# A Genetic Algorithm for the Bus Driver Scheduling Problem

*Teresa G. Dias*[*] *Jorge P. Sousa*[*] *João F. Cunha*[*]

[*] Faculdade de Engenharia da Universidade do Porto / INEGI
Rua Dr. Roberto Frias,4200-465 Porto, Portugal
Email: {`tgalvao, jsousa, jfcunha`}`@fe.up.pt`

## 1 Introduction

The bus driver or crew scheduling problem is an extremely complex part of the operational planning process of transport companies. Its combinatorial nature and the large size of real 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 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*, i.e., places where drivers can be replaced. Several successive pieces-of-work may form a leg, i.e., 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 satisfy 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, that may therefore encompass different characteristics that are usually very difficult to handle by traditional algorithms. Linearity or differentiability conditions are not imposed to the objective function that may express several complex criteria, such as setting targets for average duty length and range. Infeasible solutions may also be allowed, but the objective function can include penalties trying to avoid solutions with undesirable features.

## 2 Models and algorithms for the bus driver scheduling problem

Traditionally the bus driver scheduling problem is formulated as a set covering/partitioning problem. In the set covering model *overcovers* are allowed, meaning that more that one duty covers the same piece-of-work. The set partitioning problem 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 [3]. 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.

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 work we propose a relaxation of the set partitioning model in which *leftovers* or *uncovered pieces-of-work*, i.e., pieces-of-work not associated 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:

$$\begin{array}{lll}
\textit{minimise} & \sum_{j \in P} c_j x_j + \sum_{i \in I} z_i y_i & \\
 & & \\
\textit{subject to} & \sum_{j \in P} a_{ij} x_j + y_i = 1 & \forall \mathrm{i} \in \mathrm{I} \\
 & x_j \in \{0,1\} & \forall \mathrm{j} \in \mathrm{P} \\
 & y_\mathrm{i} \in \{0,1\} & \forall \mathrm{i} \in \mathrm{I}
\end{array}$$

where:

| | | |
|---|---|---|
| I | = | $i$: piece-of-work; |
| P | = | $j$: candidate feasible duty; |
| $c_j$ | : | cost of feasible duty $j$; |
| $z_i$ | : | penalty associated to 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 $c'_j = -c_j$) [1].

In this work 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 usually are considered by each particular company in the construction of the objective function (see section 3.2).

## 3 A genetic algorithm for the bus driver scheduling problem

In this work 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 [2].

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

## 3.1 Coding scheme

For our purposes, the bus driver scheduling problem is basically composed by 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 by a relatively small number of duties.

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.

| 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: a chromosome

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 (e.g. 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). Specialised crossover and mutation operators have been designed for this coding.

## 3.2 The fitness function

The bus driver scheduling problem can, in practice, involve several conflicting objectives. 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 length. Some other companies try to limit the number of duties of a certain type.

In our GA we consider a significant set of different criteria that can be divided in two groups: criteria involving duties and criteria involving leftovers.

*Criteria involving the duties*

(i) minimise the total cost of duties: the costs are calculated by the company based on real values;

(ii) minimise the number of duties;

(iii) maximise the total length of the schedule covered;

(iv) maximise the ratio (total length of schedule covered / number of duties).

*Criteria involving the leftovers*

(v) minimise the number of single leftovers;

(vi) minimise the number of grouped leftovers (if we assume that the successive uncovered pieces-of-work are grouped as a single leftover);

(vii) minimise total length of leftovers.

*Additional criteria that can be "viewed" as goals*

(viii) a target mean duration for each type of duties;

(ix) a target mean duration for all the duties in the solution;

(x) a minimum and maximum percentage of duties of each type;

(xi) a target mean duration of the leftovers;

(xii) a maximum and/or minimum duration of the leftovers.

The user can build the fitness function choosing between the different criteria and defining his own priorities. He can choose between a single fitness function in which all objectives and penalty terms are merged linearly in a single function, or a multi-objective fitness function that can be built through an interactive process that helps him to specify the priorities of his company. The user can control the algorithm choosing the appropriate criteria at each run, giving the appropriate weights to the different criteria (thus defining a multi-objective fitness function), saving each solution obtained, and proceeding until a satisfying solution is achieved.

### 3.3 Generation of the initial population

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.

### 3.4 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 the parent selection. In this work we have used the proportionate selection (roulette-wheel) and the tournament selection (binary tournament).

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.

### 3.5 Crossover operator

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 with the aim being 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 in two phases. In the first phase, we adapt an approach similar to the one used to generate the members of the initial population. A duty of either one of the parents is 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, randomly chosen piece-of-work, the subset of available generated duties that cover that piece-of-work is constructed. One of these 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.

### 3.6 Mutation operators

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 are removed from the selected chromosome and a list with all the duties that became 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.

### 3.7 Population replacement scheme

A fundamental decision in the implementation of 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 (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. However, when the best element of the population is kept for the next generation, this variation is called a generation replacement with *elitism* [2].

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.

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).

Finally, it should be noted that all the parameter values could 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.

## 4 Computational results

The Genetic Algorithm was coded in C++ and the tests were performed on a 200 MHz Pentium with 32Mb Ram. We have chosen two different groups of problem instances. The first group is a sub-set of the set partitioning problems used by Hoffman and Padberg [4] that are available electronically on the OR Library [5]. We chose this set of instances because for the real driver scheduling problems we did not have any computational results with exact algorithms for comparison purposes. The algorithm found feasible solutions for all the 32 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.

The second group is composed of real problems that were provided by a set of Portuguese transport companies. 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.

## 5 Conclusions and future work

In this work we have made an exhaustive experimental evaluation of Genetic Algorithms specially designed to solve bus driver scheduling problems. 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 real problems from several medium size Portuguese urban bus companies. The results prove that Genetic Algorithms can quickly produce very satisfactory solutions, bringing automatic solutions closer to the planners expectations.

It should also be noted that Genetic Algorithms offer a natural framework for computational parallelisation, provide feasible 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 next future.

## References

[1] Darby-Dowman K and Mitra G, An extension of set partitioning with application to scheduling problems, European Journal of Operational Research, 21,pp 200–205, 1985.

[2] Davis L (ed), Handbook of Genetic Algorithms, Van Nostrand Reinhold: New York, 1991.

[3] Garfinkel R and Nemhauser G, Integer Programming, John Wiley and Sons: New York, 1972.

[4] Hoffman K and Padberg M, Solving airline crew scheduling problems by branch and cut, Management Science, 39, 6, pp 657-682, 1993.

[5] Beasley JE, OR-Library: distributing test problems by electronic mail, Journal of the Operational Research Society, vol.41, no.11, pp 1069-1072, 1990.