Fast Heuristic to Generate New Workdays in a Column Generation Procedure Applied to the Bus Driver Scheduling Problem

Sylvain M. R. Fournier
WPLEX Software Ltda.
Rod SC 401, 8600 Bloco 5, Sala 101 - Santo Antônio de Lisboa, Florianópolis - SC, 88050-000
sylvain@wplex.com.br

Eduardo Otte Hülse
WPLEX Software Ltda.
Rod SC 401, 8600 Bloco 5, Sala 101 - Santo Antônio de Lisboa, Florianópolis - SC, 88050-000
eduardo.hulse@wplex.com.br

Éder Vasco Pinheiro
WPLEX Software Ltda.
Rod SC 401, 8600 Bloco 5, Sala 101 - Santo Antônio de Lisboa, Florianópolis - SC, 88050-000
eder.pinheiro@wplex.com.br

ABSTRACT

Paying driver wages corresponds to about half of the total operational cost of any bus transit company, justifying the need to optimize the drivers' workdays to avoid idle time and overtime. The Bus Driver Scheduling Problem is commonly modeled with a set-partitioning formulation and solved by column generation. This paper describes how columns are generated in this particular Crew Scheduling Problem where several workday types are possible, each one with its own features and user-defined proportion bounds to balance the workday types. A depth-first search method by layers using two lists of labels quickly returns new workday candidates. Tests on big instances (thousands of trips) show final solutions of good quality obtained within reasonnable processing time.

KEYWORDS. Bus Driver Scheduling Problem, Column Generation, Time Constrained Shortest Path

Paper topic: Logistics and Transportation

1. Introduction

In their planning process, bus transit companies seek to meet their passengers' demand at a cost as low as possible. They need to optimize every step of their planning strategy: trip timetabling, bus scheduling, driver scheduling and crew rostering. In particular, scheduling the bus drivers' workdays is a tough challenge, since the drivers are usually subject to several legal constraints due to safety and labor rights. For instance, working 6 hours continuously without any break is forbidden for Brazilian bus drivers.

WPLEX Software provides a complete software suite for bus companies. The application called WPLEX-ON provides bus company planners with a set of useful tools to lighten their planning process. In particular, the users can generate automatically a driver schedule from scratch — up to thousands of trips —, according to their companies' specific constraints. Due to the size of the problem, the bus company planners accept a suboptimal schedule but the software must answer fast (a few hours at most).

The software package models the problem as the widely-studied Bus Driver Scheduling Problem, described in section 2, and solves it using a Branch-and-Price algorithm detailed in section 3. The specific features in the problem tackled in this paper are the driver briefing and debriefing actions and the proportion bounds for workday types. This paper aims to explain how the new workdays are quickly produced at each step of the column generation so that the overall algorithm remains fast. This subproblem is described in section 4, whereas section 5 focuses on the search within the graph and how the subproblem is actually solved using a couple of label lists. Results are presented in section 6 to validate this algorithm and section 7 enumerates some conclusions and draws perspectives for this work.

2. Scheduling bus drivers' workdays

Bus transit operation planners usually organize their drivers workday as a late step in their planning process. The daily trips are sequenced into **blocks** that define the bus fleet schedule. Some buses may have to perform **deadheads** without passengers so that they can be quickly relocated to another route with higher passenger demand and start another trip. For our purposes, a **piece of work** (or simply task) will refer to any driving task (trip or deadhead) in a bus schedule. Note that a piece of work can also be a sequence of trips if no driver change is allowed in between. These pieces of work are used as input data for drivers workdays scheduling.

In the literature, this problem is known as a the **Bus Driver Scheduling Problem** which is a specific case of the widely-studied **Crew Scheduling Problem** (CSP). In the remaining part of this section, we describe some specific features tackled by this paper.

The drivers workdays are subject to both legal rules and needs from the bus company. Drivers may have a time interval in their workday in order to get a **break** or a meal. Several companies have two or three possible **workday types**, each one with its own legal constraints and salary calculation. Table 1 shows some workday examples from bus companies in Brazil.

Table 1: Workday types in most Brazilian bus companies.

Name	Is break paid?	Break duration	Workday duration	Overtime
Standard	yes	15 to 30 min	7h20	up to 2h
Double	no	2 to 6h	8h	up to 2h
Simple	no break		6h	none

Several constraints define each workday type. When a break is required to split the workday into two parts, the working duration before and after the break are constrained to minimum and maximum driving time. The break duration and the global working duration are also subject to lower and upper bounds.

Some workday types may be limited to a certain percentage of all workdays. The short workday type — called "Simple" in Table 1 and mainly used to meet the high quantity of trips during rush hours — can't be applied to too many workdays in the drivers schedule: since they are short, the company would have to hire a lot of drivers to cover them, which is not feasible. Typically, if available, workdays of this type are limited to at most 15% of all workdays. To the best of our knowledge, no other bus driver scheduling paper considers these proportion constraints.

Another original feature in our problem definition is the driver **briefing** and **debriefing** actions. Whenever a bus switches drivers, the leaving driver has to sign some paperwork (debriefing), whereas the arriving driver also checks the vehicle before starting the engine (briefing). Most bus companies choose to consider these actions as part of the workdays.

The daily wage earned by each driver is a complex function of several variables; it contains specific multipliers for overtime and for night time and depends on the vehicle technologies driven along the workday. A workday i can be performed over several vehicles, each of which may be of a specific kind, such as articulated buses, standard buses or minibuses. The cost of the most expensive kind (denoted $C_{\Theta(i)}$) is used in the workday cost calculation. If a driver works both in an articulated bus and a minibus over his workday, even if he drives the articulated bus for a single trip, only the articulated bus cost is taken into account. This vehicle kind cost is multiplied by the workday time, corrected by multipliers for overtime and night time, as follows.

Bus companies compensate the fact that a driver works late (usually after 10:00 PM) by paying him more on this late time through a night time multiplier C_N. This multiplier adds up to the possible overtime factor C_X for overtime the driver may be working at the end of his workday. Usually, $C_X = 1.5$. For a given workday i, we denote by:

- $t_S(i)$ the total day standard time in the workday,
- $t_N(i)$ the total night standard time,
- $t_X(i)$ the total day overtime,
- $t_{NX}(i)$ the total night overtime.

Consider the workday type called "Standard" in table 1, and a workday of this type spanning from 3:00 PM to 11:00 PM. For this workday, each minute after 10:00 PM is considered as night time and each minute after 10:20 PM is overtime. Hence, in this example, the corresponding values in minutes are: $t_S(i) = 420$, $t_N(i) = 20$, $t_X(i) = 0$ and $t_{NX}(i) = 40$.

The workday cost is defined as:

$$c_i = C_{\Theta(i)} \times (t_S(i) + C_N t_N(i) + C_X t_X(i) + C_N C_X t_{NX}(i)) \tag{1}$$

No extensive mathematical formulation for this problem is given in this paper since most constraints are enumerated by Medina and Fournier [2013], especially for workday-related constraints. That paper can be studied for a more comprehensive problem description. The next section describes the algorithm used to solve this particular Bus Scheduling Problem.

3. A Branch-And-Price algorithm

Due to the nonlinear complex costs (described above) and several workday types, a **set-partitioning** formulation was chosen to model the CSP, and a **Branch-and-Price** algorithm to solve it, like Barnhart et al. [1998] and Lübbecke and Desaulniers [2005].

Let T be the set of pieces of work to schedule and K the set of all the available workday types, each of which has a minimum proportion $0 \leq m_k^- \leq 1$ and a maximum proportion $0 \leq m_k^+ \leq 1$. In other words, for each $k \in K$, if Q_k is the actual proportion of workdays of type k among all workdays in the solution, then $m_k^- \leq Q_k \leq m_k^+$ must hold. Obviously, if $m_k^- = 0$ or 1 (and similarly for m_k^+), the master problem can be simplified easily in a preprocessing step.

Let $I(\theta)$ be the set of all workdays currently in the model covering piece of work $\theta \in T$, and I_k be the set of all workdays of type $k \in K$. $I = \cup_{k \in K} I_k$ is the set of all workdays in the model. Note that the family of sets $\{I_k\}_{k \in K}$ is exclusive (a workday can only have a single workday

type), whereas the set family $\{I(\theta)\}_{\theta \in T}$ is not. Binary variable x_i equals 1 if and only if workday $i \in I$ of cost c_i is in the solution.

Our **master problem** can be formulated as follows:

$$\min \sum_{i \in I} c_i x_i \tag{2}$$

$$\sum_{i \in I(\theta)} x_i = 1 \quad , \quad \forall \theta \in T \tag{3}$$

$$0 \leq (1 - m_k^-) \sum_{i \in I_k} x_i - m_k^- \sum_{i \notin I_k} x_i \quad , \quad \forall k \in K \tag{4}$$

$$0 \leq m_k^+ \sum_{i \in I_k} x_i - (1 - m_k^+) \sum_{i \notin I_k} x_i \quad , \quad \forall k \in K \tag{5}$$

$$x_i \in \{0, 1\} \quad , \quad \forall i \in I \tag{6}$$

Constraints (3) and (6) define a well-known set-partitioning problem, while constraints (4) and (5) are specific of the proportion requirements defined above. They can be easily deduced from the definition of proportion Q_k and replacing its value in the minimum and maximum inequalities:

$$\forall k \in K, \begin{cases} Q_k = \dfrac{\sum\limits_{i \in I_k} x_i}{\sum\limits_{i \in I} x_i} = \dfrac{\sum\limits_{i \in I_k} x_i}{\sum\limits_{i \in I_k} x_i + \sum\limits_{i \notin I_k} x_i} \\ m_k^- \leq Q_k \leq m_k^+ \end{cases}$$

With no variable for individual pieces of work, the workdays costs don't need to be decomposed along them. Besides, the workday constraints described in section 2 are not stated in this formulation because they are applied when defining the x_i variables (see section 4.2), each of them refers to a workday that is already known to be valid.

In many cases (Desrochers and Soumis [1989]; Abbink et al. [2007]) a set-covering formulation (with a '$\geq$' sign replacing the equality in constraint (3)) is preferred as its linear relaxation is easier to solve than that of the set-partitioning model. Here, a set-partitioning formulation is better unless the over-covering of some pieces of work — considered as a ride given by a driver to another one — is allowed.

Many columns are generated at each node of the Branch-and-Bound tree (as suggested by Desaulniers et al. [1999]). Section 4 describes how new variables are built and selected for the master problem. At the end of each column generation, if the solution is binary the algorithm stops. Otherwise a new branching is triggered as described by Fournier [2009].

Note that the problem must be solved within a reasonable time even for instances reaching thousands of pieces of work; therefore, the algorithm may be suboptimal, but a good solution must be found in little time. For instance, the column generation process can be stopped if the solution improvement has been too little for a given number of iterations.

The first iteration defines in the master problem only "**tripper**" variables — possibly invalid workdays made of one single piece of work. In later iterations, the linear relaxation of the master problem can yield no solution during the branching step because of under-covered pieces of work. To avoid this situation, the variables always contain a set of trippers covering all pieces of work, beside the variables generated as described in the next section.

4. Generating new workday candidates

Figure 1 illustrates the column generation technique, in which a variable has a good potential to improve the master problem solution if its **reduced cost** c_i^* is negative (Desaulniers et al. [2005]). Let π_θ be the dual cost related to piece of work θ in constraint (3) and π_k^- and π_k^+ the dual

costs related to workday type k in constraints (4) and (5), respectively. Let $T(i)$ be the set of pieces of work covered by workday i and $K(i)$ its type. By definition, variable x_i's reduced cost is:

$$c_i^* = c_i - \left(\sum_{\theta \in T(i)} \pi_\theta + \pi^-_{K(i)} + \pi^+_{K(i)} \right) \tag{7}$$

To find good candidates to be added to the master problem, we generate variables such that the value of $-\left(\sum_{\theta \in T(i)} \pi_\theta + \pi^-_{K(i)} + \pi^+_{K(i)}\right)$ is as low as possible. After computing the newly generated variable cost c_i, we just check whether $c_i^* < 0$ to decide if the variable should be kept.

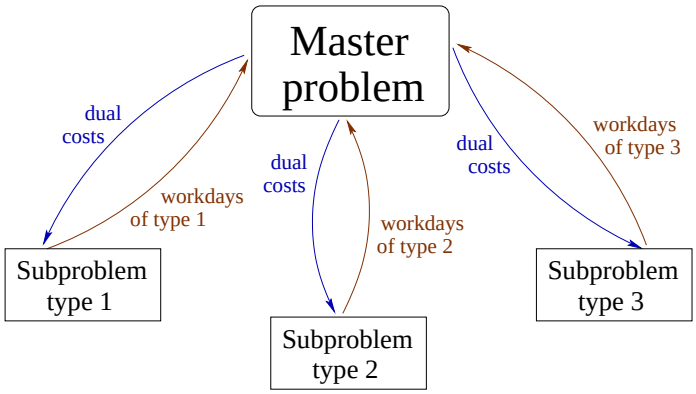

Figure 1: Multi-subproblem column generation: after solving the master problem, the constraints dual costs update the edge costs in every subproblem (downward arrows), which after solving, updates the master problem with new variables of every workday type (upward arrows).

As detailed in section 4.1, for a given workday type, a graph is created using all the π_θ dual costs as edge costs. A constrained minimum cost path is then solved in the graph to generate promising variables.

This method is able to generate variables separately for each workday type. In equality (7) defining the reduced cost, the $\sum_{\theta \in T(i)} \pi_\theta$ component is computed while generating the variable (inside the subproblem), whereas the $\pi^-_{K(i)} + \pi^+_{K(i)}$ term is added afterwards, depending on the workday type in which the variable was generated. Note that the variable cost c_i is computable only after the variable is generated (and not on the fly) since a workday cost is a complex nonlinear function, as stated in section 2.

4.1. Task-based subproblem graph

To generate a new set of workdays of a given type to enter the master problem, we use the technique described by Desrochers et al. [1992]. The same kind of graphs was used for a scheduling problem by Lopes and de Carvalho [2007] and for a cutting stock problem by Alves and de Carvalho [2008]. Here, a graph is created from the bus schedule, with s and t standing for its source and sink nodes, respectively. Figure 2 shows a simple example graph with one vehicle block including both tasks u and v and another block with one single task w.

The graph is built in such a way that any (st)-path (between the source and sink nodes) defines a workday i and that its cost, which is the sum of its edge costs, is exactly $c_l = -\sum_{\theta \in T(i)} \pi_\theta$. Each piece of work $\theta \in T$ is related to four nodes in the graph: its briefing and debriefing nodes (d^-_θ and e^+_θ), its departure node d_θ and end node e_θ.

For any couple of pieces of work u and v in the same block, let $[u, v]$ be the sequence of all pieces of work between u and v (including u and v). By definition, $[u, u] = u$.

For every couple of tasks u and v, we define:

- task arcs between nodes $(d_u e_v)$ in the same block, which means covering the task set $[u, v]$,
- break arcs between two nodes related to non sequential pieces of work u and v $(e_u^+ d_v^-)$, which stands for a break after performing task u and before task v,
- break arcs between two nodes $(e_u d_v)$ such that v follows u in the same block — in this case, the briefing and debriefing don't have to be performed,
- workday sign-on arcs (between s and $d_\theta^-, \forall \theta \in T$) and sign-off (between e_θ^+ and $t, \forall \theta \in T$),
- briefing arcs (between d_θ^- and $d_\theta, \forall \theta \in T$) and debriefing (between e_θ and $e_\theta^+, \forall \theta \in T$).

A duration of a task arc $(d_u e_v)$ is defined as the duration of all its related pieces of work $[u, v]$: it is the difference between the end time of task v and the start time of task u.

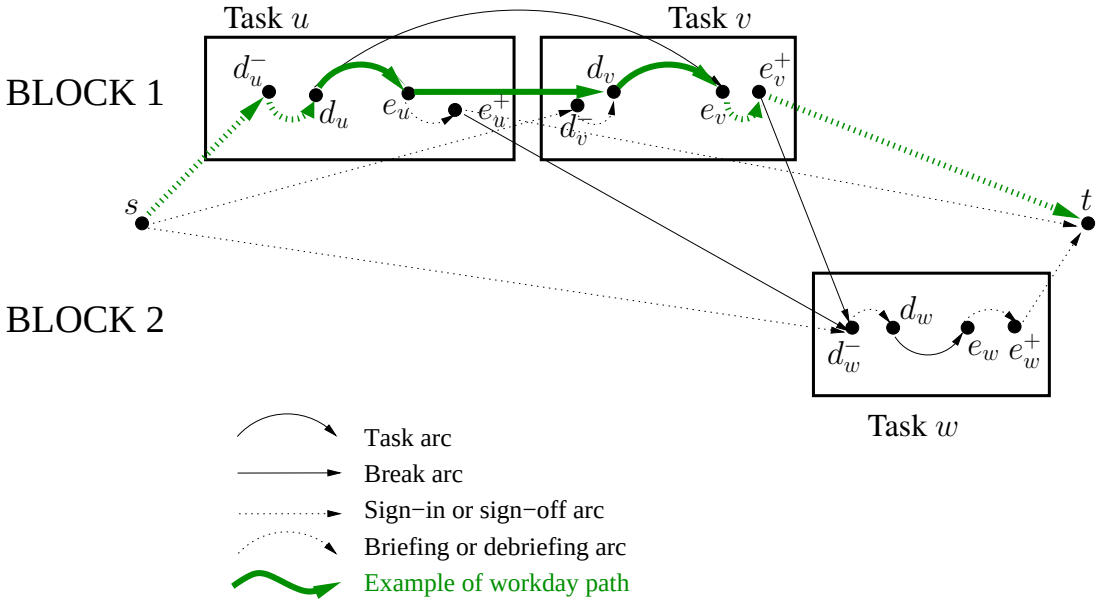

Figure 2: Example of 3-task subproblem graph.

Any (st)-path in this graph defines a driver workday. For example, in Figure 2, the highlighted path $(sd_u^- d_u e_u d_v e_v e_v^+ t)$ defines a workday covering both tasks u and v, and:

- a briefing before task u (arc $(d_u^- d_u)$),
- a break between task u and task v (arc $(e_u d_v)$),
- a debriefing after task v (arc $(e_v e_v^+)$).

Path $(sd_u^- d_u e_v e_v^+ t)$ defines the same workday but with no break between pieces of work u and v.

4.2. Workday type constraints

Medina and Fournier [2013] give an extensive list of workday type constraints, such as:

- minimum and maximum break duration,
- maximum workday extension (difference between workday end time and start time),
- minimum and maximum workday duration (if unpaid, the break duration is ignored),
- minimum and maximum work duration before and after the break.

These constraints are applied in two ways in the workday generating process. Some are considered while defining the graph arcs. For instance, a workday where no break is necessary (as "Simple" in Table 1) will need no break arc in its associated graph. In addition, a task arc shouldn't be defined in the graph if the duration of its related pieces of work exceeds the maximum workday extension. Again, the "Simple" workday type (limited to 3h30) wouldn't define any task arc of duration over 3h30.

Other more complex constraints have to be checked on (st)-paths or even on subpaths. The procedure IsValid scans all the workday type constraints to determine whether a path is valid or could lead to a valid (st)-path. Section 5.1 shows how this procedure is applied to subpaths.

4.3. Arc costs definition

Recall that for any piece of work $\theta \in T$, π_θ is the dual cost of the partitioning constraint (of type (3)) associated with θ, after solving the linear relaxation of the master problem. At each step of the algorithm, while the costs for all non-task arcs are set to 0, the cost of each task arc between pieces of work u and v is defined as:

$$c_{(d_u e_v)} = -\sum_{\theta \in [u,v]} \pi_\theta \tag{8}$$

The cost of the highlighted path in Figure 2 is by definition:

$$c_l = c_{(d_u e_u)} + c_{(d_v e_v)} = -\pi_u - \pi_v \tag{9}$$

Let i be the workday defined by this path and k the workday type associated with the graph (which obviously is also i's type). We can confirm that the value: $c_i + c_l - \pi_k^- - \pi_k^+$ is exactly the workday's reduced cost c_i^* as stated in equality (7).

Solving a Constrained Shortest Path Problem (or CSPP) — known to be NP-complete (Garey and Johnson [1979]) — in such a graph will help find good candidates for negative reduced cost workdays to be added in the master problem. Some studies were performed on how to solve the CSPP exactly, such as Lozano and Medaglia [2013] through a pruning algorithm, Santos and Mateus [2007] using a Genetic Algorithm and Irnich and Desaulniers [2005] with Dynamic Programming.

Nevertheless, as spending time to find the minimum cost path will not necessarily lead to a negative reduced cost (the workday cost c_i — computed afterwards — may be very high), we will focus on returning a low cost path which may not always be of minimum cost. In addition, it is important that several paths are returned in order to raise the odds of generating negative reduced cost workdays. Section 5 describes how the labels are selected and expanded to the neighbor nodes.

5. Solving the subproblem with a depth-first-based graph search

To find low cost paths in the graph defined above, we use a label-based dynamic programming algorithm that will search the graph in depth, so that it can return the path solutions quickly (see section 5.1). For this matter, a heuristic function for the way in which the graph is explored is described in section 5.2.

5.1. Label expansion

A label is a subpath between the source node s and any node in the graph, and is defined by the accumulated cost of all the task edges along the subpath.

Let $l = (sd_u^- d_u)$ be the label chosen at the current step, in the graph defined in Figure 2. Let $k \in K$ be the related workday type. It is then expanded to all its outgoing arcs, namely to $(d_u e_u)$ and $(d_u e_v)$, generating two new labels:

- $l_1 = (sd_u^- d_u e_u)$ of cost $c_{l_1} = c_l + c_{(d_u e_u)}$
- $l_2 = (sd_u^- d_u e_v)$ of cost $c_{l_2} = c_l + c_{(d_u e_v)}$

We define the procedure EXPAND so that $l_1 = \text{EXPAND}(l, e_u)$ and $l_2 = \text{EXPAND}(l, e_v)$.

Each time a new label is created, some workday constraints are checked in order to avoid expanding it uselessly in a later step. See section 4.2 for more details on workday constraints. For example, if the total work duration of all the tasks covered by the label already exceeds the maximum allowed workday duration, the label can be discarded. For this matter, beside its path and its accumulated cost, each label maintains a number of other properties, such as the accumulated work duration: it can usually be easily computed from a label to the next, using the duration of the task edge added to the path.

This validation of the new label before trying to expand it is defined as the ISVALID procedure: if not ISVALID(l, k), label l is discarded and will not be expanded in the next steps. Note that l may still be valid for another workday type. In this case, it could be generated in the

corresponding graph. For example, the label corresponding to the highlighted path in Figure 2 is not valid for the workday type named "Simple" in Table 1 because it contains a break arc. However, it may be valid for the "Standard" workday type.

5.2. Heuristic label selection

First, a zero cost label is created and opened at the source node. At each step, a value is associated to each open label according to a heuristic function defined using the current label cost and all its neighbor edge costs. The least value defines which label should be selected to expand to its neighbors and create new opened labels.

At each step on a given node, all the neighbor edge costs are considered, beyond the current label cost, to compute a heuristic function that will define which label should be selected next. This heuristic function is defined differently for labels before and after a break arc, as follows: let l be the current label from the set P of labels to be compared, n_l the last node of the path defined by label l and $V_k(n_l)$ all outgoing nodes for node n_l. Note that a node's neighborhood depends on workday type k as the graphs may be slightly different for distinct workdays. Label l's cost is denoted by c_l and arc (ij)'s cost is $c_{(ij)}$. The heuristic function is:

$$h(l) = c_l + \begin{cases} \min\limits_{n \in V_k(n_l)} c_{(n_l n)}, & \text{if } l\text{'s path doesn't contain any break arc} \\ \max\limits_{n \in V_k(n_l)} c_{(n_l n)}, & \text{otherwise} \end{cases} \tag{10}$$

The reason why after a break the maximum cost is considered instead of the minimum cost is that the number of arcs leading to a valid workday after a break may be greatly reduced because of the workday constraints (such as the maximum workday duration). So in many cases it may be more accurate to consider the worst case (through the maximum cost) than the best case.

The way it is defined, the heuristic function can overestimate the cost of a (st)-path, as it only considers the direct neighbors, and there may be a low-cost arc further on. Moreover, it considers the maximum neighbor cost after the break, giving an upper bound for the cost value in the next step from this label. Hence this heuristic function is not admissible for an A-star algorithm, which means theoretically it will not necessarily find the minimum cost path. In practice however, the algorithm almost always finds the minimum cost path, particularly because of the breadth parameter described in the next section.

The selection of the next label among the available labels in P through the minimum value of the heuristic function is defined as procedure SELECT, which returns $l^* = \text{argmin}_{l \in P} h(l)$ and removes l^* from P.

5.3. Depth-first search with breadth parameter

In order to find a shortest path quickly, a depth-first search strategy choses the next label to expand according to the least value of the heuristic function described in the previous section. When the sink node reaches a predefined number of associated labels N_A, the search process is stopped and all these labels are returned as the next workday candidates.

However, such a pure depth-first search method would often lead to suboptimal path solutions, especially if the least cost edges are only available in the second part of a workday. That is why two distincts sets of labels are used along the process:

- the usual priority queue P of the next candidates, from which the minimum cost label is selected at each step,
- the new open labels set O, which is the set of recently created labels.

When a label from P is expanded, the new labels generated from its expansion to its neighbors are not selectable at once for the next steps. Instead, they are accumulated in a separate set O until a given number of labels N_B have been selected in P. Then, the whole recent open label set O is added to P and the recently created labels accumulation is restarted in O. This introduces a breadth component in the depth-first search process, as labels are expanded by layers.

Note that the lower the N_B value, the more "depth-aggressive" the strategy is. It becomes a pure depth-first search when choosing $N_B = 1$.

Algorithm 1 is the pseudo-code for the pricing problem while Algorithm 2 shows specifically how the graph minimum cost path is solved for each workday type. The latter introduces calls to three procedures that have already been detailed, namely:

- SELECT is the label selection procedure described in section 5.2,
- EXPAND is the creation of a new label from its father label (see section 5.1),
- ISVALID returns true if the label (defining a partial workday) violates no workday constraint and is also introduced in section 5.1.

Algorithm 1 Solving the pricing problem.

$Z \leftarrow \{\}$ ▷ Set of negative reduced cost variables to be added to the master problem
for each $k \in K$ **do** ▷ For each workday type
  $F \leftarrow$ LOWCOSTPATHS(k) ▷ Solve this specific workday type subproblem
  for each $l \in F$ **do** ▷ For each low-cost label
    Build up workday i related to l and define variable x_i of cost c_i
    $c_i^* \leftarrow c_i + c_l - \left(\pi_k^- + \pi_k^+\right)$ ▷ Compute the variable reduced cost
    if $c_i^* < 0$ **then**
      $Z \leftarrow Z \cup \{x_i\}$ ▷ x_i added to set Z
    end if
  end for
end for
return Z

6. Results

In this section, we tested Algorithm 2 on real life instances from Brazilian bus transit companies. Table 2 describes the instances main features; its last column indicates whether the instance includes at least one proportion constraint (see section 3). The instances are named **I-**n**-**w**-C** after their number of pieces of work n, the number of workday types w and a string C indicating whether proportion constraints are defined and which kind of them. The algorithm was implemented in Java and solved on a 4 × 2.5 GHz workstation with 4 GB of RAM.

Table 2: Instances features.

Instance	# Pieces of work	# Vehicle blocks	# Workday types	Proportion constraints
I-590-3-N	590	116	3	None
I-1508-1-N	1,508	41	1	None
I-3744-1-N	3,744	61	1	None
I-925-4-N	925	139	4	None
I-590-2-N	590	116	2	None
I-590-1-N	590	116	1	None
I-590-3-M	590	116	3	Maximum
I-590-3-Mm	590	116	3	Both
I-925-4-M	925	139	4	Maximum
I-925-4-Mm	925	139	4	Both

Table 3 presents the solutions obtained on the set of instances without workday type proportion constraint. All 3 instances containing 590 pieces of work were solved very fast (less than a minute). On the other hand, although instance I-925-4-N contains fewer pieces or work than I-1508-1-N, the algorithm was slower to solve it. This can be explained by the 4 workday types in

Algorithm 2 Solving a given workday type subproblem.

procedure LOWCOSTPATHS(k) ▷ Returns low cost paths for workday type k subproblem
  $P \leftarrow \{(s)\}$ ▷ Priority queue initialized with a single label (on source node)
  $O \leftarrow \{\}$ ▷ Set of open labels (recently created)
  $F \leftarrow \{\}$ ▷ Set of final labels (at the sink node)
  while ($|F| < N_A$ and $P \cup O \neq \{\}$) **do**
    $P \leftarrow P \cup O$
    $O \leftarrow \{\}$
    $i \leftarrow 0$ ▷ Number of selected labels
    while ($i < N_B$ and $P \neq \{\}$) **do**
      $i \leftarrow i + 1$
      $l^* \leftarrow$ SELECT(P) ▷ Best label selected and removed from P
      for each $n \in V_k(n_{l^*})$ **do** ▷ For each neighbor node
        $l_n \leftarrow$ EXPAND(l^*, n) ▷ New label l_n created from l^* on node n
        if ISVALID(l_n, k) **then** ▷ l_n can't violate any workday constraint of type k
          if $n = t$ **then** ▷ Checks if the current node is the graph sink
            $F \leftarrow F \cup \{l_n\}$ ▷ l_n added to set F
          else
            $O \leftarrow O \cup \{l_n\}$ ▷ l_n added to set O
          end if
        end if
      end for
    end while
  end while
  return F
end procedure

instance I-925-4-N, whereas instance I-1508-1-N only allows to use one workday type and hence contains only one subproblem at each step.

The fourth column ("Cost per workday") gives an insight on the quality of the solution compared to the other instances. The lowest mean cost per workday is obtained for instance I-3744-1-N which is the biggest one in our set. Its 3,744 pieces of work are organized into only 61 blocks, which gave more possible combinations of pieces of work than for instance I-925-4-N which contains fewer than 7 pieces of work per block. As expected, the more we reduce the number of workday types for the same instance (which leaves fewer workday options), the more the solution cost rises (see instances I-590-3-N, I-590-2-N and I-590-1-N).

Table 3: Results for instances without proportion constraints.

Instance	Total cost	Quantity of workdays	Cost per workday	Processing time
I-590-3-N	27,817.60	181	153.69	18s
I-1508-1-N	9,750.50	84	116.08	735s
I-3744-1-N	15,276.16	132	115.73	5,096s
I-925-4-N	39,809.82	221	180.13	2,525s
I-590-2-N	28,568.01	183	156.11	22s
I-590-1-N	31,278.00	254	123.14	58s

Table 4 gathers the performance results for the instances containing workday type proportion constraints. Note that for each instance, the solution cost and the processing time rise as more proportion constraints are defined in the problem. For the solution cost the explanation is straight-

Table 4: Results for instances with proportion constraints.

Instance	Total cost	Quantity of workdays	Cost per workday	Processing time
I-590-3-M	27,985.78	188	148.86	727s
I-590-3-Mm	29,167.09	207	140.90	670s
I-925-4-M	40,319.80	230	175.30	106s
I-925-4-Mm	40,343.06	233	173.15	242s

forward: a more constrained problem will yield a worse solution. We observed that the workday type proportion constraints introduce a kind of degeneracy in the Branch-and-Bound process: when the solution achieves only few fractionary variables, it then gets in a loop in which some variables are made integer by branching, while some others of the same workday type become fractionary in turn. The only exception for this observation is instance I-925-4-N for which the processing time is higher than that of any of its constrained versions.

Interestingly, the mean cost per workday lowers while the number of proportion constraints rises: using fewer workdays automatically heighten overtime, which naturally makes the mean cost per workday higher.

Although we have noticed that in our tests, the solutions obtained by the algorithm were systematically better than any manual solution from the bus transit planners, we feel the need to compare our algorithm against others from the literature. However, as stated previously, no other approach of our knowledge introduces workday types proportion constraints.

7. Conclusion and future work

For several decades, the Crew Scheduling Problem has been tackled successfully through a set-partitioning formulation solved with column generation. In this paper, we defined a specific Bus Driver Scheduling Problem in which the workdays can be of several types — and where the proportion of each one can be bounded — and we described how the pricing problem is solved. At each step and for each workday type, a graph is built using the constraints dual costs, and a minimum-cost path is found in order to generate new negative reduced cost variables for the master problem. To decide which label should be selected at each step, we defined a heuristic function in order to target the graph sink node as fast as possible. The results obtained using this pricing heuristic are promising especially considering the processing time.

To improve this process, the pricing algorithm could become a A-star algorithm if the heuristic function was made admissible. For example, by considering a lower bound on the cost of all possible edges after the current label until the sink node, instead of considering the direct neighbors only. Alternatively, it may also be possible to run a forward-backward shortest path algorithm as described by Wilson and Zwick [2013]. We could also introduce a linear time-depending cost on the graph arcs so that part of the labels cost would be an estimation of the final variable real cost; this would avoid situations where a low cost (st)-path is found but its corresponding variable has a high cost, preventing it from being added to the master problem.

References

Abbink, E., Wout, J. V. and Huisman, D. (2007). Solving Large Scale Crew Scheduling Problems by using Iterative Partitioning. In C. Liebchen, R. K. Ahuja and J. A. Mesa, eds., *ATMOS 2007 - 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems*. Dagstuhl, Germany: Internationales Begegnungs und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. ISBN 978-3-939897-04-0. 4

Alves, C. and de Carvalho, J. M. V. (2008). A Stabilized Branch-and-Price-and-Cut Algorithm for the Multiple Length Cutting Stock Problem. *Computers and Operations Research* **35**, 1315–1328. 5

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

Desaulniers, G., Desrosiers, J. and Solomon, M. M. (1999). Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems. Technical Report G-99-36, Les Cahiers du GERAD. 4

Desaulniers, G., Desrosiers, J. and Solomon, M. M. (2005). *Column Generation*. Boston: Springer. ISBN 9780387254852. 4

Desrochers, M., Gilbert, J., Sauvé, M. and Soumis, F. (1992). CREW-OPT: Subproblem Modeling in a Column Generation Approach to Urban Crew Scheduling. In Desrochers and Rousseau, eds., *Computer-Aided Transit Scheduling*. Springer-Verlag, Berlin, pages 395–406. 5

Desrochers, M. and Soumis, F. (1989). A Column Generation Approach To The Urban Transit Crew Scheduling Problem. *Transportation Science* **23**, 1–13. 4

Fournier, S. (2009). Branch-and-Price Algorithm for a Real-life Bus Crew Scheduling Problem. In L. Buriol, M. Ritt and A. Benavides, eds., *ERPOSul 2009 Anais*. Porto Alegre (RS, Brazil). 4

Garey, M. R. and Johnson, D. S. (1979). *Computer and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co. ISBN 0716710455. 7

Irnich, S. and Desaulniers, G. (2005). *Column Generation*, chapter Shortest Path Problems with Resource Constraints. Boston: Springer. ISBN 9780387254852, pages 33–65. 7

Lopes, M. J. P. and de Carvalho, J. M. V. (2007). A Branch-And-Price Algorithm for Scheduling Parallel Machines With Sequence Dependent Setup Times. *European Journal of Operational Research* **176**, 1508–1527. 5

Lozano, L. and Medaglia, A. L. (2013). On an Exact Method for the Constrained Shortest Path Problem. *Computers and Operations Research* **40**, 378–384. 7

Lübbecke, M. and Desaulniers, G. (2005). *Column Generation*, chapter A Primer in Column Generation. Boston: Springer. ISBN 9780387254852, pages 33–65. 3

Medina, J. and Fournier, S. (2013). Generating Breaks in a Transit Bus Crew Scheduling Problem. In SOBRAPO, ed., *Anais do SBPO 2013*. Rio de Janeiro (RJ, Brazil). 3, 6

Santos, A. G. and Mateus, G. R. (2007). Crew Scheduling Urban Problem: an Exact Column Generation Approach Improved by a Genetic Algorithm. In *2007 IEEE Congress on Evolutionary Computation*. 7

Wilson, D. B. and Zwick, U. (2013). A Forward-Backward Single-Source Shortest Paths Algorithm. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. 11