Transportation Research Part B 112 (2018) 216–229

Contents lists available at ScienceDirect

Transportation Research Part B

journal homepage: www.elsevier.com/locate/trb

Vehicle and Crew Scheduling for Flexible Bus Transportation Systems

Vincent Boyer[a,*], Omar J. Ibarra-Rojas[b], Yasmín Á. Ríos-Solís[a]

[a] Systems Engineering Department, Universidad Autónoma de Nuevo León, Mexico
[b] Research Center of Physics and Mathematics, Universidad Autónoma de Nuevo León, Mexico

ARTICLE INFO

Article history:
Received 10 February 2017
Revised 20 April 2018
Accepted 23 April 2018
Available online 1 May 2018

2010 MSC:
00-01
99-00

Keywords:
Vehicle Scheduling Problem
Crew Scheduling Problem
Flexible Bus Transportation System
Mixed-Integer Problem
Variable Neighborhood Search Algorithm

ABSTRACT

This article deals with the Flexible Vehicle and Crew Scheduling Problem faced by urban bus transport agencies that have to assign their resources (vehicles and drivers) to cover timetables generated at the tactical level. We aim for high quality and fast to compute solutions for this problem, considering vehicle characteristics, driver qualifications requirements for each line, and labor regulations, that is, drivers have a limited duty length, mandatory rests, a restricted consecutive driving time, and a limited extra working hours. Moreover, the starting time of the drivers shift is not fixed a priori and the breaks can be allocated anywhere in the schedule as long as labor regulations are satisfied. Thus, flexibility is required to compute drivers duty but it is also needed in scenarios where the available number of drivers and vehicles changes almost everyday. We propose a mixed-integer linear programming model and a variable neighborhood search for this problem and show the efficiency of our approaches with a large set of instances.

1. Introduction

At the operational level of the transport planning process, urban bus agencies have to assign their main resources (vehicles and drivers) to cover timetables (TT) generated at the tactical level. This assignment is decomposed into two subproblems known as the Vehicle Scheduling Problem (VSP) and the Crew Scheduling Problem (CSP). Generally, the objective is to minimize the operational costs in terms of the vehicles usage and drivers wage, while operational constraints for vehicles and labor regulations for drivers are satisfied. In particular, the VSP defines the set of trips that must be covered by each vehicle and the CSP defines the daily duties (in terms of blocks of trips) that must be assigned to a specific driver. The VSP and CSP (also known as Driver Scheduling Problem in the context of bus transit systems) are both NP-hard problems.[1] A sequential approach for solving these problems leads to suboptimal solutions for the global planning process of transit systems so integrated approaches have been considered in the literature (Borndörfer et al., 2008; Freling et al., 2003; 2001; Steinzen et al., 2007).

In developed countries, the VSP and the CSP are generally solved every six months or when a new trip is added to the timetable. Nevertheless, in many developing countries, the setting becomes more complex since the VSP and the CSP decisions have to be made every week (or even every day) since the fleet of vehicles and the available drivers drastically

* Corresponding author.
E-mail addresses: vincent.boyer@uanl.edu.mx (V. Boyer), omar.ibarrarj@uanl.edu.mx (O.J. Ibarra-Rojas), yasmin.riossls@uanl.edu.mx (Y.Á. Ríos-Solís).

[1] The VSP with a single depot and homogeneous fleet is polynomial.

https://doi.org/10.1016/j.trb.2018.04.008
0191-2615/

Trips	1	2	3	4	5	6	7	8	9	10	11	12	13
TT	6:00	7:03	8:03	8:33	9:03	9:33	10:05	10:35	11:07	11:40	12:40	13:40	14:35
VSP	v_1	v_6	v_1	v_3	v_1	v_2	v_3	v_1	v_3	v_1	v_1	v_5	v_4
CSP	d_1	d_6	d_1	d_3	d_1	d_2	d_3	d_1	d_3	d_1	d_1	d_5	d_4

Trips	1	2	3	4	5	6	7	8	9	10	11	12
TT	6:01	6:35	7:01	8:01	8:42	9:42	10:43	11:20	11:58	12:40	13:58	14:58
VSP	v_2	v_4	v_2	v_2	v_5	v_5	v_2	v_5	v_2	v_3	v_2	v_2
CSP	d_2	d_4	d_2	d_2	d_5	d_5	d_2	d_5	d_2	d_3	d_2	d_2

Trips	1	2	3	4	5	6	7	8	9	10	11	12	13
TT	6:10	7:10	7:35	8:35	9:14	10:05	10:14	11:14	11:35	12:35	13:35	14:10	15:10
VSP	v_3	v_3	v_4	v_4	v_6	v_4	v_6	v_6	v_4	v_4	v_4	v_3	v_3
CSP	d_3	d_3	d_4	d_4	d_6	d_4	d_6	d_6	d_4	d_4	d_4	d_3	d_3

Fig. 1. Feasible solution for three lines: blue, green, and orange lines. For each table, the first line corresponds to the label of the trips, the second one (TT) is the timetable of each trip, the third line (VSP) is the vehicle scheduling linking a trip with a vehicle, and the last line (CSP) is the driver scheduling linking a driver to a vehicle and a trip. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

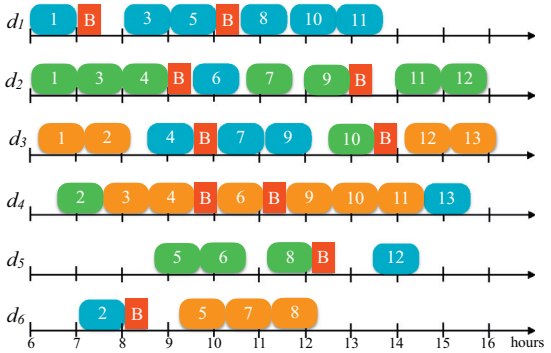

Fig. 2. Schedules of the drivers $d_1, \ldots, d_6$. Each blue, green or orange rectangle is a trip; red rectangles are the legal breaks that the drivers must have to avoid working blocks of more than four hours. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

change over time. In our case study, in Monterrey, Mexico, up to 10% of the vehicle fleet may have an incident every day: breakdowns, maintenance, crashes, or burglaries. Moreover, the number of available drivers also varies from day to day: low salaries generate rotation, drivers are usually hired per week, and other cultural reasons (e.g., after payday up to 30% of the drivers will either not show up or not pass the alcohol breathalyzer). We have also observed this behavior in other Latino-American cities. Hence, we aim to solve the single depot VSP and the CSP in this *flexible* context in an integral way with a fast algorithm that yields high-quality solutions.

This *flexible transportation system* has some advantages for scheduling the vehicles and the drivers to the fixed trips. The duty of the drivers may change every week, and the driver breaks can be placed anywhere during the day as long as the legal restrictions are satisfied. To illustrate the gain obtained by considering this flexible context we introduce an example. Fig. 1 shows three tables corresponding to a feasible solution for three lines: blue, green, and orange line. For each table, the first row corresponds to the label of the trips, the second one (TT) is the timetable of each trip, the third row (VSP) is the vehicle scheduling linking a trip with a vehicle, and the last row (CSP) is the driver scheduling linking a driver to a vehicle and a trip. This solution uses six drivers and six vehicles. A driver cannot work more than 10 hours (eight normal hours plus two extra hours), and by legal restrictions, a driver cannot drive more than four hours in a row without a break of at least 30 min.

Fig. 2 shows the schedules of the drivers $d_1, \ldots, d_6$. In the horizontal axis, we have the hours. Each blue, green or orange rectangle is a trip (for this example, we consider that round times of the trips are all equal to one hour). Red rectangles are the legal breaks of the drivers separating driving blocks of less than four hours. Note that the duty of a driver may not start

and end at a specific time slot, that is, for instance, the duty of driver d_4 starts at 6:35 and ends at 15:35. If between two trips the driver does not rest for at least 30 minutes then this small break is not considered as a legal break and it must be added to the consecutive driving time (see, for instance, the idle time between trips 6 and 8 of the green line for driver d_5). From the solution of Fig. 2, we can also notice that the break for each driver is at a different time. In particular, driver d_4 performs three trips, followed by a break because he could not make another trip without exceeding the four hours time limit. Then he only makes trip 6 of the orange line and rests again to be able to follow with a block of four trips in a row.

Therefore, we do not have generic duties that can be assigned to any driver, they must be designed together with the vehicle assignment to minimize the number of drivers needed in the solution. Additionally, we consider *compatibility* between each pair: vehicle-driver, driver-line, and line-vehicle. Indeed, not all the drivers are allowed to operate the new buses of the fleet, a driver may not know the route of a line, or a vehicle may have its line number painted so it cannot be used for another line (in Fig. 2, driver d_1 is only allowed to operate on the blue line). We also do not limit the number of vehicle swaps that a driver may have along his daily schedule. To the best of our knowledge, this is the first time that these characteristics are considered all together by an integrated approach for VSP and CSP. We name this problem as the Flexible Vehicle Crew Scheduling (FVCS) problem.

In this context, the FVCS problem consists in assigning a set of trips of circular transit lines (at one depot) to a vehicle and a driver, considering the following constraints: (i) the compatibility of the pairs trip-vehicle, trip-driver, and vehicle-driver for the different lines; and (ii) labor regulations, that is, drivers have a limited duty length, mandatory rests, restricted consecutive driving time, and limited extra working hours.

Therefore, we need to identify each driver and each vehicle in order to define a feasible trip-vehicle-driver assignment for each line.

It is neither common nor trivial to consider trip-vehicle, trip-driver, and vehicle-driver assignments, together with the classic scheduling decisions. Moreover, since we do not consider a fixed duty structures, the starting time of the shifts and the breaks allocation are also part of the decision. Then, our major contributions are the following: (i) the definition of a new integration of the VSP and the CSP, named the FVCS problem; (ii) a network flow formulation for the proposed problem; and (iii) a Variable Neighborhood Search metaheuristic able to solve large instances of the FVCS problem.

The structure of the paper is as follows. Section 2 is dedicated to the literature review. The problem description and the mathematical formulation of the FVCS problem are presented in Section 3. Section 4 describes the proposed Variable Neighborhood Search procedure for the solution of the FVCS problem. Computational results are shown in the Section 5 with instances randomly generated from the real data of a Mexican urban bus transport agency. Finally, conclusions and future work are presented in Section 6.

2. Related literature

There are two types of integrated approaches (see a review of Transit Network Planning in Ibarra-Rojas et al., 2015): (i) partial integrations and sequential solution procedures; and (ii) complete integrations were decisions of the integrated problems must be taken simultaneously. Obviously, complete integrated approaches lead to better results since they consider all the degrees of freedom but they are more difficult to define and to solve. In this work, we present a complete integration of the VSP and CSP.

Some complete integrated approaches assume the existence of an homogeneous fleet and generic drivers. This assumption implies that the vehicles have the same characteristics and can be assigned to the same set of trips, and drivers are alike and can be assigned to any feasible duty without considering their qualifications. Then, VSP and CSP can be formulated as a network flow problem and a set covering/partition problem, respectively (see a review of formulations in Freling et al., 2003). Moreover, a complete integrated formulation for VSP and CSP can also be defined through set covering/partition formulations which are commonly solved by column generation approaches and Lagrangian Relaxations techniques (e.g., Haase et al., 2001; Borndörfer et al., 2002; Freling et al., 2001; 2003).

If heterogeneous fleets are considered, the Vehicle Scheduling Problem must identify the type of vehicle required to cover each trip. However, vehicles are commonly considered alike within each type of fleet. This problem is a particular case of the Multi-Depot Vehicle Scheduling Problem (MDVSP) where one depot has a single fleet type and it can be formulated as network flow problem. The MDVSP has been integrated with the CSP by Gaffi and Nonato (1999). The authors propose a formulation based on a quasi-assignment model for the MDVSP and a set of linking constraints that ensure compatibility of the driver schedule. They assume that a driver must be assigned to the same vehicle for the whole planning period and each duty trip starts and ends at the same depot. A column generation approach allows to solve instances of up to 257 trips and 28 depots using a computational time between two and six hours on average (but it exceeds 24 hours for the largest instances).

de Groot and Huisman (2004) propose to divide the MDVSP according to the following rules: i) splitting the problem into several instances of VSP and CSP, that is, assigning each trip to a depot (or fleet type); and ii) splitting an instance into a predetermined number of smaller ones based on the number of trips and trip-vehicle assignments. Numerical results suggest that the effect of dividing these instances do not significantly deteriorate the quality of the solutions while the proposed approach outperforms the classical sequential method.

Column generation algorithms have been also implemented for integrated approaches considering flexible timetables (e.g., Gintner et al., 2008; Kéri and Haase, 2007a; 2007b). However, the intractability of large instances is recognized and

metaheuristic approaches are proposed to obtain high-quality solutions in a small amount of computational time. For example, Steinzen et al. (2007) develop an Evolutionary Algorithm for the integration of MDVSP and CSP consisting of: (i) generating potential population members by exploring trip-depot assignments; and (ii) determining the vehicle and driver schedules through a Lagrangian relaxation combined with a column generation approach. The proposed solution method is tested on instances of up to 200 trips and four depots. Numerical results show reductions between 5.1% and 10% of the number of duties compared to the current operation of the system.

Finally, Leone et al. (2011) integrate the MDVSP and the CSP considering several hard constraints, such as limited spread-over, a limited number of breaks, upper bounds for idle time, and limited working time. Due to the large number of constraints in the proposed mathematical formulation, small instances are solved with a commercial solver, and a GRASP metaheuristic is used to obtain feasible solution for instances of up to 400 trips.

As we previously mentioned, vehicles are assumed alike within each fleet type and the Crew Scheduling Problem defines generic duties which are later assigned (in the Crew Rostering problem) to the drivers. Exceptions for this assumption are the following two approaches. First, the study of Vaidyanathan et al. (2007) that integrates the MDVSP and the CSP considering a FIFO rule for drivers at relief points, that is, if a driver d arrives at a relief point before another driver d', driver d must cover his next trip before d' does.

The authors propose a network flow formulation where possible starting times for the rests are part of the input (based on a set of duty structures). Decomposition and relaxation techniques are implemented to solve the proposed problem. Second, the study of Laurent and Hao (2008) integrates the VSP and the CSP considering a trip-vehicle compatibility. Therefore, an optimal trip-vehicle assignment must be achieved besides the scheduling decisions by considering work regulation constraints: maximum spread time, minimum working time, and driver changeovers (but does not model driver's break times). The problem is formulated through constraint programming and a GRASP metaheuristic is implemented to solve instances up to 249 trips, 50 drivers, and 47 vehicles.

3. The Flexible Vehicle Crew Scheduling Problem

To define our problem, we assume that a bus company has to cover a set L of lines. Each line $l \in L$ is composed of a set of trips $Trips(l)$ ($|Trips(l)| = numTrips(l)$) and each trip $i \in Trips(l)$ have a fixed schedule $sched_i$. Let $Trips = \cup_{l \in L} Trips(l)$ be the set with all the trips of all the lines. The round time of each trip $i \in Trips$ is $roundTime_i$. The bus company has an heterogeneous fleet V, thus not all vehicles can be assigned to any trip. Indeed, some vehicles have the number of the line painted, some others have electronic panels, new buses may be assigned to lines close to downtown and old ones to the suburbs. We denote by $V(i)$ the set of vehicles that can be assigned to trip $i \in Trips$. The aim of the FVCS problem is to minimize the total operational cost based on vehicles usage and drivers wage. Indeed, if a vehicle is not needed for the day, it can go to preventive maintenance. So a fix cost *vehicleCost* for using a vehicle is imposed.

The set of drivers D may also have a compatibility constraint with the trips of the lines. Let $D(i)$ be the set of drivers that can perform the trip $i \in Trips$. Moreover, the vehicle $v \in V$ can be assigned to only a subset $D(v)$ of drivers. Analogously, let $V(d)$ be the set of vehicles that can be operated by driver $d \in D$. Each driver has a regular work day of *regularDay* hours with a pay of *payDay*. The work day can be extended by *extraHours* hours with an additional pay of *extraPay*. In our case study, a driver works for 8 h and can be hired for doing two more extra hours if needed. Note that the drivers are paid the whole day even if they have worked less than 8 h. The same policy applies to the extra hours. Labor regulations require that a break of *restTime* must be given to a driver that has been driving for *maxTimeNoRest* of hours. In our case study, the drivers need a break of at least half an hour if they have been driving for four hours in a row. A driver may have had a 10 minutes break between a pair of trips but as the duration of this break is less than *restTime*, it counts as a nonstop duty.

In network flows terms, let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a graph with vertices $\mathcal{V} = \{o\} \cup Trips$ where o denotes the depot, and $\mathcal{A} = \{(i, j) \in Trips \times Trips : sched_i + roundTime_i \leq sched_j\}$ corresponds to the arcs. To model the FVCS problem, we introduce the following binary variables.

$$z^d = \begin{cases} 1, & \text{if driver } d \in D \text{ is on duty during the day,} \\ 0, & \text{otherwise.} \end{cases}$$

$$y^d = \begin{cases} 1, & \text{if driver } d \in D \text{ makes extra hours during the day,} \\ 0, & \text{otherwise.} \end{cases}$$

$$r^d_j = \begin{cases} 1, & \text{if a legal break can be placed before covering trip } j, \text{ for driver } d \in D(j) \text{ and } j \in Trips, \\ 0, & \text{otherwise.} \end{cases}$$

$$w^d_{ij} = \begin{cases} 1, & \text{if driver } d \in D \text{ makes trip } i \text{ just before trip } j, \text{ for } (i, j) \in \mathcal{A}, \\ 0, & \text{otherwise.} \end{cases}$$

$$s^v_{ij} = \begin{cases} 1, & \text{if vehicle } v \in D \text{ makes trip } i \text{ just before trip } j, \text{ for } (i, j) \in \mathcal{A}, \\ 0, & \text{otherwise.} \end{cases}$$

We highlight that variable $w^d_{oj} = 1$ represents the case where trip $j \in Trips$ is the first trip of driver $d \in D(j)$ while $w^d_{io} = 1$ represents the case where trip $i \in Trips$ is the last trip of driver $d \in D(i)$ (likewise for the variables s^v_{io} and s^v_{oj}).

Several auxiliary real variables for the FVCS are also introduced to model the labor constraints. Let x^d be the duty length for driver $d \in D$, h_j^d the idle time of driver $d \in D(j)$ before covering trip $j \in Trips$, and variable p_j^d the consecutive working time of driver d without a legal break before covering trip j.

The objective function (1) of the FVCS is to minimize the cost of the drivers (including the extra hours), plus the cost of the vehicles used during the day:

$$\min \quad \sum_{d \in D} \left(payDay \cdot z^d + extraPay \cdot y^d \right) + vehicleCost \sum_{v \in V} \sum_{j \in Trips : v \in V(j)} s_{oj}^v. \tag{1}$$

The constraints related to the workday of every driver $d \in D$ is defined as follows:

$$x^d = \sum_{j \in Trips : d \in D(j)} \left(h_j^d + roundTime_j \sum_{i:(i,j) \in \mathcal{A}} w_{ij}^d \right), \tag{2}$$

$$x^d \leq regularDay \cdot z^d + extraHours \cdot y^d, \tag{3}$$

$$y^d \leq z^d. \tag{4}$$

Constraints (2) state that the duty length for driver $d \in D$ corresponds to the idle time plus the driving time. Constraints (3) guarantee that the total time that a driver can be on duty may not be greater than the regular day length plus the extra hours. Constraints (4) assure that the drivers may work extra hours only if they have worked a regular day.

The next set of constraints are the flow balancing constraints for the network $\mathcal{G}$. Constraints (5) imply that if a trip j is done by vehicle v, then there must be another trip that follows j with this vehicle. Constraints (6) are similar to the previous ones but in this case, we consider the trips and the drivers. Each trip must be covered exactly once by a vehicle and driver as indicated in Constraints (7) and (8). Moreover, Constraints (9) guarantee that if a vehicle v covers trip j, a driver d compatible with v must cover that trip. Finally, only one trip can be the first one for a driver and, similarly, only one trip can be the first one for a vehicle as expressed in Constraints (10) and (11), respectively.

$$s_{oj}^v + \sum_{i:(i,j) \in \mathcal{A}} s_{ij}^v = s_{jo}^v + \sum_{k:(j,k) \in \mathcal{A}} s_{jk}^v, \quad j \in Trips, \quad v \in V(j), \tag{5}$$

$$w_{oj}^d + \sum_{i:(i,j) \in \mathcal{A}} w_{ij}^d = w_{jo}^d + \sum_{k:(j,k) \in \mathcal{A}} w_{jk}^d, \quad j \in Trips, \quad d \in D(j), \tag{6}$$

$$\sum_{v \in V(j)} \left(s_{oj}^v + \sum_{i:(i,j) \in \mathcal{A}} s_{ij}^v \right) = 1, \quad j \in Trips, \tag{7}$$

$$\sum_{d \in D(j)} \left(w_{oj}^d + \sum_{i:(i,j) \in \mathcal{A}} w_{ij}^d \right) = 1, \quad j \in Trips, \tag{8}$$

$$\sum_{i:(i,j) \in \mathcal{A}} s_{ij}^v \leq \sum_{d \in D(j) \cap D(v)} \left(w_{oj}^d + \sum_{i:(i,j) \in \mathcal{A}} w_{ij}^d \right), \quad j \in Trips, v \in V(j), \tag{9}$$

$$\sum_{j \in Trips : v \in V(j)} s_{oj}^v \leq 1, \quad v \in V, \tag{10}$$

$$\sum_{j \in Trips : d \in D(j)} w_{oj}^d \leq 1, \quad d \in D. \tag{11}$$

Finally, we introduce the constraints on the drivers duty, and in particular, the legal duration limit on the consecutive working time. Constraints (12) defines the auxiliary variables h_j^d that represent the idle time of driver d before covering trip j. Constraints (13) and (14) activate the variables r_j^d if a legal break can be placed before trip j for driver d, where M is a big number that can be fixed to $regularDay + extraHours - restTime - \min_{j \in Trips}\{roundTrip_j\}$. Constraints (15) combined with the objective function define the consecutive working time of driver d (with no rest) before covering trip j as $p_j^d = p_i^d + sched_j - sched_i$ if $w_{ij}^d = 1$ and $r_j^d = 0$. Constraints (16) set $p_j^d = 0$ when driver d can take a rest before covering trip j, that is, $r_j^d = 1$. M_2 is a big number that can be fixed to $maxTimeNoRest - \min_{j \in Trips}\{roundTrip_j\}$. Constraints (17) guarantee

that the consecutive working time without rest for any trip does not exceed the *maxTimeNoRest* time. Note that Constraints (15) are quadratic but with a classic linearization technique (see Williams, 2013) the product $w_{ij}^d p_i^d$ can be replaced by an auxiliary variable in order to have a linear formulation. The detailed linearization procedure of Constraints (15) is shown in the Appendix A.

$$h_j^d = \sum_{i:(i,j)\in\mathcal{A}} w_{ij}^d \cdot (sched_j - sched_i - roundTime_i),\ d \in D(j),\ \ j \in Trips \tag{12}$$

$$h_j^d - restTime < M \cdot r_j^d,\ d \in D(j),\ \ j \in Trips \tag{13}$$

$$restTime - h_j^d \leq M \cdot \left(1 - r_j^d\right),\ d \in D(j),\ \ j \in Trips \tag{14}$$

$$M_2 \cdot r_j^d + p_j^d \geq \sum_{i:(i,j)\in\mathcal{A}} w_{ij}^d \cdot \left(p_i^d + sched_j - sched_i\right),\ d \in D(j), j \in Trips \tag{15}$$

$$p_j^d \leq M_2 \cdot (1 - r_j^d),\ d \in D(j),\ \ j \in I \tag{16}$$

$$p_j^d + roundTrip_j \leq maxTimeNoRest,\ d \in D(j),\ \ j \in I. \tag{17}$$

The proposed mixed integer formulation for the FVCS problem is then:

$$\begin{aligned}
\min \quad & (1)\\
\text{s.t.} \quad & (2)-(14)\\
& \text{linearization of } (15)\\
& (16), (17)\\
& z^d, y^d, r_j^d, w_{ij}^d, s_{ij}^v \in \{0,1\}, d \in D, v \in V, (i,j) \in \mathcal{A}\\
& x^d, h_j^d, p_j^d \in \mathbb{R}^+, j \in Trips, d \in D(j).
\end{aligned}$$

The FVCS is an NP-hard problem since it involves several knapsack constraints, and it can be reduced to a multiple knapsack problem which is NP-hard in the strong sense (Martello and Toth, 1990; Chekuri and Khanna, 2005). In Section 5, we present experimental results obtained when solving this formulation with a commercial solver, but as expected, the solver struggles to obtain even a feasible solution for the real size instances. In the next section, we propose a Variable Neighborhood Search metaheuristic for the solution of FVCS in order to tackle these large instances.

4. Variable Neighborhood Search procedure for the FVCS problem

To solve the FVCS problem, we propose a Variable Neighborhood Search algorithm (VNS) introduced by Mladenović and Hansen (1997). The VNS consists in iteratively exploring different neighborhoods of a current solution to improve it. We design a procedure to construct an initial solution for the FVCS problem, three specific neighborhoods based on the different costs involved in the objective function of the problem, and a shaking procedure to avoid local optima.

For the VNS, we use the following concepts. Let set *Assign* contains all the tuples (*j, d, v*) which represent the possible assignment of trip $j \in Trips$ to driver $d \in D(j)$ and vehicle $v \in V(j) \cap V(d)$. A solution S of the VNS is a subset of *Assign* such that all the constraints presented in the previous section are satisfied. Let $T(S) \subset Trips$ be the set of trips that have been assigned in solution S, that is, $T(S) = \{j : (j,d,v) \in S$ for some $d \in D(j)$ and some $v \in V(j) \cap V(d)\}$. Similarly, let set $D(S) \subset D$ be the set of drivers that have been assigned in solution S, and $V(S) \subset V$ the set of vehicles assigned to a trip in solution S. An important characteristic of the FVCS problem is the length of the shift assigned to driver $d \in D(S)$ in solution S that we denote as $l_d(S)$. Finally, $\overline{D}(S) \subset D(S)$ represents the set of drivers with extra hours in solution S while $\underline{D}(S) = D(S) \setminus \overline{D}(S)$ is the set of drivers without extra hours in solution S.

More specifically, a tuple (*j, d, v*) is considered feasible for a solution S when $j \notin T(S)$ and the driver d and the vehicle v are currently available to perform the trip j. Availability of a driver includes the satisfaction of the constraints on its shift (breaks and duration). Then, the fitness $f(S)$ of a solution S is defined as $f(S) = W(S) + P(S)$ where:

$$W(S) = \sum_{d\in D(S)} payDay + \sum_{d\in\overline{D}(S)} extraPay + \sum_{v\in V(S)} vehicleCost,$$

$$P(S) = \sum_{j\in Trips\setminus T(S)} tripCost,$$

Algorithm 1 VNS.

Ensure: A Feasible Solution
  SBest ← *Constructor*(∅)
  $i \leftarrow 0$
  while $i < MaxIter$ **do**
    SCurr ← *Shake*(*SBest*)
    $k \leftarrow 1$
    while $k \leq n$ **do**
      $S \leftarrow NS_k(SCurr)$ {The Neighborhood k is explored}
      if $f(S) < f(SCurr)$ **then**
        SCurr ← *S*
        $k \leftarrow 1$ {The exploration of the Neighborhoods is reset}
      else
        $k \leftarrow k + 1$
      end if
    end while
    if $f(SCurr) < f(SBest)$ **then**
      SBest ← *SCurr* {*SBest* is updated}
      $i \leftarrow 0$ {The counter i is reset}
    else
      $i \leftarrow i + 1$
    end if
  end while
  return *SBest*

$P(S)$ is a penalization cost proportional to the number of trips not assigned in S. Indeed, recall that the FVCS problem requires all the trips to be assigned to a driver and a vehicle. Thus, the cost *tripCost* is taken relatively high in comparison to the wages and vehicle costs in order to prioritize the assignment of the trips and to avoid unfeasible solutions. Furthermore, relaxing the trip assignment constraints allows the VNS to accept unfeasible solutions in order to escape some local optima.

The general steps of the VNS are described in Algorithm 1. The VNS starts with an initial feasible solution obtained with the function *Constructor* and is considered, at the beginning, as the best-found solution so far (*Sbest*). Then, iteratively, this best solution is perturbed (function *Shake*) and the resulting current solution (*SCurr*) is explored successively by the different neighborhood search functions NS_k, $k = 1 \dots n$, where k is the index of the neighborhood and n is the total number of neighborhoods. Every time the current solution is improved, the exploration restarts from the first neighborhood, otherwise the next neighborhood is explored. The overall process is repeated until *MaxIter* non-improvements of the best solution.

The objective of the constructor is to build an initial solution for the VNS. Additionally, this constructor is also used in the shaking procedure and in the neighborhoods exploration every time a solution needs to be repaired. The pseudo code of the function *Constructor* is detailed in Algorithm 2. At each iteration of the algorithm, an unassigned trip j^* is selected randomly and a feasible tuple $(j^*, d, v) \in Assign$ is added to the solution S. In order to minimize the wages cost and the vehicles cost, the priority is given to drivers and vehicles that are already assigned to a trip.

In particular, the set $A_0^* \leftarrow \{(j, d, v) \in Assign \mid j = j^*,\ d \in \overline{D}(S), v \in V(S)\}$ contains all the tuples where trip j^* is assigned to a compatible driver that has a shift with extra hours and to a compatible vehicle that has covered at least one other trip. Set A_1^* is similar to A_0^* but this time the drivers are taken from set $D(S)$, that is, from the set of drivers that have at least one other trip assigned (not necessarily the drivers with extra hours in his shift). Set A_2^* corresponds to the tuples chosen from the set of drivers that have at least one other trip assigned but the set of vehicles is not restricted. Finally, set A_3^* does not restrict the set of drivers nor the set of vehicles. Note that, all the infeasible tuples with respect to the solution S are removed from A_0^*, A_1^*, A_2^*, and A_3^*.

One can note that we do not use a strictly Greedy criteria for the selection of the tuple $(j, d, v) \in Assign$ at each iteration of the Constructor. Indeed, the number of feasible assignations in *Assign* can be very large for the real size instances, and evaluating all of them at each iteration consumes a lot of processing time. Hence, we use a random selection from subsets of assignments with similar cost (sets A_0^*, A_1^*, A_2^*, and A_3^*), prioritizing the less expensive ones. This strategy permits in most cases to avoid the exploration of all the elements in *Assign*, since the first one encountered through a random enumeration, that satisfies the criteria of the explored subset, is added to the solution.

The *Shake* procedure, as described in Algorithm 3, partially destroys and reconstructs a feasible solution in order to obtain a new one. This procedure gives diversity in the VNS and allows to escape local minima. It consists of selecting randomly a vehicle that has been assigned to at least one trip in the current solution, withdrawing all the trips and drivers coupled to this vehicle, and then reconstructing the solution through the *Constructor* method. The idea is to randomly reassign these trips to other drivers and vehicles, giving priorities to the ones already used.

Algorithm 2 Constructor.

Require: A Solution S_0 to Repair
Ensure: A Feasible Solution
$S \leftarrow S_0$, $T' \leftarrow Trips \setminus T(S)$
while $T' \neq \emptyset$ **do**
Select the earliest trip $j^* \in T'$
$T' \leftarrow T' \setminus \{j^*\}$
$A_0^* \leftarrow \{(j,d,v) \in Assign \mid j = j^*,\ d \in \overline{D}(S), v \in V(S)\}$
$A_1^* \leftarrow \{(j,d,v) \in Assign \mid j = j^*,\ d \in D(S), v \in V(S)\}$
$A_2^* \leftarrow \{(j,d,v) \in Assign \mid j = j^*,\ d \in D(S)\}$
$A_3^* \leftarrow \{(j,d,v) \in Assign \mid j = j^*\}$
Remove from A_0^*, A_1^*, A_2^*, and A_3^* all the infeasible tuples w.r.t. the solution S
$A^* \leftarrow \emptyset$
if $A_0^* \neq \emptyset$ **then** $A^* \leftarrow A_0^*$
else if $A_1^* \neq \emptyset$ **then** $A^* \leftarrow A_1^*$
else if $A_2^* \neq \emptyset$ **then** $A^* \leftarrow A_2^*$
else if $A_3^* \neq \emptyset$ **then** $A^* \leftarrow A_3^*$
end if
if $A^* \neq \emptyset$ **then**
Select randomly $(j^*, d^*, v^*) \in A^*$
$S \leftarrow S \cup \{(j^*, d^*, v^*)\}$
end if
end while
return S

Algorithm 3 Shake.

Require: A Solution S_0 to Shake
Ensure: A Feasible Solution
$S \leftarrow S_0$
Select a vehicle $v^* \in V(S)$ randomly
$S \leftarrow \{(j,d,v) \in S \mid v \neq v^*\}$
$S \leftarrow Constructor(S)$
return S

Algorithm 4 NS_1 (Reducing the number of drivers with extra hours).

Require: A Solution S_0
Ensure: A Feasible Solution
$S \leftarrow S_0$, $D' \leftarrow \overline{D}(S)$
while $D' \neq \emptyset$ **do**
Select $d^* \in D'$ randomly, $D' \leftarrow D' \setminus \{d^*\}$, $S' \leftarrow S$
while $l_{d^*}(S') > regularDay$ **do**
Remove in S' the first or the last trip assigned to d^* {The one that produces the greatest decrease in $l_{d^*}(S')$}
end while
$S' \leftarrow Constructor(S')$
if $f(S') < f(S)$ **then** $S \leftarrow S'$, $D' \leftarrow \overline{D}(S)$
end while
return S

We define three different neighborhoods for the improvement phase of the VNS: NS_1, NS_2, and NS_3. The first neighborhood, NS_1, is detailed in Algorithm 4 and aims to reduce the number of shifts with extra hours. At each iteration, a driver with extra pay is selected, some of his trips are removed until the length of his shift is below *regularDay*, and then we try to reassign these trips to other drivers via the Constructor function.

The second neighborhood, NS_2, tries to reduce the number of vehicles by randomly eliminating all the trips assigned to a chosen vehicle v^* and reassigning them to other used vehicles via the Constructor function. NS_2 is described in Algorithm 5.

The last neighborhood, NS_3, consists in reducing the number of drivers where all the trips of a random selected driver are removed from his shift and reassigned in priority to other already used drivers. Algorithm 6 gives further details on neighborhood NS_3.

Algorithm 5 NS_2 (Reducing the number of vehicles).

Require: A Solution S_0
Ensure: A Feasible Solution
  $S \leftarrow S_0$
  for $v^* \in V(S)$ **do**
    $S' \leftarrow \{(j, d, v) \in S \mid v \neq v^*\}$
    $S' \leftarrow Constructor(S')$
    if $f(S') < f(S)$ **then** $S \leftarrow S'$
  end for
  return S

Algorithm 6 NS_3 (Reducing the number of drivers).

Require: A Solution S_0
Ensure: A Feasible Solution
  $S \leftarrow S_0$
  for $d^* \in D(S)$ **do**
    $S' \leftarrow \{(j, d, v) \in S \mid d \neq d^*\}$
    $S' \leftarrow Constructor(S')$
    if $f(S') < f(S)$ **then** $S \leftarrow S'$
  end for
  return S

Table 1
Generated instances.

	Periods	Lines l	Trips per l
$Instances^2$	2 and 7	1 and 2	[7,15]
$Instances^{day}$	7	1,2,5,15, and 25	[22,46]
$Instances^{adj}$	7	1,2,5,15, and 25	[7,46]

All these neighborhood search algorithms use the procedure *Constructor* to repair a solution. With the Constructor, the Shake procedure and the three different neighborhoods, we obtain a VNS that finds high-quality solutions of the FVCS problem in small computational times as it is shown in the computational experiments presented in Section 5.

5. Experimental results

In this section, we empirically validate the mixed integer formulation and the VNS for the FVCS problem. All experiments were carried out on a MacPro with 2x2.4 GHz quad-core Intel zeon, with 20GB of RAM and using the version 12.7 of the CPLEX solver.

5.1. Instances

We have randomly generated 400 instances that are divided into three sets based on the real data of a bus company. The first one, set $Instances^2$, only considers instances with one or two lines and two or seven planning periods. The second one, $Instances^{day}$, contains instances simulating a whole day with seven periods where three of them correspond to the morning, midday, and evening rush hours. The third one, $Instances^{adj}$, is composed of instances where the number of available vehicles and drivers is more restricted in order to further test the capacity of our VNS to find feasible solutions in such a situation. As shown in Table 1, we have generated instances with one or two lines for set $Instances^2$ with a minimum of 7 and a maximum of 15 trips per line. For the set $Instances^{day}$, we have instances with 1, 2, 5, 15, and 25 lines with a minimum of 22 and a maximum of 46 trips per line. The set $Instances^{adj}$ includes instances between 1 and 15 lines with 7 to 46 trips per line.

The set $Instances^2$ aims to test the limits of the proposed formulation, while $Instances^{day}$ are to test the VNS approach. The *real* instances that we have observed in our case study have 7 periods, 15 lines, and around 30 trips per line in a rush hour, as considered in $Instances^{day}$. The number of vehicles and the number of drivers in each instance is equal to the maximum number of trips in a line per the number of lines. Thus, there are much more vehicles and drivers than needed to cover all the trips of all lines. This situation is consistent with what has been observed in our case study, since most bus companies in developed countries have to deal daily with no-show drivers or buses involved in minor accident due to the heavy traffic. Nevertheless, in order to show the efficiency of our approach, we created a third set of adjusted instances named $Instances^{adj}$ where the number of vehicles and drivers is only 10% more than required to cover all the trips.

Table 2
Results of the CPLEX solver and the VNS for the set *Instances*[2].

			CPLEX		VNS		
Lines	Periods	($\delta_{D(j)}$, $\delta_{V(j)}$, $\delta_{V(d)}$)	%_Gap	CPU_time (s)	%_Gap	CPU_time (s)	%_imp
1	2	(1, 1, 1)	0.00	0.27	0.00	0.08	0.00%
2	2	(1, 1, 1)	0.00	1.67	0.00	0.26	0.00%
1	7	(1, 1, 1)	40.19	3600	40.19	0.52	0.00%
2	7	(1, 1, 1)	453.79	3600	206.23	1.95	43.84%
1	2	(1, 0.5, 0.5)	0.00	0.67	0.00	0.08	0.00%
2	2	(1, 0.5, 0.5)	0.00	4.47	0.00	0.25	0.00%
1	7	(1, 0.5, 0.5)	53.17	3600	53.01	0.40	0.08%
2	7	(1, 0.5, 0.5)	359.11	3600	174.55	1.60	36.74%
1	2	(0.5, 1, 0.5)	0.00	0.48	0.00	0.06	0.00%
2	2	(0.5, 1, 0.5)	0.00	4.24	0.00	0.29	0.00%
1	7	(0.5, 1, 0.5)	47.69	3600	46.65	0.39	0.75%
2	7	(0.5, 1, 0.5)	250.72	3600	127.75	1.53	31.76%
1	2	(0.5, 0.5, 1)	0.00	0.21	0.00	0.07	0.00%
2	2	(0.5, 0.5, 1)	0.00	1.39	0.00	0.32	0.00%
1	7	(0.5, 0.5, 1)	31.80	3600	31.22	0.41	0.43%
2	7	(0.5, 0.5, 1)	350.33	3600	185.91	1.77	32%
1	2	(0.5, 0.5, 0.5)	0.00	0.48	0.00	0.07	0.00%
2	2	(0.5, 0.5, 0.5)	0.00	2.51	0.00	0.25	0.00%
1	7	(0.5, 0.5, 0.5)	34.90	2520.21	34.09	0.30	0.53%
2	7	(0.5, 0.5, 0.5)	227.90	3600	143.91	1.53	23.33%

In practice, we have seen that a trip can be covered by approximately half of the total list of drivers, and the same statement applies to the number of vehicles that can perform a trip and the number of vehicles than can be operated by a driver. Nevertheless, in order to simulate different scenarios, we define as $\delta_{D(j)}$ the probability that a driver is allowed to perform the trip j, $\delta_{V(j)}$ the probability that a vehicle is allowed to be assigned to the trip j, and $\delta_{V(d)}$ the probability that a vehicle is allowed to be used by the driver d. To evaluate the impact of these probabilities in our solution method, *Instances*[2] and *Instances*^{day} are such that $(\delta_{D(j)}, \delta_{V(j)}, \delta_{V(d)}) \in \{(0.5, 0.5, 0.5), (1, 0.5, 0.5), (0.5, 1, 0.5), (0.5, 0.5, 1), (1, 1, 1)\}$. The adjusted set *Instances*^{adj} where only generated such that $(\delta_{D(j)}, \delta_{V(j)}, \delta_{V(d)}) = (1, 1, 1)$ to be able to assure feasibility of the instances. For each case we have generated 10 instances.

The following parameters are also based on our case study. The working day is $regularDay = 8$ h and the daily salary of a driver is $dayPay = 500$, but a driver can do two more extra hours ($extraHours = 2$ h) for $extraPay = 100$ more. The round times of the trips are between [50,100] minutes for the non-rush hours, and between [100,150] minutes for the rush ones.

The timetables were generated as follows: The maximum *headway* between trips of the same line is equal to the size of the period divided by the number of trips in the period; Then, we distribute uniformly the trips over the period where the headway between two consecutive trips is taken randomly in [1, *headway*]. The fixed cost of a vehicle has been set to $vehicleCost = 300$. Indeed, the bus company wishes to have as few as possible vehicles in use to be able to apply preventive maintenance but it is a qualitative criterion that we had to adjust in preliminary experimentation: this cost must be less than the wage of a driver to avoid focusing on the vehicles rather than on the drivers.

Our instances are such that every trip has at least one vehicle that can serve it, every driver is able to serve at least one trip, every driver is compatible with at least one vehicle, and vice-versa. Note that, in all the experimentation presented in the sequel, the proposed VNS reports for all the instances a solution were all the trips are covered. That is to say, the penalization cost in the fitness function of the best-found solutions of the VNS is always equal to zero when the procedure stops.

5.2. Results for the set Instances[2]

As we mentioned before, *Instances*[2] are designed to test the limit of the proposed mixed integer linear formulation for the FVCS problem when exactly solved by the integer linear programming solver CPLEX 12.7. We set a maximum of one hour of processing time and the relative tolerance on the gap[2] to 0. In our preliminary experimentation with the proposed formulation, CPLEX was not able to report a feasible solution for all the small instances within the time limit. In order to have a fair comparison with our VNS and to improve the results returned by CPLEX, we relax the Constraints (7) and (8) and allow that some trips can be uncovered with the same penalty cost used in the fitness function of the VNS. Table 2 presents the results obtained with CPLEX and our VNS for the set *Instances*[2].

The firsts three columns refer to the characteristics of the instance type. Columns "%_Gap" and "%CPU_time (s.)" show the average gap and average processing time (in seconds) for each instance class (Recall that each class contains 10 instances).

[2] The gap is computed as $|bestbound - bestinteger|/(1e-10+|bestinteger|)$, where *bestbound* and *bestinteger* are the best integer objective and the objective of the best node remaining, respectively.

Table 3
Results of the VNS for the set $Instances^{day}$.

Lines	$(\delta_{D(j)}, \delta_{V(j)}, \delta_{V(d)})$	%_Drivers	%_Vehicles	CPU_time (s)
5	(1,1,1)	49.20	26.00	15.45
15	(1,1,1)	46.99	24.17	246.08
25	(1,1,1)	46.29	23.66	969.25
5	(1, 0.5, 0.5)	50.19	25.99	11.84
15	(1, 0.5, 0.5)	46.63	24.06	160.08
25	(1, 0.5, 0.5)	46.40	23.87	565.17
5	(0.5, 1, 0.5)	49.57	25.73	12.19
15	(0.5, 1, 0.5)	46.75	24.32	171.59
25	(0.5, 1, 0.5)	45.46	23.18	720.93
5	(0.5, 0.5, 1)	48.74	26.06	11.53
15	(0.5, 0.5, 1)	46.99	24.23	191.04
25	(0.5, 0.5, 1)	46.27	23.51	644.74
5	(0.5, 0.5, 0.5)	49.62	25.98	12.97
15	(0.5, 0.5, 0.5)	45.60	23.34	167.29
25	(0.5, 0.5, 0.5)	45.35	23.19	611.33

Finally, column "%_imp" represents the average relative improvement of the feasible solution obtained by the VNS over the one found by CPLEX.

Notice that for all the classes of instances, the proposed VNS returns an equivalent or a better solution than CPLEX within a relatively low processing time. In particular, our VNS yields optimal solutions for the smaller instances with one or two lines and two planning periods. Moreover, in the case of instances with two lines and seven planning periods, the VNS clearly outperforms CPLEX. The poor gap observed for the instances with 7 periods reflects the poor quality of the linear relaxation of the mathematical formulation which impacts greatly the performance of CPLEX. For the set $Instances^2$, the VNS appears to be more efficient in finding solutions closer to optimality than CPLEX.

We also conducted computational test where the best solution returned by the VNS is passed to CPLEX as an initial solution. In this case, CPLEX was able to find a better solution for only 3 instances out of the 200 ones, and the average improvement on these instances was less than 1%. Hence, we do not present the detail results of this experimentation since they do not bring further information on the performance of our VNS.

5.3. Results for the set $Instances^{day}$

The set of instances $Instances^{day}$ were generated to evaluate the performance of the VNS. For this set of instances, in most cases, CPLEX is not able to provide at least one feasible solution based on the proposed MILP formulation, and for the largest instances, it stays stocked at the root of the branch-and-bound algorithm.

Table 3 presents the results obtained with the VNS where the objective is to analyze the processing time overhead, when the number of lines to schedule increases, and the overall solution quality based on the quantity of drivers and vehicles used to cover all the trips. The first two columns refer to the number of lines and the compatibility parameters considered for each instance class. Column (%_Drivers) and column (%_Vehicles) show the average percentage of the drivers and vehicles used, respectively, in the best solution returned by the VNS. The last column (CPU_time) displays the average processing time in seconds.

We can remark that the obtained solutions use around 50% of the drivers and 25% of the vehicles available. These percentages stay relatively stable independently of the number of lines to schedule. Recall that the number of vehicles and the number of drivers in each instance is equal to the maximum number of trips in a line per the number of lines. Thus, there are much more vehicles and drivers than needed to cover all the trips of all lines. Furthermore, in the FVCS, there is no particular constraints on the use of the vehicles, hence, as expected, the vehicles assignment is more efficient than the one of the drivers that have a restricted duty length and rests.

The overall processing time is below 1000 s. The hardest instances are the ones with compatibility factors (1,1,1) that correspond to instances where all the drivers can drive any trip, all vehicles can be assigned to any trip, and any vehicle can be operated by any driver. This high compatibility increases the number of feasible assignments in the solution space yielding to a longer computational time. Nevertheless, the instances with the smallest compatibility factors, that is, the ones with (0.5, 0.5, 0.5) are not the easiest ones in terms of computational time. Thus, the relation between the compatibilities and the complexity of the instance should be studied further.

Fig. 3 shows the evolution of the processing time according to the number of lines to schedule. Note that the curve is quadratic which allows our VNS approach to handle relatively large instances in a short time. With 25 lines, we are dealing with real size instances that we aim to solve with our VNS, nevertheless, this figure shows that we can expect that the proposed solution procedure could tackle larger instances contrary to the MILP formulation. Since the linear relaxation of the proposed model is very poor, we do not compare its bound with the one of the VNS.

In order to analyze the performance of each neighborhood, we compute the percentage of times the solution is improved with respect to the number of calls (%_#_Imp) and the percentage of the total processing time used (%_CPU). The average

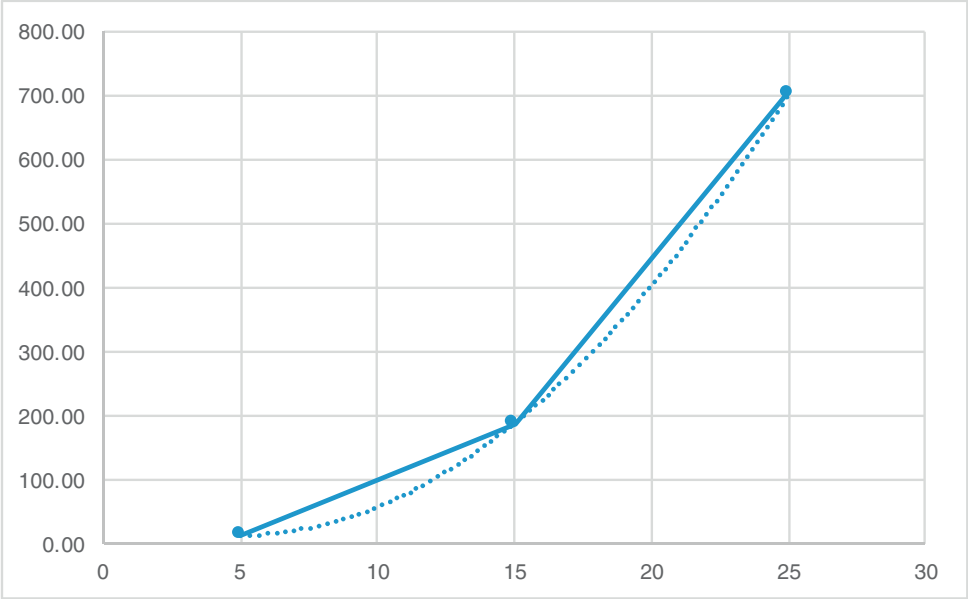

Fig. 3. CPU time according to the number of lines.

Table 4
Neighborhood Performance of the VNS for the set $Instances^{day}$.

		NS_1		NS_2		NS_3	
Lines	$(\delta_{D(j)}, \delta_{V(j)}, \delta_{V(d)})$	%_#_Imp	%_CPU	%_#_Imp	%_CPU	%_#_Imp	%_CPU
5	(1,1,1)	15.40	1.75	46.93	58.50	22.50	39.02
15	(1,1,1)	18.19	1.28	45.85	60.25	24.27	38.12
25	(1,1,1)	18.53	1.05	44.49	60.62	26.01	38.05
5	(1, 0.5, 0.5)	21.16	2.63	47.20	55.03	18.30	41.48
15	(1, 0.5, 0.5)	22.33	1.53	45.11	55.52	24.58	42.54
25	(1, 0.5, 0.5)	21.95	1.40	45.25	56.13	23.79	42.15
5	(0.5, 1, 0.5)	17.70	2.15	47.90	56.76	16.41	40.27
15	(0.5, 1, 0.5)	20.81	1.48	44.77	57.26	18.57	40.85
25	(0.5, 1, 0.5)	20.21	1.28	46.72	58.70	22.51	39.73
5	(0.5, 0.5, 1)	15.11	1.76	48.48	57.53	20.52	39.89
15	(0.5, 0.5, 1)	19.45	1.34	43.87	56.87	23.46	41.41
25	(0.5, 0.5, 1)	17.32	1.14	45.73	58.51	28.03	40.05
5	(0.5, 0.5, 0.5)	20.19	2.45	47.76	55.45	17.51	41.18
15	(0.5, 0.5, 0.5)	26.18	2.00	42.48	54.26	21.42	43.32
25	(0.5, 0.5, 0.5)	21.97	1.34	45.17	56.29	22.66	42.06
	Average	19.77	1.64	45.85	57.18	22.04	40.67

results obtained for the instances $Instances^{day}$ is presented in Table 4. The neighborhood NS_2 appears to be the best to improve the incumbent solution followed by NS_3 and then NS_1. However, the performance of NS_2 and NS_3 is impacted by a high consumption of the overall processing time. From these results, it makes sense why this particular order of exploration of the neighborhoods gave us the best results. Indeed, NS_1 does not perform very well but it is very fast, hence it is inexpensive to explore this neighborhood first. NS_2 performs much better than NS_3 with a slightly CPU time overhead, that is why priority is given to NS_2. NS_3 is then explored last to check if a better solution can be eventually found before leaving the VNS. The average improvement of the solution of these three neighborhoods is very low (less than 2% in average) as expected. Indeed, they aim at removing one shift with overtime, or one used vehicle, or one driver, which have a relatively small impact on the total cost of a solution.

5.4. Results for the set $Instances^{adj}$

The set $Instances^{adj}$ contains instances where the number of drivers and vehicles are much smaller than the one in the set $Instances^{day}$. Even if the set $Instances^{day}$ is correlated to what has been observed in practice, the objective of testing our VNS with such instances is to evaluate its capacity to find feasible solution when the number of available resources is further restricted. As in the previous section, we report in Table 5 the results obtained with our VNS, and in Table 6 the

Table 5
Results of the VNS for the set $Instances^{adj}$.

Lines	%_Drivers	%_Vehicles	CPU_time (s)	%_Imp
1	88.83	85.93	0.60	−5.12
2	89.89	89.92	1.59	23.25
5	90.54	89.86	9.02	–
15	90.68	90.60	130.68	–
25	90.77	90.65	473.61	–

Table 6
Neighborhood performance of the VNS for the set $Instances^{day}$.

	NS_1		NS_2		NS_3	
Lines	%_#_Imp	%_CPU	%_#_Imp	%_CPU	%_#_Imp	%_CPU
1	12.68	4.45	49.21	52.16	18.04	37.11
2	14.26	3.30	47.77	54.42	22.01	39.77
5	16.63	1.94	48.00	55.96	21.04	41.07
15	16.64	1.38	45.99	55.64	24.79	42.52
25	19.82	1.36	44.91	55.14	25.44	43.13
Average	16.01	2.48	47.17	54.67	22.26	40.72

performance of each neighborhood. Note that, the last column of Table 5, %_Imp reports the average relative improvement of the bound of our VNS over CPLEX.

From Table 5, one can note that around 90% of the drivers and vehicles are used in the final solution returned by our VNS. Our heuristic reports a feasible solution for all of these instances within a relatively low processing time compared to $Instances^{day}$. The performance of the different neighborhoods is consistent with what has been observed previously. The column %_Imp only contains the results obtained for the small instances since CPLEX was not capable return a feasible solution for the rest of the instances within an hour of execution time. One can remark that the solutions of CPLEX for the instances with 1 line are slightly better than the one of the VNS. However, for the instances with more than 2 lines the VNS significantly outperforms CPLEX in both quality of the solution and processing time. Hence, this computational test shows that our VNS is sufficiently robust to find a feasible solution for cases where the available resources are limited.

6. Conclusions

We present a complete integrated approach for the vehicle and Crew Scheduling Problem where a set of trips of circular transit lines (at one depot) must be assigned to vehicle-driver pairs. The FVCS considers a flexible operation with the following assumptions: (i) there exist trip-vehicle, trip-driver, and vehicle-driver compatibilities constraints due to the vehicles characteristics and the drivers qualifications; and (ii) the drivers have a limited duty length, mandatory rests, a restricted consecutive driving time, and a limited extra working hours.

The driver's break times can be allocated anywhere in their schedules as long as labor regulations are satisfied. Moreover, we need to identify each driver and each vehicle in order to define a trip-vehicle-driver assignment based on their respective compatibility. To the best of our knowledge, this problem has not been addressed in the literature.

For the solution of the FVCS problem, we have defined a network flow formulation that allows us to obtain optimal solutions for small instances and a Variable Neighborhood Search metaheuristic to solve large instances in a small computational time. The experimentation carried out shows that the mathematical model is impractical for instances with more than 2 lines, contrary to the proposed VNS, which is able to tackle the assignment of 25 lines in less than 1200 s.

Further research lines include the natural case of multiple depots and the case where the vehicles have a restriction on their usage like fuel charges. Moreover, the case where the timetables are optimized together with the vehicle and driver scheduling is of most importance in order to further reduce the operative costs.

Conflict of interest

The authors declare that they have no conflict of interest.

Acknowledgment

We wish to thank Victoria Chávez for the fruitful discussions.

Appendix A. Linearization of Constraints (15) of the FVCS formulation

Recall that Constraints (15) combined with the objective function define the consecutive working time of driver d (with no rest) before covering trip j as $p_j^d = p_i^d + sched_j - sched_i$ if $w_{ij}^d = 1$ and $r_j^d = 0$:

$$M_2 r_j^d + p_j^d \geq \sum_{i:(i,j)\in\mathcal{A}} w_{ij}^d \left(p_i^d + sched_j - sched_i\right)$$

Since the formulation is quadratic, we use a classic linearization technique (see Williams, 2013) where the product $w_{ij}^d p_i^d$ is replaced by an auxiliary variable u_{ij}^d. Then the Constraints (15) can be rewritten, for a given arc $(i, j) \in \mathcal{A}$ and a driver $d \in D(i) \cap D(j)$, as follows:

$$u_{ij}^d \leq M_2 \cdot w_{ij}^d,$$
$$u_{ij}^d \leq p_i^d,$$
$$u_{ij}^d \geq p_j^d - M_2 \cdot (1 - w_{ij}^d).$$

References

Borndörfer, R., Löbel, A., Weider, S., 2002. Integrierte Umlauf und Dienstplanung im Nahverkehr. Technical Report 02-10. ZIB, Konrad-Zuse Zentrum, Berlin, German.

Borndörfer, R., Löbel, A., Weider, S., 2008. A bundle method for integrated multi-depot vehicle and duty scheduling in public transit. In: Computer-Aided Systems in Public Transport. Springer, pp. 3–24.

Chekuri, C., Khanna, S., 2005. A polynomial time approximation scheme for the multiple knapsack problem. SIAM J. Comput. 35, 713–728.

Freling, R., Huisman, D., Wagelmans, A.P.M., 2001. Applying an Integrated Approach to Vehicle and Crew Scheduling in Practice. In: Voß, S., Daduna, J. (Eds.), Computer-Aided Scheduling of Public Transport. In: Lecture Notes in Economics and Mathematical Systems, 505. Springer, Berlin, Germany, pp. 73–90.

Freling, R., Huisman, D., Wagelmans, A.P.M., 2003. Models and algorithms for integration of vehicle and crew scheduling. J. Scheduling 6, 63–85.

Gaffi, A., Nonato, M., 1999. An integrated approach to ex-urban crew and vehicle scheduling. In: Wilson, N. (Ed.), Computer-Aided Transit Scheduling. Springer, Berlin, Germany, pp. 103–128. 471 of Lecture Notes in Economics and Mathematical Systems

Gintner, V., Kliewer, N., Suhl, L., 2008. A crew scheduling approach for public transit enhanced with aspects from vehicle scheduling. In: Hickman, M., Mirchandani, P., Voß, S. (Eds.), Computer-aided Systems in Public Transport, 600. Springer Berlin Heidelberg, pp. 339–359.

de Groot, S., Huisman, D., 2004. Vehicle and crew scheduling : solving large real-world instances with an integrated approach. In: Proceedings of the 9th International Conference on Computer-Aided Scheduling of Public Transport.

Haase, K., Desaulniers, G., Desrosiers, J., 2001. Simultaneous vehicle and crew scheduling in urban mass transit systems. Transp. Sci. 35, 286–303.

Ibarra-Rojas, O., Delgado, F., Giesen, R., Muñoz, J., 2015. Planning, operation, and control of bus transport systems: a literature review. Transp. Res. B 77, 38–75.

Kéri, A., Haase, K., 2007a. Simultaneous vehicle and crew scheduling with trip shifting. Oper. Res. Proc. 2007, 467–472.

Kéri, A., Haase, K., 2007b. Vehicle and crew scheduling with flexible timetable. Oper. Res. Proc. 2006, 339–342.

Laurent, B., Hao, J.K., 2008. Simultaneous vehicle and crew scheduling for extra urban transports. In: Proceedings of the 21st International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems: New Frontiers in Applied Artificial Intelligence. Springer-Verlag, Berlin, Heidelberg, pp. 466–475.

Leone, R., Festa, P., Marchitto, E., 2011. A bus driver scheduling problem: a new mathematical model and a GRASP approximate solution. J. Heuristics 17, 441–466.

Martello, S., Toth, P., 1990. Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons, Inc..

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Comput. Oper. Res. 24, 1097–1100.

Steinzen, I., Becker, M., Suhl, L., 2007. A hybrid evolutionary algorithm for the vehicle and crew scheduling problem in public transit. In: Proceedings of the IEEE Congress on Evolutionary Computation, Singapore, pp. 25–28.

Vaidyanathan, B., Jha, K., Ahuja, R., 2007. Multicommodity network flow approach to the railroad crew-scheduling problem. IBM J. Res. Dev. 51, 325–344.

Williams, H., 2013. Model Building in Mathematical Programming. Wiley.