European Journal of Operational Research 263 (2017) 258–264

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Innovative Applications of O.R.

Using a general-purpose Mixed-Integer Linear Programming solver for the practical solution of real-time train rescheduling

Matteo Fischetti[a,*], Michele Monaci[b]

[a] DEI, Università di Padova, Via Gradenigo 6/A, Padova I-35131, Italy
[b] DEI, Università di Bologna, Viale Risorgimento 2, Bologna I-40136, Italy

A R T I C L E   I N F O

Article history:
Received 21 September 2016
Accepted 30 April 2017
Available online 4 May 2017

Keywords:
Combinatorial optimization
Railways optimization
Train rescheduling
Mixed-Integer Linear Programming
Real-time optimization

A B S T R A C T

At a planning level, train scheduling consists of optimizing the routing and scheduling for a set of trains on a railway network. In real-time operations, however, the planned schedule constantly needs to be verified and possibly updated due to disruptions/delays that may require train rerouting or cancelation. In practice, an almost immediate reaction is required when unexpected events occur, meaning that trains must be rescheduled in a matter of seconds. This makes the time-consuming optimization tools successfully used in the planning phase completely inadequate, and ad-hoc (heuristic) algorithms have to be designed.

In the present paper we develop a simple approach based on Mixed-Integer Linear Programming (MILP) techniques, which uses an ad-hoc heuristic preprocessing on the top of a general-purpose commercial solver applied to a standard event-based MILP formulation. A computational analysis on real cases shows that our approach can be successfully used for practical real-time train rescheduling, as it is able to deliver (almost) optimal solutions within the very tight time limits imposed by the real-time environment.

1. Introduction

Train scheduling consists of routing and scheduling a set of trains traveling on a railway network, satisfying some operational constraints and optimizing some measure of the efficiency of the infrastructure. Different models have been proposed in the literature for describing this problem: the reader is referred to Szpigel (1972), Jovanovic and Harker (1991), Oliveira and Smith (2000), Caprara, Fischetti, and Toth (2002) and Mascis and Pacciarelli (2002), among others, and to Caprara, Kroon, Monaci, Peeters, and Toth (2007, chap. 3) and D'Ariano, Samà, D'Ariano, and Pacciarelli (2014) for recent surveys on this topic.

In principle, the above models could be used not only for planning but also at an operational real-time level. In this latter context, the nominal train timetable is fixed but some unexpected events (e.g., delays and/or disruptions) occur on the network, introducing possible conflicts and making the nominal schedule infeasible. In this case, suitable repair operations must be quickly decided and implemented to recover feasibility, a task that often involves several trains with a redefinition of their timetable and/or routing.

Typically, rescheduling has the main objectives to produce a conflict-free schedule that is as close as possible to the *ideal* (or nominal) planned timetable, while minimizing operational costs related to the rescheduled trains. To restore the timetable within a reasonable time horizon, rescheduling usually needs to consider only a subset of the trains, meaning that the size of the rescheduling problem is not as large as in the planning phase. However, a main complication is that recovery actions must be decided in a very short time, as they must be validated by a human operator and henceforth implemented. Thus, an automatic support decision system must be able to propose conflict-free solutions in real-time. Typically, the available time for producing a new schedule after the originating event has been detected ranges from 2 to 10 seconds, depending on the time-window size and on the number of affected trains.

In this paper we describe the outcome of our collaboration with Alstom Ferroviaria SpA (just Alstom in what follows), a global leader in the world for railways infrastructure. Alstom developed in recent years a proprietary decision support system, called ICONIS, to monitor and control railway traffic in stations and railway lines. In 2013, Alstom activated international collaborations with experts in the field, with the aim of improving its dispatching system. In

* Corresponding author.
E-mail addresses: matteo.fischetti@unipd.it (M. Fischetti), michele.monaci@unibo.it (M. Monaci).

http://dx.doi.org/10.1016/j.ejor.2017.04.057
0377-2217/© 2017 Elsevier B.V. All rights reserved.

particular, a set of real instances has been provided to all collaborators, representing simulated disruption within a given time horizon for a line nearby the city of London.

The test bed is made by 29 instances: 10 small (disruption time horizon of 15 minutes), 9 medium (30 minutes) and 10 large (60 minutes) instances. Even the small instances involve reordering and rerouting of several trains (up to 41), so they are far from trivial. The interested reader is referred to D'Ariano et al. (2014) for a description of this project and of its challenges. Reasonable time limits for computation were set by Alstom to 2 seconds (small instances), 5 seconds (medium instances), and 10 seconds (large instances), on "any reasonable hardware"—meaning even a local cluster with tens PCs running in parallel, if required.

In view of our specific integer programming expertize, we were asked to provide a feasibility study to answer a challenging question: In view of the greatly improved technology, can a modern general-purpose Mixed-Integer Linear Programming (MILP) solver be successful in performing real-time rescheduling in a practically effective way? This paper answers to that question essentially in the affirmative. We found that, even for the small instances, modern MILP technology is still *not* able to solve the rescheduling models within the few-second time frame required. However, to our own pleasant surprise we found that simple heuristic preprocessing operations have a dramatic effect in the MILP performance, and allow one to achieve the orders-of-magnitude speedup required in practice.

We want to stress here that we do not claim that our approach beats all the sophisticated ad-hoc methods from the literature, and in fact we do not compare it with any such methods (many of which are proprietary). As a matter of fact, similar methods have already been considered; see, e.g., (Cacchiani et al., 2014; Fang, Yang, & Yao, 2015; Pellegrini, Marliere, Pesenti, & Rodriguez, 2015). Rather, we want to share with the reader the lesson we learnt from this project—train rescheduling can become much easier once an apparently trivial preprocessing step is applied. We are confident that, in spite of its simplicity, this is an important message that will attract the attention of researchers and practitioners working in the field.

The remaining part of the paper is organized as follows. In Section 2 we sketch the MILP model proposed by Mascis and Pacciarelli (2002), D'Ariano, Corman, Pacciarelli, and Pranzo (2008) and Mannino and Mascis (2009) that we used in our experiments, and computationally analyze it by using a state-of-the-art commercial MILP solver (IBM ILOG Cplex 12.6.2). In Section 3, we introduce two heuristic preprocessing procedures based on bound tightening and variable fixing, and analyze their practical performance in Section 4. Some comments are finally drawn in Section 5.

2. State of the art

As already mentioned, train (re)scheduling can be modeled by using the MILP paradigm. In our approach, we use a standard event-based model for job-shop scheduling problems (Balas, 1969) where variables are associated with the time instant in which relevant operations take place.

The model, specialized for train scheduling by Mascis and Pacciarelli (2002), is based on the definition of an *alternative graph* used to model event incompatibilities and involves big-M coefficients to express disjunctive terms. Main variables of the model are binary variables y selecting at most one of the possible routes for each train to be rescheduled, binary variables x associated with (unknown) precedences between events using a same unsplittable resource (e.g., a track or a platform), and continuous variables t expressing the unknown time instant when a certain event will be rescheduled. The model also involves additional continuous variables z_j giving the delay of some relevant events j with respect to planned time. The objective function used in our experiments is the average of all such z_j's. As customary in scheduling models, the alternative graph contains a dummy node ω (say) that represents the end of the schedule. By construction, variable z_ω gives the maximum delay in the schedule—a feature that plays an important role in our approach. The reader is referred to Mascis and Pacciarelli (2002), D'Ariano et al. (2008, 2014) and Mannino and Mascis (2009) for details on the alternative-graph model and on the corresponding MILP model, and to Lamorgese and Mannino (2015) and Lamorgese, Mannino, and Piacentini (2016) for a recent solution approach based on advanced decomposition techniques. For the sake of completeness, the full model we used in our study is reported in the Appendix.

A possible way for rescheduling trains on a rail network is of course to solve the model above by using a general-purpose MILP solver. To the best of our knowledge, however, this approach was never used for real applications as the associated computing times would be by far too large for the real-time setting. Actually, computational experiences reported in the literature (see, e.g., Lamorgese & Mannino, 2015; Lamorgese et al., 2016) show that this approach is not suitable for real-time applications. To verify whether this is still the case when using the most recent and powerful MILP solvers, we performed a preliminary set of experiments aimed at directly solving the instances in our testbed. As mentioned, our dataset includes 10 "small" instances associated with a time horizon of 15 minutes (numbered from 1 to 10), 9 "medium" instances with a 30-minute horizon (numbered from 11 to 19), and 10 "large" instances for which the time horizon is 60 minutes (numbered from 20 to 29). For these instances, a good (almost optimal) solution needs to be computed within 2, 5 and 10 seconds, respectively.

Table 1 reports the main characteristics of each instance, namely the number of trains (NT) to be rescheduled, the total number of alternative routes (NR), the number of variables and constraints in the associated MILP model (#rows and #cols, respectively), and the value z_{BST} of the best known solution. (Using the notation in the Appendix, $NT = |\Theta|$ and $NR = \sum_{\theta\in\Theta} |R_\theta|$.) Computational results refer to one of the best commercial solvers on the market, namely IBM ILOG Cplex in its version 12.6.2 (`Cplex` in the following), with a time limit of 1 hour on a quadcore Intel Xeon E3-1220V2 running at 3.1 gigahertz, with 16 gigabytes of RAM.

We report results for `Cplex` in its default settings as well as in a heuristic version that proved to be more effective in our experiments, in which all cut-generation procedures are disabled. To take advantage of the effects of erraticism of the MILP solver (Fischetti & Monaci, 2014; Lodi & Tramontani, 2014, chap. 2), each version of `Cplex` was run 10 times using 10 different random seeds. For each algorithm, we report the best result obtained among the 10 executions, namely: the minimum computing time t_{TOT} required to solve the instance, the value z_{FIN} of the best solution found, the minimum time t_1 required to compute a feasible solution and the associated value z_1. All times are expressed in wall-clock seconds; entries in bold face in column z_1 refer to the cases where this solution was found within the target computing time of 2/5/10 seconds. Observe that our experiments simulate the behavior of an architecture in which 10 identical PCs are available and used in parallel to solve the same instance, and all PCs are halted whenever one solves the instance to optimality or reaches the 1-hour time limit (columns t_{TOT} and z_{FIN}) or when one finds its first feasible solution (columns t_1 and z_1).

The results of Table 1 confirm that, even using a state-of-the-art MILP solver and 10 fast quadcore PCs, the direct solution of the model above is not a viable option for real-life applications. Indeed, in some cases `Cplex` is not even able to compute a feasible solution within 5 minutes. Even restricting to the smallest instances, there are cases with less than 40 trains for which com-

Table 1
Solution of our instances using Cplex in two different settings by using 10 quadcore PCs (1 hour time limit). Boldface entries meet the target computing time of 2/5/10 seconds.

Instance						Cplex				Cplex heu			
ID	NT	NR	#cols	#rows	z_{BST}	t_{TOT}	z_{FIN}	t_1	z_1	t_{TOT}	z_{FIN}	t_1	z_1
1	33	55	4508	8528	0.90	1.57	0.90	1.52	**13.46**	1.37	0.90	1.16	**49.55**
2	37	46	3499	6289	15.76	0.89	15.76	0.51	**68.57**	0.36	15.76	0.21	**30.49**
3	41	62	5331	10,127	151.02	8.86	151.02	2.41	290.28	9.28	151.02	1.61	**159.23**
4	38	58	10,056	19,653	8.52	27.22	8.52	4.41	9.08	42.99	8.52	2.91	14.01
5	40	43	2867	4548	15.51	0.50	15.51	0.47	**15.51**	0.55	15.51	0.34	**95.93**
6	34	42	5742	10,606	8.82	2.55	8.82	2.55	8.82	2.03	8.82	1.99	**8.82**
7	36	44	4994	9187	13.44	1.42	13.44	1.03	**58.46**	1.51	13.44	1.46	**13.67**
8	32	40	4462	8216	3.86	0.71	3.86	0.62	**3.86**	1.00	3.86	0.97	**3.86**
9	33	45	9363	17,827	40.50	51.44	40.50	51.39	40.50	18.92	40.50	16.14	52.85
10	37	48	6994	13,226	90.78	17.69	90.78	12.80	128.78	9.96	90.78	3.30	90.78
11	40	73	14,506	28,139	3.41	31.88	3.41	6.11	29.26	9.00	3.41	4.11	**34.93**
12	48	79	15,182	197,594	44.11	490.36	44.11	20.17	223.53	485.57	44.11	9.46	241.85
13	54	72	16,232	30,958	25.36	1410.71	25.36	197.49	194.66	948.29	25.36	8.48	41.45
14	52	59	8529	15,181	10.75	23.10	10.75	3.96	**66.17**	9.95	10.75	2.63	**20.63**
15	49	53	7318	12,559	24.30	33.19	24.30	3.88	**140.66**	10.05	24.30	2.80	**24.59**
16	42	51	11,996	22,459	67.53	90.12	67.53	13.59	141.47	34.68	67.53	7.34	1024.36
17	44	57	12,683	24,020	23.84	116.57	23.84	18.21	54.42	51.62	23.84	27.92	135.25
18	43	54	9081	16,778	11.91	18.61	11.91	2.26	**45.98**	10.65	11.91	2.85	**116.00**
19	41	55	15,145	28,876	58.81	170.77	58.81	147.36	69.41	123.07	58.81	61.85	97.15
20	59	65	16,431	30,025	189.35	109.98	189.35	11.64	198.49	33.23	189.35	4.52	**237.52**
21	57	83	34,339	67,124	7.36	265.14	7.36	28.86	19.78	94.92	7.36	17.80	10.92
22	58	87	58,036	114,982	14.62	2529.88	14.62	710.88	31.06	1938.17	14.62	332.58	65.21
23	64	80	26,061	49,277	37.14	656.77	37.14	105.01	38.64	523.92	37.14	187.67	38.92
24	58	91	41,827	81,587	8.11	542.45	8.11	469.67	22.49	174.25	8.11	90.17	20.82
25	66	97	34,052	762,030	44.75	3600.00	121.53	355.92	121.53	3600.01	45.48	103.59	299.78
26	90	128	52,542	101,789	69.11	3600.00	98.87	1141.91	98.87	3600.04	69.11	51.64	95.23
27	59	64	19,014	34,616	34.57	234.90	34.57	17.45	120.74	258.49	34.57	17.19	68.56
28	85	105	56,520	108,630	–	3600.00	–	–	–	3600.00	–	–	–
29	52	59	8529	15,181	10.75	23.08	10.75	4.04	**66.17**	9.93	10.75	2.64	**20.63**

puting a feasible solution takes more than 10 seconds. Results are slightly better for the heuristic version of Cplex, but even in this case a feasible solution can be found within the target computing time only in 13 out of 29 cases. Though the use of faster computers could in principle reduce the reported times, it is clear that a speedup of 1–2 orders of magnitude would be required to make this approach sufficiently reliable to be used in operation.

Finally, note that none of the 10+10 Cplex runs could find a feasible solution for instance 28 even within the 1-hour time limit, hence we removed this instance from our testbed in our next experiments.

3. Model preprocessing

In the attempt of improving Cplex's heuristic performance, we implemented and tested a number of ideas. Eventually, we found a satisfactory implementation that allowed us to meet the tight real-time time limits imposed by Alstom. We then followed an inverse path and disabled/simplified the new features until we found which are the ideas that really made a difference. It turned out that the performance boost was due to just two very simple preprocessing operations, which we describe in the following.

3.1. Heuristic bound tightening

A main reason for the hardness of the MILP model is the presence of big-M coefficients that are used to activate/deactivate constraints according to the alternative arcs and/or train routes that are selected in the solution. It is well known that the presence of such large coefficients may lead to weak continuous relaxations, making the model unsolvable in practice. Indeed, reducing the value of these coefficients may be highly beneficial for a MILP solver; see, e.g., the very recent experience reported in Belotti et al. (2016).

In event-based models for scheduling applications, big-M reduction boils down to the determination of tight lower and upper bound on variables t_j giving the time instant in which operations take place. As a matter of fact, modern MILP solvers are quite successful in tightening the model coefficients (including variable bounds) by using sophisticated probing and propagation algorithms akin to those successfully used in Constraint Programming.

During our preliminary experience, we performed a number of tests to evaluate the sensitivity of the MILP solver with respect to the imposed variable bounds. We soon discovered that the single upper bound on variable z_ω has a prominent role, in that imposing a tight upper bound on z_ω triggers the tightening of the domain of many other variables. Indeed, as we already observed, in our MILP model variable z_ω gives the maximum delay in the schedule among all relevant events, hence its upper bound immediately affects all the delay variables z_j's. As a consequence, imposing a restrictive upper bound on z_ω typically allows the automatic MILP preprocessing to reduce the number of possible routes for each train, to fix many binary variables, and to tighten some constraint coefficients.

We therefore implemented a simple scheme where we heuristically define a value UB_ω (say) and impose an upper bound to the time of the last operation through the (possibly invalid) constraint $z_\omega \leq UB_\omega$. Reasonably small values of UB_ω likely produce good heuristic solutions, as the objective function for rescheduling problems is naturally driven towards the case where z_ω is close to its smallest-possible value. (This is in fact what happens in the Alstom's instances addressed in our computational study.) In addition, small values of UB_ω allow for large variable fixings and bound tightenings, hence they can have a dramatic effect in reducing computing times. Of course, if UB_ω is too small, the resulting problem becomes infeasible—a property that is typically discovered almost immediately by MILP preprocessing.

As one is interested in automatically estimating the smallest UB_ω that does not lead to infeasibility, in our code we perform a number of iterations starting with a “small” value for UB_ω and increasing this value until a feasible solution is found and the MILP solver is applied with the given time limit. As already observed, infeasible UB_ω's are typically detected very quickly by MILP preprocessing, so the overhead spent in the initial trials with too-small UB_ω's is acceptable and the overall approach is quite effective in practice.

It is worth mentioning that, in a first version of our code, we implemented an ad-hoc heuristic check (based on longest-path computations on an acyclic digraph) to determine whether a given value of UB_ω would produce an infeasible instance, thus quickly discarding too-small values for UB_ω. According to our computational experience, however, the MILP-solver internal preprocessing (that also implements propagation for bound tightening) was equally fast in detecting this kind of infeasibility, so in the end we decided to remove our ad-hoc check to simplify our implementation as much as possible.

3.2. Heuristic variable fixing

The second complication in our MILP model is the presence of many binary variables y associated with alternative routes for each train. These variables appear in big-M constraints, as the precedence constraints must be disabled for paths that are not used in the solution. Therefore, to reduce the complexity of the model one can heuristically reduce, for each train, the number of available routes.

Without loss of generality, we assume that the first available route for each train is the *ideal* route, i.e., the route corresponding to the nominal schedule before disruption. Routing each train along its ideal route is a desirable option, denoted as *non-rerouting* by the planners, as this does not require major changes in the resources required by the train but only adds appropriate delays in the planned timetable. Using different routes, instead, introduces many more complications—though it can be unavoidable in case of major disruptions.

A natural heuristic choice is to force non-rerouting for all trains, meaning that one is allowed to delay some trains at some points but cannot change their paths. However, this choice may lead to problems that are infeasible with respect to the actual UB_ω value, i.e., it may be impossible to reschedule all trains within the given time horizon without rerouting some of them. Thus, an alternative strategy is to choose one route for each train, in a random way, removing all the remaining routes and checking the resulting problem for feasibility.

4. Computational experiments

In this section we report on the performance of our heuristic approach built on top of IBM ILOG Cplex 12.6.2. All the experiments in this section were performed on a set of identical computers, each equipped with the Intel Core i7-2820QM quadcore processor running at 2.3 gigahertz and with 16 gigabytes RAM. This processor is a second-generation Intel Core i7 launched at the beginning of 2011. Note that significantly faster Intel Core i7 seventh-generation processors are available at the time of writing (February 2017), that are about twice as fast as the one used in our tests.

All runs are executed by using `Cplex` in its default mode (including cut generation) unless otherwise specified. In case of multi-thread runs, we selected the opportunistic parallel mode as this is most appropriate in a real-time setting where speed is more important than reproducibility of the results.

To obtain diversified results, we ran our heuristic in parallel on *NPC* (say) independent computers, without communication between the single runs. For example, case $NPC = 4$ simulates a situation in which 4 identical computers are available and the same instance is solved in parallel on each computer, without any communication: when the time limit is reached, all runs are stopped, the individual solutions are compared and the best one is selected. We report results for $NPC = 1, 4$ and 8.

Each run has an input parameter $s = \{1, 2, \dots, NPC\}$ that is used both as the “random seed” for `Cplex` (parameter `CPX_PARAM_RANDOMSEED`, that ensures different runs for each computer), and to produce diversified route-fixing strategies according to the scheme described below. To further diversify the runs, for s odd an aggressive presolve strategy is selected (parameter `CPX_PARAM_PRESLVND` = 3). The runs with $s \in \{1, 8\}$ are executed without any route-fixing. Run with $s = 2$ fixes the first (i.e., ideal) path for each train, while for $s \in \{3, \cdots, 7\}$ the path is selected in a random way using s to initialize the internal random sequence.

Each run uses the mechanism outlined in Section 3.1 to automatically determine a suitable upper bound for z_ω by trying, in sequence, values UB_ω = 600, 900, 1200, 2000, 2500, 3000, 4000, 5000, $+\infty$, until the problem becomes feasible or the time limit is exceeded. To be more specific, at the very beginning of the run we read the input file containing the MILP model, and define the Cplex's parameters (and possibly fix a certain path for each train) according to the input parameter s. Then we fix UB_ω = 600, and invoke the Cplex's MILP solver on the resulting model for the allowed time limit reduced by the wall-clock time spent so far. On return from Cplex, if the time limit was not exceeded we check whether the current model was *proved* to be infeasible by Cplex, in which case we consider the next UB_ω value in the sequence (900, 1200, etc.) and repeat; otherwise we just stop. According to our computational tests, for each instance in our testbed (except of course the removed instance 28), at least one among the parallel runs was able to determine the first feasible UB_ω in very short computing time. Indeed, for the runs on $NPC = 8$ parallel PCs, for 2/3 of our instances the very first attempt (i.e., $UB_\omega = 600$) produced a feasible solution and preprocessing took no time. For some hard instances, one had to try several UB_ω values (at most 7, in our experiments) but the required preprocessing time was always below 0.2 seconds.

It is worth observing that preprocessing time is included in the reported runtimes, hence this phase is taken into account in the time limit. As already explained, each parallel run on a different PC autonomously computes its own UB_ω value, that also depends on the different routes that are fixed. Again, we stress that in our scheme there is absolutely no communication among the parallel runs—such a communication would in fact require some kind of synchronization, and could even result into a slow-down of the overall computation.

Table 2 gives the summary of the results for our heuristic algorithm with different configurations, assuming a time limit TL of 2 (for the small instances 1–10), 5 (for the medium instances 11–19) or 10 (for the large instances 20–29) wall-clock seconds. For each instance and configuration of our algorithm, we report the ratio between the best solution found within the time limit and the optimal (or best known) solution value for the given instance, the latter value being reported in column z_{BST}. The left-hand-side part of the table (columns labeled “Standard PC”) refers to experiments performed on our (relatively old) hardware. As already noticed, computing times could be significantly reduced by using a more recent processor. So in the columns labeled “50% faster PC” we also report the results obtained after 3, 7 and 15 seconds on our hardware, which would meet the Alstom's requirement assuming the use of a more recent processor with a modest 50% speedup.

Reported results are obtained by using $NPC = 1, 4$ and 8 computers, in parallel. On each computer, we consider both the 4-thread option (which is the default setting on our quadcore PCs) as well as the 1-thread option, the latter being interesting for

Table 2
Ratio between the value of the best solution found by our heuristic (within the required time limit TL of 2, 5 and 10 seconds) and the value of the best known solution.

Instance			Standard PC							50% faster PC						
NPC			1	4	8	1	4	8	cpx_8	1	4	8	1	4	8	cpx_8
#threads			1	1	1	4	4	4	cpx_4	1	1	1	4	4	4	cpx_4
ID	z_{BST}	TL	Best solution ratio at time limit							Best solution ratio at time limit						
1	0.90	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	15.76	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
3	151.02	2	–	1.31	1.31	40.20	1.31	1.31	–	1.05	1.05	1.01	1.00	1.00	1.00	1.01
4	8.52	2	–	1.17	1.02	–	1.17	1.02	–	–	1.17	1.02	1.00	1.02	1.02	–
5	15.51	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
6	8.82	2	1.00	1.00	1.00	1.00	1.00	1.00	–	1.00	1.00	1.00	1.00	1.00	1.00	1.00
7	13.44	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
8	3.86	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
9	40.50	2	–	2.65	1.97	–	2.65	2.65	–	–	2.65	1.97	–	2.65	1.97	–
10	90.78	2	–	1.99	1.08	–	1.99	1.08	–	–	1.99	1.08	–	1.99	1.08	–
11	3.41	5	1.00	1.00	1.00	1.00	1.00	1.00	–	1.00	1.00	1.00	1.00	1.00	1.00	–
12	44.11	5	1.27	1.27	1.27	1.20	1.14	1.05	–	1.26	1.26	1.26	1.01	1.01	1.01	–
13	25.36	5	–	2.21	2.21	–	1.86	1.86	–	–	2.20	2.20	–	1.38	1.38	–
14	10.75	5	1.11	1.11	1.11	1.11	1.11	1.11	–	1.11	1.11	1.11	1.11	1.11	1.11	5.42
15	24.30	5	1.01	1.00	1.00	1.00	1.00	1.00	1.01	1.01	1.00	1.00	1.00	1.00	1.00	1.01
16	67.53	5	–	1.93	1.93	–	–	–	–	–	1.93	1.93	–	2.18	2.28	–
17	23.84	5	1.00	1.00	1.00	1.00	1.00	1.00	–	1.00	1.00	1.00	1.00	1.00	1.00	–
18	11.91	5	1.00	1.00	1.00	1.00	1.00	1.00	3.23	1.00	1.00	1.00	1.00	1.00	1.00	1.00
19	58.81	5	–	1.96	1.96	–	2.23	1.90	–	–	1.90	1.77	–	1.90	1.90	–
20	189.35	10	–	–	1.00	–	–	1.00	1.24	–	1.24	1.00	–	1.25	1.00	1.24
21	7.36	10	–	1.79	1.00	–	1.79	1.00	–	–	1.79	1.00	–	1.79	1.00	–
22	14.62	10	1.19	1.19	1.09	1.06	1.09	1.05	–	1.12	1.12	1.08	1.03	1.00	1.00	–
23	37.14	10	–	–	–	–	–	3.62	–	–	3.61	3.61	–	2.65	2.65	–
24	8.11	10	1.00	1.00	1.00	1.00	1.00	1.00	–	1.00	1.00	1.00	1.00	1.00	1.00	–
25	44.75	10	–	–	–	–	–	–	–	1.85	1.85	1.75	1.01	1.52	1.10	–
26	69.11	10	–	1.19	1.19	–	1.10	1.14	–	–	1.19	1.19	–	1.04	1.04	–
27	34.57	10	–	–	–	–	84.69	84.16	–	–	1.39	1.39	–	1.37	1.37	–
29	10.75	10	1.11	1.11	1.11	1.11	1.11	1.11	3.31	1.11	1.11	1.11	1.11	1.11	1.11	1.11

very short time limits as single-thread runs have a faster access to memory and can trigger automatic overclocking of the processor (the so-called "turbo boost" feature of Intel Core i7).

To better evaluate the practical impact of our preprocessing and train-path fixing strategies, Table 2 also reports the performances of our code when these two latter features are disables, but all remaining Cplex's parameters are set as in our own code. The two columns labeled "'cpx_8/cpx_4" in the table (one for "Standard PC" and the other for "50% Faster PC") refer to runs with $NPC = 8$ PCs and #threads = 4, and are intended be compared with the two columns labeled "8/4" that appear on their left in the same table.

Results for $NPC = 1$ (hence $s = 1$) clearly show the impact of the variable bound $z_\omega \leq UB_\omega$ described in Section 3.1: using a single PC (4 threads) already allows one to compute, within the given time limit, a feasible solution in 16 cases out of 29—remind that this figure was equal to 13 for the heuristic version of `Cplex` executed on 10 PCs. Further improvements are obtained when heuristically fixing some paths in the solution. Using $NPC = 8$ standard PCs, we are able to compute a feasible solution within the given time limit for 26 instances. In addition, in most cases the value of the solution found is very close to the optimal (or best known) solution value. In only 3 cases the ratio is about 2, which is considered acceptable by Alstom's experts.

Using a more recent architecture (columns "50% Faster PC") and $NPC \in \{4, 8\}$ PCs (both 1-thread and 4-threads), we were able to find a very good solution in most of the cases within the time limit—except for instance 28, for which no feasible solution could be found even by the best-tuned `Cplex` on 10 PCs with a one-hour time limit, hence it is not reported in the table. The solution quality at the time limit is very good in all cases.

A comparison of columns "'cpx_8/cpx_4" with their "8/4" counterpart clearly shows the very significant performance improvement obtained by the preprocessing and path-fixing strategies described in the present paper. Indeed, removing these latter features leads to a much less effective heuristic that is able to find a feasible solution in 12 (out of 28) cases only, even in the most powerful "50% Faster PC" hardware setting with 8 4-thread computers—while our proposed heuristic was successful in all those 28 instances.

Note that, as expected, using more PCs typically improves the heuristic performance. However, it can be the case that a worse solution is obtained with a larger number of PCs, due to the use of the opportunistic parallel mode (compare, e.g., the "50% Faster PC" entries associated with instance 16).

Though the motivation for our study was to evaluate the possible use of MILP technology to solve the train rescheduling problem, we mention here that alternative approaches that do not make an explicit use of a MILP model have been tested on our benchmark. In particular, the results reported in (D'Ariano et al., 2014) show that the integration of the branch-and-bound algorithm of D'Ariano, Pacciarelli, and Pranzo (2007) with the tabu search scheme of Corman, D'Ariano, Pacciarelli, and Pranzo (2010), allows for the solution of all our instances within 30 seconds of computation on an Intel Core 2 Duo E6550@2.33 gigahertz. However, D'Ariano et al. (2014) only report the time needed to compute the best solution found (9 seconds, on average) and provide no detailed information about the quality of the solution found within the required time limit of 2, 5 and 10 seconds. As a consequence, a direct comparison with our approach is not possible, not even to report the number of instances for which a feasible solution was found within the tight given time limits. In any case, we believe it is fair to conclude that both our method and the one by D'Ariano et al. (2014) are viable options in a practical setting.

5. Conclusions

We have addressed the possibility of using a black-box MILP solver for the train rescheduling problem in a real-time context.

Our computational experience, as well as that reported by many other authors including D'Ariano et al. (2014), Lamorgese and Mannino (2015) and Lamorgese et al. (2016), has shown that the direct usage of such a solver may require a computing time that is not compatible with the allowed time limits.

In the present paper, we have presented a simple heuristic preprocessing framework (to be implemented on the top of the MILP solver), that leads to a dramatic performance improvement and makes the black-box MILP approach viable in practice. We have also shown how our heuristic can be effectively run in parallel on a set of independent computers.

We have reported experiments on a real-world testbed provided by Alstom Ferroviaria SpA (a global leader in the world for railways infrastructure) for an UK railway network. Our results show that the proposed approach qualifies as an effective way to solve the rescheduling problem in real-time. Indeed, our heuristic algorithm has been able to compute high-quality solutions within computing times of just few seconds, which are compatible with real-time requirements. Finally, we have also shown that the same MILP solver (run on the same hardware and with exactly the same parameter configuration) without our preprocessing phase has a much degraded performance, in that it failed in even finding any feasible solution in many cases—hence it cannot be considered a reliable tool to be used in practice.

Future work should address the applicability of similar ideas to different real-time train rescheduling contexts. Indeed, in our preprocessing phase we did not take into account the relative importance of trains (e.g., passenger trains against freight trains), while in practice one can allow for large delays for one category in order to reduce delays for the other. Also of interest is the generalization of our approach to more complex objective functions, e.g., to the non-convex step functions that are often used by train operators to assess punctuality.

Acknowledgments

We want to thanks Alstom Ferroviaria SpA for proving us all data used in the experiments. Special thanks are due to Massimo Rosti and Davide Nucci for their helpful support. MILP instances in our experiments have been provided by D'Ariano et al. (2014). This research was partially supported by MiUR, Italy (PRIN project "Mixed-Integer Nonlinear Optimization: Approaches and Applications"), and by the Vienna Science and Technology Fund (WWTF) through Project ICT15-014. Thanks are also due to three anonymous referees for their constructive comments.

Appendix. Mathematical model

We next sketch the alternative graph model for train rescheduling, and refer to D'Ariano et al. (2008); D'Ariano et al. (2014); Mannino and Mascis (2009); Mascis and Pacciarelli (2002) for details.

Let us denote by $\Theta = \{1, \ldots, NT\}$ the index set of the trains to be rescheduled, e.g., those that are traveling on a certain area within a given time horizon. Each train θ has an associated set R_θ of alternative routes, characterized by different operations to be performed in different points of the network. In the corresponding *alternative graph* $G = (N, F \cup A)$, node set N contains a node for each operation associated with any alternative routes of the trains, plus two dummy nodes α and ω that act as source and terminal nodes, respectively. The arc set $F \cup A$ is instead constructed as follows. For each train $\theta \in \Theta$ and route $r \in R_\theta$, we have a set $F_{r\theta}$ of *fixed arcs*, each arc corresponding to a pair of operations that must be executed according to a given order (fixed and known in advance). In addition, for each pair of routes (r_1, r_2) associated with two different trains, there is a set $\mathcal{A}_{r_1 r_2}$ of pairs of *alternative arcs*, each pair including an arc associated with route r_1 and an arc associated with r_2. These arcs are mutually incompatible in that they require the use of a same resource, hence they must be scheduled according to a certain precedence (to be decided by the optimizer). In the arc set, $F = \bigcup_{\theta \in \Theta, r \in R_\theta} F_{r\theta}$ contains all the fixed arcs and models all known precedences among operations (including the obvious ones that define the journey of a train along a certain route), whereas A includes all the alternative arcs and corresponds to unknown precedences.

The MILP model we use is based on the definition of the following decision variables:

$$y_{r\theta} = \begin{cases} 1 & \text{if train } \theta \text{ is routed through route } r; \\ 0 & \text{otherwise} \qquad \theta \in \Theta;\, r \in R_\theta \end{cases} \tag{1}$$

$$x_{kjhi} = \begin{cases} 1 & \text{if arc } (k, j) \text{ is selected while arc } (h, i) \text{ is not}; \\ 0 & \text{otherwise} \qquad \{(k, j), (h, i)\} \in \mathcal{A} \end{cases} \tag{2}$$

where $\mathcal{A} = \bigcup_{\theta_1, \theta_2 \in \Theta, \theta_1 < \theta_2, r_1 \in R_{\theta_1}, r_2 \in R_{\theta_2}} \mathcal{A}_{r_1 r_2}$ contains all the pairs of incompatible arcs.

In addition the model uses, for each operation j, two variables t_j and z_j to represent the time instant in which the operation takes place and the associated delay, respectively. As to the dummy event ω, variable z_ω is actually used to indicate the maximum delay among all operations.

The model can be stated as follows:

$$\min f(z) \tag{3}$$

$$\sum_{r \in R_\theta} y_{r\theta} = 1, \quad \theta \in \Theta \tag{4}$$

$$z_j \geq t_j - p_j, \quad j \in N \tag{5}$$

$$t_\alpha = 0 \tag{6}$$

$$t_j - t_i + M(1 - y_{r\theta}) \geq d_{ij}, \quad \theta \in T, r \in R_\theta, (i, j) \in F_{r\theta} \tag{7}$$

$$\begin{aligned} & t_i - t_h + M(1 - y_{r_1\theta_1}) + M(1 - y_{r_2\theta_2}) + Mx_{kjhi} \\ & \geq d_{hi}, \qquad \theta_1, \theta_2 \in \Theta, \theta_1 < \theta_2, r_1 \in R_{\theta_1}, r_2 \in R_{\theta_2}, \\ & \qquad \{(k, j), (h, i)\} \in \mathcal{A}_{r_1 r_2} \end{aligned} \tag{8}$$

$$\begin{aligned} & t_j - t_k + M(1 - y_{r_1\theta_1}) + M(1 - y_{r_2\theta_2}) + M(1 - x_{kjhi}) \\ & \geq d_{kj}, \qquad \theta_1, \theta_2 \in \Theta, \theta_1 < \theta_2, r_1 \in R_{\theta_1}, r_2 \in R_{\theta_2}, \\ & \qquad \{(k, j), (h, i)\} \in \mathcal{A}_{r_1 r_2} \end{aligned} \tag{9}$$

$$z_\omega \leq UB_\omega \tag{10}$$

$$z_j \geq 0, \quad j \in N \tag{11}$$

$$x_{kjhi} \in \{0, 1\}, \quad \{(k, j), (h, i)\} \in \mathcal{A} \tag{12}$$

$$y_{r\theta} \in \{0, 1\}, \quad \theta \in \Theta, r \in R_\theta \tag{13}$$

where $f(\cdot)$ denotes some linear objective function, to be defined by the user, and M denotes a very large positive value (big-M). In the model, each arc (i, j) has an associated duration d_{ij} representing the minimum time difference between operations i and j, in case both are executed. In addition, each operation j has associated an input "release time" p_j that is used to define the associated delay z_j in (5). In our experiments, the objective function $f(z) = \frac{1}{N}\sum_{j\in N} z_j$ measures the average delay in the final solution.

Constraints (4) ensure that exactly one route is selected for each train, while (5) define the delay variables z_j used in the objective function. Constraints (7) model the implications "$y_{r\theta} = 1 \rightarrow t_j \geq t_i + d_{ij}$", i.e., they force the precedences related to the fixed arcs for selected routes, while they are deactivated in case $y_{r\theta} = 0$. Similarly, because of the big-M coefficients, constraints (9) and (10) are only active when $y_{r_1\theta_1} = y_{r_2\theta_2} = 1$, in which case they impose the disjunction "$(t_i \geq t_h + d_{hi}) \vee (t_j \geq t_k + d_{kj})$" corresponding to the incompatible arc pair $\{(k, j), (h, i)\} \in \mathcal{A}_{r_1 r_2}$. Finally, (10) imposes an input upper $UB_\omega \in \Re_+ \cup \{+\infty\}$ on variable z_ω representing the maximum delay occurred in the schedule—this is due to an ad-hoc definition of the duration $d_{i\omega}$ of the arcs (i, ω) entering the terminal node ω.

References

Balas, E. (1969). Machine sequencing via disjunctive graphs: An implicit enumeration algorithm. *Operations Research, 17*(6), 941–957. doi:10.1287/opre.17.6.941.

Belotti, P., Bonami, P., Fischetti, M., Lodi, A., Monaci, M., Nogales-Gómez, A., et al. (2016). On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications, 65*(3), 545–566.

Cacchiani, V., Huisman, D., Kidd, M., Kroon, L., Toth, P., Veelenturf, L., et al. (2014). An overview of recovery models and algorithms for real-time railway rescheduling. *Transportation Research Part B, 63*, 15–37.

Caprara, A., Fischetti, M., & Toth, P. (2002). Modeling and solving the train timetabling problem. *Operations Research, 50*(5), 851–861.

Caprara, A., Kroon, L., Monaci, M., Peeters, M., & Toth, P. (2007). Passenger railway optimization. In C. Bernhart, & G. Laporte (Eds.), *Handbook in OR & MS: Vol. 14* (pp. 129–197). Elsevier, North-Holland.

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

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

D'Ariano, A., Pacciarelli, D., & Pranzo, M. (2007). A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research, 183*(2), 643–657.

D'Ariano, A., Samà, S., D'Ariano, P., & Pacciarelli, D. (2014). Evaluating the applicability of advanced techniques for practical real-time train scheduling. *Transportation Research Procedia, 3*, 279–288.

Fang, W., Yang, S., & Yao, X. (2015). A survey on problem models and solution approaches to rescheduling in railway networks. *IEEE Transactions on Intelligent Transportation Systems, 16*(6), 2997–3016.

Fischetti, M., & Monaci, M. (2014). Exploiting erraticism in search. *Operations Research, 62*(1), 114–122.

Jovanovic, D., & Harker, P. (1991). Tactical scheduling of rail operations: The scan i system. *Transportation Science, 25*(1), 46–64.

Lamorgese, L., & Mannino, C. (2015). An exact decomposition approach for the real-time train dispatching problem. *Operations Research, 63*(1), 48–64.

Lamorgese, L., Mannino, C., & Piacentini, M. (2016). Optimal train dispatching by Benders-like reformulation. *Transportation Science, 50*(3), 910–925.

Lodi, A., & Tramontani, A. (2014). Performance variability in mixed-integer programming. In *Theory driven by influential applications* (pp. 1–12). http://pubsonline.informs.org/doi/abs/10.1287/educ.2013.0112. doi:10.1287/educ.2013.0112.

Mannino, C., & Mascis, A. (2009). Optimal real-time traffic control in metro stations. *Operations Research, 57*(4), 1026–1039.

Mascis, A., & Pacciarelli, D. (2002). Job shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research, 143*(3), 498–517.

Oliveira, E., & Smith, B. (2000). A job-shop scheduling model for the single-track railway scheduling problem. *Technical report: Research report 2000.21*. School of Computing, University of Leeds.

Pellegrini, P., Marliere, G., Pesenti, R., & Rodriguez, J. (2015). RECIFE-MILP: An effective MILP-based heuristic for the real-time railway traffic management problem. *IEEE Transactions on Intelligent Transportation Systems, 16*(5), 2609–2619.

Szpigel, B. (1972). Optimal train scheduling on a single track railway. *Operational Research*, 343–351.