Genetic algorithms for the Bus Driver Scheduling Problem: a case study

Article *in* Journal of the Operational Research Society · March 2002

DOI: 10.1057/palgrave.jors.2601312 · Source: OAI

Teresa Galvão Dias
University of Porto
91 PUBLICATIONS **914** CITATIONS
SEE PROFILE

Jorge Pinho de Sousa
University of Porto
139 PUBLICATIONS **1,983** CITATIONS
SEE PROFILE

João Falcão e Cunha
University of Porto
99 PUBLICATIONS **2,100** CITATIONS
SEE PROFILE

Journal of the Operational Research Society (2002) 53, 324–335 ©2002 Operational Research Society Ltd. All rights reserved. 0160-5682/02 $15.00
www.palgrave-journals.com/jors

Genetic algorithms for the bus driver scheduling problem: a case study

TG Dias,* JP de Sousa and JF Cunha
Universidade do Porto/INEGI, Porto, Portugal

This paper describes an application of genetic algorithms to the bus driver scheduling problem. The application of genetic algorithms extends the traditional approach of Set Covering/Set Partitioning formulations, allowing the simultaneous consideration of several complex criteria. The genetic algorithm is integrated in a DSS but can be used as a very interactive tool or a stand-alone application. It incorporates the user's knowledge in a quite natural way and produces solutions that are almost directly implemented by the transport companies in their operational planning processes. Computational results with airline and bus crew scheduling problems from real world companies are presented and discussed.

Journal of the Operational Research Society **53,** 324–335. DOI: 10.1057/palgrave/jors/2601312

Keywords: genetic algorithms; bus driver scheduling problem; decision support systems

Introduction

The Bus Driver Scheduling Problem (BDSP) is an extremely complex part of the operational planning process of transport companies. Its combinatorial nature and the large size of the problems has led to the development of a large number of models and techniques that are applied in practice, according to the complexity and the particular characteristics of each company.

The process of bus driver scheduling consists of constructing a set of legal shifts, which together cover all the trips planned for a group of vehicles. Although there are interactions between vehicle scheduling and crew scheduling, these problems are usually solved separately. Crew scheduling is subject to constraints that are specific to each company. Generally these constraints arise from government legislation, union agreements and some operational rules that are settled between drivers and the transport company.[1]

The planning process starts by the definition of vehicle schedules, aiming at minimising the number of vehicles required. Then the daily work of each vehicle is divided into units, called *pieces-of-work*, that start and finish at *relief points*, ie points where drivers can be replaced. Several successive pieces-of-work may form a *leg*, ie a part of a driver's duty in one day. Along with the driving time, a duty usually also includes meal breaks, overtime periods, as well as the times needed to operate the vehicles inside the depot. A set of pieces-of-work that satisfies all the constraints is a *feasible duty.* A solution for the bus driver scheduling problem is a set of feasible duties that hopefully cover all the vehicle trips scheduled for a route or a small set of routes. Several objectives and goals that will allow judgements of cost and quality also guide the process of constructing such a solution.

For this type of problem Genetic Algorithms are interesting, because they do not impose any particular restrictions on the objective function structure, which may therefore encompass different characteristics that are usually very difficult to handle by traditional algorithms. Linearity or differentiability conditions are not imposed on the objective function which may express several complex criteria, such as setting targets for average duty duration and range. Infeasible solutions may also be allowed, but the objective function can include penalties trying to avoid solutions with undesirable features.

Models and algorithms for the bus driver scheduling problem

Like many other combinatorial optimisation problems, the bus driver scheduling problem is *NP-hard* and therefore optimising approaches are not in general very useful in practice, considering the very large size of real problems. The trade-off between quality of solutions and computational time in practice leads to the use of heuristic procedures that aim at efficiently producing good solutions as close as possible to those obtained manually in real time. These include 'run-cutting' heuristics, ie sequential constructive algorithms that 'formalise' manual procedures.

**Correspondence: TG Dias, Faculdade de Engenharia da Universidade do Porto/INEGI, R Dr Roberto Frias, 4200-465 Porto, Portugal.*
E-mail: tgalvao@fe.up.pt

The heavy computational effort of mathematical programming techniques and the lack of generality of 'run-cutting' heuristics led to the development of systems that combine optimisation models and heuristic procedures.[1] These systems usually incorporate graphic tools that help the operator to manually adjust solutions and dynamically interact with the system. They are currently being successfully used in many transport companies.

Generation of feasible duties

Theoretically, the bus driver scheduling problem can be solved by first generating a large number of feasible duties, and by applying then a mathematical programming technique that selects a set of duties that cover all the trips with minimal cost.

In this approach, the first practical issue is the number of feasible duties that are generated. In fact, even for small problems, the total number may be too large and it is therefore necessary to reduce it. One possible technique is to eliminate potential duties with particular long breaks, with breaks at unsuitable times of the day, or with very little work content before or after a meal break.[2] Additionally, some systems use heuristic procedures that select potentially interesting duties and eliminate less efficient ones.[3,4]

One can, for example, examine all the possible pieces-of-work to determine whether two consecutive ones can be combined together. Since each duty must start, finish or have a break at a relief point, the reduction of the number of these points will lead to a lower number of potential feasible duties. Such reduction techniques can be found in commercially available systems, such as HASTUS,[5,6] BUSMAN[7] or in the GIST DSS[8] used in this work as the basis for experiments.

Nevertheless, it is necessary to ensure the generation of a sufficient large number of potentially interesting feasible duties. The quality of the final solution will, of course, depend on the number and quality of the generated duties. This generation is performed in a controlled way by a specific heuristic procedure.

Formulation of the bus driver scheduling problem

The bus driver scheduling problem can be formulated as follows:

$$\text{minimise} \quad \sum_{j\in P} c_j x_j$$

$$\text{subject to} \quad \sum_{j\in P} a_{ij} x_j \geqslant 1 \quad \forall i \in I \qquad (1)$$

$$\sum_{j\in P} b_{ij} x_j \geqslant u_j \quad \forall i \in I \qquad (2)$$

$$x_j \in \{0, 1\} \quad \forall j \in P \qquad (3)$$

where:

$I = \{i$: piece-of-work$\}$;
$P = \{j$: candidate feasible duty$\}$;
$c_j =$ cost of feasible duty j;
$x_j = 1$, if duty j is in the solution,
$= 0$, otherwise;
$a_{ij} = 1$, if the piece-of-work i belongs to duty j,
$= 0$, otherwise;
b_{ij}, $u_j =$ real values representing operational properties of the solution.

Each row of the problem formulation matrix A (of the a_{ij}) corresponds to a piece-of-work and each column (or variable) corresponds to a feasible duty defined by the pieces-of-work that are identified by the non-zero elements. We say that row i is covered by column j if $a_{ij} = 1$. Constraints (1) impose that each piece-of-work is part of at least one duty of the solution.

Constraints (2) force the satisfaction of some operational properties of the solution (eg percentage of types of duties, number of duties or maximum working time). These constraints make the problem very hard to solve because we cannot take advantage of the *set covering* structure of the model, associated to constraints (1). The 'inclusion' of constraints (2) in the objective function, with appropriate penalty coefficients, is a common way of tackling this issue.[9,10]

It should be noted that the real BDSP cannot be modelled simply as a set covering or partitioning problem, as additional difficult constraints need to be added.

Set covering and set partitioning models

In most practical cases, the above formulation is simplified and constraints (2) are removed. The resulting formulation is that of a *set covering* problem. For this model, a binary vector $[x_j]$ fulfilling constraints (1) defines a *cover*. Existence of *overcovers* is allowed by the '$\geqslant$' type constraints, meaning that more that one duty covers the same piece-of-work.

The *set partitioning* model is a special case in which constraints (1) are equality constraints. In practice, this second model forces that there is only one driver in each vehicle at any time. The main difficulty of this particular model is that, if a limited number of generated duties is available, we cannot usually guarantee that there is a feasible solution.[11] Therefore it is common to start by applying the set covering approach, even if the number of overcovers in the final solution may often be very high.

The techniques involved in the resolution of these two problems are very similar. Sometimes it is possible to reduce the size of the problem by applying dominance rules.[11–13] It is also common to use the values of greedy heuristics to obtain good initial solutions.[12,13] The values of these initial feasible solutions will be upper bounds for the value of the optimal solution.

Then, a linear relaxation is used to calculate a lower bound for the optimal solution.[14] The gap between the lower and the upper bounds can be tightened by the application of primal and dual greedy heuristics.[13] Lagrangian relaxation is also used to improve the lower bounds.[14]

A relaxed set partitioning model

In practice, these two models present several problems. Although the set partitioning model is the best suited for the bus driver scheduling problem, we cannot guarantee the existence of a feasible solution (and in many real problems it really does not exist). The relaxation to the set covering problem overcomes this difficulty. Unfortunately, this approach results in a solution that often contains too many overcovers. Overcovers are not attractive, since they correspond to duty idle times. Moreover, in a real problem, when a piece-of-work is covered by several duties, we have a new decision process in which we have to decide which duty is going to be effective for that piece-of-work and which duties are going to be idle. For this reason the transport companies we have been working with have serious difficulties in the implementation of covering solutions.

In this paper we propose a relaxation of the set partitioning model in which *leftovers* or *uncovered pieces-of-work*, ie pieces-of-work not associated with or covered by any feasible duty, are allowed. Leftovers are obviously undesirable, as they correspond to trips with no drivers assigned, so they must be penalised in the objective function. In practice, leftovers are assigned to crews using extra work time, or grouped together with leftovers from other routes, sometimes ignoring constraints.

This *relaxed set partitioning model* is defined as follows:

$$\begin{aligned}
\text{minimise} \quad & \sum_{j\in P} c_j x_j + \sum_{i\in I} z_i y_i \\
\text{subject to} \quad & \sum_{j\in P} a_{ij} x_j + y_i = 1 \qquad \forall i \in I \\
& x_j \in \{0, 1\} \qquad \forall j \in P \\
& y_i \in \{0, 1\} \qquad \forall i \in I
\end{aligned}$$

where:

$I = \{i$: piece-of-work$\}$;

$P = \{j$: candidate feasible duty$\}$;

$c_j =$ cost of feasible duty j;

$z_i =$ penalty associated with not covering piece-of-work i;

$x_j = 1$, if duty j is in the solution,
$= 0$, otherwise;

$y_i = 1$, if piece-of-work i is not covered,
$= 0$, otherwise;

$a_{ij} = 1$, if piece-of-work i belongs to duty j,
$= 0$, otherwise.

This model may be seen as a generalisation of the *set packing problem* (with $z_i = 0$ and an appropriate transformation of the cost vector $c'_j = -c_j$).[15]

In this paper we use the set partitioning model to test the Genetic Algorithm with a group of standard test problems. For the real problems we use an extension of this model that includes some additional operational objectives and constraints. For instance, the uncovered pieces-of-work have some features that are usually considered by each particular company in the construction of the objective function.

Genetic algorithms: a general local search heuristic model

General presentation

A Genetic Algorithm (GA) may be described as a mechanism that imitates the genetic evolution of species. GAs were first introduced by Holland[16] and since then a lot of work has been done, namely in the OR field. One of the main differences between GAs and other local search heuristics (eg tabu search, simulated annealing) is that GAs search is based in a population of solutions not in a single solution.[17] The mechanisms used to perform this search and to allow the evolution to new generations are called *operators.* The main operators are *selection, crossover* and *mutation.* Another characteristic of GAs is that solutions must be coded in finite length strings over a finite alphabet. These strings are called *chromosomes.* In this paper we will use the terms chromosome, individual and solution with the same meaning.

A GA starts with a population of chromosomes (ie individuals or solutions) that can be created randomly or by processes that use prior knowledge of the specific problem. All the chromosomes are evaluated using some criteria that help us to rank the individuals according to their relative *fitness.* The *selection operator* is then applied to choose the better individuals. There are a lot of possible strategies to select the chromosomes that will create new individuals, hopefully better than their parents. The *crossover operator* is responsible for the recombination process. Each time the crossover operator is applied, two or more of the selected individuals are chosen to mate and it combines pieces of them to form new and possibly better individuals. According to Goldberg[18] the combination of crossover and selection operators is the core of the innovation process that explains the success of GAs.

While crossover creates new individuals by recombining two or more parents, the *mutation operator* performs changes in a single individual. Mutation randomly searches in the neighbourhood of a particular solution. Its role is very important to guarantee that the whole search space is reachable. There are several ways of implementing crossover and mutation operators. We can incorporate specific knowledge of the problem and heuristics that guide the search to improve certain features of the solutions.

The new individuals are evaluated and they will replace the worst individuals of the current population. This process is repeated until a satisfactory solution is achieved or a prefixed number of generations is performed.

Hybrid genetic algorithms

In this paper we will distinguish between a traditional GA and a hybrid GA. A traditional GA usually uses a binary coding alphabet and the crossover and mutation operators do not include any knowledge about the structure and domain of the problem. In a hybrid GA we incorporate problem-specific knowledge in the operators or in the coding scheme. Hybrid GAs are less general than traditional ones, but they usually outperform these when applied to difficult problems.[19,20]

The application of GAs to constrained problems, such as most combinatorial optimisation problems, involves special attention to handling constraint satisfaction. Usually, traditional GA operators do not produce feasible solutions, even if both parents are feasible.

There are several ways to handle the infeasibility of solutions:

(i) Generation of all the possible solutions, discarding the infeasible ones. This approach allows the application of a traditional GA,[21] but in highly constrained problems it is sometimes very difficult to obtain a single feasible solution.

(ii) The design of a coding scheme that ensures the feasibility of the solutions.[19] It would be an interesting alternative, but it is not always possible to design a coding scheme guaranteeing that the application of the traditional crossover and mutation operators would maintain the feasibility of the solutions. Hence, this approach is usually associated with the design of specialised operators that incorporate problem-specific knowledge.[20]

(iii) The design of specialised operators that, starting from feasible solutions, only produce feasible solutions.[21,22] The design of such operators may be impossible for some problems and quite natural for others. For instance, for the set partitioning problem it is not possible to guarantee the construction of a new feasible solution based on two (or more) feasible ones. However, for the set covering problem[21,22] and for the relaxed partitioning problem it is quite simple to design these operators. A reasonable alternative to overcome this difficulty is to design specialised operators that try to 'minimise' the extent or the number of violated constraints.

(iv) The application of a penalty function to handle constraints violation.[23] This procedure is very common but it is difficult to choose the appropriate penalty function and its weights.

(v) The application of a heuristic procedure, a *repair operator*, to transform each unfeasible solution into a feasible one. Although this approach can be computationally expensive, it has been successfully applied to several combinatorial optimisation problems.[24]

(vi) The design of a function (the *unfitness function*) to measure the unfeasibility of the solutions.[24] The separation between the fitness and unfitness functions may be very useful when it is difficult to build feasible solutions.

In our hybrid genetic algorithm we have used procedures (iii) and (iv) for the set partitioning problem and procedures (ii) and (iii) for the relaxed set partitioning problem.

Related work

In the last few years, genetic algorithms have been applied to a wide number of OR problems, including the set covering[25,26] and the set partitioning[23,24] problems. In particular, for the set partitioning problem, Levine[23] applied a parallel GA to real airline crew scheduling test problems.[27] Chu[24] has developed a new GA that incorporates a function to deal with infeasible solutions (the unfitness function) and he has compared his algorithm with Levine's using the same test problems. In his work, Chu proposed a genetic algorithm approach for several combinatorial optimisation problems.

For the bus driver scheduling problem, the IMPACS and TRACSII research team[22,28,29] have been using GAs with the aim of reducing the time and computer limitations involved in their traditional approach based on a set covering ILP model. Portugal[30] experimented several metaheuristics, namely tabu search and genetic algorithms, to solve the same model.

A genetic algorithm for the bus driver scheduling problem

Coding scheme

For our purposes, the bus driver scheduling problem is basically composed of two types of information, the set of candidate *duties* and the set of *pieces-of-work*. In general the number of elements of the first set is considerably larger than the second set (there are much more duties than pieces-of-work), but each solution is composed of a relatively small number of duties.

The chromosome structure that is most commonly used[23–25] is given by a binary vector with fixed length, where each gene is associated with a duty, and its value is either *one* or *zero*, according to the presence or absence of the duty in the solution. For example, if for a given problem we have 15 candidate duties and a solution is composed by duties 1, 5, 8, 11 and 12, the corresponding chromosome will have length 15 and the following structure: 100010010011000.

One advantage of this coding scheme comes from the fact that a binary representation is used, which makes the data structures extremely simple, economical and easy to manipulate by the standard operators. However, each chromosome contains only the information about the subset of duties that represent one solution. To get any additional information, such as the pieces-of-work in each duty, one needs to perform a set of rather time consuming operations. Moreover, this coding scheme does not ensure, by itself, the feasibility of the solutions.

As an alternative, we propose a *pieces-of-work coding* scheme (PWC), associating the two fundamental types of information contained in a solution: the duties and the set of pieces-of-work. Each piece-of-work corresponds to a gene of the chromosome, and each gene is characterised by the duty that covers the piece-of-work in that particular solution. The genes are ordered by bus and for each bus they are ordered by time.

Figure 1 represents a solution for a problem with 14 pieces-of-work. The value of each gene corresponds to the duty that covers the piece-of-work (eg the duty **5** covers pieces-of-work **2** and **11**). When it is not possible to cover all the pieces-of-work those that remain uncovered are given the value 0. Obviously, this particular solution is infeasible for the set partitioning problem, but it is a feasible *relaxed partitioning solution.*

This coding always represents feasible relaxed partitioning solutions (they may contain leftovers, but no overcovers). Naturally, specialised crossover and mutation operators had to be designed for this particular coding scheme.

The approach using relaxed partitioning solutions is quite simple and it seems suitable for solving the bus driver scheduling problem in the urban transport companies we have been working with. The adopted coding scheme, although not binary, is extremely simple and natural, making chromosome manipulation very easy by the specialised operators we have designed. Moreover, in many practical cases, feasible set partitioning solutions are obtained.

The fitness function

The bus driver scheduling problem can, in practice, involve several conflicting objectives. Usually it is solved as a set covering or as a set partitioning problem, where the objective function is the sum of the individual costs of each duty in the solution. The cost of a duty is computed as a function of the duration, size of the breaks, extra work, etc. In relaxed partitioning solutions, leftovers must also be penalised.

Gene	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	1	5	3	2	1	4	1	0	2	3	5	2	4	0

Figure 1 Solution with fourteen pieces of work.

In real companies there are a set of objectives and constraints that cannot be included in the cost of each duty. For example, some companies want to minimise the overall deviation from a target mean duty duration. Some other companies try to limit the number of duties of a certain type.

The way the penalty function is calculated depends also on the particular company. Since in practical problems leftovers cannot always be avoided, companies want to control features such as the duration of each leftover. Consider for example the following two solutions for the same problem:

Solution A: 1, 3, 0, 4, 3, 1, 5, 0, 9, 4, 5, 4, 9;

Solution B: 2, 3, 2, 7, 3, 0, 0, 0, 9, 8, 7, 8, 9.

Both solutions have leftovers (Solution A for pieces-of-work 3 and 8, and Solution B for pieces-of-work 6, 7 and 8) that must be penalised in the evaluation of each solution. Suppose that the pieces-of-work corresponding to the leftovers in Solution A have a duration of 30 minutes and 1 hour respectively, and they do not belong to the same vehicle schedule. The leftovers on Solution B have a duration of 30 minutes, 1 hour and 30 minutes, respectively, and they correspond to contiguous pieces-of-work in the same vehicle schedule. These three uncovered pieces-of-work can be interpreted as a single leftover with a total duration of 2 hours. This information may be very important if one wants to plan in advance how the leftovers will be assigned to the drivers, for instance through extra time or extra workers.

In our genetic algorithm, the most important (minimisation) criteria used are the following:

(i) total cost of duties;
(ii) number of single leftovers;
(iii) number of grouped leftovers (if we assume that the successive uncovered pieces-of-work are grouped as a single leftover);
(iv) total duration of leftovers;
(v) deviation from a target mean duration of duties and leftovers.

The user can build the fitness function and change the default values (weights) according to the criteria of his own company. At present, all objectives and penalty terms are merged linearly in a single fitness function, although the user can change their weights dynamically as the algorithm is running. Our experience showed that merging too many criteria in the fitness function is not advantageous, because the individual weights lose strength in the overall function. The user can control the algorithm by choosing the appropriate criteria at each run, giving the appropriate weights to the different criteria (thus defining a multi-objective fitness function), and by proceeding until a satisfactory solution is achieved.

Generation of the initial population

Every chromosome in the initial population must be a feasible relaxed partitioning solution (ie it cannot have any overcover, although it may contain leftovers). The specialised operators will maintain the feasibility of the solutions.

The procedure to generate the initial population is very simple and works as follows. For each chromosome a list of the *available* duties is created. A duty is *available* if none of the pieces-of-work it covers is already covered by any duty in that solution. An available duty is randomly selected and inserted in the chromosome and this process is repeated until there are no more available duties. Naturally, every time a duty is added to the chromosome, the list of available duties is updated.

Parent selection scheme

The parent selection method assigns a probability for reproduction to each individual in a population, according to some rule that gives more opportunities to the individuals with better fitness. There are several different methods for parent selection. In this paper we have used the proportionate selection (roulette wheel) and the tournament selection (binary tournament).

The proportionate selection method assigns a probability of selection $ps(i)$ to each individual i in the population, according to the ratio of i's fitness to the overall population fitness:

$$ps(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)} \qquad i = 1 \ldots n$$

The simplest type of proportionate selection method, the *roulette-wheel selection*,[31] chooses individuals through n simulated spins of a roulette wheel. The roulette wheel contains one slot for each population element. The size of each slot is directly proportional to its respective $ps(i)$, and therefore population members with higher fitness values are likely to be selected more often than those with lower fitness values.

The proportionate selection can eventually promote the premature convergence of the algorithm, if there is a 'super-individual' (a individual with a fitness value much higher than the others). To avoid this situation we have used a *linear normalisation* of the fitness function. In this technique the chromosomes are ordered by decreasing values of the evaluation function. The fitness values begin with a constant value and decrease linearly. The constant initial value was fixed at the population size and the decrement rate at 1.

The tournament selection method is much simpler and more efficient than the proportionate selection method since it does not require calculating the fitness values of each individual. In this method k elements are randomly chosen from the population. The selected parent will be the individual with higher fitness value. We have used the binary tournament method with $k = 2$.

These two methods performed very similarly and we cannot claim that one is better than the other. Linear normalisation of the fitness function in the proportionate selection method was crucial to avoid premature convergence. In most tests we have used tournament selection because it is faster, but we cannot say that it led to better results.

Crossover

The crossover operator is applied to pairs of selected chromosomes in order to generate one, two or more 'children'. Usually, we restrict the offspring to a single child in order to reduce the possibility of premature convergence. We have designed a specialised operator that takes advantage of the coding scheme and maintains the solution with no overcovers. The procedure is divided into two phases. In the first phase, we adopt an approach similar to the one used to generate the members of the initial population. A duty of one of the parents is randomly selected. If it is available, it is added to the chromosome under construction. The process is repeated until none of the duties in both parents is available. In the second phase we try to reduce the leftovers that may still exist. For each uncovered piece-of-work, randomly chosen, the subset of available generated duties that cover that piece-of-work is constructed. One of these duties is selected according to some rule. Hence, while there is still a not covered piece-of-work, an available duty from the set of all the generated duties is selected and added to the chromosome.

This operator has several different versions according to the procedures we have used to select the available duties. These procedures may be completely random or they can be based on a given criterion. For example, we can select the duty that minimises the ratio *cost number of pieces-of-work*, the duty that covers the greatest number of pieces-of-work, or the longest duty. However, we noticed that, although the sorting procedures give very good solutions in the first generations, they easily lead to the premature convergence of the algorithm. Therefore, we can use the sorting procedures in the first generations, moving then to a random procedure (under the user's control).

Mutation

We have designed two different mutation operators. The first, called *basic_mutation*, is based on the same philosophy we have used for the crossover operator and works as follows. A small percentage of duties is removed from the selected chromosome and a list with all the duties that have become available is built. Then the process of selecting and inserting an available duty into the chromosome is repeated until there are no more available duties for that chromosome. Special attention must be given to this procedure in order to avoid that a chromosome identical to the original chromosome is created. Again, the process of selecting the available duties may be random or based on a sorting criterion.

The second mutation operator, called *improve_mutation*, is a knowledge-based operator and it is used to directly improve a given solution through small changes in its neighbourhood. In this operator, for each free piece-of-work, we try to fill it with a duty that also covers the same pieces-of-work of one of the adjacent duties. In the example presented in Figure 1, we have two free pieces-of-work, corresponding to genes 8 and 14. If this chromosome was selected for *improve_mutation*, we would try to find duties that cover the same pieces-of-work of duties 1, 2 or 4. Therefore, suppose that duty 6 covers pieces-of-work 1, 5, 7 and 8. Then, duty 1 would be replaced by duty 6, and we would have one less leftover in the solution (see Figure 2).

We have associated a very low probability to this operator, although in some cases it might seem quite attractive to apply it to the best chromosomes in the population. However, we noticed that this procedure could lead the algorithm to premature convergence.

Population replacement scheme

A fundamental decision in a genetic algorithm implementation is the choice of the strategy for the replacement of populations. The replacement techniques (generation replacement, steady-state replacement) can be controlled through one single parameter in the interval (0,1], the *generation gap*, which is the proportion of individuals in the population to be replaced in each generation. When the value of the generation gap is 1, *generation replacement* is the strategy adopted and the whole population is replaced. When the best element of the population is kept for the next generation, we have a generation replacement with *elitism.*[19]

In a *steady-state replacement* scheme, a percentage of the population (given by the generation gap parameter) is replaced by the new offspring. The elements to be replaced can be selected randomly or according to the inverse fitness (the worst individuals are replaced). Typically, in steady-state replacement, only a few individuals are replaced in each generation.

Gene	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	6	5	3	2	6	4	6	6	2	3	5	2	4	0

Figure 2 Mutated chromosome.

We have tried different values for the generation gap, and steady-state replacement was the strategy normally adopted with a generation gap of at most 0.25 (this means that no more than 25% of the population is replaced in each generation).

Another important decision in the replacement procedure concerns duplicate solutions. If duplicate solutions are allowed, after a few generations the whole population is formed by identical chromosomes. If duplicate solutions are not allowed, the algorithm may become significantly slower, because we must compare each child with all the solutions in the current population. In our work, we allow duplicate solutions in the population and control premature convergence of the population through a sorted array W with the k worst individuals in the population. When the fittest element of W has the same fitness value as the best element in the population, premature convergence is detected and a total replacement with elitism is performed, meaning that we keep the best element in the population and replace all the other elements. The value of k varies between 10–50% of the population size.

Finally, it should be noted that all the parameter values can be combined during the execution of the algorithm. Therefore, according to the algorithm evolution, one can change the adopted strategies in order to control and tune the mechanisms for diversification and intensification of the search.

Computational results

The genetic algorithm was coded in C++ and the tests were performed on a 200 MHz Pentium with 32 MB RAM. The algorithm was tested on different groups of problem instances. The first group is a subset of the problems used by Hoffman and Padberg[32] that are available electronically in OR-Library.[27] We have chosen this set of instances because we did not have any computational results for the real driver scheduling problems with exact algorithms for comparison purposes. Therefore it should be noted we have performed these tests because, although the bus driver scheduling problem is different and more complex than those presented in the literature, this seemed to be the only way to compare results with previous work.

The second group is composed of real problems that were provided by a set of Portuguese transport companies. These companies are using a decision support system, the GIST system, that includes a module for the bus driver scheduling problem. The algorithms available in the GIST system are based on a set covering formulation of the problem. The solutions provided by the system do not satisfy the user's practical requirements because they generate too many overcovers. Since the GIST system also includes a very interactive graphical interface, most companies solve problems using the graphical and editing features, in what might be called a 'semi-manual' approach. As the solutions produced in this semi-manual approach are, in general, considered by the users as quite satisfactory, we decided to compare directly our GA solutions with those.

Set partitioning test problems

The first set of test problems refers to real airline crew scheduling problems, formulated as pure set partitioning problems with no additional information.

The fitness function we used only includes the sum of the individual costs of the duties and a penalty function for the infeasible set partitioning solutions. Let s be a solution for the set partitioning problem. The fitness function is defined as:

$$f(s) = cost(s) + penal(s)$$
$$penal(s) = \sum_{i=1}^{m} M\left(1 - \sum_{j=1}^{n} a_{ij}x_j\right)$$

where M is a constant empirically chosen. Note that each member of the sum in $penal(s)$ is either M or zero, since overcovers are not allowed. The term $cost(s)$ corresponds to the sum of the individual costs of the duties in the solution.

For this set of problems we have not used the *improve_ mutation* operator, since the necessary information about the start/finish time of the pieces-of-work is not available.

Ten independent trials of the algorithm were performed for each problem. For each trial, the algorithm was stopped whenever the optimal solution was found or after 2000 generations. The population size was fixed to 100 individuals, the crossover rate was set to 0.5, the mutation rate was set to 0.01 and the generation gap was set to 0.9. These parameter values were found through some experimentation.

We have used the pre-processing procedures given by Hoffman and Padberg[32] to reduce the original sizes of the problems. In Table 1 we present the computational results obtained for the set of selected problems after the reducing routines.

The *No Opt* column presents the number of optimal solutions found. The *BestFeas* column is either the best feasible solution found, or *Opt*, when the optimum was found. *Gap%* is the percentage deviation from the optimal solution of the best solution found. *AvgFeas* is the average value of all the best solutions found. *AvgGap%* is the average percentage deviation from the optimal solution. *Levine's Best* is the best solution found by Levine.[23] *Run time* is the average execution time in CPU seconds.

The algorithm found feasible solutions for all the tested problems. It failed to find the optimal solution in 3 problems. In most cases, the average percentage deviation from the optimal solution was around 1%. These results show that our genetic algorithm is effective for the set partitioning problem, although its results are worse than those presented by Chu and Beasley.[24] In fact, these authors managed to find the optimum for all these problems. Computational times are of the same order of magnitude. We present the run times in the last column of Table 1. For 3 problems, marked with an X we did not find the optimal solution. Our results are clearly better than the results obtained by Levine[23](see second to last column of Table 1). This author failed to find feasible solutions in 3 problems (marked with an X).

Real bus driver scheduling test problems

For this set of tests we have chosen some examples from two companies, having compared their currently implemented solutions with those provided by our genetic algorithm. For these problems all the information about the duties and the pieces-of-work was available, such as the start/finish time and start/finish location of each piece-of-work and the cost, type and duration of each duty, together with information on buses and lines.

The criteria used to compare the solutions were the cost and the number of the duties, the number of leftovers, the percentage of the schedule covered by feasible duties and the average duration of the duties in the solution. We did not directly compare our GA with the algorithms already integrated in the GIST system, because they are based on a different approach (a set covering model) that almost never provides partitioning solutions.

The comparison between 'manual' solutions and automatic ones is a difficult task when the criteria are not just the costs of the solutions. Comparisons are not completely objective because manual solutions implicitly take into account additional, non-expressed criteria that obviously are not considered in the objective function. Usually, manual solutions take a few days to prepare and they are adjusted daily, trying to improve some of their features. Experts with years of experience perform the adjustments, sometimes breaking the rules taken into account in previous stages.

Generally, the main objective of the transport company is to obtain a solution with a minimum number of duties and leftovers. The trade-off between the number of duties and the number of leftovers is not a clear question for all the companies, but most of them undoubtedly prefer to have one more duty than an extra leftover. However, planners have an overall perspective of a solution that incorporates other criteria such as *balance* (measured by the deviation to the average duration of the duties), *operational compatibility* (involving criteria such as the average duration of leftovers, the location and time the leftovers can occur) and *desirability* (percentages of each type of duty, individual features of each duty).

In Table 2 we present 15 problems from the two major Portuguese urban mass transit companies: STCP and Carris. In Table 3 we present the manual solution currently in use by the companies. Usually, companies could need several

Table 1 Computational results for the set partitioning test problems

Problem	*Rows*	*Columns*	*Density*	*Optimum*	*No Opt*	*No Feas*	*Best Feas*	*Gap%*	*Avg Feas*	*Avg Gap%*	*Levine's best*	*Run time*
NW07	36	3108	21.87	5476	5	10	Opt	0	6148.4	10.94	Opt	52
NW08	24	356	22.35	35894	10	10	Opt	0	Opt	0	37078	2
NW09	40	2305	16.05	67760	0	10	69016	1.82	69018.4	1.82	X	X
NW10	24	659	21.25	68271	10	10	Opt	0	Opt	0	X	4
NW11	39	6488	16.68	116256	0	10	116925	0.57	117283.2	0.88	X	X
NW12	27	454	19.06	14118	10	10	Opt	0	Opt	0	15110	10
NW15	29	463	21	67743	10	10	Opt	0	Opt	0	Opt	0.8
NW19	40	2145	21.59	10898	1	10	Opt	0	11645	6.41	11060	100
NW20	22	566	24.99	16812	10	10	Opt	0	Opt	0	16965	7.1
NW21	25	426	24.33	7408	10	10	Opt	0	Opt	0	Opt	4.3
NW22	23	531	24.22	6984	10	10	Opt	0	Opt	0	7060	0.75
NW23	18	473	24.86	12534	10	10	Opt	0	Opt	0	Opt	4.8
NW24	19	926	33.22	6314	5	10	Opt	0	6342	0.44	Opt	5.2
NW25	20	844	30.15	5960	5	10	Opt	0	6207	3.99	Opt	9.8
NW26	23	542	23.12	6796	4	10	Opt	0	6830	0.5	Opt	6.2
NW27	22	926	30.76	9933	10	10	Opt	0	Opt	0	Opt	8.3
NW28	18	825	38.43	8298	9	10	Opt	0	8337	0.47	Opt	4.1
NW29	18	2034	30.99	4274	0	10	4324	1.16	4386.2	2.56	4378	X
NW30	26	1884	29.81	3942	5	10	Opt	0	4025	2.06	Opt	49.7
NW31	26	1823	29.21	8038	7	10	Opt	0	8044.4	0.08	Opt	38.2
NW32	18	251	25.81	14877	10	10	Opt	0	Opt	0	Opt	1.2
NW33	23	2415	30.75	6678	6	10	Opt	0	6679.6	0.02	Opt	20.1
NW34	20	750	28.16	10488	5	10	Opt	0	10600.5	1.06	Opt	17
NW35	23	1403	27.02	7216	3	10	Opt	0	7300.4	1.16	Opt	25.3
NW36	20	1408	36.14	7314	10	10	Opt	0	Opt	0	7336	48.7
NW37	19	639	25.89	10068	5	10	Opt	0	10179.6	1.1	Opt	23
NW38	23	911	31.44	5558	3	10	Opt	0	5585.6	0.49	Opt	27.1
NW39	25	567	26.28	10080	10	10	Opt	0	Opt	0	Opt	3.2
NW40	19	336	26.86	10809	4	10	Opt	0	10846.8	0.35	10848	19.1
NW41	17	177	22.33	11307	10	10	Opt	0	Opt	0	Opt	1.6
NW42	23	895	26.05	7656	3	10	Opt	0	7686.4	0.4	Opt	6.6
NW43	17	982	26.43	8904	3	10	Opt	0	9018.8	1.27	9146	16.7

Table 2 Real bus driver scheduling test problems

Problem			
Problem no.	*Rows*	*Columns*	*Total length*
Problem 1	118	2487	166:46
Problem 2	102	568	117:31
Problem 3	208	15144	166:47
Problem 4	138	7158	105:39
Problem 5	195	4227	172:56
Problem 6	140	6099	84:35
Problem 7	137	6828	77:40
Problem 8	161	15995	130:56
Problem 9	150	12976	142:34
Problem 10	132	9783	120:12
Problem 11	158	6620	132:27
Problem 12	192	21533	114:10
Problem 13	185	8677	195:24
Problem 14	251	9635	244:57
Problem 15	131	4949	135:15

hours to build each one of these solutions. Table 4 shows the 'best solution' obtained with the hybrid GA. These solutions were obtained interactively using the GA application that allows the user to change dynamically the criteria and the parameters of the algorithm while it is running. This application also allows the user to save intermediate solutions, for future analysis. In these conditions we have fixed a limit of 15 minutes for the algorithm to achieve the 'best solution'.

In all problems, the hybrid GA found solutions with a lower cost. Moreover, the number of leftovers is smaller and the percentage of the schedule covered by feasible duties is higher or at least equal to the manual solutions. For the 15 problems, the GA found seven set partitioning feasible solutions, while there is only one feasible set partitioning solution in the company's actual solutions. Moreover it should be noted that the manual solutions very often include 'illegal' duties.

Conclusions

In this paper we have made an exhaustive experimental evaluation of genetic algorithms specially designed to solve the bus driver scheduling problem. Our genetic algorithm uses a new coding scheme for the relaxed partitioning problem and considers a complex objective function that incorporates the most relevant features of a quality solution.

The performance of this algorithm was evaluated with standard test airline crew scheduling problems, and with real problems from several medium-size Portuguese urban bus companies. The results show that genetic algorithms can quickly produce very satisfactory solutions, bringing automatic solutions closer to the planners' expectations. Solutions that could take several hours to build manually can now take minutes to reach. Therefore, users are actually fully using these algorithms in their daily planning activity.

It should also be noted that genetic algorithms offer a natural framework for computational parallelisation, provide satisfactory solutions at the end of each iteration, and the evaluation functions can become as complex as required. These features have to be further explored. Moreover, further comparisons with alternative approaches are going to be performed in the near future.

Table 3 Actual solutions for the real bus driver scheduling test problems

Problem	*Companies actual solution*						
Problem no.	*Cost*	*No. duties*	*No. leftovers*	*%Schedule covered*	*Total duration of solution*	*Average duration of duties*	*Total duration of leftovers*
Problem 1	135000	22	5	93.1	155:18	7:08	11:28
Problem 2	99540	15	3	90.4	106:15	7:05	11:16
Problem 3	125869	22	3	94.9	158:23	7:11	8:24
Problem 4	77337	14	3	93.4	98:43	7:03	6:56
Problem 5	149605	20	8	85.2	147:22	7:22	25:34
Problem 6	70426	10	4	87.8	74:16	7:25	10:19
Problem 7	80297	9	3	85.1	66:09	7:21	11:31
Problem 8	110645	16	6	85.2	111:37	6:58	19:19
Problem 9	94075	20	3	93.5	133:19	6:39	9:15
Problem 10	120347	16	2	95	114:16	7:08	5:56
Problem 11	312522	19	4	90.13	119:23	6:17	13:04
Problem 12	152850	18	0	100.0	114:10	6:20	0:00
Problem 13	255000	30	2	96.9	189:28	6:06	5:56
Problem 14	360000	35	8	87.8	215:14	6:08	29:43
Problem 15	138734	22	2	96.6	130:42	5:56	4:33

Table 4 Computational results for the real bus driver scheduling test problems

Problem	*GA solution*						
Problem no.	*Cost*	*No. duties*	*No. leftovers*	*%Schedule covered*	*Total duration of solution*	*Average duration of duties*	*Total duration of leftovers*
Problem 1	131033	23	3	98.1	163:32	7:06	3:14
Problem 2	85707	16	1	97.6	114:43	7:10	2:48
Problem 3	121878	23	1	99.2	165:30	7:11	1:17
Problem 4	75087	15	0	100.0	105:39	7:02	0:00
Problem 5	135934	23	4	95.8	165:49	7:12	7:07
Problem 6	55251	11	0	100.0	84:35	7:41	0:00
Problem 7	78248	10	2	96.2	74:39	7:27	3:01
Problem 8	104964	19	2	98.14	128:30	6:45	2:26
Problem 9	85322	21	1	98.25	140:04	6:40	2:30
Problem 10	109861	17	0	100.0	120:12	7:04	0:00
Problem 11	200894	22	0	100.0	132:27	6:01	0:00
Problem 12	95359	18	0	100.0	114:10	6:20	0:00
Problem 13	224650	31	0	100.0	195:24	6:18	0:00
Problem 14	265000	39	2	97.4	238:41	6:07	6:16
Problem 15	123092	22	0	100.0	135:15	6:08	0:00

References

1 Wren A and Rousseau J-M (1995). Bus driver scheduling – an overview. In: Daduna JR, Branco I and Paixão JMP (eds). *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems, No. 430. Springer: Berlin, pp 173–187.

2 Parker ME and Smith BM (1981). Two approaches to computer crew scheduling of public transport. In: Wren A (ed). *Computer Scheduling of Public Transport.* North-Holland: Amsterdam, pp 193–221.

3 Ryan DM and Foster BA (1981). An integer programming approach to scheduling. In: Wren A (ed). *Computer Scheduling of Public Transport.* North-Holland: Amsterdam, pp 269–280.

4 Falkner JC and Ryan DM (1992). EXPRESS: Set partitioning for bus crew scheduling in christchurch. In: Desrochers M and Rousseau J-M (eds). *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems, No. 386. Springer: Berlin, pp 359–378.

5 Rousseau J-M and Desrosiers J (1995). Results obtained with crew-opt, a column generation method for transit crew scheduling. In: Daduna JR, Branco I and Paixão JMP (eds). *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems, No. 430. Springer: Berlin, pp 349–358.

6 Rousseau J-M, Lessard R and Blais J-Y (1985). Enhancements to the HASTUS crew scheduling algorithm. In: Rousseau J-M (ed). *Computer Scheduling for Public Transport*, No. 2. North-Holland: Amsterdam, pp 295–310.

7 Williamson RH (1985). BUSMAN: The United Kindom's integrated approach to transit scheduling, In: Rousseau J-M (ed). *Computer Scheduling of Public Transport*, No. 2. North-Holland: Amsterdam, pp 19–43.

8 Cunha JF and Sousa JP (2000). The bus stops here – GIST: decision support system for public transport planning. *OR/MS Today* **27**: 48–53.

9 Fischer M (1981). Lagrangean relaxation methods for combinatorial optimisation. *Mngt Sci* **27**: 1–18.

10 Carraresi P, Gallo G and Rousseau, J-M (1983). Relaxation approaches to large scale bus driver scheduling problems. *Transport Res* **16B**: 383–397.

11 Garfinkel R and Nemhauser G (1972). *Integer Programming.* John Wiley and Sons: New York.

12 Paixão JP (1983). Algorithms for Large Scale Set Covering Problems. PhD thesis, Imperial College, London.

13 Beasley JE (1987). An algorithm for the set covering problem. *Eur J Opl Res* **31**: 85–93.

14 Paixão JP and Pato MV (1989). A structural Lagrangean relaxation for two duty period bus driver scheduling problem. *Eur J Opl Res* **39**: 213–222.

15 Darby-Dowman K and Mitra G (1985). An extension of set partitioning with application to scheduling problems. *Eur J Opl Res* **21**: 200–205.

16 Holland, J (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press: Michigan.

17 Pirlot M (1992). General local search heuristics in combinatorial optimization: a tutorial. *Belg J Op Res Stat Comput Sci* **32**: 1–2.

18 Goldberg, DE (1998). The design of innovation: lessons from genetic algorithms, lessons for the real world. *IlliGAL Report No. 98004.*

19 Davis L (ed) (1991). *Handbook of Genetic Algorithms.* Van Nostrand Reinhold: New York.

20 Grefenstette JJ (1987). Incorporating problem specific knowledge into genetic algorithms. In: Davis L (ed). *Genetic Algorithms and Simulated Annealing.* Morgan Kaufmann: Los Altos, Calif, pp 42–60.

21 Galvão Dias T (1995). Aplicação de Algoritmos Genéticos ao Problema da Geração de Serviços de Tripulações (Genetic algorithms applied to the crew scheduling problem, in Portuguese). MSc thesis, Faculdade de Engenharia da Universidade do Porto, Porto, Portugal.

22 Clement R and Wren A (1995). Greedy genetic algorithms, optimising mutations and bus driver scheduling. In: Daduna JR, Branco I and Paixão JMP (eds). *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems, No. 430. Springer: Berlin, pp 213–235.

23 Levine D (1994). A parallel genetic algorithm for the set partitioning problem. PhD thesis, Department of Computer Science, Illinois Institute of Technology.

24 Chu P (1997). A genetic algorithm approach for combinatorial optimisation problems. PhD thesis, The Management School, Imperial College of Science, Technology and Medicine, London.

25 Beasley JE and Chu PC (1996). A genetic algorithm for the set covering problem. *Eur J Opl Res* **94**: 392–404.

26 Al-Sultan KS, Hussain MF and Nizami JS (1996). A genetic algorithm for the set covering problem. *J Opl Res Soc* **47**: 702–709.

27 Beasley JE (1990). OR-library: distributing test problems by electronic mail. *J Opl Res Soc* **41**: 11, 1069–1072.

28 Kwan ASK, Kwan RSK and Wren A (1999). Driver scheduling using genetic algorithms with embedded combinatorial traits. In: Wilson NHM (ed). *Computer-Aided Transit Scheduling*, Lecture Notes in Economics, Vol. 471. Springer-Verlag: Berlin, pp 81–102.

29 Wren A and Wren DO (1995). A genetic algorithm for public transport driver scheduling. *Comput Opn Res* **22**: 101–110.

30 Portugal R (1998). Metaheuristicas para a Geracùão de Serviços para o Pessoal Tripulante (Metaheuristics for the bus driver scheduling problem, in Portuguese). *MSc thesis*, Faculdade de Ciências de Universidade de Lisboa, Lisboa, Portugal.

31 Goldberg DE (1989). *Genetic Algorithms in Search. Optimisation and Machine Learning*. Addison-Wesley: Reading, MA.

32 Hoffman K and Padberg M (1993). Solving airline crew scheduling problems by branch and cut. *Mngt Sci* **39**: 657–682.

Received October 1998;
accepted September 2001 after two revisions