Fuzzy Evolutionary Approaches for Bus and Rail Driver Scheduling

By

Jingpeng Li

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

THE UNIVERSITY OF LEEDS
SCHOOL OF COMPUTING

July 2002

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

Acknowledgements

I would like to thank my supervisor, Dr. Raymond R. S. Kwan, for his guidance and supervision.

I would like to thank my family members for their patient and support, and close friends for their various kinds of help.

Finally I wish thank and acknowledge the full-scholarship for my three years' PhD studies, from the School of Computing, University of Leeds.

Abstract

Bus and train driver scheduling is a process of partitioning blocks of work, each of which is serviced by one vehicle, into a set of legal driver shifts. The main objectives are to minimise the total number of shifts and the total shift cost. Restrictions imposed by logistic, legal and union agreements make the problem more complicated.

The generate-and-select approach is widely used. A large set of feasible shifts is generated first, and then a subset is selected, from the large set, to form a final schedule by the mathematical programming method. In the subset selection phase, computational difficulties exist because of the NP-hard nature of this combinatorial optimisation problem. This thesis presents two evolutionary algorithms, namely a Genetic Algorithm and a Simulated Evolution algorithm, attempting to model and solve the driver scheduling problem in new ways.

At the heart of both algorithms is a function for evaluating potential driver shifts under fuzzified criteria. A Genetic Algorithm is first employed to calibrate the weight distribution among fuzzy membership functions. A Simulated Evolution algorithm then mimics generations of evolution on the single schedule produced by the Genetic Algorithm. In each generation an unfit portion of the working schedule is removed. The broken schedule is then reconstructed by means of a greedy algorithm, using the weight distribution derived by the Genetic Algorithm. The basic Simulated Evolution algorithm is a greedy search strategy that achieves improvement through iterative perturbation and reconstruction. This approach has achieved success in solving driver scheduling problems from different companies, with comparable results to the previously best known solutions.

Finally, the Simulated Evolution algorithm for driver scheduling has been generalized for the set covering problem, without using any special domain knowledge. This shows that this research is valuable to many applications that can be formulated as set covering models. Furthermore, Taguchi's orthogonal experimental design method has been used for the parameter settings. Computational results have shown that for large-scale problems, in general the proposed approach can produce superior solutions much faster than some existing approaches. This approach is particularly suitable for situations 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:

Li, J. and Kwan, R.S.K. (2002) "A fuzzy genetic algorithm for driver scheduling," (to appear in) European Journal of Operational Research (Special Issue on Fuzzy Scheduling and Planning).

Li, J. and Kwan, R.S.K. (2002) "A fuzzy evolutionary approach with Taguchi parameter setting for the set covering problem," Proceedings of the 2002 IEEE Congress on Evolutionary Computation, IEEE Service Center, pp.1203-1208.

Li, J. and Kwan, R.S.K. (2001) "A fuzzy theory based evolutionary approach for driver scheduling," in Spector, L. et al. (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufman, pp. 1152-1158.

Li, J. and Kwan, R.S.K. (2001) "Evolutionary driver scheduling with fuzzy evaluation," Proceedings of the Graduate Student Workshop at Genetic and Evolutionary Computation Conference, pp.421-424, July 8, San Francisco, USA.

Li, J. and Kwan, R.S.K. (2001) "A fuzzy simulated evolution algorithm for the driver scheduling problem," Proceedings of the 2001 IEEE Congress on Evolutionary Computation, IEEE Service Center, pp.1115-1122.

Li, J. and Kwan, R.S.K. (2000) "Genetic algorithms for the bus and rail driver scheduling problem," in the 11th Conference of Young Operational Research, pp. 39, March 28-30, Cambridge, UK.

Kwan, R.S.K., Kwan, A.S.K. and Li, J. (2000) "Genetic algorithms for integer solutions of bus and rail driver scheduling," (presented at) The 17th International Symposium on mathematical Programming, August 7-11, Atlanta, USA.

Contents

List of Tables

List of Figures

Chapter One

Introduction

1.1 The bus and rail driver scheduling problem

Bus and train driver scheduling is a process of partitioning blocks of work, each of which is serviced by one vehicle, into a set of legal driver shifts. The main objectives are to minimise the total number of shifts and the total shift costs. This problem has attracted much interest since the 1960s.

Bus driver scheduling and train driver scheduling are fundamentally the same problem, but the former can be regarded as a special case of the latter since all the features in bus operation can be found in train operation but not vice versa (Kwan 1999). In comparison, both problems involve allocation of driver work to shifts that are similar in structure, have similar working hours with at least one meal break, and must obey similar labour agreement rules.

Compared with bus driver scheduling, train driver scheduling is much more complex, some of the factors are listed as follows (Kwan 1999):

- In case that a driver need to travel as passenger from one place to another, in train driver scheduling it is necessary to find out the exact departure and arrival time for the driver to travel. However, in bus driver scheduling a constant allowance for the travelling is usually adequate due to short distance to be travelled and frequent services;
- The bus driver scheduling problem is often solved as a single-depot problem, which is much easier than the multi-depot case typical of train driver scheduling, since bus drivers are usually restricted to buses from their home depots;
- Compared to train work, bus work is less fragmented, and does not contain non-wheel turning work such as unit preparation and disposal. Hence bus driver shifts usually contain work on no more than three different buses;
- Meal break rules in the bus situation are simpler than those in the train situation, and the maximum number of meal breaks rarely exceeds two;
- In bus driver scheduling, there is usually no restriction on route and traction knowledge.

The driver scheduling problem is commonly solved using a set covering approach. A large set of possible legal shifts is first generated by heuristics, which are usually highly parameterized to reflect on the driver work rules of individual companies. Then, a least cost subset covering all the work is selected to form a solution schedule. The success and limitations of such a standard approach, as exemplified in TRACS II, which used a blend of heuristics and Integer Linear Programming, have been discussed in Kwan et al. (2000).

1.1.1 Vehicle work

Vehicle (i.e. bus or train in this thesis) units are normally scheduled before driver schedules are compiled. In a vehicle schedule, the work is usually denoted as a set of blocks. A block is a sequence of trips operated by one vehicle in one day, beginning with a pull-out from, and ending with a pull-in, to a depot. A vehicle leaving from and returning to the depot more than once in a day constitutes multiple blocks.

Diagram 1:
Depart from depot at Manchester Piccadilly at 02:32

Location	Arrival Time	Departure Time	
Manchester Picc.		02:32	
Huddersfield	03:04	03:06	
Leeds	03:28	03:31	
York	04:00	04:22	
Middlesborough	05:28	06:18	Clean
York	07:15	07:18	
Leeds	07:47	07:51	
Huddersfield	08:12	08:14	
Manchester Picc.	08:51	08:56	
Manchester Air.	09:11	09:28	Clean
Manchester Picc.	09:45	09:50	
Huddersfield	10:22	10:24	
Leeds	10:49	10:54	
York	11:23	11:26	
Middlesborough	12:26	13:18	Clean
York	14:15	14:18	
Leeds	14:45	14:51	
Huddersfield	15:12	15:14	
Manchester Picc.	15:49	15:55	
Manchester Air.	16:10	16:28	Clean
Manchester Picc.	16:43	16:49	
Huddersfield	17:22	17:24	
Leeds	17:52	17:54	
York	18:22	18:26	
Middlesborough	19:27	20:18	Clean
York	21:12	21:21	
Leeds	21:48	21:51	
Huddersfield	22:12	22:14	
Manchester Picc.	22:51	22:55	
Manchester Air.	23:11	23:28	Clean
Manchester Picc.	23:44	23:50	
Huddersfield	00:22	00:24	
Leeds	00:46	00:53	
York	01:22		Clean

Figure 1.1: An example of a unit diagram (Kwan 1999)

Figure 1.1 displays an example of train unit diagrams. Train unit diagrams are the printed form of train schedules, which mainly include the wheel turning information about the arrival and departure times of a vehicle unit at individual stops within the operation. It should be noted that, although wheel turning work forms the major part of a driver's schedule, some non-wheel turning work not shown in Figure 1.1 also needs to be done by a driver.

1.1.2 Definitions

In this section, some important concepts will be introduced. Figure 1.2 is an example of a vehicle block, which timelines the vehicle work. A *Relief Opportunity* is a time and place where a driver can leave the current vehicle, for reasons such as taking a meal break, or transferring to another vehicle. The work between two consecutive relief opportunities on the same vehicle is called a *piece of work*.

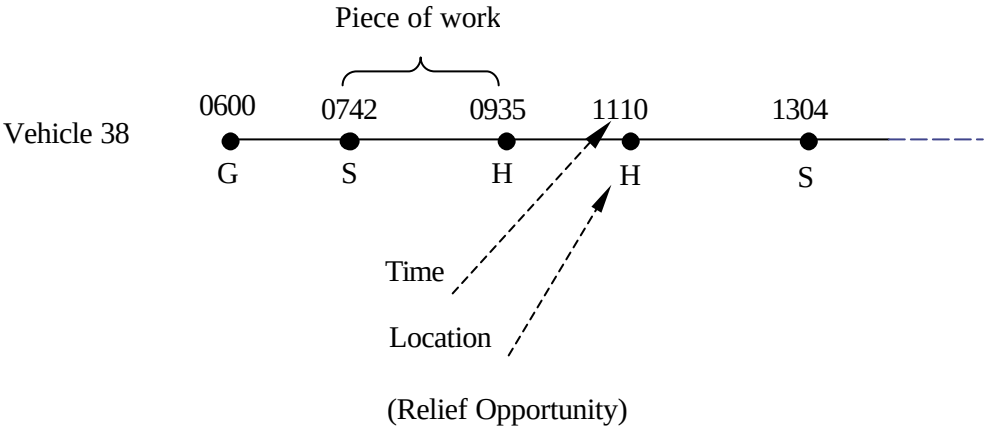


Figure 1.2: An example of a vehicle block

The work a single driver carries out in one day is called *a shift*, which consists of several *spells* of work. *A spell* contains a number of consecutive pieces of work on the same vehicle, and *a driver schedule* is a solution that uses a set of shifts to cover all the required driver work. To make it clearer, Figure 1.3 gives an example of driver shifts, which contains a 2-

spell shift and a 3-spell shift. From this figure, it is obvious that different partitions of vehicles will result in different shifts.

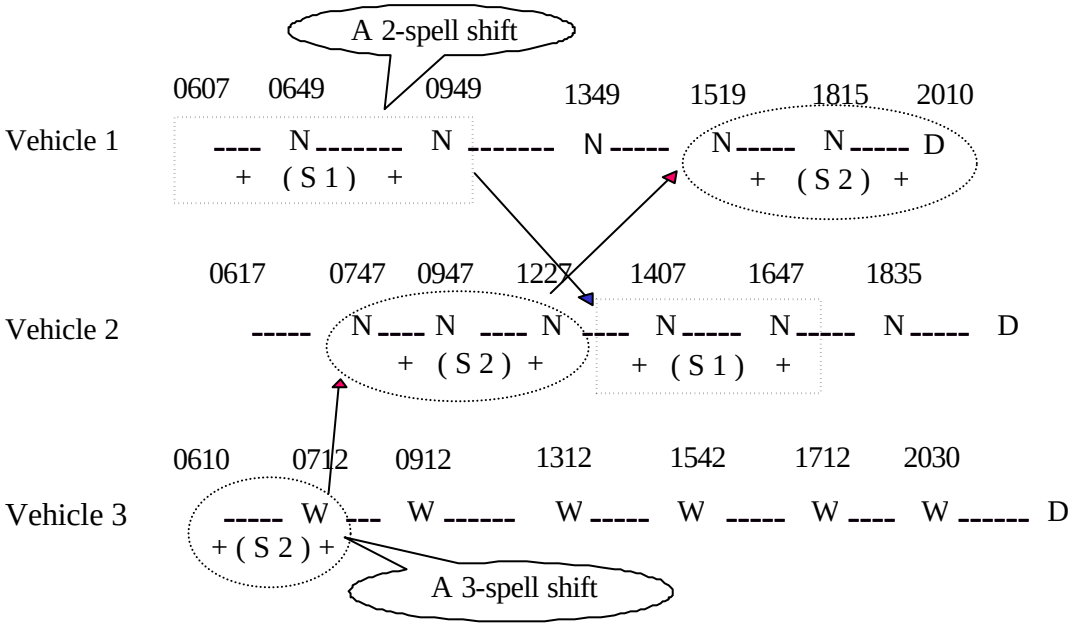


Figure 1.3: An example of driver shifts

1.1.3 Labour agreement rules

The rules for the construction of shifts are mostly provided by past practice and local conditions, and are agreed between the transport operators and the union as internal regulations, while others are statutory. Some rules are hard rules that define the Labour Agreement between management and unions. Other rules are soft rules that are used to prevent the system forming a prohibitively larger number of possible shifts.

Most rules are relevant to transport operators in general even though their parameters may differ. However, individual operators may have their own extra rules. The following is a list of some global rules typically used by different operators:

- Maximum total working time;

- Minimum time allowance for signing on and off at the depot;
- The total spreadover (normally the paid hours for a driver from sign on to sign off) range;
- The minimum length of a meal break, which is normally a fixed time for meal plus travel time to the canteen;
- The maximum time for a driver to work without a meal break, which is usually four to five hours and may contain work on more than one vehicle with a join up required for the driver to change vehicle.

1.1.4 Shift types

There are two main types of shift: *straight* shifts of two stretches separated by a meal break of thirty to sixty minutes, and *split* shifts with a spreadover of maybe about twelve hours containing two stretches separated by a long break of several hours. A stretch is the period from the start of a shift to the start of meal break, or from the end of meal break to the end of shift.

According to which time period of the day shifts cover, *straight* shifts can be further divided into the following three types: early shifts that take early buses/trains out of depot and cover part of the morning peak, late shifts that take late buses/trains into the depot and cover part of the afternoon peak, and middle shifts. Normally, straight shifts in UK contain two stretches separated by a meal break, although some may also contain two or more meal breaks or even a straight run without any meal break. However, straight bus runs without meal break are common in North America.

1.2 Research relevance

Driver scheduling is a worldwide problem. Since efficient schedules can make significant monetary savings for the transportation operators, this problem has attracted much research interest over the past forty years. Although a great deal of progress has been made, there are still several aspects that need to be further investigated and improved. Any advances in the techniques available for solving such problems are highly significant.

A much researched approach for the driver scheduling problem is "generate-and-select", in which a large set of feasible shifts are generated first, and then a subset is selected, from the large set, to form a final schedule by mathematical programming. In the selection stage, computational difficulties exist because of the NP-hard nature of this combinatorial optimisation problem (Kwan 1999).

The Integer Linear Programming process in the selection stage might fail to produce an integer solution for very large size problems. This usually happens when the branch and bound process automatically stops the tree search because a stipulated number of nodes have been explored and the search tree is getting too large to handle practically. Therefore, the following aspects still need to be improved.

1) Optimising ability: to converge to an optimal or near optimal integer solution;
2) Executing speed: even with today's computer power, Integer Linear Programming can be still time consuming;
3) Problem size: there is a limit on the problem size to be handled by the Integer Linear Programming process.

This research attempts to improve the robustness of the selection process to produce final solutions, by investigating possible methods to replace the branch and bound process, or even

the whole mathematical programming method. Since the selection process is not problem-specific, using evolutionary algorithms as alternatives to explore the search space and select a subset as the final solution seems attractive.

Genetic Algorithms simulate natural evolution by maintaining a group of solutions and adding new solutions by the crossover and mutation operators. They are useful approaches to problems requiring an efficient search over a large solution space, and are particularly suitable for obtaining approximate solutions for multivariable optimisation problems where mathematical analyses are difficult. On the other hand, Simulated Evolution algorithms mimic the evolutionary process on a single solution. Each element of the solution in each generation must constantly prove its functionality under the current conditions in order to remain unaltered. The purpose of this process is to create gradually stable structures that are perfectly adapted to the given constraints.

In this research, the use of a Genetic Algorithm and a Simulated Evolution algorithm will be investigated, tried and critically assessed.

1.3 Structure of the thesis

Following this introduction chapter, Chapter 2 gives a literature review about some computerised approaches to the driver scheduling problem and the set covering problem. The remaining chapters in this thesis present the research in stages by investigating and building a suitable approach to solve the driver scheduling problem, and finally generalize this approach for the set covering problem.

Chapter 3 presents a refined greedy algorithm to evaluate all the potential shifts based on fuzzy subsets theory, and to decide which shift is going to be selected in the process of constructing a schedule. The main idea is to set up five criteria, characterized by fuzzy

membership functions, to evaluate the structure and generally the goodness of a shift. Since each criterion reflects only one aspect of the shift structure, an overall evaluation could be made by aggregation of all the criteria. Three different kinds of aggregation operator have been investigated, and the most suitable one will be determined.

Chapter 4 describes a Genetic Algorithm for calibrating the weight distribution amongst the above fuzzified criteria, so that a single-valued weighted evaluation can be computed for each shift. Although driver schedules are constructed as by-products through generations of evolution, they are not expected to be very close to optimum because of the greedy nature of the construction method used.

Chapter 5 designs a novel evolutionary approach to improve the solutions further based on a crude solution by the simple heuristic or a refined solution by the Genetic Algorithm. The evolutionary algorithm combines the features of iterative improvement and constructive perturbation with the ability to avoid getting stuck at local minima. Its framework is a Simulated Evolution algorithm, in which the steps of *Evaluation* and *Reconstruction* have been fuzzified.

Chapter 6 reports on the generalisation from driver scheduling to the class of set covering. A function is first designed to evaluate, under fuzzified criteria, the structure of each column. This function is embedded into the *Construction* step and the *Evaluation* step of an Simulated Evolution algorithm. In each generation an unfit portion of the working solution is removed. The broken solution is then repaired by a greedy algorithm. Taguchi's experimental design is utilized to reliably set the seven parameters involved, with a small number of experiments.

Finally, Chapter 7 gives the conclusions about the achievements of this research and suggests some work for future research.

Chapter Two

Review of Techniques for Driver Scheduling and Set Covering

2.1 Introduction

Bus and train driver scheduling has attracted much interest since the 1960's. Wren and Rousseau (1995) gave an overview of the approaches, many of which have been reported in a series of international workshop conferences (Desrochers and Rousseau 1992; Daduna et al. 1995; Wilson 1999; Voß and Daduna, 2001). Besides a brief review about some heuristics for the general set covering problem, this chapter gives an extensive literature review about the driver scheduling techniques divided into the following three groups: early heuristics, mathematical programming methods, meta-heuristics and others.

With regard to the driver scheduling techniques, Section 2.2.1 reviews some early heuristic systems while Section 2.2.2 reviews some successful systems based on mathematical programming methods. Some recent work on meta-heuristics and others (such as Tabu search, genetic algorithms, Ant system, and constraint satisfaction) are introduced in Section 2.2.3.

Section 2.3 briefly reviews some heuristics for the general set covering problem, including a standard genetic algorithm, a parallel genetic algorithm, an artificial neural network algorithm, a simple Lagrangian heuristic, and a simulated annealing algorithm. A summary is given in Section 2.4.

2.2 Driver scheduling techniques

2.2.1 Early heuristic methods

By the 1980s the early methods for driver scheduling problems were mainly heuristics based. This is because the computer was not powerful enough to run the mathematical solvers, and the techniques in the mathematical solvers were also not as advanced as nowadays. Many of the approaches are first to construct an initial schedule by using a heuristic process, and then attempt to improve this schedule by making limited alterations.

The early heuristics were useful in some applications, since they were customised for individual companies and thus could be fully tailored to meet the specific requirement for individual company. The drawbacks of these approaches are they were not easily portable to other companies and had to be considerably modified to fit new conditions. Furthermore they were not suitable for general optimisation (Wren and Rouseau, 1995). The purely heuristic approaches were subsequently abandonment with the advent of mathematical programming approaches aided by heuristics.

2.2.1.1 TRACS

TRACS (Techniques for Running Automatic Crew Scheduling) was developed at the University of Leeds from 1967 (Parker and Smith, 1981). The system is based on the assumption that a poor initial solution cannot be altered into a good solution by heuristic

improvements, which might be true since meta-heuristics were not available at that time. Therefore, this method took great efforts to produce an initial solution as good as possible. An initial schedule is constructed in the following steps:

1) To form early shifts from the beginning of the bus schedule, leaving sufficient work for the first halves of split shifts to cover the morning peak.
2) To form late and middle shifts at the end of the bus schedule, leaving suitable work for the second halves of split shifts to cover the afternoon peak.
3) To form split shifts by matching the first and second halves.
4) To attach remaining work to existing shifts if possible.

This initial schedule contains two-spell and three-spell shifts. A concept of marked time was used to guide the formation of shifts. For early shifts, a marked time is the latest time by which the first driver of each bus must be relieved. For late shifts, a marked time is the earliest time at which the last driver can start work on each bus.

To minimise the number of drivers during the peaks, meal break chains were attempted to be formed. In this chain, drivers take meal breaks in turn, that is, the first driver finishing his meal break will relieve the second driver who, after his meal break, will take over the bus from the third driver, and so on.

The initial schedule is improved by two sets of procedures. One set attempts to reduce the number of shifts and unallocated pieces of work. Each shift is considered to determine whether the work in it can be contained in other shifts. This procedure redistributes work between shifts so that shifts with long spreadovers are assigned more work. This will make short shifts shorter so that they become redundant and can be removed eventually.

Another set of procedures is for cost reduction. There are several procedures to achieve this target, including swapping or moving stretches of shifts, re-matching first and second stretches of shifts, switching changeovers of a shift to another relief time, and relocating small pieces of work around the middle of the day.

2.2.1.2 Other heuristic methods

RUCUS (RUn CUtting and Scheduling) (Bennington and Rebibio, 1975; Bodin et al., 1985) was developed in the late 1960s. It is also a system that first generates an initial solution and then refines it using heuristics. It first forms 1-spell shifts and then 2-spell shifts, and after this process any unallocated pieces of work are regarded as short overtime spells. This limited the wide use of the system and eventually led to the abandon of RUCUS, since many operators did not use overtime and even if they had to they tried to restrict it. Moreover, it is generally inefficient to leave out “difficult" work in such a way. Once an initial solution has been constructed, the system begins to use local search moves trying to improve this solution. It either exchanges some pieces of work in one shift with pieces of work in another shift, or moves selected relief opportunities forward or backward. A repair procedure is then used to fix any infeasible shifts due to the changes. In case that there are still infeasible shifts left in the final schedule, manual intervention may be needed.

HOT (Hamburg Optimisation Techniques) (Hoffstadt 1981; Daduna and Mojsilovic, 1988) was developed and used by the schedulers at the Hamburger Hochbahn AG since the 1970s. It starts by trying to form good shifts, one at a time, for each morning bus, and then for each evening bus. Any work that cannot be treated in this process is formed into partial shifts, which are then combined into full shifts by a variant of the Hungarian Algorithm (Taha 1997). Little improvement can be achieved to the schedule once it is constructed, and sometimes it may even leave unassigned pieces of work. Although this system has been used in several German bus operations, it is believed that it is no longer in widespread use.

COMPACS (COMPuter Assisted Crew Scheduling) was an interactive system developed in the early 1980's (Wren et al., 1985), and was later incorporated into the BUSMAN scheduling package (Chamberlain and Wren, 1992). It combines the heuristics of TRACS and the interactive features of an early interactive system known as TRICS. One useful feature of COMPACS is that it can give an estimate on the possible number of shifts of each type. The estimate is useful since it can guide the schedulers to build up the schedule interactively. COMPACS can also produce an entire schedule automatically by means of a simplified version of TRACS: it retained the initial solution generation phase of TRACS, but not the improving moves. Hence the solution quality might be poor.

2.2.2 Mathematical programming methods

2.2.2.1 Mathematical model of set covering and set partitioning

The driver scheduling problem can be formulated as a set covering problem and expressed as an Integer Linear Programming (ILP) problem. The basic model is shown below.

n	Number of potential shifts
m	Number of driving work pieces to be covered
c_j	Cost of shift j
a_{ij}	0-1 integer constants: 1 indicates shift j covers work piece i, and 0 otherwise
x_j	0-1 integer variables: 1 indicates shift j is used in the solution, and 0 otherwise

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

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

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

Objective (2.1) minimizes the total cost. Constraint (2.2) ensures that each piece of work is covered by at least one driver, and constraint (2.3) requires that whole shifts be considered. In practice, the model has been extended to cater for other practical objectives and constraints (Fores et al., 1999).

By changing the constraint (2.2) into the following form, it becomes a set partitioning problem:

$$\text{Subject to:} \quad \sum_{j=1}^{n} a_{ij} x_j = 1, \quad i \in \{1, 2, \ldots, m\} \tag{2.4}$$

In the set covering model, each piece of work must be covered by at least one shift, while in a set partitioning model, each piece of work must be covered by exactly one shift. In theory, the solution to the ILP is the schedule with minimum cost. However, the total number of all possible shifts is usually too large to be practically solved by the ILP. The very large shift set therefore has to be reduced to a manageable size by heuristics, or else the problem has to be decomposed into several sub-problems and solved separately.

2.2.2.2 IMPACS

IMPACS (Integer Mathematical Programming for Automatic Crew Scheduling) was developed for bus operation in the late 1970s. Parker and Smith (1981) presented the

prototype and Wren and Smith (1988) gave a full description of the system. This system was installed in London Transport in 1984 and in Greater Manchester Buses in 1985.

IMPACS is based on a set covering model described above: a large number of possible shifts with associated costs are first generated, a subset is then selected to cover all the pieces of work at minimum cost. Since the number of variables and constraints may be too enormous to be handled for the computer power at that time, IMPACS employs a number of heuristics to reduce the number of variables and constraints. The reduction processes are described as follows:

1) Reduce the number of pieces of work by heuristics. This method is based on the idea of de-selecting some relief opportunities that are unlikely to be used by any shifts. If a relief opportunity is not used as a changeover time, it can be omitted: the pieces of work at either side of the relief can be combined into one and thus the number of pieces of work is reduced.

2) Generate a reasonable set of legal shifts by heuristics. This is done by using 'hard' rules defined by the labour agreement rules and some 'soft' rules to restrict the number of shifts produced.

3) Reduce the number of shifts by heuristics. Some shifts are regarded less important than the others in forming an efficient schedule, e.g. the shift that has been included in another shift but has a larger cost, and the shift whose every piece of work is also covered by at least a specified number of other shifts. These shifts are removed from the large set for further reduction.

4) Select a subset of shifts using ILP. IMPACS solves this set covering problem by first obtaining a relaxed LP solution and then finding an integer solution using the

branch-and-bound technique, which searches for integer solutions through a tree structure. The search terminates after a satisfied solution found or a certain number of nodes reached.

5) Improve the driver schedule by heuristic. The best integer solution found can be further improved by heuristic procedures similar to those used in early heuristic systems. Before using these procedures, all the relief opportunities de-selected previously need to be restored.

In order to solve large problems, IMPACS also provides a decomposition module although a risk of suboptimal solution may be caused. After decomposition, the first subproblem is solved. The inefficient shifts in the first subproblem are carried forward to the next sub-problem, and so on. Schedules of all sub-problems are finally combined into one schedule as a whole.

Although originally developed for bus operations, great efforts have been attempted to adapt IMPACS to the train driver scheduling problem (Parker et al., 1995). Since train operations are more complicated and also its problem size is usually much larger, the adaptation has proven difficult.

2.2.2.3 TRACS II

TRACS II (Techniques for Running Automatic Crew Scheduling, Mark II) is a new generation of driver scheduling software developed from 1994. It is being used on behalf of many train companies (including First Group, the largest bus company in the UK). TRACS II has shown significant savings when compared with standard methods for scheduling drivers of railways and buses, and has helped to negotiate more flexible working rules with the trade unions.

TRACS II follows almost the same approach as IMPACS, but the components have been considerably redesigned to cope with the complexity of rail operations and to incorporate new algorithmic advances. Basically, it consists of seven modules (i.e., VALIDATE, TRAVEL, BUILD, SIEVE, SCHEDULE, REDUCE, and DISPLAY) in the following three phases.

1) The first phase is for shift generation. VALIDATE is run first to check validity of the data set. Then TRAVEL compiles a list of all possible opportunities for drivers' passenger travel between relief points. BUILD finally generates a large set of possible legal shifts, based on a set of parameters to define the legality of a shift and to avoid producing an excessive number of shifts. The heuristics in BUILD are more complicated than those of IMPACS.

2) The second phase is for refinement of the shift sets. SIEVE is used to reduce the shift set: it ranks the shifts generated by BUILD and eliminates some of the less efficient shifts formed. Recently, SIEVE is often only used to remove duplicated shifts because of the use of a column generation technique, which allows larger shift sets to be handled. MERGE is used to combine different potential shift sets that have passed through the SIEVE process, in the case that BUILD has been run several times, e.g. there is a need for a second run to cover work that has previously been left.

3) The third phase is for shift selection using ILP. In this phase, SCHEDULE also applies the set covering ILP model to select the most economical subset from a large shift set. SCHEDULE is originated from the ILP process used in IMPACS. However, TRACS II can run any size of problem, while IMPACS could handle up to 30,000 input shifts.

SCHEDULE has two main choices of optimisation procedures: the dual steepest edge approach and the primal column generation approach. To lessen the effect of degeneracy in set covering problem, a dual steepest edge approach was implemented by Willers et al. (1995), showing a significant reduction in executing time for the relaxed LP solution over the primal simplex approach used in earlier versions of TRACS II. Fores et al. (2001) suggested that this approach be executed on problems containing no more than 30,000 shifts, where all shifts can be explicitly considered within the limits imposed for storage and efficiency. To run any size of problem, the primal column generation approach was introduced in Fores et al. (1999; 2001). It selects an initial subset of the shifts from the submitted set in the beginning, and finds the optimal solution of the subset using a revised simplex method. A certain number of new shifts with favourable reduced costs are then added into the subset, and the new set is re-optimised. When no more shifts can be added into the subset to improve the objective function, the relaxed LP solution is the overall optimal solution.

Although TRACS II is successful, there are still computational difficulties for its ILP method (Kwan 1999). One inherent problem is that the branch and bound algorithm does not always find an integer solution. Referring to the stopping conditions, two different situations may happen:

1) The search terminates after all nodes of the search tree have been fathomed. This is usually caused by the side constraints operating during the branch and bound searches and the reduction heuristic excluding some vital shifts before the branch and bound process.

2) The search terminates after one of the limits of the number of created nodes or active nodes has been reached. This is the most common situation, in which there may be some integer solutions but the search terminates before reaching it. This usually

implies that their objective values may be much higher than the relaxed optimum, and the node selection strategy makes them hard to find. The solution is to take a number of actions, such as generating a different shift set, reducing the size of the shift set input to the ILP, increasing the target of total number of shifts, and imposing side constraints. Any of the above actions requires the ILP to be rerun. For some problems, to find an integer solution needs many iterations, which can be time-consuming.

Another computational difficulty of the ILP method is about the problem size handled. Even employing the column generation technique, there is still a limit on the problem size that the ILP can handle in practice. In the case that a problem is too large to be solved as a single problem (e.g. Northern Spirit and ScotRail), the decomposition may risk losing optimality.

2.2.2.4 Other mathematical programming methods

HASTUS (Lessard et al., 1981; Rousseau and Blais, 1985; Blais and Rousseau, 1988) is a widely used commercial package with graphical user interface to deal with driver scheduling, vehicle scheduling, and rostering. The HASTUS driver scheduling component is divided into two parts, HASTUS-micro and HASTUS-macro. HASTUS-macro constructs an initial schedule and HASTUS-micro produces the final schedule. HASTUS-macro uses linear programming to generate a pseudo-schedule that provides an estimate of the number of drivers needed. The pseudo-schedule is built by pseudo-shifts, which are generated using simplified relief opportunities by simply cutting the day into user-defined time slots. Then HASTUS-micro uses the pseudo-schedule to create a final schedule, by producing real shifts that relate to those in the HASTUS-macro solution as close as possible.

EXPRESS (Falkner and Ryan, 1992) is a bus driver scheduling system based on a set partition model specially developed for Christchurch Transport, New Zealand. Its earlier version and a

study of the use of set partition are presented in (Falkner and Ryan, 1988). During the search process, the strictness of the model is diminished by the addition of slack variables. It then uses a version of the original ZIP similar to those being used in IMPACS and TRACS II. However, its branching model is slightly different. In this system, the branch and bound algorithm branches on the pieces of work (constraint branching) rather than the relief opportunities.

2.2.3 Meta-heuristic methods and others

Although TRACS II works well, its solution is still not necessarily optimal, particularly when large problems have to be decomposed. New research is therefore directed at finding new ways to handle driver scheduling. Some approaches such as Tabu search, genetic algorithms, ant system, and constraint programming have been designed, aiming to tackle part or all of the problem. This research uses a mixture of artificial intelligence and operational research methods to enhance or replace TRACS II.

2.2.3.1 Tabu search

Tabu search is proposed by Glover (1989, 1990) for solving hard combinatorial problems. The basic idea is to escape from local optima by allowing the acceptance of non-improved solutions. Basically, Tabu search is an iterative technique that moves step by step from an initial solution towards a solution close to the global optimum.

Cavique et al. (1999) presented a Tabu search called Run-cutting for crew scheduling. Starting with an initial solution produced by an approach similar to that used in TRACS, the method only allows two-spell shifts or even less efficient one-spell shifts. Tabu search is embedded in the improvement phase to reduce the number of shifts. This algorithm iteratively removes some inefficient shifts and sometimes their adjacent shifts from the current solution,

and then applies the re-cutting algorithm to construct shifts to repair the broken schedule. Tabu search is used to ensure that pieces of work that appear frequently in one-spell shifts are given higher priority to be incorporated into two-spell shifts and thus become more likely to be efficient. The Tabu search approach was found very efficient at improving the initial solution after the first few iterations, but then found it difficult to make further improvements. The reason might be that they only concentrated on inefficient shifts and sometimes an efficient shift needs to be changed to make the leap to a really efficient schedule. They also presented another Tabu search algorithm called Run-ejection, which applies a matching technique that performs better. The reason might be that it expands the search, not only changing inefficient shifts.

There are several special features in this operation. These algorithms only construct one-spell and two-spell shifts. The complexity will increase significantly if more spells are allowed in shifts. Shift costs are not considered, and thus the objective is only to minimise the number of shifts. Since these algorithms were developed for the Lisbon Underground, it would be difficult to adapt to other bus or rail operations.

Shen and Kwan (2001) developed HACS, which also used a Tabu search for the driver scheduling problem. The HACS approach is based on a representation of the problem involving sequences of links. The links and its associated active relief opportunities compose a solution space. A significant feature of HACS is that infeasible intermediate solutions are allowed, thus allowing more chances to escape from local optima. To get rid of infeasibility and fulfil the objectives they applied four neighbourhood structures in sequence: swapping two links, swapping two spells, inserting one spell, and recutting blocks. The first three concentrate on refinement of links with fixed relief opportunities, while the last one considers variable active relief opportunities while links are reconstructed. HACS has been extended to handle windows of relief opportunities in (Shen 2001), where a time range replaces a relief time in a relief opportunity.

HACS starts from a rough initial solution, and can deal with complex problems by simply adjusting the cost function and the penalty function to the rules stipulated in specific problems. Compared with the results of TRACS II, its solution quality is slightly worse.

2.2.3.2 GAs

GAs are general-purpose search and optimisation methods originated from Holland (1975) and developed subsequently to solve a wide range of real-world problems (Goldberg 1989). These algorithms are based on the mechanics of genetics and natural selection, and represent the search space as a coded population of potential solutions. The population is then manipulated according to the survival of the fittest principle, providing good practical solutions.

A number of GAs have been developed for the driver scheduling problem, among which the GA in Kwan et al. (2001) performs best. The role of this GA is to derive a small selection of good shifts to seed a greedy schedule repair technique. These good shifts can be preferred shifts, whose fractional values in the relaxed LP solution generated by TRACS II are higher than a pre-defined constant such as 0.2. The reason why only these shifts are represented is that empirical evidence in Kwan (1999) has shown that at least 50% and on average 74% of the shifts in the final TRACS II solution were in the LP solution. The schedule constructed is fed back to the GA for fitness evaluation. At the same time, the group of shifts (called 'a Relief Chain') is identified and recorded as a learning property of the corresponding member in the population. That learning property will be inherited by the offspring and used to enlarge the set of seeding shifts when the GA interfaces with the schedule construction heuristic.

The new approach has been extensively tested using real data sets, some of which are very large problem instances. This method has produced schedules that are comparable to solutions found by ILP, and are generally better than those compiled by experienced schedulers.

2.2.3.3 Ant system

Forsyth and Wren (1997) applied an optimisation method called the ant system to produce driver schedules. The ant system was developed by Dorigo et al (1995) based on behaviour of ants searching for food, which can be modelled into a search algorithm as follows. The basic idea is that, when ants move, they leave pheromone trails that can be detected by other ants and slowly evaporate over time.

In the beginning, the ants depart from a nest in random directions. Once food is found, the ants are most likely to return to the nest along their own pheromone trail, thus strengthening the trail. Since ants have an in-built bias towards following strong pheromone trails, subsequently more ants are likely to follow the shortest path, strengthening the trail even more. Although a number of paths exist between the food and the nest due to the randomness in the ants' movements, the pheromone trails in the shortest path will become strongest, since ants that follow the path are likely to return back soonest.

The ant system for driver scheduling uses TRACS II to generate potential shifts. Each ant will create a solution at each iteration. A heuristic is used to select relief opportunities, and then the ant chooses a shift from the set that start at that relief opportunity. The process repeats until all the work is covered. As the system progresses, the good combinations of shifts are more likely to be followed and so over time the solutions are continuously improved. Unfortunately, this method does not produce results comparable to the TRACS II solutions.

2.2.3.4 Constraint satisfaction

The constraint satisfaction approach (Tsang 1993) is an emergent technology for declarative description and effective solving of large, particularly combinatorial, problems especially in areas of planning and scheduling. It provides a powerful and easy system for modelling restrictions and using these restrictions to search for a solution.

Layfield et al. (1999) used constraint programming to produce a component that is similar to SELECT in IMPACS and could be slotted into TRACS II. Its purpose is to remove relief opportunities that are unlikely to be used in good schedules, thus reducing the problem size. The program first produces the morning part of the schedule simulating the manual scheduling process. It puts a limit on the number of spells that each bus can be broken up into, to prevent too short shifts being produced. A morning schedule is constructed by using randomised heuristics to build the partial schedule one shift at a time. Several morning schedules are constructed, and the relief opportunities that are not used in these schedules are removed. This program can also be used to produce the evening part of a schedule. The process has speeded up TRACS II in several cases, but its solution cost is mostly slightly higher.

Curtis (2000) used constraints to reduce the search space of a set partitioning model, based on previous work on air crew scheduling (Guerinik and Caneghem, 1995; Rodosek et al., 1996). He modelled the bus driver scheduling problem as a constraint satisfaction problem, where variables were defined as pieces of work and the domain of each variable was the set of indices of the shifts that covered the piece of work. This work uses the large set of potential shifts produced by of TRACS II. The relaxed LP solution in TRACS II is applied to guide the 'variable ordering'. Test results from small problems are comparable to those of TRACS II. Curtis also described an iterative repair process to construct driver schedules, in which a local

search method called GENET was applied to solve the problem modelled as a constraint satisfaction problem.

2.3 Heuristics for general set covering problems

Set covering problems are difficult zero-one optimization problems, which have been proven to be NP-complete (Garey and Johnson, 1979). They are often encountered in a wide area of applications such as resource allocation (Revelle et al., 1970), crew scheduling (Rubin 1973; Smith and Wren, 1988), location of emergency service (Toregas et al., 1971), assembly line balancing (Salveson 1955), and simplification of Boolean expressions (Breuer 1970).

Besides the exact algorithms (Beasley 1987; Fisher and Kedia, 1990; Beasley and Jornstern, 1992), there is an abundant literature dealing with heuristics, some of which are discussed in the following sections.

2.3.1 A GA by Beasley and Chu

Beasley and Chu (1996) used a GA for non-unicost set covering problems. In its chromosome presentation, each gene position denotes one of the columns in the zero-one matrix, and has a value of 1 or 0 depending on whether the variable is or is not present in the solution. A crossover operator called ‘fusion’ is designed to combine two parent strings: the choice of whose gene values are passed to the child is made based on the relative fitness of the two parents. For the mutation operator, they applied a variable mutation rate. At the early stage of the GA, the mutation rate is set to be lower to allow minimal disruption. As the GA progresses, the mutation rate increases since the crossover operator becomes less effective. When the GA finally converges, the mutation rate will stay at a constant rate.

The solutions generated by the crossover and mutation operators may be infeasible, i.e. some rows are not covered. To repair these infeasible solutions, Beasley and Chu presented a heuristic that could not only maintain the feasibility of the solution, but also provide an additional local optimisation step to make the GA more efficient.

They tested the performance of the approach on a large set of randomly generated problems. Computational results showed that this heuristic can produce optimal solutions for small-size problems and high-quality solutions for large-size problems.

2.3.2 An Artificial Neural Network algorithm by Ohlsson et al.

Artificial Neural Network (ANN) has attracted much research during the past decades. Most of the activities involve feed-forward architectures for pattern recognition or function approximation. However, ANN can also be used for difficult combinatorial optimisation problems. This is usually done by first mapping the problem onto an energy function, and then finding configurations with low energy function values by the method of iterating some mean field equations.

Ohlsson et al. (2001) developed a mean field feedback ANN algorithm for the set covering problem. They used a multilinear penalty function to obtain a convenient encoding of the inequality constraints. An approximate energy minimum is achieved by iterating a set of mean field equations, in combination with annealing. In contrast to most existing search and heuristics techniques, this ANN model is not based on exploratory search to find the optimal configuration. Rather, the neural units of ANN find their way in a fuzzy manner through an interpolating and continuous space towards good solutions.

This algorithm has been tested against some very large-scale problems (up to 5000 rows and 10^6 columns). Computational results shows that this approach can produce results typically within a few percent from the optimal solutions, and its executing speed is extremely fast.

2.3.3 A sophisticated GA by Solar et al.

During recent years, parallel GAs have been used to discover how the interchange of genetic information for separated populations affects the final solution. Normally a parallel GA is implemented based on an "Island Model" where separate and isolated sub-populations evolve independently and in parallel. Fit members occasionally migrate between sub-populations, allowing the distribution and sharing of good genetic material of fit members and helping to maintain genetic diversity. The exploration of different solution spaces could optimise the search in terms of both computational time and solution quality.

Solar et al. (2002) presented a parallel GA model to solve the set covering problem. The chromosome representation used is the traditional one: a bit string with n bits where n is the number of columns in the problem. Since new chromosomes generated by genetic operators could violate some problem constraints, solution representations do not always ensure their feasibility. Therefore a feasibility operator is designed to repair all infeasible solutions

They proposed the following population scheme: independent populations are associated with nodes. Each node executes a single GA and creates a new local population. When all nodes are ready with new generations, each node sends the best local individual to the master node. The master node then selects the best individual received, and broadcasts it to all slave nodes. Each independent slave node replaces the worst local individual with the new best global received. In other words, the interchange of information between parallel searches is the selection of the best global, which replaces the worst of each node.

The parallel GA has been tested by using ten problems up to 500 rows and 5000 columns. The final solutions obtained are not very satisfactory: the percentage deviations are in the range from 3.3% to 10%, and only one optimal value was achieved once in more than 1000 experiments.

2.3.4 A Lagrangian heuristic by Caprara et al.

A number of attempts have been made by using the Lagrangian-based heuristics for the set covering problem (Beasley 1990; Haddadi 1997; Caprara et al., 1999). The more recent work of Caprara et al. will be introduced herein, which consists of three phases of subgradient optimisation, heuristic, and column fixing.

The subgradient phase is to find a near-optimal Lagrangian multiplier vector quickly, by means of an aggressive policy. The heuristic phase is to generate a sequence of near-optimal multiplier vectors, and for each vector compute a heuristic solution. The column fixing phase is to select a subset of "good" columns, and fix to 1 the corresponding variables. In this way an instance with a reduced number of columns and rows is obtained, on which the three-phase procedure is executed iteratively until the solution cannot be improved.

The algorithm was extensively tested on very large size problems, involving up to 5,000 rows and 1,000,000 columns. In 92 out of the 94 test instances, the optimal or the best known solutions can be found quickly. Furthermore, among the 18 instances that the optima are unknown, in 6 cases their solutions are better than the previous best known solutions.

2.3.5 A simulated annealing approach by Sen

Simulated annealing is a stochastic optimization technique based on an analogy from statistical mechanics, where a substance is reduced to its lowest energy configuration by a

sequence of steps that involve alternate heating and cooling. Sen (1993) used a simulated annealing for the set covering problem, which consists of the following four steps:

1) Encode the points in the solution space by using an n-bit string that represents the n columns. A value of 1 in the j-th string position means that the column j is chosen to be in the cover.
2) Formulate an evaluation function that evaluates the goodness of a point in the solution space.
3) Design a set of moves that can be used to alter the points in the solution space. Three types of moves are defined in the annealing scheme: the first and second involves either adding or removing a column from the chosen cover by flipping a randomly picked bit; and the third involved replacing one column with another in the chosen cover by interchanging the values of two bit positions with different values.
4) Decide an annealing schedule, such as the setting of starting temperature, rule for temperature decrements, and the stopping criteria to halt the algorithm.

This approach has only been implemented on some small size problems with good results. For large size problems, its performance would be difficult to estimate.

2.4 Summary

This chapter reviews the driver scheduling approaches, which can be mainly divided into two groups: constructive approach and generate-and-select approach. This chapter also gives a brief introduction to some heuristic approaches for the general set covering problem.

The constructive approach for driver scheduling does not need artificial rules or explicit reduction in problem size. The early heuristics are based on the constructive approach. They relied on good initial schedules constructed based on human schedulers' knowledge (e.g.

TRACS and RUCUS), and try to improve this initial schedule by simple refinement. Therefore the early constructive heuristics are generally difficult to adapt to new situations, and extensive final manual adjustments to the schedule are needed. Furthermore, the solution quality cannot be guaranteed. The application of Tabu search with windows of relief opportunities described by Shen and Kwan (2001) has exploited more powerful modern meta-heuristics, and is less dependent on human schedulers' knowledge.

The generate-and-select approach, which generates a large set of potential shifts and then selects an efficient subset, is currently the most successful for bus and train driver scheduling. The generation phase is problem-oriented, while the selection phase is algorithm-oriented. This makes the approach adaptable to different conditions through the generation phase. The algorithms for selection can be mathematical programming, genetic algorithms, ant system, and constraint programming, among which the mathematical programming (i.e. in TRACS II) performs best. Unfortunately, the number of potential shifts is usually too enormous to enable the selection algorithms (even for TRACS II) to find an optimal solution within reasonable time. Hence there is room for improvement to this approach.

The research presented in this thesis attempts to improve the selection phase for driver scheduling, by investigating two evolutionary algorithms, namely a GA and a Simulated Evolution algorithm, which will be described from Chapter 3 to Chapter 5. Chapter 6 will report on the generalisation of the Simulated Evolution algorithm from driver scheduling to the class of set covering problems.

Chapter Three

A Fuzzy Theory Based Greedy Heuristic for Driver Scheduling

3.1 Introduction

Bus and rail driver scheduling can be formulated as a set covering Integer Linear Programme (ILP). All the legal potential shifts are first constructed. Then, a least cost subset covering all the work is selected to form a solution schedule. A typical problem may have a solution schedule requiring over 100 shifts chosen from a potential set of up to 50,000.

Set covering is one of the oldest and most studied NP-hard problems (Karp 1972; Johnson 1974; Lovász 1975; Chvátal 1979). Given a ground set U of m elements, and a weight for each set, the goal is to cover U with the smallest possible number of sets. In the case of driver scheduling, there is the additional objective of minimising the total weight.

Since the set covering problem is unlikely to be solved optimally in polynomial time, there has been a lot of work in exploring the possibility of obtaining efficiently near-optimal

solutions. For example, the greedy algorithm repeatedly chooses the unused set that covers the largest number of remaining elements. In this research, a more comprehensive evaluation of the sets (potential shifts) that uses the product of over-cover penalty and structural coefficient has been developed to decide which shift is going to be selected in the process of constructing a schedule. Much of the work in this chapter has been introduced in (Kwan et al., 2000a; Li and Kwan, 2000, 2001,2001a, 2001b and 2002a).

The new polynomial time algorithm for driver scheduling evaluates all potential shifts based on fuzzy subsets theory, a means of presenting uncertain information put forward by Zadeh (1965), and developed subsequently to solve real-world complex problems (Kaufmann 1975; Dubois and Prade, 1980; Klir and Yuan, 1995). The main idea of the proposed approach is to set up five criteria, characterized by fuzzy membership functions, to evaluate the structure and generally the goodness of a shift. Since each criterion reflects only one aspect with regard to the shift structure respectively, an overall evaluation could be made by aggregation of all the criteria. Three different kinds of fuzzy aggregation operator, namely intersection operator, union operator, and arithmetic mean operator, will be investigated in this decision-making, and the one most suitable for driver scheduling will be determined.

The rest of this chapter is organized as follows. Section 3.2 introduces some fundamental notions in fuzzy theory, such as crisp sets, fuzzy subsets, and general aggregation operators. Section 3.3 presents a refined greedy heuristic for driver scheduling. Section 3.4 describes the method of fuzzy evaluation for shift structure in detail, including the construction of the five fuzzified criteria and the evaluation process by aggregation operators in different categories. Computational results for the determination of the greedy operators in over-cover penalty and the aggregation operator in structural coefficient are reported in Section 3.5. Finally, conclusions are discussed in Section 3.6.

3.2 Fundamental notions of Fuzzy Subsets

Fuzzy knowledge, i.e. knowledge that is uncertain, vague, inexact, ambiguous, inaccurate, or probabilistic in nature, can be frequently encountered in the real world. Thinking and reasoning of human beings often involve such fuzzy information, possibly originating from inherently inexact human concepts and the matching of similar rather than identical experiences. In systems based on classical set theory and two-valued logic, it is very difficult to answer such questions that do not have completely true answers in many circumstances. However, by using their fuzzy knowledge, humans can usually give satisfactory answers, which are probably true.

Naturally a question arises: how can the fuzzy knowledge be represented? Before the introduction of fuzzy subsets, we would first give a brief review about the notions of crisp sets.

3.2.1 Crisp Sets

Let **E** be a set and **A** be a subset of **E**, denoted as

$$\mathbf{A} \subset \mathbf{E}. \tag{3.1}$$

We usually indicate that an element x of **E** is a member of **A** using the symbol $\in$:

$$x \in \mathbf{A}. \tag{3.2}$$

To present this membership we may also use another concept, a characteristic function $m_A(x)$, whose value indicates whether x is a member of **A** (yes or no):

$$m_A(x) = \begin{cases} 1, & \text{if } x \in A; \\ 0, & \text{if } x \notin A. \end{cases} \tag{3.3}$$

Thus, the membership function for a crisp set **A** is defined as

$$m_A : X \to \{0,1\}. \tag{3.4}$$

Similarly, given two subsets **A** and **B**, we can define the membership functions for the set operations (such as union, intersection and complement) as follows:

$$m_{A\cup B}(x) = \max(m_A(x), m_B(x)), \tag{3.5}$$

$$m_{A\cap B}(x) = \min(m_A(x), m_B(x)), \tag{3.6}$$

$$m_{\bar{A}}(x) = 1 - m_A(x). \tag{3.7}$$

Furthermore, **A** is a subset of **B** if and only if

$$x \in \mathbf{A} \Rightarrow x \in \mathbf{B},\ \forall x. \tag{3.8}$$

In terms of membership function, **A** is a subset of **B** if and only if

$$m_A(x) \le m_B(x), \forall x. \tag{3.9}$$

[**Example]** Consider a finite set with five elements:

$$\mathbf{E} = \{x_1, x_2, x_3, x_4, x_5\}, \tag{3.10}$$

and let

$$\mathbf{A} = \{x_1, x_3, x_5\}. \tag{3.11}$$

One writes

$$m_A(x_1) = 1,\ m_A(x_2) = 0,\ m_A(x_3) = 1,\ m_A(x_4) = 0,\ m_A(x_5) = 1. \tag{3.12}$$

Therefore we may also express **A** by accompanying the elements of **E** with their characteristic function values:

$$\mathbf{A} = \{(x_1,1),(x_2,0),(x_3,1),(x_4,0),(x_5,1)\}. \tag{3.13}$$

3.2.2 Fuzzy Subsets

Fuzziness exists when the boundary of a piece of information is not clear-cut. For example, words such as tall, young, good, or fat are fuzzy. There is no single quantitative value that defines the term tall when describing a fuzzy concept (or fuzzy variable) such as the tallness of adults. For some people, a height of 175cm is tall, and for others, a height of 185cm is regarded as tall. The concept tall has no clean boundary. A height of 200cm is definitely tall and a height of 100cm is definitely not tall. However, height of 175 has some possibility of being tall, which depends on the context being considered. In fact, a height can have some possibility of being tall and also some possibility of being short. It should be noted that these are not probabilities because the sum of all the possibilities does not need to be 1.0.

The representation of this kind of information is based on the concept of fuzzy subsets. Unlike in classical set theory where one deals with objects whose membership in a set can be clearly described, in fuzzy subsets theory, membership of an element in a set can be partial, i.e. the element belongs to a set with a certain grade (possibility) of membership.

Let us come back to the example in Section 3.2.1. Considering the subset **A** of **E** defined by (3.13), the five elements of **E** belong or do not belong to **A**, and the characteristic function takes the value of either 0 or 1.

Imagine now that this characteristic function may take any value in the interval [0,1]. Thus, an element x_i of **E** might be a member of **A** ($m_A = 1$), could be a strong member of **A** (m_A near 1), may more or less be a member of **A** (m_A neither too near 0 nor too near 1), could be a member of **A** a little (m_A near 0), or finally may not be a member of **A** ($m_A = 0$). By this method the concept of membership takes on an extension and leads to very useful developments.

The mathematical concept can be defined by the following expression:

$$\tilde{A} = \{(x_1, 0.9), (x_2, 0), (x_3, 0.4), (x_4, 0.1), (x_5, 1)\}, \tag{3.14}$$

where x_i is an element of the reference set **E** and its associated number is the value of the characteristic function of the element. This mathematical concept is called a fuzzy subset of **E**, denoted as

$$\tilde{A} \subset \mathbf{E}. \tag{3.15}$$

Thus the fuzzy subset defined by (3.14) contains a large part of x_1, does not contain x_2, contains a little more x_3, contains a little x_4, and contains x_5 completely. This enables us to construct a mathematical model, by which one can handle concepts that are not precisely defined but whose membership in a subset is somewhat qualitative. Hence one may consider: in the set of men, the fuzzy subset of very tall men; in the set of decisions, the fuzzy subset of good decisions; in the set of colours, the fuzzy subset of deep blue colours; and so on.

A rigorous definition of the fuzzy subsets is given by Zadeh [1965]:

> Let **E** be a set, denumerable or not, and let *x* be an element of **E**. Then a fuzzy subset $\tilde{A}$ of **E** is a set of ordered pairs
>
> $$\{(x, m_{\tilde{A}}(x))\}, \forall x \in \mathbf{E}, \tag{3.16}$$
>
> where $m_{\tilde{A}}(x)$ is the grade or degree of membership of *x* in $\tilde{A}$. If $m_{\tilde{A}}(x)$ takes its values in a set **M**, called the membership set, one may say that *x* takes its values in **M** through the function $m_{\tilde{A}}(x)$. Thus, the membership function for a fuzzy subset **A** is defined as
>
> $$m_{\tilde{A}} : X \rightarrow \mathbf{M}. \tag{3.17}$$

Normally, **M** is an interval of [0,1], where 1 is used to represent complete membership, 0 is used to represent complete non-membership, and all the values in between are used to represent intermediate degrees of membership. Furthermore, the membership functions for the fuzzy subset operations are defined in the same forms as in a crisp set.

There are many commonly used membership function types, which are built mainly from the following basic functions: piecewise linear functions (such as triangular and trapezoidal membership functions), the bell-shaped Gaussian distribution function, the sigmoid curve, and quadratic and cubic polynomial curves (such as the S-shaped and Z-shaped curve membership functions). For special requirements, one may also establish the problem-specific membership functions.

Summarily, we should be aware that:

- Fuzzy subsets describe vague concepts;
- A fuzzy subset admits the possibility of partial membership in it;
- The degree an object belongs to a fuzzy subset is denoted by a membership value between 0 and 1;
- A membership function associated with a given fuzzy subset maps an input value to its appropriate membership value.

3.2.3 General aggregation operators

Aggregation operations on fuzzy subsets are operations by which several fuzzy subsets are combined to produce a single subset (Klir and Yuan, 1995). In general, any aggregation operation is defined by a function

$$f : [0,1]^n \rightarrow [0,1] \tag{3.18}$$

where $n \geq 2$. When applied to n fuzzy subsets $\tilde{A}_1, \ldots, \tilde{A}_n$ defined on X, f produces an aggregation fuzzy subset $\tilde{A}$ by operating on the membership grades of each $x \in X$ in the aggregation sets. Thus,

$$m_{\tilde{A}}(x) = f(m_{\tilde{A}_1}(x),\ldots,m_{\tilde{A}_n}(x)) \tag{3.19}$$

for each $x \in X$.

The nature of aggregation of variables $m_{\tilde{A}_1}(x),\ldots,m_{\tilde{A}_n}(x)$ could be any of the following (Bloch 1996; Petrou and Sasikala, 1999):

- Aggregation is conjunctive if

$$m_{\tilde{A}}(x) \leq \min(m_{\tilde{A}_1}(x),\ldots,m_{\tilde{A}_n}(x)), \tag{3.20}$$

  which states that a conjunctive operator has confidence at most as high as the smallest membership value and looks for the simultaneous satisfaction of all combined criteria;

- Aggregation is disjunctive if

$$m_{\tilde{A}}(x) \geq \max(m_{\tilde{A}_1}(x),\ldots,m_{\tilde{A}_n}(x)), \tag{3.21}$$

  which states that a disjunctive operator has confidence at least as high as the greatest membership value and looks for a redundancy between the combined criteria;

- Aggregation is a compromise if

$$\min(m_{\tilde{A}_1}(x),\ldots,m_{\tilde{A}_n}(x)) \leq m_{\tilde{A}}(x) \leq \max(m_{\tilde{A}_1}(x),\ldots,m_{\tilde{A}_n}(x)), \tag{3.22}$$

  which may represent a cautious behaviour.

3.3 A Greedy heuristic for driver scheduling

From the viewpoint of driver scheduling, the vehicle schedule consists of a set of pieces of work $I=\{1, \ldots, m\}$ to be covered. A very large set of potential shifts $S=\{S_1, \ldots, S_n\}$ has been

generated. Each shift covers a subset of the pieces of work ($S_j \subseteq I$ for $j \in J=\{1, \ldots, n\}$), and has an associated cost c_j (hours paid). A subset of shifts ($J^* : J^* \subseteq J$) covers all the work if

$$\bigcup (S_{j^*} : j^* \in J^*) = I \,. \tag{3.23}$$

Because of the advantage of fast computational speed, besides the traditional approach of ILP, alternative approaches of using greedy techniques are sometimes employed for driver scheduling. A simple greedy heuristic for the set covering problem is at each step, to choose the unused set (shift) which covers the largest number of remaining elements (pieces of work). For the weighted set covering problem, choose the unused shift $S_j (j \in J)$ with the largest ratio $\langle S_j \rangle / c_j$, based on the assumption that at each iteration the possibility for shift S_j to be selected increases with its number of uncovered pieces of work, denoted as $\langle S_j \rangle$, and decreases with its cost c_j. In terms of driver scheduling, a straightforward greedy operator would be choosing the unused S_j with the largest uncovered worked time in each iteration.

A more refined greedy heuristic is presented herein, based on the assumption that the desirability of using shift S_j in an optimal solution increases with its functional value $F(S_j)$. This function consists of two components, and can be formulated as

$$F(S_j) = f_1(S_j) \times f_2(S_j), \forall j \in J \,, \tag{3.24}$$

where $f_1(S_j) \in [0,1]$ is called the over-cover penalty, and $f_2(S_j) \in [0,1]$ is called the structural coefficient (to be described in Section 3.4). With respect to $f_1(S_j)$, the ratio of the overlapped worked time to the total worked time in S_j should be regarded as one of the important criteria to determine whether or not to choose shift S_j. The choice of using worked time as the adaptive operator is based on experimental results, which will be given in Section 3.5.1 later.

Over-cover means a piece of work has been covered by more than one shifts. The over-cover penalty can be formulated as

$$f_1(S_j) = \sum_{k=1}^{|S_j|} (a_{jk} \times b_{jk}) \Big/ \sum_{k=1}^{|S_j|} b_{jk}, \forall j \in J . \quad (3.25)$$

Where $|S_j| =$ number of pieces of work in S_j;

$$a_{jk} = \begin{cases} 1 & \text{, if work piece } k \text{ in } S_j \text{ has not been covered by any other shifts } S_i \text{ in schedule } J^*; \\ 0 & \text{, otherwise;} \end{cases}$$

$b_{jk} =$ worked time for work piece k in S_j.

If every piece of work in S_j has been covered by other shifts in J^*, $f_1(S_j) = 0$; conversely if none of the pieces of work is overlapped, $f_1(S_j) = 1$.

This following construction heuristic is similar to that presented in (Kwan et al., 2001). The difference is the greedy operator in the proposed heuristic is to choose the shifts with the largest function value $F(S_j)$, while the greedy operator in Kwan's heuristic is to simply chose the shift with largest uncovered pieces of work.

Considering all the potential shifts in the large set with respect to the pieces of work to be covered, each piece of work i has an associated coverage list with a length of L_i, i.e. containing L_i shifts that covers it. The steps for the proposed algorithm to construct a feasible schedule are:

Step 0 Set $I' = \{1,2,...,m\}$, where m is the number of pieces of work to be covered.

Step 1 If $I' = f$ then stop: J^* is a feasible schedule. Otherwise find an index $k \in J$ having $F(S_k) = \max(F(S_j): j \in J)$ from the shortest coverage list $L_{i'}(i' \in I')$, and proceed to step 2.

Step 2 Add shift S_k to J^*, set $I' = I' - S_k$, and return to step 1.

The construction heuristic can be regarded as a process of assigning shifts until every piece of work has been covered. In the beginning, each of the pieces of work will have a so-called coverage list. Candidate shifts are then assigned to the unassigned pieces of work sequentially. The criterion of choosing the next uncovered piece of work for assignment is that it has the shortest coverage list, and within the coverage list the shift with the largest function value $F(S_j)$ is chosen.

It should be noted that in a feasible solution, over-cover is often inevitable and usually can be resolved easily by manual editing before the schedule is implemented.

3.4 Fuzzy evaluation of shift structure

The process of constructing a potential schedule by means of greedy heuristics is inherently sequential. However, among the large set of potential shifts, it would be difficult to judge which one is more effective than others because the criteria bear some uncertainty. To mitigate the problem, fuzzy evaluation, a powerful tool to describe quantitative uncertain values and relations between them, is used to introduce the concept of structural coefficient. It gives shift $S_j (j \in J)$ a quantitative value $f_2(S_j) \in [0,1]$ according to its structural state. The fitter the structure for S_j, the larger $f_2(S_j)$ is.

The main idea is to set up several criteria characterized by fuzzy membership functions, and then make decisions based on an aggregation of the fuzzified criteria. Considering the structural state of a shift in several aspects, the result will be more reliable than conventional approaches in determining the efficiency of the shift.

There are two steps in establishing the new concept. First, a number of fuzzified criteria should be obtained according to the efficiency of a shift, which describes quantitatively the characteristic of its structural state from different aspects. Secondly, fuzzy evaluation will be applied to appraise effectively the shift structural state for decision-making. These two steps are presented respectively as follows.

3.4.1 Construction of the fuzzified criteria

Driver scheduling is a specialised set covering problem. In terms of ILP, the columns are shifts, which must satisfy conditions in the Labour Agreement between management and unions, not just any possible combination of pieces of work. In TRACS II, a BUILD process is used to generate such a large set of potential shifts, by means of a set of parameters.

Explained in the following sections, the main criteria for evaluating shift structure are total worked time (u_1), ratio (u_2) of total worked time to spreadover (normally the paid hours for a driver from sign on to sign off), number of pieces of work (u_3), and number of spells (u_4) contained in a shift. Furthermore, the fractional cover by Linear Programming (LP) relaxation (u_5) is regarded as the fifth criterion.

3.4.1.1 Criterion u_1

In driver scheduling, not all the time from sign on to sign off is regarded as worked time, although it might be fully paid.

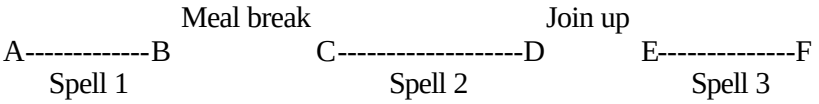


Figure 3.1: Example shift with gaps of meal break and join up

Considering a 3-spell shift in Figure 3.1, the interval between B and C is time given to a driver to take a meal break, and the interval between D and E is a join up time required for a driver to change train like a passenger from relief point D to E, plus some slack, without a meal break. The actual worked time in this shift is therefore the sum of all the on-vehicle time excluding join up and/or meal break time.

It is intuitive that shifts with longer worked time are more efficient than those with shorter worked time. Hence we can assume that the goodness of a potential shift $S_j (j \in J)$ generally increases with its total worked time. Furthermore, since in most real world driver scheduling problems only a very small proportion of the shifts in the large set will be used to produce efficient schedules, it is not desirable to have larger variations in the measure of goodness among these elite shifts. On the contrary, for shifts with longer worked time, their goodness should increase as smoothly as possible, allowing them more chances to be selected later. Based on this consideration, the kind of increase should be non-linear. Thus, the S-shape quadratic membership function ($m_{\tilde{A}_1}$), rather than the simple linear function, can be applied to define criterion u_1 as

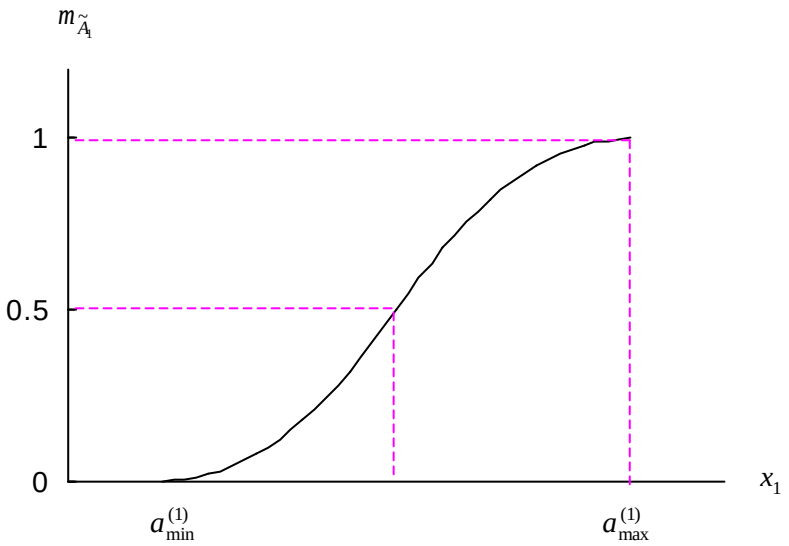
$$m_{\tilde{A}_1} = \begin{cases} 2\left(\dfrac{x_1 - a_{\min}^{(1)}}{a_{\max}^{(1)} - a_{\min}^{(1)}} \right)^2, & a_{\min}^{(1)} \le x_1 < \dfrac{a_{\min}^{(1)} + a_{\max}^{(1)}}{2} \\ 1 - 2\left(\left(\dfrac{x_1 - a_{\max}^{(1)}}{a_{\max}^{(1)} - a_{\min}^{(1)}} \right) \right)^2, & \dfrac{a_{\min}^{(1)} + a_{\max}^{(1)}}{2} \le x_1 \le a_{\max}^{(1)} \end{cases}, \qquad (3.26)$$

where x_1 = total worked time of S_j;

$a_{\max}^{(1)}$ = maximum total worked time;

$a_{\min}^{(1)}$ = minimum total worked time.

The characteristic curve of function $m_{\tilde{A}_1}$ is shown below:



3.4.1.2 Criterion u_2

Besides the absolute worked time, the relative ratio of actual worked time to spreadover (paid hours) can be regarded as another important criterion.

Shift 1. A_1----------B_1 Spell 1 C_1----------D_1 Spell 2

Shift 2. A_2----------B_2 Spell 1 C_2----------D_2 Spell 2

Shift 3. A_3----------B_3 Spell 1 C_3----------D_3 Spell 2

Figure 3.2: Example shifts with different lengths of spells and gaps

Considering three shifts each with two spells in Figure 3.2, in terms of the length of worked time, shift 1 and shift 2 are equal, while shift 3 is shorter due to its shorter second spell. In terms of the length of the gap, shift 1 is the longest, shift 2 second, and shift 3 third. Among these three shifts, shift 2 is obviously more efficient than shift 1 because of its shorter gap between spells and thus shorter paid hours. However, shift 3 is also regarded intuitively as more efficient than shift 1 because of its much shorter gap, even though its second spell is slightly shorter than that of shift 1.

The above analysis leads to the criterion rule that shifts with larger ratio of worked time to the spreadover are regarded as more efficient than those with shorter ratio. Hence an associated membership function can be designed based on the assumption that the goodness of a potential shift $S_j (j \in J)$ generally increases with this ratio. For the similar reason given in section 3.4.1.1 above, this increase should be non-linear as well. Again, the S-shape quadratic membership function ($m_{\tilde{A}_2}$), rather than the simple linear function, is applied to define criterion u_2 as

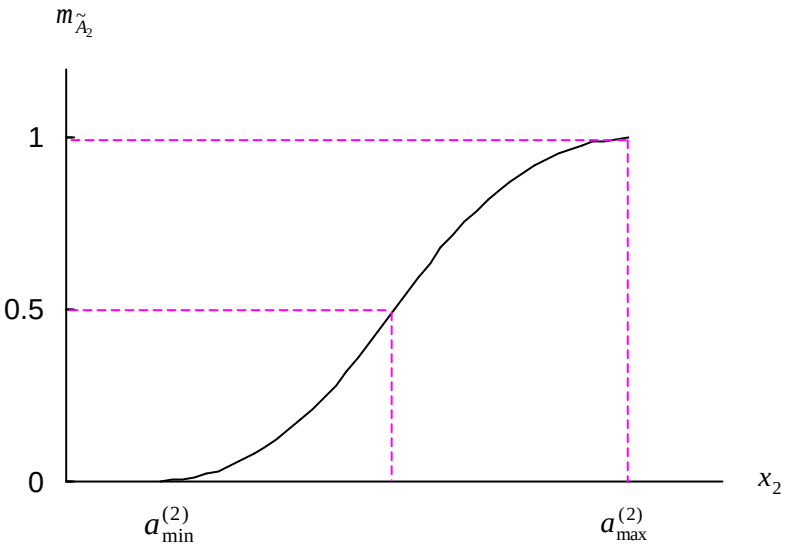
$$m_{\tilde{A}_2} = \begin{cases} 2\left(\dfrac{x_2 - a_{\min}^{(2)}}{a_{\max}^{(2)} - a_{\min}^{(2)}}\right)^2, & a_{\min}^{(2)} \le x_2 < \dfrac{a_{\min}^{(2)} + a_{\max}^{(2)}}{2} \\ 1 - 2\left(\dfrac{x_2 - a_{\max}^{(2)}}{a_{\max}^{(2)} - a_{\min}^{(2)}}\right)^2, & \dfrac{a_{\min}^{(2)} + a_{\max}^{(2)}}{2} \le x_3 \le a_{\max}^{(2)} \end{cases}, \tag{3.27}$$

where x_2 = ratio of total worked time to spreadover for S_j;

$a_{\max}^{(2)}$ = maximum ratio;

$a_{\min}^{(2)}$ = minimum ratio.

The characteristic curve of function $m_{\tilde{A}_2}$ is shown below:



3.4.1.3 Criterion u_3

A shift may contain several spells, and each spell contains a number of consecutive pieces of work. The number of pieces of work may be regarded as one of the criteria about the shift structure.

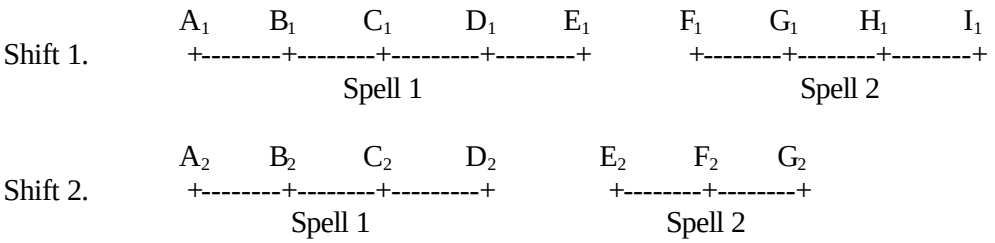


Figure 3.3: Example shifts with different number of pieces of work

Considering two 2-spell shifts in Figure 3.3, shift 1 is composed of seven pieces of work, and shift 2 is composed of five pieces of work. Driver scheduling is a bi-objective combinatorial problem. Although the main objective is to minimize the overall cost of the schedule, for practical reasons, the number of shifts in the schedule is also to be minimised. To the

objective of minimizing the number of shifts, shift 1 is more efficient than shift 2 because it covers more pieces of work and hopefully saves shifts in the final schedule.

If every shift in the final schedule covers as many pieces of work as possible, the number of shift potentially might be minimised. Hence an associated membership function can be designed based on the assumption that the goodness of a potential shift $S_j (j \in J)$ generally increases with the number of pieces it covers. Since the range of number of work pieces among the shifts is small (normally smaller than 30) and this variable is discrete, the non-linear curve using u_1 or u_2 is not appropriate. A linear membership function ($m_{\tilde{A}_3}$) is therefore applied to define criterion u_3 as

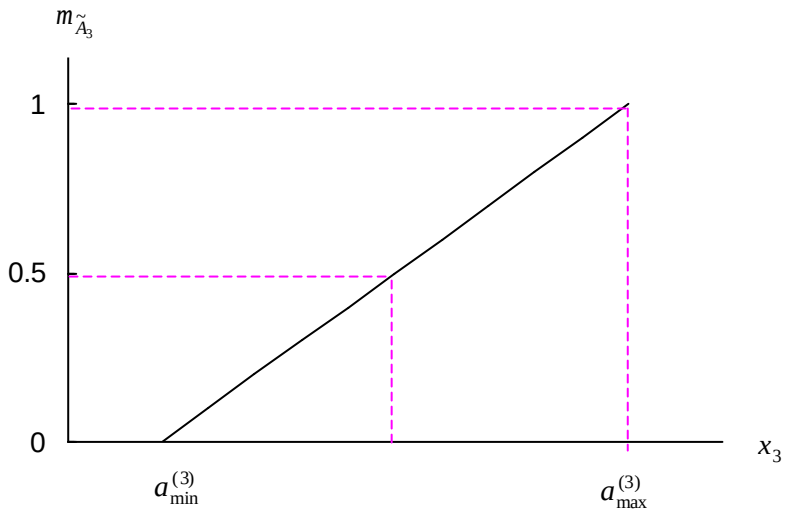
$$m_{\tilde{A}_3} = \frac{x_3 - a_{\min}^{(3)}}{a_{\max}^{(3)} - a_{\min}^{(3)}}, \tag{3.28}$$

where x_3 = number of pieces of work contained in S_j;

$a_{\max}^{(3)}$ = maximum number of pieces of work;

$a_{\min}^{(3)}$ = minimum number of pieces of work.

The characteristic representation of function $m_{\tilde{A}_3}$ is shown below:



3.4.1.4 Criterion u_4

In the current TRACS II process of generating the large set of potential shifts, the parameter that decides the maximum number of spells in a shift is usually set to be 4. The reason why shifts with more than 4-spells are not constructed is that they are seldom efficient, and also the combinations of work pieces for forming them would be so enormous that they would cause computational difficulties. However, among the shifts with up to four spells, differences of structural efficiency or user preferences still exist.

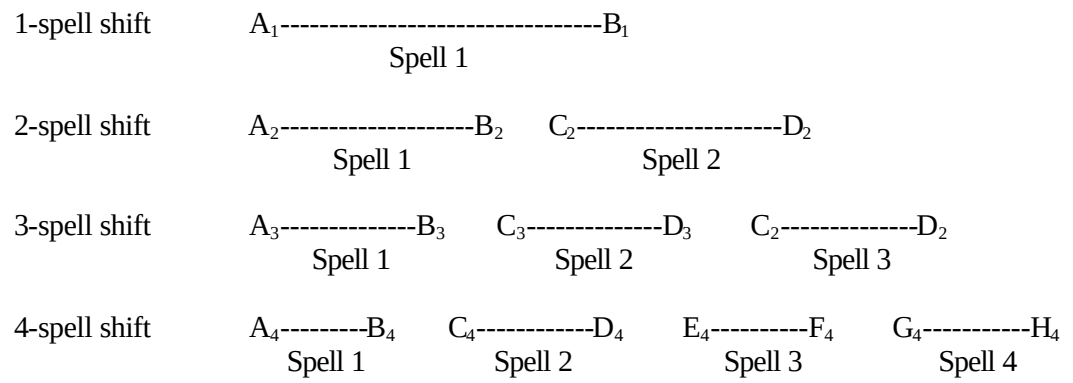


Figure 3.4: Example shifts with one to four spells

Considering the example shifts with up to four spells in Figure 3.4, 1-spell shifts are shorter shifts without a meal break in between the work pieces. These shifts may either be an overtime shift, or a half shift where a meal break can be placed at the beginning or at the end of it, and the other half would be made up with required work such as shunting. These shifts seem to be inefficient and are discouraged by transport operators, even if sometimes they are crucial in forming optimal schedules.

On the other hand, 2-spell shifts are highly encouraged because they are inherently more robust and preferred than shifts with three or four spells. Usually, a shift with more spells would inevitably result in an additional meal break, or more time for the driver to transfer

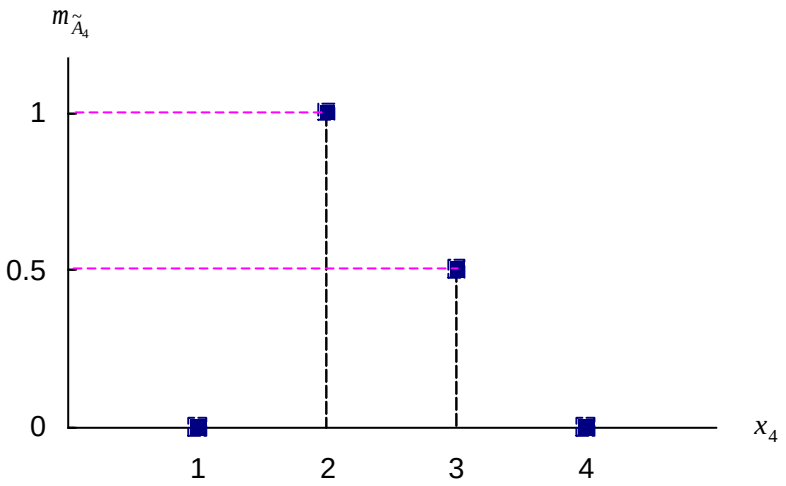
from one vehicle to another vehicle. Therefore, between the 4-spell shift and the 3-spell shift, the 4-spell one seems to be less preferable.

Based on the above consideration, membership function $m_{\tilde{A}_4}$ for the spell factor u_4 can be designed as

$$m_{\tilde{A}_4} = \begin{cases} 0, & \text{if } x_4 = 1 \text{ or } x_4 = 4 \\ 1/2, & \text{if } x_4 = 3 \\ 1, & \text{if } x_4 = 2 \end{cases}, \qquad (3.29)$$

where x_4 = number of spells contained in S_j.

The characteristic representation of function $m_{\tilde{A}_4}$ is shown below:



3.4.1.5 Criterion u_5

The common method for shift selection is Integer Linear Programming, which uses the branch-and-bound tree-search procedure to produce integer solutions. This is a non-

polynomial time algorithm, limited by the amount of search space to be explored within reasonable time. For this reason, the ILP process sometimes may be terminated before any integer solution has been found. However the relaxed LP, i.e. ignoring the integer constraints, can usually be solved quickly: it provides some useful information about the distribution of the optimum integer solution. Therefore, the relaxed Linear Programming solution u_5, if applicable, can be considered as an additional criterion.

With regard to driver scheduling, the relaxed LP solution is an assignment of possibly fractional values to shifts, in which the sum of the shifts covering any piece of work is not smaller than 1 (shown in Figure 3.5). The number of shifts used in this solution, i.e. the sum of the possibly fractional values, is a very good estimate of the lower bound on the optimal number of shifts. In practice, the optimal number of shifts is usually obtained by rounding up the number of shifts to the next higher integer.

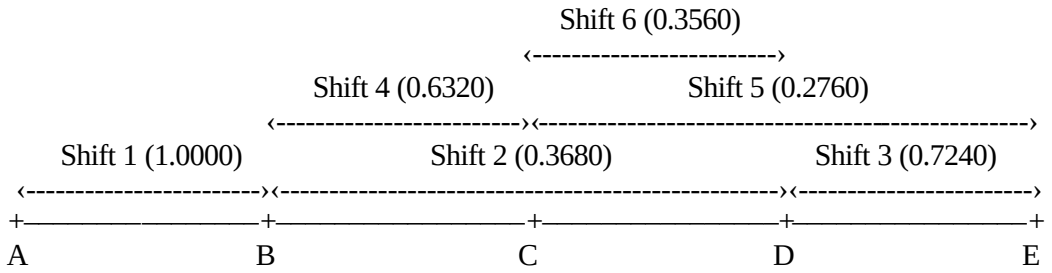


Figure 3.5: Fractional values in a relaxed LP solution

Some constraint programming systems have utilized the relaxed LP solution to solve scheduling problems in different ways (Guerinik and Caneghem, 1995; Rodosek et al., 1996; Curtis et al., 1999), in which the fractional values in this solution were employed as a guide of choosing variables to be satisfied. Although their searching approaches were slightly different, both Guerinik and Caneghem (1995) and Rodosek et al. (1996) used the fractional value of a shift as the guide to the first value chosen for their shift variables, while Curtis et al. (1999) assigned the fractional values to relief opportunities rather than shifts.

Another successful application of the relaxed LP solution, by the means of hybrid genetic algorithms, is described in (Kwan et al., 1999; Kwan 1999; Kwan et al., 2001), which included all the shifts with fractional values larger than a given parameter (such as 0.3) in the candidate seeding shift set, and reported better results than other heuristics. They investigated a large number of data sets originating from bus and train operators, and found that, among all the constructed shifts, those shifts with fractional values in the relaxed LP solution are more likely to be used in the integer solution then others. The investigative results of some selected larger cases in (Kwan et al., 1999), together with the largest instance of Gall2 obtained recently, are shown in Table 3.1.

Date	Number of shifts	(a) Shifts in the relaxed solution	(b) Shifts in an integer solution	(c) Common shifts in (a) and (b)	(c)/(b)×100 %
Wakh	30000	428	106	88	83
G34a	30701	476	106	70	66
Wkh	30000	420	109	98	90
Tl96	17430	368	112	85	76
Tl97	22917	377	112	99	88
G309	27973	425	113	76	67
Swbx	30423	474	132	104	79
Gall2	144339	627	242	219	90

Table 3.1: Relationship of shifts in an integer solution and those in the relaxed LP solution

Table 3.1 shows that, for larger cases, more than 60% of the shifts in the final integer solution exist in the relaxed LP solution: the average percentage is about 80%, while the maximum percentage even reaches 90%. Since there may be many different integer solutions using the same minimum number of shifts, those shifts in the relaxed LP solution but not in the best integer solution found by TRACS II might still be potentially good in terms of contributing to a minimum shift schedule.

To study the distribution of the shifts in the integer solution with respect to their fractional values in the relaxed LP solution, an extract of the largest Gall2 problem with an integer solution of 242 shifts is shown in Table 3.2, in which 219 shifts have the fractional values in the relaxed LP solution.

Fractional values in the relaxed solution	(1) No. of shifts	(2) No. of shifts in an integer solution	(2)/(1)×100 %
0	143712	23	0.02%
0.0000–0.0999	257	12	4.67%
0.1000–0.1999	50	4	8.00%
0.2000–0.2999	41	8	19.51%
0.3000–0.3999	41	13	31.71%
0.4000–0.4999	12	4	33.33%
0.5000–0.5999	10	3	30.00%
0.6000–0.6999	33	13	39.39%
0.7000–0.7999	18	9	50.00%
0.8000–0.8999	20	17	85.00%
0.9000–1.0000	145	136	93.79%

Table 3.2: Distribution of shifts in an integer solution with respect to their fractional values

Whether or not a shift has a fractional value in the relaxed LP solution significantly affects its chance to be included in the optimum integer solution. If not, the chance can be regarded probabilistically as zero (only 0.02% in this case). However the shifts involved in this small chance may still be vitally important to complete a perfect solution, which will be demonstrated by experiments in Section 3.5.3.3. Otherwise, the higher the fractional value, the more likely the shift will be present in the integer solution (for example, 4.67% in the interval [0.000-0.0999] and 93.79% in the interval [0.9000-1.0000]).

Furthermore, since the shifts in the integer solution are concentrated in fractional values close to 1, the membership function with respect to criterion u_5 should be non-linear. With the

advantage of being smooth and nonzero at all points, the Gaussian distribution function ($m_{\tilde{A}_5}$) is applied to define criterion u_5 as

$$m_{\tilde{A}_5} = \begin{cases} e^{-\frac{(x_5-a)^2}{b}} & \text{, if } S_j \text{ is in the fractional cover} \\ 0 & \text{, otherwise} \end{cases} \tag{3.30}$$

Let $m_{\tilde{A}_5} = 1$ when $x_5 = a^{(5)}_{\max}$, and $m_{\tilde{A}_5} = 0.01$ when $x_5 = a^{(5)}_{\min}$,

where x_5 = fractional value of S_j in the relaxed LP solution;

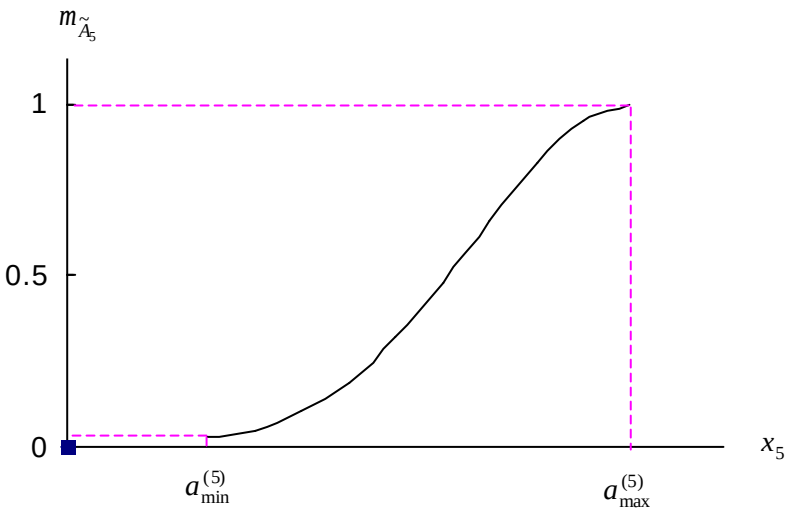
$a^{(5)}_{\max}$ = maximum value in fractional cover;

$a^{(5)}_{\min}$ = minimum value in fractional cover.

Therefore,

$$\begin{cases} a = a^{(5)}_{\max} \\ b = -\dfrac{(a^{(5)}_{\max} - a^{(5)}_{\min})^2}{\ln 0.01} \end{cases} \tag{3.31}$$

The characteristic curve of function $m_{\tilde{A}_5}$ is shown below:



3.4.2 Evaluation of shift structure by fuzzy aggregation

Five criteria u_1 (i= 1,…,5) have been abstracted for the evaluation of the shift structure, and each criterion is assigned a numerical evaluation by fuzzy membership function $m_{\tilde{A}_i}$ (i=1,…,5), where $\tilde{A}_i$ is the fuzzy subset on the i-th criterion. These evaluations refer to local feature of each criterion respectively, thus an overall evaluation on fuzzy subset $\tilde{A}$ could be made by aggregation of these five criteria.

As introduced in Section 3.2.3, the nature of aggregation of $m_{\tilde{A}_1}(x),...,m_{\tilde{A}_5}(x)$ falls into any of these three categories: conjunctive, disjunctive, or compromise. The relative influence and inter-relationship of these criteria cannot be predicted. Therefore, a specific operator from each of the three aggregation categories is selected for experiments, whose results will be reported in Section 3.5.1. The chosen aggregation operators are described as follows.

1) Conjunctive aggregation

When criteria u_i are of equal importance, the fuzzy subset $\tilde{A}$ of good shift structure with respect to the five criteria may be defined as intersection of all the fuzzy subsets $\tilde{A}_i$, i.e.

$$\tilde{A} = \bigcap_{i=1}^{5} \tilde{A}_i \ , \tag{3.32}$$

Therefore, for shift $S_j (j \in J)$, the formulation of its structural coefficient $f_2(S_j)$ by the corresponding aggregation operator is:

$$f_2(S_j) = m_{\tilde{A}}(x) = \min(m_{\tilde{A}_1}(x),...,m_{\tilde{A}_5}(x)), \forall j \in J \ . \tag{3.33}$$

This aggregation is “pessimistic” in the sense that each shift is assigned its worst evaluation.

2) Disjunctive aggregation

When criteria u_i are of equal importance, the fuzzy subset $\tilde{A}$ of good shift structure with respect to the five criteria may be also defined as union of all the fuzzy subsets $\tilde{A}_i$ i.e.

$$\tilde{A} = \bigcup_{i=1}^{5} \tilde{A}_i \ , \tag{3.34}$$

Therefore, for shift $S_j (j \in J)$, the formulation of its structural coefficient $f_2(S_j)$ by the corresponding aggregation operator is:

$$f_2(S_j) = m_{\tilde{A}}(x) = \max(m_{\tilde{A}_1}(x),...,m_{\tilde{A}_5}(x)), \forall j \in J \ . \tag{3.35}$$

This aggregation is "optimistic" in the sense that each shift is assigned its best evaluation.

3) Compromise aggregation

The above aggregation schemes assume that the criteria cannot compensate each other. When this is no longer true, other schemes may be considered, such as product, arithmetic mean, and geometric mean. Therefore, for shift $S_j (j \in J)$, the formulation of its structural coefficient $f_2(S_j)$ using the arithmetic mean operator is:

$$f_2(S_j) = m_{\tilde{A}}(x) = \sum_{i=1}^{5} w_i m_{\tilde{A}_i}, \forall j \in J \ . \tag{3.36}$$

Where $w_i (w_i \geq 0)$ denotes the corresponding weights for criteria u_i (i=1, …, 5). They all satisfy the normalizing condition

$$\sum_{i=1}^{5} w_i = 1 \tag{3.37}$$

If the i-th criterion were dominant in assessing the shift structure, its weight should have a high value.

3.5 Experiments on the greedy heuristic with fuzzy evaluation

The major concern about using heuristics is the quality of the obtained solution. Due to lack of knowledge about the true optimum, an effective method of assessing the quality of a heuristic solution is by comparing it to the best known solution. Therefore, Relative Percentage Deviation (RPD) over the best known schedule is applied to measure the quality of a heuristic schedule: the smaller the RPD is, the better the result is.

$$RPD = \left(\frac{\text{Heuristic schedule} - \text{Best known schedule}}{\text{Best-known schedule}} \right) \times 100\% \qquad (3.38)$$

Data	Type	Number of pieces of work	Number of potential shifts	Best known schedule	
				Shifts	Cost (hours paid)
Colx	Bus	127	3560	34	288.16
Gmb	Bus	154	11817	34	289.32
Neur	Train	340	29380	62	509.25
Ew3	Train	437	25099	116	1003.55
Wag3	Train	456	16636	50	403.42
F2x	Train	546	43743	64	562.22
Tram	Tram	553	6437	49	419.50
Trmx	Tram	553	29500	49	408.47
Nb2	Bus	613	22568	75	851.09
Gall2	Train	707	144339	242	2247.52
G532	Train	1164	29465	276	2083.15
Gall1	Train	1495	28639	349	2661.12
Rrne	Train	1873	50000	395	3137.20

Table 3.3: Size and the best known schedules of the test problems

Table 3.3 shows the sizes and the best known results of thirteen real world problems. For example, the largest case RRNE is a scheduling problem from the former Regional Railways North East, which had a diverse operation comprising rural, inter-urban and urban operations to cover most area of northeast England. The best known schedules are mostly obtained by

the TRACS Ⅱ: some (F2x, Gall2 and Rrne) from version 12.0b, and the others from the previous version. The only side constraint present in these data sets are on the total number of shifts. Default settings of the TRACS II parameter were used. For the cases of Neur and Nb2, the ILP process fails to find any integer solution even though the ILP has been re-run many times and each time with a higher revised target number of shifts. In this circumstance, results reached by hybrid GAs incorporating strong domain knowledge (Kwan 1999) are cited.

3.5.1 Determination of the greedy operator in over-cover penalty

Schedules derived from the three simple greedy heuristics (described in Section 3.3) are compared to decide which greedy component should be kept in the proposed algorithm. These three simple greedy heuristics, whose results are tabulated in Table 3.4 as Greedy schedule (1), Greedy schedule (2), and Greedy schedule (3), choose the potential shift $S_j (j \in J)$ with the largest uncovered pieces of work, the largest ratio $\langle S_j \rangle / c_j$, and the largest uncovered worked time respectively in each iteration.

Data	Greedy schedule (1)		Greedy schedule (2)		Greedy schedule (3)	
	Shift	Cost	Shift	Cost	Shift	Cost
Colx	41	349.32	44	382.62	40	356.70
Gmb	40	355.35	42	359.93	40	350.70
Neur	74	621.45	75	643.07	74	602.20
Ew3	128	1199.88	128	1241.78	123	1156.27
Wag3	61	501.38	59	495.66	59	490.18
F2x	76	714.18	70	693.72	74	700.95
Tram	61	558.72	61	564.12	61	558.55
Trmx	63	561.25	63	571.50	61	547.45
Nb2	84	985.32	84	1006.67	84	974.25
Gall2	295	2738.08	291	2946.22	274	2497.23
G532	322	2412.13	323	2537.32	329	2398.78
Gall1	405	3079.55	402	3308.95	410	3033.72
Rrne	457	3793.27	471	4024.17	452	3704.32

Table 3.4: Computational results of simple greedy heuristics

According to Table 3.5, the average RPD in terms of number of shifts and total cost for these three heuristics are (18.77%, 22.89%), (19.11%, 27.05%), and (16.89%, 20.36%) respectively. In general, the 3rd greedy heuristic works better with both objectives. Therefore, this greedy operator will be adopted, and uncovered worked time is used in the over-cover penalty $f_1(S_j)$, which is described in Section 3.3.

Data	Greedy schedule (1)		Greedy schedule (2)		Greedy schedule (3)	
	RPD (shift)	RPD (cost)	RPD (shift)	RPD (cost)	RPD (shift)	RPD (cost)
Colx	20.59	21.22	29.41	32.78	17.65	23.79
Gmb	17.65	22.82	23.53	24.41	17.65	21.22
Neur	19.35	22.03	20.97	26.28	19.35	18.25
Ew3	10.34	19.56	10.34	23.74	6.03	15.22
Wag3	22.00	24.28	18.00	22.86	18.00	21.51
F2x	18.75	27.03	9.38	23.39	15.63	24.68
Tram	24.49	33.19	24.49	34.47	24.49	33.15
Trmx	28.57	37.40	28.57	39.91	24.49	34.02
Nb2	12.00	15.77	12.00	18.28	12.00	14.47
Gall2	21.90	21.83	20.25	31.09	13.22	11.11
G532	16.67	15.79	17.03	21.80	19.20	15.15
Gall1	16.05	15.72	15.19	24.34	17.49	14.00
Rrne	15.70	20.91	19.24	28.27	14.43	18.08
Ave.	**18.77**	**22.89**	**19.11**	**27.05**	**16.89**	**20.36**

Table 3.5: RPD results of simple greedy heuristics

3.5.2 Selecting the aggregation operator for the structural coefficient

As described in Section 3.4.2, three different kinds of aggregation operator have been applied to evaluate the shift structure, educing the so-called structural coefficient $f_2(S_j)$. The product of structural coefficient $f_2(S_j)$ and over-cover penalty $f_1(S_j)$, denoted as $F(S_j)$, is used as the final criterion in the new approach to decide which shift should be used in the process of

constructing a schedule. In this section, the most suitable aggregation operator will be determined according to the experimental results.

3.5.2.1 Intersection operator

Defined by formula (3.33), the intersection operator is a "pessimistic" aggregation, in which each shift is assigned the worst evaluation. According to the membership functions defined, if taking the smallest value by this operator, most shifts involve zero values for their structural coefficient $f_2(S_j)$, i.e. shifts with one or four spells with regard to the criterion u_4, and shifts without fractional values in the relaxed LP solution with regard to the criterion u_5. Thus the associated overall evaluation function values $F(S_j)$ for these shifts are zero. Since in the process of constructing a schedule, each step is to choose the shift with the largest value of $F(S_j)$, shifts with zero evaluation values will never have chances to be selected. In many circumstances, these unselected shifts are vitally important, sometimes necessary, to produce a good schedule.

Data	Shift	RPD (%)	Cost	RPD (%)
Colx	47	38.24	389.90	35.31
Gmb	43	26.47	367.52	27.03
Neur	83	33.87	661.38	29.87
Ew3	134	15.52	1186.68	18.25
Wag3	57	14.00	462.37	14.61
F2x	76	18.75	711.02	26.47
Tram	60	22.45	520.77	24.14
Trmx	66	34.69	552.75	35.32
Nb2	121	61.33	1181.97	38.88
Gall2	275	13.64	2553.32	13.61
G532	294	6.52	2337.42	12.21
Gall1	382	9.46	3085.93	15.96
Rrne	466	17.97	3912.33	24.71
Ave.		**24.07**		**24.34**

Table 3.6: Experimental results by using the intersection operator

As shown in Table 3.6, the average RPD for number of shifts and total cost are 24.07% and 24.34% respectively. In general the results using this operator are no better than those of the simpler greedy heuristics. This operator is therefore discarded without further investigation.

3.5.2.2 Union operator

Defined by formula (3.35), the union operator is an "optimistic" aggregation, in which each shift is assigned its best evaluation. According to the membership functions defined, if taking the largest value by this operator, the structural coefficients $f_2(S_j)$ for most shifts are one, i.e. those shifts with two spells with regard to the criterion u_4, and shifts with a fractional value of 1.0000 in the relaxed LP solution with regard to the criterion u_5. Thus for a large proportion of shifts, their associated overall evaluation values $F(S_j)$ are equal to $f_1(S_j)$, meaning that the component of structural coefficient $f_2(S_j)$ is actually not taking effect. In recognition of this, the performance of the algorithm using such an operator should be better, but not too much better, than the simple greedy heuristics.

Data	Shift	RPD (%)	Cost	RPD (%)
Colx	38	11.76	326.30	13.24
Gmb	40	17.65	353.73	22.26
Neur	70	12.90	588.07	15.48
Ew3	126	8.62	1145.42	14.14
Wag3	62	24.00	510.33	26.50
F2x	70	9.38	686.52	22.11
Tram	67	36.73	602.13	43.54
Trmx	62	26.53	544.43	33.29
Nb2	79	5.33	931.87	9.49
Gall2	268	10.74	2532.22	12.67
G532	309	11.96	2374.92	14.01
Gall1	390	11.75	3036.88	14.12
Rrne	445	12.66	3717.95	18.51
Ave.		**15.39**		**19.95**

Table 3.7: Experimental results by using the union operator

The experimental results in Table 3.7 have demonstrated the above. The average RPD for number of shifts and total cost are 15.39% and 19.95% respectively. In general the results using the union operator are slightly better than those of the best simple greedy heuristic, i.e. the 3rd one, in terms of both the number of shifts and total cost.

3.5.3.3 Arithmetic mean operator

Both the intersection and union operators are based on the assumption that all the criteria are of equal importance, and cannot compensate each other. Since the schedules produced by these operators are not satisfactory, another operator of arithmetic mean is tested the performance of the proposed approach.

1) Individual effect of each criterion

Before studying whether these criteria are compensative with each other, it would be interesting to know the individual effect of each single criterion on the system's performance. Table 3.8 and Table 3.9 show the experimental results and their associated RPD results of using the single criteria u_i (i=1,…,5) respectively, which are obtained by simply setting $w_i = 1$ and $w_j = 0(j \neq i)$ in formula (3.36).

Data	Criterion 1		Criterion 2		Criterion 3		Criterion 4		Criterion 5	
	Shift	Cost	Shift	Cost	Shift	Cost	Shift	Cost	Shift	Cost
Colx	38	341.08	44	334.48	49	433.83	44	349.02	42	360.58
Gmb	39	353.98	45	346.75	49	447.92	46	364.38	40	334.38
Neur	72	606.32	73	578.42	80	689.13	99	711.08	75	602.80
Ew3	123	1223.68	135	1121.33	127	1258.73	169	1303.62	125	1070.90
Wag3	57	487.43	59	481.57	63	543.03	82	601.93	57	459.37
F2x	72	716.75	79	658.48	74	745.50	83	630.55	77	711.78
Tram	67	638.53	59	512.15	72	674.13	69	570.02	58	501.30
Trmx	64	600.75	61	533.75	72	657.75	66	532.23	57	475.82
Nb2	83	973.35	86	940.68	80	941.43	116	1092.52	92	977.50
Gall2	267	2713.05	284	2424.33	299	3059.88	337	2581.33	249	2297.65
G532	307	2507.58	336	2246.45	328	2667.90	421	2530.43	293	2298.90
Gall1	388	3162.12	429	2851.45	413	3393.30	544	3263.03	371	2952.70
Rrne	444	3758.58	453	3632.12	520	4439.80	513	3984.45	437	3683.65

Table 3.8: Computational results by using single criterion individually

Data	Criterion 1		Criterion 2		Criterion 3		Criterion 4		Criterion 5	
	RPD (shift)	PRD (cost)	RPD (shift)	PRD (cost)	RPD (shift)	PRD (cost)	RPD (shift)	PRD (cost)	RPD (shift)	PRD (cost)
Colx	**11.76**	**18.36**	29.41	16.07	44.12	50.55	29.41	21.12	23.53	25.13
Gmb	14.71	22.35	32.35	19.85	44.12	54.82	35.29	25.94	**17.65**	**15.57**
Neur	16.13	19.06	**17.74**	**13.58**	29.03	35.32	59.68	39.63	20.97	18.37
Ew3	6.03	21.94	16.38	11.74	9.48	25.43	45.69	29.90	**7.76**	**6.71**
Wag3	14.00	20.82	18.00	19.37	26.00	34.61	64.00	49.21	**14.00**	**13.87**
F2x	**12.50**	**27.49**	23.44	17.12	15.63	32.60	29.69	12.15	20.31	26.60
Tram	36.73	52.21	20.41	22.09	46.94	60.70	40.82	35.88	**18.37**	**19.50**
Trmx	30.61	47.07	24.49	30.67	46.94	61.03	34.69	30.30	**16.33**	**16.49**
Nb2	10.67	14.37	14.67	10.53	**6.67**	**10.61**	54.67	28.37	22.67	14.85
Gall2	10.33	20.71	17.36	7.87	23.55	36.14	39.26	14.85	**2.89**	**2.23**
G532	11.23	20.37	21.74	7.84	18.84	28.07	52.54	21.47	**6.16**	**10.36**
Gall1	11.17	18.83	22.92	7.15	18.34	27.51	55.87	22.62	**6.30**	**10.96**
Rrne	12.41	19.81	14.68	15.78	31.65	41.52	29.87	27.01	**10.63**	**17.42**
Ave.	**15.25**	**24.88**	**21.044**	**15.36**	**27.79**	**38.38**	**43.96**	**27.57**	**14.41**	**15.26**

Table 3.9: RPD results by using single criterion individually

According to the criterion u_1 (total worked time in a shift), shifts with more work content have preference to be selected in the final schedule. Given the finite work content, the construction process by such a strategy will potentially result in fewer number of shifts in the schedules. The average RPD results of this criterion in Table 3.9 demonstrate this trend: it achieves 15.25% in terms of the number of shifts, which is better than those of the three simple greedy heuristics.

According to the criterion u_2 (ratio of total worked time to spreadover), it is encouraged that shifts with larger percentage of work content have preference to be selected in the final schedule. The construction process by this strategy will potentially result in little non-worked time, reducing the total cost in the schedules. The average RPD results of this criterion in Table 3.9 also demonstrate this trend: it achieves 15.36% in terms of the total cost, which is better than those of the simple greedy heuristics.

The effects for the criterion u_3 (number of pieces of work) and the criterion u_4 (number of spells) are more complex. As described in Section 3.4.1.3 and Section 3.4.1.4, these two criteria are regarded as the effective factors for shift structure. However, if each criterion is isolated to evaluate the shift structure, the results are much worse in both objectives (shown in Table 3.9). With respect to the criterion u_3, the reason might be that it increases significantly the number of over-covered pieces of work, thus increases eventually the total cost in the schedule. With respect to the criterion u_4, probably it is because some shifts, though not many, with other than two spells are vitally important to good schedules.

Obviously, the criterion u_5 (fractional values in the relaxed LP solution) worked best separately compared with the other four criteria, according to its average RPD results in Table 3.9. Compared with the three simple greedy heuristics, it also performs best in terms of both number of shifts and total cost.

2) Combined effect of all criteria

From the experimental results of using single criterion, the individual effect of each single criterion on the system's performance is not satisfactory, and no criterion is absolutely dominant than others although the 5th criterion can produce better solutions in general. In this section, experiments of using the combined criteria by the arithmetic mean operator will be implemented to study whether and how these criteria are compensative with each other.

Table 3.10 shows the summary results of 100 runs with weight combinations generated randomly. The four important indices of Maximum, Minimum, Mean, and Standard Deviation in statistics are applied to study the distribution of the computational results in terms of both number of shifts and total cost. Let t_i be a variable of either the number of shifts or total cost in a schedule, then

$$\text{Maximum} = \max\{t_1, ..., t_{100}\}, \tag{3.39}$$

$$\text{Minimum} = \min\{t_1, ..., t_{100}\}, \tag{3.40}$$

$$\text{Mean } \bar{t} = \sum_{i=1}^{100} t_i \Big/ 100, \tag{3.41}$$

$$\text{Standard Deviation} = \sqrt{\sum_{i=1}^{100} (t_i - \bar{t})^2 \Big/ 99}\,. \tag{3.42}$$

The Mean is a measure to evaluate the average performance of the proposed algorithm, while the Standard Deviation is a summary measure of the differences of each result from the mean. According to the RPD results in Table 3.11, the average RPD of Means in number of shifts and total cost are 13.84% and 13.03% respectively. It shows that the arithmetic mean operator with different weights works much better than other aggregation operators and the simple greedy heuristics.

Data	Maximum		Minimum		Mean		Standard Deviation	
	Shift	Cost	Shift	Cost	Shift	Cost	Shift	Cost
Colx	42	350.65	37	312.22	39.44	328.34	0.97	9.94
Gmb	42	353.02	37	313.02	39.92	334.67	1.04	8.13
Neur	84	632.33	69	551.45	73.60	579.39	2.52	14.86
Ew3	146	1223.13	120	1036.43	129.84	1105.42	6.41	41.88
Wag3	71	552.35	54	432.40	59.84	479.03	3.39	24.12
F2x	87	698.90	68	594.55	73.12	636.28	3.37	21.71
Tram	66	604.40	52	448.73	57.84	515.58	4.02	45.25
Trmx	63	576.37	51	435.50	55.68	483.60	3.04	35.85
Nb2	108	1039.87	79	880.12	87.85	929.39	4.83	27.49
Gall2	316	2560.88	245	2276.42	258.32	2362.23	13.21	65.91
G532	333	2326.38	280	2164.37	296.64	2225.70	11.20	34.93
Gall1	429	2987.13	358	2798.43	378.16	2862.75	15.78	47.42
Rrne	477	3768.38	413	3432.50	437.12	3572.82	13.49	67.99

Table 3.10: Summary results of 100 runs with randomised weights

Data	Maximum		Minimum		Mean	
	RPD (Shift)	RPD (Cost)	RPD (Shift)	PRD (Cost)	RPD (Shift)	RPD (Cost)
Colx	23.53	21.66	8.82	8.35	16.00	13.94
Gmb	23.53	22.02	8.82	8.19	17.41	15.67
Neur	35.48	24.17	11.29	8.29	18.71	13.77
Ew3	25.86	21.88	3.45	3.28	11.93	10.15
Wag3	42.00	36.92	8.00	7.18	19.68	18.74
F2x	35.94	24.31	6.25	5.75	14.25	13.17
Tram	34.69	44.08	6.12	6.97	18.04	22.90
Trmx	28.57	41.10	4.08	6.62	13.63	18.39
Nb2	44.00	22.18	5.33	3.41	17.13	9.20
Gall2	30.58	13.94	1.24	1.29	6.74	5.10
G532	20.65	11.68	1.45	3.90	7.48	6.84
Gall1	22.92	12.25	2.58	5.16	8.36	7.58
Rrne	20.76	20.12	4.57	9.41	10.66	13.89
Ave.	**29.89**	**24.33**	**5.54**	**5.98**	**13.84**	**13.03**

Table 3.11: RPD results of 100 runs with randomised weights

Particularly, it achieves an encouraging finding: among the 100 randomised weight combinations, there always exist several ones that can derive rather satisfactory results for each case. For example, Table 3.11 shows that the average RPD of Min in number of shifts and total cost are as small as 5.54% and 5.98% respectively.

According to the experimental results, in general the results of using the combined weights are much better than those of using single criterion. It can be concluded that the five criteria proposed are compensative with each other. Moreover, since a significant difference exists among schedules produced by individual weight combinations, hopefully better results may be achieved if these criteria are given more suitable weights. Therefore, the arithmetic operator will be kept for further investigation.

3.6 Conclusions

A refined greedy algorithm based on fuzzy subsets theory has been presented in this chapter. The new algorithm is novel because it is the first time that fuzzy set theory has been applied to the driver scheduling problem. An effective method is proposed to solve the problem about ranking the potential shifts in each iteration. Unlike the simple greedy algorithms, the new approach employs fuzzy evaluation which depends on five fuzzified criteria about the structure of a shift including total worked time, ratio of total worked time to spreadover, number of pieces of work, number of spells, and fractional cover.

The evaluation operator in the proposed algorithm is the product of two components, namely over-cover penalty and structural coefficient. The over-cover penalty is actually an adaptive greedy operator, in which there are several alternative choices for its greedy unit. Among the three simple greedy heuristics, the one using the largest uncovered worked time is adopted because of its best performance by experiments. With regard to structural coefficient, there

are also three aggregation operators in different categories as candidates, among which the arithmetic mean operator takes into account the interactive factor of individual criterion, and performs best in general. This operator is therefore applied to aggregate the proposed criteria.

It is interesting to find that, among the results of using 100 weight combinations generated randomly, outstanding results were obtained for each test case, some of which are rather close to the best known solutions (within 2%). Therefore more sophisticated algorithms are worthwhile to be developed and further investigated, in particular we would like to investigate:

1) Whether more refined sets of weight combinations could be derived to produce even better results;
2) Whether these better results produced, if achievable, can be improved further by other strategies.

Research in these two issues are pursued by means of two different evolutionary algorithms, i.e. a GA and a Simulated Evolution algorithm, which will be presented in Chapter 4 and Chapter 5 respectively.

Chapter Four

A Genetic Algorithm for Weight Determination

4.1 Introduction

The greedy heuristic introduced in Chapter 3 constructs a schedule by sequentially selecting shifts, from a very large set of pre-generated legal potential shifts, to cover the remaining work. Individual shifts and the schedule as a whole have to be evaluated in the process. Fuzzy set theory is applied on such evaluations. For individual shifts, their structural efficiency is assessed by fuzzified criteria identified from practical knowledge of the problem domain (described in Section 3.4). These criteria are represented by fuzzy membership functions about the structure and generally the goodness of a shift. The fuzzy membership functions are weighted and combined to yield an overall evaluation. While it might be possible to derive a general relatively robust set of weights by experiments, these weights will have to be fine-tuned for individual problems for best performance.

In this chapter, a GA is described for calibrating the weight distribution amongst the fuzzified criteria, so that a single-valued weighted evaluation can be computed for each shift. Although driver schedules are constructed as by-products through generations of evolution, they are not expected to be very close to optimum because of the crude greedy nature of the schedule construction method used. Much of the work in this chapter has been published by Li and Kwan (2002a).

This GA approach belongs to the general class of hybrid GAs (memetic algorithms, genetic local search) (Ackley 1987; Moscato and Norman, 1992; Radcliffe and Surry 1994), i.e. algorithms that hybridize genetic operations with local or constructive heuristics. Furthermore, there are some similarities between the idea introduced in this paper and the GRASP algorithm (Feo and Resende, 1995) or the Adaptive Multi-Start (AMS) technique (Boese et al., 1994): they all apply adaptive construction heuristics to obtain individual feasible solutions, and perform searches based on multiple solutions to improve the local optimum. However, the formations of multi-start are very different: the proposed GA is based on an evolutionary mechanism, while GRASP is purely randomized and AMS maintains a constant number of best solutions found so far.

This chapter is organized in the following way. A brief overview of GAs is given first. The GA approach, which follows a simple 'standard' scheme, for weight determination is then described in detail. Comparative results using real-life problems, some of which are very large instances, are presented. Finally, conclusions are discussed.

4.2 A brief overview of GAs

GAs are general-purpose search and optimisation methods originating from Holland (1975) and developed subsequently to solve a wide area of real-world problems (Davis 1987; Goldberg 1989; Michaelwicz 1994; Zalzala and Fleming, 1997). These algorithms are based

on the mechanics of genetics and natural selection, and represent the search space of a coded population of potential solutions. The population is then manipulated according to the survival of the fittest principle, providing good practical solutions.

4.2.1 Basic concepts

The GA is an iterative procedure consisting of normally a constant-size population of individuals. Each individual is represented by a finite string of symbols, called the *chromosome*, to encode a possible solution in a given problem space. This space, referred to as the *search space*, comprises all possible solutions to the problem at hand. In most cases, the GA is applied to spaces that are too large to be exhaustively searched. The symbol alphabet commonly used is binary, although other representations have also been used, such as character-based encoding, real-valued encoding, and tree representation.

The standard GA proceeds as follows. An initial population of individuals is generated randomly or heuristically. In every evolutionary step, called a *generation*, the individuals in the current population are *decoded* and *evaluated* according to some predefined quality criteria, called the *fitness function*. To form a new population, or the next generation, individuals are *selected* according to their fitness. Many selection procedures are currently in use, among which the simplest one is the *roulette selection*, where individuals are selected probabilistically according to their fitness values. This ensures that the expected number of times for an individual to be chosen is approximately proportional to its relative performance in the population. Thus, high-fitness, or good, individuals have better chances of survival, while low-fitness ones are more likely to be eliminated.

Selection alone cannot introduce new individuals into the population, i.e., it cannot find new points in the search space. New individuals are generated by genetically inspired operators, of which the most widely used are *crossover* and *mutation*. Crossover is performed with

crossover probability p_c between two selected individuals, called *parents*, by exchanging parts of their chromosomes to generate two new individuals, called *offspring.* In its simplest form, substrings are exchanged after a crossover point randomly selected. This operator tends to enable the evolutionary process to move towards the promising regions of the search space. To prevent premature convergence to local optima, the mutation operator is introduced by sampling new points at random in the search space, which is carried out by flipping bits randomly with some small probability p_m.

GAs are stochastic iterative processes that are not guaranteed to find the optimal solution. The termination condition may be specified as some fixed number of generations, or as the attainment of a satisfactory fitness level.

4.2.2 Some applications

GAs have attracted much research interest over the last two decades. The following areas of application (Mitchell 1996) in both problem solving and scientific contexts are by no means exhaustive, but exemplify what GAs have been used for.

1) Optimisation: to solve a wide variety of optimisation tasks, including numerical optimisation and combinatorial optimisation problems such as circuit layout and driver scheduling;
2) Automatic programming: to evolve computer programs for special tasks, and to design other computational structures such as cellular automata and sorting networks;
3) Machine learning: to evolve particular machine learning systems, such as weights for neural networks, rules for learning classifier systems or symbolic production systems, and sensors for robots;

4) Computer-aided design: to use the feedback from the evaluation process to select the fitter designs, generate new designs through the combination of parts of the selected designs, and result in a population of high performance designs eventually;
5) Evolution and learning: to study how individual learning and species evolution affect one another;
6) Social systems: to study evolutionary aspects of social system and the evolution of cooperation and communication in multi-agent systems;
7) Economics: to model the processes of innovation, the development of bidding strategies, and the emergence of economic markets;
8) Ecology: to model ecological phenomena such as biological arm races, host-parasite coevolution, symbiosis, and resource flow;
9) Immune systems: to model various aspects of natural immune systems, including somatic mutation during an individual's lifetime and the discovery of multi-gene families evolutionary time.

4.2.3 Comparison with traditional search methods

GAs differ from the traditional search and optimisation methods significantly in the following four aspects:

1) GAs search a population of points in parallel, rather than a single point;
2) GAs work on the encoding of the solution set rather than the solution set itself, except where real-valued individuals are used;
3) GAs use probabilistic transition rules, not deterministic ones;
4) GAs do not require derivative information or other auxiliary knowledge. Only the objective function and corresponding fitness levels influence the search directions.

Furthermore, it is important to point out that, to a given problem, GAs can provide a number of potential solutions and the choice of a final solution is left for the user to decide. In cases where a specific problem does not have a unique solution, for example in multi-objective optimisation where the result is usually a group of Pareto-optimal solutions, GAs are particularly useful for identifying the alternative solutions simultaneously.

4.3 Using GA to produce near-optimal weights

The evaluation function $F(S_j)$, described in Section 3.3, involves a weight distribution among five membership functions corresponding to five fuzzified criteria. Determination of these weights is a nonlinear problem, which implies that it is impossible to treat each weight as an independent variable to be solved in isolation from other variables. There are interactions such that the combined effects of the weights must be considered in order to optimise the output.

GAs are useful approaches to problems requiring an efficient search over a large solution space, and are particularly suitable for obtaining approximate solutions for multivariable optimisation problems where mathematical analyses are difficult. In this section, the evolutionary process of using GA to generate near-optimal weights for the fuzzy membership functions will be described in detail. The basic framework of the GA is given as follows:

Step 0 Set generation $t = 0$; initial population $P(t)$ is generated with randomised weight sets.

Step 1 Apply the greedy algorithm to obtain a solution, the cost of which is used to evaluate the corresponding weight set in $P(t)$.

Step 2 If termination criterion has not been reached, continue step 3; otherwise stop.

Step 3 Set $t = t + 1$; select weight sets from $P(t\text{-}1)$ for reproduction.

Step 4 Perform adaptive crossover and mutation operators to produce offspring, and partially replenish $P(t)$ by randomly generated members. All new weight sets are evaluated by applying the greedy algorithm above to obtain a schedule.

Step 5 Return to Step 2.

4.3.1 Chromosome representation

GAs operate simultaneously on a number of potential solutions, called a population, consisting of some encoding of the parameter set. Typically, a population is composed of 50 to 100 individuals.

The first step in designing the GA is to represent the weights in a way suitable for applying the genetic operators. The chromosome representation used here is the most commonly used single–level binary string, in which the weights w_i (i=1, 2, 3, 4, 5; $w_i \in [0,1]$) are continuous variables requiring an appropriate discrete representation. Each discretised value of the variable is first linearly mapped to an integer defined in a specified range, encoded using a fixed number of binary bits. The binary codes of all the variables are then concatenated to obtain a binary string as follows:

w_1	w_2	w_3	w_4	w_5

For weight w_i, if it is encoded in m binary bits, then

$$e_{w_i} = (w_{i(\max)} - w_{i(\min)})/(2^m - 1), \forall i \in \{1,2,3,4,5\} \quad (4.1)$$

where e_{w_i} is the precision of w_i, $w_{i(\max)}$ and $w_{i(\min)}$ is the upper bound and lower bound of w_i respectively. In this research, each w_i is encoded in 6 binary bits. Hence, the problem is a 5-dimension-search, and the solution space is 2^{30}.

4.3.2 Initialisation

An initial population can be achieved simply by generating the required number of weight sets randomly. In this approach, with a population of *N* weight sets whose chromosomes are 30 bits long, *N* random 4-byte unsigned integer numbers uniformly distributed from the range of $[0, 2^{30}-1]$ each is produced and then mapped to individual weights between 0 and 1.

4.3.3 The fitness functions

Driver scheduling is a bi-objective combinatorial problem. Although the main objective of driver scheduling is to minimise the overall cost of the solution, for practical reasons, we also wish that the number of shifts in the schedule is as few as possible. In multi-criteria optimization, the primary goal is to find or to approximate the set of Pareto-optimal solutions, and recently some multiple objective versions of hybrid GAs have been proposed (Ishibuchi and Murata, 1998; Jaszkiewicz et al., 2001).

In GAs, the objective function and the fitness function are different notions. The objective function provides a measure of performance with respect to a particular parameter set, while the fitness function transfers that measure of performance into an allocation of reproductive opportunities. In maximised optimisation where the fitness function is deemed to be the objective function, these two notions are sometimes used interchangeably. However, transformation of the objective function is necessary when the objective function is to be minimised, since lower objective function values correspond to fitter weight sets.

4.3.3.1 A simple fitness function

In automatically making trade-offs between the objectives, traditional approach combines all objectives in a weighted sum cost function, and the schedule with the lowest weighted sum is

regarded as the best solution (Fores et al., 1999; Kwan 1999; Kwan et al., 1999). Based on this technique, the objective function $g(x)$ for schedule *x* can be simply formulated as

$$g(x) = \sum_{j^*=1}^{l} (c_{j^*} + 2000), \tag{4.2}$$

where *l* is the number of shifts in the schedule *x*, and c_{j^*} is the cost of shifts S_{j^*}. The constant 2000 is used here so that a heavy weight is imposed on each shift and it helps to reduce the total number of shifts.

This is a minimisation problem, thus it is necessary to transform (4.2) into a maximised fitness function $f(x)$ as follows. The larger the value of $f(x)$, the fitter the weight set is.

$$f(x) = G(g(x)) = a + b - g(x), \tag{4.3}$$

where *a* is the maximum objective value in the population;

b is the minimum objective value in the population.

4.3.3.2 A fuzzy goal-based fitness function

It is well-known that problems exist with the above weighted sum function if the Pareto surface is non-convex (Steuer 1986; Ulungu 1994). In addition, the determination of weights for individual objectives will be practically very difficult. This section presents a goal-directed search approach, where the best schedule is the one that satisfies a vector of fuzzy goals as much as possible.

Let O be the set of solutions generated by the hybrid GA. Consider that we are minimizing a *n*-valued cost vector denoted as $f(x) = (f_1(x), f_2(x),..., f_n(x))$ where $x \in \Omega$. Suppose $f_{\min} = (f^1_{\min}, f^2_{\min},..., f^n_{\min})$ is a vector that gives lower bound estimates on the individual objective, which usually are not reachable in practice, such that $\forall i, f^i_{\min} \le f_i(x), \forall x \in \Omega$; and $f_{\max} = (f^1_{\max}, f^2_{\max},..., f^n_{\max})$ is a vector that indicates user-specified upper bounds for the objectives such that $\forall i, f^i_{\max} \ge f_i(x), \forall x \in \Omega$. If $\forall i, f_i(x) \in [f^i_{\min}, f^i_{\max}]$, x will be an effective solution. It means that this solution might include some useful information, and is worthy to be preserved in the evolutionary process. For the bi-objective driver scheduling problem, the region of the effective solutions is shown in Figure 1.

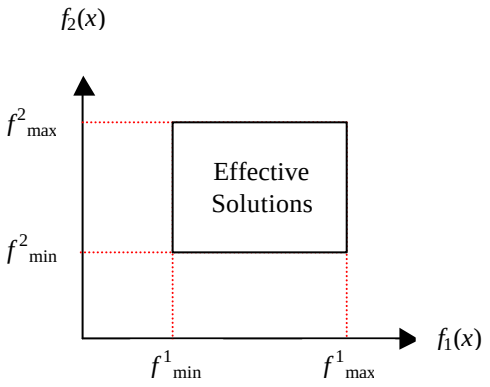


Figure 4.1: Range of effective solution set

In the proposed scheme, the effective solution set is a fuzzy set. Therefore, applying the arithmetic mean operator described in section 3.4.2, the fuzzy goal based fitness function, $f(x) \in [0,1]$, can be designed as follows:

$$f(x) = \begin{cases} \sum_{i=1}^{2} t_i \times m_i(x), \text{if } x \text{ is an effective solution} \\ 0 \qquad\qquad\qquad\quad , \text{otherwise} \end{cases} \tag{4.4}$$

where

$$\sum_{i=1}^{2} t_i = 1,\ t_i \ge 0, \tag{4.5}$$

and

$$m_i(x) = \frac{f^i_{\max} - f_i(x)}{f^i_{\max} - f^i_{\min}}, i = 1, 2. \quad (4.6)$$

Note that formula (4.6) has transformed the original minimised objectives to a maximisation problem. Therefore, the fitness function $f(x)$ in (4.4) takes the same form as its objective function. The larger the value of $f(x)$, the fitter the weight set is.

User preferences can be easily expressed in the upper bound vector $f_{\max}$. For example, by increasing the upper bound value $f^i_{\max}$ to $f^{i\ *}_{\max}$, the subsequent membership function $m^*_i(x)$ for objective i will increase, which might control the acceptance or rejection of solutions. In this work, the lower bound on the 1st objective, i.e. the total cost, is computed as the total cost of all pieces of work, based on the assumption that there is no overlapped work in a perfect schedule. The lower bound on the 2nd objective, i.e. the number of shifts, is the sum rounded up of the fractional solution derived by the LP relaxation. If the problem has not been solved using LP relaxation, the lower bound can be estimated by experienced schedulers. The upper bound adopted here is the solution value obtained by the simple greedy algorithm of choosing the unused shift with the largest uncovered work content in each iteration, which has been mentioned in section 3.3.

4.3.4 Selection

Selection models nature's survival-of-the-fittest mechanism. Fitter solutions survive while weaker ones perish. The traditional roulette wheel strategy based on fitness-proportionate selection is used here. Individuals with the best fitness values in each generation are always preserved.

Fitness-proportionate selection is a common selection method in GAs, in which the number of times a weight set is expected to reproduce is equal to its fitness divided by the average fitness of the population. A simple method for implementing this selection is "roulette-wheel sampling", which is conceptually equivalent to giving each weight set a slice of a circular roulette wheel equal in area to its fitness. The wheel is spun *N* times, where *N* is the number of weight sets in the population. On each spin, the set under the wheel's marker is selected into the pool of parents for the next generation. In more detail, this method can be implemented as follows:

1. Sum the total fitness value of weight sets in the population, denoted as *S*.
2. Repeat *N* times:
    2.1 Generate a random integer *t* in the interval [0, *S*];
    2.2 Loop through the weight sets in the population, summing the fitness values until the sum is larger than or equal to *t*. Select the weight set whose fitness value puts the sum over this limit.

By these steps, the stochastic method statistically results in the expected number of offspring for each weight set.

4.3.5 Genetic operators

To implement a GA, the genetic operators of crossover and mutation have to be used.

4.3.5.1 Multipoint crossover

Crossover is the basic operator to produce new chromosomes that have some parts of both parents' genetic material. The simplest form is single-point crossover.

In multipoint crossover, *m* crossover positions are chosen at random with no duplicates and sorted into ascending order. Then, the bits between successive crossover points are exchanged between the two parents to produce two new offspring. The section between the first allele position and the first crossover point is not exchanged between individuals. A 5-point crossover is applied to the 30-bit chromosome in the proposed GA, which is illustrated in Figure 4.2. The parts included in quotation marks in parents are exchanged each other.

Parents	⇨	Offspring
0110 ‘0011’10010 ‘10100’101100 ‘11001’		0110 ‘1111’10010 ‘00110’101100 ‘11010’
1100 ‘1111’00111 ‘00110’101001 ‘11010’		1100 ‘0011’00111 ‘10100’101001 ‘11001’

Figure 4.2: 5-point crossover

The idea behind multipoint crossover is that the parts of the chromosome that contribute most to the performance of a particular individual may not necessarily be contained in adjacent substrings (Booker 1987). Moreover, the disruptive nature of multipoint crossover appears to encourage the exploration of the search space, rather than convergence to highly fit individuals early in the search, thus making the search more robust.

4.3.5.2 Mutation

In natural evolution, mutation is a random process where one allele of a gene is replaced by another to produce a new genetic structure. In GAs, mutation is randomly applied with low probability, and modifies elements in the chromosomes. As an ancillary operator, the roles of mutation are to recover good genetic material that may be lost through selection and crossover (Goldberg 1989) and to ensure that any given string has the chance to be selected.

Binary mutation flips the value of the bit at the mutation point to be selected. The effect of mutation on the decision variable depends on the encoding scheme used. Given that mutation

is generally applied uniformly to an entire population of strings, it is possible that a given binary string may be mutated at multipoint. The effect of a 5-point mutation on a binary string is illustrated in Figure 4.3 for a 30-bit chromosome representing five weights encoded over the interval [0,1] individually.

Parent	01‘1’000	1‘1’1001	‘0’10101	0101‘1’0	011‘0’01
Offspring	01‘0’000	1‘0’1001	‘1’10101	0101‘0’0	011‘1’01

Figure 4.3: 5-point binary mutation

4.3.5.3 Adaptive probabilities of crossover and mutation

There are two essential characteristics in GAs. The first is the capacity to converge to a local or global optimum after locating the region containing such an optimum. The second is the capacity to explore new regions of the solution space in search of the global optimum. The values of Crossover Probability p_c and Mutation Probability p_m, and the type of crossover applied (Spears and Dejong, 1991) is important to the balance between these two characteristics.

To accomplish the trade-off between exploration and exploitation, Srinivas (Srinivas and Patnail, 1994) designed an algorithm that could vary p_c and p_m adaptively in response to the fitness values of the solutions: p_c and p_m are increased when the population tends to get stuck at a local optimum and are decreased when the population is scattered in the solution space.

Here we employ Srinivas’s algorithm formulated as follows, and perform 5-point crossover and mutation operators described above to the five weights. After crossover and mutation, the sum of the five weights is not 1. Renormalization for these weights is therefore needed for each generation.

$$\begin{cases} p_c = k_1(f_{\max} - f')/(f_{\max} - \bar{f}), & f' \geq \bar{f} \\ p_c = k_2, & f' < \bar{f} \end{cases} \tag{4.7}$$

and

$$\begin{cases} p_m = k_3(f_{\max} - f)/(f_{\max} - \bar{f}), & f \geq \bar{f} \\ p_m = k_4, & f < \bar{f} \end{cases} \tag{4.8}$$

where

k_1 = 0.96, k_2 = 0.96, k_3 = 0.12, and k_4 = 0.16 (values are derived from a number of experiments using different parameter combinations);

f_{max} is the largest fitness value in the population;

$\bar{f}$ is the average fitness value of the population;

f' is the larger fitness value between the two parents to be crossed;

f is the fitness value of the solution to be mutated.

4.4 Computational results

The GA approach was coded in Borland C++. All problems were run on the same Pentium II 333 MHz with 196 megabyte RAM personal computer using the Windows 98 operating system. The best known schedules are mostly obtained by the TRACS II system. In cases where TRACS II has difficulty in finding solutions, results reached by hybrid GAs incorporating strong domain knowledge (Kwan 1999; Kwan et al., 2001) are cited.

For the four larger problems Gall2, G532, Gall1 and Rrne, population sizes of 100, 200, 300, 400 and 500 have been tested. For the other problems, population sizes of 50, 100, 200 and 300 were tried. Smaller population sizes might not result in satisfactory solutions, although they would reduce the computation for each generation. Therefore some larger population sizes were also tried. According to the results by several combinations of parameters, the most

effective population sizes were found to be 400 to 500 for the four larger problems, and 100 to 200 for the other problems.

For all the problems, the number of generations for the GA is set to be 150. Furthermore, t_1 and t_2 in the GA's fuzzy goal-based fitness function are set to be both 0.5 respectively. The experimental results in terms of shift number and total cost for the schedules derived by the simple fitness function and fuzzy goal-based fitness function are compiled in Table 4.1 and Table 4.2 respectively. Elapsed time is the time following the solution of the relaxed LP of TRACS II.

Data	Shifts	RPD	Cost (hours)	RPD	Time$^+$ (seconds)	Time* (seconds)
Colx	35	2.94	296.05	2.74	22	22
Gmb	36	5.88	296.87	2.61	85	84
Neur	64	3.23	510.02	0.15	203	955
Ew3	118	1.72	1007.39	0.38	95	69
Wag3	52	4.00	413.87	2.59	24	34
F2x	66	3.13	618.95	8.20	144	>40000
Tram	51	4.08	435.10	3.72	21	24
Trmx	51	4.08	425.70	4.22	72	139
Nb2	76	1.33	851.92	0.10	241	452
Gall2	244	0.83	2282.70	1.57	220	>80000
G532	276	0.00	2147.65	3.10	285	>80000
Gall1	348	-0.29	2734.03	2.74	469	>80000
Rrne	398	0.76	3267.14	4.14	754	>40000
Ave.		**2.44%**		**2.80%**		

Table 4.1: Comparative results using simple fitness function

\+ The computing time for the GA

* The computing time for the best-known solutions

Data	Shifts	RPD	Cost (hours)	RPD	Time+ (seconds)	Time* (seconds)
Colx	35	2.94	294.02	2.03	20	22
Gmb	36	5.88	294.87	1.92	75	84
Neur	64	3.23	509.02	-0.05	230	955
Ew3	118	1.72	1006.49	0.29	105	69
Wag3	51	2.00	409.87	1.60	22	34
F2x	66	3.13	598.35	6.42	120	>40000
Tram	50	2.04	430.50	2.62	26	24
Trmx	50	2.04	420.89	3.04	79	139
Nb2	77	2.67	831.22	-2.33	351	452
Gall2	244	0.83	2274.22	1.19	262	>80000
G532	274	-0.72	2137.65	2.62	275	>80000
Gall1	347	-0.57	2714.88	2.02	509	>80000
Rrne	397	0.51	3259.20	3.89	1350	>40000
Ave.		**1.98%**		**1.94%**		

Table 4.2: Comparative results using fuzzy goal-based fitness function

\+ The computing time for the GA

* The computing time for the best-known solutions

Most of the problem instances are complex. In some cases, the ILP process of TRACS II fails to find an integer solution after a large number of nodes of the branch-and-bound search tree has been explored. In these circumstances, the target is raised by one shift and the ILP process is rerun. The process is repeated until an integer solution can be found, and may be abandoned after the target has been raised many times without success (e.g. Neur and Nb2 instances).

The computational results show that results produced by the GA approach are close to the best known schedules, and results by the fuzzy goal-based fitness function are slightly better than those by the simpler fitness function. Compared with all the best known solutions, solution by the fuzzy fitness function has 1.98% more shifts in terms of total shift number, and is 1.94% more expensive in terms of total cost on average. However, the GA's results are

obtained much faster in general, particularly for larger cases. Note that the comparison of computing time has not included the time taken by the failed ILP runs before the final results were obtained.

Fixing the parameter used above, each data set was run ten times by varying the pseudo random number seed at the beginning of each run. The results are summarized in Table 4.3.

Data	Number of Shifts					Cost			
	In table 4.2	Distribution of runs				Ave.	Min.	Max.	Std. Dev.
		-1	=	+1	=2				
Colx	35		4	5	1	294.32	293.56	297.33	2.01
Gmb	36	1	9			294.67	293.74	296.44	1.79
Neur	64		8	2		508.66	508.44	511.38	1.38
Ew3	118		9	1		1006.73	1005.38	1008.46	1.65
Wag3	51		6	4		410.13	409.23	412.65	1.81
F2x	66		7	3		598.14	589.14	607.32	4.12
Tram	50		7	3		429.89	428.50	433.22	2.37
Trmx	50		6	4		420.94	418.66	424.32	2.45
Nb2	77	1	7	2		830.24	830.11	835.42	3.02
Gall2	244	1	8	1		2275.18	2270.11	2279.69	6.58
G532	274		6	2	2	2138.63	2134.23	2142.31	5.89
Gall1	347	1	4	4	1	2713.44	2710.42	2723.14	7.89
Rrne	397	2	5	2	1	3260.11	3249.33	3267.78	8.07

Table 4.3: Results of ten runs with fixed parameters but different random seed numbers

Table 4.3 shows that the proposed GA is quite robust. Comparing the number of shifts in the ten runs with the best solutions found before the runs, on average 70.8% of the runs have the same or better results. In terms of solution costs, there is no remarkable variation between the runs. Except the four larger problems that have higher standard deviations in cost, no obvious trend has been detected.

In addition to finding the best schedule, another task for the GA is to explore whether there exists a generally good pattern of weight distribution. According to the experiments, it is found that, in most cases, the GA performs well with the weights vector (0.15, 0.15, 0.15, 0.15, 0.40). Obviously, the fractional cover factor dominates while others play similarly minor roles. However it is interesting that when we only considered the fractional cover factor as the unique factor, all the results were no better than those attained by the simple greedy algorithm.

4.5 Conclusions

In Chapter 3, it has been found that the weight distribution used for the greedy construction algorithm would affect the solution quality significantly. A GA is therefore presented in this chapter, with a simple or fuzzy goal-based fitness function, to derive such elite sets of weight distribution that could produce superior solutions. The benchmark experimental results demonstrate that the GA approach is suitable for solving large size real-world driver scheduling problems.

There are several advantages for the proposed GA. First, using the ILP process, very large problems may have to be decomposed into smaller sub-problems and solved independently. In contrast, the GA can be used to solve large problems in one go. Secondly, the GA is always able to produce a group of solutions, whereas the ILP process in some cases may need several runs to increase the target number of shifts and still not be able to find any integer solution. Thirdly, the GA executes very fast, since its greedy algorithm framework is always capable of producing a feasible solution after $\Theta(m)$ iterations, where m is the number of pieces of work.

However, due to the crude greedy and deterministic nature of the schedule construction method applied, the solutions produced by the GA are still deemed to be a bit far from optimal. Naturally, it would be interesting to know whether these superior solutions produced

could be improved further by other evolutionary strategies. This will be investigated in the next chapter.

Chapter Five

A Fuzzy Simulated Evolution Algorithm for Driver Scheduling

5.1 Introduction

Chapter 3 presents a refined greedy heuristic based on fuzzy subsets theory, which uses the product of over-cover penalty and structural coefficient to decide which shift is going to be selected in the process of constructing a schedule. The main idea is to set up five criteria, characterized by fuzzy membership functions, to evaluate the structure and generally the goodness of a shift. The fuzzy membership functions are weighted and combined to yield an overall evaluation. Experimental results have shown that this heuristic works better than a simple greedy one.

It has been found that the weight distribution of the membership functions is vitally important to the performance of the above greedy heuristic. A GA is therefore presented in Chapter 4 to derive such a near optimal weight set, and thus obtain the associated good solution.

In this Chapter an evolutionary approach is designed to improve the solutions further, based on the crude solution by the simple heuristic (Li and Kwan, 2001) or the refined solution by the GA (Li and Kwan, 2001a and 2001b). The evolutionary algorithm combines the features of iterative improvement and constructive perturbation with the ability to avoid getting stuck at local minima. Its framework is a Simulated Evolution (SE) algorithm, in which the steps of *Evaluation* and *Reconstruction* have been fuzzified. In the *Evaluation* step, each shift in a solution is evaluated by an evaluation function based on its coverage status and five fuzzified criteria. In the *Reconstruction* step, a greed-based heuristic with the above derived fuzzy evaluation function is applied to form a complete solution from a partial solution.

This chapter is organized as follows. Section 5.2 gives a brief introduction of the SE algorithm. Section 5.3 discusses the proposed SE algorithm, describing its procedure and related operators in detail. Benchmark results using real-world problems are presented in section 5.4, and conclusions are given in section 5.5.

5.2 Preliminaries about the Simulated Evolution algorithm

The SE algorithm is a general optimization technique originally proposed by Kling and Banerjee (1987) for the placement problem, based on an analogy to the natural selection process in biological environments. The biological solution to the adaptation process is the evaluation from one generation to the next one by eliminating inferior elements and keeping superior ones for subsequent states. Every element in each generation must constantly prove its functionality under the current conditions in order to remain unaltered. The purpose of this process is to create gradually stable structures which are perfectly adapted to the given constraints. To escape from local optima, nature implements genetic mutation which perturbs the genetic inheritance process, and natural calamities which interfere with natural selection.

5.2.1 Basic concepts

Basically, the algorithm iteratively operates a sequence of *Evaluation*, *Selection* and *Reconstruction* steps on one solution. Besides these three steps, some input parameters (e.g. stopping conditions) and a valid starting solution are initialised in an earlier step called *Initialization.* The outline of the basic SE algorithms is shown in figure 5.1. In the *Evaluation* step, the goodness of each element in the current solution is computed. A measure of goodness is used probabilistically to select elements to be discarded in the *Selection* step. An element with high goodness has a lower probability of being discarded. The resulting partial solution is then fed to the *Reconstruction* step, which implements application specific heuristics to derive a new and complete solution from a partial solution.

Throughout these iterations, the best solution is retained and finally returned as the final solution. The basic SE algorithm is a greedy search strategy that achieves improvement through iterative perturbation and reconstruction. Furthermore, to escape from local minima in the solution space, capabilities for uphill moves must be incorporated. This is carried out in the *Selection* step by probabilistically discarding even some superior elements of the solution. This process is analogous to mutation in GAs.

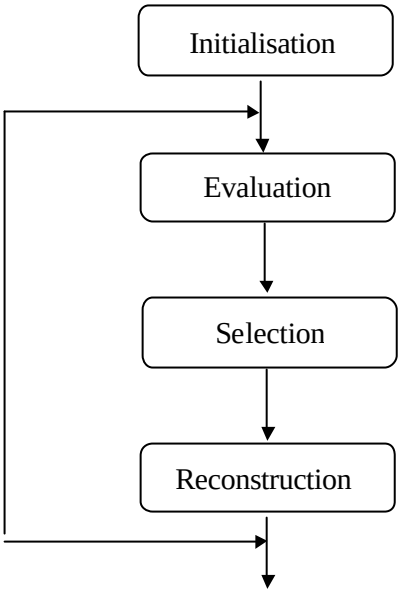


Figure 5.1: SE outline

5.2.2 SE versus other stochastic methods

The main difference between SE and Simulated Annealing is that the latter makes purely random choices to decide which move, or single change, to perform to generate a new solution. The new solution is then evaluated on a global basis, and probabilistically accepted or discarded. In contrast, by discarding ill-suited elements of a solution and then reconstructing a new solution, SE is effectively employing a long sequence of moves, not just a single move as in SA, between iterations, thus permitting more complex and more distant changes between successive solutions.

GA is quite similar to SE in the aspect of using evolution from one generation to the next. GA needs to maintain a number of solutions during each generation as parents, generating children by using crossover operators. SE, however, mimics generations of evolution on a single solution, using domain specific heuristics to repair the partial solution and derive a complete one. Therefore, SE eliminates the extra cpu-time needed to maintain a set of solutions.

Furthermore, the selection processes for GA and SE are completely different. While GA probabilistically selects a set of solutions from the parents to retain according to the fitness of each solution, SE will select the inferior elements of a solution to discard according to the goodness of each element. Due to these strong convergence characteristics, SE generally performs fewer iterations than GA. This is the major reason why SE achieves a significant speedup over Simulated Annealing while GA does not.

5.2.3 Related work

The SE algorithm was originally proposed by Kling and Banerjee (1987) to perform standard cell placement, whose objective was to arrange cells of common height and variable widths in an integrated circuit layout such that a given cost function is minimised. The SE algorithm was used to simulate an evolutionary process to achieve the objective. While obtaining results comparable to or better than the popular Simulated Annealing algorithm, SE performed its task about ten times faster.

Lin et al. (1989) presented a rip-up-and-reroute approach based on a matrix representation scheme and the SE technique to solve the detailed routing problem, the last task next to compaction in a physical design process for VLSI circuits. Experimental results showed that, when solving all the benchmarks from the literature, the SE-based approach out-performed the most successful switch-box router in terms of both quality and speed.

Ly and Mowchenko (1993) applied the SE algorithm to the task of scheduling and allocation in high level synthesis, which was concerned with mapping abstract behavioural specifications of digital systems into structural designs at the register-transfer level. The SE-based synthesis explored the design space by repeatedly ripping up parts of a design in a probabilistic manner, and then reconstructing these parts by local heuristics. This approach combined rapid design iterations and effective design space exploration to obtain superior designs.

Bhuyan (1995) presented an algorithm combined with GA and SE techniques for clustering, which is a process to partition an object space into different classes such that some optimisation criteria are satisfied. Each string in the GA's population was a solution state and consisted of a number of clusters. The global clustering procedure was based on an GA, while within each population the individuals were generated based on an SE. Experimental results

showed that this approach could produce better results than that of the best available heuristics.

Sait et al. (1999) presented a fuzzy SE algorithm for the multi-objective VLSI cell placement problem, which consisted of arranging circuit blocks on a layout surface such that cost was minimised. They proposed a fuzzy goal-based search strategy combined with a fuzzy allocation scheme. The allocation scheme tried to minimise multiple objectives and added controlled randomness as opposed to original deterministic allocation schemes. Experiments with benchmark tests showed a remarkable improvement in solution quality.

5.3 A fuzzy SE algorithm for driver scheduling

The basic idea behind the algorithm is to determine, for each current schedule, the goodness of each shift in its current position. The goodness value is a figure of merit (normalized in the range [0,1]) of how well the shift is used with respect to the other shifts to which it has relations. The goodness is high if the other shifts it is related to cause overlapped work time as little as possible in the present schedule. Conversely, the goodness is low if those shifts result in more overlapped work time. The process of evolution keeps the shifts that are well arranged (having high goodness values) in their present positions and tries to replace the others that have low goodness values. At a particular iteration, a random number in the range [0,1] is generated for each generation, and all shifts whose goodness values exceed that number are labelled as “good shifts” and survive in their present positions; the remaining shifts are labelled as “bad shifts” and do not survive (become extinct) in the current schedule. The goodness value therefore corresponds to the survival chance of a shift in a specific position. The “bad” shifts are removed from the schedule and are put into a queue for the new assignment by using constructive techniques. The above steps are iterated upon. Thus the global scheduling procedure is based on iterative improvement, while an iterative constructive process is performed within.

The algorithm has two places where probabilistic techniques help prevent the solution from getting stuck at a local minimum. The first one is the generation of the threshold value to distinguish "good" shifts from "bad" shifts. The second is in the mutation step. Periodically during the evolution-based iterative improvement process, the system state is mutated, i.e. randomly changed. Usually the mutation rate is much less than the evolution rate to ensure convergence.

In detail, the steps for the proposed fuzzy SE algorithm are described as follows, of which the *Evaluation* and *Reconstruction* steps have been fuzzified.

5.3.1 Precomputation

Among the large number of potential shifts *S*, it would be difficult to judge which one has a more effective structure than others since the criteria bear some uncertainty. If only either affirmative or negative measure is given, some middle information is lost leading to assessment deviation. Thus fuzzy evaluation is used to introduce the concept of a structural coefficient, which gives shift $S_j (j \in J)$ a quantitative value $f_2(S_j) \in [0,1]$ according to its structural state (described in Section 3.4.2). The more efficient the structure for S_j, the larger $f_2(S_j)$ is.

The structural coefficient $f_2(S_j)$ is a component of the goodness evaluation function, and is a constant once the five weights involved have been determined. To avoid the repeated computation in each iteration and thus speed up the SE algorithm, the calculation of $f_2(S_j)$ is put outside of the loop as a *Precomputation* step.

5.3.2 Initialisation

In this step, an initial solution is generated to serve as a seed for the evolutionary process. Due to the fact that the initial exchange rate is relatively high, the algorithm's performance is generally independent of the quality of the initial generation. However, if this seed is already a relatively good solution, the overall computation time will decrease. In our program, the SE algorithms have been implemented by using the following initial solutions:

- As mentioned in Section 3.5.1, an initial solution can be generated by the original greedy method for weighted set covering problem of choosing the unused shift $S_j (j \in J)$ with the largest ratio $|S_j| / c_j$, where $|S_j|$ denotes the number of pieces of work in S_j and c_j is the cost of S_j, until all the pieces of work are covered.

- As explained in Chapter 4, the GA for calibrating the weight distribution of the fuzzy evaluation function would provide, as a by-product, a good solution. This solution can be fed into the SE to serve as a good seed.

The steps described in section 5.3.3 to 5.3.6 are executed in sequence in a loop until a user specified parameter (e.g. cpu-time, total cost, or number of shifts) is reached or no improvement has been achieved for a certain number of iterations.

5.3.3 Evaluation

The first step of the iterative loop is the evaluation of the current arrangement for each shift in a schedule. A goodness value for every shift is established. The purpose of computing this measure is to determine, besides the structural goodness of shifts, which shifts are in positions

that lead to less total overlapped work time, and which shifts contribute unnecessarily to large amounts of overlapped work time.

In this step, goodness of the individual shift in a complete schedule J^* is computed. The formulation of its evaluation function is similar to that used in the greedy heuristic (described in Section 3.3). The major difference is that the former one only evaluates shifts in the current schedule, while the latter needs to evaluate all unused shifts from the large potential shift set, for the purpose of selecting some shifts to form a feasible schedule.

The evaluation function $F(S_{j^*})$ for shift $S_{j^*}(j^* \in J^*)$ should be normalized. Besides the structural coefficient $f_2(S_{j^*})$, another normalized function, which reflects the coverage status for shift S_{j^*}, should be combined. Hence the evaluation function $F(S_{j^*})$ consists of two parts: structural coefficient $f_2(S_{j^*}) \in [0,1]$ and over-cover penalty $f_1(S_{j^*}) \in [0,1]$, which can be formulated as

$$F(S_{j^*}) = f_1(S_{j^*}) \times f_2(S_{j^*}), \qquad \forall j^* \in J^*. \tag{5.1}$$

Over-cover penalty $f_2(S_{j^*})$ is based on the consideration that the ratio of the overlapped work time to total work time in $S_{j^*}(j^* \in J^*)$ is regarded as an important criterion, and thus can be formulated as

$$f_1(S_{j^*}) = \sum_{k=1}^{\left|S_{j^*}\right|} (a_{j^*k} \times b_{j^*k}) \Big/ \sum_{k=1}^{\left|S_{j^*}\right|} b_{j^*k}, \qquad \forall j^* \in J^*, \tag{5.2}$$

where $\left|S_{j^*}\right|$ = number of pieces of work in S_{j^*};

$$a_{j^*k} = \begin{cases} 0 & \text{if work piece } k \text{ in } S_{j^*} \text{ has been covered by any other shift } S_i \text{ in } J^*; \\ 1 & \text{otherwise;} \end{cases}$$

b_{j^*k} = worked time for work pieces k in S_{j^*}.

If every piece of work in S_{j^*} has been covered by other shift S_i in J^*, then $f_1(S_{j^*}) = 0$; conversely if none of the pieces of work in S_{j^*} is overlapped, $f_1(S_{j^*}) = 1$.

5.3.4 Selection

In this step it will be determined whether a shift $S_{j^*}(j^* \in J^*)$ is retained for the next generation, or discarded and placed in a queue for the new allocation. This is done by comparing its goodness $F(S_{j^*})$ to $(p_s - p)$, where p_s is a random number generated for each generation in the range [0, 1], and p is a constant smaller than 1.0. If $F(S_{j^*}) > (p_s - p)$ then S_{j^*} will survive in its present position; otherwise S_{j^*} will be removed from the current evolutionary schedule. The pieces of work it covers, except those also covered by other shifts in the solution, are then released for the next *Reconstruction*. By using this *Selection* process, shift S_{j^*} with larger goodness $F(S_{j^*})$ has higher probability of survival in the current schedule.

The purpose of subtracting p from p_s is to improve the SE's convergence capability. Without it, in the case of p_s close to 1, nearly all the shifts will be removed from the schedule, which is obviously ineffective in searching. In our experiments, p is set to be 0.3. If $p > p_s$, then set $(p_s - p)$ to be 0.

5.3.5 Mutation

To escape from local minima in the solution space, capabilities for uphill moves must be incorporated. This can be carried out in the *Mutation* step by probabilistically discarding even some superior components of the solution. Therefore, following the *Selection* step, each retained shift $S_{j^*}(j^* \in J^*)$ still has a chance to be mutated, i.e. randomly discarded from the partial solution at a given rate of p_m, and release its covered pieces of work, except those also covered by other retained shifts, for the next *Reconstruction*. The mutation rate should be much smaller than the selection rate to guarantee convergence. From empirical results we find that $p_m \leq 0.05$ yields better results.

5.3.6 Reconstruction

The *Reconstruction* step takes a partial schedule as the input, and produces a complete schedule as the output. Since the new schedule should be an evolution of the previous schedule, all shift assignments in the partial schedule should remain unchanged. Therefore, the *Reconstruction* task reduces to that of assigning shifts to all uncovered pieces of work to repair a broken schedule.

Considering all the large number of potential shifts with respect to the pieces of work to be covered, each piece of work i has an associated coverage list with a length of L_i, i.e. containing L_i shifts that covers it. The greed-based constructor assumes that the desirability of adding shift $S_j(j \in J)$ into the partial schedule increases with its function value $F(S_j)$. However, to introduce diversification, we randomly select one of the candidates, not necessarily the top candidate, from a Restricted Candidate List (RCL), which consists of k best shifts. From empirical results we find that $k \leq 3$ achieves better solutions. The steps to generate a complete schedule based on a partial solution are:

Step 1 $J^* = \{l_1, l_2, ..., l_s\}$ is a partial schedule, where l_i is the index number of each shift.

Step 2 Set $I' = I - \bigcup (S_{j^*} : j^* \in J^*)$.

Step 3 If $I' = f$ then stop: J^* is a complete solution and $C(J^*) = \sum (c_{j^*} : j^* \in J^*)$. Otherwise locate a work piece $t' \in I'$ having $L_{t'} = \min(L_{i'} : i' \in I')$, and then randomly select a shift S_r within RCL from the coverage list of row t'. Proceed to step 4.

Step 4 Add *r* to J^*, set $I' = I' - S_r$, and return to step 3.

Some of the features in the *Reconstruction* steps have been considered in the literature, such as the GRASP algorithm (Feo and Resende, 1995) for the set covering problem, the "Peckish" (Corne and Ross, 1996) or "Bias Selection" (Burke et al., 1998) for the time-tabling problem, and the repairing operator of hybrid GAs (Kwan et al., 2001) for the driver scheduling problem.

It should be noted that the shifts added during schedule *Reconstruction* might be redundant, causing all their pieces of work covered by other shifts later, even if each shift is chosen to cover at least one currently uncovered piece of work. However, in the next *Selection*, these redundant shifts will be removed automatically because of their zero goodness. Moreover, the goodness values of all shifts in the current *Reconstruction* might be different from those in the next *Selection* as well due to the updated over-cover penalties at each iteration.

The following example illustrates how the *Reconstruction* step works and may result in a redundant shift, and how the next *Selection* step can remove this redundant shift. To make it easy to understand, the structural coefficient $f_2(S_j)$ for all example shifts are regarded as

equal to 1, thus the goodness computation $F(S_j)$ only concerns with the over-cover penalty $f_1(S_j)$. Furthermore, the random number k in RCL is set to be 1.

[Example] Suppose there are seven shifts S_j (j=1,2,…,7) and eight pieces of work $I' = \{1,2,...8\}$ to be covered. The work content for each work piece is $c_{i'}$ (i'=1,2,…,8).

Let S_1 ={1,2,3,4}, S_2 ={1,2,5,6}, S_3 ={3,4,7,8}, S_4 ={3,4,5}, S_5 ={6}, S_6 ={3,4,7}, S_7 ={8}; and $c_{i'}$=1 (i'=1,2,…,8). On this simple example, the length of the coverage list $L_{i'}$=2 (for i'=1,2,5,6,7,8) and $L_{i'}$=3 (for i'=3,4).

The first step is to locate a work piece with the shortest $L_{i'}$. In case of a tie, the work piece with the smallest subscript will be chosen. Therefore, one of the shifts will be selected from coverage list L_1, which contains S_1 and S_2.

Then

$$F(S_1) = f_1(S_1) = \frac{1\times1+1\times1+1\times1+1\times1}{1+1+1+1} = 1,$$

$$F(S_2) = f_1(S_2) = \frac{1\times1+1\times1+1\times1+1\times1}{1+1+1+1} = 1.$$

Since $F(S_1) = F(S_2)$, S_1 will be selected because of its smaller subscript. Thus $I' = I' - S_1 = \{1,2,..8\} - \{1,2,3,4\} = \{5,6,7,8\}$, and $L_{i'}$=2 (for i'=5,6,7,8). The L_5 containing S_2 and S_4, rather than L_6, is chosen because of its smaller subscript.

Then

$$F(S_2) = f_1(S_2) = \frac{0\times1+0\times1+1\times1+1\times1}{1+1+1+1} = \frac{1}{2},$$

$$F(S_4) = f_1(S_4) = \frac{0\times1+0\times1+1\times1}{1+1+1} = \frac{1}{3}.$$

Since $F(S_2) > F(S_4)$, S_2 will be selected. Thus $I' = I' - S_2 = \{5,6,7,8\} - \{1,2,5,6\} = \{7,8\}$, and $L_{i'}$=2 (for i'=7,8). The L_7 containing S_3 and S_6, rather than L_8, is chosen because of its smaller subscript.

Then

$$F(S_3) = f_1(S_3) = \frac{0\times1+0\times1+1\times1+1\times1}{1+1+1+1} = \frac{1}{2},$$

$$F(S_6) = f_1(S_6) = \frac{0\times1+0\times1+1\times1}{1+1+1} = \frac{1}{3}.$$

Since $F(S_3) > F(S_6)$, S_3 will be selected. Thus $I' = I' - S_3 = \{7,8\} - \{3,4,7,8\} = f$, and a cover using S_1, S_2 and S_3 is obtained.

It can be easily observed that S_1 is redundant actually: using only S_2 and S_3 can cover all the pieces of work. Therefore shifts added by the *Reconstruction* may be redundant themselves. However, in the next *Evaluation* step, the goodness for S_1, S_2 and S_3 are computed as

$$F(S_1) = f_1(S_1) = \frac{0\times1+0\times1+0\times1+0\times1}{1+1+1+1} = 0,$$

$$F(S_2) = f_1(S_2) = \frac{0\times1+0\times1+1\times1+1\times1}{1+1+1+1} = \frac{1}{2},$$

$$F(S_3) = f_1(S_3) = \frac{0\times1+0\times1+1\times1+1\times1}{1+1+1+1} = \frac{1}{2}.$$

Because of its zero goodness value, S_1 will be discarded in the following *Selection* step, no matter what selection threshold is generated.

According to the above example, it can be noticed that the function value $F(S_j)$ for each shift S_j in the *Reconstruction* step is continuously updated, for example $F(S_2)=1$ in the first iteration and $F(S_2)=1/2$ in the second iteration. Moreover, the greedy function $F(S_j)$ in *Reconstruction* and the goodness function $F(S_j)$ in *Evaluation* are using the same formulae in two contrasting contexts: the former one is a local function to decide the next move, while the latter is a global function to give an overall evaluation. That is the reason why the goodness of shift S_1 becomes zero in the next *Evaluation* step.

5.4 Computational results

The above evolutionary approach was coded in Borland C++. All problems were run on the same Pentium II 333 MHz with 196 megabyte RAM personal computer using Windows 98 operating system. If no improvement has been achieved for 1000 iterations, the program will terminate. Furthermore, p_m in *Mutation* of SE is set to be 5.0%, and size k of RCL in *Reconstruction* is set to be 2. Elapsed time is the time following the solution of the relaxed LP of TRACS II.

The SE approach combines the two main objectives, minimising the total cost and the number of shifts in a schedule, in a weighted-sum cost function, i.e. minimizing $\sum_{j^*=1}^{l}(c_{j^*}+2000)$, where l is number of shifts in the schedule and c_{j^*} is the cost of shift S_{j^*}. Since in most driver scheduling problems the first objective is to minimize the number of shifts, a large constant of 2000 per shift gives priority to this. The best known schedules are obtained by either the TRACS II system (Proll 1997; Fores et al. 1999) or the hybrid GA by Kwan et al (1999; 2000) incorporating strong domain knowledge.

In fact, a problem exists with the above weighted sum function if the Pareto surface is non-convex (Steuer 1986; Ulungu 1994). Furthermore, the determination of weights per shift will be difficult. Therefore Section 4.3.3.2 proposes a fuzzy goal-directed search approach for the hybrid GA with some improved results, where the best schedule is the one that satisfies a vector of fuzzy goals as much as possible. However, since the best known solutions were all obtained using a weighted-sum objective function, the SE approach used the same simple weighted-sum objective function for benchmark comparisons.

Two sets of experiments have been implemented for the SE algorithm. The first set is to use an initial solution generated by a simple greedy heuristic as the SE's input, the purpose of which is to demonstrate the feasibility of using the SE algorithm in driver scheduling. The benchmark results in terms of shift number and total cost, including the qualities of the initial solution and the final SE's solution for individual cases, are compiled in table 5.1. In this experimental set, the same weight distribution of membership functions, W=(0.20, 0.10, 0.10, 0.20, 0.40), is applied to all these thirteen cases.

Data	SE's initial schedule				SE's final schedule				
	Shifts	RPD	Cost (h)	RPD	Shifts	RPD	Cost (h)	%	Time (s)
Colx	44	29.41	382.62	32.78	35	2.94	294.85	2.32	25
Gmb	42	23.53	359.93	24.41	36	5.88	296.82	2.59	14
Neur	75	20.97	643.07	26.28	63	1.61	509.77	0.10	131
Ews	128	10.34	1241.78	23.74	118	1.72	1006.41	0.28	72
Wag3	59	18.00	495.66	22.86	51	2.00	415.60	3.02	22
F2x	70	9.38	693.72	23.39	64	0.00	572.23	1.78	320
Tram	61	24.49	564.12	34.47	50	2.04	430.78	2.69	17
Trmx	63	28.57	571.50	39.91	50	2.04	422.28	3.38	69
Nb2	84	12.00	1006.67	18.28	77	2.67	836.83	-1.68	189
Gall2	291	20.25	2946.22	31.09	243	0.41	2263.33	0.70	776
G532	323	17.03	2537.32	21.80	276	0.00	2149.25	3.17	132
Gall1	402	15.19	3308.95	24.34	344	-1.43	2695.33	1.29	239
Rrne	471	19.24	4024.17	28.27	397	0.51	3277.33	4.47	969
Avg.	**19.11%**		**27.05%**		**1.57%**		**1.86%**		

Table 5.1: Results of the crude initial and the final SE's schedules

The second set is to use a seed generated by a GA (described in Chapter 4) as the SE's input, the purpose of which is to find how good the SE's final solution would be if a relatively good initial solution is used. The benchmark results in terms of both objectives, including the qualities of the GA's initial solution and the final SE's solution for individual case, are compiled in table 5.2. The GA's population size is set to be 100 for all problems, and the weight distributions of membership functions in the SE are the ones that are used to derive these seeds in the GA.

Data	Initial schedule derived by GA				SE's final schedule				
	Shifts	RPD	Cost (h)	RPD	Shifts	RPD	Cost (h)	RPD	Time (s)
Colx	36	5.88	302.51	4.98	35	2.94	294.06	2.05	24
Gmb	37	8.82	307.33	6.22	35	2.94	294.92	1.94	16
Neur	66	6.45	531.02	4.27	62	0.00	507.67	-0.31	120
Ews	118	1.72	1022.08	1.85	117	0.82	1000.18	-0.34	167
Wag3	51	2.00	416.65	3.28	51	2.00	406.55	0.78	11
F2x	66	3.13	613.86	9.19	62	-3.13	583.28	3.75	530
Tram	51	4.08	442.10	5.39	49	0.00	421.56	0.49	23
Trmx	51	4.08	427.70	4.71	49	0.00	414.38	1.45	59
Nb2	76	1.33	881.92	3.62	74	-1.33	830.60	-2.41	216
Gall2	244	0.83	2287.68	1.79	243	0.41	2250.53	0.13	981
G532	277	0.36	2152.38	3.32	271	-1.81	2104.33	1.02	130
Gall1	350	0.29	2749.32	3.31	343	-1.72	2663.05	0.07	358
Rrne	407	3.04	3399.62	8.36	390	-1.27	3242.75	3.36	1320
Avg.	**3.23%**		**4.63%**		**-0.01%**		**0.92%**		

Table 5.2: Results of the GA's initial and the final SE's schedule

Using the initial solutions derived from GAs, the SE approach has successfully solved two problems which were not solved by TRACS II with better solutions and much faster speed than other heuristics, and has produced superior results for the two larger problems (G532 and Gall1) whose sizes necessitated decomposition for TRACS II. Although the ILP of the latest TRACS II version can now solve the largest problem (Rrne) without decomposition, the SE approach has outperformed it in terms of total shift number.

Computational results show that the results of the new SE approach are very close to that of TRACS II, regardless which kind of initial solution is used as the seed. In particular, compared with all the best known solutions, the SE's solution using the GA's result as the seed is even 0.01% better in terms of total shift number, and is only 0.92% more expensive in terms of total cost on average. However, our results are much faster in general, especially for larger cases.

To test the robustness of the proposed SE algorithm, each data set started from the seed derived from the GA was run ten times by fixing the parameter and varying the pseudo random number seed at the beginning. The summary results are showed in Table 5.3. Comparing the number of shifts in the ten times with the best solutions found before the runs, on average 56.9% of the runs have the same or better results. In terms of solution costs, there is no noticeable variation between the runs. Except the last four larger cases that have higher standard deviations in cost, no obvious trend can be observed.

Data	Number of Shifts					Cost			
	In table 5.2	Distribution of runs				Ave.	Min.	Max.	Std. Dev.
		-1	=	+1	+2 or more				
Colx	35		5	5		294.12	293.76	295.83	1.61
Gmb	35		6	4		295.07	293.82	295.44	1.58
Neur	62		4	4	2	508.33	507.32	510.64	1.47
Ew3	117	1	7	2		1003.37	998.36	1006.41	2.56
Wag3	51		7	3		407.31	405.68	410.56	1.77
F2x	62		4	3	3	584.82	572.23	589.23	3.86
Tram	49		6	4		422.14	420.13	426.65	1.73
Trmx	49		5	5		415.47	413.83	420.21	2.14
Nb2	74		6	2	2	830.44	829.86	835.24	2.87
Gall2	243		9	1		2251.03	2249.35	2265.69	5.85
G532	271		5	2	3	2106.45	2102.32	2122.13	6.13
Gall1	343	1	4	3	2	2663.46	2661.72	2673.41	5.98
Rrne	390	1	3	2	4	3244.16	3238.57	3259.87	7.64

Table 5.3: Results of ten runs with fixed parameters but different random seed numbers

Figure 5.2 and Figure 5.3 respectively depicts the improvement of the schedule from aspects of total cost and shift number versus the number of iterations for the Tram case. Although the actual values may differ among various cases, the characteristic shapes of the curves are similar.

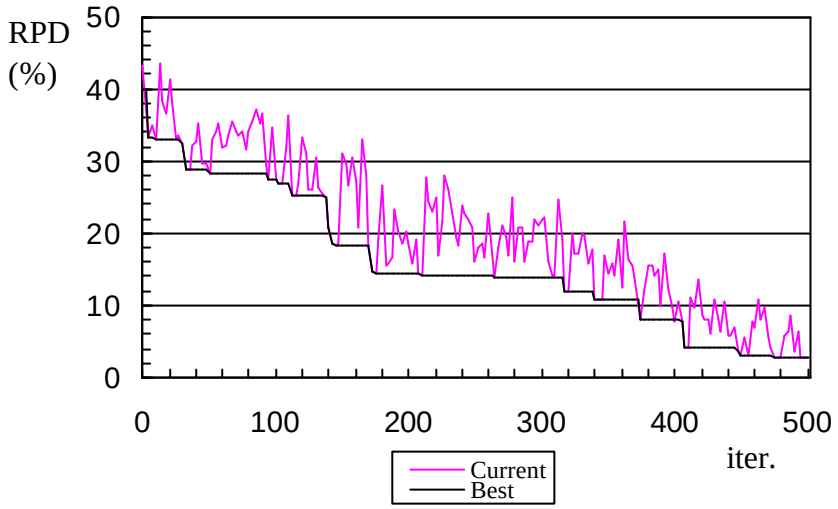


Figure 5.2: RPD of total cost (in current and best schedule respectively) versus iteration number

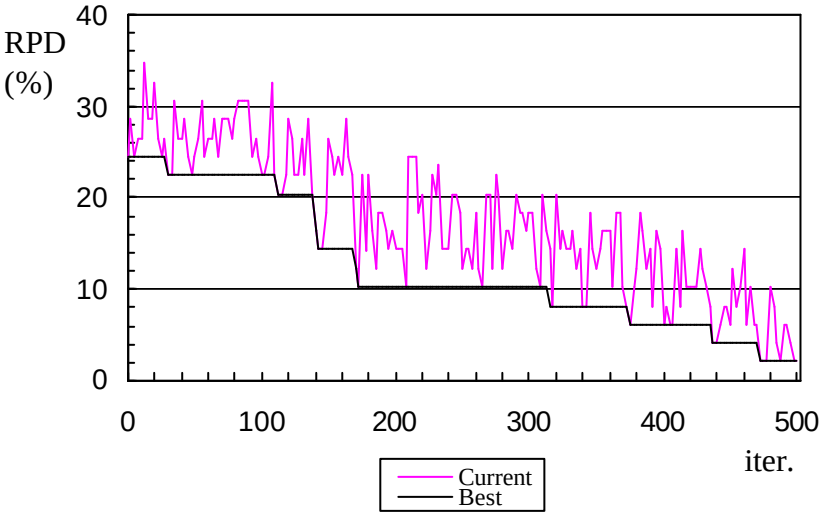


Figure 5.3: RPD of shift number (in current and best schedule respectively) versus iteration number

5.5 Conclusions

A fuzzy evolutionary approach for driver scheduling is presented in this chapter. It incorporates the idea of fuzzy evaluation into a SE algorithm, combining the features of iterative improvement and constructive perturbation, to explore solution space effectively and obtain superior schedules. This new approach is novel because it is the first time that the SE algorithm has been applied to the driver scheduling problem. Experiments with benchmark tests using data from the transportation industry demonstrate the strengths of the proposed algorithm in solving large size driver scheduling problems.

Applying SE algorithms to driver scheduling problems has several advantages. First, it is simple to carry out because it utilizes greedy algorithm and local heuristic. Secondly, due to its characteristics of maintaining only a single schedule at each generation and discarding inferior shifts from this schedule, the SE algorithm converges fast compared with other meta-heuristics. Thirdly, combined with probabilistic hill climbing, the SE achieves superior schedules by effective exploration of the solution space.

Although this work is presented in terms of driver scheduling, it is suggested that, besides general set covering problems (to be introduced in the following Chapter 6), this SE approach could also be applied to many other scheduling problems whose solutions can be decomposed into elements. Even the process of identifying fuzzified criteria could be generalized and applied elsewhere.

Chapter Six

A Fuzzy SE Approach with Taguchi Parameter Setting for the Set Covering Problem

---

6.1 Introduction

Set Covering Problems (SCPs) are difficult zero-one optimization problems, which have been proven to be NP-complete (Garey and Johnson, 1979). The SCP is the problem of finding a subset of the columns of an $m \times n$ zero-one matrix

$$A = \{a_{ij} \in \{0,1\}; i = 1,2,...,m; j = 1,2,...,n\}$$

that covers all rows at a minimum cost, based on a set of costs $\{c_j; j = 1,2,...,n\}$. Defining $x_i = 1$ if column j (with an associated cost c_j >0) is in the solution and x_j =0 otherwise, the SCP is to

$$\text{Minimize} \quad \sum_{j=1}^{n} c_j x_j \tag{6.1}$$

$$\text{Subject to: } \sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad i \in I=\{1, 2,\ldots, m\} \tag{6.2}$$

$$x_j = 0 \text{ or } 1, \quad j \in J=\{1, 2,\ldots, n\} \tag{6.3}$$

Constraint (6.2) ensures that each row is covered by at least one column, and (6.3) requires that whole columns be used. In the case that all the cost coefficients c_j are equal, the problem is called a unicost SCP, and the objective is to minimize the number of columns.

Integer Linear Programming (ILP) is the traditional approach for the SCP. The relaxed Linear Programming (LP) for the SCP, i.e. ignoring the integer constraint, is first solved. Then, the branch-and-bound tree-search procedure is used to obtain the integer solution. For large size SCPs, this method often has computational difficulty in getting an integer solution, although the relaxed LP can usually be solved relatively quickly in polynomial time (Karmarkar 1984).

In this chapter, a new fuzzy evolutionary approach for solving large-scale SCP problems is presented. The approach involves a number of parameters and evaluation weights, the combinations of which are efficiently experimented using Taguchi's orthogonal arrays (Taguchi 1986, Taguchi 1987). Data sets for the bus and rail driver scheduling problem are used to test the performance of this approach.

This chapter reports on improvements in the fuzzy SE approach, and its generalization from driver scheduling to the class of SCP:

- For fuzzy evaluation, the criteria used are now generalized for the SCP without any domain specific knowledge in their formulation.
- In SE's *Evaluation* step, a new evaluation function has been designed to replace the former function that was specialized for the driver scheduling problem.

- In SE's *Construction* step, the function for assessing which columns to be used for repairing the broken solution is different from that in the *Evaluation* step for discarding columns.
- Taguchi's experimental design (Taguchi 1986; Taguchi 1987) is utilized to reliably set the seven parameters in our proposed algorithm. This method uses orthogonal arrays to perform an initial study of the wide range of parameter space, with a small number of experiments.

The method of fuzzy evaluation for the SCP is described first. A fuzzy SE algorithm is then presented. Taguchi's orthogonal experimental design for parameter settings will be briefly introduced. Comparative results using large-scale real world problems are given, which are followed by some conclusions.

6.2 Fuzzy evaluation for set covering

Like the process of forming a schedule, the process of constructing a possible solution by the greedy heuristic is inherently sequential. Among the large number of columns to be chosen, it would be difficult to judge which one is more effective than others because the criteria bear some uncertainty. Fuzzy set theory (Zadeh 1965) is therefore used to assign each column a quantitative value according to its structural state. The fitter the structure for a column, the larger its value is.

6.2.1 Construction of the factor set

The main factors concerning the column's structure are the number of rows it covers (u_1), its cost (u_2), the ratio of its number of covered rows to its cost (u_3), and the average coverage number (number of columns covering the row) of all the rows covered by this column (u_4). Furthermore, since the relaxed LP in the set covering model can be solved relatively easily,

the fractional cover of the relaxed LP solution, if the column is included in this solution, is used as the fifth factor (u_5).

The formation of these five factors is very similar to that used in driver scheduling. However, all the factors that involve domain knowledge of driver scheduling (i.e. u_2, u_3, and u_4) have been removed. New factors particularly relevant to the general set covering problem have been added.

6.2.1.1 Factor u_1

The objective for set covering is to minimize the total cost of the solution. However in many real-world problems, the total cost is usually increased with the number of columns in the solution. To reduce the number of columns used as more as possible, the number of rows each column covers should be taken into account as a criterion about the column structure.

Considering two columns that cover different number of rows, one column should be more efficient than another because it covers more rows and thus potentially minimises the number of columns in the final solution. If every column in the solution covers as many rows as possible, it can be concluded that, in this solution, the number of column is possibly fewer thus leading to a lower total cost. Therefore, similar to the function for driver scheduling, an associated membership function can be designed based on the assumption that the goodness of a column $j (j \in J)$ generally increases with the number of rows it covers, denoted as b_j.

The membership functions $m_{j1} (j \in J)$ for factors u_1 is defined as

$$m_{j1} = \begin{cases} 2\left(\dfrac{x_{j1} - c_{j1}}{b_{j1} - c_{j1}}\right)^2, & c_{j1} \le x_{j1} < \dfrac{b_{j1} + c_{j1}}{2} \\ 1 - 2\left(\dfrac{x_{j1} - b_{j1}}{b_{j1} - c_{j1}}\right)^2, & \dfrac{b_{j1} + c_{j1}}{2} \le x_{j1} \le b_{j1} \end{cases}, \forall j \in J, \quad (6.4)$$

where $x_{j1} = b_j = \sum_{i=1}^{m} a_{ij}$;

b_{j1} = maximum number of rows;

c_{j1} = minimum number of rows.

6.2.1.2 Factor u_2

To achieve the objective of minimum total cost, the cost for individual column in the final solution should be as small as possible. Therefore, factor u_2 is based on the assumption that the goodness of a column $j (j \in J)$ generally decreases with its cost c_j. Compared with the non-linear membership function used for the driver scheduling problem, a linear functions defined as $m_{j2} (j \in J)$ below seems to perform better.

$$m_{j2} = \frac{b_{j2} - x_{j2}}{b_{j2} - c_{j2}}, \forall j \in J, \quad (6.5)$$

where $x_{j2} = c_j$;

b_{j2} = maximum cost;

c_{j2} = minimum cost.

6.2.1.3 Factor u_3

According to the above analysis for factor u_1 and factor u_2, to achieve a satisfactory solution, the ideal columns to be used would be those covering as many rows as possible, and at the same time having the cost as small as possible. Unfortunately, in practice these two factors are often contradictory because a column covering more rows usually has a larger cost.

To get a balance between the number of covered rows and the cost for a column, besides these absolute values, the relative ratio of number of covered rows to its cost can be regarded as the third important criterion. Based on the assumption that the goodness of a column $j(j \in J)$ generally increases with the ratio of the number of covered rows to its cost, denoted as b_j / c_j, the membership function $m_{j3}(j \in J)$ for factor u_3 is therefore defined as

$$m_{j3} = \begin{cases} 2\left(\dfrac{x_{j3} - c_{j3}}{b_{j3} - c_{j3}}\right)^2, & c_{j3} \le x_{j3} < \dfrac{b_{j3} + c_{j3}}{2} \\ 1 - 2\left(\dfrac{x_{j3} - b_{j3}}{b_{j3} - c_{j3}}\right)^2, & \dfrac{b_{j3} + c_{j3}}{2} \le x_{j3} \le b_{j3} \end{cases}, \forall j \in J, \qquad (6.6)$$

where $x_{j3} = b_j \Big/ c_j = \sum_{i=1}^{m} a_{ij} \Big/ c_j$;

b_{j3} = maximum ratio;

c_{j3} = minimum ratio.

6.2.1.4 Factor u_4

Considering all the columns with respect to all the rows to be covered, each row i has an associated coverage list containing L_i columns that are able to cover i, where

$$L_i = \sum_{j=1}^{n} a_{ij}, i \in I . \tag{6.7}$$

Normally each column covers at least one row, and the number of rows contained in column j is $\sum_{i=1}^{m} a_{ij}$. Therefore, for all the rows in column j, the total coverage number is $\sum_{i=1}^{m} (a_{ij} \times \sum_{j=1}^{n} a_{ij})$, and the average coverage number, denoted as a_j, is formulated as

$$a_j = \sum_{i=1}^{m} (a_{ij} \times \sum_{j=1}^{n} a_{ij}) \Big/ \sum_{i=1}^{m} a_{ij}, j \in J . \tag{6.8}$$

The average coverage number is an index about whether the components in a column are heavily covered by other columns in general. To find an economic cover, it is reasonable to regard the columns whose rows are heavily covered by others as to be less important. Based on this assumption, the goodness of column $j (j \in J)$ generally decreases with the average coverage number of all the rows covered by column j. Thus the membership functions $m_{j4} (j \in J)$ for factors u_4 is defined as

$$m_{j4} = \frac{b_{j4} - x_{j4}}{b_{j4} - c_{j4}}, \forall j \in J , \tag{6.9}$$

where $x_{j4} = a_j$;

b_{j4} = maximum average coverage number;

c_{j4} = minimum average coverage number.

6.2.1.5 Factor u_5

A popular method for column selection is ILP, which is NP-hard (Garey and Johnson, 1979). Large problems would have to be divided into sub-problems, and in some cases the ILP process may have difficulties in finding an integer solution. In contrast, the relaxed fractional problem in which the solution vector is not required to be integral, $X \in [0,1]^n$, is much easier: the optimal solution for the relaxed problem can be found in polynomial time (Karmarker 1984). In addition, Srinivasan (1995) showed that the approximation guarantee for the Randomised Rounding Algorithm (RRA) on fractional covers is

$$c_{RRA} \leq c_{Min}\left(\ln(\frac{m}{c_{Min}}) + \ln\ln(\frac{m}{c_{Min}}) + O(1)\right). \tag{6.10}$$

Here c_{RRA} is the number of sets in the subcover output by the RRA, and c_{Min} is the optimum value of the relaxed LP for the set covering problem. Although Slavík (1996) proved the performance guarantee for the RRA was even worse than that of the simple greedy algorithm, it still can be concluded that at least the relaxed solution provides some useful information about the distribution of the optimal integer solution. For the same reason described in Section 3.4.1.5, the membership function $m_{j5}(j \in J)$ for the fractional cover factor can be defined as

$$m_{j5} = \begin{cases} e^{-\frac{(x_{j5}-a)^2}{b}}, & \text{if column } j \text{ is in the fractional cover;} \\ 0, & \text{otherwise.} \end{cases} \tag{6.11}$$

Let $m_{j5} = 1$ when $x_{j5} = b_{j5}$, and $m_{j5} = 0.01$ when $x_{j5} = c_{j5}$,

where x_{j5} = fractional value of column j in the relaxed LP solution;

b_{j5} = maximum value in fractional cover;

c_{j5} = minimum value in fractional cover.

Therefore,

$$\begin{cases} a = b_{j5} \\ b = -\dfrac{(b_{j5} - c_{j5})^2}{\ln 0.01} \end{cases} \tag{6.12}$$

6.2.2 Fuzzy evaluation

As described in Section 3.5.2, three aggregation operators in different categories (namely intersection operator, union operator, arithmetic mean operator) have been investigated for the driver scheduling problem, among which the arithmetic mean operator takes into account the compensative effect of individual factors, and performs best in general. Because of the similarity of these two problems, the arithmetic mean operator is applied again to aggregate all proposed criteria for set covering.

Therefore, for column $j(j \in J)$, the formulation of its structural coefficient $f_1(j)$ by the method of fuzzy evaluation is

$$f_1(j) = \sum_{k=1}^{5} (w_k \times m_{jk}), \forall j \in J\ . \tag{6.13}$$

Where $w_k (w_k \geq 0)$ denotes the corresponding weights for factor u_k (k=1, 2, 3, 4, 5), satisfying

$$\sum_{k=1}^{5} w_k = 1. \tag{6.14}$$

The main task for the fuzzy evaluation model is to find a suitable weight distribution among the fuzzy membership functions. These five weights, along with two other parameters to be given in Section 6.3.3 and 6.3.4, could be determined by Taguchi's orthogonal experimental design, which will be described in Section 6.4.

6.3 A fuzzy Simulated Evolution algorithm

In this section, a fuzzy SE algorithm is applied to mimic generations of evolution on a single solution. It executes a sequence of *Construction*, *Evaluation*, *Selection* and *Mutation* steps in a loop until a user specified parameter (e.g. cpu-time, or the solution cost) is reached or no improvement has been achieved for a number of iterations. Throughout the evolution, the currently best solution is retained and finally returned as the final solution.

6.3.1 Construction

The *Construction* step takes a partial solution as the input, and produces a complete solution as the output. All the existing column assignments in the partial solution remain unaffected. Therefore, the *Construction* step is to assign columns to all the uncovered rows to complete a partial solution. Note that the partial solution in the first iteration of the loop is set to be empty.

Considering all columns with respect to all rows to be covered, each of the remaining unassigned rows i has a coverage list of length L_i, i.e. containing L_i possible columns that can cover it. The greed-based constructor assumes that the desirability of adding column $j (j \in J)$ into the partial solution generally increases with its function value $F'(j)$, which can be formulated as

$$F'(j) = f_1(j) \times \sum_{i \in I'} a_{ij}, \forall j \in J . \quad (6.15)$$

Where $f_1(j)$ is the structural coefficient defined as formula (6.13), and I' is the set of rows to be covered. However, to introduce diversification, one of the candidates, not necessarily the top candidate, is randomly selected from a Restricted Candidate List (RCL) consisting of columns with r largest function values $F'(j)$. From empirical studies we find that $r \leq 4$ achieves better solutions.

Let $J^* = \{1,2,\ldots,t\}$ the set of columns in a partial solution, and $S_j = \{i \mid a_{ij} = 1, i \in I\}$ the set of rows covered by column j, the steps to generate a complete solution are:

Step 1 Set $I' = I - \bigcup (S_{j^*} : j^* \in J^*)$.

Step 2 If $I' = f$ then stop: J^* is a complete solution and $C(J^*) = \sum (c_{j^*} : j^* \in J^*)$. Otherwise locate a row $t' \in I'$ having $L_{t'} = \min(L_{i'} : i' \in I')$, and then randomly select a column $S_k, k \in \{1,\ldots,r\}$, within RCL from the coverage list of row t'. Proceed to step 3.

Step 3 Add k to J^*, set $I' = I' - S_k$, and return to step 2.

Before the *Construction*, some rows may already be over-covered, i.e. covered more than once by the existing columns in the partial solution. The other columns added by the *Construction* step are each chosen to cover at least one currently uncovered row, but they increase the amount of over-cover as well. Thus some columns might become redundant later, causing all their rows covered by other columns. It should be pointed out that, in the next *Selection*, these redundant columns will be removed automatically because of their zero goodness.

A similar construction phase has been considered for the timetabling problem, under a number of different names of “Peckish” in (Corne and Ross, 1996) or “Bias Selection” in (Burke et al., 1998). The main difference for the proposed *Construction* step is it uses a more comprehensive evaluation function to direct the searching process.

6.3.2 Evaluation

Similar to the Evaluation of SE for driver scheduling (Described in section 5.3.3), the overall evaluation function $F(j^*)$ consists of two parts: structural coefficient $f_1(j^*) \in [0,1]$ and over-cover penalty $f_2(j^*) \in [0,1]$, which can be formulated as

$$F(j^*) = f_1(j^*) \times f_2(j^*), \forall j^* \in J^*. \tag{6.16}$$

The ratio of the non-overlapped number of rows to total number of rows in $j^* (j^* \in J^*)$ is also regarded as an important criterion, which can be formulated as $f_2(j^*)$ below:

$$f_2(j^*) = \sum_{i=1}^{m} b_{ij^*} \Big/ \sum_{i=1}^{m} a_{ij^*}, \forall j^* \in J^*, \tag{6.17}$$

where

$$b_{ij^*} = \begin{cases} 0, & \sum_{j^* \in J^*} a_{ij^*} > 1; \\ 1, & \text{otherwise.} \end{cases} \tag{6.18}$$

If every row in j^* has been covered by one or more other columns in J^* as well, then $f_2(j^*) = 0$; conversely if none of the rows in j^* is over-covered, $f_2(j^*) = 1$.

6.3.3 Selection

In this step it will be determined whether a column $j^*(j^* \in J^*)$ is retained for the next generation, or discarded and placed in a queue for the new allocation. This is done by comparing its goodness $F(j^*)$ to $(p_s - k_s)$, where p_s is a random number generated for each generation in the range [0, 1], and k_s is a value smaller than 1.0. If $F(j^*) > (p_s - k_s)$ then j^* will survive in its present position; otherwise j^* will be removed from the current evolutionary solution. The rows it covers, except those also covered by other columns in the solution, are then released for the next *Construction*. By using this *Selection* process, column j^* with larger goodness $F(j^*)$ has higher probability of survival in the current solution.

The purpose of subtracting $k_s \in [0,1]$ from p_s is to improve the SE's convergence capability. A suitable setting of the selection value k_s is important to the algorithm's performance.

6.3.4 Mutation

Following the *Selection* step, each retained column $j^*(j^* \in J^*)$ has a chance to be mutated, i.e. randomly discarded from the partial solution at a given rate of p_m, and releases its covered rows, except those also covered by other retained columns, for the next generation. The mutation rate p_m should be much smaller than the selection rate to guarantee convergence. Like the selection valve k_s, p_m is also an influencing parameter in the SE.

6.4 Taguchi method for parameter design

Seven parameters are investigated in the proposed algorithm, namely weight w_k for criteria u_k (k=1, 2, 3, 4, 5) in the fuzzy evaluation model, selection value k_s in the *Selection* step, and

mutation rate p_m in the *Mutation* step. These parameters will influence the SE's performance significantly, and are difficult to determine. Common approaches for parameter design lead either to a long time span for trying out all combinations, or to a premature termination of the design process with results far from optimal in most cases.

As described in Chapter 4, a Genetic Algorithm (GA) was presented in the earlier stage of this research (Li and Kwan, 2002a) to calibrate the fuzzy weight distribution, and some good solutions were obtained as by-products. However, the weights so obtained may not be always good for the SE even though the results were rather satisfactory in some cases (Li and Kwan 2001a and 2001b). The first reason is that GA and SE are evolutionary algorithms with very different mechanisms, therefore a good weight distribution under GA may not be always suitable for SE. The second reason is due to the different construction methods: the one used in GA is deterministic rather than randomized as is in SE.

For full experimentation on the fuzzy SE algorithm, the first six parameters at five value levels each would require 15,625 (5^6) possible experimental evaluations, each of which is a process of iterative improvement and thus is quite time-consuming. The time to conduct such a detailed search for the optimal solution is prohibitive. Naturally, it is desirable to reduce the number of experiments to a practical point, and still reach satisfactory solution. For the problem of choosing appropriate parameter configurations, Taguchi's orthogonal experimental design provides an effective solution.

6.4.1 Preliminaries

Orthogonal experimental design for parameter optimisation provides a systematic and efficient approach to determine near optimal parameter settings. The objective is to select the best combination of control factors (parameters) so that the product or process is most robust with respect to noise factors. The orthogonal experimental design applies orthogonal arrays

from experimental design theory to study a large number of variables with a small number of trials, significantly reducing the number of experimental configurations. Moreover, in case that parameter-interaction space is relatively smooth, the conclusions drawn from such small-scale experiments are valid over the entire experimental region spanned by the control factors and their settings.

6.4.1.1 Orthogonal Array

Orthogonal arrays are a special set of Latin squares (Dénes and Keedwell, 1974), constructed by Taguchi to lay out the experimental design. In this array, the columns are said to be mutually orthogonal or balanced. That is, for any pair of columns, all combinations of factor levels occur, and occur an equal number of times. By using the table, the required experimental situations are defined. Consider a 3-level and 4-factor orthogonal array shown in Table 6.1 below:

	A	B	C	D
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Table 6.1: Orthogonal array $L_9(3^4)$

The array is designated by the symbol $L_9(3^4)$, involving four factors A, B, C, and D, each at three levels one (1), two (2), and three (3). The array has a size of nine rows and four columns. The numbers (1/2/3) in the row indicate the factor levels and each row represents

specific test characteristics of each experiment. The vertical columns represent the experimental factors to be studied using that array. Each of the columns contains three assignments at each levels (1, 2, or 3) for the corresponding factors. These conditions can combine in nine possible ways (i.e. (1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)) for two factors, with 3^4 possible combinations of levels for all the four factors.

The orthogonal array facilitates the experimental design process by assigning factors to the appropriate columns. In this case, referring to table 1, factors A, B, C, and D are arbitrarily assigned to columns 1, 2, 3, and 4 respectively. From the table, nine trials of experiments are needed, with the level of each factor for each trial-run as indicated in the array. The experimental descriptions are reflected through the condition level. The experimenter may use different designators for the columns, but the nine trial-runs will cover all combinations, independent of column definition. In this way, the orthogonal array assures consistency of the design carried out by different experimenters. The orthogonal array also ensures that factors influencing the quality of solutions are properly investigated and controlled during the initial design stage.

6.4.1.2 Comparison to the traditional method of factorial design

The traditional method of factorial design is to investigate all possible combinations and conditions in an experiment that involves multiple factors. Let A be the number of levels for each factor, and B be the number of factors involved, then the number of possible designs N (number of trials) by this method is

$$N = A^B . \tag{6.19}$$

If the factorial design is implemented for the four 3-level factors in Table 6.1, the total number of trials needed would be a full combination of 81 (3^4) trials, rather than 9 trials by the orthogonal array $L_9(3^4)$.

The reason why by using orthogonal array superior parameter configurations could be found by only a small number of experiments is because of its mutual balance. For example, in Table 6.1, each column contains three ones, three twos, and three threes; and any pair of columns contain all combinations of levels (i.e. (1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)) exactly once. It is the characteristic of mutual balance that guarantees the choice of combinations producing elite solutions.

Furthermore, in the factorial design process, the means of levels combination laid out is not specified, which may lead to different results on the same experimental subject each time a trial is conducted. However, Taguchi's orthogonal array is able to simplify and standardize the factorial design in a manner that will produce consistent and similar results, even though the trials are implemented by different experimenters. Hence, two different investigators will have similar results and a standard design methodology.

The concept of consistent results and standard design methodology through orthogonal array analysis is important, because it allows the experimenter to produce two outcomes of the same quality standards, using the same materials, but with differences in the experimental process. This is possible since, through orthogonal array experimental analysis, the factors influencing the quality of results can be identified, controlled, and subsequently compensated during the early design stage. Thus, the quality of the outcome itself is able to adapt to the experimental process, rather than depends on the experimental process.

In summary, in case that parameter-interaction space is relatively smooth, compared with the traditional factorial design method, Taguchi's orthogonal array is considered to be superior since:

- It is efficient in handling larger numbers of factor variables;

- It can produce similar and consistent results, even though the experiments may be carried out by different experimenters;
- It enables determination of the contribution of each quality-influencing factor.

The limitation of orthogonal arrays is that they can only be applied at the initial design stage. There are some situations where orthogonal array techniques are not available, such as a process involving control factors that vary in time and cannot be quantified exactly.

6.4.2 The approach of orthogonal experimental design

Due to of the characteristic of mutual balance in orthogonal arrays, Taguchi's approach can explore the solution space as extensive as possible. Moreover, due to the following ANOVA process, Taguchi's approach can also exploit the solution space as much as possible. The goal that uses this approach in our application is to determine the best setting for each parameter so that the solution cost is minimized. Basically, this approach consists of the following steps of defining the parameter space, determining the factor levels, and analysis of variance.

6.4.2.1 Definition of the parameter ranges

The first step towards the goal is to define the ranges of seven control factors in the proposed algorithm. Without any pre-knowledge about the influence of the weights w_k (k=1, 2, 3, 4, 5) on the algorithm, it is reasonable to set the levels of w_k evenly over the full applicable range of [0, 1]. However the range for the selection value k_s and the mutation rate p_s should be much narrower, since according to the former experience, the SE usually yields better solutions with $k_s \in [0.20, 0.30]$ and $p_s \in [0.05, 0.06]$ respectively, and these two parameters are relatively independent of w_k.

6.4.2.2 Determination of the factor levels

This step is to define the initial levels of the control factors, and sequentially choose the most suitable orthogonal array. In order to facilitate the description of this step, three definitions are given first as follows:

[Definition 6.1] A factor-level table is a tableau, where each row represents a control factor and each column represents an individual level.

[Definition 6.2] In a factor-level table, degrees of freedom for a factor are the number of levels in this factor minus one, and degrees of freedom for the table are the sum of degrees of freedom for all factors.

[Definition 6.3] Degrees of freedom for an orthogonal array are the number of trials minus one. For example, for orthogonalarray $L_9(3^4)$ in Table 6.1, degrees of freedom are eight.

To gain an intuitive feel for degrees of freedom, consider taking one ball from a box of n balls. Every time we come to take one ball and have a choice, until we come to the last one, and then there is no choice. Thus we have $n-1$ choices, i.e. degrees of freedom.

In theory, the initial levels of the individual control factor can be set arbitrarily, and the associated orthogonal array can be chosen flexibly as well without rigorous regulations. However, in practice two principles below are normally complied with:

1) Degrees of freedom for the factor-level table should be no larger than degrees of freedom for the orthogonal array to be used;
2) The number of factors in the factor-level table should be no larger than the number of columns in the orthogonal array.

To achieve the goal with the necessary precision, there should be as many levels in each factor as possible. However, in this situation, the number of trials needed would be increased explosively. The most suitable orthogonal array is the one that maintains the best balance: it uses the largest number of levels, but conducts the smallest number of trials. Since the number of factors currently investigated has been fixed to seven, the task for the orthogonal experimental design is to determine the largest number of levels for each factor, and the smallest number of rows (trials) in the associated orthogonal array.

Let A be the number of levels for each factor, and L be the number of rows (trials) in the orthogonal array. Normally L is the square of an integer, denoted as

$$L = k^2,\ k \in \{2,3,...,n\}. \tag{6.20}$$

According to the 1st principle above,

$$7 \times (A-1) \le L-1 \Leftrightarrow A \le \frac{L-1}{7} + 1 \Leftrightarrow A \le \frac{k^2-1}{7} + 1 \tag{6.21}$$

$$\Rightarrow \text{if } k = 6, \text{ then } A \le 6, \text{ where } k \text{ is the smallest integer to be satisfied}$$

$$\Leftrightarrow \text{if } L = 36, \text{ then } A \le 6.$$

Therefore the largest number of levels for each factor is 6, the smallest number of trials is 36, and the most ideal orthogonal is array L_{36} (6^7). It is effective to deal with the seven factors using only 36 trials, rather than 279,936 (6^7) experimental trials.

Unfortunately, currently the author has difficulty in finding a readily available $L_{36}(6^7)$ configuration from related literature. To design such an array manually will consume a lot of time, which may be left as the future work of this research. On the contrary, an $L_{25}(5^6)$ can be found easily (Taguchi 1987), which is the optimal design to handle six 5-level factors.

As mentioned above, the design of orthogonal array is flexible. Studying the proposed algorithm, the seventh factor p_s is relatively less important than the others due to the relatively minor role of the *mutation* step in SE. To maintain a balance between necessary precision and number of experiments, factors w_k (k=1, 2, 3, 4, 5) and k_s are finally defined to be 5 levels, and p_s to be 2 levels respectively (shown in Table 6.2). These seven factors are assigned to an $L_{50}(2^1 \times 5^6)$ orthogonal array shown in Table 6.3. This is an economical and efficient design for dealing with these seven factors using only 50 trials, rather than 31,250 ($2^1 \times 5^6$) experimental trials.

Control factors	Levels				
	1	2	3	4	5
1.Weight w_1	0.1	0.3	0.5	0.7	0.9
2.Weight w_2	0.1	0.3	0.5	0.7	0.9
3.Weight w_3	0.1	0.3	0.5	0.7	0.9
4.Weight w_4	0.1	0.3	0.5	0.7	0.9
5.Weight w_5	0.1	0.3	0.5	0.7	0.9
6.*Selection* valve k_s	0.22	0.24	0.26	0.28	0.30
7.*Mutation* rate p_m	0.05	0.06	-	-	-

Table 6.2: Control factors and their levels

Trial No.	Control factors						
	1.w_1	2.w_2	3.w_3	4.w_4	5.w_5	6.k_s	7.p_m
1	0.1(1)	0.1(1)	0.1(1)	0.1(1)	0.1(1)	0.22(1)	0.05(1)
2	0.1(1)	0.3(2)	0.3(2)	0.3(2)	0.3(2)	0.24(2)	0.05(1)
3	0.1(1)	0.5(3)	0.5(3)	0.5(3)	0.5(3)	0.26(3)	0.05(1)
4	0.1(1)	0.7(4)	0.7(4)	0.7(4)	0.7(4)	0.28(4)	0.05(1)
5	0.1(1)	0.9(5)	0.9(5)	0.9(5)	0.9(5)	0.30(5)	0.05(1)
6	0.3(2)	0.1(1)	0.3(2)	0.5(3)	0.7(4)	0.30(5)	0.05(1)
7	0.3(2)	0.3(2)	0.5(3)	0.7(4)	0.9(5)	0.22(1)	0.05(1)
8	0.3(2)	0.5(3)	0.7(4)	0.9(5)	0.1(1)	0.24(2)	0.05(1)
9	0.3(2)	0.7(4)	0.9(5)	0.1(1)	0.3(2)	0.26(3)	0.05(1)
10	0.3(2)	0.9(5)	0.1(1)	0.3(2)	0.5(3)	0.28(4)	0.05(1)
11	0.5(3)	0.1(1)	0.5(3)	0.9(5)	0.3(2)	0.28(4)	0.05(1)
12	0.5(3)	0.3(2)	0.7(4)	0.1(1)	0.5(3)	0.30(5)	0.05(1)
13	0.5(3)	0.5(3)	0.9(5)	0.3(2)	0.7(4)	0.22(1)	0.05(1)
14	0.5(3)	0.7(4)	0.1(1)	0.5(3)	0.9(5)	0.24(2)	0.05(1)
15	0.5(3)	0.9(5)	0.3(2)	0.7(4)	0.1(1)	0.26(3)	0.05(1)
16	0.7(4)	0.1(1)	0.7(4)	0.3(2)	0.9(5)	0.26(3)	0.05(1)
17	0.7(4)	0.3(2)	0.9(5)	0.5(3)	0.1(1)	0.28(4)	0.05(1)
18	0.7(4)	0.5(3)	0.1(1)	0.7(4)	0.3(2)	0.30(5)	0.05(1)
19	0.7(4)	0.7(4)	0.3(2)	0.9(5)	0.5(3)	0.22(1)	0.05(1)
20	0.7(4)	0.9(5)	0.5(3)	0.1(1)	0.7(4)	0.24(2)	0.05(1)
21	0.9(5)	0.1(1)	0.9(5)	0.7(4)	0.5(3)	0.24(2)	0.05(1)
22	0.9(5)	0.3(2)	0.1(1)	0.9(5)	0.7(4)	0.26(3)	0.05(1)
23	0.9(5)	0.5(3)	0.3(2)	0.1(1)	0.9(5)	0.28(4)	0.05(1)
24	0.9(5)	0.7(4)	0.5(3)	0.3(2)	0.1(1)	0.30(5)	0.05(1)
25	0.9(5)	0.9(5)	0.7(4)	0.5(3)	0.3(2)	0.22(1)	0.05(1)
26	0.1(1)	0.1(1)	0.1(1)	0.1(1)	0.1(1)	0.22(1)	0.06(2)
27	0.1(1)	0.3(2)	0.3(2)	0.3(2)	0.3(2)	0.24(2)	0.06(2)
28	0.1(1)	0.5(3)	0.5(3)	0.5(3)	0.5(3)	0.26(3)	0.06(2)
29	0.1(1)	0.7(4)	0.7(4)	0.7(4)	0.7(4)	0.28(4)	0.06(2)
30	0.1(1)	0.9(5)	0.9(5)	0.9(5)	0.9(5)	0.30(5)	0.06(2)
31	0.3(2)	0.1(1)	0.3(2)	0.5(3)	0.7(4)	0.30(5)	0.06(2)
32	0.3(2)	0.3(2)	0.5(3)	0.7(4)	0.9(5)	0.22(1)	0.06(2)
33	0.3(2)	0.5(3)	0.7(4)	0.9(5)	0.1(1)	0.24(2)	0.06(2)
34	0.3(2)	0.7(4)	0.9(5)	0.1(1)	0.3(2)	0.26(3)	0.06(2)
35	0.3(2)	0.9(5)	0.1(1)	0.3(2)	0.5(3)	0.28(4)	0.06(2)
36	0.5(3)	0.1(1)	0.5(3)	0.9(5)	0.3(2)	0.28(4)	0.06(2)
37	0.5(3)	0.3(2)	0.7(4)	0.1(1)	0.5(3)	0.30(5)	0.06(2)
38	0.5(3)	0.5(3)	0.9(5)	0.3(2)	0.7(4)	0.22(1)	0.06(2)
39	0.5(3)	0.7(4)	0.1(1)	0.5(3)	0.9(5)	0.24(2)	0.06(2)
40	0.5(3)	0.9(5)	0.3(2)	0.7(4)	0.1(1)	0.26(3)	0.06(2)
41	0.7(4)	0.1(1)	0.7(4)	0.3(2)	0.9(5)	0.26(3)	0.06(2)
42	0.7(4)	0.3(2)	0.9(5)	0.5(3)	0.1(1)	0.28(4)	0.06(2)
43	0.7(4)	0.5(3)	0.1(1)	0.7(4)	0.3(2)	0.30(5)	0.06(2)
44	0.7(4)	0.7(4)	0.3(2)	0.9(5)	0.5(3)	0.22(1)	0.06(2)
45	0.7(4)	0.9(5)	0.5(3)	0.1(1)	0.7(4)	0.24(2)	0.06(2)
46	0.9(5)	0.1(1)	0.9(5)	0.7(4)	0.5(3)	0.24(2)	0.06(2)
47	0.9(5)	0.3(2)	0.1(1)	0.9(5)	0.7(4)	0.26(3)	0.06(2)
48	0.9(5)	0.5(3)	0.3(2)	0.1(1)	0.9(5)	0.28(4)	0.06(2)
49	0.9(5)	0.7(4)	0.5(3)	0.3(2)	0.1(1)	0.30(5)	0.06(2)
50	0.9(5)	0.9(5)	0.7(4)	0.5(3)	0.3(2)	0.22(1)	0.06(2)

Table 6.3: $L_{50}(2^1 \times 5^6)$ orthogonal array (the values in parenthesis represent the factors levels)

At the same time, it would be interesting to know how good the results derived from the above 50 trials are, without further consideration of all other possible combinations. Because of the characteristic of mutual balance in orthogonal arrays, this performance ratio can be guaranteed by the following theorem in non-parametric statistics (The triplex design group, 1987):

[Theorem 6.1] Suppose random variable X is subject to a probabilistically continuous distribution $F(X)$, and $x_1, x_2, \ldots, x_n$ are simple samples (or random observation values) of X. If $x_1, x_2, \ldots, x_n$ are sorted in ascending order, denoted as $x_1 \le x_2 \le \ldots \le x_n$, then the performance ratio for x_i ($i = 1, 2, \ldots, n$) is

$$E[F(x_i)] = \frac{i}{n+1}. \tag{6.22}$$

In particular,

$$E[F(x_n)] = \frac{n}{n+1}. \tag{6.23}$$

Formula (6.23) means that the best experimental result in these simple samples is probabilistically better than $\frac{n}{n+1}\%$ of all possible results defined in the whole discrete solution space. In this case, the best result by $L_{50}(2^1 \times 5^6)$ is better than 98.04% ($= 50/51$) results of all 31,250 trials.

6.4.2.3 Analysis of variance

The step is to analyse the results obtained from the orthogonal array to achieve the following objectives:

- To evaluate the contribution of individual quality-influencing factors in the experimental design process;

- To obtain the best, or optimum, condition for a process, so that good quality characteristics can be sustained;
- To approximate the response of the design parameters under the optimum conditions.

The contribution of individual quality-influencing factors is crucial to the control enforced on the experimental design. A statistical method, Analysis of Variance (ANOVA), is commonly used to analyse the results of the orthogonal experimental design, and to determine how much variation each factor has contributed. By studying the main effects of each factor, the general tendencies of the influencing factors can be characterized. The characteristics can be controlled, such that a lower, or a higher, value in a particular factor produces the preferred result. Thus, the levels of influencing factors to produce the best results can be predicted. For more details regarding the ANOVA method for orthogonal array, (Taguchi 1987) can be referred to.

Since the main purpose of this paper is to test the suitability of the proposed approach for the set covering problem, the author only performs an initial investigation about the wide range of parameter settings, and uses orthogonal experimental design to find a suitable, but coarse, range of the control factors. Therefore, this research simply chooses the parameter configuration from Table 6.3 that leads to the best results, and skips the follow-on process of ANOVA and further experiments.

6.5 Computational results

The algorithm presented in this paper was coded in Borland C++, and run on a Pentium II 333 MHz machine with 196 megabyte RAM using the Windows 98 operating system. To test the proposed algorithm, eight real-world large size set covering problems originating from the public transport industry are solved. Problem instances prefixed by B are bus problems, and T

are train problems. Details of these test problems, including the number of rows, number of columns, and density (percentage of ones in the a_{ij} matrix), are given in Table 6.4.

Data	Rows	Columns	Density (%)	Best known solutions		
				Cover Size	Cost (hours)	Elapsed time (seconds)
T1*	340	29380	1.90	62	509.25	955
T2	437	25099	1.26	116	1003.55	69
T3	546	43743	1.80	64	562.22	>40000
B1*	613	22568	1.58	75	851.09	452
T4	707	144339	0.51	242	2247.52	>80000
T5	1164	29465	0.36	276	2083.15	>80000
T6	1495	28639	0.30	349	2661.12	>80000
T7	1873	50000	0.27	395	3137.20	>80000

Table 6.4: Details of the test problems and related best known solutions

Note: Results of cases marked by asterisks are obtained by the hybrid GA, while others are obtained by the specialized set covering ILP.

The best known schedules are mainly obtained by the TRACS II system (Fores et al., 1999), a commercial system based on ILP. In cases (marked by asterisk) that TRACS II has difficulty in finding solutions, results achieved by hybrid Genetic Algorithms incorporating strong domain knowledge (Kwan et al., 2000) are cited. Since the purpose of the research in this chapter is to investigate the performance of the proposed approach for general set covering problems, the objective for comparison is the minimum cost, and the number of columns is only given for information. Note that these best known solutions are obtained under a balance between objectives of minimising number of shifts and total cost, and therefore may be further improved if the minimum total cost is regarded as the only objective.

To give fair comparison of the computational results, each test problem was run by using the same pseudo random number seed at the beginning of the program. The best parameter

settings are those producing the smallest solution cost among the 50 trials of each orthogonal design.

To save the computational time, in the process of applying the orthogonal experimental design of $L_{50}(2^1 \times 5^6)$, iteration number of the SE algorithm is set to be 200 for all problems. Due to the capability of relatively fast convergence, the parameter sets that produce the smallest solution cost within the shorter iteration of 200 are used to carry out further SE searches and obtain the final solutions. If no improvement has been achieved for 1000 iterations, the program will be terminated. Since the proposed algorithm, the hybrid GA, and the branch-and-bound phase of the ILP process all take the relaxed LP solutions as starting points, the elapsed time for them are compared.

6.5.1 Experiment 1

As described above, due to the characteristic of mutual balance, the orthogonal array is used to study the wide range of parameter space by a small number of experiments. Using the instance of B1, Figure 6.1 gives full demonstration about the variation of solutions for the 50 trial-runs defined by $L_{50}(2^1 \times 5^6)$.

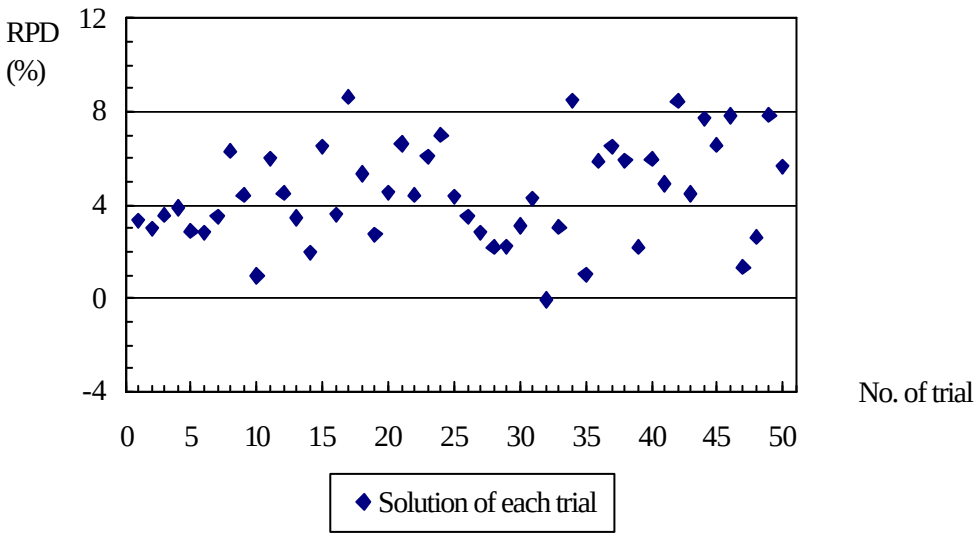


Figure 6.1. RPD of solution cost versus trial number in $L_{50}(2^1 \times 5^6)$

Table 6.5 shows the summary results of 50 trials by parameters associated with the orthogonal array $L_{50}(2^1 \times 5^6)$, and Table 6.6 shows the summary results of 50 trials by randomised parameter sets. The indices of Maximum, Minimum, Mean, and Standard Deviation in statistics are applied to study the distribution of the experimental results in terms of solution cost. Let t_i be the variable of solution cost, then

$$\text{Minimum} = \min\{t_1, ..., t_{50}\}, \tag{6.24}$$

$$\text{Maximum} = \max\{t_1, ..., t_{50}\}, \tag{6.25}$$

$$\text{Mean } \bar{t} = \sum_{i=1}^{50} t_i \Big/ 50, \tag{6.26}$$

$$\text{Standard Deviation} = \sqrt{\sum_{i=1}^{50} (t_i - \bar{t})^2 \Big/ 49}. \tag{6.27}$$

Data	Minimum		Maximum		Mean		Standard Deviation
	Cost	RPD(%)	Cost	RPD(%)	Cost	RPD(%)	Cost
T1	529.28	3.93	571.52	12.23	549.13	7.83	10.97
T2	1004.55	0.10	1074.17	7.04	1026.12	2.25	15.23
T3	586.68	4.35	663.72	18.05	628.42	11.77	17.34
B1	850.60	-0.06	924.45	8.62	889.45	4.51	18.26
T4	2269.07	0.96	2490.70	10.82	2329.33	3.64	56.83
T5	2158.53	3.62	2322.77	11.50	2222.58	6.69	37.46
T6	2764.55	3.89	2989.70	12.35	2855.26	7.30	46.13
T7	3321.70	5.88	3636.07	15.90	3514.04	12.01	64.76
Ave.		**2.83%**		**12.06%**		**7.00%**	**33.37**

Table 6.5: Summary results of 50 trials by $L_{50}(2^1 \times 5^6)$

Data	Minimum		Maximum		Mean		Standard Deviation
	Cost	RPD(%)	Cost	RPD(%)	Cost	RPD(%)	Cost
T1	535.73	5.20	570.72	12.07	547.45	7.50	10.19
T2	1017.68	1.41	1070.13	6.63	1024.04	2.04	12.72
T3	594.35	5.71	660.48	17.48	627.38	11.59	16.02
B1	869.33	2.14	912.33	7.20	883.68	3.83	16.96
T4	2291.73	1.97	2461.62	9.53	2325.59	3.47	49.34
T5	2160.42	3.71	2292.80	10.06	2219.40	6.54	30.42
T6	2798.55	5.16	2952.35	10.94	2854.21	7.26	42.15
T7	3460.3	9.98	3606.45	14.95	3502.40	11.64	63.60
Ave.		**4.41%**		**11.10%**		**6.73%**	**30.18**

Table 6.6: Summary results of 50 trials by randomised parameter sets

The Mean is a measure to evaluate the average performance of the proposed algorithm, while the Standard Deviation is a summary measure of the differences of each result from the mean. According to the RPD results in Table 6.5 and Table 6.6, the average RPD of Minimum, Maximum, and Standard Deviation of $L_{50}(2^1 \times 5^6)$ are 2.83%, 12.06%, and 33.37 respectively, while those produced by random parameters are 4.41%, 11.10%, and 30.18 respectively. Compared with results achieved by randomised parameters, on average results using orthogonal arrays have better Minimum, worse Maximum, and larger Standard Deviation. It demonstrates that results achieved by the orthogonal array are more evenly scattered throughout the solution space, some of which are inevitably elite. Therefore, it is not necessary to study the full solution space for a near-optimal solution.

On the other hand, it can be noticed that some Minimum values in Table 6.6 are also similar to those in Table 3.11. However, the chance of using randomised parameters to produce elite solutions is purely random, while the chance for using an orthogonal array to find these elite solutions is more predictable.

The final cost values of these test problems are compiled in Table 6.7. Computational results show that the solutions derived by the SE approach are very close to those of the previous best known solutions. The negative percentage deviation indicates the percentage improvement from the previous best known solution. In terms of cover size, our results are 0.92% larger on average. But in terms of solution cost, our results are only 0.02% larger on average: in 4 out of the 8 test problems, the SE-based heuristic has generated better cost values.

Data	SE's final solution				
	Cover Size	RPD (%)	Cost (hours)	RPD (%)	Elapsed time (seconds)
T1	65	4.84	507.53	-0.34	520
T2	118	1.72	994.90	-0.86	161
T3	66	3.13	565.38	0.56	730
B1	75	0.00	819.68	-3.69	168
T4	243	0.41	2246.32	-0.05	1398
T5	273	-1.09	2082.77	0.68	286
T6	345	-1.15	2674.18	0.49	316
T7	393	-0.51	3243.11	3.38	2482
Ave.		**0.92%**		**0.02%**	

Table 6.7: Comparative results

In term of the elapsed time, compared with those of other approaches, it is obvious that in general our results are obtained in much faster speed, particularly for larger cases.

In addition to finding the best solutions, another task for the first orthogonal experimental design is to explore whether there exists a generally good pattern of parameter setting. According to the experiments using orthogonal array $L_{50}(2^1 \times 5^6)$, in all cases, the best result is produced by the same parameter configuration of (0.3, 0.9, 0.1, 0.3, 0.5, 0.28, 0.06) for w_k (k=1, 2, 3, 4, 5), k_s, and p_m respectively.

6.5.2 Experiment 2

Unlike the ranges of selection value k_s and mutation rate p_s, the levels of weights w_k (k=1, 2, 3, 4, 5) are evenly set over the full applicable range of [0, 1] without the pre-knowledge about the influence of w_k on the algorithm (described in the beginning of Section 6.4.2). According to the experimental results in Section 6.5.1, it is a very surprising finding that out of 50 different parameter sets, one set gave the best results in all cases. Naturally, one might be interested to know whether those results in Table 6.7 could be improved further by simply implementing another orthogonal experimental design, which uses the same $L_{50}(2^1\times5^6)$ orthogonal array, but is carried out in narrower ranges of values for the factors.

Table 6.8 shows the definition of the control factors and their levels in the narrower ranges, which are centred respectively on the parameter configuration of (0.3, 0.9, 0.1, 0.3, 0.5, 0.28, 0.06) found above. These seven factors are assigned to an $L_{50}(2^1\times5^6)$ orthogonal array shown in Table 6.9.

Control factors	Levels				
	1	2	3	4	5
1. Weight w_1	0.20	0.25	0.30	0.35	0.40
2. Weight w_2	0.80	0.85	0.90	0.95	1.00
3. Weight w_3	0.00	0.05	0.10	0.15	0.20
4. Weight w_4	0.20	0.25	0.30	0.35	0.40
5. Weight w_5	0.40	0.45	0.50	0.55	0.60
6. *Selection* valve k_s	0.270	0.275	0.280	0.285	0.290
7. *Mutation* rate p_m	0.055	0.060	-	-	-

Table 6.8: Control factors and their levels in the narrower ranges

Trial No.	Control factors						
	1.w_1	2.w_2	3.w_3	4.w_4	5.w_5	6.k_s	7.p_m
1	0.20(1)	0.80(1)	0.00(1)	0.20(1)	0.40(1)	0.270(1)	0.055(1)
2	0.20(1)	0.85(2)	0.05(2)	0.25(2)	0.45(2)	0.275(2)	0.055(1)
3	0.20(1)	0.90(3)	0.10(3)	0.30(3)	0.50(3)	0.280(3)	0.055(1)
4	0.20(1)	0.95(4)	0.15(4)	0.35(4)	0.55(4)	0.285(4)	0.055(1)
5	0.20(1)	1.00(5)	0.20(5)	0.40(5)	0.60(5)	0.290(5)	0.055(1)
6	0.25(2)	0.80(1)	0.05(2)	0.30(3)	0.55(4)	0.290(5)	0.055(1)
7	0.25(2)	0.85(2)	0.10(3)	0.35(4)	0.60(5)	0.270(1)	0.055(1)
8	0.25(2)	0.90(3)	0.15(4)	0.40(5)	0.40(1)	0.275(2)	0.055(1)
9	0.25(2)	0.95(4)	0.20(5)	0.20(1)	0.45(2)	0.280(3)	0.055(1)
10	0.25(2)	1.00(5)	0.00(1)	0.25(2)	0.50(3)	0.285(4)	0.055(1)
11	0.30(3)	0.80(1)	0.10(3)	0.40(5)	0.45(2)	0.285(4)	0.055(1)
12	0.30(3)	0.85(2)	0.15(4)	0.20(1)	0.50(3)	0.290(5)	0.055(1)
13	0.30(3)	0.90(3)	0.20(5)	0.25(2)	0.55(4)	0.270(1)	0.055(1)
14	0.30(3)	0.95(4)	0.00(1)	0.30(3)	0.60(5)	0.275(2)	0.055(1)
15	0.30(3)	1.00(5)	0.05(2)	0.35(4)	0.40(1)	0.280(3)	0.055(1)
16	0.35(4)	0.80(1)	0.15(4)	0.25(2)	0.60(5)	0.280(3)	0.055(1)
17	0.35(4)	0.85(2)	0.20(5)	0.30(3)	0.40(1)	0.285(4)	0.055(1)
18	0.35(4)	0.90(3)	0.00(1)	0.35(4)	0.45(2)	0.290(5)	0.055(1)
19	0.35(4)	0.95(4)	0.05(2)	0.40(5)	0.50(3)	0.270(1)	0.055(1)
20	0.35(4)	1.00(5)	0.10(3)	0.20(1)	0.55(4)	0.275(2)	0.055(1)
21	0.40(5)	0.80(1)	0.20(5)	0.35(4)	0.50(3)	0.275(2)	0.055(1)
22	0.40(5)	0.85(2)	0.00(1)	0.40(5)	0.55(4)	0.280(3)	0.055(1)
23	0.40(5)	0.90(3)	0.05(2)	0.20(1)	0.60(5)	0.285(4)	0.055(1)
24	0.40(5)	0.95(4)	0.10(3)	0.25(2)	0.40(1)	0.290(5)	0.055(1)
25	0.40(5)	1.00(5)	0.15(4)	0.30(3)	0.45(2)	0.270(1)	0.055(1)
26	0.20(1)	0.80(1)	0.00(1)	0.20(1)	0.40(1)	0.270(1)	0.060(2)
27	0.20(1)	0.85(2)	0.05(2)	0.25(2)	0.45(2)	0.275(2)	0.060(2)
28	0.20(1)	0.90(3)	0.10(3)	0.30(3)	0.50(3)	0.280(3)	0.060(2)
29	0.20(1)	0.95(4)	0.15(4)	0.35(4)	0.55(4)	0.285(4)	0.060(2)
30	0.20(1)	1.00(5)	0.20(5)	0.40(5)	0.60(5)	0.290(5)	0.060(2)
31	0.25(2)	0.80(1)	0.05(2)	0.30(3)	0.55(4)	0.290(5)	0.060(2)
32	0.25(2)	0.85(2)	0.10(3)	0.35(4)	0.60(5)	0.270(1)	0.060(2)
33	0.25(2)	0.90(3)	0.15(4)	0.40(5)	0.40(1)	0.275(2)	0.060(2)
34	0.25(2)	0.95(4)	0.20(5)	0.20(1)	0.45(2)	0.280(3)	0.060(2)
35	0.25(2)	1.00(5)	0.00(1)	0.25(2)	0.50(3)	0.285(4)	0.060(2)
36	0.30(3)	0.80(1)	0.10(3)	0.40(5)	0.45(2)	0.285(4)	0.060(2)
37	0.30(3)	0.85(2)	0.15(4)	0.20(1)	0.50(3)	0.290(5)	0.060(2)
38	0.30(3)	0.90(3)	0.20(5)	0.25(2)	0.55(4)	0.270(1)	0.060(2)
39	0.30(3)	0.95(4)	0.00(1)	0.30(3)	0.60(5)	0.275(2)	0.060(2)
40	0.30(3)	1.00(5)	0.05(2)	0.35(4)	0.40(1)	0.280(3)	0.060(2)
41	0.35(4)	0.80(1)	0.15(4)	0.25(2)	0.60(5)	0.280(3)	0.060(2)
42	0.35(4)	0.85(2)	0.20(5)	0.30(3)	0.40(1)	0.285(4)	0.060(2)
43	0.35(4)	0.90(3)	0.00(1)	0.35(4)	0.45(2)	0.290(5)	0.060(2)
44	0.35(4)	0.95(4)	0.05(2)	0.40(5)	0.50(3)	0.270(1)	0.060(2)
45	0.35(4)	1.00(5)	0.10(3)	0.20(1)	0.55(4)	0.275(2)	0.060(2)
46	0.40(5)	0.80(1)	0.20(5)	0.35(4)	0.50(3)	0.275(2)	0.060(2)
47	0.40(5)	0.85(2)	0.00(1)	0.40(5)	0.55(4)	0.280(3)	0.060(2)
48	0.40(5)	0.90(3)	0.05(2)	0.20(1)	0.60(5)	0.285(4)	0.060(2)
49	0.40(5)	0.95(4)	0.10(3)	0.25(2)	0.40(1)	0.290(5)	0.060(2)
50	0.40(5)	1.00(5)	0.15(4)	0.30(3)	0.45(2)	0.270(1)	0.060(2)

Table 6.9: $L_{50}(2^1 \times 5^6)$ orthogonal array in the narrower ranges

The final results of these test problems are compiled in Table 6.10. It shows that compared with results by the parameters defined in the larger ranges, the solutions derived by the refined parameter settings have been slightly improved. On average, the solution cost of the proposed approach is 0.33% lower, and the associated cover size are only 0.69% larger. In T7, the new solution cost is worse by 3.25%. However, it should be noted that the cover size of T7 in our solution is smaller by 1.01%.

Data	SE's final solution				
	Cover Size	RPD (%)	Cost (hours)	RPD (%)	Elapsed time (seconds)
T1	64	3.23	504.13	-1.01	384
T2	118	1.72	993.35	-1.01	161
T3	66	3.13	565.38	0.56	730
B1	74	-1.33	814.53	-4.30	668
T4	243	0.41	2246.32	-0.05	1398
T5	275	-0.36	2073.36	-0.47	468
T6	348	-0.29	2670.57	0.36	420
T7	391	-1.01	3239.23	3.25	962
Ave.		**0.69%**		**-0.33%**	

Table 6.10: Comparative results by the refined parameter settings

With respect to the variation of solutions for individual cases, Figure 6.2 depicts the improvement of the cost value versus the number of iterations for the instance of T1. Although the actual values may differ among various cases, the characteristic shapes of the curves are similar.

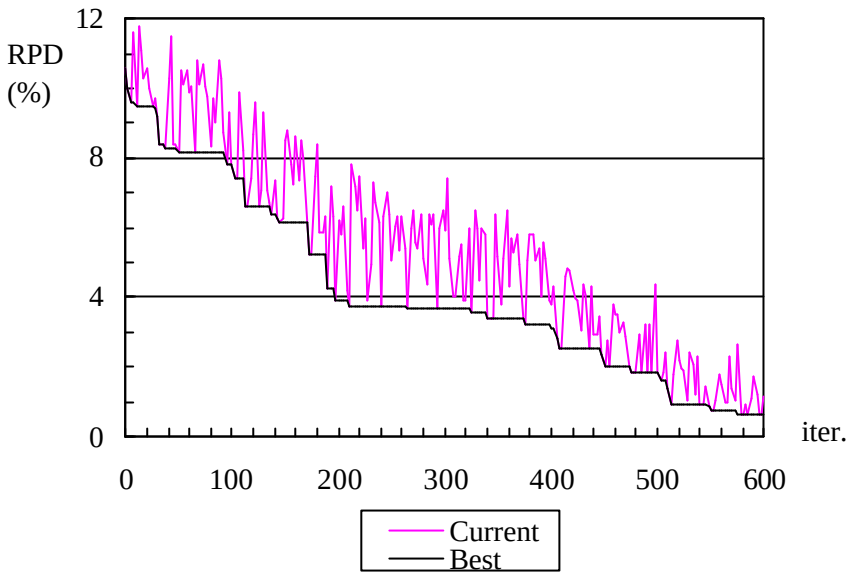


Figure 6.2: RPD of solution cost (in current and best schedule respectively) versus iteration number

6.6 Conclusions

In this chapter, a novel fuzzy SE approach for the non-unicost set covering problem has been developed. A function is first designed to evaluate the structure of each column under fuzzified factors. The formation of these factors is similar to that in driver scheduling. The difference is that factors involved in domain knowledge of driver scheduling are removed, replacing with new factors that only concern the set covering problem. This function is embedded into the *Construction* step and the *Evaluation* step of the proposed Simulated Evolution algorithm, which mimics generations of evolution on a single solution. In each generation an unfit portion of the working solution is removed. The broken solution is then repaired by a greedy algorithm specialized for the set covering problem.

In the proposed approach, there are seven investigated control factors in total. Using the "change-one-factor-at-a-time" method of experimentation, a prohibitively large number of 31,250 experiments needs to be carried out. In this research, the $L_{50}(2^1 \times 5^6)$ orthogonal array

from Taguchi's experimental design theory is applied to reduce the number of experiments to 50.

Instances from driver scheduling have been used to test the evolutionary approach. These test problems can be regarded as general set covering problems, since they are handled without the benefit of domain knowledge of driver scheduling. It has demonstrated that for very large-scale problems, in general the proposed approach can produce superior solutions much faster than some other approaches. Particularly, this approach is suitable for situations where quick and high-quality solutions are desirable.

Future work may test the performance of this approach on randomly generated problems, which can be obtained from the OR-Library (Beasley 1990).

Chapter Seven

Conclusions

7.1 Summary

The bus and rail driver scheduling problem and its commercial importance have been presented. A review of meta-heuristics used for the set covering problem has also been provided. The success and limitations of the current methods for driver scheduling have been described. Early heuristic approaches are heavily dependent on problem-specific knowledge and hard to adapt between different organisations, and they generally have difficulty in producing satisfactory results. On the other hand, mixtures of heuristics and mathematical programming have been very successful, although they still have their weaknesses. This fact stimulated investigations on other approaches.

This research has investigated two evolutionary algorithms to model and solve the driver scheduling problem. These algorithms start from a predefined large set of possible legal shifts, from which a small set of shifts is selected to produce a schedule.

The new approach has achieved success in solving large-scale driver scheduling problems from different companies, with comparable results to the TRACS II system, a commercial system based on an ILP solver with more than 100 person-years devoted in its development. In particular, in several very large cases, our results are even better, and indicate directions for further research.

Based on fuzzy set theory, the first phase of this research presents a novel evolutionary approach for the driver scheduling problem, which involves solving a set covering model. At the heart of this approach is a function for evaluating, under fuzzified criteria, potential driver shifts. A GA is first employed to calibrate the weight distribution among fuzzy membership functions. A SE algorithm then mimics generations of evolution on the single schedule produced by the GA. In each generation an unfit portion of the working schedule is removed. The broken schedule is then reconstructed by means of a greedy algorithm, using the weight distribution derived by the GA. The basic SE algorithm is a greedy search strategy that achieves improvement through iterative perturbation and reconstruction.

The next phase of this research reports on new improvement in the fuzzy SE approach, and its generalisation of the approach to the class of set covering. The set covering problem is basically to cover the rows of a zero-one matrix with a subset of columns at minimal cost, which has a very wide area of important applications. Taguchi's orthogonal experimental design is applied. This has the effect of comprehensively evaluating the combinations of factors, although only a small fraction of the possible combinations is explicitly experimented upon.

7.2 Achievements in this research

Driver scheduling problems are a worldwide problem, and any advances in the techniques available for solving such problems are highly significant. The major contributions of the research are listed as follows:

1) This research presents a refined greedy algorithm based on fuzzy subsets theory to effectively solve the problem about ranking the potential shifts in each iteration. The new algorithm is novel because it is the first time that fuzzy set theory has been applied to the driver scheduling problem.

2) A GA is presented to achieve the goal of solving the large problems without decomposition. The benchmark experimental results have shown that its results are closed to the best known solutions (normally no worse than 2%) in much faster speed.

3) To further improve the GA's results and still maintain a balance between time complexity and accuracy, this research applies a general optimization technique of SE algorithm. It is also the first time that the SE algorithm has been introduced for the driver scheduling problem. The computational results are very close to the best known solution (normally no worse than 1%). For some large cases which cause difficulties for TRACS II, its results are even better, and are more than 50 times faster in speed.

4) Although the main theme of this research is presented in terms of driver scheduling, it is suggested that, the proposed SE approach could also be applied to many other scheduling problems whose solutions can be decomposed into elements. Even the process of identifying fuzzified criteria could be generalized and applied elsewhere.

5) The fuzzy SE approach for the driver scheduling problem has been generalized successfully to the set covering problem, without using any special domain knowledge. It means that this research is valuable to many applications that can be formulated using the set covering model. Furthermore, the proposed Taguchi's orthogonal experimental design for the related parameter settings is rather unconventional.

7.3 Future work

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

1) With respect to the ability of treating side constraints such as restricting the number of shifts of some depots, solution quality could be improved further if they are taken into account. This might be implemented by adding a criterion of side constraints (u_6) as the additional measure for shift structure.

2) With respect to the SE algorithm, further research could be continued to improve its searching efficiency. For example, the selection threshold in the Selection step and the mutation rate in the Mutation step are currently set to be constants in all generations, which might be improved by more sophisticated, such as adaptive, operators.

   Furthermore, the two objectives of minimising cost and number of shifts in the SE algorithm have been combined in the form of a weighted-sum function, which will cause problems if the Pareto surface is non-convex. Most popular evolutionary algorithms for multiobjective optimisation maintain a population of solutions, from which individuals are selected for reproduction. However, since only one solution is

produced in each iteration of the SE, these population-based strategies are difficult to apply. Knowles and Corne (2000) presented a Pareto Archived Evolution Strategy (PAES), which employs a local search to the true multiobjective problem to find diverse solutions in the Pareto optimal set. Further research could be implemented by using the idea of PAES to achieve a better balance between the two objectives of driver scheduling.

3) With respect to the further improvement about solution quality for the set covering problem, according to the experimental results there is no significant improvement in general even if the refined parameter settings have been used. The reason might be that it is only the relative levels of two or three parameters that are important, or that some weights are much more sensitive than others. Therefore, to improve the solution quality further, the statistical method, analysis of variance, needs to be carried out to analyse the results of the orthogonal experimental design, and to determine how much variation each factor has contributed.

Bibliography

Ackley, D.H. (1987) “A Connectionist Machine for Genetic Hillclimbing,” Kluwer Academic Press.

Beasley, J.E. (1987) “An algorithm for set covering problems,” European Journal of Operational Research, vol. 31, pp. 85-93.

Beasley, J.E. (1990) “A Lagrangian heuristic for set covering problems,” Naval Research Logistics, vol. 37, pp. 151-164.

Beasley, J.E. (1990) “OR-Library: Distribution test problems by electronic mail,” Journal of the Operational Research Society, vol. 41, pp. 1069-1072.

Beasley, J.E. and Chu, P.C. (1996) “A genetic algorithm for the set covering problem,” European Journal of Operational Research, vol. 94, pp. 392-404.

Beasley, J.E. and Johnstern, K. (992) “Enhancing an algorithm for set covering problems,” European Journal of Operational Research, vol. 58, pp. 293-300.

Bennington, G. and Rebibio, K. (1975) “Overview of RUCUS vehicle scheduling program (BLOCKS),” in Bergmann D. and Bodin, L. (Eds.), Preprint: Workshop on Automated Techniques for Scheduling of Vehicle Operators for Urban Public Transportation Services.

Bessiere C. and Cordier, M.O. (1993) “Arc-consistency and arc-consistency again,” Proceedings of AAAI-93, pp. 108-113.

Bhuyan, J. (1995) “A combination of genetic algorithm and simulated evolution techniques for clustering,” Proceedings of the 23rd ACM Annual Conference on Computer science, pp. 127-134.

Blais, J.Y. and Rousseau, J.M. (1988) “Overview of HASTUS current and future versions,” in Daduna, J.R. and Wren, A. (Eds.), Computer-aided Transport Scheduling, Springer-Verlag, pp. 175-187.

Bloch, I. (1996) “Information combination operators for data fusion: a comparative review with classification,” IEEE Transaction on System, Man and Cybernetics, vol. 81(2), pp. 52-67.

Bodin, L.D., Ball, M.O. and Greenberg, J. (1985) “Enhancements to the RUCUS II crew scheduling system,” in Rousseau, J.M. (Ed.), Computer Aided Scheduling of Public Transport 2, North-Holland, pp. 279-294.

Boese, K., Kahng, A. and Muddu, S. (1994) “A new adaptive multi-start technique for combinatorial global optimization,” Operations Research Letters, vol. 16, pp. 101-113.

Booker, L. (1987) “Improved search in genetic algorithms,” Proceedings of the 1st International Conference on Genetic Algorithms, pp. 101-111.

Breuer, M. (1970) “Simplification of the set covering problem with application to Boolean expression,” Journal for the Association of Computing Machinery, vol. 17, pp.166-181.

Burke, E.K., Newell, J.P. and Weare, R.F. (1998) “Initialisation strategies and diversity in evolutionary timetabling,” Evolutionary Computation (Special Issue on Scheduling), vol. 6, pp. 81-103.

Caprara, A., Fischetti, M. and Toth, P. (1999) “A heuristic method for the set covering problem”, Operations Research, vol. 47, pp. 730-743.

Carlier, J. and Pinson, E. (1989) “An algorithm for solving the job-shop problem,” Management Science, vol. 35, pp. 164-176.

Cavique, L., Rego, C. and Themido, I. (1999) “Subgraph ejection chains and tabu search for the crew scheduling problem,” European Journal of Operational Research, vol. 50, pp. 608-616.

Chvátal, V. (1979) “A greedy heuristic for the set-covering problem,” Mathematics of Operations Research, vol. 4, 233-235.

Chamberlain, M. and Wren, A. (1992) “Developments and recent experience with the BUSMAN and BUSMAN II system,” in Desrochers, M. and Rousseau, J.M. (Eds.), Proceedings of the Fifth International Workshop on Computer-Aided Scheduling of Public Transport, Spring-Verlag, pp. 1-15.

Corne, D. and Ross, P. (1996) "Peckish initialization strategies for evolutionary timetabling," in Burke, E.K. and Ross, P. (Eds.), pp. 69-81, LNCS 1153, Springer-Verlag.

Curtis, S.D. (2000) “Constraint Satisfaction Approaches to Bus Driver Scheduling,” PhD Thesis, School of Computing, University of Leeds.

Curtis, S.D., Smith, B.M. and Wren, A. (1999) "Forming bus driver schedules using constraint programming," Proceedings of First International Conference on the Practical Applications of Constraint Technologies and Logic Programming, pp. 239-254.

Daduna, J.R., Branco, I. and Paixao, J.M.P. (Eds.) (1995) "Computer-Aided Transit Scheduling," Proceedings, Lisbon, Portugal, Springer-Verlag.

Daduna J.R. and Mojsilovic, M. (1988) "Computer-aided vehicle and duty scheduling using the HOT programme system," in Wren, A. (Ed.), Computer-Aided Transit Scheduling, Springer-Verlag, pp. 133-146.

Davis, L. (1987) "Genetic Algorithms and Simulated Annealing," London, Pitman.

Dénes, J. and Keedwell, A.D. (1974) "Latin Square and Their Applications," English Universities Press Limited, Budapest.

Desrochers, M. and Rousseau J.M. (Eds.) (1992) "Computer-Aided Transit Scheduling," Proceedings, Montreal, Canada, Springer-Verlag.

Dorigo, M., Maniezzo, V. and Colorni, A. (1995) "Ant system: optimisation by a colony of cooperating agents," IEEE Transactions on Systems, Man, and Cybernetics, vol. 26, pp. 29-41.

Dubois, D. and Prade, H. (1980) "Fuzzy sets and systems: theory and applications," Academic Press, New York, London.

Falkner, J.C. and Ryan, D.M (1988) “Aspects of bus crew scheduling using a set partitioning model,” in Daduna, J.R. and Wren, A. (Eds.), Computer-aided Transport Scheduling, Springer-Verlag, pp. 91-103.

Falkner, J.C. and Ryan, D.M. (1992) “EXPRESS: set partitioning for bus crew scheduling in Christchurch,” in Desrochers, M. and Rousseau, J.M. (Eds.), Computer-aided Transport Scheduling, Springer-Verlag, pp. 359-378.

Feo, T. and Resende, M. (1995) “Greedy randomized adaptive search procedures,” Journal of Global Optimization, vol. 6, pp. 109-133.

Fisher, M.L. and Kedia, P. (1990) “Optimal solution of the set covering/partitioning problems using dual heuristics,” Management Science, vol. 36, 674-688.

Fores, S., Proll, L. and Wren, A. (1999) “An improved ILP system for driver Scheduling,” in Wilson, N.H.M. (Ed.), Proceeding of the Seventh International Workshop on Computer-Aided Scheduling of Public Transport, Spring-Verlag, pp. 43-61.

Fores, S., Proll, L. and Wren, A. (2001) “Experiences with a flexible driver scheduler,” in Voß, S. and Daduna, J.R. (Eds.), Computer-Aided Scheduling of Public Transport, Springer-Verlag, pp. 137-152.

Forsyth, P. and Wren, A. (1997) “An ant system for driver scheduling,” Technical Report 97.25, School of Computing, University of Leeds.

Garey, M.R. and Johnson, D.S. (1979) “Computers and Intractability: a Guide to the Theory of NP-Completeness,” Freeman, San Francisco.

Glover, F. (1989) “Tabu search –part 1,” ORSA Journal on Computing, vol. 1, pp. 190-206.

Glover, F. (1990) “Tabu search –part 2,” ORSA Journal on Computing, vol. 2, pp. 4-32.

Goldberg, D.E. (1989) “Genetic Algorithms in Search, Optimization and Machine Leaning,” Addison-Wesley.

Guerinik, N. and Caneghem, M.V. (1995) “Solving crew scheduling problems by constraint programming,” in Montanari, U. and Rossi, F. (Eds.), Principles and Practice of Constraint Programming, pp. 481-498.

Haddadi, S. (1997) “Simple Lagrangian heuristic for the set covering problem.” European Journal of Operational Research, vol. 97, pp. 200-204.

Hoffstadt, J. (1981) “Computerized vehicle and driver scheduling for the Hamburger Hochbahn Aktiengesellschaft,” in Wren, A. (Ed.), Computer Scheduling of Public Transport, North-Holland, pp. 35-52.

Holland, J.H. (1975) “Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence,” University of Michigan Press.

Ishibuchi, H. and Murata, T. (1998) “Multi-objective genetic local search algorithm and its application to flowshop scheduling,” IEEE Transaction on System, Man, and Cybernetics, vol. 28, pp. 392-403.

Jaszkiewicz, A., Hapke, M. and Kominek, P. (2001) “Performance of multiple objective evolutionary algorithms on distribution system design problem – computational experiments,”

in Zitzler, E., Deb, K., Thiele, L., Coello, C.A.C., and Corne, D. (Eds.), Evolutionary Multi-Criteria Optimization, LNCS 1993, Springer, Berlin, pp. 241-255.

Johnson, D.S. (1974) "Approximation algorithms for combinatorial problems," Journal of Computer and System Science, vol. 9, pp 256-278.

Karmarkar, N. (1984) "A new polynomial-time algorithm for Linear Programming," Combinatorica, vol. 4, pp. 373-395.

Karp, R.M. (1972) "Reducibility among combinatorial problems," in: Miller, R.E. and Thatcher, J.W. (Eds.), Complexity of Computer Computations, Plenum Press, New York, pp. 85-103.

Kaufmann, A. (1975) "Introduction to the theory of fuzzy subsets: fundamental theoretical elements," Academic Press, New York, San Francisco, London.

Kling, R.M. and Banerjee, P. (1987) "ESP: a new standard cell placement package using Simulated Evolution," Proceedings of the 24th ACM/IEEE Design Automation Conference, pp. 60-66.

Klir, G.J. and Yuan, B. (1995) "Fuzzy sets and fuzzy logic: theory and applications," Prentice Hall, New Jersey.

Knowles, J.D. and Corne, D.W. (2000) "Approximating the nondominated front using the Pareto archived evolution strategy," Evolutionary Computation, vol. 8, pp. 149-172.

Kwan, A.S.K. (1999) "Train Driver Scheduling," PhD Thesis, School of Computing, University of Leeds.

Kwan, A.S.K., Kwan, R.S.K., and Wren, A. (1999) "Driver scheduling using genetic algorithms with embedded combinatorial traits," in Wilson, N.H.M. (Ed.), Computer-Aided Transit Scheduling, Springer-Verlag, pp. 81-102.

Kwan, R.S.K., Wren, A. and Kwan, A.S.K. (2000) "Hybrid genetic algorithms for scheduling bus and train drivers," Proceedings of 2000 IEEE Congress on Evolutionary Computation, IEEE Service Center, pp. 285-292.

Kwan, R.S.K., Kwan, A.S.K. and Li, J. (2000a) "Genetic algorithms for integer solutions of bus and rail driver scheduling," (presented at) The 17th International Symposium on mathematical Programming, August 7-11, Atlanta, USA.

Kwan, R.S.K, Kwan, A.S.K. and Wren, A. (2001) "Evolutionary driver scheduling with relief chains," Evolutionary Computation, vol. 9, pp. 445-460.

Layfield, C.J., Smith, B.M. and Wren, A. (1999) "Bus relief opportunity selection using constraint programming," Practical Application of Constraint Technologies and Logic Programming Conference, The Practical Application Company Ltd, pp. 537-552.

Lessard, R., Rousseau, J.M. and Dupuis, D. (1981), "HAUSUS I: a mathematical programming approach to the bus driver scheduling problem," in Wren, A. (Ed.), Computer Scheduling of Public Transport, North-Holland, pp.255-267.

Li, J. and Kwan, R.S.K. (2000) "Genetic algorithms for the bus and rail driver scheduling problem," in the 11th Conference of Young Operational Research, pp. 39, March 28-30, Cambridge, UK.

Li, J. and Kwan, R.S.K. (2001) “A fuzzy simulated evolution algorithm for the driver scheduling problem,” Proceedings of the 2001 IEEE Congress on Evolutionary Computation, IEEE Service Center, pp.1115-1122.

Li, J. and Kwan, R.S.K. (2001a) “A fuzzy theory based evolutionary approach for driver scheduling,” in Spector, L. et al. (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufman, pp. 1152-1158.

Li, J. and Kwan, R.S.K. (2001b) “Evolutionary driver scheduling with fuzzy evaluation,” Proceedings of the Graduate Student Workshop at Genetic and Evolutionary Computation Conference, pp.421-424, July 8, San Francisco, USA.

Li, J. and Kwan, R.S.K. (2002) “A fuzzy evolutionary approach with Taguchi parameter setting for the set covering problem,” Proceedings of the 2002 IEEE Congress on Evolutionary Computation, IEEE Service Center, pp.1203-1208.

Li, J. and Kwan, R.S.K. (2002) “A fuzzy genetic algorithm for driver scheduling,” (to appear in) European Journal of Operational Research (Special Issue on Fuzzy Scheduling and Planning).

Lin, Y.L., Hsu, Y.C. and Tsai, F.S. (1989) “SILK: a Simulated Evolution router,” IEEE Transaction on Computer-Aided Design, vol. 8, pp. 1108-1114.

Lovász, L. (1975) “On the ratio of optimal integral and fractional covers,” Discrete Mathematics, vol. 13, pp. 383-390.

Ly, T.A. and Mowchenko, J.T. (1993) “Applying Simulated Evolution to high level synthesis,” IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, vol. 12, pp. 389-409.

Michaelwicz, Z. (1994) “Genetic Algorithms + Date Structures = Evolution Programs,” Springer-Verlag.

Mitchell, M. (1996) “An Introduction to Genetic Algorithms,” The MIT Press, Massachusetts.

Moscato, P. and Norman, M.G. (1992) “A ‘memetic’ approach for the traveling salesman problem: implementation of a computational ecology for combinatorial optimization on message-passing systems,” In Valero, M., Onate, E., Jane, M., Larriba, J.L., and Suarez, B. (Eds.), Parallel Computing and Transputer Applications, Ed. IOS Press, Amsterdam, pp. 187-194.

Ohlsson, M., Peterson, C. and Söberderg, B. (2001) “An efficient mean field approach to the set covering problem,” European Journal of Operational Research, vol. 133, pp. 583-599.

Parker, M.E. and Smith, B.M. (1981) “Two approaches to computer crew scheduling,” in Wren, A. (Ed.), Proceedings of the Second International Workshop on Computer-Aided Scheduling of Public Transport, North-Holland, pp. 193-222.

Parker, M.E., Wren, A. and Kwan, R.S.K. (1995) “Modelling the scheduling of train drivers,” in Daduan, I.R., Branco, I. and Paixao, M.P. (Eds.), Proceedings of the Sixth International Workshop on Computer-Aided Scheduling of Public Transport, Spring-Verlag, pp. 359-370.

Petrou, M. and Sasikala, K.R. (1999) “Generalised fuzzy aggregation operators,” in Perner, P. and Petrou, M. (Eds.), LNAI 1715, Springer-Verlag, pp. 195-207.

Proll, L. (1997) “Stronger formulations of mixed integer Linear Programs: an example,” International Journal of Mathematical Education in Science and Technology, vol. 28, pp. 707-712.

Radcliffe, N.J. and Surry, P.D. (1994) “Formal memetic algorithms,” in Fogarty, T., (Ed.), Evolutionary Computing, Springer-Verlag, Berlin, pp. 1-16.

Revelle, C.D., Marks, D. and Liebman, J.C. (1970) “An analysis of private and public sector facilities location models,” Management of Science, vol. 16, pp. 692-707.

Rodosek, R., Wallace, M.G., and Hajian, T. (1996) “A new approach to integrate mixed integer programming,” CP96 Workshop on Constraint Programming Applications: An Inventory and Taxonomy.

Rubin, J. (1973) “A technique for the solution of massive set covering problems with application of airline crew scheduling,” Transportation Science, vol. 7, pp. 34-48.

Sait, S.M., Youssef, H. and Ali, H. (1999) “Fuzzy Simulated Evolution algorithm for multi-objective optimization of VLSI placement,” Proceedings of 1999 IEEE Congress on Evolutionary Computation, IEEE Press, pp. 91-97.

Salveson, M. (1955) “The assembly line balancing problem”, Journal of Industrial Engineering, vol. 6, pp. 18-25.

Shen, Y. and Kwan, R.S.K. (2001) “Tabu search for driver scheduling,” in Voß, S. and Daduna, J.R. (Eds.), Computer-Aided Scheduling of Public Transport, Springer-Verlag, pp. 121-135.

Shen, Y. (2001) “Tabu Search for Bus and Train Driver Scheduling with Time Windows,” PhD Thesis, School of Computing, University of Leeds.

Sen, S. (1993) “Minimal cost set covering using probabilistic methods,” Proceedings of the 1993 ACM Symposium on Applied Computing: States of the Art and Practice, pp. 157-164.

Slavík, P. (1996) “A Tight analysis of the greedy algorithm for set cover,” ACM Symposium on Theory of Computing, pp. 435-441.

Smith, B.M. and Wren, A. (1988) “A bus crew scheduling system using set covering formulation,” Transpn. Res., vol. 22, pp. 97-108.

Solar, M, Parada, V. and Urrutia, R. (2002) “A parallel genetic algorithm to solve the set-covering problem,” Computer and Operations Research, vol. 29, pp. 1221-1235.

Spears, W.M. and Dejong, K.A. (1991) “An analysis of multipoint crossover,” Proceedings of 1990 Workshop of the Foundations of Genetic Algorithms, pp. 301-315.

Srinivas, M. and Patnaik, L.M. (1994) “Adaptive probabilities of crossover and mutation in Genetic Algorithms,” IEEE Transaction on System, Man and Cybernetics, vol. 24, pp. 656-667.

Srinivasan, A. (1995) “Improved approximations of packing and covering problems,” ACM Symposium on Theory of Computing, pp. 268-276.

Steuer, R.E. (1986) “Multiple Criteria Optimization – Theory, Computation and Application,” Wiley, New York.

Taha, H. (1997) “Operations Research (An Introduction)”, Prentice-Hall Inc., USA

Taguchi, G. (1986) “Introduction to Quality Engineering,” Asian Productivity Organization, Tokyo.

Taguchi, G. (1987) “System of Experimental Design,” vols. 1 and 2, UNIPUB/Kraus International Publications, New York.

The triplex design group of Chinese Association of Statistics (1987) “Orthogonal Method and Triplex Design”, Science Press, Beijing.

Toregas, C., Swain, R., Revelle, C. and Bergman, L. (1971) “The location of emergency service facilities,” Operations Research, vol. 19, pp. 1363-73.

Tsang, E.P.K. (1993) “Foundations of Constraint Satisfaction,” Academic Press.

Ulungu, E.L. and Teghem, J. (1994) “Multi-objective combinatorial optimization problems: a survey,” Journal of Multi-Criteria Decision Analysis, vol. 3, pp. 83-101.

Voß, S. and Daduna, J.R. (Eds.) (2001), Computer-Aided Scheduling of Public Transport, Springer-Verlag, pp. 121-135.

Willers, W.P., Proll, L.G. and Wren, A. (1995) “A dual strategy for solving the linear programming relaxation of a driver scheduling system,” Annals of Operations Research, vol. 58, pp.519-531.

Wilson, N.H.M. (Ed.) (1999) “Computer-Aided Transit Scheduling,” Proceedings, Cambridge, MA, USA, Springer-Verlag.

Wren, A., Smith, B.M and Miller, A.J. (1985) “Complementary approaches to crew scheduling,” in Rousseau, J.M. (Ed.), Proceedings of the Third International Workshop on Computer-Aided Scheduling of Public Transport, North-Holland, pp. 263-278.

Wren, A. and Rousseau, J.M. (1995) “Bus driver scheduling – an overview,” in Daduna, J.R., Branco, I. and Paixao J.M.P. (Eds.), Computer-Aided Scheduling of Public Transport, Springer-Verlag, pp. 173-187.

Wren, A. and Smith, B.M. (1988) “Experiences with a crew scheduling system based on set covering,” in Daduna, J.R. and Wren, A. (Eds.), Proceedings of the fourth International Workshop on Computer-Aided Scheduling of Public Transport, Spring-Verlag, pp. 104-118.

Zadeh, L.A. (1965) “Fuzzy sets,” Information and Control, vol. 8, pp. 338-353.

Zalzala, A.M.S. and Fleming, P.J. (Eds.) (1997) “Genetic Algorithms in Engineering Systems”, Institution of Electrical Engineers, London.