A Math-Heuristic Framework for the ROADEF/EURO Challenge 2014

Haahr, Jørgen Thorlund; Bull, Simon Henry

Publication date:
2014

A Math-Heuristic Framework for the ROADEF/EURO Challenge 2014

Jørgen Thorund Haahr and Simon Henry Bull
Department of Engineering Management
Technical University of Denmark

June 29, 2014

1 Overview

The proposed solution framework can be classified as a *math heuristic* as it combines exact with heuristic methods. The heart of the solution framework is an Simulated Annealing (SA) framework that iteratively destroys and builds random train routes. However, in order to improve convergence (and runtime) a few smaller Mixed Integer Program (MIP) problems are solved in advance in order to avoid resource use conflicts and improve resource utilization. Figure 1 illustrates the flow (and main components) of the solution framework. The full Rolling Stock Unit Management (RSUM) problem is decomposed into four sequential steps which will be described in the following sections of this document. In the first step arrivals and departures are matched in order to get the best possible matching, such that e.g. the number of cancellations is minimized. Next, a platform slot is reserved for all arrivals and departures such that as many as possible are assigned to preferred platforms. Thirdly, a track group usage pattern is chosen for all arrival/departure sequences, such that no pairs of patterns are in conflict and such that no pattern is in conflict with the pre-specified imposed resource usages. Fourthly, non-overlapping facility usage slots are reserved for all maintenance activities (these are generated as a results of the matching). Finally, an SA approach iteratively removes and reroutes a group of *related* (see section 4) trains as specified by the found matching.

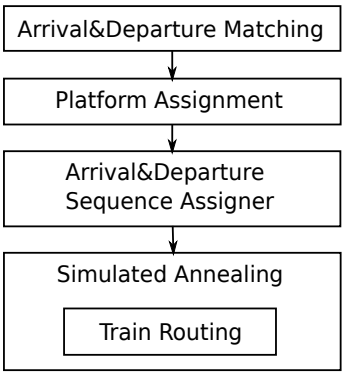

Figure 1: An overview of the solution framework flow

2 Matcher

The first subproblem in the solution framework tries to match departures with compatible trains, i.e., initial trains or arrivals. The matching is formulated as a mathematical model consisting of linear constraints (and an objective) and solved using column generation, due to the large number of variables. The primary goal is to minimize the number of uncovered departures while a secondary goal is to maximize train re-uses.

The objective is formulated as a minimization of the number of unmatched departures and cost of non-satisfied train reuse :

$$\begin{aligned}\min & \sum_{d \in D} cancellationCost_d \cdot c_d \\ & + \sum_{d \in D} \sum_{t \in Comp(d)} reuseCost_t^d \cdot m_t^d \\ & + \sum_{p \in P} reuseCost_p \cdot \lambda_p\end{aligned}$$

A few heuristic artificial heuristic costs are also be added to improve the ability to perform the routing afterwards. The constraints are:

$$\sum_{t \in Comp(d)} m_t^d + \sum_{p \in P} \alpha_p^d \lambda_p \geq 1 - c_d \qquad \forall d \in D \tag{1}$$

$$\sum_{t \in Comp(d)} m_t^d + \sum_{p \in P} \alpha_p^d \lambda_p + \sum_{t \in Comp(d)} Block_d^t \cdot m_t^d + \sum_{p \in P} Block_p^d \cdot \lambda_p \leq 1 \qquad \forall d \in D \tag{2}$$

$$\sum_{d \in Comp(t)} m_t^d + \sum_{p \in P} \beta_p^t \lambda_p \leq 1 \qquad \forall t \in T \tag{3}$$

$$\sum_{d \in D} \sum_{t \in Comp(d)} Maint_{t,d}^{day} \cdot m_t^d + \sum_{p \in P} Maint_p^{day} \lambda_p \leq MaintLimit_{day} \quad \forall day \in H \tag{4}$$

Where m_t^d is a binary variable indicating whether train t is matched to departure d. The binary variables c_d indicate whether departure d is cancelled or not. The binary variables λ_p indicate whether a linked-departure pattern p is chosen. Since it is not trivial to model linked departures the m_t^d variables only indicate choices for non-linked matches while all linked arrival and departures are modelled using patterns. A pattern is the arrival/departure trajectory of a *real* train, i.e., a pattern is a sequence of matches where all (except possibly the last) have non-linked departures. A full enumeration of these patterns is intractable which is why column generation is used, the column generation process is describe in subsection 2.1.

The cancellation cost also includes the cost of a non-satisfied reuse, if such is present. The sets D, T and H respectively represent all departures, all trains and all days of the planning horizon. The $Maint_{t,d}^{day}$ and $Maint_p^{day}$ coefficients denote how many maintenance operations are needed at day $day \in H$. The $Block_d^t$ and $Block_p^d$ indicate whether the assignment blocks departure d. The coefficients α_p^d and β_p^t respectively denote whether a pattern contains departure d or train t. The set $Comp(d)$ is the set of trains which are compatible with d, likewise the $Comp(t)$ is the set of departures that are compatible with train t. These two sets are generated in a preprocessing step. Constraints (1) ensure that every departure is assigned to some train, unless there is a cancellation. Constraints (2) ensure that at most one train is assigned to every departure, and also block departures for trains that assume that the departure is cancelled. Note that where

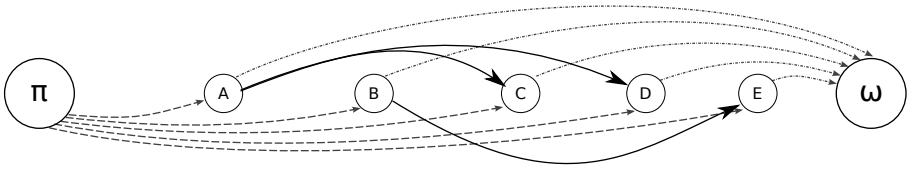

Figure 2: An illustration of the underlying graph for the matching subproblem

t is a linked train m_t^d assumes that the linked departure d' is cancelled. Constraints (3) ensure that each train is assigned at most once. Finally, Constraints(4) ensure that the total number of maintenance operations (every day) is respected. For simplicity it is here assumed that, given a matching (t,d), the day that a maintenance operation is performed is fixed. With the additions of more variables, it is possible to make the day of maintenance operations a choice.

Instead of enumerating all possible train and departure matches the sets $Comp(d)$ and $Comp(t)$ are computed. First, any pairs with a train t arriving after the departure d are removed as such clearly cannot be matched. Second, all pairs where the train is not compatible with the departure are removed. Third, pairs are removed by inspecting the timespan between train and departure and comparing this to the minimum time required for routing and required maintenance appointments. Finally, all arrivals (or departures) are removed where there exists no feasible arrival (or departure) sequence, due to imposed resources. Since runtime is scarce a number of heuristic choices can also be employed here. In order to reduce the MIP size the solution framework furthermore removes matches longer than a certain timespan. Due to the team size and time all joint arrivals and departures are removed.

A model (solved in a full branch-and-price framework) can be used to generate a true lower bound on uncovered departures, or even a lower bound on the minimum cost. Such a lower bound could prove useful for heuristic methods.

2.1 Column Generation Subproblem

Column generation is a well-described technique used with success for solving MIP problems, e.g. the Vehicle Routing Problem with Time Windows (VRPTW). It is assumed that the reader is familiar with column generation solution methods. The subproblem can be solved as a Resource Constraint Shortest Path Problem (RSCPP). The underlying graph consists of one node per linked arrival and departure pair in addition to one source and one sink node, see Figure 2. The edges constitute matching choices. Three families of edges are added. First, edges originating from the source to every node in the graph represent compatible arrivals that are matched to the linked departure of the node. Second, edges are added between nodes that represent compatible linked continuations, i.e., linked arrival/departures that connect to another linked arrival/departure. An example: In the (π, A, D, ω) path the departure of the initial train matching (represented by edge (π, A)) is linked to one arrival (node A). The departure of the next matching ((A, D)) is to another linked arrival (node D). The departure of the last matching is not linked to any arrival, and thus the sequence ends.

The problem is a RSCPP due to the maintenance constraints. In addition to the objective coefficients, the duals from Constraints (1) and (2) are added to edges of the corresponding departure, and the duals from Constraint (3) are added to edges of the corresponding trains. The appropriate dual from Constraints (4) is added every time a maintenance operation is scheduled. Labels in the Shortest Path enumeration method keep track of total cost, remaining DBM and remaining TBM. Domination is possible if a label has lower cost ($\leq$), and at least equal remaining DBM and TBM ($\geq$). Labels originate from the source vertex and are extended by traversing available arcs and deciding whether to perform maintenance (DBM or TBM or both).

3 Platform Assigner

Platforms must be assigned to all covered arrivals and departures. Once an arrival/departure matching is known, a platform assignment can be performed. In a highly utilized network the arrivals and departures may be competing for the same platforms, which motivates an exact solution approach. In the solution framework a MIP is formulated that assigns one compatible platform to every covered arrival and departure:

$$\max -\sum_{a \in A} cancellationCost \cdot s_a - \sum_{d \in D} cancellationCost \cdot s_d + \sum_{(i,j) \in NC} c \cdot d_{i,j}$$

The constraints are:

$$\sum_{p \in Comp(a)} x_a^p \geq 1 - s_a \qquad \forall a \in A \tag{5}$$

$$\sum_{p \in Comp(d)} x_d^p \geq 1 - s_d \qquad \forall d \in D \tag{6}$$

$$x_i^p + x_j^p \leq 1 \qquad \forall p \in P, (i,j) \in C \tag{7}$$

$$end_i + M(1 - x_i^p) + d_{i,j} \leq begin_j + M(1 - x_j^p) \qquad \forall p \in P, (i,j) \in NC \tag{8}$$

$$\tag{9}$$

Two sets of binary variables are used x_a^p and x_d^p respectively indicating whether arrival a or departure d is assigned to platform p. A set of continuous variables $d_{i,j}$ exists for measuring the (approximate) slack between two consecutive platform usages. The ideal objective is to maximize the smallest slack variable. However, for performance reasons the objective is changed to maximise the slack sum (i.e. average). A coefficient $c \in \mathbf{R}$ is used for scaling the distance. C is the set of all usage pairs that are overlapping in time. NC denotes all pairs of consecutive (in time) usages. Constraints (5)-(6) ensure one assignment (or cancellation). Constraints (7) ensures that two assignments (arrival or departure) cannot be assigned to the same platform if they overlap in time. Constraints (8) measure the slack between two consecutive events if they appear on the same platform. The $start_j \in \mathbf{R}$ and $end_j \in \mathbf{R}$ are determined by the minimal usage required for the corresponding arrival or departure usage.

Note that an arrival or a departure is not limited to using the assigned platform, however the found platform assignment will ensure a minimal cancellation due to lack of a available platforms.

4 Simulated Annealing

In the final step a SA approach is used to search for a good solution. The initial solution is an empty solution and in every iteration a train is selected for routing, and a randomized path is generated for the train (based on the matching and allocated resource allocations). Before routing the neighbourhood of the selected train is also removed (if the path is blocked). The neighbourhood is the set of other trains in the current solution that intersect usages on the selected path. After adding the generated path for the selected train the neighbourhood is re-inserted (if possible) in random order. The new solution is then accepted or rejected depending on new solution cost and the current temperature. An overview is shown in Algorithm 4.1.

4.1 Router

Given a existing resources usages and a valid resource path (with unspecified entrances, exits and gates) through the infrastructure the *Router* aims to find non-conflicting usages for that path. Multiple feasible solutions may exist, but ties are broken by assigning an artificial cost to dwell times at each resource. The *Router* recursively explores all available usage windows on a single resources. If compatible windows (earliest entrance and latest exit) are found for all resources on

Algorithm 4.1: `SA`

```
   Data: asdf
1  current ← GenerateEmptySolution()
2  T ← T_init
3  while T_term < T do
4      for i ∈ {0, ..., iterations} do
5          t ← RandomTrain()
6          p ← RandomPath(t)
7          n ← FindNeighborhood(p)
8          solution' ← current
9          solution' ← Destroy(solution', p)
10         solution' ← Destroy(solutoin', n)
11         solution' ← Route(solution', p)
12         solution' ← Route(solution', n)
13         δ ← Cost(solution') − Cost(current)
14         if δ < 0 then
15             current ← solution'
16         else if Random(0, 1) < e^(−δ/T) then
17             current ← solution'
18     T ← T · α
```

the path then the lowest cost assignment is made (using the artificial costs). The optimal solution can be sought by continuing the search and pruning unexplored paths whenever possible.

5 Final Remarks

The problem has proven to be very difficult. Implementing the solution framework has been time-consuming and the work is still not completed. Routing of joint arrivals and departures has been omitted due to time constraints.