Driver Scheduling Generation Using a Population Training Algorithm

Geraldo Regis Mauri
Luiz Antonio Nogueira Lorena
National Institute for Space Research
Computing and Applied Mathematics Laboratory
São José dos Campos SP, Brazil
{*mauri,lorena*}*@lac.inpe.br*

Abstract

This paper describes an application of the Population Training Algorithm (PTA) to generate driver schedules for public bus transportation. The PTA employs heuristics in fitness definition, guiding the population to settle down in search areas where the individuals cannot be improved by such heuristics. The drive schedules are represented by columns in a large scale set partitioning problem, which are generated when solving the linear programming relaxation. The computational results are compared against a Simulated Annealing metaheuristic, using randomly generated instances based on real problems.

1. Introduction

Since the 60's research has been done to the development of models dedicated to crew scheduling generation satisfying imposed labor restrictions and providing the smallest possible cost. This is a permanently studied problem, since the transportation system realities are in continuous transformation and demand efficient management of the available resources. This increasing interest is confirmed by specialized conferences in the last decades, as for example the International Conference on Computer-Aided Scheduling of Public Transport [4],[5], [14], [12] and [11].

In spite of this, little has been done in Brazil, where the number of restrictions becomes the problem more complex than in most of the developed countries. Besides, the access democratization of the public transportation system in Brazil depends on the warranty of tolerable tariffs for the users, which depends so much of efficiency increase and reduction of operational costs, as well as of subsidies concession in specific situations under social control [1].

It is well known that in urban public transportation companies the operational labor represents one of the largest costs [3]. A small reduction can represent a considerable gain in the total cost, reducing significantly tariffs for the final user, justifying any effort in the sense of minimizing the costs with labor.

Several approaches appeared before in literature to solve the driver scheduling problem. The problem is known to be NP-hard (difficult for large instances). Metaheuristics such as *Genetic Algorithms, Tabu Search, Simulated Annealing*, among others, allow to easily include several types of labor conditions ([11],[9],[8]). Column generation techniques also shown very good results ([6],[13]).

This work presents a new alternative to generate good driver schedules, the evolutionary approach called Population Training Algorithm (PTA), which generates high quality columns to a set partitioning formulation solved by a column generation technique. The paper is organized as follows. Section 2 presents the problem definition and modeling. The PTA approach is detailed in section 3, while section 4 describes the column generation process and the PTA/Linear Programming interaction. The computational results are presented in section 5, and conclusions are summarized in section 6.

2. Problem Description and Modeling

The Driver Scheduling Problem - DSP, also known in the European countries as Crew Scheduling Problem - CSP [15] or Bus Driver Scheduling Problem - BDSP [8], consists in attributing to the drivers and collectors (crews) the job of driving the vehicles, in such a way that the trips of the different lines assisted by the company are executed with the smallest possible cost. This process of attributing tasks to the crews, also called of driver scheduling, is the construction of a group of legal shifts that cover all of the schedule blocks of a vehicle, that is part of a large scale of vehicles reflecting all the operations of an organization [15].

The driver scheduling must consider a group of rules that are specific to an organization. These rules are usually derived from national and local rules, being obligatory or not. Typically, there are restrictions in the total time worked, in the total extension of the shift (duration between the start and the end of the shift) etc.

The modeling approached in this work takes as base the entities proposed by Mauri [9] due to its identification with others found in the literature, its clarity and the results obtained for real problems. This modeling treats the problem through the entities: *Tasks*, *Drivers* (or *Crews*) and *Shifts* (*daily duties*).

The problem is described as the one of formation of a matrix, where drivers appear in columns and tasks in lines (see Figure 1). Each element $a_{ij} \in \{0,1\}$, *i* ∈ *M* = $\{1,...,m\}$ and *j* ∈ *N* = $\{1,...,n\}$, where *m* is the number of tasks (lines), and *n* the number of drivers (columns), and a_{ij} = 1 if the task *i* belongs to *j* driver's shift, and 0 otherwise.

$$A = \begin{bmatrix} a_{11} & a_{12} & \rightarrow & a_{1n} \\ a_{21} & a_{22} & \rightarrow & a_{2n} \\ \downarrow & \downarrow & \rightarrow & \downarrow \\ a_{m1} & a_{m2} & \rightarrow & a_{mn} \end{bmatrix}$$

Figure 1. DSP representation.

This matrix will be used to solve the following set partitioning problem (SPP):

Minimize:

$$\sum_{j=1}^{n} c_j x_j \qquad (1)$$

Subject to:

$$\sum_{j=1}^{n} a_{ij} x_j = 1 \qquad i = 1,...,m \qquad (2)$$

$$x_j \in \{0,1\} \qquad j = 1,...,n \qquad (3)$$

where: c_j is the cost of column *j* and x_j is equal to 1 if column *j* belongs to the solution, and 0 otherwise. This is a classic formulation, constantly used in the literature ([6],[13]).

The costs considered in this work penalize extra times, idle times and overlapping times. They are formed by

$$c_j = ET_j + OT_j + OV_j + IT_j \qquad (4)$$

where:

$$ET_j = max(0, [FT(t_{lst}) - ST(t_{fst})] - MWT)$$

$$OV_j = max(0, [FT(t_{lst}) - ST(t_{fst})] - NWT)$$

$$OT = \sum_{i=1}^{m-1} \max(0, [FT(t_i) - ST(t_{i+1})])$$

$$IT_j = max(0, NWT - [FT(t_{lst}) - ST(t_{fst})]) \quad +$$

$$... \sum_{i=1}^{m-1} \max(0, [ST(t_{i+1}) - FT(t_i)])$$

c_j is the cost of the driver *j*; ET_j is the extra-time in the total time worked for the *j* driver (time after 8 hours + 2 extra hours); OT_j is the overlapping time of driver *j* (time that a driver is, theoretically, accomplishing two tasks at the same time); OV_j is the extra-time in the normal time of work established by law (also known as overtimes) to driver *j* (8 hours); IT_j is the idle time of driver *j* (time the driver is not working during his shift); *MWT* is the maximum working time (8 hours + 2 hours) ; *NWT* is the normal working time (8 hours); $FT(t_i)$: is the finish time of the task *i*; $ST(t_i)$ is the start time of the task *i*; t_{fst} is the first task of the *j* driver's shift; t_{lst} is the last task of the *j* driver's shift.

To the company is essential (obligatory) to avoid extra times and overlapping but the idle and overtimes are tolerable (non-essential). Columns with cost zero are the best ones, but should not give a good shift. The SPP requires that all the costs be minimized while all the tasks are performed.

The number of possible columns (shifts) of matrix *A* can be huge, and the PTA will be used to generate good columns, reducing the number of columns for the final set partitioning model.

3. Population Training Algorithm

The Population Training Algorithm (PTA) is a kind of evolutionary technique first employed in [10], and derived of the Constructive Genetic Algorithm (CGA). The CGA was proposed by Lorena and Furtado [7] and has a number of innovative features compared to traditional genetic algorithms. These include a ranked population of dynamic size composed of schemata and structures. The schemata and structures are directly evaluated in a common basis, using a double fitness process, called fg-fitness.

The schemata are not used in PTA and the fg-fitness will be performed by heuristics. An individual is considered well adapted if it cannot be better regarding the employed training heuristic. The adaptation in the population training is, therefore, used to guide the search to promising areas.

The representation of a shift is given by: S_k = (a_{1k}, a_{2k},..., a_{mk}) or, for example, S_k = (1, 0, 1, 1,..., 0), meaning that the tasks 1, 3, 4,..., are in the *k* driver's shift.

The two functions used in evolutionary training are defined by *g*(S_k) = cost of shift *k* (used during the columns generation - see expression (7)), and *f*(S_k) = Best *g*(S'_k) | $S'_k \in$ Neighborhood(S_k) (the training heuristic is detailed in Figure 2).

The evolutionary process is developed privileging the individuals presenting small differences [*g*(S_k) - *f*(S_k)] and small *g*(S_k) (costs), attributing to them the following ranks:

$$\delta(S_k) = d \times [g_{max} - g(S_k)] - [g(S_k) - f(S_k)] \qquad (5)$$

where g_{max} is random shift of high cost and *d* is a constant percentage of g_{max}. The population is dynamically controlled by an evolution parameter, denominated α, and updated as

$$\alpha = \alpha + Step \times PS \times \frac{\delta_{bst} - \delta_{wst}}{RG} \qquad (6)$$

where *Step* is a constant that controls the evolutionary process speed, *PS* is the current population size, (δ_{bst} - δ_{wst}) is the variation among the *ranks* of the best and worst individuals, respectively, and *RG* is the number of remaining generations to finish the process.

The parameter α is compared to the ranks in expression (5), and if $\alpha \geq \delta(S_k)$ then the shift S_k is eliminated from the population. The population at the evolution time α is dynamic in size and can be emptied during the process.

3.1. PTA Operators

For the PTA application in column generation, a simple local search heuristic is employed for the training function *f*, evaluating several (size of the neighborhood) alternative individuals in a neighborhood (see Figure 2).

```
Let Sk be any shift
f(Sk) := g(Sk);
S'k := clone(Sk);
for(x:=1 to size of the neighborhood)
  i := select(a task of S'k);
  j := select(any task);
  remove(the task i of S'k);
  add(the task j in S'k);
  compute(g(S'k));
  if(g(S'k) < f(Sk))
    f(Sk) := g(S'k);
  end_if
end_for
```

Figure 2. Training heuristic.

A local search mutation is also implemented selecting a task of a certain driver's shift (randomly) and changing to other, also randomly selected. This process is repeated while the new shift is invalid, i. e., it not assist the essential restrictions (see Figure 3).

The crossover generates valid individuals in the following way: two individuals are selected (base and guide) and merged. The population is maintained classified by decreasing values of ranks (as in expression (5)) and the base is selected at the first portion of the population, while the guide is selected from the total population. To guarantee a valid offspring, a random position is selected in the merged individual, the tasks in previous positions are removed and the subsequent ones inserted in a sequential way, until the new chromosome will be a valid shift (see Figure 4).

```
Individual before mutation: 1 0 0 1 0 1
Selected task: 1 0 0 1 0 1
Change of tasks:
1st change
        1 0 1 0 0 1 → invalid
2nd change
        1 1 0 0 0 1 → valid
Individual after mutation: 1 1 0 0 0 1
```

Figure 3. Mutation.

```
Base individual: 1 0 0 1 0 1
Guide individual: 0 0 1 1 1 0
Merged chromosomes: 1 0 1 1 1 1
Selected position: 1 0 1 1 1 1
The new individual's generation:
        0 0 1 1 → invalid
        0 0 1 0 1 → valid
        0 0 1 0 1 1 → valid
New individual: 0 0 1 0 1 1
```

Figure 4. Crossover.

Figure 5 gives a pseudo-code of the PTA implemented. It is interesting to note that the PTA forms populations of several sizes, guided by the objective of choosing good costs columns (shifts), with the enough covering of the tasks. The best columns will include a diversified number of tasks, characterized by the training heuristic that is driving the evolutionary process.

4. PTA/Linear Programming Interaction

Solving the SPP ((1)-(3)) can be a challenge, and the Linear Programming (LP) relaxation is used in conjunction with the PTA to generate a manageable set of columns to commercial solvers [2]. The algorithm described in Figure 6 resumes the PTA/LP interaction.

An initial population is randomly generated containing only valid columns (columns that assist the essential restrictions) and covering all the tasks. The LP is solved and new columns are generated through PTA, considering the values of the LP dual variables to construct the fitness functions *f* and *g*.

The *g*(S_k) function is defined by

$$g(S_k) = \frac{c(S_k)}{\sum_{i=1}^{m} \lambda_i a_{ik}} \qquad (7)$$

$$\theta_k = c(S_k) - \sum_{i=1}^{m} \lambda_i a_{ik} \quad (8)$$

where *c(S_k)* is done by expression (4); θ_k is the reduced cost of column *k*, and λ_i is the dual variable for task *i*.

```
1.  Initialize the parameters;
2.  create(a random initial population);
3.  while(gen. number < max. number gen.)
4.    select(base_column);
5.    select(guide_column);
6.    new_column := crossover(base,guide);
7.    evaluate(new_column);
8.    if (rand () < mutation percentage)
9.      mutation(new_column);
10.     evaluate(new_column);
11.   end_if
12.   calculate the rank(new_column);
13.   if (rank(new_column) > alpha)
14.     add(new_column in the population);
15.   end_if
16.   update(alpha);
17.   while(exist rank(column) < alpha)
18.     eliminate(column);
19.   end_while
20. end_while
```

Figure 5. PTA algorithm.

We can observe through expressions (7) and (8) that, for negative reduced costs, the *g* function values are situated in the interval [0,1]. Therefore, the training heuristic that defines the corresponding function *f* values (best *g* in a neighborhood) will assign small differences (*g* - *f*) for columns that have negative reduced costs. The population is then indirectly trained for individuals with negative reduced costs (improving columns to LP), preventing the generation of an excessive number of columns, and consequently, accelerating the process.

The columns with negative reduced costs ($\theta_k < 0$) are added to the current LP, and the new SPP is solved again through LP. The process is repeated for a certain number of iterations, and when the solution value of LP stabilizes (the same objective values are obtained for 5 consecutive iterations PTA/LP), or a certain number of iterations be reached, the process is interrupted.

However, the solution value of LP can stabilize very early, resulting in poor solutions when the SPP is solved. To prevent this, when LP stabilizes, instead of just add to the new LP the columns with negative reduced costs, all valid columns generated by PTA are added. This correction refreshes the column generation and the process continues converging.

Finally the LP is converted to the SPP and solved by the CPLEX [2]. The final problem still could be large to be solved exactly, and so the best feasible solution is kept after some time limit or gap limit (percentage difference of best SPP and LP values).

```
1.  create(population addressed to the
          problem);
2.  solve(LP);
3.  while(improving LP)
4.    execute(PTA);
5.    remove(invalid columns);
6.    calculate(reduced costs for new
              columns);
7.    if(LP not stabilizes)
8.      add(columns with negative reduced
           costs);
9.    else
10.     add(every generated columns);
11.   end_if
12.   solve(LP);
13. end_while
14. convert(LP to SPP);
15. solve(SPP);
```

Figure 6. PTA/LP algorithm.

5. Computational Tests

For validation of the implemented method, some tests were accomplished on randomly generated instances based on real problems of a Brazilian company of public transportation. The number of tasks on instances is: 25, 50, 100, 250 and 500. These instances contain tasks (start time and finish time) randomly selected of a real problem.

All the tests are performed in a laptop with Intel Celeron® processor of 2.0 GHz and 256Mb of RAM memory. The whole implementation was developed in the C++ language with calls out to the CPLEX software (see [2]).

The parameters' values used by PTA are shown in Table 1. Besides, in all the tests, the g_{max} values used were obtained starting from the largest *g* function of the generated individuals in the initial population. The value of *d* was fixed to *1/g_{max}*, in all the tests.

Table 1. PTA parameters.

NT	25	50	100	250	500
SIP	10^3	10^3	10^3	3×10^3	6×10^3
Step	10^{-5}	10^{-5}	10^{-5}	10^{-7}	10^{-10}
MNG	2×10^3	2×10^3	2×10^3	5×10^3	8×10^3
PB	0.4	0.4	0.4	0.4	0.4
PM	0.2	0.2	0.2	0.2	0.2
NS	7	7	7	7	7

Table 1 provide the following information:

- NT: number of tasks (that also identify the instance);
- SIP: size of the initial population;

- *Step*: constant that controls the speed of the evolutionary process (see expression (6));
- MNG: maximum number of generations;
- PB: individuals' percentage of the population selected as a base for crossover (base percentage);
- PM: probability of a mutation's occurrence;
- NS: neighborhood size.

The computational results are shown in Table 2. The PTA/LP approach was capable to generate good quality schedules for all instances in reasonable computational times. All the essential restrictions were assisted and the non-essential are minimized.

The results are compared against a Simulated Annealing (SA) approach applied to an initial set of columns randomly generated and searching on the training heuristic (Figure 2) neighborhood to take moves on the space of solutions. The penalties in column costs are the same defined in expression (4). Two SA solutions are reported, differing by the number of executions. SA-20 performed 20 executions for distinct initial sets of columns.

Table 2. Computational tests.

Number of tasks		**ND**	**OV (min)**	**IT (min)**	**Times (seg)**
25	PTA/LP	12	15	2356	0.30
	SA	12	15	2356	1.98
	SA-20	12	15	2356	≈ 39.60
50	PTA/LP	20	0	2600	41.64
	SA	20	7	2607	95.79
	SA-20	20	0	2600	≈1915.80
100	PTA/LP	40	0	7395	4.98
	SA	40	0	7395	67.19
	SA-20	40	0	7395	≈1343.80
250	PTA/LP	82	1058	8186	229.68
	SA	84	907	8995	434.80
	SA-20	82	1216	8344	≈8696.00
500	PTA/LP	151	1567	10961	6717.30
	SA	153	1246	11600	12132.34
	SA-20	153	1198	11552	≈242646.80

Table 2 displays the following information:

- ND: number of drivers;
- OV: total of overtimes (minutes);
- IT: total idle time (minutes);
- Times : execution times (seconds);
- PTA/LP: proposed algorithm;
- SA: Simulated Annealing [9].

Table 3 shows the number of iterations and number of columns generated by the PTA/LP algorithm in computational tests. It should be noted that in the last two experiments described in Table 2, the SPP was interrupted after 150 and 6000 seconds of execution, respectively.

Table 3. PTA/LP details.

Number of tasks	**25**	**50**	**100**	**250**	**500**
Number of iterations	5	21	9	121	501
Number of columns generated	522	11614	2315	18825	138674

6. Conclusion

This paper described an application of the PTA to generate good schedules to drives in public transportation. The PTA is integrated to traditional column generation techniques and was capable to solve the sub-problem that generates new columns in an implicitly way. This is performed defining properly the fg-fitness with dual prices information. It is somewhat general and can be used in other problems that column generation is an indicated solution method. The PTA/LP computational results are very good and obtained in reasonable execution times, compared to a Simulated Annealing metaheuristic.

7. Acknowledgements

The authors acknowledge FAPESP (process 03/04547-2) and CNPq (process 304598/2003-8) for the financial support given to the development of this work.

References

[1] Associação nacional de transporte público - ANTP website. [online]. *Available: http://www.antp.org.br.*

[2] Ilog cplex 7.5 reference manual. ©copyright 2001 by ilog. September 2001.

[3] C. F. Bouzada. Análise das despesas administrativas no custo do transporte coletivo por ônibus no município de belo horizonte. *Dissertation (Master's degree), School of Government, João Pinheiro Foundation, Belo Horizonte, Brazil*, 2002.

[4] J. R. Daduna, I. Branco, and J. M. P. Paixão. Proceedings of the sixth international workshop on computer-aided scheduling of public transport. *Computer-Aided Transit Scheduling, Springer-Verlag*, 1995.

[5] M. Desrochers and J. M. Rousseau. Proceedings of the fifth international workshop on computer-aided scheduling of public transport. *Computer-Aided Transit Scheduling, Springer-Verlag*, 1992.

[6] M. Desrochers and F. Soumis. A column generation approach to the urban transit crew scheduling problem. *Transportation Science*, 23:1–13, 1989.

[7] L. A. N. Lorena and J. C. Furtado. Constructive genetic algorithm for clustering problems. *Evolutionary Computation*, 3(9):309–327, 2001.

[8] H. R. Lourenço, J. P. Paixão, and R. Portugal. Multiobjective metaheuristics for the bus-driver scheduling problem. *Transportation Science*, 35:331–343, 2001.

[9] G. R. Mauri. Resolução do problema de programação de tripulações de um sistema de transporte público via simulated annealing. *Technical Report 02/2003, Department of Computer Science - Federal University of Ouro Preto, Brazil*, 2003.

[10] A. C. M. Oliveira and L. A. N. Lorena. 2-opt population training for minimization of open stack problem in: G. Bittencourt and G. L. Ramalho (eds.). *Advances in Artificial Intelligence, Springer Lecture Notes in Artificial Intelligence Series*, 2507:313–323, 2002.

[11] Y. Shen and R. S. K. Kwan. Tabu search for driver scheduling, in S. Voss and J. R. Daduna (eds). *Computer-Aided Scheduling of Public Transport, Berlin: Springer-Velag*, pages 121–135, 2001.

[12] S. Voss and J. Daduna. Lecture notes in economics and mathematical systems. *Computer-Aided Scheduling of Public Transport, Springer, Berlin*, 2001.

[13] W. E. Wilhelm. A technical review of column generation in integer programming. *Optimization and Engineering*, 2:159–200, 2001.

[14] N. H. M. Wilson. Proceedings of the seventh international workshop on computer-aided scheduling of public transport. *Computer-Aided Transit Scheduling, Berlin: Springer-Verlag*, 1999.

[15] A. Wren and J. Rousseau. Bus driver scheduling - an overview in: J. R. Daduna, I. Branco and J. M. P. Paixão (eds.). *Computer-Aided Transit Scheduling Transport, Springer verlag*, pages 173–87, 1995.