Rolling stocks maintenance scheduling

Tom Ray[1,2], Ronan Bocquillon[1], Vincent T'kindt[1]

[1] Laboratoire d'Informatique Fondamentale et Appliquée de Tours, Polytech Tours, France
{ronan.bocquillon, vincent.tkindt}@univ-tours.fr

[2] SNCF Voyageurs, Ingéniérie du matériel cluster ouest, Saint-Pierre-Des-Corps, France
tom.ray@sncf.fr

Keywords : *Rolling stock maintenance, Mathematical Programming, Constraint Programming, Heuristics*

1 Introduction

Passenger trains have a very precise schedule due to the transportation demand and railway systems aim to exploit rolling stocks to their maximum capacity. Maintaining a healthy network of rolling stocks can be really difficult because it must rely on an effective maintenance schedule that does not impact the transportation plan. But while some maintenance operations are known beforehand, some repairing that could not have been predicted still needs to be done. These jobs are brought to our knowledge through the train itself. The time allowed to fix these malfunctions is relatively short (from a few hours to a few days). It is allowed to schedule a complete repair, or a partial repair named diagnosis that ensures that the train can be used in normal condition even if the operation is not completely done. The aim of our study is to find an efficient way to schedule the starting times of the maintenance jobs, completely or not, so that their due dates are met.

2 Problem definition and associated models

2.1 Problem definition

Let $I = \{1, ..., n\}$ be the set of jobs to schedule and $J = \{1, ..., m\}$ be the set of tracks available for the maintenance. Each job i has a repair duration p_i, a diagnosis duration p_i^d, a due date d_i, a tardiness cost w_i, a diagnosis cost u_i and a need for a specific infrastructure. Each track j has one or more infrastructures. Starting from the compatibility between the infrastructure requirements of the jobs, those available on the tracks and the availabilities of trains and tracks, we define $\mathcal{T}_{ij}^d$ as the set of time intervals at which job i can start on track j. We define T_i as the tardiness of job i. Let S_i be the starting time of job i in a given schedule, then we have: $T_i = S_i - d_i$. In this problem, we aim to minimize the sum of the weighted tardiness of jobs while limiting the number of performed diagnosis, especially on highly important repairs. We denote by ϵ the total cost allowed for the performed diagnosis. This problem is noted $P|\mathcal{T}_{ij}^d|\sum w_i T_i$ and is strongly NP-Hard.

2.2 Mixed Integer Linear Programming and Constraint Programming

We present a time-indexed mixed integer linear programming (MILP) model based on binary variables representing the times t at which jobs start. We define x_{ijt} as the binary variables representing the times t that can be chosen as the starting time of job i on track j. We also define y_i as the variable associated with job i, representing the decision on the realisation mode (complete repair or diagnosis).The constraints of the model ensure that for each operation, one starting time is selected, they also ensure that the jobs do not overlap.

We also present a constraint programming (CP) model based on interval variables. For this model we define J_i as the interval variable associated with job i, J_{ij} as the interval variable representing the possibility of job i being scheduled on track j and J^c_{ij} (resp. J^d_{ij}) as the interval variables representing the possibility of job i being scheduled on track j in complete repair mode (resp. in diagnosis mode). To guarantee that exactly one track and one mode (complete repair or diagnosis) is selected for each job, we use two levels of alternatives: the first one ensures that exactly one track is selected for each job, while the second ensures that exactly one mode is selected for each job. For each track, we use a disjonctive constraint to ensure that the jobs do not overlap. The complete models are not reported in the paper but they will be presented during the conference.

3 Two local search heuristics

In this section, we introduce two local search heuristics. The concept used is the same for both as they are local search heuristics: we define a neighbourhood of solutions to explore and we try to iteratively improve our current solution, step by step, until we are stuck into a local optimum or we have reach a given time limit. Each heuristic exploits two procedures: the first one, called intensification, explores the neighbourhood of a solution. The second one, called diversification, is used in case we are stuck into a local optimum to try to jump to another neighbourhood that may be more interesting. The first heuristic is a Local Branching (LB) heuristic [2] that exploits the MILP formulation. For this heuristic, we use a Hamming distance constraint to define the neighbourhood of the current solution. This distance counts every change between two consecutive iterations. The intensification (resp. the diversification) process consists in upper bounding (resp. lower bounding) the hamming distances. The second heuristic is a variable partitioning local search (VPLS) heuristic [1] that exploits the constraint programming formulation. The neighbourhood is defined by randomly selecting multiple non-overlapping intervals and a set of tracks. At each iteration, we free everything that has been scheduled in the selected intervals and tracks. Everything else is set exactly as it is in the current solution. Then, we repeat as many iterations as possible within a given time limit. The complete description of the proposed heuristics are not reported in the paper but they will be presented during the conference.

4 Conclusions

We conducted experiments on a set of randomly generated instances that follow a structure similar to the case of rolling stocks fleets and maintenance sites located in Paris, more specifically in the northern part of the city, and represent scenarios that the planners may encounter during their work. After evaluation, we conclude that LB and VPLS improve the results of their respective parent model alone but also that on the average LB improves the results of both models. The VPLS heuristic is faster than LB in most cases but not necessarily more efficient. Therefore, VPLS is more interesting to use with a reduced time budget. But as long as efficiency is considered, LB outperforms VPLS. More detailed results and perspectives will be discussed during the conference.

References

[1] F Della Croce, Andrea Cesare Grosso, F Salassa, et al. Matheuristics: embedding milp solvers into heuristic algorithms for combinatorial optimization problems. In *Heuristics: theory and applications*, pages 31–52. NOVA Publisher, 2013.

[2] Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98(1):23–47, Sep 2003.