Applied Timetabling for Railways: Experiences with Several Solution Approaches

Julian Jordi [a], Ambra Toletti [a], Gabrio Caimi [a,1], Kaspar Schüpbach [b]
[a] SBB AG, Eigerstrasse 13, 300 Bern 65, Switzerland
[1] E-mail: gabrio.caimi@sbb.ch, Phone: +41 (0) 79 124 77 29
[b] ELCA Informatik AG, Switzerland

Abstract
As part of the smartrail 4.0 program, SBB is focusing with the project TMS (Traffic Management System) on algorithmic supported, optimized and integrated capacity planning. For solving this problem, we have experimented with different approaches from the literature and have compared their quality and performance for our specific instances. In this contribution, we present the results of this comparison and discuss how we want to use this for having the best possible solution for our ambitious goal.

Keywords
Timetabling, MILP, Constraint Programming, Alternative Graph, Machine Learning

1 Introduction

Swiss Federal Railways (Schweizerische Bundesbahnen, short SBB) is pushing ahead digitization and automation of railway planning and operations. Customers are to benefit from higher capacities, less disturbances, better radio communication, improved customer information and lower overall costs. Railway infrastructure utilization is to be increased by shorter headway times and more precise planning. For this purpose, SBB has launched the *smartrail 4.0* program along the following principles.

- Algorithmically supported, optimized and integrated capacity planning
- Advanced control systems for railway operations
- New generation of digital interlocking systems
- Significant reduction in quantity and diversity of signaling systems
- Network-wide roll-out of the ETCS cab signalling
- Increased data transmission capacity
- Highly available and precise tracking of trains

The smartrail 4.0 program is organized in 4 principal streams: ETCS Interlocking (EI), Localization, Connectivity and Security (LCS), Automatic Train Operation (ATO) and Traffic Management System (TMS). In the context of smartrail 4.0, the TMS-stream strives to reach the following goals:

- Integrated and automated planning
- Automation of the operations centers. The employee develops himself from user to manager of the system.
- Enabling of efficient, real-time and precise automation and control of train movements and speeds.
- Precise coordinated remote control of departure, driving and arrival of trains.

The opportunities for the railway system behind these goals are pointed out in Weidmann et al. (2014).

The program clearly aims for an evolutionary approach towards automation of the planning and operation systems, in order to realize improvements step by step. During transition, it is crucial to take the human factor and the interaction between manual and automated processes into account. Certain roles will have to prepare the inputs and have to understand and post-process computational results. Particularly in the long-term planning process with many commercial and political aspects, the human factor will remain still important for long time, even if also at this stage algorithmic decision support will be important.

The key factor for the success of TMS is to find the right approach which enables on the one hand to have a strong algorithmic performance to solve big instances of the size of Switzerland, and on the other hand to enable a continuous integration in the current processes, with particular focus on the man-machine interaction.

2 Problem Formulation

The timetabling rules we consider were set by experienced railway planners. Planners also provided the test scenarios on which the different formulations are tested (see 5.2).

The Timetabling Problem considered in this paper is as follows: Given a list of trains to be scheduled, and for each train

- a list of commercial requirements, such as earliest departure times, latest arrival times, minimum dwell times and connections to other trains.
- a directed acyclic *route graph* that defines the routes the train could take from origin to destination. Each arc in the graph is called a *route section* and has associated to it the minimum running time for this train on this edge and a list of resources that are occupied while the train is on this section. A section may also have a non-negative *penalty* attached that is counted in the objective function if this section is used in the solution

Choose for each train exactly one path from origin to destination in the route graph and assign to each node on this path a time such that

- all commercial requirements are satisfied
- all minimum running times on the route sections are satisfied
- no resource occupation conflict results
- the objective function (see below) is minimized

The only commercial requirement that may be violated are latest arrival times. A violation of a latest arrival is a *delay*. All other rules are hard constraints. The *objective function* is the sum of all delays plus the sum of all route penalties for the chosen routes.

In this paper, we only consider *blocking* resources. To avoid *resource occupation conflicts*, it must be allocated to one train movement exclusively. The next train may only allocate it once the previous train has released it and a given release time has elapsed.

3 Approaches

3.1 MILP Model

One standard approach is to formulate this timetabling problem as a Mixed Integer Linear Programming (MILP) and to solve it by general-purpose MILP solvers. These can in principle solve instances to optimality, if problem size and computation time limits permit. For this study, we have used IBM Cplex Version 12.8.

In our model, each train run is modelled as a series of events with trip sections in-between. Attached to the events are commercial requirements such as earliest/latest departure/arrival times and connections. The sections contain information on required process durations and resource occupations.

Continuous decision variable are introduced for the event times. Binary variables define the routing alternatives taken and the precedence order of train pairs on sections with common resource occupations.

Our model is similar to the ones described in Pellegrini et al. (2012) and Fischetti and Monaci (2015). Dependencies between event times are modelled as simple time difference constraints: Trip times, stop times, connection times and release times. Big-M constraints are used to switch off all constraints which become oblivious depending on binary routing and precedence decisions.

Different to Pellegrini et al. (2012) where each origin-destination route gets it own decision variable, we introduce one variable for each local routing alternative. Continuity of the local decisions is enforced by flow conservation constraints. Other differences are that we minimize the weighted sum of delays and that we currently omit constraints on rolling stock circulation.

As in Fischetti and Monaci (2015), we limit the maximum delay allowed for each event as hard constraint and only introduce precedence decision variables and constraints for pairs of trains, which are temporally close enough to be potentially conflicting.

Further we merge precedence decisions for resource occupations of neighboring sections, whenever change of precedence ordering is not possible.

A comparison of the solver performance with and without these enhancements is available in Schupbach et al. (2018).

3.2 CP Optimizer

IBM's CP Optimizer[1] (CPO for short) is a general-purpose constraint solver with a strong focus on scheduling problems. As part of this focus it provides *interval* variables out of the box as central model element. An interval variable represents an activity that is to be

[1] https://www.ibm.com/analytics/cplex-cp-optimizer

planned. An expressive and intuitive constraint language is available to model conditions on the activities. See Laborie et al. (2018) for a general overview of CP Optimizer.

The expressiveness and flexibility of the language makes it comparatively easy to incorporate new planning rules into the model. In our experience this makes CPO especially suitable for prototyping new ideas before implementing them also in other solvers.

Also, CPO naturally allows a resource view on the problem, rather than a train-pair view as in 3.1 and 3.3. Instead of demanding that *for each train pair* occupying a common resource, one of the train has to precede the other, one constrains that *for each resource*, the intervals *of all trains* must not overlap. This leads to models that grow only linearly in the number of trains and thus remain fairly compact.

The CPO Model

For each route section of each service intention introduce an interval variable. It represents the activity of traveling through this route section for this service intention.

Time windows (such as earliest departure, latest arrival, etc.) are handled by setting the allowed domain of the associated interval variables accordingly. Similarly, minimum running times are guaranteed by setting a minimum length for the interval. For connections, introduce timing constraints such as startBeforeEnd between the appropriate intervals.

Resource occupation conflicts are avoided with appropriate no-overlap constraints. Routing alternatives are handled using the presence status of the intervals. Just as in the flow formulation for the MILP, one has to make sure that the solver chooses exactly one path through the routing graph for the solution.

3.3 ASP

Answer Set Programming (ASP) is a declarative problem solving approach. For our application we used Potassco, the library developed at Potsdam University. See Gebser et al. (2012) for a description of the language and tools. We transformed the problem described in section 2 into a set of declarations. Then, the Potassco grounder aggregated the declarations and defined the decision variables for the optimizer. These decision variables correspond to the routes and time events that are not naturally involved by the declarations. Finally, the optimizer selected the route and the time variables that return the best objective.

Similarly to the other formulations, for each route section of each service intention a binary decision must be taken. The selection of the previous section is a precondition for each route choice. Similarly to the MILP formulation (Section 3.1) time variables represent the entrance of a train into a section and the exit from the last sections.

The time variables are linked together via difference logic constraints see for example Kaminski et al. (2017), which are linear inequalities that are activated when the declared preconditions are satisfied (cfr. Big-M in Section 3.1). For instance, minimum travelling time constraints are activated if the corresponding route sections are selected. Analogously, minimum time separations are activated if different trains are routed through common resources.

3.4 Alternative Graph Library

The Alternative Graph Library (AGLib) is an academic optimization framework specialized for scheduling and routing trains in real-time. [2]

As described in Samà et al. (2015), the solver splits scheduling and routing into separate steps. Starting from an initial set of routes, the first schedule is computed applying a branch-and-bound scheme on the so-called alternative graph. Then, in the routing step, neighborhood search algorithms look for improving routings. Applying scheduling and routing steps alternatingly, AGLib iteratively improves solution quality.

The objective function built into AGLib is the minimization of the maximum delay, measured over all events. When comparing to the cost function described in the challenge formulation, this can lead to different optimal solutions and needs to be kept in mind when reading the following results section 5.

4 SBB Crowd Sourcing Challenge on Train Schedule Optimization

From August through November 2018 SBB conducted a crowd-sourcing Challenge[3] to solicit algorithms for solving the timetabling problem described in section 2.

By design, the challenge was limited to a fixed set of nine problem instances. Due to technical limitations, it was not possible to evaluate the algorithms on an independent test set. This setup probably explains why greedy approaches were so successful, as they could be fine-tuned on these particular instances. Nonetheless, the winning algorithms appear reasonably generic, using at most a handful of parameters.

Among the leading participants a wide variety of approaches and technologies were tried. A common theme among the very best submissions was the use of greedy algorithms, although MILP and Constraint Programming approaches as well as enhancing greedy scheduling with reinforcement learning techniques (Q-Learning) were also quite successfully applied.

Some of the leading participants have made their code publicly available[4]. We encourage an academic review and possible extension of these ideas. In particular, we point out an interesting greedy approach (4.1) and an approach incorporating reinforcement learning ideas (4.2).

4.1 Greedy Scheduling using Resource Usage Density

The winning algorithm[5] of the Challenge greedily solves the most *critical* conflicts first. A *conflict* is defined as any two train sections that potentially occupy a common resource simultaneously. This also takes into account routing alternatives (so there may be many conflicts on a given resource between two trains). *Criticality* of a conflict is determined by a function that takes into account usage density of the associated resource and the planning flexibility of the associated trains (if the trains have a large time window, the conflict is not

[2] Many Thanks to Andrea D'Ariano and Marcella Samá from the Roma Tre University for sharing AGLib with SBB for evaluation purposes.

[3] https://www.crowdai.org/challenges/train-schedule-optimisation-challenge

[4] Links to the code repositories are embedded in the following discussion thread on the challenge site: https://www.crowdai.org/topics/what-tools-did-you-use/discussion

[5] https://github.com/iquadrat/sbb-train-scheduler

critical).

For each conflict, at most four resolutions are possible: Train 1 before Train 2, vice versa, Train 1 takes a different path, Train 2 takes a different path. These resolutions are again weighted and the preferred resolution is greedily applied. If a resolution leads to infeasibilty, the next one is tried. If all lead to infeasibilities, backtrack to previous conflict.

This greedy approach is similar in spirit to some Travel Advance Heuristics (TAH), such as the *critical first*-TAH in Khadilkar (2017), but it takes into account routing alternatives and the inherent flexibility of service intentions.

The algorithm is a satisfiability solver, not an optimizer. It stops after the first feasible solution has been found. The maximum allowed train-wise and total delays can be specified via parameters.

4.2 Reinforcement Learning for Railway Scheduling

To the best of our knowledge, applications of Machine Learning methods to railway scheduling are still very limited. Only quite recently have some encouraging results been published, for example by Khadilkar (2018).

In the challenge, one solver[6] used a discrete event simulator augmented with reinforcement learning techniques. The simulator tries to schedule trains one section at a time on a first-come-first-served basis. In each step, the simulator looks at the train and where on its route it is currently located. It then checks whether the train can proceed. If not (for example if it has to wait for a connection, or all following sections are occupied), it waits a fixed amount of seconds. If it *can* proceed, the next route section is chosen ϵ-greedily among all possible ones. That is, with probability (1-ϵ) the route with highest *desirability* is chosen and with probability ϵ a random choice is made. How *desirable* the route sections are is encoded in a table. These values are initialized and then updated (learned) during the simulation episodes. Choices that lead to delays are penalized. An episode finishes when all trains have reached their destination. When an episode finishes, a new one is immediately started. This repeats until the time limit is reached.

While the specific algorithm used is very rudimentary and the effectiveness of the learning questionable, it nevertheless offers a glimpse into methods that appear to hold great potential - as evidenced by the success of reinforcement learning in other fields, such as Google's AlphaZero.

5 Tests and Results

5.1 Test Setup

We evaluate the different solvers on the problem instances of the Crowd Sourcing Challenge[7]. These instances are derived from the actual timetable on the triangle of lines Zurich - Thalwil, Thalwil - Chur and Thalwil - Lucerne. They differ in the time window they cover and the number of routing alternatives available to the trains. The complexity of the instances generally increases with their number. Instance 01 is too trivial to be interesting and is excluded. Instance 02 contains 58 trains with virtually no routing alternatives for a

[6] https://github.com/deuxnids/sbb_challenge

[7] https://github.com/crowdAI/train-schedule-optimisation-challenge-starter-kit/tree/master/problem_instances

total of roughly 4300 route sections. Instance 09 contains 287 trains and with all routing alternatives a total of over 34000 route sections.

Most instances are solvable with zero delay. In this sense, they represent a 'normal' planning scenario where trains are scheduled without disturbances and with enough capacity on the network available. Instance 05 is special in that it simulates the effect of the closure of one track on a double track line and it is not solvable without significant delay; its optimal objective value is 32.98.

The solvers are run on pods in a cloud platform configured to have a CPU limit of 8 cores and 40GB of RAM, with the exception of the crowdsourced solver IQUADRAT, which is run on an office notebook with Intel i5-6300U CPU @ 2.4GHz and 12 GB of RAM. To simulate 8 usable cores, IQUADRAT, as a single-threaded solver, is run with the 8 different configurations listed in 1.

The solver MILP-ND (as in 'no-delay') uses the MILP model as described in 3.1, but assumes that no delay is allowed for any train. This drastically reduces the model size and search space compared to the MILP solver, where a maximum delay of 30 minutes for each train is assumed.

Timeout for all solvers and instances is 900 seconds.

Table 1: Settings used for the IQUADRAT solver

max_penalty	max_penalty_per_intention	connetion_badness_factor	provides best solution for instance
0	0	5	01, 02, 03, 06, 07, 08, 09
5	5	5	04
10	5	5	-
20	5	5	-
20	10	5	-
40	10	5	-
40	15	5	05
60	15	5	-

5.2 Results and Discussion

Table 2 summarizes the results. We observe the following patterns.

The MILP solver shows decent results for smaller and medium-sized instances. In particular, it can prove optimality for instances 02-04. With bigger instances with more routing alternatives, computation times quickly explode. The no-delay feasiblity checker MILP-ND can solve much larger instances, also 06-08, provided they are feasible.

Instance 05 with the closed track, which is not solvable with zero delay, is best solved by IQUADRAT and AGLib. AGLib impresses by mostly keeping up with the other solvers while only using one CPU core. Instance 05, in particular, highlights how AGLib is very efficient in improving the initial train routes in the right place and at the right time, namely where and when it is most critical. AGLib has problems with instance 09 since the initial route given to each train in the input file is not feasible in terms of computing a conflict-

free train schedule. A future version of AGLib is expected to be able to recover from such infeasibilities.

In the set of instances used for this comparison, the algorithm by IQUADRAT works exceptionally well as a feasibility checker for a provided bound. For the medium and larger instances that are solvable with zero delay, it finds solutions much quicker than all other approaches. Further study with a larger variety of instances should be performed to assess the validity of the approach in general.

CPO is an all-rounder in the wide range of scenarios 02-08, but is outperformed by the specialists for with/without delay instances. ASP shows decent results for smaller and medium-sized instances but, same as bare MILP, does not inherenty scale. Model building, grounding and solving would have to be wrapped into an iterative framework that solves complicated instances step by step.

Table 2: Solve durations for the problem instances. Numbers represent average solve duration in seconds. Numbers in parentheses represent the objective value of the best solution found. Note that CPO and AGLib use different objective functions, cf. 3.2 and 3.4. 'infeas.' means the problem is infeasible. 'no sol' means the solver provided no feasible solution within the time limit.

Instance	MILP	MILP-ND	CPO	ASP	AGLib	IQUADRAT
02	8.8 (0)	1.4 (0)	7.6 (0.6)	23 (0)	4.3 (0.2)	4.0 (0)
03	49 (0)	3.0 (0)	15 (1.4)	52 (0)	8.9 (0.5)	8 (0)
04	164 (0.08)	8.8 (1)	20 (15.8)	80 (1)	18 (7.5)	16 (4.7)
05	900 (57.83)	infeas.	900 (84.4)	900 (44.85)	321 (124.6)	391 (39.95)
06	no sol.	219 (0)	211 (161.2)	no sol.	1409 (105.9)	75 (0)
07	no sol.	613 (0)	276 (218)	no sol.	699 (133.2)	128 (0)
08	no sol.	266 (0)	100 (99.6)	no sol.	338 (141.3)	44 (0)
09	no sol.	no sol.	no sol.	no sol.	no sol.	277 (0)

6 Conclusions and Outlook

In this paper, we have compared several different solution methods for the specific train scheduling problem (routing and timetabling) that we face at SBB. We can conclude that solving routing and scheduling in one MILP-instance seems to be too much for larger instances. A MILP model remains in any case our benchmark for solving smaller instances

exactly. None of the approaches alone will work for the system we need to build, in terms of size but also in terms of business requirements (the 9 instances analyzed in this paper are still small compared to what we have to solve in reality). Therefore, we are of the opinion that decomposition methods for solving the problem are unavoidable.

This comparison also gave us some input to continue our work towards increasing the tractable scenario sizes. We plan to pursue the following research directsion:

- Implement iterative MILP solvers using row and column generation, thereby reducing model size.
- The strategy of first solving a no-delay feasibility problem with reduced search space is promising for each of the solvers.
- Further investigation of routing and scheduling heuristics of AGlib and evaluate possibilities for parallelization.

Furthermore, we see much promise in combining different solvers by sharing solutions across them. For example, we plan to use initial solutions from quick heuristic solvers, such as AGLib, ans using them as starting solutions for the MILP-solvers, or even inject them during a solve run.

References

M. Fischetti and M. Monaci. Using a general-purpose MILP solver for the practical solution of real-time train rescheduling. Technical report, 2015.

M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

R. Kaminski, T. Schaub, and P. Wanko. A tutorial on hybrid answer set solving with clingo. In G. Ianni, D. Lembo, L. Bertossi, B. Faber, W.and Glimm, G. Gottlob, and S. Staab, editors, *13th International Summer School of the Reasoning Web*, volume 10370 of *Lecture Notes in Computer Science*, page 167–203. Springer-Verlag, 2017.

H. Khadilkar. Scheduling of vehicle movement in resource-constrained transportation networks using a capacity-aware heuristic. In *Proceedings of the American Control Conference*, 2017.

H. Khadilkar. A Scalable Reinforcement Learning Algorithm for Scheduling Railway Lines. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–11, 2018.

P. Laborie, J. Rogerie, P. Shaw, and P. Vilím. IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG. *Constraints*, 2018.

P. Pellegrini, G. Marlière, and J. Rodriguez. Real time railway traffic management modeling track-circuits. In *ATOMOS 2012, 12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, page 12p, France, Sept. 2012.

M. Samà, A. D'Ariano, D. Pacciarelli, and F. Corman. A variable neighborhood search for fast train scheduling and routing during disturbed railway traffic situations. *Computers & Operations Research*, 01 2015.

K. Schupbach, G. Caimi, and J. Jordi. Towards Automated Capacity Planning in Railways. In *IRSA 2017: Proceedings: 1st International Railway Symposium Aachen, Germany, 28-30 November 2017*, pages 310–325, Aachen, Nov 2018. 1st International Railway

Symposium Aachen, Aachen (Germany), Publication Server of the University Library, RWTH Aachen University.

U. Weidmann, M. Laumanns, M. Montigel, and X. Rao. Dynamische kapazitätsoptimierung durch automatisierung des bahnbetriebs. *ETR Eisenbahntechnsiche Revue*, 12, 12 2014.