A Hybrid Metaheuristic for the Bus Driver Rostering Problem

Vítor Barbosa [1,2], Ana Respício [2,3] and Filipe Alvelos [4,5]
[1]Escola Superior de Ciências Empresariais, Instituto Politécnico de Setúbal, Setúbal, Portugal
[2]Centro de Investigação Operacional, Faculdade de Ciências da Universidade de Lisboa, Lisboa, Portugal
[3]Departamento de Informática, Faculdade de Ciências da Universidade de Lisboa, Lisboa, Portugal
[4]Departamento de Produção e Sistemas, Universidade do Minho, Braga, Portugal
[5]Centro Algoritmi, Universidade do Minho, Braga, Portugal

Keywords: Hybrid Metaheuristic, Column Generation, Genetic Algorithms, Rostering.

Abstract: This paper presents a new decomposition model for the Bus Driver Rostering Problem and proposes the hybridization of column generation and genetic algorithms to achieve good quality rosters in short time. The decomposition model is based on the definition of a subproblem for each driver, which is responsible for the creation of valid work-schedules for the rostering period. Column generation is used to obtain an optimal linear solution. This solution and the subproblems' solutions obtained during the column generation are then used by the genetic algorithm to find good quality combinations of drivers' schedules, i.e. good quality rosters. Computational tests show the efficiency and effectiveness of the proposed approach.

1 INTRODUCTION

Rostering consists in defining the "work-schedule" for each of the workers in a company for a given period. A roster is a plan presenting the work-schedules for all workers. A work-schedule defines, for each day, if the worker is assigned to work or has a day-off and, in the first case, which daily task has to be performed. The Rostering Problem arises because the company usually has diverse tasks to assign on each day, sometimes needing particular skills, and on the other hand, the labour and company rules (days-off, rest time, etc.) restricts the blind assignment of tasks to workers.

Rostering is addressed in many types of business as surveyed in (Ernst et al., 2004). A particular attention has been paid to nurse rostering (Burke et al., 2004); (Moz and Pato, 2007) and airline crew rostering (Kohl and Karisch, 2004).

In this paper we consider the Bus Driver Rostering Problem (BDRP). The literature about the BDRP is short (Moz et al., 2009). In fact, most of the papers focusing the bus drivers rostering address both rostering and shift scheduling as in (De Leone et al., 2010); (Dorne, 2008); (Rodrigues et al., 2006), where, before the rostering phase, there exists a phase where the shift/duties are built by defining the sequence of trips and rest time of each bus and only after the driver is assigned. In (Wren, 1996) the distinctions and similarities between scheduling, timetabling and rostering are discussed.

In this paper we consider the BDRP as described in (Moz et al., 2009), where it is assumed that the set of tasks (duties) to assign in each day are already defined by aggregating sets of consecutive trips and rest times. For each task, the start time and total duration should be considered to avoid the assignment of invalid consecutive tasks (according to labour rules), respect the maximum work time allowed and obtain the amount of paid overtime.

The BDRP and most rostering problems are NP-Hard combinatorial optimization problems (De Leone et al., 2010); (Dorne, 2008); (Moz et al., 2009), being computationally hard to obtain optimal solutions. To avoid the computational burden to achieve solutions by using exact methods, many authors approach the problem with heuristic methods which are usually faster in the achievement of good solutions. Examples of the use of non-exact methods can be found in (Burke et al., 2003); (De Leone et al., 2010); (Lučić and Teodorović, 2007); (Moz et al., 2009); (Ruibin et al., 2010).

We propose a non-exact method to address the BDRP where the column generation (CG) method (Barnhart et al., 1998); (Dantzig and Wolfe, 1960); (Desaulniers et al., 2005); (Desrosiers et al., 1984)

Barbosa V., Respício A. and Alvelos F..
A Hybrid Metaheuristic for the Bus Driver Rostering Problem.
DOI: 10.5220/0004283700320042
In *Proceedings of the 2nd International Conference on Operations Research and Enterprise Systems* (ICORES-2013), pages 32-42
ISBN: 978-989-8565-40-2

and a genetic algorithm (GA) (Holland, 1992); (Mitchell, 1996); (Reeves, 1997) are combined to obtain good quality rosters in short time.

We present an integer programming model adopted for the BDRP, named compact model because the growth of the number of constraints and variables when the size of the problem increases is bounded by a polynomial. We also present a decomposition model resulting from grouping blocks of independent constraints (related to a single driver) of the compact model into independent subproblems and connecting them in a restricted master problem where the variables represent valid driver work-schedules, obtained by solving the subproblems. Column generation is used to obtain a linear combination of solutions where all tasks are assigned. All the information generated during the CG process is used as a base for the GA. An individual (chromosome) represents a roster where each gene represents a driver according to the locus and the value of the gene (allele) identifies a subproblem solution (driver schedule). The linear solution may also be used to identify the subproblems' solutions included in the optimal linear solution, allowing the inclusion of part of these solutions in the population used by the GA.

In the first section we present the BDRP, the compact model as well as the decomposition model whose linear relaxation will be solved by CG. The second section presents the solution strategies focusing in the presentation of the SearchCol framework (Alvelos et al., 2010); (Alvelos et al., 2013) which allows the search on the column by diverse heuristic methods in addition to the GA. The third section presents details about the implementation of the decomposition model in the framework and explains the GA operation. Some results showing the behaviour of the GA as search method are presented in section four. The last section offers some concluding remarks.

2 THE BUS DRIVER ROSTERING PROBLEM

The Bus Driver Rostering Problem (BDRP) consists of defining the work-schedule for all drivers for a rostering period, fulfilling all tasks on each day and respecting the labour rules.

In this section we present the integer programming model adopted for the BDRP, the compact model, and the corresponding decomposition model resulting from the application of the Dantzig-Wolfe decomposition (Dantzig, 1963); (Dantzig and Wolfe, 1960). This decomposition model meets the requirements of SearchCol framework which combines column generation and metaheuristic (MH) search, as described in (Alvelos et al., 2010); (Alvelos et al., 2013).

2.1 Compact Model

The compact model for the BDRP was derived from the one presented in (Moz et al., 2009). In our case, we only consider one objective function to minimize the total cost of the roster, resulting in the removal of the constraints related with the minimization of the number of drivers used without complete schedules.

We consider a rostering horizon of four weeks (28 days). The parameters and the variables used in the model are:

V – The set of drivers available to perform tasks;

ρ^v – Cost paid to driver $v \epsilon V$ for each time unit of extra work. This cost allows the distinction of different salary categories of workers, $v \epsilon V$;

C – Fixed cost paid for using a driver (equal for all drivers). The cost is not applied if the driver has no tasks assigned during the rostering period (his schedule is filled up with consecutive days off);

g – Maximum number of consecutive days without a day-off;

T_h^w – Set of tasks on day h that must be assign to a driver (this set does not include the "special" task that represents a day-off), h=-g+1,…,0,1,…,28;

T_h – Set of tasks to be assigned on day h (includes the "special" task that represents the driver day-off, which is the last one on each subset corresponding to a day), h=-g+1,…,0,1,…,28;

T_{ih}^v – Set of tasks which can be assigned to driver ($v \epsilon V$) on day h if he does task i on the previous day (h-1). Due to minimum rest periods, depending on the start-time and end-time of the tasks, they are considered "early tasks" and "late tasks", and an early task cannot succeed immediately a late task, $v \epsilon V$, $i \epsilon T_{h-1}$, h=1,…,28;

t_{ih} – Duration (in time units) of task i on day h, $i \epsilon T_h^w$, h=1,…,28;

$\bar{t}$ – Contractual daily work time (limit over wich the work is considered overtime);

t'_{ih} – Overtime time units of task i on day h, results from max $\{0, t_{ih} - \bar{t}\}$, $i \epsilon T_h^w$, h=1,…,28;

b_1 – Maximum total assigned work time (in time units) in each week of the rostering period;

b_2 – Maximum total assigned work time (in time units) in all the rostering period;

d_s – Minimum number of Sundays with day-off assigned to each driver during all the rostering period;

d_w – Minimum number of days-off assigned to each driver in each week of the rostering period;

q – Number of work days where work tasks should be assigned (tasks from T_h^w) to get a complete schedule to the driver. The remaining days of the rostering period are filled with the mandatory days-off;

$e_{i0}{}^v$ – Assumes value 1 if driver v was assigned to task i on the last day of the previous rostering period, otherwise it has value 0, $v \epsilon V$, $i \epsilon T_h^w$;

$e_0{}^v$– Number of consecutive work days (without any day-off) the driver v did after the last day-off in the previous rostering period, $v \epsilon V$;

ϑ – Index of the "special" task which represents the day-off (always the last task in the sets where the task appears);

y_{ih}^v – Binary decision variable representing if the task i from day h is assigned to driver v, assuming the value 1 if true, 0 otherwise, $v \epsilon V$, $i \epsilon T_h^w$, h=1,…,28;

η^v – Binary decision variable representing the use of the driver v in the rostering. The variable assumes the value 1 if at least one work task is assigned to driver v, 0 otherwise, $v \epsilon V$.

Based on these parameters and decision variables the compact integer programing model is:

$$Min \sum_{v \epsilon V} \sum_{h=1}^{28} \sum_{i \in T_h^w} \rho^v t'_{ih} y_{ih}^v + C\eta^v \quad (1)$$

Subject to:

$$\sum_{v \in V} y_{ih}^v = 1 \ , i \in T_h^w, h = 1, \dots, 28, \quad (2)$$

$$\sum_{i \in T_h} y_{ih}^v = 1 \ , \ v \in V, h = 1, \dots, 28, \quad (3)$$

$$y_{l,h-1}^v + \sum_{j \in T_h \setminus T_{ih}^v} y_{jh}^v \leq 1 \ , \ v \in V, i \in T_{h-1}, \ h = 2, \dots, 28, \quad (4)$$

$$e_{i0}^v + \sum_{j \in T_{1 \setminus T_{i1}^v}} y_{j1}^v \leq 1 \ , \ v \in V, i \in T_0, \quad (5)$$

$$\sum_{l=0}^{g} \sum_{i \in T_{h+l}^w} y_{i,h+l}^v \leq g \ , \ v \in V, h = 1, \dots, 28 - g, \quad (6)$$

$$\sum_{l=1}^{-e_0{}^v+g+1} \sum_{i \in T_l^w} y_{il}^v \leq g - e_0{}^v \ , \ v \in V, \quad (7)$$

$$\sum_{h=7(l-1)+1}^{7l} y_{\vartheta h}^v \geq d_w \ , \ v \in V, l = 1, \dots, 4, \quad (8)$$

$$\sum_{l=1}^{4} y_{\vartheta,7l}^v \geq d_s \ , \ v \in V, \quad (9)$$

$$\sum_{h=7(l-1)+1}^{7l} \sum_{i \in T_h^w} t_{ih} \, y_{ih}^v \leq b_1 \ , \ v \in V, l = 1, \dots, 4, \quad (10)$$

$$\sum_{h=1}^{28} \sum_{i \in T_h^w} t_{ih} \, y_{ih}^v \leq b_2 \ , \ v \in V, \quad (11)$$

$$\sum_{h=1}^{28} \sum_{i \in T_h^w} y_{ih}^v - \mathrm{q}\eta^v \leq 0 \ , \ v \in V, \quad (12)$$

$$y_{ih}^v \in \{0,1\} \ , \ v \in V, i \in T_h, h = 1, \dots, 28. \quad (13)$$

$$\eta^v \in \{0,1\} \ , v \in V \quad (14)$$

The objective function (1) minimizes the sum of the overtime costs drivers and the fixed costs by using drivers. Constraints (2) assure that each task from each day is assigned to one, and only one, driver from the set of drivers. Constraints (3) assure that each driver has one task assigned in each day of the rostering period (which can be the "special" task representing the day-off). Constraints (4) and (5) prevent the assignment of incompatible sequences of tasks in the schedule of a driver (avoid the assignment of an early task after a late task). Constraints (4) consider the first day of the rostering period, where data from the last day from previous rostering period are needed. Constraints (5) consider the following days. Constraints (6) and (7) prevent the assignment of work tasks in more than g consecutive days (maximum number of work days without a day-off). Constraints (7) consider the initial days of the rostering period where information from the previous period is considered in the constraints. Constraints (8) force the assignment of at least d_w days-off ("special" task with index ϑ) in each week of the rostering period. Constraints (9) force the assignment of at least d_s days-off on Sundays during the rostering period. Constraints (10) prevent, in each week, the assignment of tasks with a total duration exceeding $b1$, the week limit defined by labour rules. Constraints (11) prevent the assignment of a complete schedule with a total duration exceeding $b2$, the total work time limit defined contractually for the rostering period. Constraints (12) force the binary variable η^v to be set to 1 if at least one work task is assigned to driver v in the rostering period, the variable is set to 0 if the driver schedule is filled with days-off (meaning that driver v is not used). Constraints (13) and (14) define the variables y_{ih}^v and η^v, respectively, as binary variables.

2.2 Decomposition Model

Considering the previously presented compact model, it is easy to observe that almost all the constraints make use of variables for a single driver and only constraints (2) aggregate variables corresponding to all drivers. Neglecting constraints (2), we have one independent problem for each driver. This fact justifies the decomposing of the compact model "by driver".

We obtain the following model for the subproblem of a generic driver v. Note that the objective function takes into account the dual variables of the constraints of the master problem – to be introduced below.

Subproblem formulation for driver *v* (SP^v):

$$Min \sum_{h=1}^{28} \sum_{i \in T_h^w} (\rho t'_{ih} y_{ih} - \pi_{ih}\, y_{ih}) + C\eta - \pi_v \quad (15)$$

Subject to:

$$\sum_{i \in T_h} y_{ih} = 1\ , h = 1, \dots, 28, \quad (16)$$

$$y_{i,h-1} + \sum_{j \in T_h \setminus T_{ih}} y_{jh} \leq 1\ , i \in T_{h-1}, h = 2, \dots, 28, \quad (17)$$

$$e_{i0} + \sum_{j \in T_1 \setminus T_{i1}} y_{j1} \leq 1\ , i \in T_0, \quad (18)$$

$$\sum_{l=0}^{g} \sum_{i \in T_{h+l}^w} y_{i,h+l} \leq g\ , h = 1, \dots, 28 - g, \quad (19)$$

$$\sum_{l=0}^{g-e_0+1} \sum_{i \in T_l^w} y_{il} \leq g - e_0, \quad (20)$$

$$\sum_{h=7(l-1)+1}^{7l} y_{\vartheta h} \geq d_w\ , l = 1, \dots, 4, \quad (21)$$

$$\sum_{l=1}^{4} y_{\vartheta, 7l} \geq d_s, \quad (22)$$

$$\sum_{h=7(l-1)+1}^{7l} \sum_{i \in T_h^w} t_{ih}\, y_{ih} \leq b_1\ ,\ l = 1, \dots, 4, \quad (23)$$

$$\sum_{h=1}^{28} \sum_{i \in T_h^w} t_{ih}\, y_{ih} \leq b_2, \quad (24)$$

$$\sum_{h=1}^{28} \sum_{i \in T_h^w} y_{ih} - q\eta \leq 0, \quad (25)$$

$$y_{ih} \in \{0,1\}\ , i \in T_h, h = 1, \dots, 28, \quad (26)$$

$$\eta \in \{0,1\}; \quad (27)$$

Where:

y_{ih} - Binary variable representing if task *i* from day *h* is assigned to driver associated with this subproblem, assuming the value 1 if true, 0 otherwise, $i \epsilon T_h^w$, *h*=1,…,28;

η - Binary variable representing the use of the driver associated with this subproblem. The variable assumes the value 1 if at least one work task is assigned to driver, 0 otherwise (schedule full of days-off);

ρ – Cost paid to driver *v* for each time unit of extra work;

π_{ih} – Dual variable associated to the linking constraint of task *i* of day *h* (constraints (29) from the RMP);

π_v – Dual variable associated to the convexity constraint (constraint (30) from the RMP) inserted in the restricted master problem associated with this subproblem (driver *v*);

T_{ih} – Subset of T_{ih}^v (defined in the compact model) related to the subproblem driver *v*;

e_0 – Number of consecutive work days (without day-off) the subproblem driver did after the last day-off in the previous rostering period;

e_{i0} – Assumes value 1 if subproblem driver was assigned to task *i* on the last day of the previous rostering period, otherwise it has value 0, $i \epsilon T_h^w$;

All other parameters remain the same as in the compact model.

Considering the subproblem model (SPv), the compact model can be rewritten considering the convex combination of the extreme points resulting from the subproblems' solutions, leading to a master problem (MP) that considers all possible columns. Without loss of generality, we can assume that the set of columns to be considered is known, thus resulting in the following restricted master problem (RMP).

RMP formulation:

$$Min \sum_{v \in V} \sum_{j \in J^v} p_j^v \lambda_j^v + \sum_{i\, \epsilon\, T_h^w} \sum_{h=1}^{28} M(\delta_{ih}^+ + \delta_{ih}^-) + \sum_{v\, \epsilon\, V} M(\sigma_v^+ + \sigma_v^-) \quad (28)$$

Subject to:

$$\sum_{v \in V} \sum_{j \epsilon J^v} a_{ih}^{jv} \lambda_j^v + \delta_{ih}^+ - \delta_{ih}^- = 1\ , i \in T_h^w,\ h = 1, \dots, 28, \quad (29)$$

$$\sum_j \lambda_j^v + \sigma_v^+ - \sigma_v^- = 1\ , v \in V, \quad (30)$$

$$\lambda_j^v \epsilon \{0,1\},\ j \in J^v, v \in V, \quad (31)$$

$$0 \leq \delta_{ih}^+ \leq 1, i \in T_h^w, h = 1, \dots, 28, \quad (32)$$

$$0 \leq \delta_{ih}^- \leq 1, i \in T_h^w, h = 1, \dots, 28, \quad (33)$$

$$0 \leq \sigma_v^+ \leq 1, v \in V, \quad (34)$$

$$0 \leq \sigma_v^- \leq 1, v \in V \quad (35)$$

Where:

λ_j^v– Variable associated to the schedule *j* of driver *v*;

$\delta_{ih}^+, \delta_{ih}^-$ – Artificial variables associated to the linking constraint (for task *i* on day *h*) to make the problem possible until the first convex combination of extreme points is achieved by the column generation;

σ_v^+, σ_v^- – Artificial variables associated to the convexity constraint (for subproblem/driver *v*) to make the problem possible until the first convex combination of extreme points is achieved by the column generation;

J^v – Set of schedules for driver *v* generated by column generation;

p_j^v – Cost of the schedule *j* obtained from the subproblem of driver *v*;

a_{ih}^{jv} – Assumes value 1 if task *i* of day *h* is assigned in the schedule *j* of driver *v*;

M – Very big value used to penalize the use of artificial variables in the solution of the restricted master problem.

The linking constraints (29) and convexity constraints (30) have dual variables π_{ih} and π_v, respectively, which are present in the objective function of the subproblem.

The linking constraints (29), as was the case in the corresponding constraints from the compact model (2), assure that all the tasks are assigned.

Since the variables on the RMP are linear, the solution of the RMP can share a task among multiple drivers, but the sum of the columns including that task should be 1.

3 SOLUTION STRATEGIES

Given the models presented in the previous section, we now present methods used to obtain the optimal integer solution, integer solutions (approximate) and linear solutions with better lower bounds than the direct linear solution.

3.1 Optimal Integer Solution

The most direct way to obtain the optimal integer solution for the BDRP is by solving the compact model using branch and bound through efficient software implementation like CPLEX (ILOG, 2009). As stated in (Moz et al., 2009), the bus rostering problem is classified as NP hard and computational tests previously completed by the authors shown that it takes a considerable amount of time to obtain optimal solutions. In part of the instances the time limit (four hours) was achieved without proving the optimality of the solution found. We also tested to directly solve some instances and the behaviour was the same, the solver faced difficulties to prove optimality before the test time limit defined.

3.2 Hybrid Metaheuristic

The core of our research is to find good solutions through the hybridization of column generation and metaheuristics, as proposed in (Alvelos et al., 2010) using the SearchCol framework. The SearchCol framework proposes the use of metaheuristics to find good combinations of the subproblem solutions (schedules) generated during the column generation method considering the linear optimal solution of the restricted master problem as an indicator of the quality of each column in the search space.

This framework provides the possibility of running multiple times the CG with new constraints to force the generation of new schedules with unassigned tasks.

The framework concept, details about the implementation, search spaces and solutions representation after the column generation are presented in (Alvelos et al., 2013). The overview of the search for solutions within the framework follows the high-level algorithm in Figure 1:

```
1: Column generation
2: Search
3: repeat {
4: Set column generation perturbation
5: Optimize perturbed column generation
6: Search
7: } until Stopping criterion fulfilled
```

Figure 1: SearchCol general algorithm.

In step 1 of the algorithm (Figure 1), it is possible to obtain lower bounds to the linear solution of the problem better than those obtained from the compact model, which is a well-known characteristic of the Dantzig-Wolfe decomposition reformulated models (Lübbecke and Desrosiers, 2005). These lower bounds are important in the enumeration methods (like branch and bound) used to obtain optimal integer solutions since they are the reference to evaluate the need to explore branches of the search tree (reducing the search) and also the reference to calculate the gap between the integer and the linear solutions.

In step 2 of the algorithm, it is possible to obtain integer solutions by selecting one solution from each subproblem, satisfying the linking constraints. A possible strategy is to round up the linear values of the optimal RMP solution, however that solution will assign the same task to more than one driver (in our problem) every time the linking constraint has more than one variable with nonzero value.

After the column generation, the variables associated with the columns may be set of type integer and the resulting MIP can be optimized by branch and bound in order to obtain the best available integer solution on the new search space (the set of columns generated by CG). If this new search space contains a considerable number of admissible solutions for each subproblem, this process is very time consuming.

The idea behind SearchCol is the use of heuristic search methods to select the columns (solutions of the subproblems) which should integrate the integer solution. Currently, the framework has as methods (metaheuristics), among others, multi-start local search (MSLS) and variable neighborhood search (VNS) to improve a single solution. We now integrate genetic algorithms as the search method, being the first population based metaheuristic implemented in the SearchCol framework.

Our contribution is the integration of a new problem in the SearchCol framework and the development of a new “Search” phase (steps 2 and 6) which can be used by most of the other problems solved using this framework. The next section

presents the details on the integration of the new problem and on the new search method.

4 IMPLEMENTATION

This section presents details about how the decomposition model presented in section 0 was implemented in the SearchCol framework and how the genetic algorithms were used to explore the search space resulting from the column generation.

4.1 Implementing the Decomposition Model

To integrate the new problem in the Searchcol framework we defined a new class (BDRostering) responsible to store all the parameters used in the decomposition model and offering methods that allow for reading the instances from files and mapping the data to the correct variables in the class. The BDRostering class is used as base class to another class (DecBDRostering) which also inherits from class Decomposition (part of SearchCol framework) where the methods needed to implement are defined and given access to internal implementation of the RMP and subproblems implementation.

The class DecBDRostering needs to implement a set of methods used by the framework to:

- define the decomposition, create all the subproblems, define the linking constraints on the RMP and the matrix with their coeficients used to define columns, etc;
- update the subproblems objective function costs in each CG iteration;
- construct the new column added to the RMP from the subproblem solution;
- optimize the subproblems.

The methods implemented on the DecBDRostering are those where knowledge about the problem/decomposition is needed, all the algorithm steps were already implemented within the base class (and other classes) of the framework.

4.2 Search Columns with Genetic Algorithms

The conclusion of the column generation process gives us two main sources of useful information to build valid solutions for the rostering problem:

- The first one is the set of columns added to the RMP. We know that each one corresponds to a valid schedule for a given driver;
- The second one is the optimal solution of the RMP. Even being a fractional solution, the selection of columns used in that solution can be set as the primary search space to explore in the search of good integer solutions. The value of the variable associated to each column can be seen as an indicator of the quality of that column, if the variable is close to one it means the solution associated to the corresponding driver should be tried in the rostering. The optimal solution value is also the lower bound to consider in the search of valid integer solutions.

Given the set of valid schedules to each driver (column generated by corresponding subproblem), a roster consists in the selection of a schedule to all drivers of the bus company. The only remaining constraint is the one that forces the accomplishment of all the tasks, which is not achieved by randomly selecting a schedule to each driver. The challenge is to find combinations of schedules that assign all the tasks while minimizing the total rostering cost.

4.2.1 Genetic Algorithms Integration

The use of Genetic Algorithms (GA) (Holland, 1992); (Mitchell, 1996); (Reeves, 1997) as the metaheuristic used in the "Search" phase of the SearchCol global algorithm arises naturally since the resemblance of the solution representation used and the chromosomes present in the GA.

C1	C7	C23	…	C10
Driver 1	Driver 2	Driver 3	…	Driver n

Figure 2: Roster solution representation as a chromosome.

As presented in Figure 2, a roster can be considered a chromosome with *n* genes, where each allele is the identification of a driver schedule selected from the set of valid schedules (columns added from the corresponding supbproblem solution) generated by the column generation iterations. In order to create a new chromosome representing a roster each gene must be filled with the identification of a solution generated by the driver subproblem corresponding to that gene locus. In the iterations of the column generation, every time a subproblem solution is considered attractive to be added as a new column, the original solution is saved with data from which subproblem produced the solution and the column added (column order number) to the RMP. Keeping this information updated, at the end of column generation, we may consider we have virtual pots (as illustrated in

Figure 3) for each driver, with valid solution inside, from where we can pick a solution with the guarantee that it is a valid solution for the driver.

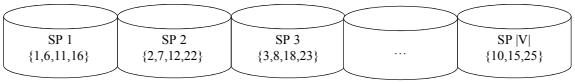

Figure 3: Valid schedules pots for each driver.

4.2.2 GA Initial Population Generation

The GA rely on the existence of a population (Pop) of individuals where the GA operators are applied in order to evolve through generations.

We need to create populations assuring a good dispersion of the individuals on the search space. To reach that target, we cannot focus only in the solutions that integrate the optimal solution of the RMP at the end of column generation (assumed of good quality) but also explore the others solutions.

The SearchCol framework already provides a set of distinct methods to create global solutions (individuals) where those different scenarios are considered. All available methods, detailing the solutions considered for selection in each one, are detailed in (Alvelos et al., 2013).

Since those methods are already available and we need diversity on the individuals integrating the population, we have created a set of parameters to define the percentage of individuals generated by each method in the initial population of the GA, easily allowing changes/trials at runtime.

4.2.3 GA Operators

The evolution of populations is achieved by selecting the best individuals to integrate the mating pool (MP) and applying over those individuals the variation operators commonly used in the GA, the crossover operator and the mutation operator.

The method we are using as selection operator is the tournament. We randomly select pairs of individuals and compare them using a bi-level evaluation function (presented below). The one with lower infeasibility is selected. If both have the same infeasibility, the evaluation function that calculates the value of the original objective function is used to select the best one.

The crossover operator is used to generate offspring that share characteristics from both parents. Usually the operator creates the offspring by selecting subsequences of genes alternatively from both parents. Our current implementation has two implementations of commonly used crossovers: the one point crossover and the two point crossover.

Considering the following parents for an instance with 9 drivers:

Parent 1:	1	7	23	4	5	9	20	38	10
Parent 2:	11	22	8	26	29	17	13	31	25

Applying the one point crossover, considering the point between the fourth and the fifth genes, results in:

Offspring 1:	1	7	23	4	29	17	13	31	25
Offspring 2:	11	22	8	26	5	9	20	38	10

Applying the two point crossover, considering the first point between the third and the fourth genes and the second point between the sixth and the seventh genes, results in:

Offspring 1:	1	7	23	26	29	17	20	38	10
Offspring 2:	11	22	8	4	5	9	13	31	25

The mutation operator is used to change an individual by randomly modifying one or more genes. This operator produces a small dispersion on the search space avoiding the stagnation in local optimums. Currently, when mutation is applied, each locus is drawn to be changed and, if selected, the selected gene is replaced by another allele, chosen from the pot of solutions of the subproblem associated to that position/driver.

4.2.4 Evaluation of Individual

One of the important pieces of the GA is the function used to evaluate an individual in such a way that the result can be used to compare two or more individuals and identify which is better and which is worst. The definition of good evaluation functions is very important since the result is often used to decide the inclusion or not in the mating pool, allowing the persistence of an individual (or descendants) along generations.

In our problem we consider a bi-level function to evaluate an individual by its feasibility and infeasibility values. The feasibility is the value obtained by applying the objective function (1) from the original formulation of the BDR problem. The infeasibility is measured by the number of tasks not assigned to the group of all drivers. A task is not assigned if the linking constraint (29) is not satisfied by having the left hand side smaller than the right hand side, otherwise it may be considered feasible, even if a task is assigned to more than one driver, since the correction is done easily by removing the duplicated task to one of the drivers.

When comparing individuals, the infeasibility is the first level evaluation value. The feasibility value is only used to compare individuals with the same value of infeasibility, since it is normal that individuals with higher infeasibility (more unassigned tasks) return better feasibility values (cost), because the tasks not assigned may have costs not considered.

4.2.5 GA Runs

A GA run consists in constructing an initial population and letting it evolve through generations until achieving a stopping criterion. We are considering as stopping criterion the reaching of a number (defined by parameter) of consecutive generations without improving the evaluation value.

Figure 4 describes the most important steps of a run of the algorithm.

```
Generate Initial Population
repeat{
 Build Mating Pool (MP):
 for (population size){
  Select a pair of individuals
  Apply tournament to select the best
  Add best to MP
 }
 Apply Crossover:
 for (population size/2){
  Select pair of individuals from MP
  If selected to crossover{
   Apply crossover operator to pair
   Add offspring to next population
  }else
   Add pair of individuals to next
   population
 }
 Apply Mutation:
 for each individual of population
  If selected to mutation
   Apply mutation operator
 Update Best:
 for each individual of population{
  Evaluate Individual
  Update best found
 }
}while(iterations without improvement
< limit)
```

Figure 4: GA pseudocode.

The entire algorithm is parameterized. The generation of the initial population uses parameters to decide the percentage of individuals created by each of the methods already available in the framework. We also defined parameters to the population size, number of generations without improvement (for stopping criterion), probability of crossover and probability of mutation.

5 RESULTS

To evaluate the effectiveness of the proposed search method over the search space of schedules (resulting from solving the decomposition model with column generation) some tests were made over a subset of the instances used in (Moz et al., 2009). The instances tested were the designated as P80 and the same parameters were used, except for the number of drivers: we use a pool of 36 drivers. In the instances, the group of drivers is divided in four categories of overtime cost, starting from a cost factor of one and doubling for the next group, resulting that the last group overtime is 8 times more expensive.

The results show the effectiveness of the column generation over the decomposition model to generate a good search space of schedules where a complete roster can be obtained and also that the GA can be a faster option to search for the best roster in that search space.

All the tests ran on a Dell Optiplex 380 with an Intel Core 2 Duo CPU E7500, 2,93GHz, 4 Gb of RAM, operating system Windows Vista 32 bits and IBM ILOG 12.3 installed.

In this stage we only tested the first search phase, which means we only used the steps 1 and 2 from Figure 1.

To test the search space obtained by the column generation, the step 2 used was the direct optimization of the resulting RMP, setting the variables as binary, designated as MipSearch. This procedure searches for the optimal solution within the set of schedules available in the search space. In the tests, the total time was limited to 7200s (2 hours) and the column generation (step1) was limited to 1800s (1/2 hour).

Table 1 presents the obtained results. The Feasibility column presents the total cost of the roster found and the Infeasibility column displays the number of tasks not assigned.

The first observation from the results presented in Table 1 is that CPLEX spent all available time applying branch-and-bound to search for the optimal solution, indicating that it is time consuming to find the best available solution. On 3 of the 11 instances, no feasible solution was found (complete roster with all tasks assigned).

The previous results also show that this search method is not a good option if the entire algorithm

Table 1: MipSearch Results.

Instance	Infeasibility	Feasibility	Time Search	Time CG
P80_0	93	-	5400	1800
P80_1	0	6477	5400	1800
P80_2	0	4628	6815,2	384,7
P80_3	0	8762	5400	1800
P80_4	0	6809	5400	1800
P80_5	0	6648	5400	1800
P80_6	0	7182	5400	1800
P80_7	0	6819	5400	1800
P80_8	12	-	5400	1800
P80_9	0	5599	5563,2	1636,4
P80_10	3	-	5400	1800

from Figure 1 is used (with multiple searches in the cycle – step 6), since it consumes too much time.

To test the GA as search method, two sets of tests were run, where the difference between them is in the search space generation. In the first set of tests (GA1), in each iteration of the column generation a schedule from each driver is added to the RMP (if attractive) and in the second one (GA2), only one schedule is added from a single driver, changing sequentially the driver through the iterations. Both options were tried since, though generating columns for all subproblems in each iteration results in a fast growth of the number of variables in the RMP, making it difficult to optimize, it also may result in similar solutions to all drivers which may be useful to the GA. The Mip Search was applied over the search space resulting from the column generation with one subproblem solved in each iteration.

The following parameters were used in the GA:

Population Size = 200;

Crossover | Mutation Probability = 80% | 15%;

Stopping Criterion = 5000 generations without improvement.

Each initial population is composed by:

70% of individuals selected randomly from the pots of schedules available (40% with uniform distribution for each schedule and 30% with biased distribution for each schedule according to the optimal solution of the RMP);

10% of individuals composed by rounding the linear solution of the RMP;

10% of individuals composed by the first solution generated by column generation;

10% of individuals composed by the last solution generated by column generation.

Table 2 shows the results obtained by GA1 and GA2. In each instance test, after the column generation (limited to 1800s) 10 runs of the GA were performed. Columns (1) display the number of unassigned tasks for the best solution found by each GA, while columns (2) present the average of this number for the 10 runs. Columns (3) display the corresponding cost value, while columns (4) present the average cost in the 10 runs. The last column of each group shows the average search time spent by the GA to find the solutions. When value Best (1) is zero, the algorithm was able to find a feasible solution for the global problem. The GA were clearly more effective in the second scenario obtaining valid rosters to 7 of the 11 instances, only one less than the MipSearch. Curiously, in the harder instances (were the MipSearch did not found a solution, instances 0, 8 and 10) the GA1 obtained better results, even better than MipSearch for the P80_0 (considering only the Infeasibility value).

Since the search space of the GA1 is larger than GA2, it is natural to have higher average search time, but in both scenarios, the search time is much faster than the MipSearch, claiming to be a good option to be used as the "Search" step in the entire algorithm from Figure 1 to improve the first solution obtained getting new schedules after applying perturbations on the RMP. The GA2 found valid rosters in almost all instances. The three instances where the MipSearch was unable to find a solution have a high number of tasks to assign, making it difficult to find the exact combination of schedules where all tasks are assigned.

The results obtained reveal promising to the entire SearchCol algorithm, since they suggest that GA are effective and fast as search method. The resulting Feasibility values obtained by the best GA configuration are on average 17% higher than the best found by MipSearch, however the time spent by the GA in the search is less than 5% (average 1,8%) of the time spent by MipSearch.

6 CONCLUSIONS

This paper presents a new formulation to the BDRP and a new hybrid metaheuristic to obtain valid rosters from the proposed model without large time consumption. The proposed decomposition model splits the problem of tasks assignment into multiple subproblems were the assignment concerns only one driver, and a restricted master problem where the variables are associated to work-schedules created by the subproblems and where the assignment of all tasks is assured by combining multiple work-schedules from all the drivers.

The column generation is used to create new work-schedules by solving one or more subproblems in each iteration until an optimal linear solution is obtained.

Table 2: GA results.

	GA1					GA2				
	Infeasibility		Feasibility		Time	Infeasibility		Feasibility		Time
Instance	Best(1)	Average(2)	Best(3)	Average(4)	(Average)	Best(1)	Average(2)	Best(3)	Average(4)	(Average)
P80_0	29	83,6	7748	7316,4	223,9	123	134,6	6251	6376,7	117,2
P80_1	0	13,2	7970	7096,2	91,0	0	7,3	8324	8325,1	78,7
P80_2	0	0	5854	6743,9	52,6	0	0	5625	5697,2	31,9
P80_3	9	13,3	8919	9494,1	87,3	0	5,3	9329	10865,4	64,3
P80_4	5	11,1	7107	6798,1	77,4	0	2,2	7986	8583,2	64,7
P80_5	3	7,6	7820	7594,5	84,0	0	4,2	7600	8582,7	61,4
P80_6	2	3,5	8216	8304,9	66,2	0	0	8481	9156,6	38,1
P80_7	13	18	6745	6840,2	96,0	2	5,2	9343	8757,9	45,9
P80_8	31	50,6	7711	6769,4	90,0	32	53,2	8605	6891,7	67,4
P80_9	0	2,9	7017	7344	74,6	0	0,2	6661	7287	56,9
P80_10	12	53,3	6188	6026,4	132,5	43	58	6219	5923,8	99,3

The set of subproblem solutions (driver work-schedules), added as new columns in the CG, defines the search space where the genetic algorithms are used to find the best combination of solutions that at first assigns the major number of tasks and after reduces the total cost.

Results of applying the MipSearch (solving the RMP as a Mip) over the work-schedules generated by column generation are presented, with the best values obtained within a given time limit. The computational tests show that the proposed metaheuristic is skilled to obtain valid rosters. Two configurations were run using a genetic algorithm as "search" method, showing that GA is faster than the MipSearch.

Future work will include testing different parameters' configurations for the GAs and comparing our approach with the existing ones.

ACKNOWLEDGEMENTS

This work was partially funded by projects PTDC/EIA-EIA/100645/2008 ("SearchCol–Meta-heuristic Search by column generation" / FEDER through "COMPETE – Programa Operacional Factores de Competitividade" and FCT – Foundation for Science and Technology) and PEst-OE/MAT/UI0152 (FCT – Foundation for Science and Technology).

REFERENCES

Alvelos, F., de Sousa, A. and Santos, D. (2010). SearchCol: Metaheuristic Search by Column Generation. In M. Blesa, C. Blum, G. Raidl, A. Roli & M. Sampels (Eds.), *Hybrid Metaheuristics* (Vol. 6373, pp. 190-205): Springer Berlin / Heidelberg.

Alvelos, F., Sousa, A. and Santos, D. (2013). Combining column generation and metaheuristics. In E.-G. Talbi (Ed.), *Hybrid metaheuristics* (pp. 285-334): Springer.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P. and Vance, P. H. (1998). Branch-and-Price: Column Generation For Solving Huge Integer Programs. *Operations Research, 46*(3), 316-329.

Burke, E. K., De Causmaecker, P., Berghe, G. and Van Landeghem, H. (2004). The State of the Art of Nurse Rostering. *Journal of Scheduling, 7*(6), 441-499.

Burke, E. K., Kendall, G. and Soubeiga, E. (2003). A Tabu-Search Hyperheuristic for Timetabling and Rostering. *Journal of Heuristics, 9*(6), 451-470.

Dantzig, G. B. (1963). *Linear Programming and Extensions*. Princeton, New Jersey: Princeton University Press.

Dantzig, G. B. and Wolfe, P. (1960). Decomposition Principle for Linear Programs. *Operations Research, 8*(1), 101-111.

De Leone, R., Festa, P. and Marchitto, E. (2010). A Bus Driver Scheduling Problem: a new mathematical model and a GRASP approximate solution. *Journal of Heuristics*, 1-26.

Desaulniers, G., Desrosiers, J. and Solomon, M. M. (2005). *Column Generation*. New York: Springer.

Desrosiers, J., Soumis, F. and Desrochers, M. (1984). Routing with time windows by column generation. *Networks, 14*, 545-565.

Dorne, R. (2008). Personnel Shift Scheduling and Rostering. In C. Voudouris, D. Lesaint & G. Owusu (Eds.), *Service Chain Management* (pp. 125-138): Springer Berlin Heidelberg.

Ernst, A. T., Jiang, H., Krishnamoorthy, M. and Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research, 153*(1), 3-27.

Holland, J. H. (1992). *Adaptation in natural and artificial systems*: MIT Press.

ILOG, I. (2009). *User's Manual for CPLEX*.

Kohl, N. and Karisch, S. E. (2004). Airline Crew Rostering: Problem Types, Modeling, and Optimization. *Annals of Operations Research, 127*(1), 223-257.

Lübbecke, M. E. and Desrosiers, J. (2005). Selected Topics in Column Generation. *Oper. Res., 53*(6), 1007-1023.

Lučić, P. and Teodorović, D. (2007). Metaheuristics approach to the aircrew rostering problem. *Annals of Operations Research, 155*(1), 311-338.

Mitchell, M. (1996). *An introduction to genetic algorithms*: MIT Press.

Moz, M. and Pato, M. (2007). A genetic algorithm approach to a nurse rerostering problem. *Computers & Operations Research, 34*(3), 667-691.

Moz, M., Respício, A. and Pato, M. (2009). Bi-objective evolutionary heuristics for bus driver rostering. *Public Transport, 1*(3), 189-210.

Reeves, C. R. (1997). Genetic Algorithms for the Operations Researcher. *INFORMS Journal on Computing, 9*(3), 231-250.

Rodrigues, M. M., de Souza, C. C. and Moura, A. V. (2006). Vehicle and crew scheduling for urban bus lines. *European Journal of Operational Research, 170*(3), 844-862.

Ruibin, B., Burke, E. K., Kendall, G., Jingpeng, L. and McCollum, B. (2010). A Hybrid Evolutionary Approach to the Nurse Rostering Problem. *Evolutionary Computation, IEEE Transactions on, 14*(4), 580-590.

Wren, A. (1996). Scheduling, timetabling and rostering — A special relationship? In E. Burke & P. Ross (Eds.), *Practice and Theory of Automated Timetabling* (Vol. 1153, pp. 46-75): Springer Berlin / Heidelberg.