Real-time train routing and scheduling through mixed integer linear programming: Heuristic approach

Paola Pellegrini, Guillaume Douchet, Grégory Marliere, Joaquin Rodriguez

International Conference
on Industrial Engineering and Systems Management
IESM'2013
October 28 - October 30
RABAT - MOROCCO

Real-time train routing and scheduling through mixed integer linear programming: Heuristic approach ⋆

Paola Pellegrini, Guillaume Douchet, Grégory Marlière and Joaquin Rodriguez

Ifsttar – ESTAS, Univ. Lille Nord de France
rue Élisée Reclus 20, 59666 Villeneuve d'Ascq, Lille, France

Abstract

In railway traffic management, when an unexpected event perturbs the system, finding an effective train routing and scheduling in real-time is a key issues. Making the right routing and scheduling decisions may have a great impact on the efficiency of the system in terms of delay propagation. However, the time available for making these decisions is quite short: in few minutes a viable set of routes and schedules must be delivered to the dispatching system. In this paper, we assess the performance of a mixed integer linear programming (MILP) formulation exploited as a heuristic approach: we seek for the best feasible solution given a limited and predefined computation time. We run an experimental analysis on instances representing traffic in the Lille Flandres station, France. The results show that the approach tested is very promising, often finding the optimal solution to the instances tackled. Moreover, we show how the performance can be improved by tuning the parameters of the MILP solver.

Key words: railway traffic management, routing, scheduling, mixed integer linear programming

⋆ This paper was not presented at any other revue. Corresponding author P. Pellegrini. Tel. +32-(0)3-20438404. Fax +32-(0)3-20438398.

Email address: {paola.pellegrini, guillaume.douchet, gregory.marliere, joaquin.rodriguez}@ifsttar.fr (Paola Pellegrini, Guillaume Douchet, Grégory Marlière and Joaquin Rodriguez).

1 Introduction

When a disruption occurs, railway traffic management is a hard task which, in the practice, is typically handled by dispatchers without the aid of effective optimization tools. The available tools often propose a solution, optimized to some extent, in terms of train scheduling: it may be convenient to change train priorities at strategic locations, letting a train pass before another one through a switch, for example. However, the best priority order is typically quite difficult to identify. In most of the cases, the available tools are unable to change train predefined routes at junctions, where a route is a sequence of track sections which connect an entry to an exit point: at junctions, multiple lines cross and the presence of switches typically allows several possibilities for crossing it, even if these possibilities are currently unexploited.

In the literature, several papers have proposed optimization algorithms for dealing with the problem of routing and scheduling trains in case of perturbation. These algorithms can be grouped in terms of: train rerouting possibility (exclusion of train rerouting in, e.g., [5] and [7], consideration of train rerouting in, e.g., [4] and [16]), speed variation dynamics consideration (fixed-speed model in, e.g., [3] and [12], variable-speed model in, e.g., [6] and [11]), and interlocking system representation (route-lock sectional-release in, e.g., [13] and [14], route-lock route-release in, e.g., [2] and [15]).

In a previous work [13], we proposed a mixed integer linear programming (MILP) formulation for solving to optimality the problem of scheduling and routing trains in case of perturbation, using a fixed-speed model and representing the route-lock sectional-release interlocking system. This formulation proved to be quite promising, even if the computation time for solving complex instances to optimality is often too long for real-time purposes.

In this paper, we assess the performance of this MILP formulation when it is used as a heuristic approach: we fix *a priori* a limited computation time which is available to the MILP solver. In the following, we will refer to this approach as the MILP-heuristic. We measure the performance of this MILP-heuristic as the difference between the best feasible solution found and the optimal one, found running the MILP solver without imposing a time limit.

We base this assessment on instances obtained by perturbing a real timetable representing traffic in the Lille Flandres station, in the north of France, and we use the IBM ILOG CPLEX Concert Technology for C++ (IBM ILOG CPLEX version 12) [9] as MILP solver. We make a first assessment of the performance of the MILP-heuristic by using the CPLEX default parameter settings. Moreover, we make a further assessment by using CPLEX with the parameter settings selected thanks to an automatic tuning procedure.

The rest of the paper is organized as follows. Section 2 shortly describes the main features of the MILP formulation which we use. Section 3 introduces the automatic tuning tool which we exploit for finding the appropriate CPLEX parameter settings. Section 4 details the experimental analysis, and Section 5 concludes the paper.

2 Main features of the MILP formulation

In the MILP formulation which we consider in this paper, continuous variables represent the relevant times at which trains cross track sections, and binary variables represent precedence relations between trains using common track sections. Moreover, we associate a binary variable to each pair of train and route: these variables are valued one if the train uses the route, and zero otherwise.

Through these variables, we can define multiple sets of constraints:

- time concerning constraints: a train cannot be operated earlier than its arrival in the infrastructure, and it must occupy for a certain amount of time each track section which it crosses. Sometimes, trains with passenger transfers are scheduled (trains in connection); in this case, their arrival and departure time must be coherent;
- constraints for managing delays: delay can be assigned only at locations where a stop or deceleration order can be actually transmitted to a driver in the practice, given the design of the infrastructure;
- constraints due to the change of rolling stock configuration: the arrival and departure time of trains resulting from the turnaround, join or split of one another must be coherent;
- capacity constraints: at most one train uses a track section at a time.

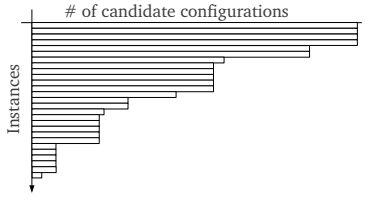

Fig. 1. Graphical representation of the computation performed by one iteration of the racing approach. As the evaluation proceeds, the racing algorithm focuses more and more on the most promising candidates, discarding a configuration as soon as sufficient evidence is gathered that it is suboptimal [1]: the number of candidate configurations tested decreases throughout the iterations.

When an unexpected event occurs, trains may suffer a delay at their arrival in the infrastructure. This delay is typically named primary delay and it may cause the emergence of conflicts within the infrastructure itself. The additional delay due to these conflicts is named secondary delay. The objective function which we consider here is the minimization of the total secondary delay.

For a detailed description of the problem and the formulation we refer the reader to Pellegrini et al. [13].

3 Parameter tuning

The aim of parameter tuning is identifying the appropriate parameter settings for effectively tackling a class of instances through an algorithm. In our context, the aim is to identify the appropriate CPLEX parameter settings for tackling instances representing traffic at Lille Flandres station through the formulation sketched in Section 2.

Selecting the appropriate parameter settings for tackling a class of instances through an algorithm is an optimization problem in itself. In fact, the set of possible configurations typically has a very large cardinality, where a configuration is a combination of parameter settings, one setting for each parameter to be tuned. Performing parameter tuning corresponds to exploring the space of configurations to identify the optimal (or a good suboptimal) one: the objective function value associated to each point in the configuration space is the performance of the algorithm to be tuned when run with the configuration itself. The difficulty of this optimization process increases with the number of possible configurations and with the non-linearity of the relations between parameters. A further element of complexity is the measurement of the performance of the algorithm to be tuned: such a performance on a whole class of instances can be only estimated, and this estimation requires a large number of tests on as many instances as possible.

Iterated Race for Automatic Algorithm Configuration (IRace) [10] is an effective algorithm for tackling the parameter tuning problem. It is an iterated racing algorithm [1] for choosing a candidate configuration out of a set of possible ones. IRace performs a number of iterations, in each of which the optimization algorithm to be tuned is run multiple times for testing its performance on several instances, given a set of candidate configurations. On the basis of the results achieved after some instances, a configuration can be discarded if it appears suboptimal: for each instance (each representing one step of the race) the ranking of the results obtained using the different configurations is computed and a statistical test is performed for deciding whether to discard some candidates or not. The set of configurations considered at a specific step h contains all the candidates that survived after step $h-1$ (Figure 1). IRace bases the identification of candidates to be discarded on the Friedman two-way analysis of variance by ranks [8]. An important advantage offered by this statistical test is connected with the nonparametric nature of a test based on ranking, which does not require to formulate hypothesis on the distribution of the observations.

Starting from a user-supplied domain for each parameter setting, IRace automatically selects the number of candidates to be tested at each iteration: it obtains them through a random sample of the space of possible configurations. The probability distribution considered at the first iteration is uniform. Subsequently, the probability is updated: the distribution is biased to increase the probability associated to configurations similar to those which performed best so far.

The resources available for the tuning process are defined in terms of total number of runs, where a run is one execution of the algorithm with a given configuration on one instance, or total computation time. When the available resources have been completely exploited, the tuning process finishes and the candidate that appears the best is returned.

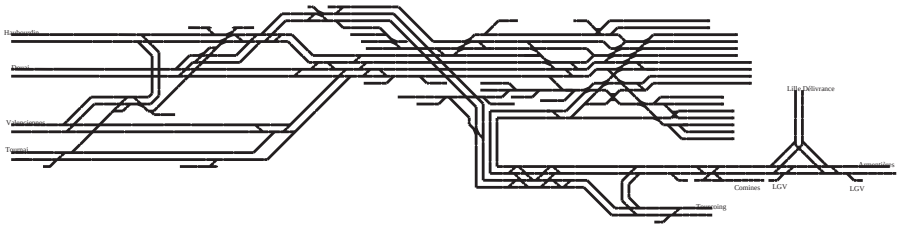

Fig. 2. Infrastructure of the control area including the Lille-Flandres station.

4 Experimental analysis

In this experimental analysis, we tackle 30 instances representing traffic in the Lille Flandres station, France (Figure 2). They are obtained by perturbing a real one-day timetable. In particular, starting from the original timetable, we impose a primary delay to 20% of trains: we randomly select the trains to be delayed and we randomly draw their delay in the interval between 5 and 15 minutes [11]. Both these random selections are based on uniform probability distributions. By replicating the random assignment of primary delay 30 times, we obtain 30 different perturbed one-day timetables. For each of these 30 perturbed one-day timetable, we consider four sets of instances differing in the time horizon: 30, 40, 50 and 60 minutes. Each time horizon includes the peak-time of the day, ending at 7:30 am: we consider all trains which arrive in the infrastructure between 7:00 and 7:30 am, in the example of the 30-minute instances.

The allowed computation time is 180 CPU seconds for each instance, independently on the size of the time horizon considered for the set of instances. We run the experiments on Intel Xeon 24 core 2.3 GHz processor with 32 GB RAM, under Linux Ubuntu distribution version 12.04. In this setup, we consider only sequential computation for CPLEX. We compare the performance of the MILP-heuristic when applied with the default parameter settings and with the tuned one.

For tuning CPLEX parameters, we generate a additional 100 instances for each time horizon, and we let IRace execute 1000 runs. We run a separate tuning for each set of instances. After few exploratory runs, it appeared evident that a specific setting of two parameters allows a strong improvement of the performance of the formulation on the instances representing traffic at Lille Flandres. Hence, we did not include these parameters in the tuning phase and we imposed two fix non-default parameters: 4 for the MIP emphasis switch parameter (MIPEmphasis) and 2 for the MIP dynamic search switch (MIPSearch). The former controls the trade-offs between speed, feasibility, optimality, and moving bounds. The latter defines the search strategy adopted by CPLEX, selecting between dynamic search and conventional branch-and-cut. Through IRace, we tune four parameters:

- MIP probing level (Probe): it sets the amount of probing on variables to be performed before MIP branching;
- MIP dive strategy (DiveType): it controls the type of dive to perform in the branch-and-bound tree;
- symmetry breaking (Symmetry): it decides whether symmetry breaking reductions will be automatically executed in the MIP model during the preprocessing phase;
- row multiplier factor for cuts (CutsFactor): it limits the number of cuts that CPLEX can add to the model: this number is equal to the number of constraints of the original model multiplied by the value of this parameter.

Table 1
Parameter settings available for tuning. In italics: default settings.

param.	settings	param.	settings	param.	settings	param.	settings
Probe	-1, *0*, 1, 2, 3	Symmetry	*−1*, 0, 1, 2, 3, 4, 5	DiveType	*0*, 1, 2, 3	CutsFactor	2, 3, *4*, 5, 6, 7, 8

Table 2
Summary of the main characteristics of the instances tackled: mean values on the 30 instances of each set. In the last column, parameter settings selected through tuning: {Probe, Symmetry, DiveType, CutsFactor}.

hor.	trains	trains with prim. del.	total prim. del. (sec)	parameter setting	hor.	trains	trains with prim. del.	total prim. del. (sec)	parameter setting
30	22	3	609	{−1, −1, 3, 7}	40	29	5	611	{3,−1,3,5}
50	36	6	617	{−1,−1,0,5}	60	45	8	619	{−1,3,2,5}

Table 1 reports the settings tested for each parameter (for a detailed explanation of the meaning of each setting, we refer the reader to the CPLEX manual [9]). We indicate in italics the default settings. In turn, Table 2

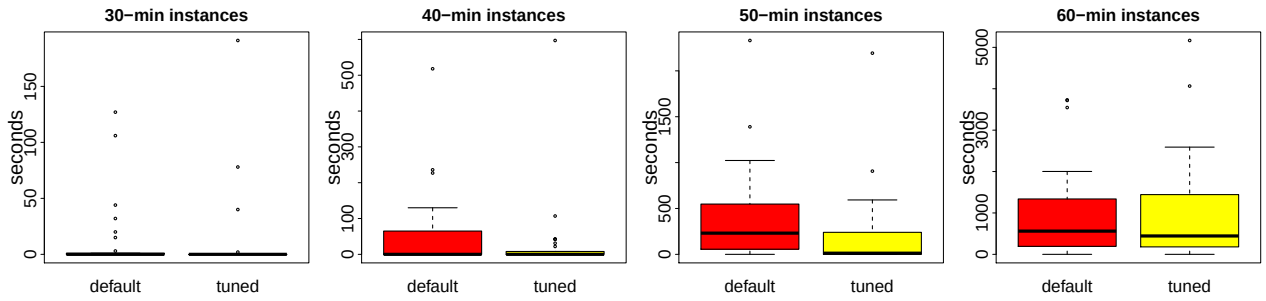

Fig. 3. Distribution of the absolute error made by the two versions of the MILP-heuristic compared to the optimum.

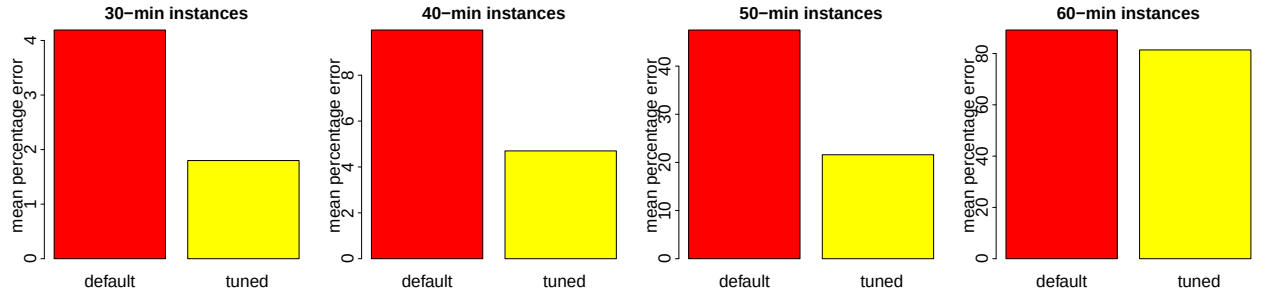

Fig. 4. Mean percentage error made by the two versions of the MILP-heuristic compared to the optimum.

reports the main characteristics of each instance set and the parameter setting selected by IRace.

The distribution of the absolute errors made by the two versions of the MILP-heuristic, namely the default and the tuned one, shows a quite evident improvement achieved through tuning. Figure 3 reports these distributions through boxplots: each box represents the distribution of the observations corresponding to the 30 perturbed instances tackled. The horizontal line within the box represents the median of the distribution, while the extremes of the box represent the first and third quartiles, respectively; the whiskers show the smallest and the largest non-outliers in the data-set and dots correspond to the outliers. Figure 4 shows the mean percentage error made by the two versions of the MILP-heuristic for each set of instances. The tuned version is always preferable with respect to the default one when the error is computed in percentage terms. This relation is less evident when considering the absolute error. It is due to the peculiar behavior of the two versions: the default version typically makes a larger relative error than the tuned one on instances with small value of the optimum, resulting in a small absolute error and a large percentage one. For example, consider two instances with optimum 10 and 100, respectively. Let the default version make an absolute error of 5 and 10 on the two instances, and the tuned one make an absolute error of 2 and 18. The mean absolute error will be 7.5 for the default version and 10 for the tuned one. However, the corresponding mean percentage error will be 30% and 20%, respectively. Anyway, despite the presence of cases as this one, the median of the distribution of the error of the tuned version is always smaller than the one of the default version; the difference is statistically significant for both the 40 and the 50-minute instances, according to the Wilcoxon rank-sum test with a confidence level of 0.95.

Table 3 reports the mean absolute and percentage error made by the two versions, as well as the number of instances in which the corresponding version obtains the best results. This further representation of the results

Table 3
Mean absolute and percentage error, number of instances in which each version outperforms the other one (instances in which best results) and on which each version attains the optimal solution (instances solved to optimality).

horizon	mean absolute error		mean percentage error		instances in which best results		instances solved to optimality	
	default	tuned	default	tuned	default	tuned	default	tuned
30	12	10	4%	2%	3	6	22	26
40	51	30	10%	5%	3	9	19	20
50	393	202	47%	22%	6	20	5	11
60	986	470	89%	81%	14	15	1	1

supports the previous statement according to which parameter tuning appears promising, even if in the current experiments it did not bring to striking improvements. The table also indicates the number of instances in which each version attains the optimal solution. This number is very high for the sets of smaller instances and it decreases, as expected, when the duration of the time horizon considered increases.

As for what concerns the overall performance of the tuned MILP-heuristic, we consider the results quite encouraging: in the most challenging set of instances including about 45 trains with a primary delay of more than 600 seconds, the mean absolute error is 470 seconds.

5 Conclusions

In this paper, we analyzed the performance of a heuristic algorithm for tackling the problem of routing and scheduling trains in case of perturbation. This heuristic is based on a mixed integer linear programming formulation: we limit the solver execution time and we measure the performance of the algorithm in terms of error made when comparing the best solution found within the time limit to the optimum. We compare the performance of the heuristic with the solver default parameters and after tuning them though an automatic procedure.

We ran experiments on instances representing traffic in the Lille Flandres station, France, considering alternative time horizons of 30, 40, 50 and 60 minutes at peak time. We limit the computation time to 180 CPU seconds. The results show that the heuristic achieves encouraging results, especially when the solver parameters are tuned. It can deal with difficult realistic instances even when a short computation time is available, as must be done in the real-time railway traffic management. In particular, when instances represent traffic entering the station within a time horizon of up to 40 minutes, the tuned version of the algorithm achieve a performance only few percentage points worse than the optimum.

In future research, we will assess the advantages offered by alternative automatic tuning procedures. Moreover, we will test our heuristic on instances with different characteristics and we will compare its performance with those of other algorithms.

References

[1] M. Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*. Springer, Berlin, Germany, 2009.

[2] G. Caimi, M. Fuchsberger, M. Laumanns, and M. Lüthi. A model predictive control approach for descrete-time rescheduling in complex central railway station approach. *Computers & Operations Research*, 39:2578–2593, 2012.

[3] F. Corman, A. D'Ariano, D. Pacciarelli, and M. Pranzo. A tabu search algorithm for rerouting trains during rail operations. *Transportation Research Part B*, 44:175–192, 2010.

[4] A. D'Ariano, F. Corman, D. Pacciarelli, and M. Pranzo. Reordering and local rerouting strategies to manage train traffic in real-time. *Transportation Science*, 42(4):405–419, 2008.

[5] A. D'Ariano, D. Pacciarelli, and M. Pranzo. A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, 183:643–657, 2007.

[6] A. D'Ariano, M. Pranzo, and I.A. Hansen. Conflict resolution and train speed coordination for solving real-time timetable perturbations. *IEEE Transactions on Intelligent Transportation Systems*, 8(2):208–222, 2007.

[7] M.M. Dessouky, Q. Lu, J. Zhao, and R.C. Leachman. An exact solution procedure to determine the optimal dispatching times for complex rail networks. *IIE Transactions*, 38(2):141–152, 2006.

[8] J.H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19:1–141, 1991.

[9] IBM Corporation. User's manual for cplex. http://publib.boulder.ibm.com/infocenter/cosinfoc/v12r2, 2012.

[10] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The `irace` package, iterated race for automatic algorithm configuration. Technical Report 2011-04, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2011.

[11] R.M. Lusby, J. Larsen, M. Ehrgott, and D.M. Ryan. A set packing inspired method for real-time junction train routing. *Computers & Operations Research*, 2012.

[12] M. Mazzarello and E. Ottaviani. A traffic management system for real-time traffic optimisation in railways. *Transportation Research Part B*, 41:246–274, 2007.

[13] P. Pellegrini, G. Marlière, and J. Rodriguez. A mixed-integer linear program for the real-time railway traffic management problem modeling track-circuits. In *5th International Seminar on Railway Operations Modelling and Analysis, RailCopenhagen 2013, Copenhagen, Denmark*, 2013.

[14] J. Rodriguez. A constraint programming model for real-time train scheduling at junctions. *Transportation Research Part B*, 41:231–245, 2007.

[15] J. Törnquist and J.A. Persson. N-tracked railway traffic re-scheduling during disturbances. *Transportation Research Part B*, 41:342–362, 2007.

[16] J. Törnquist Krasemann. Design of an effective algorithm for fast response to re-scheduling of railway traffic during disturbances. *Transportation Research Part C*, 20:62–78, 2012.