Journal of Universal Computer Science, vol. 23, no. 5 (2017), 479-504
submitted: 14/7/15, accepted: 23/4/17, appeared: 28/5/17

Solving a Large Real-world Bus Driver Scheduling Problem with a Multi-assignment based Heuristic Algorithm

Ademir Aparecido Constantino
(Universidade Estadual de Maringá, Maringá, Brazil
ademir@din.uem.br)

Candido Ferreira Xavier de Mendonça Neto
(Universidade Estadual de São Paulo, São Paulo, Brazil
cfxavier@usp.br)

Silvio Alexandre de Araujo
(Universidade Estadual Paulista, São José do Rio Preto, Brazil
saraujo@ibilce.unesp.br)

Dario Landa-Silva
(University of Nottingham, Nottingham, UK
dario.landasilva@nottingham.ac.uk)

Rogério Calvi
(Universidade Estadual def Maringá, Maringá, Brazil
rcalvi.uem@gmail.com)

Allainclair Flausino dos Santos
(Universidade Estadual de Maringá, Maringá, Brazil
allainclair@gmail.com)

Abstract: The bus driver scheduling problem (BDSP) under study consists in finding a set of duties that covers the bus schedule from a Brazilian public transportation bus company with the objective of minimizing the total cost. A deterministic 2-phase heuristic algorithm is proposed using multiple assignment problems that arise from a model based on a weighted multipartite graph. In the first phase, the algorithm constructs an initial feasible solution by solving a number of assignment problems. In the second phase, the algorithm attempts to improve the solution by two different procedures. One procedure takes the whole set of duties and divides them in a set of partial duties which are recombined. The other procedure seeks to improve single long duties by eliminating the overtime time and inserting it into another duty. Computational tests are performed using large-scale real-world data with more than 2,300 tasks and random instances extracted from real data. Three different objective functions are analyzed. The overall results indicate that the proposed approach is competitive to solve large BDSP.

Key Words: bus driver scheduling, crew management, heuristic, transportation, large real-world instances.
Category: F.2.1, G.1, I.1.2, I.2.8, I.6

1 Introduction

The Bus Driver Scheduling Problem (BDSP), or more generally named as the Crew Scheduling Problem (CSP) in transportation context, consists basically in generating a work schedule for a crew subject to a number of constraints and aiming to optimize a given objective function. This problem is known to be NP-hard and it has been extensively investigated as reported in the operational research literature since 1960's. Several approaches were revised (e.g. [Bodin et al., 1983, Wren and Rousseau, 1995, Wren and Wren, 1995, Beasley and Chu, 1996]) and the progress of new developments has been reported in a series of specialized workshops [Daduna and Wren, 1988]. A good summary of this development has been presented in the international workshops on Computer-Aided Scheduling of Public Transport (in 2009 this title was changed to "Conference of Advanced Systems of Public Transport" - CASPT) [CASPT2015, 2015].

The Crew Rostering Problem (CRP) and the Crew Scheduling Problem (CSP) are related problems that arise in crew management of large transportation companies [Vera Valdes, 2010, Ernst et al., 2004]. Although the problems are related, usually CRP and CSP are solved separately and sequentially. Where the Crew scheduling is related to construct of shifts for a short period of time, e.g. for a day. In this phase the shifts are not yet assigned to individual crews. The crew rostering is a second phase in crew management in which the shifts generated during the crew scheduling phase are sequenced in order to form a roster for each crew for a larger planning horizon (typically a week or a month). The main focus of crew scheduling is the cost reduction, whilst the main focus of crew rostering is more related to aspects such as quality of life, rather than related to costs [Vera Valdes, 2010].

Among the most commonly approaches using mathematical programming for solving the BDSP are the classical set covering problem (SCP) and the set partition problem (SPP) [Fores, 2001, Portugal et al., 2009], both problems are well-known to be NP-hard [Borndörfers, 2010] and its investigations have been extensively reported in the Operational Research literature. The basic idea of this approach is to use (create) a large set of feasible duties (called "duties superset" or "shift-pool") [Shijun Chen, 2013] according to labour agreement rules. It uses a matrix that each row corresponds to a task and each column corresponds to a pre-compiled potential duty and a decision variable. The total number of possible duties (columns) is usually very high [Li et al., 2015, Shijun Chen, 2013], taking a long computational time to be built. Thus, it is usual to apply some heuristic techniques to reduce the number of the columns or to divide the problem into various sub-problems, solving each sub-problem separately (e.g. heuristic techniques to solve the SCP [Beasley and Chu, 1996, Caprara et al., 1999, Wren and Wren, 1995, Ohlsson et al., 2001, Mauri and Lorena, 2007, Li and Kwan, 2003]).

Despite these heuristic techniques, we found exact approaches based on the column generation (CG) technique to solve two real-world instances, both with up to 246 tasks [Yunes et al., 2005]. Their objective is to minimize the number of bus drivers instead of minimizing the total cost. Thus, they consider the unicost set covering (all columns have the same cost equal to one). Others researchers have investigated the CG technique (e.g., [Fores, 2001, Santos and Mateus, 2007, Shijun Chen, 2013, Li et al., 2015]). Recently [Shijun Chen, 2013] presented an improved CG algorithm which was applied to a set of real problem instances with up to 701 trips[1].

A few papers have reported the use of a combination of CG with other approaches to solve the BDSP, e.g., [Santos and Mateus, 2007] reports the use of CG combined with Genetic Algorithm to solve instances with up to 138 tasks and [Li et al., 2015] reports the use of a CG based hyper-heuristic to solve real instances with 500 tasks.

Differently of the SCP or the SPP, a new mathematical formulation of the BDSP under special constraints imposed by Italian transportation rules was presented by [De Leone et al., 2011a]. Unfortunately, this formulation could only be useful when applied to small or medium-sized instances (up to 136 tasks). [De Leone et al., 2011a, De Leone et al., 2011b] report the use of a Greedy Randomized Adaptive Search Procedure (GRASP) that may be applied to larger instances. However, it can be applied to instances with at most 161 tasks.

Although most papers found in the literature report methods that use the SCP or the SPP to solve the BDSP, some studies use heuristics/meta-heuristics without using the SCP or the SPP. For example: the HACS algorithm that uses a tabu search [Shen and Kwan, 2001]; an evolutionary algorithm combined with fuzzy logic [Li and Kwan, 2003], a heuristic method, namely ZEST, derived from the process of manual scheduling [Liping Zhao, 2006]; a hybrid heuristic that combines GRASP and Rollout meta-heuristics to solve instances with 250 tasks [D'Annibale et al., 2007]; an algorithm based on Variable Neighborhood Search (VNS) to solve instances with 501 tasks [Ma et al., 2016]; an algorithm based on Iterated Local Search (ILS) [Silva and Reis, 2014] which is similar with VNS and Very Large-scale Neighborhood Search (VLNS); and a Self-Adjusting algorithm, based on evolutionary approach [Li, 2005].

Some papers report approaches that consider the scheduling of crews and the scheduling of vehicles simultaneously (or integrated), trying to solve both problems at the same time. However, the computational time has been a critical issue. For example: a combination of CG and Lagrangian relaxation which solves instances randomly generated as well as real-world data instances in which the number of tasks varies between 194 and 653 trips [Huisman et al., 2005]; a

[1] The number of tasks is not informed, but usually it is much smaller than the number of trips.

GRASP based algorithm was applied to real-world instances with up to 249 tasks [Laurent and Hao, 2008], a set partition/covering-based approaches were applied to instances with up to 400 tasks [Mesquita and Paias, 2008]; and an integrated approach to solve real-world instances with up to 653 trips [De Groot and Huisman, 2008].

Table 1: Summary of the literature review comparing different features

Reference	Objective Function	Approach	#tasks[3]	#duties[3]
[Santos and Mateus, 2007]	MinCost[1]	CG+Genetic	138	-
[De Leone et al., 2011a]	MinCost,MinDrivers[2]	GRASP, VNS	161	44
[Yunes et al., 2005]	MinDrivers	Unicost SCP+CG	246	≤ 50
[Portugal et al., 2009]	MinCost, others	SCP/SPP	347	-
[Li et al., 2015]	MinCost+MinDrivers	Hiper-heuristic	500	145
[Mauri and Lorena, 2007]	MinCost	Heuristic+CG	500[a]	153
[Ma et al., 2016]	MinCost	VNS Meta-heuristic	501	44
[Li and Kwan, 2003]	MinCost+MinDrivers	Fuzzy Genetic	613	75
[Huisman et al., 2005]	MinCost	CG	$\ll$653[a,c]	117
[Shijun Chen, 2013]	MinDrivers	SC+CG	$\ll$701[c]	100
Our case	MinCost[b]	Deterministic heuristic	2,313	340

[1] *MinCost* means minimizing the total cost.
[2] *MinDrivers* means minimizing the number of drivers.
[3] #tasks e #duties means the number of tasks and duties respectively.
[a] set of artificial benchmark instances.
[b] including vehicle changes and others functions are investigated.
[c] number of trips (the number of tasks is not provided).

Some approaches *focus on the bus lines*, i.e. the set of tasks is divided according to each bus line. Thus, the problem instance size is clearly reduced and solved separately (e.g. [Yunes et al., 2005]). However, [Portugal et al., 2009] reports an approach that *focus on the network*, i.e. the set of tasks is not divided according to each bus line and each task can be assigned to any driver despite the bus line it belongs to. In the work reported here we follow this approach. The set of tasks arises from 55 different bus lines and, due to the planning strategy of the company, are not separated. This means that a driver can drive in different bus lines during his/her daily duty. The company desires to minimize the number of bus line changes during a daily duty. Thus, this information is incorporated in our objective function as detailed in Section 3.

In this paper we propose a new heuristic algorithm (named GraphBDSP) based on solution of successive linear assignment problems that uses a weighted multipartite graph which represents the problem data. This approach tries to find disjoint paths in this graph minimizing the total cost (paid time) where each path means a duty (for a driver). The algorithm has two phases, a constructive phase and an improvement phase. To the best of our knowledge this is the first time this approach is applied to solve a BDSP. We applied GraphBDSP with real-world problem instances from a Brazilian transportation bus company with 55 bus lines and more than 2,300 tasks, with about 340 duties and with two depots. In order to compare our results against the results reported in the literature we summarize our bibliography review in terms of instance sizes, features and approaches (see Table 1). According to our review, our problem instance is four times larger than the largest problem instance reported in the literature. Since we are dealing with an NP-hard problem spending a high computational effort to solve smaller instances, as reported, we believe that a heuristic technique can be justified to obtain good quality feasible solutions. The main advantages of the GraphBDSP is its effectiveness, its efficiency (low asymptotic complexity) and its facility in terms of parameter tuning.

In this context, this paper makes three contributions. The first contribution is the proposition of a new heuristic approach to solve a huge BDSP instance. Further, to the best of our knowledge, no paper considers this same set of labor rules (constraints and objective function). Also, it is worth noticing that this algorithm requires only one parameter to tune due to the simplicity of the criterion defined by [Cordeau et al., 2002]. The second contribution of this paper is to make available those new problem instances in order to promote new researches around this subject, including a large real-word problem instance. To the best of our knowledge, no paper deals with more than 500 tasks by exact approach. The third contribution is the application of three different objective functions which are compared, concluding that is more important to focus on idle time and overtime time instead of focusing on paid time only.

This paper is organized as follows. Section 2 gives the problem description. Section 3 presents the GraphBDSP in detail. Section 5 reports the computational results of GraphBDSP compared with set covering methods, giving lower bounds. Finally, Section 6 presents the conclusion remarks.

2 The Bus Driver Scheduling Problem

This paper considers a specific case of the bus driver scheduling problem from a Brazilian company where the work schedule is a set of disjoint duties in which a whole set of pre-defined bus tasks must be covered. In this case the total cost (paid time) and the size of the crew must be minimized. A feasible duty must meet the company regulations and the specific Brazilian legal restrictions established by both law and contracts with labor unions which further increases the complexity of the problem.

Terminology

Below we report some terminology used in this paper:

- A *block* represents a sequence of trips to be operated in a day by a vehicle from the time it leaves the depot (garage or the place where it is parked) until it returns to the same depot. These blocks are results from the vehicle scheduling[2].
- A *relief opportunity* is a pair of place and time, into a block, in which a driver is allowed to leave or assume a vehicle.
- A *task* is a work defined by two consecutive relief opportunity and represents the minimum portion of work that can be assigned to a driver. Thus a task may involve one or more trips, usually two trips, or a deadhead[3].
- A *bus line* is a route (itinerary) over which a bus regularly travels, although it is possible to use a bus in different lines.
- A *running table* is part of a block when the bus is used over the same line. Each row of a running table containing four entries which represent a *task* t. These entries are defined by $t = (sl, st, el, et)$, the first two entries indicate where and when the task starts the next two entries indicate where and when it ends, respectively. The information of which bus line a task t belongs is given by $bl(t)$ and the values of each entry of the task t is given by $sl(t)$, $st(t)$, $el(t)$ and $et(t)$, respectively.
- A *crew* is a team of a driver and a conductor, but in some cases it may consist of a driver only. When we *assign* a crew to a task t, that means that the crew is running the corresponding bus line from the place sl and time st to the place el and time et.
- A *piece of work* (POW) is a sequence of consecutive tasks t_i, t_{i+1}, ..., t_j, in the same bus line, assigned to the same crew where $et(t_k) \leq st(t_{k+1})$.
- A *stretch* is a sequence of consecutive pieces of work (not necessarily on the same bus line) without an intervening meal-break (rest time). Note that, by definition, a POW is a stretch and a single task is a POW. A POW (or a stretch) $P = (t_i, t_{i+1}, ..., t_j)$ starts at the same location (resp. time) where (resp. when) its first task starts. Thus, $sl(P) = sl(t_i)$ and $st(P) = st(t_i)$. Also, it ends at the same location (resp. time) where (resp when) its last task ends. Thus, $el(P) = el(t_j)$ and $et(P) = et(t_j)$.

[2] We assume that the vehicle blocks is determined a priory by solving the vehicle scheduling. In this case we used the solutions provided by the public transport company.

[3] Trip without passengers.

– There are two types of breaks:
  - a *rest time* (rest break, meal-break) which is an unpaid time between two task assigned consecutively to the same driver; and
  - a *idle time* (idle break) which is also a time between two tasks assigned consecutively to the same crew, however, idle time is a paid time used to complete a duty.

– A *duty* D is an ordered sequence of tasks t_1, t_2, ..., t_k (composed of one or more stretches) that can be assigned to a crew in a single day work.

– A *spreadover time* (given by *sot*) of a duty D is the time between the start of the first task of D and the end of the last task of D, $sot(D) = et(t_k) - st(t_1)$.

In this work, when we assign a task t to the end of a duty D, the task is always assigned to the same crew of duty D forming a new sequence where the task t will be in the last position. Therefore, for simplicity we say that we assign task t to the end of duty D.

Figure 1 shows two bus lines A and B. Each bus line consists of a set of four entry columns where each row indicates all the tasks a bus must perform in a day (bus 1 and bus 2 for line A and bus 4, bus 5 and bus 6 for line B). This figure also shows two duties:

The Duty$_X$ where the first 3 tasks of the duty are Task$_1$, Task$_2$ and Task$_5$. Note that, the first two tasks of the duty forms a POW and there is a rest time between Task$_2$ and Task$_5$; and

The Duty$_Y$ where the first 3 tasks of the duty are Task$_6$, Task$_3$ and Task$_4$. The whole duty consists of these 3 tasks, therefore, the crew will have a 367 minutes of idle time to complete their day work. Note that the crew assigned to this duty must change the bus line between Task$_6$ Task$_3$. Thus these 3 tasks form a stretch .

2.1 Problem Specification

A feasible duty must meet the company regulations and the specific Brazilian legal restrictions established by both law and contracts with labor unions as follows.

Rule 1: The stretch time on a duty may not exceed 360 minutes, and if it exceeds this time, a rest time of at least 90 minutes must be applied. Rest time on a duty (unpaid time) may not exceed 300 minutes, otherwise the break time over that is considered idle time (paid time).

Rule 2: The overtime per duty may not exceed 120 minutes.

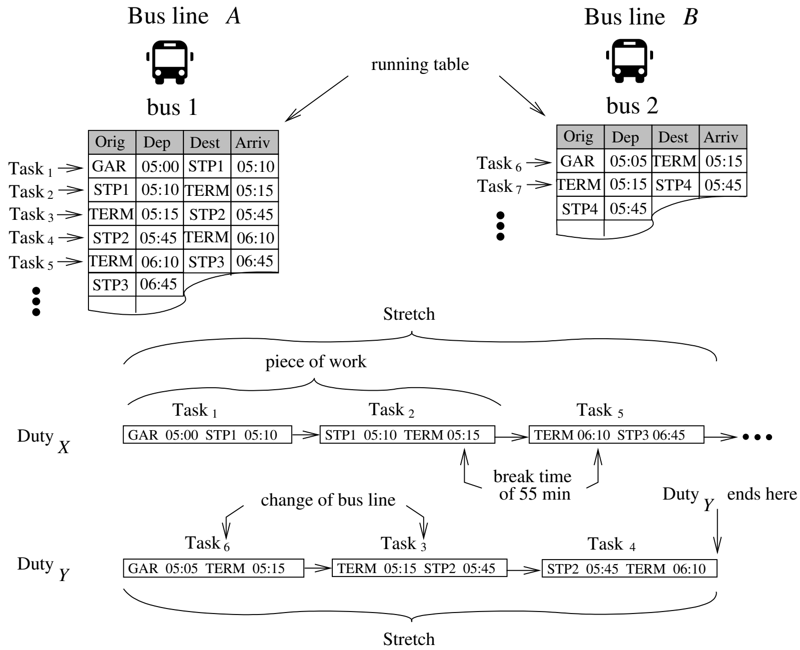


Figure 1: Two bus lines showing a lists of tasks and two duties.

Rule 3: If a duty includes *night working*, between 10pm and 5am, a period of 52 minutes is considered to be 60 minutes of work (called night time) and an additional payment of 20% of ordinary wage must be applied over such night working time. For overtime (extra working time) an additional payment of 50% of ordinary wage must be applied.

Rule 4: The total paid time per duty must be at least 432 minutes.

Rule 5: The spreadover time of a duty may not exceed 780 minutes.

Table 2 summarizes a set of parameters related to these rules. In Section 3.3 we give a mathematical formulation of this rules which are used by our objective function. As solution to this problem means a set of duties covering all tasks of a working day in such way that the total cost based on the paid time (worked time) and the number of vehicle change are minimized, consequently, minimizing the total number of duties. According to our literature review, there is not linear programming formulation that satisfies this set of rules and this formulation is still a challenge. In addition, taking into account our main problem instance is large, thus a need for heuristic solution approaches is justified.

Table 2: Parameter notation from the problem.

Parameter/Value	Meaning	Reference
$st_{max} = 360$ min	Maximum stretch time duration	Rule 1
$rt_{min} = 90$ min	Minimum time for a rest	Rule 1
$rt_{max} = 300$ min	Maximum time for a rest	Rule 1
$pt_{min} = 432$ min	Minimum paid time for a duty	Rule 4
$ot_{max} = 120$ min	Maximum overtime for a duty	Rule 2
$sot_{max} = 780$ min	Maximum spreadover time duration	Rule 5
$fnt = 60/52$	Correction factor for night time worked	Rule 3
$pnt\% = 20\%$	Percentage of additional payment for night time	Rule 3
$pot\% = 50\%$	Percentage of additional payment for overtime	Rule 4

3 Proposed Algorithms

The proposed algorithm GraphBDSP is based on successive solution of (linear) assignment problems in order to construct and to improve different paths (duties) in a multipartite graph. The assignment problem [Pentico, 2007] is a well-known problem in operation research and it can be solved in polynomial time. The assignment problem has been quite used due to its low computational complexity to get optimal solution, e.g. recently it was successfully used to tackle a nurse scheduling problem [Constantino et al., 2014]. Given a cost matrix $[c_{ij}]$ of dimension $n \times n$, the assignment problem consists in associating each row to a column in such a way that the total cost of the assignment is minimized. Using a binary variable $x_{ij} = 1$ if row i is associated with column j, the assignment problem can be formulated as follows:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1}$$

$$\textbf{Subject to: } \sum_{i=1}^{n} x_{ij} = 1, j = 1, \ldots, n \tag{2}$$

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, \ldots, n \tag{3}$$

$$x_{ij} \in \{0, 1\}, i, j = 1, \ldots, n \tag{4}$$

Our GraphBDSP algorithm is decomposed into two phases (algorithms) which are run in sequence. 1) construction of an initial solution; and 2) solution improvement. In both phases the assignment problem (1)-(4) is intensely used, where the number n and the cost matrix $[c_{ij}]$ have different meaning (values) for each phase of the algorithm, which are explained in detail in the following sections.

3.1 Data and Solution Representation

First of all our algorithm starts with a weighted multipartite *directed graph* $G = (T, E)$ (or simply a *graph*) where T is the set of *vertices* representing the tasks the whose crews must be allocated and E is the set of *edges* $e_{ij} = (t_i, t_j)$ indicating that the same crew can perform the task t_j after the task t_i. In this case we say that t_i is *adjacent* with t_j and that e_{ij} *leaves* vertex t_i and *enters* vertex t_j.

It is possible to place the set of vertices into *layers* T_1, T_2, ..., T_k ($T = \cup_{m=1}^{k} T_m$) such that all edges are "forward", i.e. for each edge $e_{ij} = (t_i, t_j)$, vertex t_i is placed on layer T_l and vertex t_j is placed on layer T_m where $m > l$. The vertices are arranged into layers in such way that each layers is made up of vertices representing tasks, if two tasks can be scheduled one after other, then they are not in the same layer, i.e. if task t_j can be scheduled after the task t_i, then $t_i \in T_l$ and $t_j \in T_m$, $l < m$. A simple greedy algorithm can build such partition. Let s_m denote the size of the layer T_m ($1 \leq m \leq k$).

Let a *path* $D = (T', A')$ be a sequence $(v_1, v_2, ..., v_u)$ of vertices of G such that v_p is adjacent with v_{p+1} ($1 \leq p \leq u - 1$). Its easy to show that a path meets each layer at most once. Two paths D_r and D_s are *disjoint* if and only if $D_r \cap D_s = \emptyset$. In the reminder of this work, we will refer to the vertices of G as tasks and refer to the paths constructed by the algorithm as duties. Thus our algorithm tries to find a set of disjoint paths (duties) that meets all vertices (tasks) of the graph (a partition of T) minimizing the total cost. A solution is this set of disjoint paths.

Let a *partial duty* be an incomplete duty, this means that it is possible to include more task to obtain a complete duty. A partial duty is always referred with two indexes were the second index indicates the context of that part. Thus, a complete duty (or simply a duty) is referred with a single index. For example, we use D_i to refer to a duty i to be carried out by a crew in a single day which can be split into two partial duties $D^l_{i,m-1}$ (called *left partial duty*) and $D^r_{i,m}$ (called *right partial duty*), where $D^l_{i,m-1}$ contain the tasks $t_{i,1}$, $t_{i,2}$, ..., $t_{i,m-1}$ and $D^r_{i,m}$ contain the tasks $t_{i,m}$, $t_{i,m+1}$, ..., $t_{i,k}$.

Our graph is edge-weighted graphs but the weight of each edge is computed dynamically. Since the additional cost of adding a task to a partial duty, or joining two partial duties, may not depend only on the last task of the partial duty $D^l_{i,m-1}$, but it may depend on the cost of the whole duty. Next section we give more detail how to compute this weight using a cost function.

3.2 Cost Function

Let λ-*cost* be a function that receives two parameters: a partial duty $D^l_{i,m-1}$ and a partial duty $D^r_{j,m}$ and returns the cost (related to paid time in minutes) of assigning the duty $D^r_{j,m}$ at the end of duty $D^l_{i,m-1}$ forming the duty D_i. If D_i is *feasible* the λ-cost function returns the paid time in minutes; otherwise it returns ∞ (if it is *unfeasible*). The λ-*cost* function is defined in equation 5.

$$\lambda(D^l_{i,m-1}, D^r_{j,m}) = \sum_{r=1}^{5} \lambda_r \tag{5}$$

where λ_r is defined according to the set of rules stated in Section 2.1:

Rule 1: Let $bt(D_i) = \max_{u=1}^{k-1}\{(st(t_{i,u+1}) - et(t_{i,u})) \mid ((et(t_{i,u}) - st(D_i)) \leq st_{max}) \wedge ((et(D_i) - st(t_{i,u+1})) \leq st_{max})\}$ be the longest break time of the duty D_i which is candidate to be a rest time. If $sot(D_i) \leq st_{max}$ we set $\alpha = 0$; otherwise we set $\alpha = \infty$. Let $rt(D_i)$ be the rest time of the duty D_i, i.e. $rt(D_i) = min\{bt(D_i), rt_{max}, \alpha\}$. If $rt(D_i) = 0$ or $rt(D_i) \geq rt_{mim}$ we set $\lambda_1 = 0$, otherwise we set $\lambda_1 = \infty$.

Rule 2: Let $ot_{day}(D_i)$ and $ot_{night}(D_i)$ be the overtime worked during the day and night (between 10pm and 5pm), respectively. Thus, $ot(D_i) = ot_{day}(D_i) + ot_{day}(D_i) \cdot fnt\%$. If $ot(D_i) \leq ot_{max}$ we set $\lambda_2 = 0$, otherwise we set $\lambda 2 = \infty$.

Rule 3: Let $wt(D_i)$ be the amount of working time of the duty D_i, i.e. $wt(D_i) = \min\{sot(D_i) - rt(D_i), pt_{min}\}$. Let $nt(D_i)$ be the amount of night time worked and let $pt(D_i)$ be the amount of paid time of the duty D_i, i.e. $pt(D_i) = (wt(D_i) - nt(D_i)) + (nt(D_i) \cdot (1 + pnt\%)) + (ot(D_i) \cdot pot\%)$. Thus we set $\lambda_3 = pt(D_i)$.

Rule 4: If $pt(D_i) \geq pt_{min}$ we set $\lambda_4 = 0$, otherwise we set $\lambda_4 = \infty$.

Rule 5: If $ot(D_i) \leq ot_{max}$ we set $\lambda_5 = 0$, otherwise we set $\lambda_5 = \infty$.

A δ-*penalty* function is defined as the value of a quantified penalty to reflect the bus line change and the feasibility of join $D^l_{i,m-1}$ to $D^r_{j,m}$. The δ-*penalty* function is defined in equation 6.

$$\delta(D^l_{i,m-1}, D^r_{j,m}) = \delta_1 + \delta_2 \tag{6}$$

where δ_1 and δ_2 are computed as follows:

a)Let $vc(D_i)$ be the number of bus line change in D_i, thus we set $\delta_1 = (pblc \cdot vc(D_i))$, where *pblc* is a penalty for a bus line change.

b)$\delta_2 = 0$ is $(et(D^l_{i,m-1}) \leq st(D^r_{j,m}))$ or $(el(D^l_{i,m-1}) = sl(D^r_{j,m}))$, otherwise $\delta_2 = \infty$.

We define a cost function f which receives as parameter a pair of partial duties $D^l_{i,m-1}$ and $D^r_{j,m}$ and returns the cost of assigning all tasks of the partial duty $D^l_{i,m-1}$ at the end of the partial duty $D^r_{j,m}$.

In this work, we investigate three versions as follows:

$f_1(D^l_{i,m-1}, D^r_{j,m}) = \lambda(D^l_{i,m-1}, D^r_{j,m}) + \delta(D^l_{i,m-1}, D^r_{j,m})$, i.e. the in function f_1 only considers the paid time of assigning the partial duty $D^l_{i,m-1}$ at the end of the partial duty $D^r_{j,m}$.

$f_2(D^l_{i,m-1}, D^r_{j,m}) = (\lambda(D^l_{i,m-1}, D^r_{j,m}) - \lambda_3) + \delta(D^l_{i,m-1}, D^r_{j,m}) + it(D^l_{i,m-1}, D^r_{j,m})$, where $it(D^l_{i,m-1}, D^r_{j,m})$ returns the total idle time of the duty. This function focuses on idle time (paid break time) of the duty.

$f_3(D^l_{i,m-1}, D^r_{j,m}) = (\lambda(D^l_{i,m-1}, D^r_{j,m}) - \lambda_3) + \delta(D^l_{i,m-1}, D^r_{j,m}) + it(D^l_{i,m-1}, D^r_{j,m}) + (ot(D_i) \cdot pot\%)$. This function focuses on idle time (paid break time) and overtime.

Let $\mathcal{D}$ be the set of duties kept by the algorithm in a given time (the current solution). The cost f associated with $\mathcal{D}$ is given by $f(\mathcal{D}) = \sum_{i=1}^{|D|} f(D_i)$, where f can be f_1,f_2 or f_3. These functions return ∞ if $\mathcal{D}$ is *unfeasible*.

3.3 Construction Phase

In the construction phase a feasible schedule is built, i.e., a set of feasible duties, adding one task at a time according to a assignment problem, until all tasks have been assigned. Starting with an empty duties, iteratively each duty is constructed.

Let d be the number of estimated duties needed to assign all tasks, thus, it must be enough to assign the tasks to one of the d partial duties. We used $d = 2.5nv$, where nv is the number of vehicles obtained by the vehicle scheduling.

We define *dummy task* t' as a not real task and there is no restriction to assign it to any partial duty at any release opportunity without changing any cost or property. In addition, $sl(t') = el(t') = NULL$, $st(t') = et(t') = 0$. But since it is assigned to any partial duty, then $sl(t')$ and $st(t')$ ($el(t')$ and $et(t')$) give information of its first real right (left, respectively) neighbor task in that duty. Note that a dummy task may be assigned to any partial duty, but in case of join two partial duty, a dummy task assumes the properties of its first real neighbor task, i.e. if a dummy task $t'_{i,m-1} \in D^l_{i,m-1}$ then it assumes the properties of its first real left neighbor task in $D^l_{i,m-1}$, similarly, if a dummy task $t'_{i,m} \in D^r_{i,m}$ then it assumes the properties of its first real right neighbor task in $D^r_{i,m}$.

Dummy duty is introduced in order to simplify our duty notation, keeping the same size and starting and ending at the same layer for all duties. But in a computational implementation, all dummy duties may be eliminated by using of pointers linking all sequence of real tasks in a duty, i.e. a duty may be represented by a dynamic list.

During this construction phase a duty D_i is constructed incrementally by assigning a task by iteration. Some of these tasks may be dummy tasks. Initially, we start with $D_{i,0} = \emptyset$ and $D_{i,m}$ is the partial duty obtained by assigning the task t at the end of the partial duty $D_{i,m-1}$. In this step the task t becomes the task t_m of the duty $D_{i,m}$, i.e. $t_m \leftarrow t$, thus in this construction phase we considers $D^r_{i,m}$ containing the task t_m only, i.e. $D^r_{i,m} = \{t_m\}$ in order to use our cost function f.

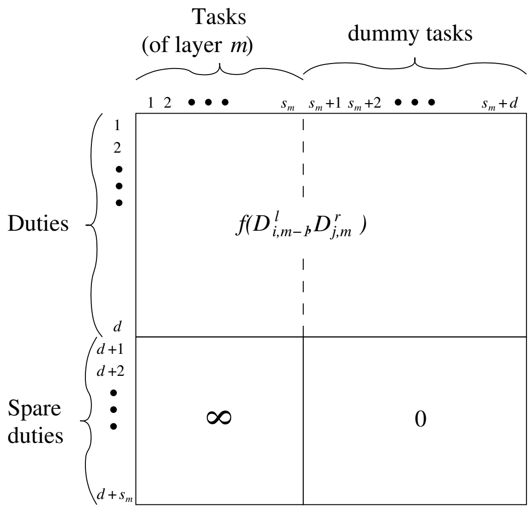
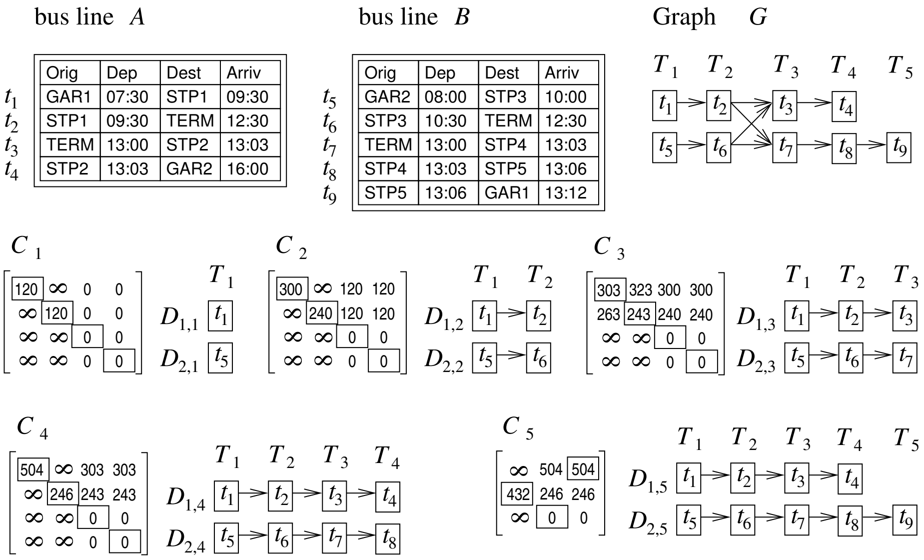


Figure 2: The cost matrix C for each iteration m.

We construct a cost (weight) matrix C of order $d + s_m$ at the beginning of each iteration m as shown in Figure 2. The figure also shows the meaning of the entries of the cost matrix C, i.e. the tasks of the layer m correspond to the tasks of columns 1, 2, ..., s_m, the next d columns correspond to dummy tasks, the lines 1, 2, ..., d corresponds to the d duties and the remaining s_m lines correspond to the spare duties which cannot be used for assigning real tasks.

Summarily, the procedure to construct the initial solution can be described by Algorithm 1. It start with d empty duties. Solving the assignment problem with this cost matrix C we find how to assign each task t from T_m to each partial duties $D^l_{i,m-1}$, $i = 1, \cdots, d$.



bus line A

	Orig	Dep	Dest	Arriv
t_1	GAR1	07:30	STP1	09:30
t_2	STP1	09:30	TERM	12:30
t_3	TERM	13:00	STP2	13:03
t_4	STP2	13:03	GAR2	16:00

bus line B

	Orig	Dep	Dest	Arriv
t_5	GAR2	08:00	STP3	10:00
t_6	STP3	10:30	TERM	12:30
t_7	TERM	13:00	STP4	13:03
t_8	STP4	13:03	STP5	13:06
t_9	STP5	13:06	GAR1	13:12

Figure 3: On the top, 2 tables of 2 bus lines containing 9 tasks, and the graph generated by those tables; On the next two rows it is shown a cost matrix C_m built in iteration m followed by the set of the partial duties constructed according with the solution of the Assignment Problem for that matrix.

Algorithm 1: *ConstructSolution*(G)

```
/* receive a multipartite graph G as defined                    */
begin
    Set all duties D_{i,0} = ∅ for i = 1, 2, ..., d;
    for m ← 1 to k do
        Take the layer T_m from G;
        Set up the cost matrix C according to function f;
        Solve the assignment problem described by cost matrix C;
        Assigning the tasks from T_m to each partial duty D^l_{i,m−1},
          i = 1, ··· , d according to assignment solution;
    Remove all empty duty from 𝒟;
    return 𝒟;
```

Figure 3 shows a complete run of Algorithm 1. We start with the 2 tables on the top left of the figure and build the graph on the top right. Now we run Algorithm 1. In the first iteration ($m = 1$) it finds that $T_1 = \{t_1, t_5\}$. For this layer, it builds the matrix C_1. One of the solutions (of the Assignment Problem) is marked with rectangles which sets $D_{1,1} = (t_1)$ and $D_{2,1} = (t_5)$ (see first pair of partial duties on the right of matrix C_1). In the second iteration ($m = 2$) it finds that $T_2 = \{t_2, t_6\}$. For this layer, it builds the matrix C_2, again, the solution is marked with rectangles which sets $D_{1,2} = (t_1, t_2)$ and $D_{2,2} = (t_5, t_6)$ (shown in the right of matrix C_2). The same applies to iteration 3 and 4. At the last iteration ($m = 5$) it finds that $T_5 = \{t_9\}$. For this layer, it builds the last matrix C_5 where the solution is marked with rectangles and sets $D_{1,5} = (t_1, t_2, t_3, t_4)$ and $D_{2,5} = (t_5, t_6, t_7, t_8, t_9)$. Note that it assigned dummy tasks to duty $D_{1,5}$ in this last step. Since all tasks are set into layers, the algorithm ends. Thus, we set $D_1 \leftarrow D_{1,5}$ and $D_2 \leftarrow D_{2,5}$.

3.4 Improvement Phase

A *duty cut* m on a duty D_i, denoted by $C^m_{D_i}$, is a partition of the tasks of D_i into two sets of tasks forming two partial duties: *left partial duty* $D^l_{i,m}$ and *right partial duty* $D^r_{i,m+1}$, $m = 1, \cdots, k-1$.

A *cut* m on the set $\mathcal{D}$, denoted by $C^m_{\mathcal{D}}$, is a set of *duty cut* m, i.e. is the set of the partial duties formed by the duty cuts on all duties on $\mathcal{D}$ such that $\mathcal{D} = \mathcal{D}^l_m \cup \mathcal{D}^r_{m+1}$ where $\mathcal{D}^l_m = \{D^l_{i,m}, i = 1, \cdots, |D|\}$ and $\mathcal{D}^r_{m+1} = \{D^r_{i,m+1}, i = 1, \cdots, |\mathcal{D}|\}$.

Lemma 1. *If $C^m_{\mathcal{D}}$ is cut m on the set $\mathcal{D}$ then duties from $\mathcal{D}^l_m$ may be recombined with $\mathcal{D}^r_{m+1}$ forming a new set of duties $\mathcal{D}'$ in such a way that $f(\mathcal{D}) \geq f(\mathcal{D}')$.*

Proof. Let $\mathcal{D}$ be an instance of the set of duties constructed by algorithm 1. Let $C^m_{\mathcal{D}}$ be a cut m on $\mathcal{D}$ $(1 \leq m \leq k-1)$. Let E be a square matrix $|\mathcal{D}| \times |\mathcal{D}|$ where each line i $(i = 1, 2, ..., |\mathcal{D}|)$ corresponds to the partial duty $D^l_{i,m} \in \mathcal{D}^l_m$, each column j $(j = 1, 2, ..., |\mathcal{D}|)$ corresponds to the partial duty $D^r_{j,m+1} \in \mathcal{D}^r_{m+1}$ and each entry $e_{i,j}$ corresponds to the cost of assigning all tasks of the partial duty $D^r_{j,m+1}$ at the end of the partial duty $D^l_{i,m}$, i.e. $e_{i,j} = f(D^l_{i,m}, D^r_{j,m+1})$ (the function f used here is the same used in the construction phase) is element of the cost matrix E used by assignment problem. Since, the set of duties $\mathcal{D}$ belongs to the matrix E (it is the diagonal $e_{i,i}$ for $i = 1, 2, ..., |\mathcal{D}|$) and the solution of the assignment problem that recombines $\mathcal{D}^l_m$ with $\mathcal{D}^r_{m+1}$ into a new set of duties $\mathcal{D}'$, then $f(\mathcal{D}) \geq f(\mathcal{D}')$, the assertion follows from the minimum solution found by the assignment problem. □

Let *Recombine*($C^m_{\mathcal{D}}$) be an operation which receives *cut* m on the set $\mathcal{D}$, $1 \leq m \leq k-1$, and recombine the partial duties as stated by Algorithm 2.

Let $\mathcal{D}'$ be the set of duties obtained by performing a *Recombine* operation on the set of duties $\mathcal{D}$. It follows from Lemma 1 that $f(\mathcal{D}) \geq f(\mathcal{D}')$. However, if $f(\mathcal{D}) > f(\mathcal{D}')$ then new set of duties is an improvement over the previous one,

Algorithm 2: *Recombine*$(C^m_{\mathcal{D}})$

begin
  Recombine the partial duties $\mathcal{D}^l_m$ and $\mathcal{D}^r_{m+1}$ of $\mathcal{D}$ into a new set of duties $\mathcal{D}'$ according to the assignment problem solution using the cost matrix E as defined in the proof of Lemma 1;
  Remove all empty duty from $\mathcal{D}'$;
  return $\mathcal{D}'$;

otherwise ($f(\mathcal{D}) = f(\mathcal{D}')$) and both sets are equivalent. In any case, set $\mathcal{D} \leftarrow \mathcal{D}'$ at the end of each iteration.

The general improvement algorithm (Algorithm 5) uses two algorithms: $Imp1$ and $Imp2$. Algorithm $Imp1$ (Algorithm 3) performs successive *cuts* m on the set $\mathcal{D}$ and tries to recombine them. Algorithm $Imp2$ (Algorithm 4) scans tasks with overtime and tries to reassign them to other duties in order to reduce the total cost.

Algorithm 3: $Imp1(\mathcal{D})$

begin
  for $m \leftarrow 1$ **to** $k-1$ **do**
    $\mathcal{D} \leftarrow Recombine(C^m_{\mathcal{D}})$;
  Remove all empty duty from $\mathcal{D}$;
  return $\mathcal{D}$;

Let $lot(D_i)$ be the index of the last no dummy task of duty D_i if $ot(D_i) > 0$ (it has overtime), otherwise $lot(D_i) = 0$, i.e. $lot(D_i) = \max_{u=1}^{k} \{u \mid (t_{i,u} \in D_i \wedge (t_{i,u})$ is not dummy task $\wedge (ot(D_i) > 0), 0\}$.

Algorithm 4: $Imp2(\mathcal{D})$

begin
  for $i \leftarrow 1$ **to** $|\mathcal{D}|$ **do**
    $m \leftarrow lot(D_i) - 1$;
    if $m > 0$ **then**
      $\mathcal{D} \leftarrow Recombine(C^m_{\mathcal{D}})$;
  Remove all empty duty from $\mathcal{D}$;
  return $\mathcal{D}$;

Let n_{it} be a positive integer number that defines the maximum iteration without improvement as a stop criterion. We used this parameter n_{it} because we noted that the incumbent solution may be changed without improving its cost in an iteration, but this changing gives an opportunity to improve the incumbent solution in a next iteration. Thus, the general improvement algorithm is stated by Algorithm 5. The algorithm ends after performing the two procedures $Imp1$ and $Imp2$ without improvement for n_{it} times.

Algorithm 5: $ImproveSolution(\mathcal{D})$

`/* receive a solution` $\mathcal{D}$ `from the construction phase */`

begin
    $i_1 \leftarrow 0$; $i_2 \leftarrow 0$;
    repeat
        $i_1 \leftarrow i_1 + 1$;
        repeat
            $i_2 \leftarrow i_2 + 1$;
            $\mathcal{D}' \leftarrow Imp1(\mathcal{D})$;
            if $f(\mathcal{D}') < f(\mathcal{D})$ **then**
                $i_1 \leftarrow 0$; $i_2 \leftarrow 0$;
            $\mathcal{D} \leftarrow \mathcal{D}'$;
        until $i_2 > n_{it}$;
        $i_2 \leftarrow 0$;
        repeat
            $i_2 \leftarrow i_2 + 1$;
            $\mathcal{D}' \leftarrow Imp2(\mathcal{D})$;
            if $f(\mathcal{D}') < f(\mathcal{D})$ **then**
                $i_1 \leftarrow 0$; $i_2 \leftarrow 0$;
            $\mathcal{D} \leftarrow \mathcal{D}'$;
        until $i_2 > n_{it}$;
    until $i_1 > n_{it}$;
    return $\mathcal{D}$;

Figure 4 shows a sample of an iteration performed by algorithm Imp_1. The initial bus lines, graph and duties built by algorithm 1 are the same shown in Figure 3. The cost of the initial solution $\mathcal{D}$ is $f_1(D_1)+f_1(D_2) = 504+432 = 936$. There are 5 layers in the graph. Thus, there are 4 possible cuts: cut_1 between layers T_1 and T_2, cut_2 between layers T_2 and T_3, cut_3 between layers T_3 and T_4 and cut_4 between layers T_4 and T_5. In this figure it is considered the cut_2 (see Figure 4 top right). This cut will generate the $C^2_{\mathcal{D}} = \{\{D^l_{1,2}, D^l_{2,2}\}, \{D^r_{1,3}, D^r_{2,3}\}\}$ where $D^l_{1,2} = (t_1, t_2)$, $D^r_{2,2} = (t_5, t_6)$, $D^r_{1,3} = (t_3, t_4)$, $D^r_{2,3} = (t_7, t_8, t_9)$. The computation of the matrix E is as follows: $e_{1,1} = f(D^l_{1,2}, D^r_{1,3}) = 504$ which

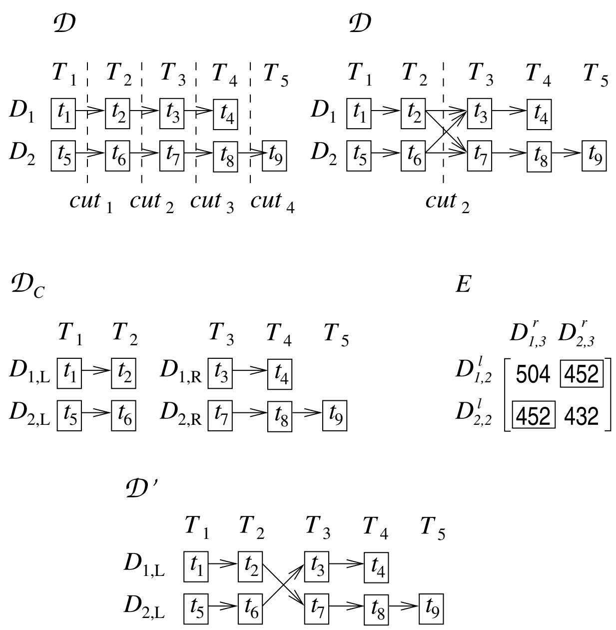


Figure 4: A sample of an iteration with a cut between layers T_2 and T_3.

is the cost of 480 minutes of work time plus 50% of 48 minutes of overtime; $e_{1,2} = f(D^l_{1,2}, D^r_{2,3}) = 452$ which is the cost of 312 minutes of work plus 120 minutes of idle time and 20 minutes as a penalization for bus line change; $e_{2,1} = f(D^l_{2,2}, D1, 3^r) = 432$ which is the cost of 420 minutes of work plus 12 minutes of idle time and 20 minutes as a penalization for bus line change. The solution of the Assignment Problem is shown in rectangles which is the set $\{e_{1,2}, e_{2,1}\}$. Thus, $\mathcal{D}' = \{D'_1, D'_2\}$ where $D'_1 = (t_1, t_2, t_7, t_8, t_9)$ and $D'_2 = (t_5, t_6, t_3, t_4)$. The cost of the solution $\mathcal{D}'$ is $f(\mathcal{D}') = 904$ which is smaller than the cost $f(\mathcal{D}) = 936$. Thus, the $\mathcal{D}'$ is an improvement over $\mathcal{D}$. Naturally, if the penalization for a bus line change were too high (above 36 minutes), this improvement should not be possible. Note that, procedure Imp_2 may only perform cut_3 which cut the last non dummy task t_4 of duty D_1 which has overtime. Therefore, in this particular case, procedure Imp_2 does not improve the solution.

4 Computational Complexity Analysis

The assignment problem can be solved in O(n^3) running time [Carpaneto and Toth, 1987], where n assume different values according to the algorithm phase. In the construction phase $n = d + s_m$, while $n = |\mathcal{D}|$ in the improvement phase.

Anyway d, s_m e $|\mathcal{D}|$ depend on the task number nt, that is $k < max_s \ll |\mathcal{D}| \ll d \ll nt$, where $max_s = max\{s_m, m = 1, \ldots, k\}$. An assignment problem is solved k times for each iteration of Imp_1. The Imp_2 is only applied for the last layer where there is overtime. Thus, the time complexity of Imp_1 and Imp_2 is $\mathrm{O}(nt^4)$. Usually these procedures Imp_1 and Imp_2 are not applied more than nt times. Therefore, we conclude that the overall complexity of our algorithm is $\mathrm{O}(nt^5)$.

5 Computational Results and Analysis

Algorithm GraphBDSP is tested using real-world instances[4] from a large Brazilian metropolitan transportation company obtained of different days in a year. The instances are listed below; being the numerical part of the name an indication of the number of tasks (e.g. CV412 contains 412 tasks): CV412, CC130, CC251, CC512, CC761, CC1000, CC1253, CC1517, CC2010 and C2313. The last three instances (CC1517, CC2010, C2313) are real cases, while the remaining were randomly created by extracting of vehicle blocks from these real instances.

Our algorithm was coded in Pascal language. All these computational tests were carried out on a PC with an Intel 2.8Ghz processor and 8GB of RAM memory, running Windows operational system.

For the solution of the assignment problem we used the algorithm proposed by [Carpaneto and Toth, 1987], which combines the Hungarian method and the Shortest Augmenting Path method. To get the solution of the integer linear programming (ILP) the CBC solver [Ralphs, 2015] were used. CBC is maintained by IBM researchers, which is pretty competitive to current state-of-the-art commercial ILP solvers.

5.1 Computational Results

Table 3 lists the results obtained by our GraphBDSP using the functions f_1, f_2 and f_3 by the construction of the initial solution. We set up the parameters $pblg = 1$ (penalty for each bus line change) and $n_{it} = 4$ (number of iterations without solution improvement). For each instance, the number of duties (Ndt) in the solution, the solution cost in minutes paid are shown. The best solution cost for each instance is highlighted in bold. The LB column shows the value of lower bound for the BDSP computed according to the mathematical model (ILP model) for personnel scheduling into a fixed place (named MPF), following the model proposed by [Bodin et al., 1983]. In this model, information on the spatial availability of the driver is ignored, thus a solution reached by this model is not a feasible BDSP solution.

[4] The instances are available for download by this URL `http://gpea.uem.br/benchmark.html`.

Table 3: Results obtained by the proposed algorithm compared with LB

Instance	GraphBDSP using f_1		GraphBDSP using f_2		GraphBDSP using f_3		LB
	Ndt	Solution cost	Ndt	Solution Cost	Ndt	Solution Cost	
CC130	19	8,451.30	19	8,472.50	19	**8,389.40**	8,057.00
CC251	40	17,667.50	40	17,690.00	40	**17,600.00**	15,655.00
CV412	69	30,810.00	69	30,795.00	66	**29,512.50**	25,427.00
CC512	80	35,612.50	79	**35,105.00**	80	35,312.50	30,858.00
CC761	109	48,395.00	110	48,632.50	107	**47,532.90**	43,010.00
CC1000	152	67,090.00	147	65,019.40	146	**64,873.60**	57,000.00
CC1253	191	84,580.00	188	83,261.10	187	**82,842.90**	72,261.00
CC1517	232	102,729.50	225	**99,852.80**	227	100,507.00	89,191.00
CC2010	297	131,482.50	290	**128,964.20**	292	129,637.80	116,019.00
CC2313	339	150,649.70	331	**147,215.00**	340	150,522.50	131,800.00

Running time was short, being 4min:4s the longest for instance CC2313 using function f_1. The best results were obtained using functions f_2 and f_3.

5.2 Improvement Procedures Analysis

Table 4 indicates how much the initial solution cost can be reduced by using each procedure Imp_1 and Imp_2 (alone and combined) in all cases using function f_3. Columns $Red\%$ mean the percentage of reduction in relation to initial solution cost.

Table 4: Comparison between improvement procedures

Instance	Initial Sol.	Experiment 1		Experiment 2		Experiment 3	
		Imp_1	$Red\%$	Imp_2	$Red\%$	$Imp_1 + Imp_2$	$Red\%$
CV412	33,394.80	**29,512.50**	11.63	30,897.30	7.48	**29,512.50**	11.63
CC1000	74,637.00	65,442.00	12.32	68,550.50	8.15	**64,873.60**	13.08
CC1253	95,622.00	82,955.40	13.25	88,486.90	7.46	**82,842.90**	13.36
CC2313	172,660.40	150,635.00	12.76	158,088.00	8.44	**150,522.50**	12.82

Table 4 also shows that the solutions obtained by using only Imp_1 (experiment 1) were better than those obtained by using only Imp_2 (experiment 2). However, the combination of the two procedures (experiment 3) generated better solutions for instances CC130, CC251, CC412, CC1000, CC1253 and CC2313.

The highest gain by using both Imp_1 and Imp_2 was with the instance CC1000, which obtained an additional reduction of 0.88% in relation to the reduction obtained by using only Imp_1. Although such a gain was small in percentage, it is worth noting that its economic value can be rather significant.

5.3 Results with the Set Covering Problem - SCP

This section presents results for the BDSP modelled as an SCP. For small instances, the optimal value could be obtained by using ILP solver, whereas for large instances, lower bounds were computed using the subgradient method to solve a Lagrangian relaxation problem for SCP according to [Beasley, 1987, Umetani and Yagiura, 2007]. As mentioned at the beginning of this paper, the total number of possible columns in an SCP instance is usually very high, mainly for the largest BDSP instances. Thus, we used a classical heuristic technique to reduce (generate) the number of columns based on a maximum number of PWOs per column ($MaxPC$), limiting the minimum and maximum duration for each PWOs of 60 and 100 minutes, respectively. A similar procedure was also used by [De Leone et al., 2011a]. In this paper the values of $MaxPC$ were estimated taking into account some previous real scheduling provided by the transportation company. Table 5 presents results obtained from BDSP solution using the SCP model. An asterisk (*) before the number indicates that all possible columns have been generated. Values without the asterisk indicate that the instance could have more columns with more $MaxPC$ PWOs, but they were not generated in order to keep the problem in a solvable size.

Table 5: Results obtained with SCP model

Instance	$MaxPC$	Rows	Columns	ZILP	ZLGP
CC130	*3	36	172	9,530.60	9,528.90
CC251	*4	77	1,483	18,691.20	18,587.40
CV412	*4	136	6,730	30,455.00	30,228.10
CC512	*5	162	16,875	37,112.50	36,468.69
CC761	*5	234	47,552	51,752.50	50,242.08
CC1000	*6	315	176,734	67,971.50	65,821.81
CC1253	*6	398	314,149	86,349.00	84,052.23
CC1517	*6	480	579,666	105,178.5	102,315.24
CC2010	4	623	1,206,504	138,035.4	133,055.62
CC2313	4	715	1,610,242	-	149,648.34

Table 5 also shows the number of Rows and Columns obtained from the combination of PWOs. Column ZILP is the cost of the solution obtained by solving of ILP, ZLGP is the Lagrangian lower bound [Beasley, 1987, Umetani and Yagiura, 2007]. To obtain ZILP a running time limit of 24 hours was adopted. Table 5 shows it was possible to obtain the ZILP value for nearly all the instances, except for instance CC2313 due to the high number of columns and the high running time without obtaining the solution (more than 24 hours waiting).

5.4 Comparison of Results

Table 6 shows relative percentage deviation, named GAP. First column is the GAP from the best solution cost from GraphBDSP (Table 3) in relation to LB. Second column is the GAP from the best solution cost from SCP (ZILP or ZLGP value) (Table 5) in relation to LB (Table 3). Third column is the GAP from the best solution cost from GraphBDSP (Table 3) in relation to the best solution cost from SCP (Table 5).

Note that the GraphBDSP GAP to LB are apparently high due to LB quality since the MPF formulation is a fairly relaxed model to BDSP [Bodin et al., 1983], but it is the best known model to get lower bound for BSDP. The GraphBDSP solution costs were kept above the lower bound (LB) by between 4.13% and 16.07%.

Table 6: Comparison GraphBDSP results against LB and SCP results

Instance	GraphBDSP GAP to LB (%)	SCP GAP to LB (%)	GraphBDSP GAP to SCP (%)
CC130	4.13	18.29	-11.97
CC251	12.42	19.39	-5.84
CV412	16.07	19.77	-3.09
CC512	13.76	20.27	-5.41
CC761	10.52	20.33	-8.15
CC1000	13.81	19.25	-4.56
CC1253	14.64	19.50	-4.06
CC1517	11.95	17.93	-5.06
CC2010	11.16	18.98	-6.57
CC2313	11.70	13.54	-1.63

The third column in Table 6 indicates that, for all instances, the solution costs obtained by GraphBDSP are lower than the solution cost obtained by SCP. As explained in Section 5.3, this happens because the SCP instances are heuristically constructed, thus the optimum solution for these instances does not mean optimum solution for the BDSP, even for the Lagrangian lower bound for these instances. Note that, we got the optimum solution for the SCP instances (except instance CC2313).

6 Conclusions

We presented a deterministic 2-phase algorithm, named GraphBDSP, to tackle the bus driver scheduling problem based on Brazilian real instances, from an urban public transportation company. This algorithm produced competitive results comparing with SCP-based approach providing a good result for huge real instances with more than 2,300 tasks within reasonable computing time. To the best of our knowledge this is the largest real instance in the literature and the GraphBDSP represents a new approach applied to the bus driver scheduling problem. The results are compared against to lower bounds computed by mathematical programming. The computational performance of the GraphBDSP was very satisfactory regarding both the solution quality and the running time.

We compared three different cost functions, which the function f_3 based on idle time and overtime presented best results for most cases instead of focusing on paid time only. Although the best solution for the large instance was achieved with the function f_2 (idle time). Anyway, the objective function f_1 focusing on paid time was not the best option.

GraphBDSP is a deterministic algorithm that uses no random operations, i.e., it always finds the same solution for the same input. In addition, it has only one parameter to tune (n_{it}), so it uses an easy parameter tuning. GraphBDSP is quite flexible to changes of rules. The adapt5ations regarding the new rules are needed to compute the cost matrix for each bipartite graph, without changing the model. Thus, we believe it is extensible to other crew scheduling problems.

Our algorithm meets the criteria defined by [Cordeau et al., 2002] for heuristic methods. The simplicity criterion is met because the proposed algorithm requires only one parameter to tune and uses a classical well-known assignment problem, which is easily solved by polynomial running time algorithm. The flexibility criterion is also observed when incorporating new rules. A reasonable accuracy and speed criteria are also observed as shown in section 5.

A fundamental feature of this algorithm is that it can be carried out to solve both the static scheduling problem (tasks do not change throughout the day) and the dynamic scheduling problem (when tasks can be changed due to unexpected events). In other words it is suitable to be used in re-scheduling for an unexpected situation.

Acknowledgments

We would like to thank the anonymous referees for their constructive comments, which led to a clearer presentation of this paper. We would also like to thank CNPq (process 306754/2015-0), CAPES and Fundação Araucária for their financial support.

References

[Beasley, 1987] Beasley, J. E. (1987). An algorithm for set covering problems. *European Journal of Operational Research*, 31:85–93.

[Beasley and Chu, 1996] Beasley, J. E. and Chu, P. C. (1996). A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2):392–404.

[Bodin et al., 1983] Bodin, L., Golden, B., Assad, A., and Ball, M. (1983). Routing and scheduling of vehicles and crews - the state of the art. *Computers and Operations Research*, 10(2):63–212.

[Borndörfers, 2010] Borndörfers, R. (2010). *Aspects of Set Packing, Partitioning, and Covering*. PhD thesis, Technische Universität Berlin.

[Caprara et al., 1999] Caprara, A., Fischetti, M., and Toth, P. (1999). A heuristic method for the set covering problem. *Operations Research*, 47:730–743.

[Carpaneto and Toth, 1987] Carpaneto, G. and Toth, P. (1987). Primal-dual algrorithms for the assignment problem. *Discrete Applied Mathematics*, 18(2):137 – 153.

[CASPT2015, 2015] CASPT2015 (2015). Conference on Advanced Systems in Public Transport. `http://www.caspt.org`.

[Constantino et al., 2014] Constantino, A. A., Landa-Silva, D., de Melo, E. L., de Mendonça, C. F. X., Rizzato, D. B., and Romão, W. (2014). A heuristic algorithm based on multi-assignment procedures for nurse scheduling. *Annals of Operations Research*, 218(1):165–183.

[Cordeau et al., 2002] Cordeau, J. F., Gendreau, M., Laporte, G., Potvin, J. Y., and Semet, F. (2002). A guide to vehicle routing heuristics. *Journal of the Operational Research Society*, 53(5):512–522.

[Daduna and Wren, 1988] Daduna, J. and Wren, A. (1988). *Computer-aided Transit Scheduling: Proceedings of the Fourth International Workshop on Computer-Aided Scheduling of Public Transport*. Lecture Notes in Economics and Mathematical Systems. Springer-Verlag.

[D'Annibale et al., 2007] D'Annibale, G., Leone, R. D., Festa, P., and Marchitto, E. (2007). A new meta-heuristic for the bus driver scheduling problem: GRASP combined with rollout. In *2007 IEEE Symposium on Computational Intelligence in Scheduling, CISched 2007, Honolulu, Hawaii, USA, April 2-4, 2007*, pages 192–197.

[De Groot and Huisman, 2008] De Groot, S. W. and Huisman, D. (2008). *Vehicle and Crew Scheduling: Solving Large Real-World Instances with an Integrated Approach*, pages 43–56. Springer Berlin Heidelberg, Springer Berlin Heidelberg, Berlin, Heidelberg.

[De Leone et al., 2011a] De Leone, R., Festa, P., and Marchitto, E. (2011a). A bus driver scheduling problem: a new mathematical model and a GRASP approximate solution. *J. Heuristics*, 17(4):441–466.

[De Leone et al., 2011b] De Leone, R., Festa, P., and Marchitto, E. (2011b). Solving a bus driver scheduling problem with randomized multistart heuristics. *International Transactions in Operational Research*, 18(6):707–727.

[Ernst et al., 2004] Ernst, A. T., Jiang, H., Krishnamoorthy, M., Owens, B., and Sier, D. (2004). An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research*, 127(1-4):21–144.

[Fores, 2001] Fores, S. (2001). *Column generation approaches to bus driver scheduling.* PhD thesis, School of Computing.

[Huisman et al., 2005] Huisman, D., Freling, R., and Wagelmans, A. P. M. (2005). Multiple-depot integrated vehicle and crew scheduling. *Transportation Science*, 39(4):491–502.

[Laurent and Hao, 2008] Laurent, B. and Hao, J.-K. (2008). Simultaneous vehicle and crew scheduling for extra urban transports. In Nguyen, N., Borzemski, L., Grzech, A., and Ali, M., editors, *New Frontiers in Applied Artificial Intelligence*, volume 5027 of *Lecture Notes in Computer Science*, pages 466–475. Springer Berlin Heidelberg.

[Li et al., 2015] Li, H., Wang, Y., Li, S., and Li, S. (2015). A Column Generation Based Hyper-Heuristic to the Bus Driver Scheduling Problem. *Discrete Dynamics in Nature and Society*, 2015:1–10.

[Li, 2005] Li, J. (2005). A self-adjusting algorithm for driver scheduling. *Journal of Heuristics*, 11(4):351–367.

[Li and Kwan, 2003] Li, J. and Kwan, R. S. (2003). A fuzzy genetic algorithm for driver scheduling. *European Journal of Operational Research*, 147(2):334 – 344.

[Liping Zhao, 2006] Liping Zhao (2006). A heuristic method for analyzing driver scheduling problem. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 36(3):521–531.

[Ma et al., 2016] Ma, J., Ceder, A. A., Yang, Y., Liu, T., and Guan, W. (2016). A case study of Beijing bus crew scheduling: a variable neighborhood-based approach: VNS Algorithm for Bus Crew Scheduling. *Journal of Advanced Transportation*, 50(4):434–445.

[Mauri and Lorena, 2007] Mauri, G. R. and Lorena, L. A. N. (2007). A new hybrid heuristic for driver scheduling. *Int. J. Hybrid Intell. Syst.*, 4(1):39–47.

[Mesquita and Paias, 2008] Mesquita, M. and Paias, A. (2008). Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem. *Computers and Operations Research*, 35(5):1562–1575.

[Ohlsson et al., 2001] Ohlsson, M., Peterson, C., and Söderberg, B. (2001). An efficient mean field approach to the set covering problem. *European Journal of Operational Research*, 133(3):583–595.

[Pentico, 2007] Pentico, D. W. (2007). Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, 176(2):774–793.

[Portugal et al., 2009] Portugal, R., Loureno, H., and Paixo, J. (2009). Driver scheduling problem modelling. *Public Transport*, 1(2):103–120.

[Ralphs, 2015] Ralphs, T. (2015 (accessed 30-May-2015)). CBC MILP. `https://projects.coin-or.org/Cbc`.

[Santos and Mateus, 2007] Santos, A. G. d. and Mateus, G. R. (2007). Hybrid approach to solve a crew scheduling problem: an exact column generation algorithm improved by metaheuristics. In *7th International Conference on Hybrid Intelligent Systems (HIS 2007)*, pages 107–112.

[Shen and Kwan, 2001] Shen, Y. and Kwan, R. S. K. (2001). *Computer-Aided Scheduling of Public Transport*, chapter Tabu Search for Driver Scheduling, pages 121–135. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Shijun Chen, 2013] Shijun Chen, Y. S. (2013). An improved column generation algorithm for crew scheduling problems. *Journal of Information and Computational Science*, 10(1):175–183.

[Silva and Reis, 2014] Silva, G. P. and Reis, A. F. d. S. (2014). A study of different metaheuristics to solve the urban transit crew scheduling problem. *Journal of Transport Literature*, 8(4):227–251.

[Umetani and Yagiura, 2007] Umetani, S. and Yagiura, M. (2007). Relaxation heuristics for the set covering problem. *Journal of the Operations Research Society of Japan*, 50(4):350–375.

[Vera Valdes, 2010] Vera Valdes, V. A. (2010). *Integrating Crew Scheduling and Rostering Problems.* PhD thesis, Universit di Bologna.

[Wren and Rousseau, 1995] Wren, A. and Rousseau, J.-M. (1995). *Bus Driver Scheduling - An Overview*, volume 430 of *Lecture Notes in Economics and Mathematical Systems*, pages 173–187. Springer Berlin Heidelberg.

[Wren and Wren, 1995] Wren, A. and Wren, D. O. (1995). A genetic algorithm for public transport driver scheduling. *Computers and Operations Research*, 22(1):101–110.

[Yunes et al., 2005] Yunes, T. H., Moura, A. V., and De Souza, C. C. (2005). Hybrid column generation approaches for urban transit crew management problems. *Transportation Science*, 39(2):273–288.