GENERATING BREAKS IN A TRANSIT BUS CREW SCHEDULING PROBLEM

Juliette Medina
WPLEX Software Ltda.
http://www.wplex.com.br
Rod SC 401, 8600 Corporate Park bloco 5 sala 101
88050-000 Santo Antônio de Lisboa, Florianópolis SC
juliette.medina@wplex.com.br

Sylvain Fournier
WPLEX Software Ltda.
http://www.wplex.com.br
Rod SC 401, 8600 Corporate Park bloco 5 sala 101
88050-000 Santo Antônio de Lisboa, Florianópolis SC
sylvain@wplex.com.br

RESUMO

Apresentamos neste artigo uma abordagem para a geração de multiplos intervalos de descanso em jornadas diárias de trabalho de motoristas de ônibus urbano, de forma a atender a nova lei federal brasileira que regulamenta a jornada de trabalho dos motoristas de veículos rodoviários de carga e de passageiros. Para isso, usamos uma heurística dedicada e gulosa que parte de uma jornada composta por uma sequência de viagens. Além disso o descanso pode ter duração menor que o intervalo existente entre viagens. Mostramos, por fim, que o algoritmo é rápido e capaz de gerar jornadas viáveis e eficientes.

PALAVRAS CHAVES. **Programação de Descansos de Motorista, Programação de Tripulação de Ônibus, Heurística Gulosa, Logística e Transportes.**
Logística e Transportes - L&T

ABSTRACT

In this paper an approach for generating multiple breaks in daily transit bus crew workdays is presented, so as to comply with the new Brazilian federal law that rules the workdays for goods and passengers transportation vehicle drivers. For this matter we use a dedicated and greedy heuristic which entry is a workday made of a sequence of trips. Moreover, a break may have a shorter duration than the existing interval between trips. Finally, we show that this algorithm is fast and able to generate valid and efficient workdays.

KEYWORDS. **Driver Break Scheduling, Transit Bus Crew Scheduling Problem, Greedy Heuristic, Logistics and Transportation.**
Logistics and Transportation - L&T

1. Introduction

For a transit bus company, scheduling the bus drivers' workdays is a difficult challenge. It is necessary to take lots of legal constraints into account for each driver to be able to work under adequate conditions. In particular, in a long duration workday (typically over 6 hours), it is necessary to include a lunch break so that the driver can rest and have his meal. In Brazil, this break can be paid or not and may have a constrained duration. For example, some bus companies set up the double shift workday type for some of their drivers, in which an unpaid break splits the workday into two parts and which duration should usually be over 3 hours.

The Brazilian government has just passed a new labour rule number 12.619/20121[1] to ensure reasonable levels on drivers' working conditions, and to reduce traffic accidents. According to the law, the one-hour meal break can now be split into many shorter breaks. Additionally, a driver has to stop and take a 15-minute break if he has been driving for 4 hours without resting.

In section 2 we briefly describe the context in which this paper takes place. Section 3 defines the break generation problem we face, and we provide details of our solving heuristic in section 4. Some real-life results are given in section 5 and in section 6 we draw some conclusions and perspectives for the work described in this paper.

2. The Transit Bus Crew Scheduling Problem

The Transit Bus Crew Scheduling Problem is a widespread and complex optimization problem faced by transit bus companies. The more trips have to be performed in the company daily schedule, the more benefits are expected out of the use of an automatic tool to generate the drivers' workdays.

WPLEX Software provides an application (WPLEX-ON) to help bus companies to schedule their day-to-day operation. In this piece of software, there are some automatic tools, amongst which two of them produce a set of workdays. The first one generates the set of all workdays from scratch, given the set of trips that have to be performed and the bus schedule for a day of operation. The second one improves a set of existing workdays by swapping several times pieces of work between two chosen workdays. In the next two sections, we decribe the way in which these algorithms include the break generation we deal with in this paper.

2.1. Solving the Crew Scheduling Problem from scratch

In the classical Transit Bus Crew Scheduling Problem (CSP), a set of tasks (or pieces of work), most of which are bus trips, must be performed by drivers, minimizing the crew total cost. Because of the number of workday constraints imposed by the law and sometimes by the transit bus company and the difficult calculation of a single workday cost, we formulate it as a widely used set partionning formulation, as described by Fournier (2009).

[1] http://www.planalto.gov.br/ccivil_03/_Ato2011-2014/2012/Lei/L12619.htm

Let T be the set of tasks to perform. Let $I(t)$ be the set of workdays covering task $t \in T$, and I be the set of all already generated workdays. Every binary variable x_i denotes a possible workday, and equals 1 if and only if workday i is part of the solution, and c_i is this workday cost. The CSP can be formulated as follows:

$$\min \sum_{i \in I} c_i \cdot x_i \tag{1}$$

$$\sum_{i \in I(t)} x_i = 1 \quad , \quad \forall t \in T \tag{2}$$

$$x_i \in \{0, 1\} \quad , \quad \forall i \in I \tag{3}$$

In our Branch-and-Price procedure, we generate every x_i variable from the set of tasks (or pieces of work) that the resulting workday would cover, using a dedicated sub-problem. At each such generation, the algorithm needs to determine:

- if it is possible to **schedule breaks** in the workday idle time (time periods between the tasks),
- if the resulting workday (with such breaks) would be valid regarding the constraints defined in section 3,
- the cost of the resulting workday, if it is valid.

If the new workday is valid and satisfies the condition on its cost defined in the column generation process, it is added to the variable set I for further use in the algorithm.

2.2. Improving the crew using Local Search

WPLEX-ON also allows its user to generate a cheaper crew from an already manually set crew though a sequence of simple local search moves. At each step, two workdays are chosen and in both workdays, a task set is selected for a swap attempt with the other workday. For each such attempt, the following steps are applied:

1. swap a worday task set with the other workday task set, if possible,
2. **redefine the breaks** for both workdays,
3. determine if the new workdays are valid,
4. compute the cost of both resulting workdays.

A *First-improvement* strategy is applied: if the two new workdays are overall cheaper than the former ones, the move is performed and the algorithm goes forward to the next step. Otherwise, another swapping move is tested (with other task sets or workdays) until no more improving move is found.

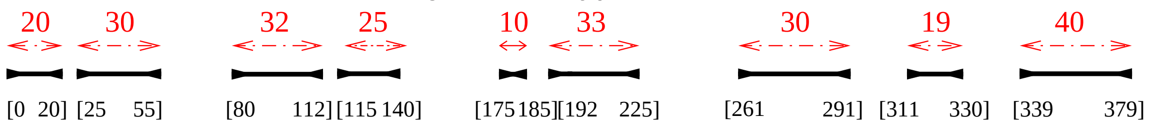
3. Generating breaks in a Transit Bus Crew Scheduling Problem

Meal break scheduling has often been tackled as a subproblem of Shift Scheduling (such as in Rekik et al. (2010) and Musliu et al. (2009)) where each operator has a defined workday time window and breaks must be inserted to cut this time window into several working parts. One of the constraints is the staff requirements per period: a minimum number of employees must be present at any time. In the Crew Scheduling Problem, some authors (such as Chen et al. (2013)) restrict the breaks to a given time period (lunch and dinner time). Our case is similar but here the idle time between each couple of tasks defines several possible time windows where can be defined several breaks. These time windows are different for every workday, as every workday has its own set of tasks to be performed.

Each driver's workday is composed of several pieces of work which can be made of trips, deadheads or other events such as bus preparation and inspection at the beginning of the workday. We define each piece of work i by its time period $[S_i, E_i]$. Therefore, each possible break interval is given by $I_i = [E_i, S_{i+1}]$ where $i \in [1, n-1]$.

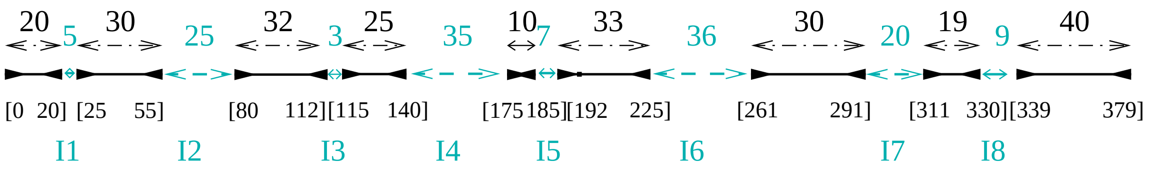
Let's consider an example with a workday made of 9 pieces of work (see figure 1). In our example, the first piece of work is defined by time period $[S_1, E_1] = [0, 20]$.

Figure 1. Workday pieces of work



The breaks intervals are given by (see figure 2):
$I_1 : [E_1 = 20, S_2 = 25]\,, I_2 : [E_2 = 55, S_3 = 80]\,, I_3 : [E_3 = 112, S_4 = 115]\,, \ldots$

Figure 2. Possible break intervals



In addition to these $n-1$ inter-task intervals, a break interval I_n, called "post-workday" break interval, may be defined at the end of the workday (after the n-th piece of work), in case it is impossible to define a valid workday using merely the inter-task intervals. However this should be avoided as it doesn't make sense to consider a break as a resting period at the end of the workday. That is why this break is introduced in the breaks set only if there is no other solution and has an upper bound P_{max} on its end time.

For the i-th such break interval ($I_i, i \in [1, n]$), we denote:

- x_i a binary variable usch that:

$$x_i = \begin{cases} 1 \text{ if a break is defined inside the interval,} \\ 0 \text{ otherwise.} \end{cases} \tag{4}$$

- the related break defined by its start and end dates: $B_i = [\alpha_i, \beta_i] \subseteq I_i = [E_i, S_{i+1}]$.

If an interval I_i is not chosen to contain a break, by convention we set the corresponding break's start time and end time to 0 (see constraint (5)). Constraint (6) states that any break must be included in the corresponding interval:

$$\forall i \in [1, n], \quad \text{if } x_i = 0 \quad \text{then } \alpha_i = \beta_i = 0 \tag{5}$$

$$\forall i \in [1, n-1], \quad \text{if } x_i = 1 \quad \text{then } E_i \leq \alpha_i < \beta_i \leq S_{i+1} \tag{6}$$

Note that in usual Crew Scheduling Problems, it is not clearly stated that the breaks may be shorter than the whole interval between pieces of work (meaning that $B_i = I_i$)

although in real-life schedules, drivers may be idle for some time before and after their breaks. This is nonetheless an important feature in our problem definition.

Without loss of generality, we consider that the workday starts at time 0 and ends at $E_n + (\beta_n - \alpha_n)$, which is the end time of the last piece of work or of the post-workday break, if any. w is the total workday duration for which the driver is paid. It is important to notice that $w < E_n$ in the case of unpaid breaks (as will be underlined in constraint (13)).

N' is the set of indices for the chosen break intervals: $N' = \{i \in [1, n] | x_i = 1\}$. In addition, we introduce the following indices:

- $first = \min N'$ the first interval containing a break in the solution,
- $last = \max N'$ the last interval containing a break,
- $i^+ = \min_{j>i} N'$ the interval in N' just after $i \in N'$,

3.1. Constraints

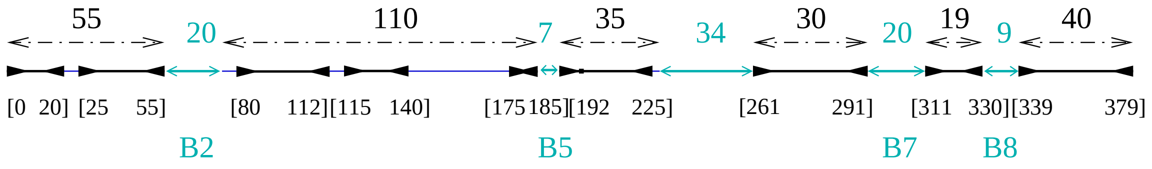
- l_{first} and u_{first} lower and upper bounds on the duration of the first resulting piece of work in the driver's workday:
$$l_{first} \leq \alpha_{first} \leq u_{first} \tag{7}$$
- l_{last} and u_{last} lower and upper bounds on the duration of the last resulting piece of work. In case a post-workday break is necessary to generate a valid workday, this constraint doesn't hold:
$$l_{last} \leq E_n - \beta_{last} \leq u_{last} \tag{8}$$
- l_{inter} and u_{inter} lower and upper bounds on the durations of each intermediate resulting piece of work:
$$\forall i \in [1, n-1],\ \text{if } x_i = 1 \text{ then } l_{inter} \leq \alpha_{i^+} - \beta_i \leq u_{inter} \tag{9}$$
- b_{min} minimum break duration:
$$\forall i \in [1, n], b_{min} \leq \beta_i - \alpha_i \tag{10}$$
- b_{total} required total duration for the break set:
$$\sum_{i=1}^{n} (\beta_i - \alpha_i) = b_{total} \tag{11}$$
- Post-workday break start time and end time (if any):
$$E_n \leq \alpha_n < \beta_n \leq \min(P_{max}; E_n + b_{total}) \tag{12}$$
- w_{min} and w_{max} minimum and maximum workday durations:
$$w_{min} \leq w \leq w_{max} \text{ with } w = \begin{cases} E_n + (\beta_n - \alpha_n) \text{ if the breaks are paid,} \\ E_n - \sum_{i=1}^{n} (\beta_i - \alpha_i) \text{ otherwise.} \end{cases} \tag{13}$$
- n_{max} the maximum number of breaks.
$$\sum_{i=1}^{n} x_i \leq n_{max} \tag{14}$$

Note that if $n_{max} = 0$, the problem is trivial and if $n_{max} = 1$, constraint (9) doesn't hold. In our example, the constants values are defined in table 1. Moreover, we suppose the breaks are unpaid. An example of valid break set is given on figure 3.

Table 1. Symbol definitions and values in the example

Symbol	Definition	Value
$[S_i, E_i]$	[start,end] time for piece of work i	
I_i	break interval between pieces of work i and $i+1$	
n	total number of possible break intervals	9
n_{max}	maximum number of breaks	5
$[l_{first}, u_{first}]$	[lower,upper] bound on the first piece of work duration	[30,60]
$[l_{last}, u_{last}]$	[lower,upper] bound on the last piece of work duration	[30,70]
$[l_{inter}, u_{inter}]$	[lower,upper] bound for each intermediate piece of work	[12,120]
w_{min}/w_{max}	minimum / maximum workday duration	289 / 480
b_{min}/b'_{min}	minimum break duration before / after preprocessing	5 / 5
b_{total}/b'_{total}	total break duration before / after preprocessing	300 / 90
P_{max}	maximum post-workday end time (if any)	390
x_i	binary variable indicating if interval I_i is in the solution	
B_i	break associated with interval I_i	
$[\alpha_i, \beta_i]$	[start,end] time for break B_i	
w	total workday duration	
α_{first}	first break start time	
β_{last}	last break end time	
N'	set of chosen break intervals	

Figure 3. Example of valid break set



3.2. Objectives

The main goal is to generate a break set satisfying the constraints that have just been described. However some secundary objectives can be defined in order to make the driver's workday easier to be performed.

- Minimize the "post-workday" break duration (if any):

$$\min\left(\beta_n - \alpha_n\right) \tag{15}$$

- Minimize the number of breaks: it is better for the drivers to have few long breaks than several short breaks:

$$\min \sum_{i=1}^{n} x_i \tag{16}$$

- Maximize the duration of the longest break:

$$\max\left(\max_{i \in N'}(\beta_i - \alpha_i)\right) \tag{17}$$

4. Dedicated multi-stage heuristic

To solve our problem, we use a dedicated heuristic to generate breaks because we need a fast algorithm which would work as an additional step in a complex Crew Scheduling Problem. Our heuristic is greedy and returns a break set if and only if it results in a valid driver's workday. The general procedure is described in algorithm 1.

Algorithm 1: Generate the workday breaks

```
1  if UnpaidBreaks then
2      UpdateBreakConstraints()
3  end
4  GenerateValidIntervalSet()  see algorithm 2
5  if NOT IsValidIntervalSet then
6      CreatePostWorkdayBreak()
7      GenerateValidIntervalSet()  see algorithm 2
8      ReducePostWorkdayBreakDuration()
9  end
10 RemoveUnnecessaryIntervals()
11 SetUpBreakDurations()
```

Algorithm 2: Generate valid interval set

```
1  forall $i \in [1, n-1]$ do
2      $x_i \leftarrow 1$
3      $\alpha_i \leftarrow E_i$
4      $\beta_i \leftarrow S_{i+1}$
5  end
6  RemoveTooShortIntervals()
7  FindFirstPossibleInterval() see algorithm 3
8  FindLastPossibleInterval()
9  RemoveIntervalUntilValidBreakSet()
```

4.1. Preprocessing step

In a preprocessing step, we update the break time constraints **if the breaks are unpaid**, using the total working time constraints: indeed, in this case, the total workday duration has to take into account the total break duration, as previously stated in constraint (13). If the total break duration is too long, the minimum workday constraint defined by w_{min} (left-hand side of constraint (13)) can be violated.

Similarly, b_{total} is updated using the fact that a too long total break duration can result in a too short workday. Applying the preprocessing update to our example (recall that the breaks are unpaid):

$$b'_{min} = \max\{b_{min}; E_n - w_{max}\} = \max\{5; 379 - 480\} = 5$$

b_{min} is unmodified because the workday duration w was already valid.

$$b'_{total} = \min\{b_{total}; E_n - w_{min}\} = \min\{300; 379 - 289\} = 90$$

In this case, the b_{total} is updated because if the total break duration is over 90, w will be shorter than w_{min}. This new value for b_{total} is now the longest duration for which constraint (13) on w_{min} is satisfied.

4.2. Defining the first and last break interval candidates

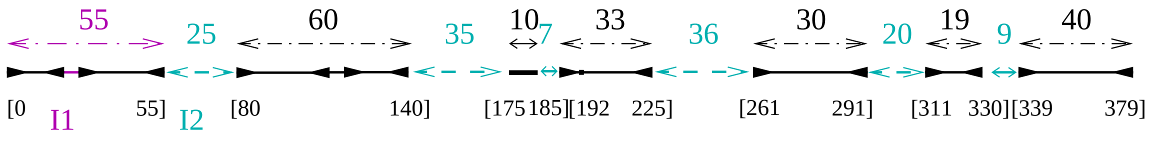
Algorithm 3: Find The First Possible Break

```
1  $i \leftarrow 1$
2  FirstBreakFound $\leftarrow$ FALSE
3  while NOT FirstBreakFound OR $i \neq n - 1$ do
4      if $S_{i+1} - \max\{E_i; l_{first}\} \geq b'_{min}$ then
5          $\alpha_i \leftarrow \max\{E_i; l_{first}\}$
6          $\beta_i \leftarrow S_{i+1}$
7          FirstBreakFound $\leftarrow$ TRUE
8      else
9          $x_i \leftarrow 0$
10     end
11     $i \leftarrow i + 1$
12 end
```

In algorithm 3 we determine the earliest starting date for the first break so that constraint (7) is not violated, as well as the first break interval candidate. We apply the same kind of procedure for constraint (8) dealing with the last break interval.

In our example, where $l_{first} = 30$ and $u_{first} = 60$, I_1 can't be in the solution because $E_1 = 25 \leq l_{first} = 30$. As a consequence, the first break interval candidate is I_2 (see figure 4) and the value for α_2 may be up to $60 = u_{first}$, therefore it is the only possible first interval (meaning necessarily $x_2 = 1$).

Figure 4. After eliminating the impossible first break interval I_1



4.3. Generating a valid break interval set

At this point, we still have to consider the lower and upper bounds on each intermediate piece of work (constraints (9)), and constraints (14) on the maximum number of breaks. Here constraint (11) is relaxed and replaced with a lower bound on the total break time:

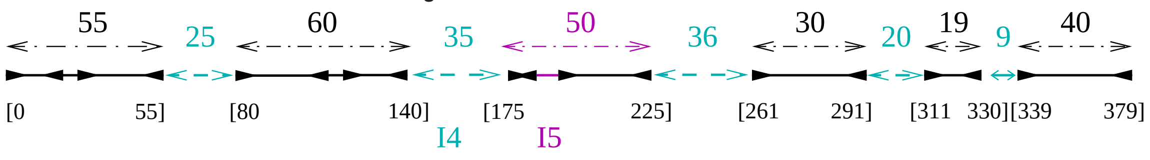
$$\sum_{i=1}^{n} (\beta_i - \alpha_i) \geq b'_{total} \tag{18}$$

Constraint (11) will only be considered again at the last stage of our algorithm (see further section 4.5). We define an interval set as **valid** if it satisfies all the constraints defined in section 3.1 considering this last constraint (18) instead of constraint (11).

We try to discard every interval, starting with the shortest ones, until the interval set becomes valid. In our example, our interval set is not valid: constraint (9) with $l_{inter} = 12$ is not fulfilled beetween breaks intervals I_4 and I_5: $(E_5 - S_5) = 10 < l_{inter}$.

We start removing the shortest break interval which is I_5. Our set is now valid (see figure 5): $(E_6 - S_5) = 43 \geq l_{inter}$ so I_4 and I_6 can be consecutive intervals $(4^+ = 6)$. Note that it may still have been possible to include break [187,192] in the solution. However our algorithm considers it is unnecessary regarding all the constraints.

Figure 5. A valid break interval set

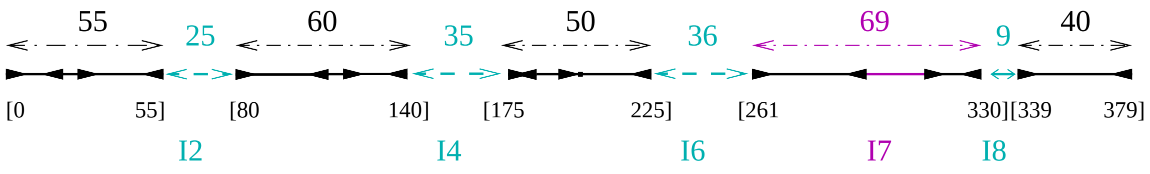


4.4. Removing unnecessary break intervals

We try to remove unnecessary breaks intervals starting from the shortest to the longest (in our example I_8, I_7, I_2, I_4, I_6) in order to keep the longest intervals as far as possible, until the interval set is no longer valid. In our example:

- It is impossible to remove I_8 and set $x_8 = 0$ as constraint (8) on $u_{last} = 60$ would not be fulfilled any more: $(E_9 - S_8) = (379 - 311) = 68 > u_{last}$.
- On the other hand, I_7 can be discarded without violating any constraint (see figure 6).
- As was already stated in section 4.2, I_2 must be kept in the solution.
- I_4 and I_6 also can't be removed because of constraints (9) on $u_{inter} = 120$.

Figure 6. After removing the unnecessary break intervals



4.5. Determining breaks inside the chosen intervals

At this point, the interval set is minimal, which means that no interval can be removed without violating at least one constraint. We now need to reconsider constraint (11) and set values for α_i and $\beta_i, \forall i \in N'$. Every break duration is reduced, starting with the shortest ones. In our example, the total break time is: $\forall i \in N', T = \sum_{i=1}^{n}(\beta_i - \alpha_i) = 105$. Here, the total duration that has to be removed from the breaks is $T_{rem} = T - b'_{total} = 15$ (see figure 7).

- We first reduce the duration of break B_8: it can be shrunk to 5 minutes.
- B_2 can be reduced as well: it will last 14 minutes (see the final solution given on figure 7).

5. Computational results

In this section we aim to show two behaviours. First, our algorithm should be able to run very fast as it must be applied every time a new workday is generated from any set of pieces of work: it must be determined if it is possible to include breaks to make the workday valid. Second, it should give a cheaper solution than a similar algorithm where the breaks are restricted to the whole intervals between pieces of work.

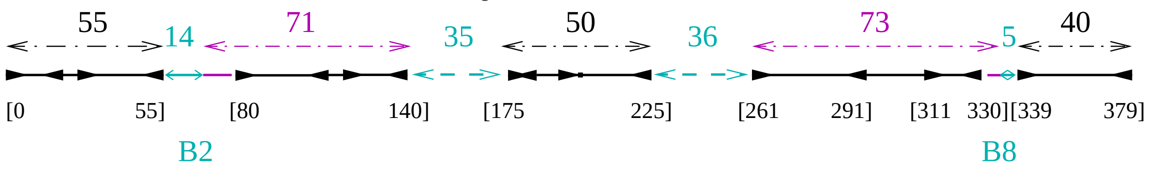
For that matter, we run the whole Crew Scheduling algorithm that includes the break generation for real-life instances and table 2 shows the results we obtained. Our algorithm is written in Java 1.6 and for these tests we used an Intel Core2 Quad CPU Q8400 processor in a 8 GB memory PC.

Table 2. Running instances of the Transit Bus Crew Scheduling Problem

Instance	# Pieces	# Workdays	Cost	(hh:)mm:ss	# Steps	# Break calls
CS1	2570	349	35458	02:46:52	250	32.10^6
CS2	2570	354	36014	03:49:49	515	58.10^6
LS3	2570	343	35458	03:08:24	90	138489
CS4	104	28	15085	00:01	2	916
CS5	104	45	22494	00:01	2	1002
LS6	104	26	14478	00:02	9	822
LS7	206	44	4600	00:17	181	2578
LS8	598	41	4847	00:22	175	8382
LS	206	43	4473	00:07	8	1301
CS12	206	43	4509	06:17	758	3467045
CS9	598	184	17931	07:14	174	2657004
CS9-I	598	183	17996	06:44	222	952702
CS10	206	181	17934	00:17	69	19365
CS10-I	206	182	18048	00:02	4	729
CS11	598	44	4645	40:52	519	17106388
CS11-I	598	41	4741	26:04	739	8169161

The first column is the instance identifier. All the instances with the “CS” prefix are regular Crew Scheduling instances (see section 2.1), whereas instances with the “LS” prefix have been run under a local search approach, where a set of workdays already exists and is improved by simple local search moves (see section 2.2). In both cases, the algorithm has to validate each time it generates a new workday: it tries to insert breaks using the algorithm described in this paper and if such breaks are created successfully, the workday is kept for further steps of the global algorithm. All the instances are real-life customer schedules. Note for example that CS1 and CS2 is the same instance run using different optimization parameters.

Figure 7. Final solution



The second column shows how many pieces of work there are (overall) in the given instance. From the third column on, the information is about solving the Crew Scheduling model:

- the number of workdays in the final solution (third column),
- the final solution cost in Brazilian R$ (fourth column),
- the total processing time in (hours,) minutes and seconds (fifth column),
- the number of steps of the Crew Scheduling algorithm, which is basically every time it reaches a new solution, possibly fractionary (sixth column),
- the number of calls of the Break Scheduling algorithm described in this paper (seventh column).

Note that from columns 2 and 3, the average number of pieces of work per driver can be calculated for each instance to get an insight of the average size of each break generation problem that has to be solved.

It can first be noticed that the break generation algorithm is called a huge number of times, usually over 3000 times per second, which was expected. Note that it can give us an upper bound on the average processing time of our break generation, as solving Crew Scheduling implies a lot of other procedures. It confirms that our break generation is really fast and succeeds in running a lot of times without giving a too bad impact on the overall processing time.

Furthermore, on the three last line groups in table 2, we compare an instance CSa with the same instance CSa-I that is solved using a simple break generator where every break is the whole interval between the pieces of work. In the case of our algorithm, the solution cost obtained is lower. In some cases (such as CS9), our heuristic resulted in more workdays in the final solution than without using the heuristic. In fact, what matters is the total cost, as the other heuristic may create workdays with more extra time (which is more expensive than using more workdays without extra time). Note also that the number of times breaks are generated is higher when our heuristic is applied. This can be simply explained by the fact that fewer workdays are valid in the other more constrained case and more pieces of work combinations to build up a workday are discarded in a preprocessing step.

6. Conclusion and perspectives

In this paper, we describe a very fast break generation algorithm included in a complex Crew Scheduling model. This algorithm is adapted to a new law that has just passed in Brazil and states that bus drivers' breaks may be split into several parts. We showed that it is an improvement to allow the breaks to be strictly included in the interval between the driver's pieces of work, despite the idle time it creates, because this leads to much more possible valid workdays. With our customers' configurations, this algorithm is sufficient to generate satisfying breaks.

However, as it is greedy and focuses on returning few breaks, in few circunstances our algorithm can generate bad solutions or can even fail to find one (removing a key interval prematurely). One example of sub-optimal behaviour would be a case containing three break intervals with the one in between just a little shorter than the other ones. Our algorithm would then discard the interval in the middle although it may lead to a solution where both other break intervals are needed to satisfy the problem constraints, whereas the discarded interval alone would have been sufficient. Nevertheless, in usual customer instances, this particular kind of input hasn't yet been found. This drawback could be partly solved using a post-processing local search procedure to improve the solution, such as Beer et al. (2008). There could also be a kind of backtrack, especially in the case the algorithm fails. As it is mainly a constraint-based model, we could even study a solution using a constraint satisfaction approach such as Wolf (2009) or Curtis et al. (1999) in similar problems.

References

Beer, A., Gartner, J., Musliu, N., Schafhauser, W. and Slany, W. (2008). Scheduling breaks in shift plans for call centers. In *Proceedings of PATAT 2008 - The 7th International Conference on the Practice and Theory of Automated Timetabling*. Montreal (Canada). 12

Chen, S., Shen, Y., Su, X. and Chen, H. (2013). A crew scheduling with chinese meal break rules. *Journal of Transportation Systems Engineering and Information Technology* **13**, 90–95. 3

Curtis, S. D., Smith, B. M. and Wren, A. (1999). Forming bus driver schedules using constraint programming. In BlackPool, ed., *The Pratical Application Company*, volume 239. 12

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). 2

Musliu, N., Schafhauser, W. and Widl, M. (2009). A memetic algorithm for a break scheduling problem. In *The 8th Metaheuristic International Conference (MIC 2009)*. Hamburg (Germany). 3

Rekik, M., Cordeau, J.-F. and Soumis, F. (2010). Implicit shift scheduling with multiple breaks and work stretch duration restrictions. *Journal of Scheduling* **13**, 49–75. 3

Wolf, A. (2009). Constraint-based task scheduling with sequence dependent setup times, time windows and breaks. *GI Jahrestagung 2009* , 3205–3219. 12