = 50. However, no clear difference is observed and the initially chosen value appears satisfactory.

5.5.3 Flip trial parameter N

This experiment is to test the number of flip trials parameter N. In this experiment, each test case is run five times, N is varied and the remaining parameters in the algorithm are fixed. The results are shown in Table 5.11.

Test Case	Trains (Avg.)				Cost (Avg.)			
	N =10	20	30	50	N =10	20	30	50
W1	49	47	48	51	5.08	4.97	5.00	5.25
W2	39	39	40	42	4.11	3.66	4.06	4.25
W3	24	24	24	25	2.06	1.97	2.01	2.15
W4	42	41	41	43	4.32	3.98	3.95	4.70
W5	71	67	68	73	7.50	6.62	6.88	7.77
W6	58	54	53	59	5.93	5.79	5.56	6.18
W7	44	43	43	46	4.79	4.34	4.62	4.95
W8	80	78	79	84	8.39	7.83	8.28	9.02

Table 5.11: Sensitivity analysis of the parameter N

Table 5.11 shows that choosing N between 10 and 30, the algorithm performs well in most cases. In general, we get slightly better results when N is set to 20. It is noted that a high value of N = 50 provides low quality results, both in terms of the number of trains and the generalised cost. This may be because a large number of flip trials must be collected in the search history before the predictive choice model can be used; as a result it is seldom used.

5.5.4 Fixing iterations parameter F

The other experiment is to test sensitivity of the number of fixing iterations parameter F. This parameter is designed to guard against inaccurate forecasts by the predictive choice model. When F is not too large, and the model predicts a wrong value for a variable, that variable will be refreshed after a small number of iterations; note also that the limits on the

number of fixing variables $nbFixX$, $nbFixY_1$, and $nbFixY_0$ may not be reached in this case. The sensitivity test of F is shown in Table 5.12. In this experiment, each test case is run five times, F is varied and the remaining parameters in the algorithm are fixed.

Test case	Trains (Avg.)				Cost (Avg.)			
	F =50	100	200	300	F =50	100	200	300
W1	48	48	49	50	5.00	4.88	5.04	5.13
W2	41	39	40	41	4.08	4.03	4.07	4.46
W3	24	24	24	25	2.01	1.97	2.03	2.12
W4	42	41	43	43	4.37	3.91	4.57	4.50
W5	72	71	68	74	7.50	7.40	6.80	7.64
W6	53	54	59	61	5.67	5.65	6.14	6.21
W7	44	43	44	45	4.52	4.39	4.47	4.75
W8	79	77	80	85	8.19	7.70	8.50	8.64

Table 5.12: Sensitivity analysis of the parameter F

Table 5.12 shows that the number of fixing iterations F affects the quality of schedule, both positively and negatively. Choosing F = 50, the results are relatively good, but when F is increased to 100, the results tend to be improved slightly. However, setting large a value for F between 200 and 300 decreases the effectiveness of the algorithm in most cases. This may be because an incorrect forecast by the predictive choice model for some variables prevents the search finding feasible solutions for the number of fixing iterations.

5.6 Conclusions

The construction and use of the predictive choice model is presented in this chapter. The predictive choice model is based on discrete choice theory and the random utility concept. The model explicitly considers the inconsistencies of the algorithm in choosing a good value for binary variable in a probabilistic way. We have shown that the constraint-based local search algorithm incorporating the predictive choice model is able to improve the container rail schedules obtained by constraint-based local search alone.

With sufficient trial history, the predictive choice model will predict a good choice of value for a variable. The variable will be fixed at its predicted value for a number of iterations determined by the magnitude of an associated probability. At this point, the propagation of consistency between the variables is enforced, leading to intensified exploration of the search space.

This intensification technique is novel because it is the first time that the predictive choice model has been tailored for a local search method (in this case, the constraint-based local search algorithm) and applied to a combinatorial optimisation problem. Experiments based on real life data demonstrate the strengths and usefulness of the proposed technique.

Even though the constraint-based local search incorporating the predictive choice model has been developed to solve a specific container rail scheduling problem, it can be adapted to other combinatorial optimisation problems. In Chapter 6, the application of this approach to the generalised assignment problem is described.

Chapter Six

Constraint-based local search for the generalised assignment problem

---

6.1 Introduction

The computational results presented in Chapter 5 demonstrate that the incorporation of the predictive choice model in constraint-based local search leads to an effective algorithm for the container rail scheduling problem. However, the solution approach, especially the predictive choice model, is not strongly dependent on problem-specific knowledge. It seems appropriate to investigate whether this approach can be successfully applied to other combinatorial optimisation problems. In this chapter we apply the approach to one such problem, the generalised assignment problem (GAP).

GAP is a difficult combinatorial optimisation problem, which is known to be *NP*-hard (Sahni and Gonzalez, 1976). GAP considers the minimum cost assignment of n jobs to m agents such that each job is given to one and only one agent subject to resource capacity constraints on the agents. GAP has several applications in industry such as computer and

communication networks, facility location, vehicle routing, manufacturing systems, resource scheduling, etc (Chu and Beasley, 1997).

The container rail scheduling problem (described in Chapter 3) has some similarities to GAP in that the demand (resource consumed) assigned to a departure timeslot must not exceed the capacity of a train (resource capacity) and each demand can only be served by one train (job is processed by only one agent). However, the container rail scheduling problem has identical train capacity constraint for all timeslots, whilst in GAP, the capacity of the agents are different. In addition, in the container rail scheduling problem, timeslot consistency has to be maintained, i.e. if a timeslot is selected for a customer, a train has to depart at that timeslot. Each customer has a number of potential booking time ranges sparsely distributed through the available departure timeslots and a rail carrier attempts to find the minimum number of timeslots to serve all the demand, in contrast to GAP in which all agents are available to process jobs.

Data sets publicly available in OR-library (reproduced by Chu and Beasley, 1997) are used to test our constraint-based local search algorithm (CLS) for GAP.

This chapter is organised as follows: Section 6.2 describes GAP and reviews some solution methods. A formulation of the problem is given in this section. Section 6.3 describes the procedure of CLS for GAP. Section 6.4 illustrates the search intensification technique using the predictive choice model. Computational experiments with constraint-based local search incorporating the predictive choice model are given in Section 6.5 and compared with other solution methods. Finally conclusions are given.

6.2 Generalised assignment problem

GAP is a problem of assigning jobs to agents with minimum total cost such that

- each job must be assigned to exactly one agent
- each agent requires a known amount of a single resource and incurs a known cost to perform each job
- the cost and resource requirements of agent, job pairs may be different
- each agent has a limited amount of the resource

This problem is well-known and proved to be *NP*-hard. Finding a feasible solution for GAP is also *NP*-hard (Sahni and Gonzalez, 1976; Narciso and Lorena, 1999). In this research, we consider GAP as a maximisation problem, i.e. jobs are processed by agents with maximum total profit.

6.2.1 Related work

In this section, we present a review of solution methods found in the literature for GAP. The solution methods can be categorised into two groups: exact and heuristic methods. An extensive survey on previous solution methods was done by Chu (1997). However, exact methods have a computational disadvantage in solving large-size problems. We focus on heuristic methods that can find good quality solutions within a reasonable computational time.

Osman (1995) proposed a hybrid algorithm, SA/TS, which combines simulated annealing and tabu search. The algorithm uses a λ - generation mechanism which describes how a solution can be altered to generate neighbour solutions. In SA/TS, the non-monotonic cooling scheme of simulated annealing and the oscillation strategy of tabu search are used, which are considered as a hybrid strategy. Osman also proposed a tabu search alone for GAP. Both the SA/TS and TS use a frequency-based memory that records information used for diversification purposes.

Chu and Beasley (1997) proposed a GA-based heuristic for solving GAP. The heuristic incorporates a problem-specific encoding of a solution structure. Fitness-unfitness pair evaluations are used to handle both feasible and infeasible solutions. Apart from using mutation and crossover operators, a two-phase heuristic improvement operation is used. In the first phase, the operator tries to recover feasibility by reducing the unfitness score. The second phase is to improve the cost of the solution without further violating the capacity constraints.

Yagiura et al (1999) proposed a variable depth search algorithm for GAP. The algorithm incorporates an adaptive use of modified shift and swap neighbourhoods where some moves are tabu in order to overcome local minima. The method also allows the search to visit infeasible solutions, modifying the objective function to penalise the overloaded capacity of the agents. Yagiura et al (2004) also proposed a tabu search with an ejection chain for GAP. The ejection chain approach is embedded in a neighbourhood construction in order to create more complex and powerful local moves. The ejection chains are constructed by using the information from a Lagrangian relaxation of the problem. The sorted cost for each job is also maintained and used in a variable selection process in order to make a move more

efficiently. In addition, the algorithm incorporates an automatic mechanism for adjusting search parameters to maintain a balance between visits to feasible and infeasible regions.

Diaz and Fernandez (2001) proposed a tabu search heuristic for GAP. They presented a relaxed formulation of GAP that allows the search to cross the capacity-infeasibility boundary using a penalty term. A candidate move strategy is also used in which the move with lowest cost is selected and performed. A strategic oscillation scheme is then used to permit alternating between feasible and infeasible solutions. Search diversification and intensification strategies are implemented by means of frequency-based memory.

Lourenco and Serra (2002) proposed hybrid meta-heuristic search techniques for GAP. The algorithm is based on a greedy randomised adaptive heuristic and a MAX-MIN ant system that takes into consideration the search information gathered in earlier iterations of the algorithm in order to construct a good initial solution. In this phase, a relative resource consumed probability is used by considering the ratio of resource to agent capacity. The agent that has the highest probability is chosen for an initial solution. Then, a descendent local search and tabu search are used to improve the search. Several neighbourhoods are studied and used, including one based on ejection chains that can produce good moves.

Feltl and Raidl (2004) proposed an improved hybrid genetic algorithm for GAP. The algorithm is based on the hybrid genetic algorithm proposed by Chu and Beasley (1997). The algorithm includes two initialisation heuristics; constraint-ratio and linear programming heuristics to create more promising solutions, which are mostly feasible. In addition, a modified selection and replacement strategy in GA is used, which assigns infeasible

solutions a fitness value depending only on the relative capacity excess and always ranks them worse than any feasible solution.

6.2.2 Problem formulation

We model GAP as a constraint satisfaction problem (CSP) in order to introduce a constraint-based local search (CLS) to solve this class of CSP. The following notation will be used.

Sets:

I : set of agents, = {1,2,3,…, m }

J : set of jobs, = {1,2,3,…, n }

Parameters:

p_{ij} : assignment profit of job j to agent i

a_{ij} : resource required for processing job j by agent i

b_i : capacity of agent i

Decision variable:

x_{ij} : 1, if job j is assigned by agent i, 0 otherwise

In a CSP, optimisation criteria and operational constraints are represented as soft and hard constraints respectively. A feasible solution for a CSP is an assignment to all constrained variables in the model that satisfies all hard constraints, whereas an optimal solution is a feasible solution with the minimum total soft constraint violation. In GAP, the soft constraint is the maximum total profit of assigning jobs to agents. The hard constraints are capacity constraints and coverage constraints.

6.2.2.1 Soft constraint

Maximum total profit - this constraint aims to maximise total profit of assigning jobs to agents, which is defined as:

$$\sum_{i \in I} \sum_{j \in J} p_{ij} x_{ij} \geq \theta \tag{6.1}$$

where: θ is an upper bound on total profit, e.g. $\theta = \sum_{j \in J} \max\{p_{ij} : i \in I\}$.

6.2.2.2 Hard constraints

Coverage constraint **-** this constraint ensures that each job is assigned to exactly one agent, which is defined as:

$$\sum_{i \in I} x_{ij} = 1; \qquad \forall j \in J \tag{6.2}$$

In CLS, the coverage constraints are maintained during the search; as a result, these constraints are never violated.

Resource capacity - this constraint ensures that the demand must not exceed the capacity of an agent, which is defined as:

$$\sum_{j \in J} a_{ij} x_{ij} \leq b_i; \qquad \forall i \in I \tag{6.3}$$

6.2.2.3 Violation strategy

For GAP, the violation of resource capacity constraints is used to evaluate local moves. Any amount of resource required in an agent exceeding its capacity is penalised with the same violation h_c. The violation penalty for resource capacity constraint h_c is defined as:

$$\sum_{j \in J} a_{ij} x_{ij} \begin{cases} \leq b_i, \text{violation} = 0 \\ \\ > b_i, \text{violation} = h_c \end{cases}, \forall i \in I \tag{6.4}$$

The objective of the problem is transformed into the soft constraint (6.1) which will never be satisfied. The soft violation for the maximum total profit constraint is defined as:

$$\sum_{i \in I} \sum_{j \in J} p_{ij} x_{ij} \begin{cases} \geq \theta, \text{violation} = 0 \\ \\ < \theta, \text{violation} = h_s \end{cases} \tag{6.5}$$

where: h_s is a soft violation penalty for maximum total profit constraint, $h_s = \theta - \sum_{i \in I} \sum_{j \in J} p_{ij} x_{ij}$, and θ is an upper bound on total profit.

When the search visits the infeasible region (i.e. h_c >0), we evaluate the solutions by the total constraint violation, which can be written as:

$$h = H_s + \left(H_c \times \alpha\right) \tag{6.6}$$

where: h is the total constraint violation, H_s is the violation for the total profit constraint, H_c is the violation for the capacity constraints, and α is a feasibility parameter.

The parameter α represents the violation penalty of using one unit of overloaded agent capacity. The parameter can be adjusted empirically in order to balance the trade-off between hard and soft violations, i.e. as α is increased the search algorithm treats hard constraints as more important relative to the soft constraint.

6.3 Constraint-based local search

The constraint-based local search algorithm (CLS) was originally designed to solve the container rail scheduling problem (Chapter 4). As conceptually simple and convenient, CLS could be adapted and applied to GAP.

CLS starts with an initial random assignment, in which the capacity constraint for each agent can be violated. (Note that CLS maintains coverage constraints (6.2) during the search; therefore these constraints are never violated). In the iteration loop, the algorithm randomly selects a violated constraint, i.e. an assigned agent for which the demand exceeds its resource capacity. Having selected a violated constraint, the algorithm randomly selects two variables in that constraint. For GAP, we categorise the flip trial procedure into two cases:

- Case 1: current value of $x_{ij} = 0$ (agent i does not process job j)
- Case 2: current value of $x_{ij} = 1$ (agent i processes job j)

For case 1, when either the first or second selected variable holds a current value of 0, the algorithm changes the current value of the variable to its complementary binary value, i.e. flipping x_{ij} to 1. At this point, the agent previously processing that job is released in order to satisfy the coverage constraint (6.2). For case 2, when the selected variable has a current value of 1, there are a number of potential agents available to process job j (there are many alternative agents k for flipping $x_{k\neq i,j}$ to 1). In this case, for each selected variable, two flip sub-trials are performed for two randomly selected agents $k \neq i$ to process job j. The sub-trial with the smaller total hard violation is chosen for comparison with the other flip trial.

Between the two flip trials corresponding to the two selected variables, the algorithm selects the alternative with the smaller total hard violation. This alternative becomes the new current solution. Whenever the hard constraint violation H_c is zero, a feasible solution is found. The algorithm stops when no improvement to the best feasible solution found has been achieved for a specified number of iterations (the procedure of CLS is described in Chapter 4 in detail).

6.3.1 Initial experiment

A standard set of GAP problem instances were used to test the CLS. These problems are maximisation problems $S1$, publicly available in Beasley's OR-library (http://mscmga.ms.ic.ac.uk /jeb/orlib/gapinfo.html). These problems were used to test the set partitioning heuristic of Cattrysse et al. (1994) as well as the hybrid simulated annealing/ tabu search heuristic, and tabu search alone, proposed by Osman (1995). The test problems have the following characteristics:

1. The number of agents m is set to 5, 8 and 10.
2. The ratio $r = m/n$ is set to 3, 4, 5 and 6 to determine the number of jobs n.
3. a_{ij} values are integers generated from a uniform distribution $U(5,25)$.
4. p_{ij} values are integers generated from a uniform distribution $U(15,25)$.
5. The b_i values are set to $\left((0.8/m)\times\sum_{j\in J} a_{ij}\right)$.

The test problems $S1$ are divided into 12 groups, gap1 to gap12, according to the size of the test problems. Each group contains five problems.

The experiments were performed on a PC-Pentium 2.4 GHz. The algorithm is coded in Visual Basic 6. For each of the test problems 10 runs are performed using different random number seeds at the beginning of each run. The feasibility parameter α is set manually. For all test problems, we initially performed a few runs in order to determine good values of the parameter. Conceptually, we tried to set the parameter relatively low so that the search targets more on the quality of solutions than on feasibility. If no feasible solution was found, we then increased the value of the parameter. In addition, after the value of feasibility parameter has been chosen, we then want to find a good value of the stopping criterion parameter. The concept is that the algorithm should not spend more computational time than necessary; in contrast, more computational time should be given if the solution could be improved further. We varied this stopping criterion parameter from 500 to 3000 iterations for each test problem, and observed that the parameter value = 1000 iterations is a reasonable setting as in many cases, solutions do not tend to be improved after this limit. Note that this stopping criterion parameter will also be tested and discussed in more detail in Section 6.5.

Table 6.1 shows the results obtained from CLS for the test problems. The results are shown in terms of the average percentage deviation from optimal solution values and average computational time for each problem. The average percentage deviation is defined as:

$$\sigma = \sum_{i=1}^{N} \frac{S^* - S_i}{N \times S^*} \times 100 \tag{6.7}$$

where: σ is the average percentage deviation from S^*, S^* is the optimal solution value, S_i is the solution value of the i-th run, and N is the number of runs.

Problem	m	n	Optimal	Best	σ	Avg. time
gap1-1	5	15	336	316	6.01	0.4
gap1-2	5	15	327	307	6.18	0.3
gap1-3	5	15	339	323	4.79	0.7
gap1-4	5	15	341	325	4.74	0.6
gap1-5	5	15	326	304	6.85	0.4
gap2-1	5	20	434	418	3.71	0.6
gap2-2	5	20	436	416	4.61	1.4
gap2-3	5	20	420	409	2.66	2.0
gap2-4	5	20	419	393	6.24	1.1
gap2-5	5	20	428	402	6.14	1.4
gap3-1	5	25	580	552	4.84	0.7
gap3-2	5	25	564	541	4.12	1.1
gap3-3	5	25	573	554	3.33	0.6
gap3-4	5	25	570	553	3.01	1.3
gap3-5	5	25	564	547	3.04	1.7
gap4-1	5	30	656	634	3.36	4.0
gap4-2	5	30	644	629	2.36	4.3
gap4-3	5	30	673	638	5.28	4.2
gap4-4	5	30	647	626	3.28	6.3
gap4-5	5	30	664	638	3.94	8.3
gap5-1	8	24	563	545	3.25	5.0
gap5-2	8	24	558	539	3.46	9.1
gap5-3	8	24	564	537	4.80	11.4
gap5-4	8	24	568	529	6.89	5.7
gap5-5	8	24	559	524	6.30	5.7
gap6-1	8	32	761	735	3.47	13.2
gap6-2	8	32	759	727	4.23	11.1
gap6-3	8	32	758	724	4.53	8.9
gap6-4	8	32	752	720	4.32	9.0
gap6-5	8	32	747	712	4.76	10.8
gap7-1	8	40	942	895	5.06	11.9
gap7-2	8	40	949	926	2.45	15.4
gap7-3	8	40	968	922	4.80	15.1
gap7-4	8	40	945	902	4.60	13.0
gap7-5	8	40	951	907	4.67	12.9
gap8-1	8	48	1133	1095	3.37	23.2
gap8-2	8	48	1134	1093	3.64	21.7
gap8-3	8	48	1141	1105	3.16	18.1
gap8-4	8	48	1117	1078	3.51	19.9
gap8-5	8	48	1127	1097	2.69	22.1

Problem	m	n	Optimal	Best	σ	Avg. time
gap9-1	10	30	709	688	2.99	37.2
gap9-2	10	30	717	696	2.96	42.4
gap9-3	10	30	712	684	3.96	37.2
gap9-4	10	30	123	114	7.39	39.1
gap9-5	10	30	706	677	4.15	48.9
gap10-1	10	40	958	911	4.98	54.2
gap10-2	10	40	963	921	3.58	49.0
gap10-3	10	40	960	935	2.62	67.9
gap10-4	10	40	947	913	3.60	58.1
gap10-5	10	40	947	911	2.47	57.8
gap11-1	10	50	1139	1093	4.10	75.7
gap11-2	10	50	1178	1139	3.34	66.1
gap11-3	10	50	1195	1144	4.29	70.1
gap11-4	10	50	1171	1138	2.83	86.6
gap11-5	10	50	1171	1123	4.14	63.5
gap12-1	10	60	1451	1403	3.32	92.1
gap12-2	10	60	1449	1316	9.32	94.0
gap12-3	10	60	1433	1386	3.29	78.3
gap12-4	10	60	1447	1405	2.57	81.1
gap12-5	10	60	1446	1390	3.90	85.6

Table 6.1: Results – CLS alone

Table 6.1 shows that CLS can find feasible solutions for all problems. This indicates that CLS is a general solving method as it can readily apply to GAP without any modifications. However, solutions obtained by CLS can be far from the optimal solution and special features to enhance the performance of CLS are necessary.

6.3.2 Variable selection scheme

In CLS, once a violated constraint has been chosen, the algorithm randomly selects two variables in that constraint in order to perform trial flips. For GAP there is only one set of

binary decision variables, i.e. a variable x_{ij} represents whether job j is processed by agent i or not. By randomly choosing variables from the violated constraint, diversified exploration of the search space may not be achieved because the number of assigned agents to jobs ($x_{ij} = 1$) is significantly less than the number of unassigned agents to jobs ($x_{ij} = 0$). Therefore for GAP, two alternative variable selection schemes are introduced as follows:

- Scheme 1: Randomly select two assigned jobs ($x_{ij} = 1$) in the violated constraint.
- Scheme 2: Randomly select any two jobs in the violated constraint ($x_{ij} = 0$) or ($x_{ij} = 1$).

The algorithm selects one of the schemes at random so that a wide exploration of the search space may be achieved. To test the effect of the two-alternative variable selection schemes, we carried out computational experiments with the same data set and parameters as used in the initial experiment (Section 6.3.1). The results are shown in Table 6.2.

Problem	m	n	Optimal	Best	σ	Avg. time
gap1-1	5	15	336	324	3.59	0.3
gap1-2	5	15	327	321	1.85	0.3
gap1-3	5	15	339	333	1.78	0.4
gap1-4	5	15	341	333	2.38	0.5
gap1-5	5	15	326	318	2.49	0.3
gap2-1	5	20	434	428	1.40	0.6
gap2-2	5	20	436	423	3.00	0.9
gap2-3	5	20	420	414	1.44	1.4
gap2-4	5	20	419	413	1.45	0.8
gap2-5	5	20	428	421	1.64	1.0
gap3-1	5	25	580	563	2.96	0.5
gap3-2	5	25	564	552	2.13	0.7
gap3-3	5	25	573	566	1.24	0.5
gap3-4	5	25	570	562	1.41	0.9
gap3-5	5	25	564	551	2.34	1.2
gap4-1	5	30	656	649	1.08	3.0
gap4-2	5	30	644	636	1.25	3.7
gap4-3	5	30	673	654	2.85	3.1
gap4-4	5	30	647	640	1.09	4.8
gap4-5	5	30	664	649	2.28	5.9
gap5-1	8	24	563	552	1.96	3.7
gap5-2	8	24	558	550	1.44	6.5
gap5-3	8	24	564	548	2.88	7.1
gap5-4	8	24	568	558	1.78	4.5
gap5-5	8	24	559	547	2.18	4.0
gap6-1	8	32	761	743	2.39	9.4
gap6-2	8	32	759	746	1.73	7.6
gap6-3	8	32	758	745	1.73	6.7
gap6-4	8	32	752	733	2.53	6.4
gap6-5	8	32	747	729	2.42	7.1
gap7-1	8	40	942	928	1.51	12.1
gap7-2	8	40	949	932	1.38	11.8
gap7-3	8	40	968	950	1.87	10.5
gap7-4	8	40	945	935	1.06	12.3
gap7-5	8	40	951	940	1.16	10.5
gap8-1	8	48	1133	1107	2.32	16.0
gap8-2	8	48	1134	1106	2.48	16.9
gap8-3	8	48	1141	1113	2.49	14.6
gap8-4	8	48	1117	1087	2.71	16.6
gap8-5	8	48	1127	1105	1.96	15.8

Problem	m	n	Optimal	Best	σ	Avg. time
gap9-1	10	30	709	691	2.55	32.8
gap9-2	10	30	717	704	1.83	32.3
gap9-3	10	30	712	696	2.27	24.2
gap9-4	10	30	123	121	1.64	28.4
gap9-5	10	30	706	689	2.01	30.4
gap10-1	10	40	958	933	2.63	45.0
gap10-2	10	40	963	930	1.90	32.6
gap10-3	10	40	960	942	1.46	49.0
gap10-4	10	40	947	929	1.93	45.3
gap10-5	10	40	947	925	1.48	40.0
gap11-1	10	50	1139	1109	2.67	56.3
gap11-2	10	50	1178	1162	1.37	44.3
gap11-3	10	50	1195	1162	2.80	66.3
gap11-4	10	50	1171	1158	1.12	62.5
gap11-5	10	50	1171	1135	3.08	45.6
gap12-1	10	60	1451	1407	3.08	69.8
gap12-2	10	60	1449	1429	1.39	76.3
gap12-3	10	60	1433	1399	2.39	62.0
gap12-4	10	60	1447	1417	2.09	71.7
gap12-5	10	60	1446	1405	2.86	69.0

Table 6.2: Results – CLS with two-alternative variable selection scheme

Table 6.2 shows that CLS with the two-alternative variable selection schemes performs better than without it. We obtain better results in terms of average percentage deviation from optimal solution for all test problems and, in almost all cases, in terms of average computational time. In addition, we observe that the average computational time tends to decrease quite significantly for the larger-sized problems and when the number of jobs increases.

6.3.3 Refined improvement

From previous experiments, although CLS can find feasible solutions for all problems, it lacks a feature that enables a move from a current feasible solution to a better feasible solution. This decreases the performance of CLS for GAP and most likely for other optimisation problems in which there are many feasible solutions. Therefore, a refined-improvement procedure is introduced to improve a feasible solution found by CLS. The main concept is to allow only feasible improving moves and to search more intensively on a feasible solution. When CLS finds a feasible solution, the refined-improvement procedure is called. The procedure of the refined improvement is outlined as follows:

Step 1 Record the total profit for a feasible solution A.

For each job:

Step 2 Order the agents for this job in decreasing order of profit value p_{ij} for this job.

For each agent not processing this job:

Step 3 Swap this job with a job currently processed by this agent. This is illustrated in Figure 6.1 below:

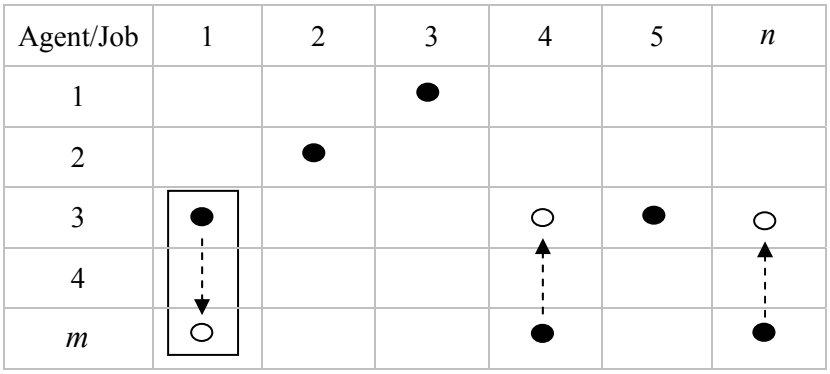

Figure 6.1: Interchange assignment

Step 3 If the new assignment is feasible and increases the total profit, replace A, total profit. Repeat step 2 for next job, if any exist. If the new assignment is not feasible or feasible but does not increase total profit, repeat step 3 with the next job, if any exist processed by this agent.

At first glance, the refined improvement procedure seems to be computationally costly. However, the refined improvement does not perform a complete assignment to all decision variables, i.e. only the few variables associated with the interchange assignment are considered for checking feasibility and solution quality. This reduces the computational effort spent by the refined improvement process significantly.

To test the performance of CLS with the refined improvement procedure, we performed the experiments with the same data set and parameters as used in the previous experiments. The results are shown in Table 6.3.

Problem	*m*	*n*	Optimal	Best	σ	Avg. time
gap1-1	5	15	336	336	0.12	0.3
gap1-2	5	15	327	326	0.18	0.2
gap1-3	5	15	339	337	0.12	0.5
gap1-4	5	15	341	336	0.10	0.5
gap1-5	5	15	326	326	0.12	0.3
gap2-1	5	20	434	431	0.14	0.6
gap2-2	5	20	436	433	0.23	0.9
gap2-3	5	20	420	417	0.24	1.5
gap2-4	5	20	419	417	0.19	0.8
gap2-5	5	20	428	425	0.14	1.1
gap3-1	5	25	580	573	0.30	0.5
gap3-2	5	25	564	561	0.13	0.9
gap3-3	5	25	573	565	0.35	0.5
gap3-4	5	25	570	565	0.22	1.1
gap3-5	5	25	564	556	0.36	1.5
gap4-1	5	30	656	654	0.16	3.4
gap4-2	5	30	644	642	0.36	4.3
gap4-3	5	30	673	666	0.26	3.5
gap4-4	5	30	647	645	0.28	6.5
gap4-5	5	30	664	662	0.34	6.4
gap5-1	8	24	563	555	0.36	3.3
gap5-2	8	24	558	552	0.27	7.2
gap5-3	8	24	564	556	0.36	9
gap5-4	8	24	568	565	0.13	4.6
gap5-5	8	24	559	556	0.14	4.4
gap6-1	8	32	761	754	0.23	12.9
gap6-2	8	32	759	748	0.37	7.4
gap6-3	8	32	758	751	0.23	7.3
gap6-4	8	32	752	746	0.20	7.1
gap6-5	8	32	747	739	0.27	9.5
gap7-1	8	40	942	937	0.27	12.6
gap7-2	8	40	949	945	0.34	12.4
gap7-3	8	40	968	960	0.21	12.6
gap7-4	8	40	945	933	0.32	10.3
gap7-5	8	40	951	942	0.24	9.7
gap8-1	8	48	1133	1128	0.21	25.7
gap8-2	8	48	1134	1125	0.35	19.8
gap8-3	8	48	1141	1135	0.18	20.6
gap8-4	8	48	1117	1110	0.39	22.1
gap8-5	8	48	1127	1118	0.30	18.4

Problem	m	n	Optimal	Best	σ	Avg. time
gap9-1	10	30	709	702	0.47	45.6
gap9-2	10	30	717	715	0.43	39.6
gap9-3	10	30	712	705	0.88	33.2
gap9-4	10	30	123	122	0.37	30.4
gap9-5	10	30	706	703	0.73	37.7
gap10-1	10	40	958	950	0.38	57.6
gap10-2	10	40	963	955	0.69	40
gap10-3	10	40	960	951	0.35	66.5
gap10-4	10	40	947	940	0.47	56.6
gap10-5	10	40	947	939	0.50	62.8
gap11-1	10	50	1139	1128	0.47	80.1
gap11-2	10	50	1178	1170	0.57	74.6
gap11-3	10	50	1195	1183	0.68	68.6
gap11-4	10	50	1171	1168	0.58	82.6
gap11-5	10	50	1171	1165	0.45	56.7
gap12-1	10	60	1451	1441	0.68	91.6
gap12-2	10	60	1449	1435	0.51	110.8
gap12-3	10	60	1433	1426	0.47	92.7
gap12-4	10	60	1447	1432	0.40	92.3
gap12-5	10	60	1446	1438	0.51	84.3

Table 6.3: Results – CLS with refined improvement

Table 6.3 shows that the quality of solutions is improved by the refined improvement procedure at the expense of some increase in computation time. For small-sized problems, in many cases CLS finds optimal solutions or relatively close to optimal. The average percentage deviation from the best solution is decreased substantially compared to those in Tables 6.1 and 6.2. This may be because the solutions are improved by the deterministic process of the refined improvement procedure and the search intensively investigates feasible regions recently found to be good. In contrast, in CLS without the refined improvement procedure, finding good feasible solutions depends on the randomising strategy.

6.3.4 Candidate agent list

The candidate agent list is another feature for CLS to solve GAP. From the problem data sets, we observe that resource consumed by a job for some agents may have a very high value. In this case, when that demand is processed, it is very likely to exceed the capacity of an agent. In contrast, a job processed by some agents may have a very low profit compared with the other agents for that job. In this case assigning that agent to the job is very unlikely to lead to the maximum total profit. We aim to exploit this information to get rid of the unpromising agents at the beginning of the search. This helps increase the efficiency of CLS in two ways: 1) CLS starts with a good initial solution and 2) it limits the size of the search space within promising regions.

Since the trade-off between getting high total profit and using low demand is a major problem, to obtain a good candidate agent list we first calculate a relative demand index and a relative profit index for all assignments of job j to agent i. Then these two indices are combined into a demand-profit index. To make use of this strategy, a candidate agent list for all jobs is sorted in descending order of the demand-profit index. This index is used to restrict the number of potential agents for each job and guide the search in choosing good moves.

A relative demand index represents an estimated chance of assigning the demand of job j to agent i in an optimal solution. The index is weighted both by the agent capacity and by the other agents' demand in each job. A large demand consumed in each job tends to violate the agent capacity; therefore it is unlikely to be selected by the algorithm. To obtain a relative

demand index, the mean and standard deviation of the demand are used. The relative demand index U_{ij} is calculated by the following steps:

To simplify our discussion, let u_{ij} be the proportion of agent capacity i to demand j, i.e. $u_{ij} = b_i / a_{ij}$; μ_{uj} and σ_{uj} be the mean and standard deviation of u_{ij} over all agents; L_{uj} be a variation limit for u_{ij}, i.e. $L_{uj} = \mu_{uj} - \sigma_{uj}$.

Step 1 If u_{ij} is greater than L_{uj}, $\Delta u_{ij} = u_{ij} - L_{uj}$, otherwise $\Delta u_{ij} = 0$.

Step 2 Calculate the relative demand index $U_{ij} = \Delta u_{ij} / \sum_i \Delta u_{ij}$ for all i, j

The variation limit L_{uj} handles the demand variation of agents in each job. u_{ij} greater than this variation limit means that the demand placed on agent i by job j is small, and the assignment tends to be chosen by the algorithm. The smaller u_{ij} is the less likely it is that agent i would process job j. The difference between u_{ij} and the variation limit (i.e. $\Delta u_{ij} = u_{ij} - L_{uj}$) is used to measure how much u_{ij} varies from the variation limit. Suppose that a demand required to process job j by agent i, a_{ij} is given in Table 6.4.

a_{ij}	1	2	3	4	5	6	7	b_i
1	14	23	8	16	8	25	25	36
2	23	22	11	11	12	10	7	34
3	6	22	24	10	24	9	11	38
4	8	14	9	5	6	19	6	27
5	13	13	10	20	25	16	10	33

Table 6.4: An example of the demand matrix

Table 6.5 gives the corresponding values of the relative demand index U_{ij} obtained by the above procedure.

U_{ij}	1	2	3	4	5	6	7
1	0.13	0.06	0.43	0.06	0.40	0.03	0.00
2	0.01	0.05	0.18	0.18	0.18	0.34	0.36
3	0.53	0.13	0.00	0.27	0.02	0.47	0.17
4	0.21	0.23	0.17	0.49	0.40	0.03	0.31
5	0.12	0.53	0.22	0.00	0.00	0.13	0.15

Table 6.5: The relative demand index

A relative profit index represents an estimated possibility of the profit of job j processed by agent i contributing to the maximum total profit. The index is weighted both by the maximum profit for agent i and by the other agents' associated profit for that job. The mean

and standard deviation of profit are used to handle profit variation. The relative profit index V_{ij} can be obtained similar to the relative demand index:

Let v_{ij} be the proportion of profit i to the maximum profit for agent i, $v_{ij} = p_{ij} / \max\{p_{ij} : j = 1,2,3,...,n\}$; μ_{vj} and σ_{vj} be the mean and standard deviation of v_{ij} over all agents; L_{vj} be a variation limit for v_{ij}, i.e. $L_{vj} = \mu_{vj} - \sigma_{vj}$.

Step 1 If v_{ij} is greater than L_{vj}, $\Delta v_{ij} = v_{ij} - L_{vj}$, otherwise $\Delta v_{ij} = 0$.

Step 2 Calculate the relative profit index $V_{ij} = \Delta v_{ij} / \sum_i \Delta v_{ij}$ for all i, j.

Suppose that the profit of job j processed by agent i, p_{ij} is given in Table 6.6.

p_{ij}	1	2	3	4	5	6	7	$p_{i(\max)}$
1	22	18	24	15	20	18	16	24
2	21	16	17	16	19	25	17	25
3	16	25	24	16	17	19	20	25
4	22	22	20	16	19	17	25	25
5	15	15	21	25	16	16	22	25

Table 6.6: An example of the profit matrix

Table 6.7 shows the relative profit index is obtained from the above procedure.

V_{ij}	1	2	3	4	5	6	7
1	0.38	0.17	0.40	0.10	0.44	0.18	0.00
2	0.28	0.04	0.00	0.12	0.25	0.53	0.02
3	0.01	0.47	0.37	0.12	0.05	0.19	0.19
4	0.33	0.32	0.10	0.12	0.25	0.08	0.48
5	0.00	0.00	0.16	0.56	0.00	0.02	0.31

Table 6.7: The relative profit index

Now, the demand-profit index W_{ij} can be obtained, i.e. $W_{ij} = \left(U_{ij} + V_{ij}\right)/2$. A high value of W_{ij} indicates a high estimated chance of assigning job j to agent i with maximum feasible total profit. The demand-profit index is shown in Table 6.8.

W_{ij}	1	2	3	4	5	6	7
1	0.25	0.11	0.41	0.08	0.42	0.10	0.00
2	0.14	0.04	0.09	0.15	0.22	0.44	0.19
3	0.27	0.30	0.17	0.19	0.04	0.33	0.18
4	0.27	0.28	0.14	0.30	0.33	0.05	0.40
5	0.06	0.27	0.19	0.28	0.00	0.07	0.23

Table 6.8: The demand-profit index

Then, the candidate agent list for each job is sorted in descending order of the demand-profit index W_{ij}, which is shown in Table 6.9.

Candidate list	1	2	3	4	5	6	7
1	3	3	1	4	1	2	4
2	4	4	5	5	4	3	5
3	1	5	3	3	2	1	2
4	2	1	4	2	3	5	3
5	5	2	2	1	5	4	1

Table 6.9: The candidate agent list for each job

The candidate list parameter C is introduced to let us limit the number of candidate agents empirically at the beginning of the search and the algorithm starts with an initial random assignment within the number of candidate agents specified by the parameter C. Then the search algorithm only evaluates a job assigned to the limited number of promising agents. This helps reduce the computational effort significantly and targets the search for an optimal solution. For example, in Table 6.9, setting C = 3, agent 3, 4, and 1 are only considered for job 1.

For GAP, it is the first time that the variations of demand and profit are considered, and the trade-off between them is handled by the relative demand-profit index. The design of this technique is similar to that of timeslot violation discussed in Section 4.3.2. The use of the candidate agent list is fruitful because practitioners with little computer background can easily modify the relative demand-cost index in order to improve the performance of the solving algorithm. To test the performance of CLS with the candidate agent list, we performed the experiments with the same data set and parameters as used in the previous experiments. The candidate list parameter C is set manually. For all test problems, we

initially performed a few runs in order to determine good values of the parameter. Firstly, the parameter was roughly set to 50-60% of the number of agents for each test problem. We then slightly increased the value of the parameter whether it resulted in any improvement on the quality of solutions. The results are shown in Table 6.10.

Problem	*m*	*n*	Optimal	Best	σ	Avg. time
gap1-1	5	15	336	336	0.10	0.1
gap1-2	5	15	327	327	0.16	0.2
gap1-3	5	15	339	336	0.13	0.3
gap1-4	5	15	341	340	0.11	0.2
gap1-5	5	15	326	326	0.10	0.2
gap2-1	5	20	434	433	0.15	0.3
gap2-2	5	20	436	431	0.26	0.8
gap2-3	5	20	420	419	0.25	1.1
gap2-4	5	20	419	419	0.17	0.5
gap2-5	5	20	428	428	0.12	0.7
gap3-1	5	25	580	578	0.34	0.4
gap3-2	5	25	564	563	0.15	0.7
gap3-3	5	25	573	570	0.39	0.4
gap3-4	5	25	570	568	0.24	0.5
gap3-5	5	25	564	564	0.37	1
gap4-1	5	30	656	651	0.17	2.7
gap4-2	5	30	644	640	0.39	1.8
gap4-3	5	30	673	673	0.29	2.7
gap4-4	5	30	647	642	0.31	2.8
gap4-5	5	30	664	658	0.38	3.5
gap5-1	8	24	563	563	0.36	2.8
gap5-2	8	24	558	557	0.28	4.2
gap5-3	8	24	564	561	0.40	4.3
gap5-4	8	24	568	568	0.13	2.7
gap5-5	8	24	559	558	0.15	3.4
gap6-1	8	32	761	757	0.23	7.3
gap6-2	8	32	759	758	0.38	5.5
gap6-3	8	32	758	757	0.23	5.4
gap6-4	8	32	752	751	0.21	6.1
gap6-5	8	32	747	746	0.26	6
gap7-1	8	40	942	935	0.29	6.7
gap7-2	8	40	949	945	0.33	6.2
gap7-3	8	40	968	967	0.20	7.4
gap7-4	8	40	945	944	0.33	7.5
gap7-5	8	40	951	951	0.21	7.1
gap8-1	8	48	1133	1126	0.25	15.4
gap8-2	8	48	1134	1122	0.42	13.3
gap8-3	8	48	1141	1135	0.21	13.9
gap8-4	8	48	1117	1104	0.47	13.4
gap8-5	8	48	1127	1117	0.36	13.8

Problem	m	n	Optimal	Best	σ	Avg. time
gap9-1	10	30	709	699	0.57	29.1
gap9-2	10	30	717	712	0.52	28.5
gap9-3	10	30	712	706	1.06	23.4
gap9-4	10	30	123	122	0.44	18.6
gap9-5	10	30	706	690	0.88	30.7
gap10-1	10	40	958	942	0.46	28.3
gap10-2	10	40	963	949	0.83	26.2
gap10-3	10	40	960	950	0.42	45.3
gap10-4	10	40	947	932	0.56	37.1
gap10-5	10	40	947	925	0.60	31
gap11-1	10	50	1139	1113	0.56	39.1
gap11-2	10	50	1178	1172	0.68	36.5
gap11-3	10	50	1195	1178	0.82	55.8
gap11-4	10	50	1171	1164	0.70	42.2
gap11-5	10	50	1171	1158	0.54	33.4
gap12-1	10	60	1451	1430	0.82	66.4
gap12-2	10	60	1449	1438	0.61	65.7
gap12-3	10	60	1433	1413	0.57	61.6
gap12-4	10	60	1447	1425	0.48	64.4
gap12-5	10	60	1446	1424	0.61	50.6

Table 6.10: Results – CLS with candidate agent list

From Table 6.10, we observe that CLS with candidate agent list performs very well. For small-sized problems, in many cases, it finds optimal solutions or very near optimal solutions within little computational time. However, the candidate agent list strategy becomes less powerful when the number of agents and jobs increases. This may be because the trade-off between total profit and infeasibility becomes less clear and the search space cannot further be limited and intensified using the candidate agent list strategy.

Note that from our experiments, all three extensions (two-variables selection, refined improvement, and candidate agent list) improve on the results of CLS alone and do not

significantly increase run time. This indicates the usefulness of using CLS with all these extensions.

6.4 Search intensification technique

We adapt the predictive choice model in discussed chapter 5 for GAP. The model learns from the search history and extracts problem specific knowledge automatically. After a specified number of iterations, the search history is analysed. The model predicts good assignments of jobs to agents. These assignments will be fixed for a number of iterations determined in a probabilistic way, leading to intensified exploration of the search space.

6.4.1 Violation history

In CLS, after choosing a violated constraint, a two-variable selection strategy is used in each flip trial. The first variable is randomly chosen from those appearing in a violated constraint (i.e. an overloaded agent) as the variable of interest, the second variable is randomly selected from that violated constraint, and provides a basis for comparison with the variable of interest. The procedure of the collection of violation history is described in Section 5.3.1.1.

6.4.2 Variable fixing

After a specified number of iterations, the trial history is analysed. Some variables may have high probability of a particular value given by the predictive choice model. These variables will be fixed at their predicted value for a number of iterations determined by the magnitude of the probability. The search space would be intensified and the algorithm targets an

optimal solution. The decision variable x_{ij} may hold a current value 0 or 1 and its predicted value can either be 0 or 1 during the search, which we categorise into two groups: local fix and global fix respectively.

Local fix. The local fix is a process of preventing the algorithm assigning a job to an agent that may not lead to an optimal solution, i.e. fixing x_{ij} = 0 for the number of iterations. Although locally fixing x_{ij} at 0 is not very effective as in a complete assignment there are many unassigned agents to jobs (i.e. x_{ij} that hold a current value 0), the number of potential agents to process a job is reduced quickly.

Global fix. The global fix provides a strong propagation of consistency within each job for the potential number of agents. When the global fix is called (i.e. a predicted value of x_{ij} = 1), exactly one agent processes the job, thereby preventing the algorithm selecting the remaining potential agents for that job.

6.5 Computational experiments

Apart from the first set of test problems $S1$, we test a second set of problems $S2$, which contains 24 large-sized minimisation problems. These problems were used to test the GA proposed by Chu and Beasley (1997), the variable depth search proposed by Yagiura et al (1999a), and tabu search with ejection chain proposed by Yagiura et al (1999b). The problems in $S2$ are divided into four classes according to the way they were generated.

1. Type A. a_{ij} are integers generated from a uniform distribution $U(5,25)$, c_{ij} are integers generated from a uniform distribution $U(10,50)$, and $b_i = 0.6 \times (n/m) \times 15 + 0.4R$, where $R = \max_i \sum_{j \in J, I_j = i} a_{ij}$, and $I_j = \min\{i \mid c_{ij} \le c_{kj}, \forall k \in I\}$
2. Type B. a_{ij} and c_{ij} generated as in Type A and b_i is set to 70% of the value given for Type A.
3. Type C. a_{ij} and c_{ij} generated as in Type A and $b_i = 0.8\sum_{j \in J} a_{ij} / m$.
4. Type D. a_{ij} are integers generated from a uniform distribution $U(1,100)$, $c_{ij} = 111 - a_{ij} + e$, where e are integers generated from a uniform distribution $U(-10,10)$ and $b_i = 0.8\sum_{j \in J} a_{ij} / m$.

Types B and C problems are more difficult than Type A because the resource capacity constraints are tighter. Type D problems are most difficult to solve because a_{ij} and c_{ij} are inversely correlated.

Computational experiments, using CLS incorporating the predictive choice model with two-alternative selection scheme, refined improvement procedure and candidate agent list are performed. For each of the test problems 10 runs are performed using different random number seeds at the beginning of each run. For problems in $S1$, we use the same data set and parameters as used in the previous experiments. The stopping criterion is set to 3000 iterations for problems $S2$. For the predictive choice model, we set the number of flip trials N = 10, decision parameter D = 75, the number of fixing iterations F ranges from 50 to

100, the number of fixing unassigned agents in each job (x_{ij} = 0) ranges from 1 to 3, the number of fixing assigned agent (x_{ij} = 1) ranges from 5 to 30. Note that the concepts of how to set good values for these parameters and their sensitivity tests are given in Chapter 5. We do not claim here that these are the best parameter values, but some care was taken. The results for test problems $S1$ are shown in Table 6.11 – 6.12.

Problem	m	n	Optimal	Best	σ	Avg. Time (s)
gap1-1	5	15	336	336	0.00	0.1
gap1-2	5	15	327	327	0.00	0.1
gap1-3	5	15	339	339	0.00	0.2
gap1-4	5	15	341	341	0.00	0.2
gap1-5	5	15	326	326	0.00	0.1
gap2-1	5	20	434	434	0.00	0.2
gap2-2	5	20	436	436	0.00	0.4
gap2-3	5	20	420	420	0.00	0.6
gap2-4	5	20	419	419	0.00	0.3
gap2-5	5	20	428	428	0.00	0.4
gap3-1	5	25	580	580	0.00	0.2
gap3-2	5	25	564	564	0.00	0.3
gap3-3	5	25	573	573	0.00	0.2
gap3-4	5	25	570	570	0.00	0.4
gap3-5	5	25	564	564	0.00	0.5
gap4-1	5	30	656	656	0.02	1.2
gap4-2	5	30	644	644	0.07	1.4
gap4-3	5	30	673	673	0.05	1.2
gap4-4	5	30	647	647	0.08	2.0
gap4-5	5	30	664	664	0.09	2.2
gap5-1	8	24	563	563	0.09	1.5
gap5-2	8	24	558	558	0.14	2.7
gap5-3	8	24	564	564	0.10	3.2
gap5-4	8	24	568	568	0.05	2.0
gap5-5	8	24	559	559	0.18	1.8
gap6-1	8	32	761	761	0.20	4.5
gap6-2	8	32	759	759	0.12	3.2
gap6-3	8	32	758	758	0.19	3.0
gap6-4	8	32	752	752	0.08	3.0
gap6-5	8	32	747	747	0.14	3.4

Problem	m	n	Optimal	Best	σ	Avg. Time (s)
gap7-1	8	40	942	941	0.18	4.1
gap7-2	8	40	949	949	0.22	4.5
gap7-3	8	40	968	968	0.20	4.3
gap7-4	8	40	945	945	0.17	4.0
gap7-5	8	40	951	951	0.25	4.0
gap8-1	8	48	1133	1133	0.12	7.4
gap8-2	8	48	1134	1133	0.20	6.2
gap8-3	8	48	1141	1141	0.15	6.0
gap8-4	8	48	1117	1117	0.18	7.0
gap8-5	8	48	1127	1127	0.22	6.0
gap9-1	10	30	709	709	0.18	13.1
gap9-2	10	30	717	717	0.36	13.0
gap9-3	10	30	712	712	0.80	10.5
gap9-4	10	30	123	123	0.25	11.0
gap9-5	10	30	706	706	0.60	13.5
gap10-1	10	40	958	958	0.20	18.0
gap10-2	10	40	963	961	0.52	15.0
gap10-3	10	40	960	960	0.32	18.5
gap10-4	10	40	947	946	0.30	16.0
gap10-5	10	40	947	947	0.20	18.0
gap11-1	10	50	1139	1139	0.20	23.2
gap11-2	10	50	1178	1178	0.35	21.0
gap11-3	10	50	1195	1193	0.50	23.0
gap11-4	10	50	1171	1171	0.41	24.0
gap11-5	10	50	1171	1169	0.36	19.0
gap12-1	10	60	1451	1449	0.52	30.0
gap12-2	10	60	1449	1449	0.44	32.2
gap12-3	10	60	1433	1431	0.35	27.1
gap12-4	10	60	1447	1447	0.46	28.5
gap12-5	10	60	1446	1445	0.31	28.0

Table 6.11: Results for maximisation problems S_1

From Table 6.11, we observe that CLS performs very well and almost finds the optimal solution for all problems. However, for those problems in which CLS fails to reach the optimal solution, all solutions are very close to optimality. The results also demonstrate that CLS is capable of producing good quality solutions in little time.

Prob. Set	SA/TS	TS	GA	TSH	ASH	CLS
gap1	0.00	0.00	0.00	0.00	-	0.00
gap2	0.00	0.10	0.00	0.00	-	0.00
gap3	0.00	0.00	0.00	0.00	-	0.00
gap4	0.00	0.03	0.00	0.00	-	0.06
gap5	0.00	0.00	0.00	0.00	-	0.11
gap6	0.05	0.03	0.01	0.01	-	0.14
gap7	0.02	0.00	0.00	0.00	0.00	0.20
gap8	0.10	0.09	0.05	0.01	0.04	0.18
gap9	0.08	0.06	0.00	0.00	0.00	0.29
gap10	0.14	0.08	0.04	0.03	0.01	0.31
gap11	0.05	0.02	0.00	0.00	0.00	0.36
gap12	0.11	0.04	0.01	0.00	0.00	0.41

SA/TS: Osman (1995), simulated annealing + tabu search

TS: Osman (1995), tabu search

GA: Chu and Beasley (1996), GA with heuristic operator

TSH: Diaz and Fernandez (2001), tabu search heuristic

ASH: Lourenco and Serra (2002), adaptive search heuristics, ant + descendent local search + tabu with restricted ejection chain

Table 6.12: Average percentage deviation from optimal solution for S_1

Table 6.12 shows the results compared with some existing methods in terms of the average percentage deviation σ from optimal value for each problem. It can be seen that the deviation from optimal values obtained by CLS are as good as or close to the values obtained by the compared methods. In cases when CLS is outperformed by other methods, the gap is small, i.e. σ is less than 1%.

In addition, we note that for CLS the computational times are very good, even though it is implemented using a relatively less powerful programming language (Visual Basic: VB). The principal reason for using VB is that, although it is less powerful than several other

programming languages, e.g. Fortran, C++, VB is still a full language and offers much shorter development times. This was expected to be significant as many algorithmic variants were to be investigated.

Next, we perform experiments on large-sized minimisation problems S_2. We compare our results to the best known solution (ϕ) for these problems. For some problems in this set, the optimal solution values were obtained by a branch-and-bound algorithm proposed by Nauss (2003); these are marked with an asterisk.

Table 6.13 shows the best values (Best), and the average percentage deviation (σ) from the best values obtained from different solution methods. The results show that our method performs well particularly in problem types A and B, in which the solution values are close to the best known solution values. For problem types C and D, CLS is outperformed by the existing methods in general. However, the gap of the solution values is not too high and computational time is very good.

Prob	m	n	ϕ	GA		TSEC		TSH		HGA		CLS		
				Best	σ	Best	σ	Best	σ	Best	*SD*	Best	σ	Time
A	5	100	1698*	1698	0.00	-	-	-	-	-	0.00	1700	0.09	38.2
	5	200	3235*	3235	0.00	-	-	-	-	-	0.00	3237	0.04	63.0
	10	100	1360*	1360	0.00	-	-	-	-	-	0.00	1361	0.22	50.3
	10	200	2623*	2623	0.00	-	-	-	-	-	0.00	2625	0.10	105.0
	20	100	1158*	1158	0.00	-	-	-	-	-	0.00	1160	0.12	94.1
	20	200	2339*	2339	0.00	-	-	-	-	-	0.00	2342	0.15	153.0
B	5	100	1843	1843	0.35	-	-	1843	0.00	-	0.08	1845	0.32	40.7
	5	200	3552	3553	0.30	-	-	3552	0.01	-	0.05	3574	0.20	89.0
	10	100	1407	1407	0.07	-	-	1407	0.00	-	0.00	1410	0.11	37.0
	10	200	2828	2831	0.31	-	-	2828	0.05	-	0.14	2839	0.32	160.8
	20	100	1166	1166	0.07	-	-	1166	0.10	-	0.21	1170	0.41	125.0
	20	200	2340	2340	0.06	-	-	2340	0.11	-	0.08	2345	0.30	246.6
C	5	100	1931*	1931	0.38	1931	0.00	1931	0.00	-	0.18	1936	0.22	53.0
	5	200	3456*	3458	0.23	3456	0.00	3457	0.04	-	0.03	3460	0.35	108.7
	10	100	1402*	1403	0.29	1402	0.00	1402	0.04	-	0.09	1414	0.28	94.0
	10	200	2806*	2814	0.48	2806	0.01	2807	0.11	-	0.16	2815	0.51	171.7
	20	100	1243*	1244	0.52	1243	0.00	1243	0.28	-	0.13	1248	0.40	215.0
	20	200	2391	2397	0.62	2391	0.03	2391	0.12	-	0.18	2405	0.31	280.5
D	5	100	6353*	6373	0.66	6354	0.04	6357	0.16	-	0.06	6465	0.71	75.0
	5	200	12743	12796	0.65	12744	0.02	12747	0.09	-	0.16	12826	0.60	271.5
	10	100	6349	6379	1.24	6356	0.17	6355	0.51	-	0.22	6380	0.82	210.2
	10	200	12436	12601	1.54	12445	0.08	12457	0.28	-	0.18	12541	1.15	630.0
	20	100	6196	6269	1.65	6215	0.39	6220	0.91	-	0.31	6280	0.73	615.4
	20	200	12264	12452	1.98	12277	0.17	12351	0.89	-	0.33	12380	1.54	920.5

GA: Chu and Beasley (1996), GA with heuristic operator

EC: Yagiura et al (2004), tabu search with ejection chain

TSH: Diaz and Fernandez (2001), tabu search heuristic

HGA: Feltl and Raidl (2004), improved hybrid GA, SD is the standard deviation of the optimal value of the LP-relaxation after the CPLEX solver was terminated due to the running time or memory limits.

Table 6.13: Results for minimisation problems S_2

The results from the two data sets indicate that CLS is a promising approach for GAP. We are able to obtain solutions of good quality that are as good as or close to the best known solutions in little computational time. Although the existing solution approaches outperform CLS in terms of the optimal solution values, we believe that the simplicity and computational advantage of our approach is a pay-off for the solving algorithm.

Since CLS takes little computational time even with a programming language, which is relatively slow, we are keen to run it for longer and expect to see better results. We set the stopping criterion parameter to 10000 for problems S_1 and 30000 for problems S_2. For each of the test problems 10 runs are performed using different random number seeds at the beginning of each run and the remaining parameters in the algorithm are fixed. The results are shown in Table 6.14 - 6.15.

Problem	m	n	Optimal	Best	σ	Avg. Time (s)
gap1-1	5	15	336	336	0.00	0.3
gap1-2	5	15	327	327	0.00	0.2
gap1-3	5	15	339	339	0.00	0.4
gap1-4	5	15	341	341	0.00	0.5
gap1-5	5	15	326	326	0.00	0.2
gap2-1	5	20	434	434	0.00	0.4
gap2-2	5	20	436	436	0.00	0.9
gap2-3	5	20	420	420	0.00	0.3
gap2-4	5	20	419	419	0.00	0.8
gap2-5	5	20	428	428	0.00	1.0
gap3-1	5	25	580	580	0.00	0.5
gap3-2	5	25	564	564	0.00	0.8
gap3-3	5	25	573	573	0.00	0.5
gap3-4	5	25	570	570	0.00	1.6
gap3-5	5	25	564	564	0.00	1.4
gap4-1	5	30	656	656	0.00	3.2
gap4-2	5	30	644	644	0.00	3.1
gap4-3	5	30	673	673	0.00	3.3
gap4-4	5	30	647	647	0.00	5.4
gap4-5	5	30	664	664	0.00	5.9
gap5-1	8	24	563	563	0.00	4.7
gap5-2	8	24	558	558	0.02	8.4
gap5-3	8	24	564	564	0.00	9.9
gap5-4	8	24	568	568	0.00	6.2
gap5-5	8	24	559	559	0.00	5.6
gap6-1	8	32	761	761	0.01	13.4
gap6-2	8	32	759	759	0.02	8.6
gap6-3	8	32	758	758	0.00	9.3
gap6-4	8	32	752	752	0.00	9.3
gap6-5	8	32	747	747	0.03	10.5
gap7-1	8	40	942	942	0.00	12.7
gap7-2	8	40	949	949	0.00	12.0
gap7-3	8	40	968	968	0.00	13.3
gap7-4	8	40	945	945	0.00	12.4
gap7-5	8	40	951	951	0.00	12.4
gap8-1	8	48	1133	1133	0.03	22.9
gap8-2	8	48	1134	1134	0.00	17.2
gap8-3	8	48	1141	1141	0.00	18.6
gap8-4	8	48	1117	1117	0.02	17.1
gap8-5	8	48	1127	1127	0.00	19.1

Problem	m	n	Optimal	Best	σ	Avg. Time (s)
gap9-1	10	30	709	709	0.05	51.1
gap9-2	10	30	717	717	0.00	50.7
gap9-3	10	30	712	712	0.02	41.0
gap9-4	10	30	123	123	0.00	42.9
gap9-5	10	30	706	706	0.00	52.7
gap10-1	10	40	958	958	0.00	64.8
gap10-2	10	40	963	963	0.00	58.5
gap10-3	10	40	960	960	0.00	72.2
gap10-4	10	40	947	946	0.05	62.4
gap10-5	10	40	947	947	0.00	70.2
gap11-1	10	50	1139	1139	0.00	89.2
gap11-2	10	50	1178	1178	0.00	94.5
gap11-3	10	50	1195	1194	0.04	103.5
gap11-4	10	50	1171	1171	0.02	108.0
gap11-5	10	50	1171	1171	0.00	85.5
gap12-1	10	60	1451	1451	0.00	129.0
gap12-2	10	60	1449	1449	0.00	132.5
gap12-3	10	60	1433	1431	0.05	122.0
gap12-4	10	60	1447	1447	0.02	118.3
gap12-5	10	60	1446	1446	0.03	126.0

Table 6.14: Results for S_1 - stopping criterion parameter = 10000

Table 6.14 shows that CLS can find optimal solutions for all test problems except two cases, gap11-3 and gap12-3. The results in terms of best solutions found are only marginally better than those in Table 6.11 despite a ten-fold increase in the stopping criterion parameter which naturally carries additional computational cost. However, from Table 6.14, we obtain much better average percentage deviation from optimal values than in Table 6.11. These are as good as or very close to the values obtained by the existing methods.

Prob	m	n	ϕ	GA		TSEC		TSH		HGA		CLS	
				Best	σ	Best	σ	Best	σ	Best	*SD*	Best	σ
A	5	100	1698*	1698	0.00	-	-	-	-	-	0.00	1698	0.02
	5	200	3235*	3235	0.00	-	-	-	-	-	0.00	3236	0.04
	10	100	1360*	1360	0.00	-	-	-	-	-	0.00	1360	0.00
	10	200	2623*	2623	0.00	-	-	-	-	-	0.00	2623	0.00
	20	100	1158*	1158	0.00	-	-	-	-	-	0.00	1159	0.03
	20	200	2339*	2339	0.00	-	-	-	-	-	0.00	2342	0.08
B	5	100	1843	1843	0.35	-	-	1843	0.00	-	0.08	1843	0.12
	5	200	3552	3553	0.30	-	-	3552	0.01	-	0.05	3554	0.07
	10	100	1407	1407	0.07	-	-	1407	0.00	-	0.00	1410	0.05
	10	200	2828	2831	0.31	-	-	2828	0.05	-	0.14	2831	0.08
	20	100	1166	1166	0.07	-	-	1166	0.10	-	0.21	1166	0.11
	20	200	2340	2340	0.06	-	-	2340	0.11	-	0.08	2342	0.15
C	5	100	1931*	1931	0.38	1931	0.00	1931	0.00	-	0.18	1932	0.05
	5	200	3456*	3458	0.23	3456	0.00	3457	0.04	-	0.03	3458	0.13
	10	100	1402*	1403	0.29	1402	0.00	1402	0.04	-	0.09	1407	0.21
	10	200	2806*	2814	0.48	2806	0.01	2807	0.11	-	0.16	2815	0.32
	20	100	1243*	1244	0.52	1243	0.00	1243	0.28	-	0.13	1245	0.22
	20	200	2391	2397	0.62	2391	0.03	2391	0.12	-	0.18	2395	0.29
D	5	100	6353*	6373	0.66	6354	0.04	6357	0.16	-	0.06	6457	0.24
	5	200	12743	12796	0.65	12744	0.02	12747	0.09	-	0.16	12761	0.38
	10	100	6349	6379	1.24	6356	0.17	6355	0.51	-	0.22	6372	0.61
	10	200	12436	12601	1.54	12445	0.08	12457	0.28	-	0.18	12492	0.89
	20	100	6196	6269	1.65	6215	0.39	6220	0.91	-	0.31	6243	0.52
	20	200	12264	12452	1.98	12277	0.17	12351	0.89	-	0.33	12369	0.91

Table 6.15: Results for S_2 - stopping criterion parameter = 30000

Table 6.15 shows that we obtain better results in terms of the best solutions found and the average percentage deviation from optimal values than in Table 6.13. For problem types A, B and C, the results are as good as or close to the results obtained by the existing methods. For problem type D, the best solutions found by CLS (and also GA and TSH) are still relatively far from the best known solutions ϕ, but the average percentage deviation has decreased.

We note that the stopping criterion parameter may not be of critical importance to the performance of our algorithm. Since this parameter mainly serves the purpose of search diversification, when the search is sufficiently diversified within a high-enough number of iterations, solutions may not be improved further by diversification and good intensification is required to improve the solutions from this point.

6.6 Conclusions

The application of constraint-based local search incorporating with the predictive choice model (CLS) to GAP is presented in this Chapter. The performance of the algorithm is evaluated with two different benchmark problem sets, and compared with other existing solution methods. The results obtained with CLS are very promising. In general we obtain high quality solutions that are as good as, or close to, the solutions obtained from the existing methods. CLS employs a random strategy to achieve a diversified exploration of the search space and incorporates a self-learning feature that learns from the search history and implicitly extracts problem knowledge. During the run, the search history is analysed and CLS predicts good assignments of jobs to agents. These assignments will be fixed in a probabilistic manner, leading to intensified exploration of the search space.

It is the first time that CLS has been applied to the solution of GAP. Although CLS cannot outperform the existing methods especially in terms of the solution values, the solutions obtained by CLS are as good as or very close to the solutions from the existing methods. We believe that the simplicity and computational advantage of our method is a pay-off for the solving algorithm. The solutions for GAP tailored by CLS may be improved if CLS is coded in a more powerful programming language e.g. C or Fortran, or using more sophisticated

heuristics to explore complex local moves, i.e. maintaining a promising sorted candidate list during the search or incorporating dynamic bound strategies derived from LP solutions as used in most existing methods for GAP.

Our proposed method may also be promising for other assignment type problems which are computationally more demanding than GAP, such as the quadratic assignment problem (Rardin, 1998), the multilevel generalised assignment problem (Laguna et al, 1995) and the blockmodel problem (Jessop, 2003). This is because the predictive choice model would be able to capture the complex interactions amongst the variables in the model and to predict the movements of the solution in the search space.

Chapter Seven

Conclusions

7.1 Summary

In this research, the container rail scheduling problem has been presented and an optimisation framework for its solution has been proposed. The container rail scheduling problem is modelled as a constraint satisfaction problem in which a demand responsive scheduling service is considered in order to improve the service offered to customers and to reduce operating costs for the rail carrier.

A constraint-based local search algorithm is developed and applied to the container rail scheduling problem. The algorithm uses a simple variable flip as a structure of local move. When all variables in the model are assigned a value, the total hard violation is calculated; a quantified measure of the violation is then used to evaluate local moves. Different measures of the violation are also used to drive the search to the promising regions of the search space.

In addition, the constraint-based local search algorithm incorporates a predictive choice model. The results using real-life data sets show some reductions in total operating costs, and enhance the level of service through demand responsive schedules.

The research also demonstrates the application of the proposed algorithmic approach to the generalised assignment problem. The constraint-based local search algorithm mainly employs a randomised strategy to achieve a diversified exploration of the search space and the predictive choice model predicts good assignment of jobs to agents. The performance of the algorithm has been assessed with benchmark problem sets.

7.2 Achievements of this research

Generating a profitable schedule is crucial to a container rail business because rail's profitability is influenced by its service offerings. Optimisation models to improve its operations and advanced solution techniques are significant. In this research, the major contributions are divided into two areas as follows:

From the application point of view, the demand responsive scheduling model incorporates the following new features that are more complex and not considered in previous work.

1. Non-uniform arrivals with distinct target times, i.e. not all containers are available at the beginning of the scheduling time horizon and must be treated as distinct customer bookings.
2. A demand responsive container rail service providing flexible schedules

3. A probabilistic decrease in customer satisfaction with deviation from target time

From the computational viewpoint, the proposed method contributes to the scheduling research in the following aspects:

1. Local search for constraint satisfaction problems has been investigated. A constraint-based local search algorithm is developed to solve combinatorial optimisation problems. The constraint-based local search plays a key role in diversifying the search. It incorporates the predictive choice model for search intensification. Good interplay between the diversification and intensification strategies is the main feature of the proposed algorithm.

2. A novel learning mechanism, the predictive choice model, has been developed. We propose a theoretical discrete choice learning model and then make it applicable to combinatorial optimisation problems, the container rail scheduling problem and the generalised assignment problem. Although the predictive choice model needs to maintain and analyse the search history and hence imposes a computational cost, this is offset against a lower run-time required to find good solutions. Computational results demonstrate the robustness and usefulness of the proposed technique.

3. The predictive choice model is not dependent on domain-specific knowledge; it does not depend on the nature of the objective function or constraints, whether linear or non-linear. This suggests that the proposed algorithm would be applicable

to other combinatorial optimisation problems in which all variables in the model are binary.

7.3 Future work

The following points provide some of the issues that could be investigated as the future work of this research.

1. With the demand responsive schedule, there might be some customers that book the rail service close to the end of a schedulable week. This may cause the train schedule to be not profitable because the proposed model always insists on satisfying all customer demand within the week. In this case, the automatic scheduling system may be required to re-consolidate the customer shipments and to compare the effect if those customers are dropped and considered in the following week. The modest computational demands of the algorithm described in Chapters 4 and 5 make this approach viable. This needs more investigation on railway practices and survey data.

2. With respect to the algorithm proposed, there are always possible ways to improve the performance of the algorithm. For example, some parameters in the algorithm are determined empirically, e.g. the feasibility parameter α that balances the trade-off between hard and soft violations. They might be tuned effectively by adaptive mechanisms based on statistical methods.

3. The predictive choice model may be extended to multiple value choice decisions, i.e. the predictive model may be based on multinomial discrete choice theory that can predict a good value for an integer variable. However, in general discrete

optimisation problems, the variable domain can be large. Therefore, predicting every single value for a variable would be computationally expensive and is not reasonable. In this case the variable domain may be partitioned into multiple groups, and the predictive choice model used to strengthen the variable domain.

4. The intensification strategy used by the predictive choice model is soundly based on a statistical method; the consistency between variables is enforced in a probabilistic way, leading to intensified exploration of the search space. However, the diversification for the constraint-based local search is achieved by the randomised selection of variables to explore and by not insisting on complete consistency at some stages of the search. Although random selection and variable selection schemes are used, how well the diversification is achieved is kept in mind implicitly. In addition, it has to exploit domain knowledge to construct effective variable selection schemes. Therefore, a variable selection strategy based on some statistical methods may be incorporated. For instance, using the cluster sampling technique, the entire set of variables could be divided into clusters and a random sample of these clusters is selected. This would make the constraint-based local search a more general solving method and a quantified measure of diversification can be done. The number of clusters to be used or the size of clusters is a key factor for the performance of diversification.

5. The constraint-based local search incorporating the predictive choice model has been applied to GAP. The results demonstrate that the method performs well and can obtain high quality solutions. Therefore, the method may be promising for other combinatorial optimisation problems, particularly those for which it is possible to

maintain the consistency of a subset of the constraints throughout the search. The method is simple and convenient to use; it does not depend on the nature of the objective function or constraints, whether linear or non-linear. As non-linear optimisation problems are generally more computationally intensive than linear ones, it may be fruitful to apply our CLS approach in this area, e.g. the blockmodel problem (Jessop, 2003) which has assignment-type constraints but for which the remainder of the constraints and the objective function take a quadratic form. The principal difficulty in solving this problem is that its continuous relaxation has a non-convex feasible region. Although this problem can be transformed into an integer linear programme, its size expands rapidly and it has been shown to be difficult to solve (Proll, 2004). The predictive choice model may be applied as the utility function of having a certain value for a variable is not affected by the nature of the constraints and the objective function. This would allow the original size of the problem to be retained.

In addition, it may be promising for problems when the constraints in the model keep changing over time, e.g. dynamic vehicle routing problems where vehicles on the pre-route assignment encounter unexpected obstruction and cannot visit their designated customers on time, and therefore vehicles have to be dynamically re-routed. Our CLS approach allows additional constraints and can handle constraints locally. To make use of the predictive choice model, the search history may be defined into two parts: the first part describes the global information guiding the search for quality solutions and the second is the local-update information trying to recover the violated constraints.

Bibliography

Aardal, K. (1998) “Capacitated facility location: separation algorithm and computational experience,” Mathematical Programming, vol. 81, pp. 149 - 175.

Aarts, E.H.L, Lenstra, J.K. and Aarts, E.L. (1997) “Local search in combinatorial optimisation, ” John Wiley & Sons.

Anderson, P.S., Palma, A. and Thisse, J. (1992) “Discrete choice theory of product differentiation,” The MIT Press.

Arshad, F., EL-Rhalibi, A. and Kelleher, G. (2000) “Information management within intermodal transport chain scheduling,” European Project PISCES Report, Liverpool John Moores University, UK.

Ausiello, G., Crescenzi, P. and Protasi, M. (1995) “Approximate solution of NP optimisation problems,” Theoretical Computer Science, vol. 150, pp. 1 - 55.

Balas, E. and Padberg, M. (1976) “Set partitioning: a survey,” SIAM Review, vol. 18, pp. 710 - 760.

Battiti, R. and Protasi, M. (1997) “Reactive search, a history-based heuristic for MAX-SAT,” ACM Journal of Experimental Algorithmics, vol. 2, pp. 1 - 31.

Beasley, J.E. (1988) “An algorithm for solving large capacitated warehouse location problems,” European Journal of Operational Research, vol. 33, pp. 314 - 325.

Ben-Akiva, M. and Lerman, S.R. (1985) “Discrete choice analysis: theory and application to predict travel demand,” MIT Press.

Blum, C., Roli, A. and Dorigo, M. (2001) “HC-ACO: The hyper-cube framework for ant colony optimisation,” In: Proceedings of the Metaheuristics International Conference, MIC 2001, Porto, Portugal, vol. 2, pp. 399 - 403.

Boffey, T.B. (1989) “Location problems arising in computer networks,” Journal of the Operational Research Society, vol.40, pp. 347 - 354.

Brannlund, U., Lindberg, P.O., Nou, A. and Nilsson, J.E. (1998) “Railway timetabling using Lagrangian relaxation,” Transportation Science, vol. 32, pp. 358 - 369.

Brucker, P., Hurink, J.L. and Rolfes, T. (2003) “Routing of railway carriages,” Journal of Global Optimization, vol. 25, pp. 313 - 332.

Chu, P.C. (1997) “A genetic algorithm approach for combinatorial optimization problems,” PhD thesis, Imperial College, London.

Chu, P.C. and Beasley, J.E. (1997) “A genetic algorithm for the generalised assignment problem,” Computers & Operations Research, vol. 24, 17 - 23.

Cook, S.A. (1971) “The complexity of theorem proving procedures,” In Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing, ACM, New York, pp. 151 - 158.

Cordeau, J., Toth, P. and Vigo, D. (1998) “A survey of optimisation models for train routing and scheduling,” Transportation Science, vol. 32, pp. 380 - 404.

Crainic, T., Ferland, J.A. and Rousseau, J.M. (1984) “A tactical planning model for rail freight transportation,” Transportation Science, vol. 18, 165 - 184.

Crainic, T.G. and Laporte, G. (1997) “Planning models for freight transportation,” European Journal of Operational Research, vol. 97, pp. 409 - 438.

Davis, M. and Putnam, H. (1960) “A computing procedure for quantification theory,” Journal of the Association for Computing Machinery, vol. 7, pp. 201 - 215.

Diaz, J.A. and Fernandez, E. (2001) “A tabu search heuristic for the generalised assignment problem, ” European Journal of Operational Research, vol. 132, pp. 22 - 38.

Feltl, H. and Raidl. G.R. (2004) “An improved hybrid genetic algorithm for the generalized assignment,” In: Proceedings of the 2004 ACM Symposium on Applied Computing, pp. 990 - 995.

Feo, T.A. and Resende, M.G.C. (1995) “Greedy randomized adaptative search procedures,” Journal of Global Optimization, vol. 6, pp. 109 - 133.

Frank, J. (1997) “Learning short-term clause weights for GSAT,” In: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97), Morgan Kaufmann Publishers, pp. 384 - 389.

Gendreau, M. (2002), “Recent advances in Tabu search,” in Essays and Surveys in Metaheuristics, C.C. Ribeiro and P. Hansen (eds.), Kluwer Academic Publishers, pp. 369 - 377.

Glover, F. (1986) “Future paths for integer programming and links with artificial intelligence,” Computers and Operations Research, vol. 13, pp. 533 - 549.

Glover, F. and Laguna, M. (1997) “Tabu search,” Kluwer Academic Publishers.

Goldberg, D.E. (1989) “Genetic algorithms in search, optimisation, and machine learning,” Addison-Wesley, Reading, Mass.

Gomes, C.P., Selman, B. and Kautz, H. (1998) “Boosting combinatorial search through randomisation,” In: Proceedings of the 15th International Conference on Artificial Intelligence (AAAI-98), AAAI Press, pp. 431 - 437.

Gorman, M.F. (1998) “An application of genetic and tabu searches to the freight railroad operating plan problem,” Annals of Operations Research, vol. 78, pp. 51 - 69.

Gu, J., Purdom, P.W., Franco, J. and Wah, B.W. (1997) “Algorithms for satisfiability (SAT) problem, a survey,” In: Du, J. Gu, and P.M. Pardalos editors, Satisfiability Problem: Theory and Applications DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 35, pp. 19 - 151.

Gualda, N.F. and Murgel, L.F. (2000) “A model for the train formulation problem,” Third International Meeting for Research in Logistics, Trois-Rivieres, May 9-10, pp. 365 - 378.

Haghani, A.E. (1989) “Formulation and solution of combined train routing and makeup, and empty car distribution model,” Transportation Research, vol. 23B, pp. 433 - 452.

Hansen, E.R (1992) “Global optimisation using interval analysis,” Marcel Dekker, Inc., New York.

Hansen, P. and Jaumard, B. (1990) “Algorithms for the maximum satisfiability problem,” Computing, vol. 44, pp. 279 - 303.

Hansen, P., Jaumard, B., Mladenovic, N. and Parreira, A.D. (2000) “Variable neighbourhood search for maximum weighted satisfiability problem,” Technical Report G-2000-62, Les Cahiers du GERAD, Group for Research in Decision Analysis.

Hansen, P. and Mladenovic, N. (2001) “Variable neighbourhood search: principles and applications,” European Journal of Operational Research, vol. 130, pp. 449 - 467.

Henz, M., Lim, Y.F., Lua, S.C., Shi, X.P., Walser, J.P. and Yap, R. (2000) “Solving hierarchical constraints over finite domains,” In: Proceedings of the 6th International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida, pp. 283 - 298.

Hoos, H.H. (2002) “An adaptive noise mechanism for WalkSAT,” In: Proceedings of the 18th International Conference on Artificial Intelligence (AAAI02), AAAI Press, pp. 655 - 660.

Hoos, H.H. and Stutzle, T. (2004) “Stochastic local search: foundations and applications,” Morgan Kaufmann Press.

Horvitz, E., Ruan, Y., Gomes, C., Kautz, H., Selman, B. and Chickering, M. (2001) “ A Bayesian approach to tackling hard combinatorial problems,” In: Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI01), pp. 235 - 244.

Huntley, C.L., Brown, D.E., Sappington, D.E. and Markowicz, B.P. (1995) “Freight routing and scheduling at CSX Transportation,” Interfaces, vol. 25, pp. 58 - 71.

ILOG (2002) “ILOG OPL studio version 3.5.1: reference manual,” ILOG Inc.

Indra-Payoong, N., Srisurapanon, V. and Laosirihongthong, T. (1998) “Factors influencing modal choice for freight transportation,” In: Proceedings of the Civil and Environmental Engineering Conference, New Frontiers and Challenges, 8-12 November, Bangkok, Thailand vol. 4, pp. 19 - 26.

Jampel, M., Freuder, E. and Maher, M. (1996) “Over-constrained systems,” Lecture Notes in Computer Science, Springer-Verlag, vol. 1106, pp. 1 - 22.

Jessop, A. (2003) “Blockmodels with maximum concentration,” European Journal of Operational Research, vol.148, pp.56 - 64.

Jiang, Y., Kautz, H. and Selman, B. (1995) “Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT,” The 1st International Joint Workshop on Artificial Intelligence and Operations Research, pp. 35 - 50.

Johnson, R. and Wichern, D. (1996) “Applied multivariate statistical analysis,” Cambridge University Press.

Joy, S., Mitchell, J. and Borchers, B. (1997) “A branch and cut algorithm for MAX-SAT and weighted MAX-SAT,” In D. Du, J. Gu, and P.M. Pardalos, editors, Satisfiability Problem: Theory and Applications, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 35, pp. 519 - 536.

Jussien, N. and Lhomme, O. (2002) “Local search with constraint propagation and conflict-based heuristics,” Artificial Intelligence, vol. 139, pp. 21 - 45.

Kallenberg, O. (1997) “Foundations of modern probability,” New York, Springer-Verlag.

Kearfott, R.B. (1998) “On proving existence of feasible points in equality constrained optimisation problems,” Mathematical Programming, vol. 83, pp. 89 - 100.

Keaton, M.H. (1989) “Designing railroad operating plans: Lagrangain relaxation and heuristic approach,” Transportation Research, vol. 23B, pp. 415 - 431.

Keaton, M.H. (1992) “Designing railroad operating plans: a dual adjustment method for implementing Lagrangian relaxation,” Transportation Science, vol. 26, pp. 263 - 274.

Kirkpatrick, S. (1984). “Optimization by simulated annealing: quantitative studies” Journal of Statistical Physics, vol. 34, pp. 976 - 986.

Kochmann, G.A. and McCallum, C.J. (1981) “Facility location models for planning a transatlantic communications network,” European Journal of Operational Research, vol. 6, pp. 205 - 211.

Kondrak, G. and Beek, P.V. (1997) “A theoretical evaluation of selected backtracking algorithms,” Artificial Intelligence, vol. 89, pp. 365 - 387.

Kraft, E.R. (2002) “Scheduling railway freight delivery appointments using a bid price approach,” Transportation Research, vol. 36A, pp. 145 - 165.

Laguna, M., Kelly, J.P., Gonzalez-Verarde, J.L. and Glover, F. (1995) “Tabu search for the multilevel generalised assignment problem,” European Journal of Operational Research, vol. 82, pp.176 - 189.

Larraaga, P. and Lozano, J.A. (2002) “Estimation of distribution algorithms: a new tool for evolutionary computation,” Kluwer Academic Publishers.

Lau H.C., Lim, Y.F. and Liu, Q. (2001) “Diversification of neighbourhood via constraint-based local search and its application to VRPTW,” In: Proceedings of Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR01), pp. 152 - 166.

Lawler, E.W. and Wood, D.E. (1966) “Branch and bound method: a survey, ” Operations Research, vol. 14, pp. 699 - 719.

Lester, I. (1996) “Adaptive simulated annealing (ASA): lessons learned,” Control and Cybernetics, vol. 25, 33 - 54.

Lourenco, H. and Serra, D. (2002) “Adaptive search heuristics for the generalized assignment problem,” Mathware and Soft Computing, vol. 9, pp. 209 - 234.

Lourenco, H.R, Martin, O.C. and Stutzle, T. (2002) “Iterated local search,” Handbook of Metaheuristics, Ed. F. Glover and G. Kochenberger, Kluwer, vol. 57, pp. 321 - 353.

Marin, A. and Sameron, J. (1996) “Tactical design of rail freight networks: part II: local search methods with statistical analysis,” European Journal of Operational Research, vol. 94, pp. 43 - 53.

Marriott, K. and Stuckey, P.J. (1998) “ Programming with constraints: an introduction” MIT press.

Mazure, B., Sais, L. and Gregoire, E. (1997) “ Tabu search for SAT,” In: Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97), pp. 281 - 285.

McAllester, D., Selman, B. and Kautz, H. (1997) “Evidence for invariants in local search,” In: Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97), pp. 321 - 326.

Mills, P. and Tsang, E. (2000) “Guided local search for solving SAT and Weighted MAX-SAT Problems,” Journal of Automated Reasoning, vol. 24, pp. 205 - 223.

Ministry of Commerce. (2002) “Thai commodity market price,” Department of Business Economics, Ministry of Commerce, Thailand.

Minton, S., Johnston, M., Philips, A. and Laird, P. (1992) “Minimizing conflicts: a heuristic method for constraint satisfaction and scheduling problems,” Artificial Intelligence, vol. 58, pp. 161 - 205.

Mitchell, J.E. (2000) “Branch and cut algorithms for combinatorial optimisation problems,” Handbook of Applied Optimisation, Oxford University Press.

Narciso, M.G. and Lorena, L.A.N. (1999) “Lagrangean/surrogate relaxation for generalised assignment problems,” European Journal of Operational Research, vol. 114, pp. 165 – 177.

National Economic and Social Development Board, Transport Statistics, 2000.

Nauss, R.M. (2003) “Solving the generalized assignment problem: an optimizing and heuristic approach, ” INFORMS Journal of Computing, vol. 15, pp. 249 – 266.

Nemhauser, G.L. and Wolsey, L.A. (1988) “Integer and combinatorial optimisation,” John Wiley & Sons.

Newman, A.M. and Yano, C.A. (2000) “Scheduling direct and indirect trains and containers in an intermodal setting,” Transportation Science, vol. 34, 256 - 270.

Newton, H.N., Barnhart, C., Vance, P.H. (1998) “Constructing railroad blocking plans to minimise handling costs,” Transportation Science, vol. 32, pp. 330 - 342.

Osman, I.H. (1995) “Heuristics for the generalised assignment problem: simulated annealing and tabu search approaches,” OR Spectrum, vol. 17, pp. 211 - 225.

Osman, I.H. and Kelly, J.P. (1996) “Meta-heuristics: an overview,” In: Osman, I.H., Kelly, J.P. (Eds), Metaheuristics: Theory & Applications, Kluwer, Boston, Chapter 1, pp. 1 - 21.

Sahni, S. and Gonzalez, T. (1976) “P-complete approximation problems,” Journal of ACM, vol. 23, pp. 555 - 565.

Pardalos, P.M., Pitsoulis, L.S. and Resende, M.G.C. (1996) “A parallel GRASP for MAX-SAT problems,” In: Jerzy Wasniewski, Jack Dongarra, Kaj Madsen, Dorte Olesen (Eds.): Applied Parallel Computing, Industrial Computation and Optimization, Third International Workshop, PARA 96 Lecture Notes in Computer Science, vol. 1184, Springer pp. 575-585.

Pearson, A.V. and Hartley, H.O. (1972) “Biometrica tables for statisticians,” vol 2, Cambridge, England, Cambridge University Press.

Pelikan, M., Goldberg, D.E., Cant-Paz, E. (1999) “BOA: the Bayesian optimization algorithm,” In: Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99, vol. I, pp. 525-532.

Pochet, Y. and Wolsey, L.A. (1988) “Lot-size models with backlogging: strong reformulations and cutting planes,” Mathematical Programming, vol. 40, pp. 317 - 335.

Proll, L. (2004) “ILP approaches to the blockmodel problem,” University of Leeds School of Computing Research Report 2004.05.

Proll, L. and Smith, B. (1998) “ILP and constraint programming approaches to a template design problem,” INFORMS Journal of Computing, vol. 10, pp. 265 - 277.

Prosser, P. (1993) “Hybrid algorithms for the constraint satisfaction problem,” Computational Intelligence, vol. 9, pp. 268 – 299.

Rardin, R. (1998) “Optimization in operations research,” Prentice Hall.

Resende, M.G.C., Pitsoulis, L.S. and Pardalos, P.M. (1997) “Approximate solution of weighted MAX-SAT problems using GRASP,” In: Du, J. Gu, and P.M. Pardalos editors, Satisfiability Problem: Theory and Applications DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 35, pp. 393 - 405.

Roli, A. (2001) “Criticality and parallelism in GSAT,” Electronic Notes in Discrete Mathematics, vol. 9, pp. 1-8.

Roli, A. and Blum, C. (2001) “Critical parallelisation of local search for MAX-SAT, ” In: Proceedings of AI, 7th Congress of the Italian Association of Artificial Intelligence, vol. 2175, pp. 147 - 158.

Salim, V. and Cai, X. (1997) “A genetic algorithm for railway scheduling with environmental considerations,” Environmental Modelling and Software, vol. 12, pp. 301 - 309.

SAS (2002) “SAS/STAT software,” SAS Inc.

Schrijver, A. (1993) “Minimum circulation of railway stock,” CWI Quarterly, vol. 6, pp. 205 - 217.

Selman, B. and Kautz, H. (1993) “Domain-independent extensions to GSAT: Solving large structured satisfiability problems,” In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-93), pp. 290-295.

Selman, B., Levesque, H. and Mitchell, D. (1992) “A new method for solving hard satisfiability problems,” In: Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92), pp. 440-446.

Selman, B. Kautz, H.A. and Cohen, B. (1994) “Noise strategies for improving local search,” In: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94), pp. 337 - 343.

Shapiro, S.S. and Wilk, M.B. (1965) “An analysis of variance test for normality (complete samples),” Biometrika, vol. 52, pp. 591 - 611.

Smith, B., Stergiou, K. and Walsh, T. (2000) “Using auxiliary variables and implied constraints to model non-binary problems,” In: Proceedings of AAAI-2000, Austin, Texas.

Steinmann, O., Strohmaier, A. and Stutzle, T. (1997) “Tabu search vs. random walk,” In: Proceedings of the 21st Annual German Conference on Artificial Intelligence: Advances in Artificial Intelligence, Lecture Notes in Computer Science, vol. 1303, pp. 337 – 348.

Strohmaier, A. (1998) “Multi-flip networks for SAT,” In: Proceedings of the 22nd Annual German Conference on Artificial Intelligence: Advances in Artificial Intelligence, Lecture Notes in Computer Science, vol. 1305, pp. 349 - 360.

Trotter, H.F. (1959) “An elementary proof of the central limit theorem,” Archiv der Mathematik, vol. 10, pp. 226 - 234.

Tsang, E.P.K. (1993) “Foundations of constraint satisfaction,” Academic Press.

Walser, J.P. (1999) “Integer optimisation by local search: a domain-independent approach,” Lecture Notes in Artificial Intelligence 1637, Springer-Verlag.

Winston, W.L. (1994) “Operations research – applications and algorithms,” Duxbury Press.

Wolfe, M.A. (1994) “An interval algorithm for constrained global optimisation,” Journal of Computational and Applied Mathematics, vol. 50, pp. 605 - 612.

Yagiura, M. and Ibaraki, T. (1999) “Analyses on the 2 and 3-flip neighbourhoods for the MAX SAT,” Journal of Combinatorial Optimisation, vol. 3, no. 1, pp. 95 - 114.

Yagiura, M. Ibaraki, T. and Glover, F. (2004) “An ejection chain approach for the generalised assignment problem,” INFORMS Journal of Computing, vol. 16, pp. 133 - 151.

Yagiura, M. Yamaguchi, T. and Ibaraki, T. (1999a) “A variable depth search algorithm for the generalised assignment problem, ” In: Voss, S. Martello, S. Osman, I.H. and Roucariol, C. (Eds.), Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimisation, Kluwer Academic Publishers, pp. 459 - 471.

Yano, C.A. and Newman, A.M. (2001) “Scheduling trains and containers with due dates and dynamic arrivals,” Transportation Science, vol. 35, pp. 181 - 191.

Appendix A

Total shipping cost for containerised cargoes as a percentage of commodity market prices. In all tables, ΔC is the difference between total shipping cost of truck and rail, i.e. $\Delta C = C_T - C_R$, Z is a standardised score of ΔC, and CDF is a cumulative probability density function of ΔC.

Customer no.	Modal cost (%)			Sort			
	Truck, C_T	Rail, C_R	ΔC	No.	ΔC	Z	CDF
1	8.64	6.42	2.22	1	1.04	-1.66	0.05
2	14.11	12.29	1.82	2	1.07	-1.59	0.06
3	9.65	8.01	1.64	3	1.11	-1.49	0.07
4	8.32	6.75	1.57	4	1.27	-1.10	0.14
5	12.12	11.01	1.11	5	1.35	-0.90	0.18
6	9.00	7.65	1.35	6	1.40	-0.78	0.22
7	13.55	11.46	2.09	7	1.40	-0.77	0.22
8	15.07	12.98	2.09	8	1.47	-0.61	0.27
9	11.78	9.87	1.91	9	1.52	-0.49	0.31
10	12.09	9.87	2.22	10	1.56	-0.39	0.35
11	9.75	8.19	1.56	11	1.57	-0.37	0.36
12	14.18	13.14	1.04	12	1.64	-0.20	0.42
13	9.14	6.98	2.16	13	1.77	0.12	0.55
14	15.21	13.12	2.09	14	1.82	0.24	0.60
15	8.50	6.98	1.52	15	1.91	0.46	0.68
16	10.08	8.68	1.40	16	1.97	0.61	0.73
17	8.08	7.01	1.07	17	2.09	0.90	0.82
18	13.01	11.24	1.77	18	2.09	0.90	0.82
19	10.52	9.12	1.40	19	2.09	0.90	0.82
20	16.17	14.01	2.16	20	2.16	1.07	0.86
21	11.11	8.65	2.46	21	2.16	1.07	0.86
22	13.27	12.00	1.27	22	2.22	1.22	0.89
23	16.05	14.58	1.47	23	2.22	1.22	0.89
24	15.07	13.10	1.97	24	2.46	1.80	0.96
			μ=1.72				
			σ=0.41				

Table A.1: Total shipping cost for cargo type I

Customer	Modal cost (%)			Sort			
no.	Truck, C_T	Rail, C_R	ΔC	No.	ΔC	Z	CDF
1	10.66	5.62	5.04	1	3.89	-2.45	0.01
2	15.04	10.66	4.38	2	4.07	-1.97	0.02
3	14.84	9.78	5.06	3	4.14	-1.79	0.04
4	13.89	8.97	4.92	4	4.23	-1.55	0.06
5	10.71	5.95	4.76	5	4.37	-1.18	0.12
6	16.05	10.11	5.94	6	4.38	-1.16	0.12
7	10.80	5.78	5.02	7	4.38	-1.16	0.12
8	10.05	5.67	4.38	8	4.45	-0.97	0.17
9	11.25	6.62	4.63	9	4.47	-0.92	0.18
10	15.11	10.21	4.90	10	4.47	-0.92	0.18
11	10.10	6.21	3.89	11	4.53	-0.76	0.22
12	9.04	3.90	5.14	12	4.54	-0.74	0.23
13	10.75	5.70	5.05	13	4.56	-0.68	0.25
14	16.01	11.54	4.47	14	4.61	-0.55	0.29
15	10.65	5.97	4.68	15	4.61	-0.55	0.29
16	8.11	3.15	4.96	16	4.63	-0.50	0.31
17	16.45	11.53	4.92	17	4.68	-0.37	0.36
18	9.98	5.04	4.94	18	4.70	-0.32	0.38
19	18.15	13.54	4.61	19	4.75	-0.18	0.43
20	10.85	5.88	4.97	20	4.75	-0.18	0.43
21	10.85	5.70	5.15	21	4.76	-0.16	0.44
22	10.70	5.81	4.89	22	4.77	-0.13	0.45
23	10.11	5.21	4.90	23	4.78	-0.11	0.46
24	11.08	6.54	4.54	24	4.80	-0.05	0.48
25	18.01	13.21	4.80	25	4.83	0.03	0.51
26	10.87	5.52	5.35	26	4.89	0.18	0.57
27	19.01	14.10	4.91	27	4.90	0.21	0.58
28	16.07	11.11	4.96	28	4.90	0.21	0.58
29	17.24	13.01	4.23	29	4.90	0.21	0.58
30	10.11	5.50	4.61	30	4.91	0.24	0.59
31	9.15	5.01	4.14	31	4.92	0.26	0.60
32	14.20	9.50	4.70	32	4.92	0.26	0.60
33	16.11	12.04	4.07	33	4.92	0.26	0.60
34	12.04	6.80	5.24	34	4.94	0.32	0.62
35	15.89	10.87	5.02	35	4.96	0.37	0.64
36	12.82	8.07	4.75	36	4.96	0.37	0.64
37	10.60	5.77	4.83	37	4.97	0.39	0.65
38	13.54	9.01	4.53	38	4.97	0.39	0.65
39	9.02	4.25	4.77	39	5.02	0.53	0.70
40	16.07	11.10	4.97	40	5.02	0.53	0.70
41	11.80	6.44	5.36	41	5.04	0.58	0.72
42	15.32	9.57	5.75	42	5.04	0.58	0.72
43				43			
	11.11	6.64	4.47		5.05	0.61	0.73

Customer no.	Modal cost (%)			Sort			
	Truck, C_T	Rail, C_R	ΔC	No.	ΔC	Z	CDF
44	9.05	4.30	4.75	44	5.06	0.63	0.74
45	12.99	8.21	4.78	45	5.11	0.76	0.78
46	9.04	4.00	5.04	46	5.14	0.84	0.80
47	11.47	6.55	4.92	47	5.15	0.87	0.81
48	16.24	11.87	4.37	48	5.24	1.11	0.87
49	12.77	8.32	4.45	49	5.35	1.39	0.92
50	10.54	5.98	4.56	50	5.36	1.42	0.92
51	18.11	13.00	5.11	51	5.75	2.45	0.99
52	10.10	5.20	4.90	52	5.94	2.95	0.99
			$\mu = 4.82$				
			$\sigma = 0.38$				

Table A.2: Total shipping cost for cargo type II

Customer no.	Modal cost (%)			Sort			
	Truck, C_T	Rail, C_R	ΔC	No.	ΔC	Z	CDF
1	13.34	11.03	2.31	1	1.50	-2.45	0.01
2	14.85	12.56	2.29	2	1.52	-2.39	0.01
3	14.28	12.29	1.99	3	1.60	-2.15	0.02
4	10.35	8.01	2.34	4	1.60	-2.15	0.02
5	12.87	10.75	2.12	5	1.65	-2.00	0.02
6	12.53	11.01	1.52	6	1.77	-1.64	0.05
7	9.30	7.65	1.65	7	1.85	-1.39	0.08
8	14.00	11.46	2.54	8	1.89	-1.27	0.10
9	15.32	12.98	2.34	9	1.98	-1.00	0.16
10	11.86	9.87	1.99	10	1.99	-0.97	0.17
11	11.72	9.87	1.85	11	1.99	-0.97	0.17
12	10.44	8.19	2.25	12	2.00	-0.94	0.17
13	15.68	13.14	2.54	13	2.09	-0.67	0.25
14	12.48	9.98	2.50	14	2.10	-0.64	0.26
15	15.64	13.12	2.52	15	2.10	-0.64	0.26
16	9.72	6.98	2.74	16	2.11	-0.61	0.27
17	10.78	8.68	2.10	17	2.11	-0.61	0.27
18	8.99	7.01	1.98	18	2.12	-0.58	0.28
19	19.53	17.02	2.51	19	2.22	-0.27	0.39
20	10.72	9.12	1.60	20	2.25	-0.18	0.43
21	16.46	14.01	2.45	21	2.29	-0.06	0.48
22	11.19	8.65	2.54	22	2.29	-0.06	0.48
23	14.64	12.00	2.64	23	2.30	-0.03	0.49

Customer no.	Modal cost (%)			Sort			CDF
	Truck, C_T	Rail, C_R	ΔC	No.	ΔC	Z	
24	17.02	14.58	2.44	24	2.30	-0.03	0.49
25	15.40	13.10	2.30	25	2.30	-0.03	0.49
26	10.98	8.68	2.30	26	2.30	-0.03	0.49
27	11.19	8.71	2.48	27	2.30	-0.03	0.49
28	9.65	7.01	2.64	28	2.30	-0.03	0.49
29	16.48	13.24	3.24	29	2.30	-0.03	0.49
30	11.46	9.12	2.34	30	2.31	0.00	0.50
31	16.55	14.01	2.54	31	2.31	0.00	0.50
32	11.29	8.65	2.64	32	2.32	0.03	0.51
33	14.74	12.00	2.74	33	2.33	0.06	0.52
34	16.98	14.58	2.40	34	2.34	0.09	0.54
35	15.54	13.10	2.44	35	2.34	0.09	0.54
36	17.75	15.64	2.11	36	2.34	0.09	0.54
37	13.87	11.78	2.09	37	2.40	0.27	0.61
38	14.39	12.09	2.30	38	2.40	0.27	0.61
39	12.08	9.75	2.33	39	2.40	0.27	0.61
40	16.28	14.18	2.10	40	2.41	0.30	0.62
41	11.44	9.14	2.30	41	2.42	0.33	0.63
42	17.61	15.21	2.40	42	2.43	0.36	0.64
43	10.81	8.50	2.31	43	2.44	0.39	0.65
44	12.08	10.08	2.00	44	2.44	0.39	0.65
45	10.66	8.08	2.58	45	2.44	0.39	0.65
46	15.31	13.02	2.29	46	2.45	0.42	0.66
47	13.26	10.52	2.74	47	2.46	0.45	0.68
48	17.67	16.17	1.50	48	2.48	0.52	0.70
49	13.00	11.11	1.89	49	2.50	0.58	0.72
50	15.59	13.27	2.32	50	2.51	0.61	0.73
51	18.48	16.05	2.43	51	2.52	0.64	0.74
52	16.84	15.07	1.77	52	2.54	0.70	0.76
53	20.57	18.03	2.54	53	2.54	0.70	0.76
54	14.17	11.62	2.55	54	2.54	0.70	0.76
55	14.73	12.29	2.44	55	2.54	0.70	0.76
56	10.31	8.01	2.30	56	2.54	0.70	0.76
57	15.42	12.75	2.67	57	2.55	0.73	0.77
58	20.21	17.11	3.10	58	2.58	0.82	0.79
59	12.07	9.65	2.42	59	2.64	1.00	0.84
60	13.92	11.46	2.46	60	2.64	1.00	0.84
61	15.54	13.24	2.30	61	2.64	1.00	0.84
62	12.93	10.52	2.41	62	2.67	1.09	0.86
63	20.05	18.45	1.60	63	2.74	1.30	0.90
64	13.22	11.11	2.11	67	2.74	1.30	0.90
65	15.67	13.27	2.40	65	2.74	1.30	0.90
66	18.27	16.05	2.22	66	3.10	2.39	0.99

Customer no.	Modal cost (%)			Sort			CDF
	Truck, C_T	Rail, C_R	ΔC	No.	ΔC	Z	
67	17.42	15.12	2.30 $\mu = 2.31$ $\sigma = 0.33$	67	3.24	2.82	0.99

Table A.3: Total shipping cost for cargo type III

Customer no.	Modal cost (%)			Sort			
	Truck, C_T	Rail, C_R	ΔC	No.	ΔC	Z	CDF
1	9.94	4.63	5.31	1	4.22	-2.95	0.01
2	10.54	4.90	5.64	2	4.52	-2.14	0.02
3	8.66	3.89	4.77	3	4.64	-1.81	0.04
4	10.49	5.14	5.35	4	4.64	-1.81	0.04
5	9.86	4.05	5.81	5	4.77	-1.46	0.07
6	9.48	4.47	5.01	6	4.89	-1.14	0.13
7	11.73	6.35	5.38	7	4.96	-0.95	0.17
8	10.56	4.96	5.60	8	5.01	-0.81	0.21
9	17.28	11.92	5.36	9	5.12	-0.52	0.30
10	10.62	4.94	5.68	10	5.12	-0.51	0.30
11	12.91	7.61	5.30	11	5.20	-0.30	0.38
12	10.69	4.97	5.72	12	5.21	-0.27	0.39
13	10.47	5.15	5.32	13	5.22	-0.24	0.40
14	10.11	4.89	5.22	14	5.22	-0.24	0.40
15	15.54	9.90	5.64	15	5.30	-0.02	0.49
16	9.76	4.54	5.22	16	5.30	-0.02	0.49
17	10.32	4.80	5.52	17	5.31	0.00	0.50
18	10.81	5.35	5.46	18	5.32	0.03	0.51
19	10.56	4.91	5.65	19	5.32	0.03	0.51
20	14.53	8.93	5.60	20	5.35	0.11	0.54
21	9.19	4.23	4.96	21	5.36	0.14	0.55
22	15.91	10.61	5.30	22	5.38	0.19	0.58
23	8.78	4.14	4.64	23	5.41	0.26	0.60
24	12.11	6.70	5.41	24	5.46	0.41	0.66
25	13.16	8.64	4.52	25	5.46	0.41	0.66
26	10.74	5.24	5.50	26	5.46	0.41	0.66
27	10.79	5.02	5.77	27	5.46	0.41	0.66
28	9.21	3.75	5.46	28	5.49	0.47	0.68
29	10.38	4.83	5.55	29	5.50	0.51	0.70
30	14.74	9.53	5.21	30	5.52	0.57	0.71
31	9.26	3.77	5.49	31	5.55	0.66	0.75

Customer no.	Modal cost (%)			Sort			
	Truck, C_T	Rail, C_R	ΔC	No.	ΔC	Z	CDF
32	10.09	4.97	5.12	32	5.60	0.78	0.78
33	11.42	5.36	6.06	33	5.60	0.78	0.78
34	11.07	5.75	5.32	34	5.64	0.88	0.81
35	11.11	6.47	4.64	35	5.64	0.88	0.81
36	10.21	4.75	5.46	36	5.65	0.91	0.82
37	8.34	3.45	4.89	37	5.68	1.00	0.84
38	10.24	5.04	5.20	38	5.72	1.10	0.86
39	9.38	3.92	5.46	39	5.77	1.25	0.89
40	11.45	7.23	4.22	40	5.81	1.34	0.91
41	9.57	4.45	5.12	41	6.06	2.03	0.98
			$\mu = 5.31$				
			$\sigma = 0.37$				

Table A.4: Total shipping cost for cargo type IV

Appendix B

Tables B.1 – B.5 show the accuracy of the predictive choice model. The same variables are tested in each table with different prediction no.

Var no.	Violation		$\aleph^*$		Probability		Φ
	$\bar{h}_0$	$\bar{h}_1$	0	1	P_0	P_1	
1	25.55	26.50	17	3	0.85	0.15	0
2	23.25	21.90	14	6	0.70	0.30	1
3	15.30	23.55	9	11	0.86	0.14	0
4	25.45	22.70	11	9	0.47	0.53	0
5	30.05	24.40	4	16	0.20	0.80	1
6	28.10	25.65	10	10	0.48	0.52	0
7	27.45	31.45	16	4	0.80	0.20	0
8	23.90	19.20	13	7	0.36	0.64	1
9	20.70	23.30	17	3	0.85	0.15	0
10	25.75	23.85	17	3	0.85	0.15	0
11	28.05	22.20	8	12	0.35	0.65	1
12	24.20	26.00	13	7	0.53	0.47	0
13	25.60	29.65	8	12	0.59	0.41	0
14	25.55	27.00	16	4	0.80	0.20	0
15	25.35	28.10	15	5	75	25	0
16	29.65	23.60	5	15	25	75	1
17	24.05	25.80	14	6	70	30	0
18	24.20	26.00	7	13	0.53	0.47	0
19	26.00	20.50	6	14	30	70	1
20	23.05	24.90	15	5	75	25	0
						$PC =$	85%

Table B.1: Prediction no. 1

Var no.	Violation		$\aleph^*$		Probability		Φ
	$\bar{h}_0$	$\bar{h}_1$	0	1	P_0	P_1	
1	22.45	27.55	6	14	0.67	0.33	0
2	25.80	25.40	13	7	0.49	0.51	1
3	22.85	20.30	13	7	0.53	0.47	1
4	25.70	29.25	10	10	0.57	0.43	0
5	24.70	19.05	5	15	0.25	0.75	1
6	28.05	25.80	12	8	0.49	0.51	0
7	22.45	28.55	6	14	0.72	0.28	0
8	32.30	27.70	7	13	0.42	0.58	1
9	20.40	24.35	18	2	0.90	0.10	0
10	20.40	24.35	14	6	0.73	0.27	0
11	31.10	24.95	10	10	0.35	0.65	1
12	25.70	28.45	17	3	0.85	0.25	0
13	29.25	27.55	8	12	0.61	0.39	0
14	22.45	27.55	14	6	0.76	0.24	0
15	25.10	30.50	17	3	0.85	0.25	0
16	32.60	24.35	3	17	0.25	0.85	1
17	28.65	26.75	13	7	0.46	0.54	0
18	25.70	28.45	17	3	0.85	0.25	0
19	32.40	26.00	7	13	0.70	0.30	1
20	20.40	27.55	15	5	0.75	0.25	0
						$PC =$	80%

Table B.2: Prediction no. 2

Var no.	Violation		$\aleph^*$		Probability		Φ
	$\bar{h}_0$	$\bar{h}_1$	0	1	P_0	P_1	
1	19.70	25.00	19	1	0.95	0.05	0
2	20.65	17.35	13	7	0.48	0.52	1
3	21.60	19.25	11	9	0.48	0.52	1
4	22.60	26.20	15	5	0.75	0.25	0
5	21.85	16.30	3	17	0.15	0.85	1
6	20.95	24.90	14	6	0.74	0.26	0
7	20.25	25.00	18	2	0.90	0.20	0
8	25.15	24.85	12	8	0.51	0.49	1
9	21.65	26.60	6	14	0.67	0.33	0
10	19.70	25.00	19	1	0.95	0.05	0
11	26.90	19.70	6	14	0.31	0.69	1
12	26.85	26.25	11	9	0.49	0.51	0
13	22.90	24.20	14	6	0.73	0.27	0
14	19.70	26.60	17	3	0.85	0.15	0
15	22.90	20.95	13	7	0.38	0.62	0
16	29.75	22.15	14	6	0.24	0.76	1
17	25.50	30.95	17	3	0.85	0.15	0
18	26.85	26.25	9	11	0.49	0.51	0
19	23.05	16.05	4	16	0.20	0.80	1
20	19.70	24.65	18	2	0.90	0.10	0
						$PC =$	80%

Table B.3: Prediction no. 3

Var no.	Violation		$\aleph^*$		Probability		Φ
	$\bar{h}_0$	$\bar{h}_1$	0	1	P_0	P_1	
1	26.25	26.90	15	5	0.75	0.25	0
2	26.20	21.45	11	9	0.40	0.60	1
3	19.15	21.95	13	7	0.56	0.44	1
4	27.85	27.15	13	7	0.51	0.49	0
5	25.85	18.80	3	17	0.15	0.85	1
6	27.80	27.90	11	9	0.51	0.49	1
7	20.35	26.25	16	4	0.80	0.20	0
8	25.25	23.65	14	6	0.42	0.58	0
9	23.65	25.40	14	6	0.68	0.32	0
10	21.45	26.90	17	3	0.85	0.15	0
11	27.10	18.70	4	16	0.20	0.80	1
12	21.15	27.65	4	16	0.66	0.33	0
13	24.70	23.95	12	8	0.48	0.52	0
14	19.65	24.20	7	13	0.80	0.20	0
15	22.20	27.15	15	5	0.75	0.25	0
16	26.35	17.90	4	16	0.19	0.81	1
17	26.80	23.55	10	10	0.46	0.54	0
18	26.15	27.65	16	4	0.80	0.20	0
19	16.30	24.60	4	16	0.20	0.80	0
20	18.55	23.20	13	7	0.67	0.33	0
						PC =	70%

Table B.4: Prediction no. 4

Var no.	Violation		$\aleph^*$		Probability		Φ
	$\bar{h}_0$	$\bar{h}_1$	0	1	P_0	P_1	
1	24.35	27.50	6	14	0.73	0.27	0
2	20.05	21.60	11	9	0.54	0.46	1
3	23.35	21.95	14	6	0.61	0.39	1
4	19.65	22.20	13	7	0.55	0.45	0
5	28.45	19.65	3	17	0.15	0.85	1
6	24.60	23.90	9	11	0.51	0.49	0
7	23.30	26.25	15	5	0.75	0.25	0
8	24.50	23.45	9	11	0.52	0.48	1
9	23.05	25.60	18	2	0.90	0.10	0
10	21.20	26.15	17	3	0.85	0.15	0
11	26.50	22.60	11	9	0.44	0.56	1
12	22.55	25.75	15	5	0.75	0.25	0
13	23.85	21.90	8	12	0.49	0.51	0
14	21.85	26.10	6	14	0.67	0.23	0
15	23.95	25.05	14	6	0.72	0.28	0
16	26.70	20.55	12	8	0.26	0.64	1
17	26.80	27.15	15	5	0.75	0.25	0
18	22.55	25.75	15	5	0.75	0.25	0
19	30.35	22.35	12	8	0.29	0.71	1
20	22.30	24.10	7	13	0.58	0.42	0
						$PC =$	80%

Table B.5: Prediction no. 5