Two-phase heuristic for SNCF rolling stock problem

Mirsad Buljubašić, Michel Vasquez, Haris Gavranović

Two-phase heuristic for SNCF rolling stock problem

Mirsad Buljubašić[1] · Michel Vasquez[1] · Haris Gavranović[2]

[1] LGI2P Laboratory, Ecole des Mines d'Alès, Nîmes, France

[2] International University of Sarajevo, Sarajevo, Bosnia and Herzegovina

Abstract A two-phase approach was adopted to solve the problem given during the ROADEF/EURO Challenge 2014 competition. The problem focuses on rolling stock management at railway sites, as defined by French Railways (SNCF). In the first phase, a train assignment problem is solved by combining a greedy heuristic procedure with integer programming. The objective is to maximize the number of assigned departures while meeting technical constraints. The second phase consists of scheduling train movements inside the station while minimizing the number of cancelled (uncovered) arrivals and departures. This schedule has to comply with resource constraints such as capacity, length, order of trains, etc. A constructive heuristic is used to build a feasible schedule, which is subject to improvement by an iterative procedure based on a local search. Experiment results, that demonstrate the effectiveness of our approach on the large scale instances provided by SNCF, are presented hereafter.

Keywords Greedy heuristic · Integer programming · Local search · Rolling stock · Scheduling

1 Introduction

This paper considers the problem of rolling stock unit management at railway sites, as defined by French Railways (SNCF) and proposed at the ROADEF/EURO Challenge 2014. The prob-

✉ Mirsad Buljubašić
mirsad.buljubasic@mines-ales.fr

Michel Vasquez
michel.vasquez@mines-ales.fr

Haris Gavranović
haris.gavranovic@gmail.com

lem involves managing trains between their arrivals and departures at terminal stations. The purpose of this paper is to describe the method developed by our team upon taking part in this challenge. The main goal was to achieve the best possible results in relation to the other competitors. This problem is currently being jointly addressed by several SNCF Departments, by breaking it down into several sub-problems (train assignment, scheduling constraints, track group conflicts, platform assignment, maintenance planning, etc.) to be solved sequentially. Consequently, the integrated problem formulation dealt with here (in the challenge) actually reflects a forward-looking approach. Between terminal station arrivals and departures, the trains in fact are always present. This aspect, unfortunately, often gets neglected in railway optimization methods (Ramond and Marcos 2014). In the past however, rail networks possessed sufficient capacity to accommodate all trains without introducing excessive constraints—but this is no longer the case. Traffic has indeed increased considerably during recent years, and a number of stations are now experiencing serious congestion (Ramond and Marcos 2014; Corman et al. 2010; Huisman et al. 2005). Current traffic trends will make this phenomenon even more challenging over the next several years. The proposed model focuses on the multiple dimensions inherent in this problem, by taking into account many different aspects. The model's scope remains within geographically limited boundaries, typically over just a few km in urban environments: train stations and their surrounding railway infrastructures are targeted by this model. The solutions to such problems involve temporary parking and shunting on station locations, which typically consists of platforms, maintenance facilities, rail yards adjacent to stations and the set of tracks linking these various resources (this infrastructure constitutes what is referred to as the "system").

This paper will be organized as follows. A description of the problem is provided in Sect. 2. The description given herein was borrowed from the official competition subject (Ramond and Marcos 2014). Section 3 addresses a related work. Our two-phase approach is described in Sect. 4: solving the problem of matching (assigning) trains to departures is considered first, followed by the problem of scheduling trains inside the station. An iterative improvement procedure, based on a local search, is presented at the end of this section. The computational results obtained from the available set of instances, provided by SNCF, are detailed in Sect. 5. The last section presents the authors' final remarks and conclusion.

2 Problem statement

A comprehensive description of the problem can be found in the competition subject (Ramond and Marcos 2014). Herein, we give a simplified, more concise formulation which is sufficient to describe our approach and make our presentation clearer and more readable.

2.1 Planning horizon

The planning horizon considered in this problem is an integral number of days ($nDays$) from morning of day 1 to midnight of day $nDays$. We use the word "time" to represent a time instant (date/time) during the horizon. The time horizon is discretized, the smallest duration taken into account being one second.

2.2 Arrivals and departures

Arrivals generate entrances of trains in the system and departures are the way trains leave the system. Arrival and departure times are provided as a fixed input and are considered to be

non-modifiable. It had to be decided which train is assigned to each departure. It is feasible, but penalizing, not to cover (i.e. to cancel) some arrivals or departures. The set of train arrivals is denoted by $\mathcal{T}$ and a train $t \in \mathcal{T}$ is defined by the following characteristics:

- arrival time $arrTime_t$,
- $idealDwell_t$ and $maxDwell_t$, respectively representing the ideal and the maximum time train should stay on the platform after $arrTime_t$,
- remaining distance before maintenance $remDBM_t$, determining whether or not t must perform maintenance operations before being assigned to a departure.

The following attributes define a departure $d \in \mathcal{D}$, where $\mathcal{D}$ represents the set of all departures:

- $depTime_d$, $idealDwell_d$, $maxDwell_d$: all similar to the train arrival attributes,
- distance $reqD_d$ of the journey following the departure; this value is compared, for a train $t \in \mathcal{T}$, with $remDBM_t$, to determine whether or not maintenance operations have to be performed on t before $depTime_d$.

In practice, the trains associated with arrivals are usually trains which were earlier assigned to departures and which spent some time out of the system before coming back. To take this into account, some arrivals are linked with departures occurring earlier in the horizon. When an arrival t has a linked departure d, there are two distinct. If d is cancelled (i.e. if no train is assigned to d), then $remDBM_t$ is the one provided in the input data for t. If d is covered, then $remDBM_t$ is replaced by the one induced by the train t' assigned to d i.e. $remDBM_t = remDBM_{t'} - reqD_d$.

2.3 Maintenance

Trains must be serviced on a regular basis to enable them to operate safely. Maintenance work helps restore $remDBM_t$ to its maximum value $maxDBM_t$. Maintenance operations can only be performed at servicing facilities. As the system's maintenance capability is limited, the number of operations which can be performed in one day, i.e. across all the system's maintenance facilities, is bounded by a maximum value represented by $maxMaint$.

2.4 Infrastructure resources

Between arrivals and departures, trains are either moving or parked on tracks that we consider to be resources. Let $\mathcal{R}$ be the set of all resources. Resources can be either platforms, maintenance facilities, yards or track groups. Platforms and maintenance facilities represent portions of tracks considered individually, while track groups and yards are aggregated types of resources which usually contain more than one track as well as switches to physically link the different tracks together. An example of resources configuration is given in Fig. 1. A resource $r \in \mathcal{R}$ has a set of neighbouring resources, defining the possible transitions for trains, in a symmetrical way. Resources represent railways infrastructure elements which are in general linear and can be accessed from both sides, and in some cases from only one side. Given this aspect, the neighbours of a resource can then be divided into two subsets, one being physically associated with each side of the linear element. The transitions between a resource, r, and one of its neighbours, are performed through one of the entry/exit points, which we call *gates*, located on either side of the resource. These gates are the physical tracks linking the various resources. Platforms and facilities, representing individual tracks, have at most one gate on each side. Only track groups and yards may have more than one gate on each side. For these types of resources, two adjacent resources might be linked by more

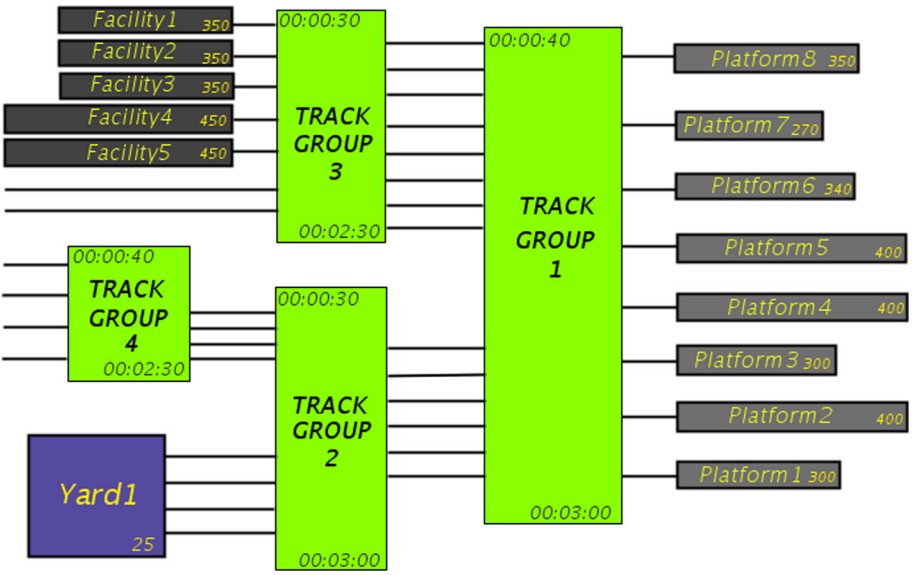

Fig. 1 Example of infrastructure with 8 platforms, a single yard, 5 maintenance facilities and 4 track groups connecting them

than one gate: the gate used for the transition between two resources has to be specified. For example, track group 1 in Fig. 1 has 8 gates and 8 neighbouring resources on the right side, and 14 gates and 2 neighbouring resources to the left.

Platforms represent tracks within the train station where passengers can board and disembark. *Maintenance facility* resources are special tracks inside maintenance workshops. They are used to periodically reset the train $remDBM$. Platforms and maintenance facilities are both characterized by a length which may not be exceeded by the total length of trains using it (length of platforms in the example shown in Fig. 1 varies from 270 to 400, while all facilities have a length of 350 or 450).

Yards are sets of tracks used to park trains. Trains can stay in yards with no time restriction, but a yard has a limited physical capacity in terms of the number of trains it can handle simultaneously (25 for the yard in Fig. 1 example). Capacity is the only constraint when parking the trains in the yards, i.e. there are no constraints on movement of trains in the yards as is the case on platforms or in maintenance facilities.

Track groups are sets of tracks used by trains to move throughout the system. Its real physical configuration in terms of tracks and switches linking them can be very complex. Its complexity is not taken into account herein; we have considered it as a black box with some indications on how to identify conflicts. The duration of use of track group k by any train (i.e. travel time) is a constant denoted by $trTime_k$. It is the time required by a train to enter the track group k on one side and exit from the opposite side. Indeed, a train entering on one side of a track group must exit from the other side. All gates on one side are reachable from all gates of the opposite side. In addition, $hwTime_k$ represents the headway of the track group: this is a safety time which must be adhered to at any place between two trains. In Fig. 1, $trTime$ and $hwTime$ are shown for each track group in the top left and bottom right corners respectively; for example, for track group 2 we have $trTime = 30\,\text{s}$ and $hwTime = 3\,\text{min}$. When several trains use a track group over the same time period, there may be conflicts

between them. A conflict occurs if the paths of two trains on the track group intersect and headway time $hwTime$ between them is not complied with. Conflicts are considered to be unfeasible. Each arrival and each departure has a fixed arrival/departure sequence. These sequences represent the routing of trains on the tracks during the last few km before reaching the platforms. Sequence is defined by an ordered set of track groups that a train has to use when arriving/departing. In Fig. 1, possible arrival sequences are *TrackGroup3-TrackGroup1* and *TrackGroup4-TrackGroup2-TrackGroup1*.

2.5 Solution representation and objective function

A solution to the problem consists of a set of schedules, one for each train. The train schedule is a sequence of events during its presence in the system, along with details such as the time of each event, the resources used, etc. With this information for every train, the status of the system and each of its resources can be derived at any time during the horizon. The objective function to be minimized is a weighted sum of the following individual costs:

1. Uncovered arrival/departure cost,
2. Platform usage cost.

The uncovered arrival/departure cost is proportional to the number of cancelled arrivals and departures. Platform usage cost is the sum of the costs of using the platforms during arrivals and departures. Each arrival $t \in \mathcal{T}$ has an ideal dwell time $idealDwell_t$, and each departure $d \in \mathcal{D}$ also has an ideal dwell time $idealDwell_d$. The cost of using the platform during arrival/departure is proportional to the difference between actual dwell time (platform use duration) and ideal dwell time. It is important to mention that the actual objective function, as defined in the competition subject (Ramond and Marcos 2014), contains several other parts, but for reasons of simplicity, have been omitted here. Furthermore, for the set of instances introduced in the competition, these two objectives are, by far, the most critical.

3 Related work

A large body of literature on train routing problems is available. However, any exact or even similar matches from previous research with the current problem could not be identified. Only variations to some of the sub-problems occurring here can be found in publications such as Lentink et al. (2003) and Freling et al. (2005). Furthermore, there is a broad range of optimization models for specific problem variants. We will not therefore be emphasizing any of the papers or related problem variants herein.

Recently, both during and after the competition, a few papers (or technical reports) were published on this topic. Cambazard and Catusse (2014) propose a methodology heavily based on modelling with both integer programming (IP) and Constraint Programming technologies for problem resolution. These authors mainly concentrate on solving the problem of assigning trains to departures and using an integer programming approach similar to that explained in this paper. Haahr and Bull (2014) propose two exact methods, IP and Column Generation, for solving the same sub-problem (called "Train Departure Matching Problem" in their paper). They report that solving the problem of assigning trains to departures exactly is very difficult, if not impossible, for a given set of instances. Most of the teams competing in the ROADEF/EURO Challenge 2014 proposed algorithms that rely on greedy procedures or integer programming or a combination of both. Modelling an entire problem, or a significant part of one, using IP is theoretically possible and has been achieved by a number of

competitors, yet the outcome proved incapable of producing satisfactory results on the given set of instances. IP techniques therefore are mainly used to solve only specific sub-problems. The breakdown of a problem into two dependent sub-problems, i.e. assignment and scheduling problems, is quite a natural step given the complexity of the initial problem and was implemented in most of the approaches presented. To the best of our knowledge, none of competitors conducted a local search (at least as a significant component of their research).

4 Two-phase approach

In our method, the problem has been broken down into two sub-problems, which are then solved sequentially. During the first phase, a train assignment problem is solved by combining a greedy heuristic procedure with integer programming (IP). The main objective here is to maximize the number of assigned departures while meeting technical constraints. Other objectives are taken into account as well, with the aim of obtaining "better" input for the subsequent phase. During the second phase, the train scheduling problem, which consists of scheduling the trains inside the station, is solved using a constructive heuristic model. The goal is to schedule as many assignments as possible, by using station resources and complying with all the constraints. An iterative improvement procedure is implemented in order to improve the resulting schedule.

4.1 Assignment problem

This section will describe the method adopted to solve the problem of matching (assigning) trains to departures. Assigning train $t \in \mathcal{T}$ to departure $d \in \mathcal{D}$ must meet the following technical constraints:

- The remaining distance before the maintenance of train t must be sufficient for departure d: $remDBM_t \geq reqD_d$.
- The time difference between arrival and departure must be large enough (for maintenance operations, parking,…).

We call an assignment (t, d) of a train $t \in \mathcal{T}$ to a departure $d \in \mathcal{D}$ *feasible* if the following holds:

1. $arrTime_t + maintTime(t, d) + addTime(t, d) \leq depTime_d$,
2. $remDBM_t \geq reqD_d$,

where:

- $maintTime(t, d)$ is the total maintenance duration required for scheduling t to d (0 if maintenance not required),
- $addTime(t, d)$ is an additional time necessary for parking and handling the train, *i.e.* in the case where the train is required to leave the arrivals platform before departure (non-immediate departure). The train may be parked either at the maintenance facility to undergo servicing or at any authorized resource before being scheduled for departure.

Additional time, $addTime(t, d)$, is a variable value to be determined; it is used to increase the chance of finding a feasible schedule, yet an excessive value of this variable may also decrease the number of assigned departures. This value has been experimentally set to lie within the range of 5–60 min. Only feasible assignments (t, d) are to be considered. Furthermore, the solution to the assignment problem must abide by the maintenance limit constraint

i.e. the total number of maintenance operations during any day must not exceed a given number $maxMaint$.

The following objectives are considered in the assignment phase:

1. maximize the number of assigned departures,
2. maximize the number of immediate departures,[1]
3. minimize the number of maintenance operations,
4. minimize the number of assignments with a large difference between departure time and arrival time (greater than 10 h for example).

These objectives are mixed and exact importance (weight) of each objective part will be given while describing the methods used for solving the problem. The reason for introducing objectives (2) and (3) is to minimize the use of track groups since minimizing track group use will obviously decrease the chance of conflict. Another goal of inserting (3) is to minimize the use of maintenance facilities, which are considered critical resources. The aim in avoiding long waiting time between arrival and departure is to minimize the use of parking resources. The following definitions will be used herein:

- $nmbM(t, d)$: the number of maintenances required to schedule train t to departure d (equals 0 or 1);
- $imm(t, d)$: equals 1 if d is immediate, 0 otherwise;
- $long(t, d)$: equals 1 if $depTime_d - arrTime_t > L$, where L is a parameter, 0 otherwise.

The assignment problem is made significantly more complex as the remaining distance for some trains is not known before scheduling the linked departures. The maximum number of maintenance operations per day constraint complicates the problem even more. A combination of greedy and integer programming (IP) algorithms has been implemented to solve this assignment problem.

4.1.1 Greedy procedure

The greedy procedure tries to match departures one by one. For each departure d, the best train is chosen in consideration of the defined objectives. The procedure can be formalised as follows:

- sort departures $d \in \mathcal{D}$ in ascending order with respect to departure time;
- for each departure $d \in \mathcal{D}$, find the "best" available train.

Departure assigned to train t will be denoted by $d(t)$ ($d(t) = -1$ in case no departure is assigned to t). After assigning train t to linked departure d, the value $remDBM$ of the corresponding linked train is updated according to the constraint. The exact choice of train for each departure is accurately described in the pseudo-code of the greedy assignment (Alg. 1). The following rules are informally applied when choosing the train for each departure d:

- consider only currently unassigned trains;
- whenever possible, choose an immediate assignment;
- assignments without required maintenance and trains with a small $remDBM$ are preferable for non-linked departures;

[1] Departure d covered by train t is said to be immediate if train t can be scheduled to d without leaving the platform.

Algorithm 1: Greedy assignment

```
1  bigM = 1 + max_{t∈T}(maxDBM_t)
2  Sort departures by time;
3  for d = 0 to |D| − 1 do
       // each departure
4      bestTrain ← −1;
5      minValue ← 2 × bigM;
6      for t = 0 to |T| − 1 do
7          if d(t) = −1 ∧ (t, d) is feasible ∧ maintenance limit not exceeded then
8              if imm(t, d) then
9                  value ← −bigM;
10             else
11                 if long(t, d) then
12                     value ← depTime_d − arrTime_t;
13                 else
14                     if d not linked then
15                         value ← remDBM_t + nmbM(t, d) × bigM;
16                     if d linked ∧ nmbM(t, d) > 0 then
17                         value ← remDBM_t − bigM;
18                     if d linked ∧ nmbM(t, d) = 0 then
19                         value ← −remDBM_t;
20             if value < minValue then
21                 minValue ← value;
22                 bestTrain ← t;
23     if bestTrain ≠ −1 then
24         t(d) ← bestTrain;
25         d(bestTrain) ← d;
26         Update linked arrival if needed;
27         Add maint. to all intervals containing [arrDay_{t(d)}, depDay_d] if required;
```

– in contrast to the previous rule, assignments with required maintenance and trains with a large $remDBM$ are preferable for linked departures;
– long waiting times between arrival and departure are not desirable.

The 'maximum number of maintenance operations per day' constraint is taken into account in the following manner. For each interval of days $[day_1, day_2]$, we define a current number of maintenance operations performed between days day_1 and day_2 by $m(day_1, day_2)$. Obviously, $m(day_1, day_2)$ must not exceed $(day_2 - day_1 + 1) \times maxMaint$. Each time a maintenance operation needs to be performed for assignment (t, d), the value $m(day_1, day_2)$ is updated for each interval of days $[day_1, day_2]$ containing $[arrDay_t, depDay_d]$, where $arrDay_t$ and $depDay_d$ represent arrival day of train t and the day of departure d respectively. Adhering to the given bound for each interval of days guarantees a feasible choice of days for each required maintenance operation; a simple procedure for choosing the exact day of maintenance, along with the proof of correctness, is given in Sect. 4.1.3. This same concept is found in the IP model, which enables a daily maintenance limit constraint to be represented linearly.

4.1.2 IP model

To enhance solutions to the assignment problem, the greedy procedure explained above is combined with a integer programming (IP) approach. The main difficulty in applying IP directly (independently) to solve the assignment problem involves linked departures. More specifically, the remaining distance before maintenance ($remDBM$) of some trains is not known before assigning the linked departures. The greedy procedure described in the previous section and IP have thus been combined as follows:

- The assignment problem is solved by a greedy procedure;
- Linked departure assignments are fixed (in updating the data on linked trains);
- The resulting assignment problem is solved once again with IP.

Let $\mathcal{F} = \{(ft_1, fd_1), (ft_2, fd_2), \ldots, (ft_k, fd_k)\} \subseteq \mathcal{T} \times \mathcal{D}$ be the set of fixed assignments. Let the set of possible assignments be defined as:

$$\mathcal{S} = \{(t_1, d_1), (t_2, d_2), \ldots, (t_n, d_n)\} \subseteq \mathcal{T} \times \mathcal{D},$$

where:

- each assignment (t_i, d_i) is feasible;
- none of the departures $d_1, \ldots, d_n$ is linked;
- none of the trains $t_1, \ldots, t_n$ has been assigned to linked departure i.e.

$$\forall i \in \{1, \ldots, n\},\ j \in \{1, \ldots, k\}\ \ t_i \neq ft_j.$$

For each pair (t_j, d_j) in $\mathcal{S}$, a decision variable x_j is defined: $x_j = 1$ if t_j is assigned to d_j, $x_j = 0$ otherwise. All the objective parts are merged into a single objective function by applying a non-negative weight for each of them. The objective function to be maximized is formalized as follows:

$$\sum_{j=1}^{n} x_j \times \big(assignmentWeight + durationWeight \times \big(1 - long(t_j, d_j)\big) + immWeight \times imm(t_j, d_j) + maintWeight \times \big(1 - nmbM(t_j, d_j)\big)\big)$$

The weights are chosen experimentally, and all results in the paper have been obtained with the following choices: $assignmentWeight = 1000$, $durationWeight = 100$, $immWeight = 10$ and $maintWeight = 1$. Next, we define the constraints included in this model. Let $nmbFixed(day_1, day_2)$ be the number of maintenance operations between days day_1 and day_2 required by fixed assignments i.e.

$$nmbFixed(day_1, day_2) = \sum_{i=1}^{k} nmbM(ft_i, fd_i) \times \mathbb{1}_{[arrDay_{ft_i}, depDay_{fd_i}] \subseteq [day_1, day_2]}$$

The constraint on maintenance is formulated by the following:

$$\forall [day_1, day_2] \quad \sum_{j=1}^{n} nmbM(t_j, d_j) \times x_j \times \mathbb{1}_{[arrDay_{t_j}, depDay_{d_j}] \subseteq [day_1, day_2]} \leq (day_2 - day_1 + 1) \times maxMaint - nmbFixed(day_1, day_2)$$

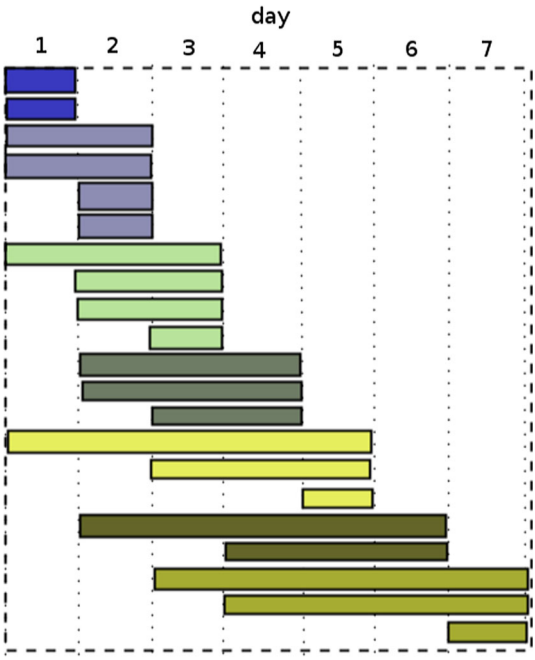

Fig. 2 Assignments ordered by departure day and arrival day

4.1.3 Choosing maintenance days

As described earlier, the maximum number of maintenance operations per day constraint is met by complying with the limit for each interval of days $[day1, day2]$, while updating the number of maintenance operations for each interval that contains $[arrDay_t, depDay_d]$ when servicing has to be performed for the assignment (t, d). The exact day for each maintenance operation still needs to be determined. A simple procedure, along with its proof of correctness, is presented in this section. This procedure functions as follows:

- sorting assignments (requiring maintenance) in ascending order by departure day and then by arrival day (example is given in Fig. 2);
- for each assignment (t, d) in a sorted list:
  - choose the first available day for maintenance, i.e. the first day in $\{arrDay_t, \ldots, depDay_d\}$ for which the maintenance limit has not been reached.

We will now prove the correctness of this procedure. Let $M = \{m_1, m_2, \ldots, m_k\}$ be the set of all assignments requiring maintenance. Next, let A_i and D_i be the arrival and departure days of assignment m_i respectively. We can now write $m_i = (A_i, D_i)$, $A_i, D_i \in \{1, 2, \ldots, nDays\}$.

Claim If the following inequality holds for each interval of days $[day_1, day_2]$:

$$\sum_{i=1}^{k} \mathbb{1}_{[A_i, D_i] \subseteq [day_1, day_2]} \leq (day_2 - day_1 + 1) \times maxMaint \tag{1}$$

then the assignment of maintenance days using the procedure described above meets the 'maximum number of maintenance operations per day' constraint.

Proof This claim will be proven by way of contradiction. Let $m_j = (A_j, D_j)$ be the first assignment for which a servicing day cannot be chosen by following the given procedure and $M_{j-1} = \{m_1, m_2, \ldots, m_{j-1}\}$ be the set of assignments with servicing days already set. This set-up means that the maintenance limit is reached for each day in $[A_j, D_j]$. (*) Let day_0 be the first day such that the maintenance limit is reached for each day in interval $[day_0, D_j]$ (the existence of day_0 is obvious and $day_0 \leq A_j$). Clearly,

- $day_0 = 1$ or
- the limit is not reached for $day_0 - 1$. (**)

Let $F = \{f_1, f_2, \ldots, f_l\} \subset M_{j-1}$ be the set of assignments that have already been assigned to one of the days in $[day_0, D_j]$ ($l = (D_j - day_0 + 1) \times maxMaint$, since the limit has been reached for the entire interval). For each $f_i = (A_{fi}, D_{fi}) \in F$, we have:

- $A_{fi} \geq day_0$: otherwise, the chosen servicing day would not be greater than $day_0 - 1$ because of the assignment rule and (**),
- $D_{fi} \leq D_j$: as a result of the sorting order.

The same holds for m_j: $A_j \geq day_0$ and $D_j \leq D_j$. Let us now consider the set $U = F \cup m_j$. As previously shown, the maintenance interval for each element of U lies in $[day_0, D_j]$, thus:

$$\sum_{i=1}^{k} \mathbb{1}_{[A_i, D_i] \subseteq [day_0, D_j]} \geq \sum_{i \in U} \mathbb{1}_{[A_i, D_i] \subseteq [day_0, D_j]} = |U| = (D_j - day_0 + 1) \times maxMaint + 1$$

which is contradictory to the main assumption (1).

4.2 Scheduling problem

The goal of the second algorithm part is to schedule the assignments generated by the first phase inside the station while abiding by all resource constraints. Trains must move through the network/graph of inter-connected resources. All types of resources and constraints associated with the trains are given in the problem description provided in Sect. 2. A constructive procedure has been implemented here to solve the scheduling problem. The output schedule is then improved by an iterative procedure based on a local search. There are three scheduling options for each train $t \in \mathcal{T}$:

1. t is scheduled for departure $d \in \mathcal{D}$;
2. t is parked inside the station until the end of the planning time frame without being scheduled for any departure;
3. t is cancelled.

The schedule, possibly an empty one, must be given for each train $t \in \mathcal{T}$. All resources used by the train must be specified, along with the exact time of entering and leaving each resource. The greedy procedure schedules the trains one by one, in a defined order (ordering will be addressed later). A complete schedule for the train is output before scheduling the next train. Nevertheless, all trains share the same resources and all constraints need to be complied with over the entire scheduling procedure.

4.2.1 Possible train movements

Modelling the scheduling problem exactly, i.e. taking all possible resource choices into consideration at every possible time instant, is not realistic given the size and structure of

the instances proposed by SNCF. We have therefore limited possible train movements to the following:

- arriving at the platform via a given set of track groups (arrival sequence),
- departing from the platform via a given set of track groups (departure sequence),
- move from arrivals platform to yard,
- move from arrivals platform to facility,
- move from parking (facility, yard) to departure platform,
- move from yard to facility,
- move from facility to yard.

The train schedule will, for each train movement, specify the set of resources deployed with the exact resource enter and exit times. The connected set of resources used while moving the train from one place to another will be called $path$. $P = (R_1, R_2, \ldots, R_k)$ denotes a path connecting resources R_1 and R_k that starts at R_1, visits resources $R_2, R_3, \ldots, R_{k-1}$ and then ends at R_k. Two consecutive resources on a path must be connected by a gate. The use of path P for given entry and exit times on each resource will be called $travel$. To simplify the scheduling procedure, the time spent on each intermediate resource on a path $(R_2, \ldots, R_{k-1})$ is always a minimum. In the case of track groups, this time is set equal to $trTime$, while in the case of other resources it equals to the constant value (minimum duration of resource usage) given as input. The travel of a given train is thus fully determined by the designated path and the starting time. Travel using path P and starting at time h will be denoted $T(P, h)$. To conclude, the schedule of a train is represented as a set of travels $\{T(P_1, h_1), T(P_2, h_2), \ldots T(P_l, h_l)\}$. All paths potentially used for any movement are constructed before the start of the scheduling procedure. This set of paths includes those for each pair of resources (r_1, r_2), such that r_1 and r_2 are of different types, with neither of them being a track group. Paths are sorted by length (i.e. total number of resources) for each pair (r_1, r_2). Furthermore, should there be many paths between two resources, only the shortest ones are to be kept (15 shortest for example). This simple pre-processing step simplifies implementation considerably.

The scheduling procedure seeks to identify a feasible schedule for a given train with small number of travels. The choice of movements depends on the type of operations that need to be carried out (e.g. maintenance), total time to be spent at the station, *etc.* Train movements are planned one by one. If no feasible travel can be found at some point during the planning procedure, then the train will be cancelled, i.e. no attempt will be made to modify the previous movements.

4.2.2 Resource consumption and travel feasibility

Resource consumption is tracked by recording the set of all previous visits for each resource in the station. A visit to a resource has the following attributes: entry time and side, exit time and side, train length, and entry and exit gates. Each time the resource needs to be visited by a train, all constraints for a given resource are checked and the visit is only allowed if found to be feasible. Then, verifying the feasibility of travel $T(P, h)$ simply requires checking the visit feasibility on each resource in path P, with corresponding entry and exit times being uniquely determined by h.

4.2.3 Time spent on a resource between two travels

One of the difficulties involved in train scheduling is to determine an exit time for the last resource in each travel. Knowing the exit time on the last resource is required in order to check constraints regarding this particular resource. An exact exit time is often not known before the next travel is planned. The following strategy was used to deal with this issue:

1. If the last travel resource is a yard: the exit time is equal to departure time if the train is to be scheduled for a departure; otherwise, the exit time is the end of the planning horizon.
2. If the last travel resource is a facility (for either parking or maintenance).
   - same as (1) in the case maintenance is not required,
   - otherwise, exit time is equal to the minimum between departure time and facility entry time plus 12 h.
3. If the last resource of travel $T(P_i, h_i)$ is a platform: travel $T(P_i, h_i)$ and $T(P_{i+1}, h_{i+1})$ are planned together. This step is equivalent to planning a single travel, with possible routes/paths being a combination of two paths (paths for travels $T(P_i, h_i)$ and $T(P_{i+1}, h_{i+1})$), yet time spent on the platform is no longer fixed and needs to be determined.

4.2.4 Travel starting time

An important decision to be made when scheduling each train is the starting time for each travel. Some starting times are fixed, such as the time of arrival and departure, while others are to be selected from a feasible set of time instants. An ideal starting time, $idealST$, will be defined for each travel. We can always calculate the earliest and latest possible travel times, est and lst, which depend on the time constraints such as minimum resource times, travel duration, fixed arrival and departure times, etc. The ideal travel starting time depends on the type of travel; for our purposes, the following was used:

- if arrival train t needs to be moved from platform (to yard or facility) then the ideal starting time will minimize the dwell cost on the platform, i.e. $idealST = arrTime_t + idealDwell_t$;
- if train t, parked at a yard or facility, needs to be moved to the platform for departure d then the ideal starting time will minimize the dwell cost on the platform, i.e. $idealST = depTime_d - idealDwell_d - travelDur$, where $travelDur$ is the duration of travel;
- if the train is to be moved from one parking resource to another (i.e. from yard to facility or vice versa) then the ideal starting time is the earliest possible starting time, est.

Once the ideal travel starting time, $idealST$, has been determined, the next step seeks to choose a starting time, between the earliest and latest possible, as close as possible to $idealST$. Formally speaking, the selected starting travel time, h, is the first one from the set

$$\{idealST, idealST - \delta, idealST + \delta, idealST - 2\delta, idealST + 2\delta, \ldots\},$$

such that $h \in [est, lst]$ and travel $T(P, h)$ is feasible for some path P. Parameter δ is chosen from the interval $[10s, 60s]$.

4.2.5 Dealing with parking capacity

Since the station has limited capacity, it is not possible for the number of trains arriving at the station to significantly exceed the number of departures from the station. Consequently, most

trains associated with arrivals must be scheduled to a departure. However, some trains may remain at the station until the end of the planning horizon, though this number is typically much smaller than the number of trains scheduled to a departure. Furthermore, if station resources are critical, it is not desirable to consume them with the trains not scheduled for any departure, which could potentially disable the departure scheduling of some trains. We have therefore used the following simple heuristic in the scheduling procedure:

1. *planning departures*: schedule each assigned train t ($d(t) \geq 0$) and if the train cannot be successfully scheduled for departure $d(t)$, then cancel it;
2. *park unassigned and cancelled trains* at the very end of the procedure (after optimizing the solution).

It should also be noted that assignments with too much time between arrival and departure are not desirable from the standpoint of yard capacity, which is taken into account during the assignment phase.

4.2.6 Choosing gates: avoiding conflicts on track groups

The main difficulty with this problem, from our experience, lies in effectively choosing the gates for each track group to enter and exit, as this gate selection will allow more trains to travel on the track groups without conflict. As defined in Sect. 2, conflicts on track groups are prohibited. For each travel $T(P, h)$ by a train t, a set of entry/exit gates on each track group in P needs to be determined. As with all other resources, the exact entry and exit times for each track group are known if the starting time, h, of the travel is given.

Let n_1 be the number of possible gates to choose for an entering track group $TG \in P$, and n_2 the number of possible gates for exiting. We then have a total of $n_1 \times n_2$ possible moves to choose from. It is simple to check whether or not the selected move conflicts with any of the previous moves on the track group. For this purpose, as with any other resource type, we have kept a set of all visits (moves) to the track group, and only those moves that do not conflict with any moves in the given set are to be allowed. A set of entry/exit gates without conflicts must be determined for the entire travel $T(P, h)$, which means that a feasible move needs to be found on each track group in P. Consequently, the number of possible combinations of moves becomes greater. One should notice that choice of exit gate on one resource automatically determines the choice of entering gate on the next one. A simple depth-first search (DFS) to find a feasible set of moves for travel has been used. This procedure explores all possible combinations of moves (one move for each track group in P) until a feasible one (without conflicts) is found. The most basic way of using a DFS procedure is to begin with the first possible gate on each track group and increase the gate index, according to a depth-first sequence, whenever a feasible choice has not been found. This manner of choosing the gates is not necessarily a good one as regards track group usage. To improve the choice of gates, let us attempt to identify a different order for exploring the possibilities in a DFS procedure. Formally, for each path $P = (R_1, R_2, \ldots, R_n)$, a "preferred" entry gate on each resource in P will be defined and the DFS procedure will explore all possibilities by starting with a preferred gate on each resource. The set of preferred gates for travel $T(P, h)$ is determined according to the first and last resources, more specifically R_1 and R_n in P, and depending on the positions of these resources relative to the neighbouring track groups, R_2 and R_{n-1}. It should be noted that the majority of travels start or end at the platform. Consequently, the most critical track groups are those either connected to or close to the platforms. We have therefore decided to define the preferred gates solely according to the relative position of the platform with respect to the connected track group. If R_1 is the $j - th$ of N platforms connected to

track group R_2 (according to gate indices) and $g_1, g_2, \ldots, g_k$ are the gates from R_i to R_{i+1} ($2 \leq i < n$), then a preferred gate is g_l, where: $l = \frac{j \times k}{N}$. The same rule is applied if R_n is a platform. For example, if the chosen platform is the top platform, then it is only natural to choose the top gate on each resource in path P. We have conducted several experiments with a more complicated choice of gates, however the results obtained only changed slightly and were not necessarily always better. Moreover, the local search procedure described at the end of this section will question this choice of gates.

4.2.7 Virtual visits

One of the difficulties in avoiding conflicts is not knowing the "future traffic", i.e. overall track group use. This issue is particularly important when choosing the starting times of travels without a fixed starting time (i.e. all travels except arrivals and departures). Very often, many different possibilities are feasible and just one has to be chosen, although choosing any one of them might potentially block more trains yet to be scheduled than choosing another one. We have introduced the concept of "virtual visits" to improve the starting time of each travel. Virtual visits can be viewed as the potential visits capable of occurring on the track groups in the future. Virtual visits will be generated for each arrival and each matched departure (by the assignment procedure) and then taken into account when choosing the starting times and gates for the travel. The set of virtual visits V is built as follows:

- for each arrival $t \in \mathcal{T}$ and each matched departure $d \in \mathcal{D}$
  - randomly choose a compatible platform p,
  - find a set of gates for arrival/departure (sequence + platform) with a minimum number of conflicts with V (as explained in previous section),
  - add the matching set of visits to the track groups to V.

The set of virtual visits is computed at the start of the scheduling phase, before scheduling any train. Next, during train scheduling, the starting time of each travel not matching an arrival or departure is selected in order to minimize the number of conflicts with virtual visits. The virtual visits of train t are removed from V when the scheduling procedure for t has been completed (t is scheduled for departure, parked or cancelled).

4.2.8 Scheduling order

Trains are to be scheduled independently and consecutively, one by one. Some trains however may have a higher scheduling priority than others. For example, cancelling a train assigned to a linked departure could cause cancellation of the linked trains, and some trains use far fewer resources than others, etc. Trains are therefore scheduled in the following order:

1. Assigned trains,
   (a) trains that may be departing immediately: only arrival and departure gates are used,
   (b) trains assigned to linked departures: uncovering a linked departure may cause more uncovered departures,
   (c) trains that do not require any maintenance,
   (d) remaining assigned trains,
2. Unassigned trains.

Constraints on linked departures are complied with, e.g. if train $t_2 \in \mathcal{T}$ is linked to departure $d \in \mathcal{D}$ and $t_1 \in \mathcal{T}$ is assigned to d, then t_1 must come before t_2 in the given order.

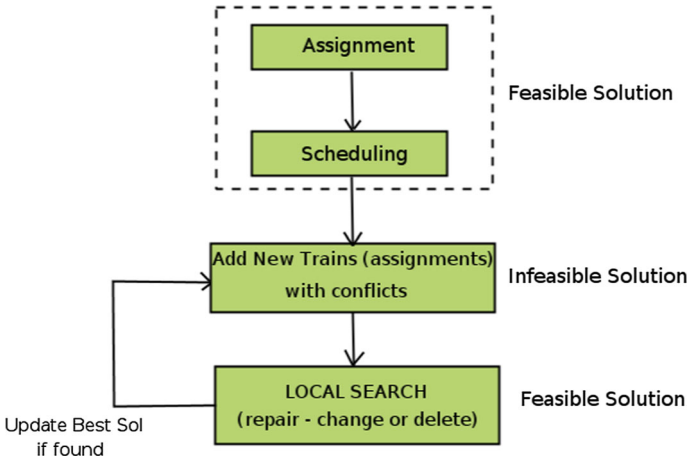

Fig. 3 Solution process

4.3 Iterative improvement procedure

This section recommends an iterative procedure for improving the schedule. This procedure operates as follows:

- (1) schedule more trains by allowing conflicts in the track groups,
- (2) resolve conflicts by means of a local search,
- repeat steps (1)–(2) until the stopping criteria are met.

The entire solution procedure is illustrated in Fig. 3.

4.3.1 Feasible to infeasible solution with more trains

The first step of this improvement procedure consists of adding more trains (and departures) to the feasible schedule by allowing conflicts. For each train added, the allowed track group conflicts are limited to a given number (e.g. a maximum of 3 conflicts per train). This scheduling procedure is the same as the one previously explained, but without adhering to the constraint on track group conflicts. All other constraints are to be met. An infeasible solution generated in this manner will serve as input for the local search procedure described below.

4.3.2 Local search to resolve track group conflicts

An infeasible solution is repaired by means of a local search procedure. The aim of this procedure is to change the choice of gates in order to reduce the number of conflicts to zero. The entry and exit times are to remain unchanged for each visit, as is the list of resources allocated for each train. Accordingly, a complete schedule for the train will either remain the same or be deleted (in the case of cancelling the train). The initial configuration (solution) is an infeasible set of visits on track groups. A visit is represented by a pair of gates $(g1, g2)$. The configuration is denoted by $\mathcal{V} = \{(g1_1, g2_1), \ldots, (g1_n, g2_n)\}$ and the initial one by $\mathcal{V}^0 = \{(g1_1^0, g2_1^0), \ldots, (g1_n^0, g2_n^0)\}$. The domain of each variable $g1_i$ $(g2_i)$ includes all gates connecting the same pair of resources as $g1_i^0$ $(g2_i^0)$ and NULL value. A visit corresponding to

$(g1, \text{NULL})$, $(\text{NULL}, g2)$ or $(\text{NULL}, \text{NULL})$ is called a partial visit. It is assumed that no conflict occurs with a partial visit. A configuration $\mathcal{V}$ is called *partial* if it contains a partial visit.

Remark Two successive visits of the same train $(g1_i, g2_i)$ and $(g1_j, g2_j)$ share a common gate, i.e. $g2_i = g1_j$. For these cases, the local search procedure will perform the same modification at both gates. The objective of the local search procedure is to minimize the number of cancelled trains. A train is cancelled if one of its visits is partial.

Greedy procedure to resolve track group conflicts
The first part of a local search is the greedy procedure to clear conflicts by deleting gates, i.e. setting the gate values to NULL. The objective here is to compute a partial, but feasible, configuration (set of visits). The heuristic is simple: delete the gate that will decrease conflicts by the greatest number until conflicts no longer exist.
Tabu search on the partial feasible configuration
The tabu search procedure starts from a partial, but feasible, configuration of gates given by the previous procedure. The goal is to assign gate values to partial visits while keeping the configuration feasible. The following two *elementary moves* are carried out:

- `ADD` gate means one of 2 possible moves:
  - $(\text{NULL}, g2_i) \Rightarrow (g1_i, g2_i)$ ($g1_i \neq$ NULL and $g1_i$ leads to the same neighbour resource as $g1_i^0$)
  - $(g1_i, \text{NULL}) \Rightarrow (g1_i, g2_i)$ ($g2_i \neq$ NULL and $g2_i$ leads to the same neighbour resource as $g2_i^0$)
- `DROP` gate means one of 2 possible moves:
  - $(g1_i, g2_i) \Rightarrow (\text{NULL}, g2_i)$ (*delete g1*)
  - $(g1_i, g2_i) \Rightarrow (g1_i, \text{NULL})$ (*delete g2*)

These modifications are also applied to the next/previous visit of the given train (see remark in Sect. 4.3.2). The local search move consists of a single `ADD` move and, should conflicts occur, is to be followed by a few `DROP` moves in order to clear the conflicts. Deleting a gate g_i (`DROP`) is allowed only if g_i has not been added for a given number of iterations (i.e. if setting g_i to NULL is not tabu). The number of iterations for which deleting a gate is tabu equals the frequency of adding this gate. The current configuration is evaluated by the following hierarchical function:

1. number of trains cancelled,
2. number of deleted gates, i.e. the number of gates with a NULL value.

At each iteration, a non-tabu move that minimizes this function is performed. If there are two or more moves with the same objective, then a random choice is made. This process is repeated until no non-tabu move exists or until a maximum number of moves without improvement has been reached. In all reported experiments this limit has been set to 300.

An illustration of the objective function change during this improvement procedure is given in Fig. 4. It can be noted that most of the improvement occurs during the early stage and not many new trains are added at the end. The improved results due to the local search are illustrated in Fig. 5.

MIP instead of Local Search The problem of repairing the conflicts explained above, and solved by local search procedure, may also be solved in different ways. One possible approach that we tested is to define a problem as a mixed integer program (which is not too complicated), with the objective of determining the maximum subset (in terms of number of

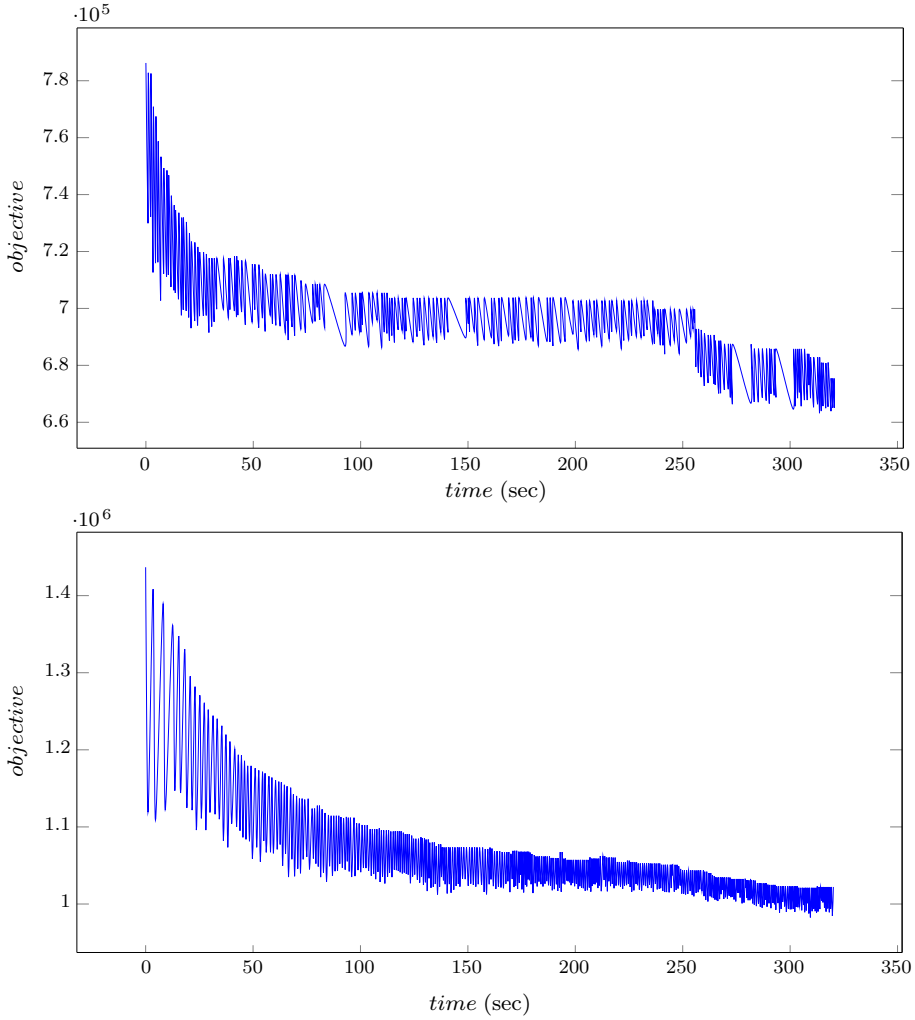

Fig. 4 Objective function oscillation during improvement phase for instances B1 and X4. The *X*-axis represents time elapsed since improvement procedure start, while the *Y*-axis represents the value of the objective function after adding new trains to the feasible solutions and after the local search procedure

scheduled trains) of train schedules without conflicts on track groups. The overall improvement procedure remains the same, we only try to "repair conflicts" by using mixed integer programming instead of local search. If we consider a single improvement phase iteration (add trains + repair) i.e. solving the problem—given a set of train schedules with conflicts, choose the gate values so that a minimum number of departures/arrivals is cancelled—it is clear that MIP will produce better (or at least the same) solutions than local search. However, using MIP instead of local search has not made any (or negligible) improvements to the final results using presented improvement framework, particularly for a 10-min running time when using MIP usually produces poorer results. The main reason for this is the fact that after many improvement phase iterations (e.g. 100) solution values obtained by local search come very close to the values of infeasible solutions, i.e. only a few trains can be added to

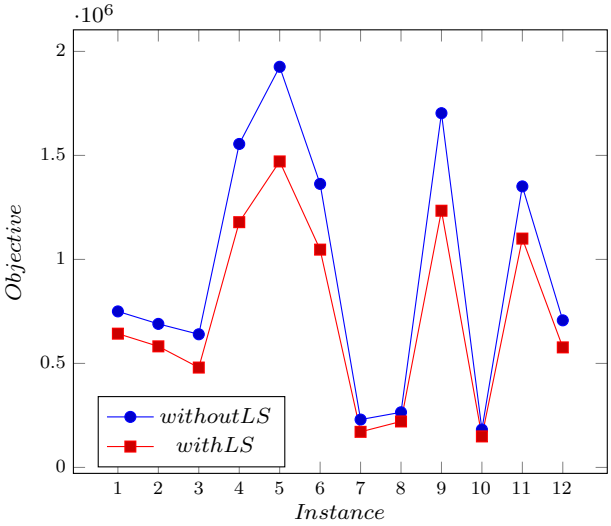

Fig. 5 Improvement by local search (instances B1–B12)

the feasible solutions. In addition, many more iterations can be done in the same running time if local search is used, which is crucial for some instances. Similar conclusions can be drawn when MIP is used with a given running time limit (thus, transforming an exact solution approach into an heuristic one).

4.4 Final algorithm

The pseudo-code of the final algorithm is given in Algorithm 2.

Algorithm 2: *FinalAlgorithm*

```
Solve assignment problem (greedy/IP);
Determine maintenance days;
Sort trains for schedule (see schedule order);
Add virtual visits;
for t ∈ 𝒯 do
    schedule train t;
    if train is not scheduled to departure cancel it;
repeat
    add trains with conflicts;
    repair conflicts (local search);
until stopping criteria is met;
Park non-scheduled trains;
```

5 Evaluation and computational results

The method has been tested on the official set of competition instances provided by SNCF. This test data consists of two sets of instances, i.e. dataset B and dataset X. The first set was made available during the competition, while dataset X is a hidden set of instances provided

Table 1 Instances characteristics

Inst	$nDays$	$\|\mathcal{T}\|$	$\|\mathcal{D}\|$	$maxMaint$	LDep
B1	7	1235	1235	30	475
B2	7	1235	1235	30	475
B3	7	1235	1235	60	0
B4	7	1780	1780	50	722
B5	7	2153	2153	60	720
B6	7	1780	1780	50	722
B7	1	304	304	100	143
B8	1	304	304	100	143
B9	7	1967	1967	100	860
B10	1	196	196	20	60
B11	7	1122	1122	20	486
B12	3	570	570	20	263
X1	7	1235	1235	30	475
X2	7	1499	1499	40	475
X3	7	1235	1235	30	475
X4	7	1780	1780	50	722
X5	7	1780	1780	50	722
X6	7	1780	1780	100	0
X7	7	1967	1967	100	860
X8	3	905	905	100	445
X9	3	905	905	100	445
X10	1	196	196	20	89
X11	7	1122	1122	20	486
X12	3	570	570	20	263

to test algorithm robustness and then made publicly available after the end of the competition. The basic characteristics of instances are given in Table 1: horizon length ($nDays$), number of arrivals ($|\mathcal{T}|$) and departures ($|\mathcal{D}|$), maximum number of maintenance operations in a day ($maxMaint$) and number of linked departures (LDep) are reported for each instance. The algorithm has been implemented in C++ and compiled using Linux gcc 4.7.2 compiler in Ubuntu 12.10. All tests were performed on a computer with an Intel Core i7-3770 CPU 3.40 GHz processor. The IP model proposed for the assignment phase was solved using the IBM ILOG CPLEX 12.6 solver.

The algorithm sequentially produces independent solutions until the time limit is exceeded and retains the best one. The results obtained on datasets B and X, in a 10-min running time, are listed in Table 2; average and best result over 20 runs with different initial seeds for the random number generation are reported. We also report the results submitted for the competition, along with the best results obtained by other competitors (our submitted program produced less good results due to the fact that IP model given in Sect. 4.1.2 was not included).

There is clearly a significant difference between the average and best solutions, sometimes more than 5%. This is due to the randomness in the heuristics proposed. However, running the algorithm for more seeds (e.g. 100 instead of 20) or with larger time limit will not yield a significant further improvement for a given set of instances.

Table 2 Results on data sets B, X: average and best values from 20 runs of 10 min are reported, as well as competition results

Instance	Challenge		With MIP included (20 runs)	
	Our result	Best others	Average	Best
B1	**699,750**	1,026,208	643,651	622,879
B2	**636,550**	980,607	582,000	561,579
B3	**545,974**	976,754	480,960	462,848
B4	**1,263,764**	1,671,107	1,179,584	1,142,660
B5	**1,573,290**	2,060,485	1,471,631	1,417,578
B6	**1,097,572**	1,586,010	1,047,493	993,580
B7	**168,369**	257,986	171,257	167,993
B8	**213,190**	291,216	221,480	208,573
B9	**1,332,256**	1,719,646	1,234,501	1,130,644
B10	168,457	**155,100**	149,301	142,600
B11	1,192,687	**1,142,072**	1,100,200	1,076,260
B12	620,527	**571,497**	577,000	556,853
X1	**790,506**	1,036,393	712,260	683,939
X2	**1,176,901**	1,323,148	890,666	874,014
X3	**735,579**	1,040,464	669,864	655,220
X4	**1,109,468**	1,608,134	1,026,635	983,693
X5	**1,012,268**	1,519,608	925,955	884,348
X6	**943,024**	1,560,462	822,780	768,395
X7	**1,642,024**	1,914,783	1,569,772	1,543,090
X8	**534,889**	777,363	560,170	553,427
X9	**732,818**	871,027	748,539	711,395
X10	193,210	**184,022**	180,707	168,407
X11	1,107,732	**988,996**	1,018,810	965,892
X12	501,218	**467,605**	477,634	455,736

Best competition results are shown in bold

The percentage of arrivals and departures cancelled varies from 11.90% on the B3 instance to 44.21% on B12, the average value being 26.17%. Train cancellation is mainly due to the fact that track group conflicts are prohibited. These conflicts are not always the actual conflicts due to a "black-box" approach introduced to model track groups in the current problem. The approach does not appear to lead to a solution that can be used in practice, due to a large number of cancelled arrivals and departures. As previously stated, we have been trying to solve the problem 'as posted' in the competition, adhering to all the given constraints to obtain the best possible result. However, the problem needs to be formulated differently in order to recommend solutions that can be used in practice. It is important to mention that the original problem formulated at the start of the competition allowed track group conflicts and there was a penalty in the objective function for each conflict. But the organizers changed this in a later phase.

6 Conclusion

This rolling stock unit management problem on a railway site is extremely difficult to solve for several reasons. Most induced sub-problems, such as the train assignment, scheduling, track group conflicts and platform assignment, are indeed complicated. In order to solve this problem, we implemented a two-phase approach that combines exact and heuristic methods. A natural way of approaching the problem consists of dividing it into two sub-problems, the first consisting of matching (assigning) trains to departures and the second consisting of planning train movements (scheduling) inside the station, and then solving both sequentially. It would also be quite natural to consider a break down of the problem by days (and subsequently combining the days), as there are hardly any arrivals and departures at night. However, the experiments we performed have shown that the quality of the solutions decreases, while not greatly increasing algorithm simplicity or computational time. This was the case, in particular, for the first version of the problem (i.e. with conflicts on track group as soft constraints), where many more trains have not been assigned to any departure. It should also be noted that the assignment procedure presented in this paper tends to schedule the arrival and departure of a train on the same day.

The presence of linked departures and a constraint on the daily maintenance limit further complicate the assignment problem. Otherwise, the problem could be solved in a polynomial time by means, for instance, of the maximum weighted matching algorithm. A common strategy for overcoming these constraints calls for modelling the assignment problem as a mixed integer program. The number of linked departures in the given set of instances however makes the IP model unworkable due to the tremendous number of variables to be generated in order to cope with the linked departures. A greedy procedure has therefore been used to solve the assignment problem. The departures are sorted by departure time and matched to the trains one by one, in a greedy manner. A simple function was used to evaluate the quality of assignment (t, d). Solutions to the assignment problem are improved by combining a greedy procedure with IP, whereby all train assignments with linked departures obtained by the greedy procedure are fixed and the remaining assignment is solved by IP. We recommended the simple idea (and proved its correctness) of modelling the constraint on the daily maintenance limit as a linear one.

The second step in solving the problem is to plan train movements inside the station while complying with all resource constraints and cancelling as few trains as possible. This problem is very complex and contains a huge number of decision variables (resources, gates and entry/exit times must be specified for each train); hence, modelling the problem exactly and solving it efficiently are likely to be impossible. We therefore opted for a constructive procedure to schedule the trains. They are scheduled sequentially, one by one, after being ordered by a given set of criteria. The possible train movements are restricted to just a few of critical importance in order to reduce problem complexity. For this purpose, a set of potential paths that trains are allowed to use was compiled at the beginning of the procedure. A concept of virtual visits was offered to expand the choice of starting times for each travel in order to address track group conflicts. An iterative procedure was adopted to improve scheduling phase solutions, by allowing infeasible solutions as regards track group conflicts and then resolving these solutions by means of a local search. The introduction of both virtual visits and an iterative improvement procedure helped greatly improve these solutions. The algorithm described in this paper was ranked first at the ROADEF/EURO Challenge 2014.

References

Cambazard, H., & Catusse, N. (2014). Roadef challenge 2014: A modeling approach, rolling stock unit management on railway sites. IFORS 2014, Barcelona 13–18 July.

Corman, F., D'Ariano, A., Pacciarelli, D., & Pranzo, M. (2010). A tabu search algorithm for rerouting trains during rail operations. *Transportation Research Part B: Methodological*, *44*(1), 175–192.

Freling, R., Lentink, R. M., Kroon, L. G., & Huisman, D. (2005). Shunting of passenger train units in a railway station. *Transportation Science*, *39*(2), 261–272.

Haahr, J., & Bull, S. H. (2014). Exact methods for solving the train departure matching problem. IFORS 2014, Barcelona 13–18 July.

Huisman, D., Kroon, L., Lentink, R., & Vromans, M. (2005). Operations Research in Passenger Railway Transportation. No ERS-2005-023-LIS, ERIM Report Series Research in Management, Erasmus Research Institute of Management (ERIM).

Lentink, R. M., Fioole, P. J., Kroon, L. G., & van't Woudt, C. (2003). Applying operations research techniques to planning of train shunting. Technical Report ERS-2003-094-LIS, Erasmus University Rotterdam, Rotterdam, The Netherlands.

Ramond, F., & Marcos, N. (2014). Trains don't vanish! ROADEF EURO 2014 challenge problem description, 2014. https://hal.archives-ouvertes.fr/hal-01057324.