A Greedy Approach for a Rolling Stock Management Problem using Multi-Interval Constraint Propagation

Hugo Joudrier, Florence Thiard

A Greedy Approach for a Rolling Stock Management Problem using Multi-Interval Constraint Propagation – ROADEF/EURO Challenge 2014

Hugo Joudrier Florence Thiard

30 July 2016

Abstract

In this article we present our contribution to the Rolling Stock Unit Management problem proposed for the ROADEF/EURO Challenge 2014. We propose a greedy algorithm to assign trains to departures. Our approach relies on a routing procedure using multi-interval constraint propagation to compute the individual schedules of trains within the railway station. This algorithm allows to build an initial solution, satisfying a significant subset of departures.

1 Introduction

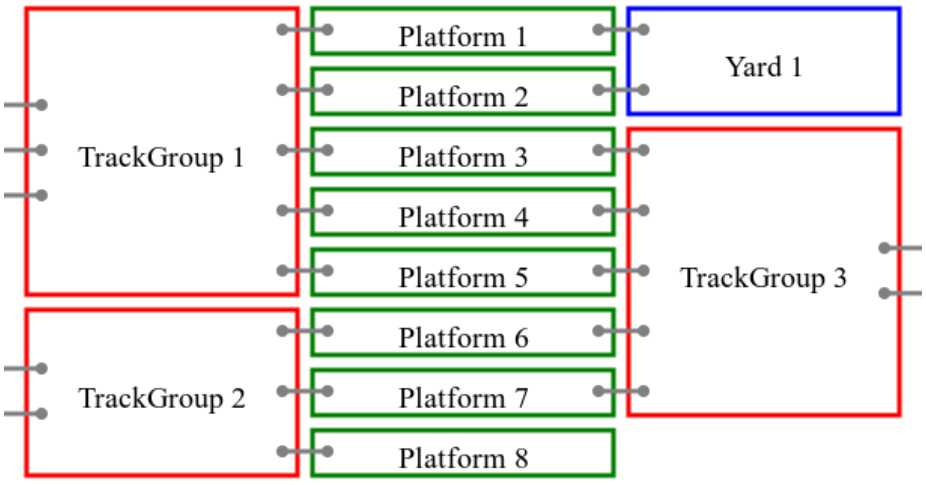


Figure 1: Example instance: railway station

In this paper, we present our approach of the Rolling Stock Unit Management problem presented in the ROADEF/EURO Challenge 2014 (Ramond and Nicolas 2014). This problem combines in a single formulation various subproblems of rolling stock management, such as assignment of trains to departures, platform assignment, routing inside the station, planning of maintenance operations... We propose a greedy algorithm to build an initial solution, allowing an incomplete coverage of arrivals and departures. This solution could serve as a basis for an optimization algorithm. Compared to the proposed problem, we make some simplifications by forbidding maintenance, junction and disjunction operations (note that train convoys might be used as such), which limits the number of coverable departures. Our solution is based on a greedy progressive assignment algorithm to assign arrivals (and corresponding trains) to departures, and a routing algorithm using multi-interval constraint propagation to prevent conflicts with already scheduled trains while keeping as much flexibility as possible.

The rest of this paper is organized as follows: Section 2 presents a simplified model of the rolling stock management problem presented in the ROADEF/EURO Challenge 2014. Section 3 describes the pre-processing phase and the model used to represent the structure of the station. Section 4 details the multi-interval routing algorithm, while Section 5 describes the general assignment procedure. Note that the assignment problem is not treated as a whole, separately from the routing problem, but progressively. Finally Section 6 presents an overview of the results and some perspectives. In the following, we will use the concepts and notations introduced by Ramond and Nicolas (2014).

2 Simplified model

The original formulation, as presented by Ramond and Nicolas (2014), aimed at treating the rolling stock management problem as a whole, integrating constraints and costs of various nature. Focusing mainly on the routing constraints, we chose to overlook some aspects of the problem in order to work on a simplified model while still providing valid solutions.

The original industrial problem consists in handling trains within a station over a biweekly planning horizon, by deciding their route and schedule inside the station, and assigning them to compatible departures. A station is composed of resources of several types (track-groups, single tracks and yards for parking, maintenance facilities for performing maintenance, platforms for arrival and departures), linked by gates. Routing of the trains inside the station must respect the length and capacity of the resources, as well as the trains' order on the resources and train/resource compatibility constraints. To be allowed to circulate, trains must be submitted to regular maintenance. Therefore, a train might be assigned to a departure only if it has enough time and mileage left before a maintenance is required for the journey. Maintenance operations can be performed in the stations on appropriate resources, at a cost. Some arrivals (joint arrivals) are composed of several trains joined together; likewise, some

departures (joint departures) must be satisfied by several joint trains. Convoys of several joint trains might be formed (respectively separated) by performing junction and disjunction operations.

The two main simplifications that were made are described below. Their main consequence is to avoid modifying characteristics of the trains in the solution.

- Not performing maintenance operations. This choice simplifies the assignment decision process: the compatibility of a given train with a given departure depends only on the instance data (including the time / distance remaining before maintenance), and not on the choice to perform maintenance on it.
- Not performing junction nor disjunction operations. That way, trains can be treated as immutable *convoys* from their arrivals to their departure. A *convoy* might be formed of a single train, or of several trains coming from a joint-arrival, which might then satisfy a joint-departure.

In this section we first extend the notations defined by Ramond and Nicolas (2014). Then, we use these new notations to formalize our simplified model.

2.1 Definitions

Definition : Convoys
A convoy v is defined by a set of trains t in $\mathcal{T}$. We note $\mathcal{V}$ the set of all the convoys.

$$\mathcal{V} := P(\mathcal{T}) \tag{1}$$

where P is the power set operator. The length and size of a convoys v in $\mathcal{V}$ can be accessed with:

$$size(v) = \#v \tag{2}$$

$$length(v) = \sum_{t \in v} length(t) \tag{3}$$

We also define $\mathcal{V}^+$, the set of convoys actually used in the solution.

Definitions : About arrivals and departures

As our model treats convoys and not individual trains, joint arrivals (respectively joint departures) can be treated as one single arrival (respectively departure). Thus, we work with a set of *extended arrivals* (respectively departures), defined in this section. An element of this set is either a unitary arrival (an arrival which is not a member of a joint arrival), or a joint arrival (a set of arrivals joint together, treated as a single element).

Let $\mathcal{U}_{arr}$ be the set of unitary arrival in opposition with $\mathcal{J}_{arr}$ the set of joint-arrivals (and similarly the set of unitary departures).

$$\mathcal{U}_{arr} := \{a \in \mathcal{A} \mid \forall \mathcal{B} \in \mathcal{J}_{arr}, a \notin \mathcal{B}\} \tag{4}$$

$$\mathcal{U}_{dep} := \{d \in \mathcal{D} \mid \forall \mathcal{B} \in \mathcal{J}_{dep}, d \notin \mathcal{B}\} \tag{5}$$

Now, we use the sets $\mathcal{J}_{arr}$ and $\mathcal{U}_{arr}$ to build a new set $Ext\mathcal{A}rr$ called set of all extended arrivals (and the set $Ext\mathcal{D}ep$ with $\mathcal{J}_{dep}$ and $\mathcal{U}_{dep}$).

$$Ext\mathcal{A}rr := J_{arr} \cup \{\{a\} \mid a \in \mathcal{U}_{arr}\} \tag{6}$$

$$Ext\mathcal{D}ep := J_{dep} \cup \{\{d\} \mid d \in \mathcal{U}_{dep}\} \tag{7}$$

Definitions : compatibility

For each convoy $v \in \mathcal{V}$ and departure in $d \in Ext\mathcal{D}ep$, a value $Comp(v, d) \in \{0, 1\}$ is defined. If $Comp(v, d) = 1$, v and d are said to be compatible. This value takes into account several parameters of the initial formulation such as size, categories, time and distance remaining before maintenance, and so on.

Similarly, a value $Comp(v, a) \in \{0, 1\}$ is defined to express compatibility between a convoy $v \in \mathcal{V}$ and an arrival $a \in Ext\mathcal{A}rr$.

Similarly, for each convoy $v \in \mathcal{V}$ and resource $r \in \mathcal{R}$, $Comp(v, a) \in \{0, 1\}$ expresses the compatibility of all trains of v with the resource r.

Definition : Connectors

Connectors are couples of gates, representing transition between resources. Using this concept, we can model the station as a directed graph.

The set of connectors noted $\mathcal{CO}$ is defined by

$$\mathcal{CO} = \left\{ (g, g') \in \mathcal{G}_r \times \mathcal{G}_{r'} \;\middle|\; \begin{array}{c} neigh_g = g' \\ neigh_{g'} = g \end{array} \right\} \tag{8}$$

Then a connector $co \in \mathcal{CO}$ is a couple of gates (g, g') such that a convoy can leave the resource r ($g \in \mathcal{G}_r$) by the gate g to enter in the resource r' by g' ($g' \in \mathcal{G}_{r'}$).

It is important to note that the connector (g, g') is not equivalent to the connector (g', g). The first one means an exit by g and an entry by g' whereas the second one means an exit by g' and an entry by g.

The train station can then be modeled by a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ where the vertices are connectors and the arcs are resources (Fig. 2). This allows to represent reverse and non-reverse resources.

Some values are associated with each edge of the graph $\mathcal{G}$ such as the $minTrTime$ and the $maxTrTime$ which are the minimal and the maximal traveling time to go from a connector to an other one through a resource. These parameter values aggregate several parameters from the original problem formulation, depending on the resources type, such as $trTime$ for trackgroups, $maxDwellTime$ for platforms, $revTime$ for the reverse operations on single tracks, platform, yards...

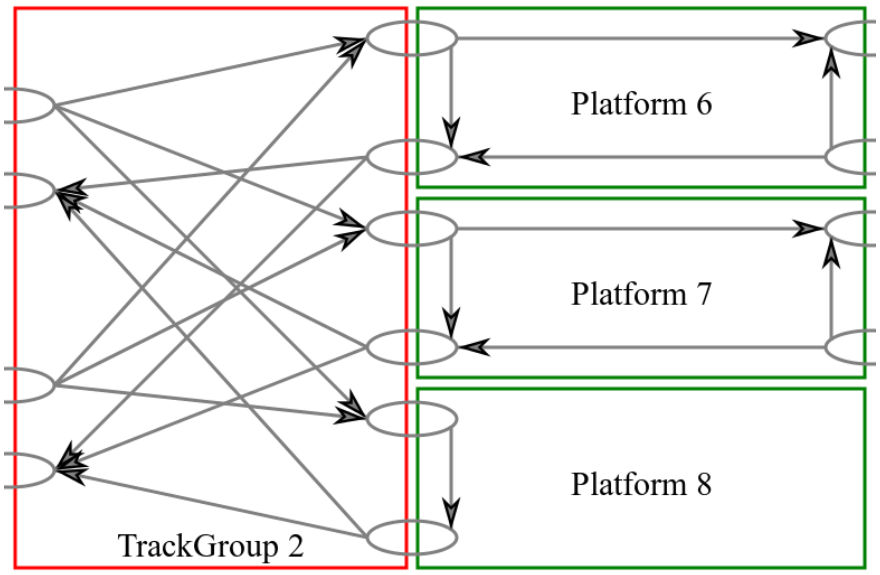


Figure 2: Directed Graph Modelisation

Definition : Path through the station
A path through the station is defined by an alternate sequence of connectors in $\mathcal{CO}$ and resources in $\mathcal{R}$.

$$p := (co_0, r_0, co_1, r_1, co_2 \dots co_{n-2}, r_{n-2}, co_{n-1}, r_{n-1}, co_n) \tag{9}$$

The size of the path p is accessible via $size(p)$ and returns the number of resources crossed along the path.

$$size((co_0, r_0, co_1, r_1, co_2 \dots co_{n-2}, r_{n-2}, co_{n-1}, r_{n-1}, co_n)) = n \tag{10}$$

An n-sized path going through resources $r_0 \dots r_{n-1}$ requires $n+1$ connectors $co_0 \dots co_n$.

Schedule of a path
The schedule of a path p, noted $sched_p$, is a tuple of instants t in the horizon $\mathcal{H}$ which are in bijection with the connectors of the path. These values describe the time of transition from a resource to the other. Let p be a path composed by n resources (Eq. 9). Then a schedule for p is defined by

$$sched_p = (t_0, t_1, t_2 \dots t_{n-2}, t_{n-1}, t_n) \tag{11}$$

where t_i in $sched_p$ is paired with co_i in p, meaning that at t_i the connector co_i is used to leave resource r_{i-1} and resource r_i.

2.2 Decision Variables

Arrival and Departure assignments

A convoy may come from an arrival (all trains composing the convoy enter the system as elements of this arrival), and be assigned to a departure (all trains composing the convoy leave the system satisfying elements of this departure).

$$\forall v \in \mathcal{V}, \quad \begin{cases} \forall a \in Ext\mathcal{A}rr, \ from(v,a) \in \{0,1\} \\ \forall d \in Ext\mathcal{D}ep, \ assigned(v,d) \in \{0,1\} \end{cases} \tag{12}$$

Paths and schedules of convoys
A convoy v in $\mathcal{V}$ may be routed, thus associated with a path $path(v)$ which is paired with a schedule $sched(v)$, of size n_v.

$$path(v) = (co_0, r_0, co_1, r_1, co_2 \dots co_{n-2}, r_{n-2}, co_{n-1}, r_{n-1}, co_{n_v}) \tag{13}$$
$$sched(v) = (t_0, t_1, t_2 \dots t_{n-2}, t_{n-1}, t_{n_v}) \tag{14}$$

2.3 Constraints

Convoys paired with Arrivals and Departures
Recall that $\mathcal{V}^+$ is the set of all convoys scheduled in the solution. Let v be a convoy in $\mathcal{V}$. If v is scheduled in the solution then v is paired with an arrival a in $Ext\mathcal{A}rr$ and a departure d in $Ext\mathcal{D}ep$.

$$v \in \mathcal{V}^+ \Rightarrow \begin{cases} \sum_{a \in Ext\mathcal{A}rr} from(v,a) = 1 \\ \sum_{d \in Ext\mathcal{D}ep} assigned(v,d) = 1 \end{cases} \tag{15}$$

Otherwise, v is not an element of $\mathcal{V}^+$ and there is no arrival nor departure assigned to it.

$$v \notin \mathcal{V}^+ \Rightarrow \begin{cases} \forall a \in Ext\mathcal{A}rr, from(v,a) = 0 \\ \forall d \in Ext\mathcal{D}ep, assigned(v,d) = 0 \end{cases} \tag{16}$$

A convoy v cannot be assigned to an incompatible arrival a or departure d. In other words :

$$Comp(v,a) = 0 \Rightarrow from(v,a) = 0 \tag{17}$$
$$Comp(v,d) = 0 \Rightarrow assigned(v,d) = 0 \tag{18}$$

Usage of resources
The usage in term of capacity and length has to be respected. Let v be a convoy, with the schedule $sched(v)$ associated to the path $path(v)$. For each convoy v in $\mathcal{V}^+$ and each instant t in $\mathcal{H}$, we define $use(v,t)$ the resource r in $\mathcal{R}$ used by the convoy v at t.

$$use(v,t) = \begin{cases} r_i \in path(v) & \textbf{if } \exists t_i, t_{i+1} \in sched_v, t_i \leq t < t_{i+1} \\ \emptyset & \textbf{otherwise} \end{cases} \tag{19}$$

Then we define the value $use(v,t,r)$, which is 1 if and only if the resource r is used at time t in the schedule $sched(v)$ of the path $path(v)$ associated to the convoy v.

$$use(v,t,r) = \begin{cases} 1 & \textbf{if } use(v,t) = r \\ 0 & \textbf{otherwise} \end{cases} \tag{20}$$

Length and capacity constraints

$$\forall r \in \mathcal{R}, \forall t \in \mathcal{H}, \begin{cases} \sum_{v \in \mathcal{V}^+} (use(v,t,r) * size(v)) \leq capa(r) \\ \sum_{v \in \mathcal{V}^+} (use(v,t,r) * length(v)) \leq length(r) \end{cases} \tag{21}$$

Valid path
Let v be a convoy in $\mathcal{V}^+$, and $path(v)$ the associated path, of size n_v. To be valid, the path, defined as in (Eq. 13), followed by v must satisfy the following constraints.

$$co_0 = (\emptyset, g_{initial}) \tag{22}$$
$$co_{n_v} = (g_{final}, \emptyset) \tag{23}$$

This means the convoy enters the station via the first connector of the path, and leaves it via the last connector of the path.

Let us now consider the i-th connector co_i, composed of two gates g_{out} and g_{in}. By definition of connectors, $neigh g_{out} = g_{in}$ and $neigh(g_{in}) = g_{out}$. Additionally, for the path to be valid, each resource must be consistent with the adjacent connectors:

$$r_{g_{out}} = r_{i-1} \tag{24}$$
$$r_{g_{in}} = r_i \tag{25}$$

A convoy v cannot use an incompatible resource r:

$$\forall r \in \mathcal{R}, Comp(v, r_i) = 0 \Rightarrow r_i \notin path(v) \tag{26}$$

Since v is a convoy of the solution, it is assigned to an arrival a in $Ext\mathcal{A}rr$ as well as a departure d in $Ext\mathcal{D}ep$. Then the path followed by the convoy v must

begin with the arrival sequence $arrSeq_a$ (Eq. 27) and end with the departure sequence $depSeq_d$ (Eq. 27).

$$arrSeq_a = (tg_0^a, \ldots, tg_{n_1}^a) \tag{27}$$

$$depSeq_d = (tg_0^d, \ldots, tg_{n_2}^d) \tag{28}$$

The arrival (resp. departure) sequence is composed by $n_1 + 1$ (resp. $n_2 + 1$) track-groups.

$$\forall r_i \in path(v), \begin{cases} i \leq n_1 \Rightarrow r_i = tg_i^a \\ n_v - n_2 - 1 \leq i \Rightarrow r_i = tg_j^d \textbf{ with } j = n_v - n_2 - 1 + i \end{cases} \tag{29}$$

Since the arrival and the departure must take place on a platform, we have:

$$r_{n_1+1} \in \mathcal{P} \quad \textbf{and} \quad r_{n_v-n_2-2} \in \mathcal{P} \tag{30}$$

Valid schedule

Let $v \in \mathcal{V}^+$ be a convoy of the solution. The schedule $sched(v)$ must respect the delay imposed by the graph $\mathcal{G}$ to go from one connector to another through a resource.

$$\forall t_i, t_{i+1} \in sched(v), \begin{cases} minTrTime(co_i, co_{i+1}) \leq t_{i+1} - t_i \\ t_{i+1} - t_i \leq maxTrTime(co_i, co_{i+1}) \end{cases} \tag{31}$$

The arrival time $arrTime_a$ and departure time $depTime_d$ must be respected. We assume the arrival sequence is composed by $n_1 + 1$ track-groups and the departure sequence by $n_2 + 1$ track-groups.

$$\forall t_i \in sched(v), \begin{cases} i = n_1 + 1 \Rightarrow t_i = arrTime_a \\ i = n_v - n_2 - 1 \Rightarrow t_i = depTime_d \end{cases} \tag{32}$$

Conflicts between convoys

On single resources

Let v_1, v_2 in $\mathcal{V}^+$ be two convoys of the solution, $path(v_1) = (co_0^1, r_0^1, \ldots, co_{n_{v_2}}^1)$, $path(v_2) = (co_0^2, r_0^2, \ldots, co_{n_{v_2}}^2)$ their respective paths, and $sched(v_1) = (t_0^1, \ldots t_{n_{v_1}}^1)$, $sched(v_2) = (t_0^1, \ldots t_{n_{v_2}}^1)$ the respectively associated schedules.

If a same single resource (platform, single track or maintenance facility) belongs to both path, the associated entry times must be different :

$$\forall r_i^1 \in sched(v_1), r_j^2 \in sched(v_2),\ r_i^1 = r_j^2 \Rightarrow t_i^1 \neq t_j^2 \tag{33}$$

Moreover, depending on the entry and exit connectors of both convoys, the following constraints on their entry and exit times must be enforced to preserve the order of the convoys on the resource (without loss of generality, we assume $t_1^i < t_j^2$) :

$$\forall r_i^1 \in sched(v_1), r_j^2 \in sched(v_2) \mid r_i^1 = r_j^2, t_1^i < t_j^2,$$
$$\begin{cases} (co_j^2 = co_{i+1}^1 \wedge co_j^2 \neq co_{j+1}^2) \Rightarrow t_j^2 > t_{i+1}^1 \\ (co_{j+1}^2 = co_{i+1}^1 \wedge co_j^2 \neq co_{j+1}^2) \Rightarrow t_{j+1}^2 > t_{i+1}^1 \\ (co_j^2 = co_{i+1}^1 \wedge co_j^2 = co_{j+1}^2) \Rightarrow (t_{j+1}^2 > t_{i+1}^1 \vee t_j^2 > t_{i+1}^1) \end{cases} \tag{34}$$

On track groups

On track groups, a specific conflict constraint applies, depending on the respective order of the entry and exit gates of both convoys. Though this constraint was implemented in the final solution, we do not reproduce it here for the sake of simplicity. One can refer to Ramond and Nicolas (2014).

At most one convoy assigned per extended arrival and departure

$$\forall a \in Ext\mathcal{A}rr, \sum_{v \in \mathcal{V}^+} from(v, a) \leq 1 \tag{35}$$

$$\forall d \in Ext\mathcal{D}ep, \sum_{v \in \mathcal{V}^+} assigned(v, d) \leq 1 \tag{36}$$

Example (Fig. 3)
Let us consider the following instance: the sets of arrivals $\mathcal{A} = \{a_1 \dots a_5\}$ and departures $\mathcal{D} = \{d_1 \dots d_6\}$; The sets $\mathcal{J}_{arr} = \{\{a_2, a_3, a_4\}\}$ and $\mathcal{J}_{dep} = \{\{d_4, d_5, d_6\}\}$. Then

$$Ext\mathcal{A}rr = \{ea_1 = \{a_1\}, ea_2 = \{a_2, a_3, a_4\}, ea_3 = \{a_5\}\}$$
$$Ext\mathcal{D}ep = \{ed_1 = \{d_1\}, ed_2 = \{d_2\}, ed_3 = \{d_3\}, ed_4 = \{d_4, d_5, d_6\}\}$$

The convoys v_1 and v_2 where

- $v_1 = \{t_1\}$, with $from(v_1, ea_1) = 1$ and $assigned(v_1, ed_2) = 1$
- $v_2 = \{t_2, t_3\}$, with $from(v_2, ea2) = 1$ and $assigned(v_2, ed_4) = 1$

are represented on the Fig. 3.

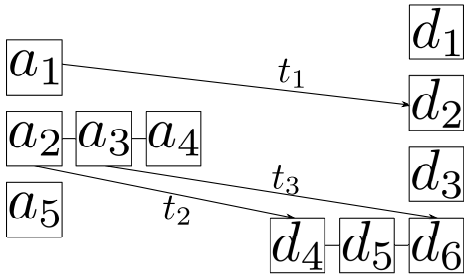


Figure 3: Convoys with provenance and assignment to departures.

2.4 Objective function

The original problem featured two optimization criteria assembled in a weighted sum: maximization of the covered arrival and departures, and minimization of performance costs (platform usage costs, preferred reuse of trains, etc.). We focused on the maximization of covered arrival and departures (while of course limited by our choices to disregard maintenance, junction and disjunction operations). With the formalism defined in this section, maximizing arrivals and departures coverage comes down to maximizing the number of trains used in the solution:

$$\max \sum_{v \in \mathcal{V}^+} size(v) \tag{37}$$

3 Data Structures and Pre-processing

In the remainder of the paper, we will use the instance represented on Fig. 1 as an example.

A maximum amount of convoy needs to be scheduled using the graph $\mathcal{G}$ previously defined. In order to reduce the cost of the routing phase by avoiding redundant treatment of very similar paths, we need a simplified structure with a higher granularity level.

The simplification consists of aggregating resources in groups (Fig. 4) of resources sharing the same characteristics and neighborhood (Rogers et al. 1991). We define the set $\mathcal{RG}$ of resources groups such that each resource $r \in \mathcal{R}$ is contained in one group $rg \in \mathcal{RG}$ with the following properties:

$$\forall rg \in \mathcal{RG}, \forall r1, r2 \in rg, neighSet_{r1} = neighSet_{r2} \wedge type(r1) = type(r2) \tag{38}$$

where *type* is defined as follow:

$$type : r \in \mathcal{R} \rightarrow \begin{cases} 0 \ if \ r \in \mathcal{K} \\ 1 \ if \ r \in \mathcal{P} \\ 2 \ if \ r \in \mathcal{S} \\ 3 \ if \ r \in \mathcal{Y} \\ 4 \ if \ r \in \mathcal{F} \end{cases} \tag{39}$$

The goal of this process is to break the systematic and redundant treatment of similar resources while allowing for a more global reasoning.

Once the simplification is done, we can define a new graph $\mathcal{G}'((\mathcal{V}', \mathcal{E}'))$, more global, enclosing the graph $\mathcal{G}$ where vertices are groups connectors $\mathcal{GCO}$ and arcs are resources groups $\mathcal{RG}$. These two graphs represent the station with different levels of granularity: the level which needs to be used depends on the step of the algorithm.

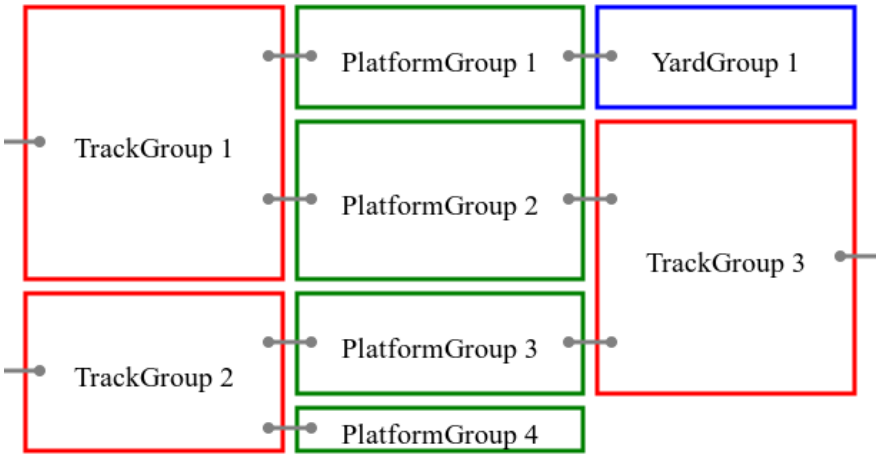


Figure 4: Resource Aggregation

To simplify further the routing phase, we choose to consider the shortest path as the preferred path between two resources depending on the category. Thus, shortest paths between any two groups of resources, or rather group connectors, are computed during the pre-processing phase using *Floyd-Warshall* algorithm (Floyd 1962). During the process, a matrix of minimal and maximal travel time along this path is computed. Each resource $r \in R$ can be considered to have a minimum and a maximum usage time. For any track-group k, the minimum bound is equal to the maximum one: $trTime_k$. The other resources have a minimum bound equal to $minResTime$ or $\max(minResTime, revTime)$ (in case of similar input/output side) and the platforms have an upper bound $maxDwellTime$.

Let $gc1$, $gc2$ be two group connectors and c a category. First, we compute the shortest path to reach $gc2$ from $gc1$. Then we note $minTrTime_c(gc1, gc2)$ the

lower time bound and $maxTrTime_c(gc1, gc2)$ the upper time bound to travel from the input connector $gc1$ to the output connector $gc2$ on the path found compatible with category c.

4 Routing Procedure

This section deals with the effective scheduling and routing of a train along a given group-level path. To schedule a train, the routing procedure considers the resource group-level path computed during the pre-processing phase, then tries to schedule the train at the resource-level, taking into account resource capacities, maximum lengths, minimal and maximal transition time, and conflicts with already scheduled trains. This is done by constraint propagation on multi-interval variables. Section 4.1 introduce the notion of multi-interval variables in this context, and Section 4.2 details the routing procedure.

4.1 Multi-interval variables

We use constraint propagation (Benhamou and Granvilliers 2006) on hull and boxes (ILOG 1999) using multi-intervals variables to filter solutions of the *Path Scheduling problem*. First we introduce the notion of time interval $\mathcal{IH}$ (Eq. 40) in the planning horizons $\mathcal{H}$, as defined by Moore (1966):

$$\mathcal{IH} := \left\{ [\underline{h}, \overline{h}] \middle| \begin{array}{l} (\underline{h}, \overline{h}) \in \mathcal{H}^2 \\ \underline{h} \leq \overline{h} \end{array} \right\} \tag{40}$$

We note $[h]$ the interval $[\underline{h}, \overline{h}]$. Let $[h_1]$, $[h_2]$ be two intervals, we define the set operations $\cap$, $\cup$:

$$\begin{array}{l} [h_1] \cap [h_2] = [max(\underline{h_1}, \underline{h_2}), min(\overline{h_1}, \overline{h_2})] \\ [h_1] \cup [h_2] = [min(\underline{h_1}, \underline{h_2}), max(\overline{h_1}, \overline{h_2})] \end{array} \tag{41}$$

A large number of interval arithmetic libraries exist, see (Knüppel 1994; Lerch et al. 2006).

A generalization of $\mathcal{IH}$ is the power set $\mathcal{P}(\mathcal{IH})$. We call $\mathcal{MIH} = \mathcal{P}(\mathcal{IH})$ the set of all the time multi-intervals in the planning horizon. Let mi_1 and mi_2 be two elements of $\mathcal{MIH}$, the multi-interval intersection is defined as below:

$$mi_1 \cap mi_2 = \bigcup_{\substack{i_1 \in mi_1 \\ i_2 \in mi_2}} i_1 \cap i_2 \tag{42}$$

We introduce the operators *lo* and *up* on intervals and multi-intervals, which represent the lower and upper bounds of these quantities (see Fig. 5 for an example).

$$\forall x \in \mathcal{IH} \left\{ \begin{array}{l} lo(x) = \underline{x} \\ up(x) = \overline{x} \end{array} \right. \quad \textbf{and then} \quad \forall y \in \mathcal{MIH} \left\{ \begin{array}{l} lo(y) = \min\limits_{x \in y} lo(x) \\ up(y) = \max\limits_{x \in y} up(x) \end{array} \right. \tag{43}$$

Mathematical comparison operators can be defined to deal with intervals and multi-intervals. Let x be an interval or a multi-interval quantity. Then the comparison of x with a real value y implies conditions over the bounds of x.

$$x \leq y \quad \Leftrightarrow \quad up(x) \leq y \tag{44}$$
$$y \leq x \quad \Leftrightarrow \quad y \leq lo(x) \tag{45}$$

The inequality constraints are used to reduce the domain of the values through a set of methods called contractors (Chabert and Jaulin 2009). Atomic and meta-contractors can be considered as basic elements of a new paradigm called contractor programming in order to perform efficient global optimization algorithm (Ninin 2015) exploiting the properties of groups of constraints. In the next section of the paper we present a contraction method for the *Path Scheduling Problem* previously introduced.

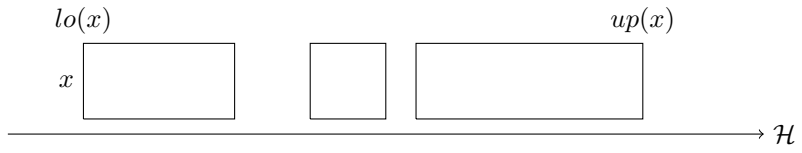


Figure 5: Example of a multi-interval, with its lower and upper bounds.

The idea behind the routing procedure, described in the next section, is to consider scheduling variables as two types of multi-interval variables, *transition* variables t_i and *usage* variables u_i. That way, for a given convoy's schedule, the multi-interval variable u_i represents the time intervals during which the corresponding resource of the convoy's path r_i is available for the convoy (with respect, amongst other, to length and capacity constraints). Similarly, the multi-interval variable t_i represents the time intervals during which the convoy may exit the previous resource and enter resource r_i through connector co_i (with respect to other convoy's transitions).

The path and schedule definition presented in the simplified model are extended accordingly, by adding to the schedule usage time variables u_i in bijection with the resources r_i in the path:

$$path(v) = (co_0, r_0, co_1, r_1, co_2 \dots co_{n-2}, r_{n-2}, co_{n-1}, r_{n-1}, co_n) \tag{46}$$
$$sched(v) = (t_0, u_0, t_1, u_1, t_2 \dots t_{n-2}, u_{n-2}, t_{n-1}, u_{n-1}, t_n) \tag{47}$$

The procedure consists in refining those multi-intervals by constraint propagation.

However, propagating constraints on multi-intervals variables is very expensive. Computing speed may be increased using lazy evaluation (Madsen and Jensen 1999). Laziness is especially used and developed in functional programming languages (Johnsson 1984) such as *Haskell* (Hudak et al. 2007). It consists into a set of methods in order to limit the number of computations by non-evaluating quantities which are not required. This approach is not yet fully

implemented in our solution, so there is still room for progress and potential speed-up.

Definition : $\mathcal{H}_{r,l,s}$
Let r be in $\mathcal{R}$, and l, s in $\mathbb{N}$. We define the set $\mathcal{H}_{r,l,s}$ as all the instants t in the planning horizons $\mathcal{H}$ such that the resource can be used by a convoy of size s with a total length l.

$$\mathcal{H}_{r,l,s} = \left\{ t \in \mathcal{H} \;\middle|\; \begin{array}{c} \sum_{v \in \mathcal{V}^+} (use(v,t,r) \times size(v)) \leq capa(r) - s \\ \sum_{v \in \mathcal{V}^+} (use(v,t,r) \times length(v)) \leq length(r) - l \end{array} \right\} \tag{48}$$

4.2 Algorithm

In this section, we detail the procedure used to route and schedule a convoy along a group-level path (Algorithm 1).

Let us consider a m-sized *convoy*, of total length l, and a group-level path $GlobalPath$. $GlobalPath$ is formalized as an alternating list of connector-groups and resource-groups (gco_0, gr_0, ..., gco_1, gr_{n-1}, gco_n).

The procedure returns, if possible, a resource-level *path* (formalized as an alternating list of connectors and resources (co_0, r_0, ..., co_1, r_{n-1}, co_n) and a matching *schedule*. A $Schedule$ is an alternate list of transition multi-interval variables, and usage multi-interval variables $(t_0, u_0, \ldots, t_{n-1}, u_{n-1}, t_n)$. t_i represents the available intervals for transition between resources r_{i-1} and r_i on connector co_i, and u_i the available intervals for usage of resource r_i.

$Schedule$ is initialized using $InitialSchedule$. In practice, we initialize all multi-interval variables to the full planning horizon, except for the transitions corresponding to the arrival and departure of the convoy, which are reduced to the arrival (respectively departure) time (see Fig. 6a).

Algorithm (Alg. 1) consists of an iteration over every path $path := (co_0, r_1, \ldots, r_{n-1}, co_n)$ enclosed in the $GlobalPath$. Each iteration is divided in 2 steps ($FilterResourceUsagePath$, and $FilterConnectorPath$), described below.

1. Refinement of the multi-interval scheduling variables (t_0, u_0, ..., t_{n-1}, u_{n-1}, t_n) associated with the path p by filtering on each resource r_k, taking into account capacity, length and usage time constraints (Method $FilterResourceUsagePath$) using the following contractors. An example of this step is unfolded (Fig. 6) on the simplified instance represented on Fig. 1, using the parameters given in Tables 1 and 2, with the parameters $minResTime = 00:02:00$ and $revTime = 00:05:00$.

$$u_k \subseteq u_{max_k} \tag{49}$$

Algorithm 1 : *Routing*

Input: *convoy*
Input: $globalPath := (gco_0,\ gr_0,\ \ldots,\ gco_1,\ gr_{n-1},\ gco_n)$
Input: $initSchedule := (t_0, u_0, \ldots, t_{n-1}, u_{n-1}, t_n)$

```
for all path = (co_0, r_0, ..., co_{n-1}, r_{n-1}, co_n) ∈ globalPath do
    schedule ← FilterResourceUsagePath(convoy, path, initSchedule)
    if (schedule) ≠ ∅ then
        schedule ← FilterConnectorPath(convoy, path, schedule)
        if (schedule) ≠ ∅ then
            return (path, schedule)
        end if
    end if
end for
return (∅, ∅)
```

where u_{max_k} is the minimal multi-interval containing $\mathcal{H}_{r_k,l,m}$. This constraint (Eq. 49) represents the limitation in term of length and/or capacity of the resource (Fig. 6a, 6b).

$$lo(t_k) \leq u_k \leq up(t_{k+1}) \tag{50}$$

This constraint (Eq. 50) translates the fact that the convoy cannot occupy the resource before entering it and after exiting it (Fig. 6b, 6c).

$$\begin{aligned} minTrTime_{r_k}(co_k, co_{k+1}) &\leq lo(t_{k+1}) - lo(t_k) \\ up(t_{k+1}) - up(t_k) &\leq maxTrTime_{r_k}(co_k, co_{k+1}) \end{aligned} \tag{51}$$

These two constraints (Eq. 51) express the relation between the enter time t_k and the exit time t_{k+1} depending on the minimal and maximal usage time of the resource r_k (Fig. 6c, 6d).

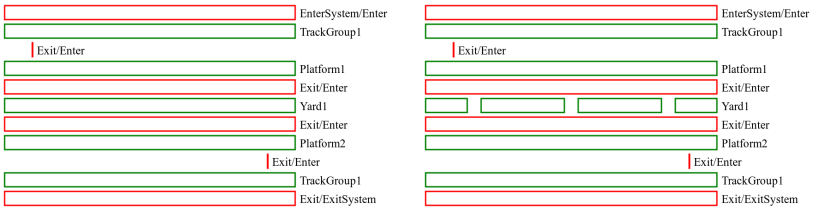
$$\forall u \in u_k, \{u\} \cap t_k \neq \emptyset \wedge \{u\} \cap t_{k+1} \neq \emptyset \tag{52}$$

This constraint (Eq. 52) means that the resource r_k is able to welcome the convoy from its entrance to its exit (Fig. 6d, 6e).

$$\begin{aligned} t_k &\subseteq u_k \\ t_{k+1} &\subseteq u_k \end{aligned} \tag{53}$$

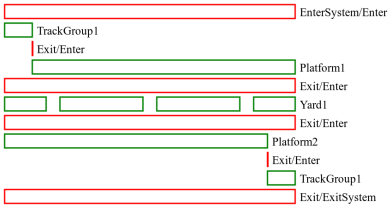
This constraint (Eq. 53) means that the convoy can enter and exit the resource r_k only when r_k is able to receive it (Fig. 6e, 6f).

2. Similarly, procedure *FilterConnectorPath* contracts the multi-interval variables $(t_0,\ u_1,\ \ldots,\ u_{n-1},\ t_n)$ associated with *path* by computing for each resource r_k the conflicts of enter time t_{k-1} and exit times t_{k+1} with already scheduled convoys occupying r_k and propagating the associated constraints. This step also propagates again, when necessary, some of the previous constraints (Eq. 51, 52).

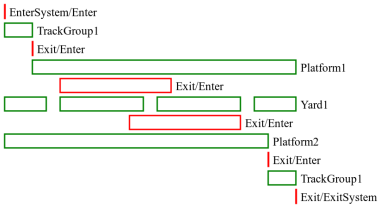


(a) Initialization

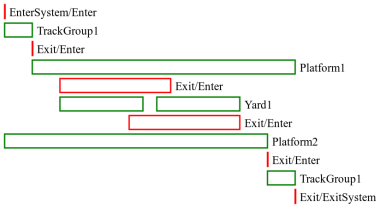
(b) Propagation Eq. 49



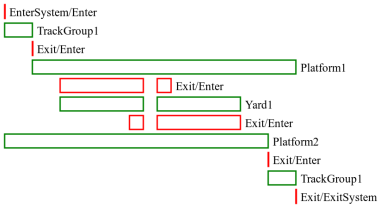
(c) Propagation Eq. 50



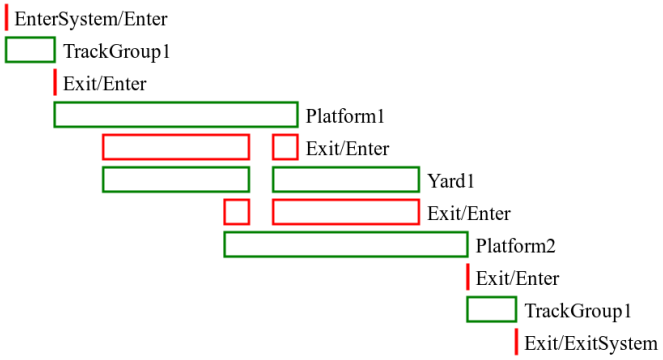
(d) Propagation Eq. 51



(e) Propagation Eq. 52



(f) Propagation Eq. 53



(g) Constraint Propagation Result

Figure 6: Multi-Interval Filtering on the example instance, from $Arr1$ to $Dep1$ (see Table 1) with the consumptions indicated in Table 2.

After both filtering procedures, if *schedule* is non-empty (which means it is possible to schedule the train along *path*), the algorithm returns, if not, it moves on to the next path.

Finally, the train is scheduled according to an earliest possible time strategy within the allowed time intervals : each transition multi-interval is contracted

Id	arrTrain	Time	Sequence	Platform	idealDwell	maxDwell
Arr1	Train1	08:00:00	TrackGroup1	Platform1	00:03:00	00:12:00
Dep1		08:17:00	TrackGroup1	Platform2	00:04:00	00:10:00

Table 1: Example instance: arrivals/departures listing

Resource	Begin	End
Yard1	08:01:00	08:02:00
Yard1	08:08:00	08:09:00
Yard1	08:15:00	08:16:00

Table 2: Example instance: imposed consumptions listing

to its lower bound, and usage multi-intervals are contracted accordingly. Note that the malleability of the structure leaves the possibility of implementing other strategies to avoid conflict with subsequent train schedules or optimize the enter and exit time on each resource to reduce penalty.

Example of propagation (Fig 6)

Fig. 6 illustrates an example of propagation. Each line represents a multi-interval variable; lines labeled "Exit/Enter" are transition variables, and lines labeled with a resource name are usage variables.

Initially (Fig. 6a), none of the variables are restricted, except for the transitions corresponding to the arrival and departure (first and last "Exit/Enter"), which are restricted to the exact arrival and departure times. Then, in Fig. 6b the usage of resource $Yard1$ is restricted according to the imposed consumptions listed in Table 2. In Fig. 6c, usage of resources $Platform1$ and $TrackGroup1$ are restricted according to the preceding "Exit/enter" transition variables: the convoy cannot use a resource before it enters it. Then, the transition variables are restricted to allow sufficient time between the entry and the exit on each resource (Fig. 6d).

Let us now focus on resource $Yard1$: the last interval of its usage variable doesn't intersect with the following transition variable, so the yard cannot be used during this interval. It is thus removed in Fig. 6e. The enter and exit transitions are adjusted accordingly in Fig. 6f, as the convoy may not enter or exist the yard at a given instant if it is not available for use.

Fig. 6g shows the result once a similar propagation has been performed $Platform1$ and $Platform2$.

5 Greedy Assignment Algorithm

The problem is complex due to the routing problem in the station. Resource usage is crucial and we should avoid to engorge one resource of the station. This idea leads to the strategy of the greedy algorithm we present in this section.

We call *immediate arrivals-departure* arrivals immediately followed by a departure on the same platform. In this case, the convoy only has to be routed through the arrival sequence and departure sequence, so the path is set except for the platform, and the convoy occupies few resources.

If a departure is not an immediate departure, the convoy has to be routed to a parking spot and parked between its arrival and its departure. We chose to park on yards only.

Our objective is the maximization of covered arrivals and departures as stated in Section 2. However, to comply with the multi-criteria objective of the original problem and limit penalties, considerations on trains reuses, preferred platform and platform usage penalties are implemented in the solution.

The algorithm is composed of three main steps.

1. First, we try to schedule immediate arrivals-departures while considering train reuses as hard constraints.
2. Then, we relax these constraints and try to schedule immediate arrivals-departures.
3. Finally, we try to schedule remaining trains from the arrival to departure with a parking phase (on yards only) in-between, first considering train reuses as hard constraints, then relaxing these constraints.

To improve coverage of joint departures without performing junctions nor disjunctions, each of these three main assignment phase follows the three steps described below.

(i) Try and assign joint arrivals to joint departures. Both departures and arrivals are taken in decreasing order of convoy size. Between a given joint arrival and a given joint departure, simple dynamic programming allows to compute an assignment maximizing the number of arrivals and departures satisfied, while respecting compatibility and trains order. When at least some members of a joint arrival/departure have been assigned, all the other members are locked and cannot be used in the subsequent assignment steps.

(ii) Try and assign unitary arrivals to unitary departures.

(iii) Try and assign members from remaining joint arrivals to remaining unitary departures (joint or not) and symmetrically remaining unitary arrivals to members of remaining joint departures. When one member is assigned, the other members are locked.

Formally, these sub-steps are described by Alg. 2, using the following definitions:

- $\mathcal{J}^+_{arr}$ (respectively $\mathcal{J}^+_{dep}$) the set of joint arrivals (respectively departures) used in the solution,

$$\mathcal{J}^+_{arr} = \{a \in \mathcal{J}_{arr} | \exists j = jointArr_a, \exists a' \in jaList_j, a' \in \mathcal{A}^+\}$$
$$\mathcal{J}^+_{dep} = \{d \in \mathcal{J}_{dep} | \exists j = jointDep_d, \exists d' \in jdList_j, d' \in \mathcal{D}^+\}$$

- $\mathcal{A}^+$ (respectively $\mathcal{D}^+$) the set of unitary arrivals (arrivals which are not members of any joint arrival) used in the solution,

$$\mathcal{A}^+ = \{a \in \mathcal{A} | arrTrain_a \in \mathcal{T}^+\} \cup \mathcal{J}^+_{arr}$$
$$\mathcal{D}^+ = \{d \in \mathcal{D} | depTrain_d \in \mathcal{T}^+\} \cup \mathcal{J}^+_{dep}$$

- by complement, the sets of joint / unitary arrivals (respectively departures) not used in the solution: $\mathcal{A}^- = \mathcal{A} - \mathcal{A}^+$, $\mathcal{D}^- = \mathcal{D} - \mathcal{D}^+$, $\mathcal{J}^-_{arr} = \mathcal{J}_{arr} - \mathcal{J}^+_{arr}$, $\mathcal{J}^-_{dep} = \mathcal{J}_{dep} - \mathcal{J}^+_{dep}$.

Algorithm 2 Assignment Greedy Algorithm Step

1: $Assign(\mathcal{J}^-_{arr}, \mathcal{J}^-_{dep})$
2: $Assign(\mathcal{A}^- - \mathcal{J}^-_{arr}, \mathcal{D}^- - \mathcal{J}^-_{dep})$
3: $Assign(\mathcal{A}^-, \mathcal{D}^-)$

Non-assigned departures and compatible arrivals are taken in increasing order. Each time, the *Routing* procedure is used to schedule the corresponding convoy. This continues until an arrival with a feasible routing has been found (then the arrival is assigned to the departure), or until there is no more fit arrivals (then the departure remains unsatisfied).

To limit the number of infeasible attempts, arrivals and departures are filtered, taking into account compatibility characteristics such as category and remaining time/distance before maintenance, but also existence of a path between two arrival/departure sequences and time between arrival and departure.

$$arrTime_a + minTrTime_{cat_{T_a}}(P_a, P_d) \leq depTime_d \tag{54}$$

The routing algorithm (Alg. 1) is used inside the greedy subroutine *Assign* (Alg. 2). The latter consists in a serie of attempts, iterating on the arrivals and the departures sets. For each couple arrival/departure, a compatible convoy is selected and a resource group-level path (computed during the pre-processing phase) is selected to link the arrival sequence with the departure sequence. Thus, the routing algorithm is applied to this convoy and global path. If the process fails to provide a valid resource-level path (Alg. 1, line 10), we move on to the next compatible couple. Otherwise, the convoy is scheduled along the path, thus the arrival and the departure are respectively removed from the available arrivals and the available departures sets.

Fig. 7 shows a simplified example on an instance with three arrivals and four departures. A departure is compatible with an arrival if they are represented using the same shape (disc, square) and if the arrival takes place before the departure on the time line. On Fig. 7, a line from an arrival to a departure is an attempt to route and schedule a convoy through the station from the arrival to the departure. A red dotted line is a failure while a solid green line is a success. Here, the couple $arr1/dep1$ is compatible and is succesfully routed and

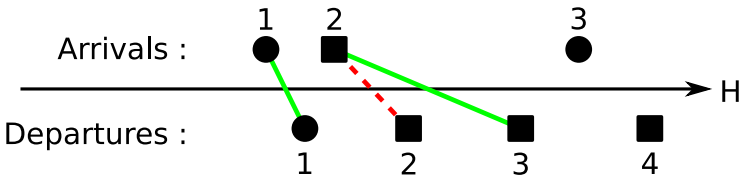


Figure 7: Greedy Algorithm : example

scheduled. The couple $arr2/dep2$ is compatible but the routing procedure fails, so the next compatible couple $arr2/dep3$ is considered. The routing procedure successfully returns a valid schedule, so $arr2$ and $dep2$ are assigned to each other, and the convoy is routed and scheduled. The last arrival, $arr3$, does not have any compatible departure, so no routing attempt is made.

In order to limit platform assignment penalties, when scheduling a convoy from an arrival to a departure, the preferred platform is tried in priority. To limit the platform usage penalties, we limit the time that a train can spend on a platform. For immediate departures, this amount of time is computed using the ideal dwell times of the departure and the arrival, and the costs defined in parameters, in order to ensure that the platform usage cost for this train does not exceed the $Uncov$ penalty for not covering the arrival. We filter the arrivals verifying this condition. For arrivals and departures separated by a parking phase, we ensure during the routing phase that the ideal dwell times are exactly respected.

6 Results and Conclusions

The experiments were performed on an Optiplex computer featuring an Intel Core 2 Duo E8400 3.00GHz CPU with 4Go RAM, running Xubuntu 64 bits. Table 3 presents the results obtained on instance set B, provided by the challenge organizers. The execution was stopped after 10 minutes of running time. Table 3 reports the results obtained when stopping after the first two stages of the algorithm (which means allowing only immediate departures), and the results obtained after full execution of the algorithm (which allows parking during the last stage). In each case, we give the number and proportion of covered departures. We also give, for information purposes, the value of the objective function used in the original formulation of the ROADEF Challenge problem (Ramond and Nicolas 2014), which aggregates various performance costs as well as penalties for uncovered departures and arrivals. However, within our simplified model, the number of covered departures alone is a more pertinent indicator.

The solutions provided within the allowed time cover 20% to 40% of departures (and as much arrivals). Of course, this is very few and not suited for a practical application. Although, considering the complexity of the problem and the size of the search space, it is actually quite reasonable for a simple greedy

	immediate departures only			with parking phase (complete algorithm)		
	t (s)	objective	covered dep	t (s)	objective	covered dep
B1	24	1812600	35% (443/1235)	t/o	1749000	38% (474/1235)
B2	23	1682080	39% (490/1235)	t/o	1656180	40% (503/1235)
B3	21	1711330	38% (472/1235)	t/o	1665330	40% (495/1235)
B4	71	2578160	36% (649/1780)	t/o	2561360	36% (657/1780)
B5	136	3190890	34% (747/2153)	t/o	3155790	35% (764/2153)
B6	72	2584130	36% (645/1780)	t/o	2559330	36% (657/1780)
B7	2	456112	34% (105/304)	68	420112	40% (123/304)
B8	2	473484	31% (96/304)	83	439884	37% (113/304)
B9	72	2978598	29% (590/1967)	t/o	2858098	32% (649/1967)
B10	< 1	364594	18% (36/196)	226	321794	29% (57/196)
B11	3	1984144	16% (190/1122)	t/o	1858544	22% (250/1122)
B12	1	1031864	16% (94/570)	t/o	982164	20% (118/570)

Table 3: Summary of the results on set instance B. The first 3 columns present the results for the algorithm limited to immediate departures only, the last 3 when subsequently allowing an intermediate parking phase in yards between arrival and departure. for each case, we give the execution time in seconds, the value of the ROADEF Challenge objective function and the proportion of covered departures. "t/o" means that the execution has reached the timeout, set to 10 minutes.

assignment algorithm. Moreover, given that our approach does not allow maintenance, junction and disjunction operations, the number of departures which can be covered is limited. This suggests that the routing algorithm itself is quite powerful, and that paired with a more advanced assignment optimization algorithm, it could lead to interesting results.

From Table 3, one can notice that most covered departures are "immediate departures", which are departures following directly the arrival on the same platform, assigned during the first two stages of the algorithm. The next stage, allowing trains to change platforms between arrival and departure by transiting through yards, gains only a few percent and is time consuming. Thus, considering only immediate departures seems a suitable approach to compute rather quickly an initial solution.

When considering immediate departures, the path and timing is already precisely defined by the arrival and departure sequences. In this case, the *Routing* procedure only serve to determine the feasibility of this path and timing for a given couple arrival/departure and select a platform, given the current state of the solution and avoiding conflict with other trains. When allowing parking, additional decisions must be made, namely the selection of the parking yard and the duration of stay. Thus, in this case the exploration performed by the *Routing* procedure is heavier and more time consuming.

Total execution times appear shorter for instances with few departures and arrivals (B7, B8, B10), as less routing attempts need to be made. The compar-

atively high execution time for instance B10 might result of the high proportion of joint departures and arrivals in this instance, as members of a joint arrival or departure may be tentatively routed several time, as convoys and as unitary trains.

The most interesting point in the solution exposed in this paper is without any doubt the association of the three-level routing algorithm with the greedy algorithm. Indeed, the multi-interval constraint propagation is flexible, efficient and return a set of possible range-value to enter and exit from each resource, allowing to find non-trivial routes within the range of possibilities.

Note that as a canonical path between any pair of resources is computed during the preprocessing phase, trains traveling with a given origin and destination are always routed along the same path. Choosing among a set of preferred paths, or computing on the fly alternative paths forbidding certain resources could be useful to avoid engorgement.

However the solution we propose here is robust, and the optimizer always return a feasible solution. An important aspect of the greedy algorithm is the ability to build a correct solution really fast on all the instances and to build a better solution if more time is available.

A simple optimization process could merely consist of choosing a random train, then removing it from the solution and finally try to build another schedule. Rather than just choosing randomly, one could select the train which generates the most conflicts: each time the algorithm tries and fails to schedule a train, the conflict value of the trains which prevented the scheduling is incremented ; at the end, the train with the highest conflict value is removed from the solution. This could be used as the basis for a local search optimization procedure.

References

Benhamou, F. and Granvilliers, L. (2006). Continuous and interval constraints. *Foundations of Artificial Intelligence*, 2, 571–603.

Chabert, G. and Jaulin, L. (2009). Contractor programming. *Artificial Intelligence*, 173(11), 1079–1100.

Floyd, R. W. (1962). Algorithm 97: shortest path. *Communications of the ACM*, 5(6), 345.

Hudak, P., Hughes, J., Peyton Jones, S., and Wadler, P. (2007). A history of haskell: being lazy with class. In *Proceedings of the third ACM SIGPLAN conference on History of programming languages* (pp. 12-1–12-55). ACM.

ILOG, S. (1999). Revising hull and box consistency. In *Logic Programming: Proceedings of the 1999 International Conference on Logic Programming*, (p. 230). MIT press.

Johnsson, T. (1984). Efficient compilation of lazy evaluation. *SIGPLAN Not.*, 19(6), 58–69.

Knüppel, O. (1994). Profil/biasa fast interval library. *Computing*, 53(3-4), 277–287.

Lerch, M., Tischler, G., Gudenberg, J. W. V., Hofschuster, W., and Krämer, W. (2006). Filib++, a fast interval library supporting containment computations. *ACM Transactions on Mathematical Software (TOMS)*, 32(2), 299–324.

Madsen, A. L. and Jensen, F. V. (1999). Lazy propagation: a junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*, 113(1), 203–245.

Moore, R. E. (1966). *Interval Analysis*. Prentice-Hall series in automatic computation. Prentice-Hall.

Ninin, J. (2015). Global optimization based on contractor programming: an overview of the ibex library. In *International Conference on Mathematical Aspects of Computer and Information Sciences* (pp. 555–559). Springer.

Ramond, F. and Nicolas, M. (2014). Trains don't vanish ! ROADEF EURO 2014 Challenge Problem Description. 39 pages.

Rogers, D. F., Plante, R. D., Wong, R. T., and Evans, J. R. (1991). Aggregation and disaggregation techniques and methodology in optimization. *Operations Research*, 39(4), 553–582.