20th EURO Working Group on Transportation Meeting, EWGT 2017, 4-6 September 2017, Budapest, Hungary

A MILP-based heuristic for a commercial train timetabling problem

Sara Gestrelius\(^a\)*, Martin Aronsson\(^a\), Anders Peterson\(^b\)

\(^a\)SICS RISE AB, Box 1263, SE-164 29 Kista, Sweden
\(^b\)Linköping University, Department of Science and Technology, SE-601 74 Norrköping, Sweden

Abstract

Using mathematical methods to support the yearly timetable planning process has many advantages. Unfortunately, the train timetabling problem for large geographical areas and many trains is intractable for optimization models alone. In this paper, we therefore present a MILP-based heuristic that has been designed to generate good-enough timetables for large geographical areas and many trains. In the incremental fix and release heuristic (IFRH), trains are added to the timetable in batches. For each batch of trains, a reduced timetable problem is solved using a mathematical integer program and CPLEX. Based on the solution, the binary variables defining meeting locations and stops are fixed, and the next batch of trains is added to the timetable. If previously fixed variables make the problem infeasible, a recovery algorithm iteratively releases fixed variables to regain feasibility. The paper also introduces a simple improvement heuristic (IH) that uses the same idea of working with batches of trains. The heuristics are tested on a real case-study from Sweden consisting of both small problem instances (approximately 300 trains and 1400 possible interactions) and large problem instances (approximately 600 trains and 5500 possible interactions). IFRH returns a feasible timetable within 30 minutes for all problem instances, and after running IH the optimality gaps are less than 5%. Meanwhile, if CPLEX is used without the heuristic framework to solve the total optimization problem, a feasible timetable is not returned within 2 hours for the large problem instances.

© 2017 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the scientific committee of the 20th EURO Working Group on Transportation Meeting.

Keywords: Train timetabling; MILP; optimization; heuristic;

---

* Corresponding author. Tel.: +46-76-1061452.
E-mail address: sara.gestrelius@ri.se
1. Introduction

The steady increase in traffic on the Swedish railways, and the competition brought about by the market deregulation, intensify the need for efficient and fair timetable construction. In Sweden, the current practice is that approximately 20 timetable constructors make the yearly timetable by hand in a planning tool called TrainPlan. This is labour intensive and tedious work, and it takes the constructors 2 months to make a draft timetable. Further, due to the high complexity of the timetabling problem, it is hard to guarantee fairness and transparency in the planning process. The capacity allocation process could be made more efficient and effective by using decision support tools with automatic timetable generation. In addition, if the timetable construction was faster, the planning process could be made more iterative. An iterative process, previously discussed in e.g. Forsgren et al. (2013) and Gestrelius et al. (2015), would enable the train operators, the maintenance contractors and the infrastructure manager to test different set-ups and better negotiate their needs. All in all, using optimizing algorithms in the yearly timetable planning process would benefit all involved parties and thus improve infrastructure use.

The timetabling problem consists of finding a conflict-free timetable for a set of trains and a given geography, and it is the core of the problem solved by the timetable constructors when making the yearly timetable. The commercial timetable problem also entails that the train paths must fulfill the operators’ requirements. The clear advantage of using optimization to generate timetables has attracted much research effort on mathematical optimization models for the train timetabling problem (see e.g. the tutorial in Harrod (2012)). Unfortunately, the execution times of the proposed optimization methods become unmanageably long when trying to solve the timetabling problem for large geographical areas and many trains, which is required to make a yearly timetable.

In this paper, we therefore propose using MILP optimization in a constructive heuristic framework for commercial train timetabling. We also introduce a simple improvement heuristic. The objective of the paper is to investigate if the proposed heuristics are able to generate good-enough timetables for a large geographical area and many trains, while also providing acceptable execution times.

1.1. Related work

Short execution times are important during real time rescheduling of trains. Therefore, the majority of work on heuristics for train timetabling problems can be found in research focused on the dispatching problem, also called the train timetable rescheduling problem. Fang et al. (2015) and Cacchiani et al. (2014) provide overviews of methods, including heuristics, for rescheduling during disturbances and disruptions. Further, Fang et al. (2015) state that hybrid approaches is a promising direction for future research.

Fixing variables to reduce the search space and thereby speed up execution is a well-known strategy (see e.g. the survey by Fang et al. (2015)). Boccia et al. (2013) present two strategies for fixing binary variables in order to speed up the solution of their mixed integer linear program (MILP) for dispatching. In the first strategy, the routing of trains is fixed, and in the second, they solve a sequence of MILPs corresponding to growing sets of trains, and fix routing and meeting variables for previously added trains. Their second approach is very similar to our incremental fix and release heuristic which incrementally adds trains and fixes their meeting and stop variables. The main difference is that our method incorporates a recovery algorithm for when the problem becomes infeasible, most notably when no feasible solution can be found that fulfills the commercial constraints representing the operators’ requests. Schachtebeck and Schöbel (2008), Acuna-Agost et al. (2011) and Törnquist and Persson (2007) also test various strategies for fixing integer variables to speed up execution times. Törnquist and Persson (2007) solve the dispatching problem for a Swedish case study, thus handling a heterogeneous traffic similar to our case study. In a later paper, Törnquist Krasemann (2012) states that the optimization-based approaches presented in Törnquist and Persson (2007) are not feasible for some dispatching scenarios, and therefore a complementing constructive heuristic is developed. The heuristic is based on constructing a solution by choosing the best next feasible event, and back-tracking in case of deadlocks.

The method presented in this paper consists of a heuristic framework for the MILP previously presented in Gestrelius et al. (2015). The focus of this paper will be the heuristic framework, and only brief introductions to the commercial timetabling problem and the MILP model will be provided. For more thorough descriptions the reader is referred to Gestrelius et al. (2015).
2. Mathematical models

The incremental fix and release heuristic (section 3.1) attempts to generate a timetable by adding batches of trains to the timetable. Once a batch of trains is added a MILP (section 2.2) is solved to find train paths for the added trains. The binary variables defining train interactions and train stops are then fixed before the next batch of trains is added. If the fixed variables render the problem infeasible a recovery algorithm is run to regain feasibility (section 3.1.1). Once a complete feasible timetable has been generated, an improvement heuristic is executed to improve the timetable quality (section 3.2).

2.1. Brief introduction to the commercial train timetabling problem

A standard train timetabling problem is to find a conflict-free timetable for all trains \( r \in R \) running on a geography \( G \) consisting of links \( l \in L \) and stations \( s \in S \). Stations consist of a set of parallel tracks and no more than \( n_s \) trains may be present at a station \( s \) at any point in time. Links can be either single track links or multi-track links.

There are two types of train interactions: crossings and overtakings. A crossing is when two trains moving in opposite directions meet, while an overtaking is when two trains moving in the same direction change order. In a feasible timetable, all the train paths respect the technical limitations of the trains and the infrastructure, and all train interactions take place at feasible interaction locations and follow safety regulations. Further, in our version of the problem, called the commercial train timetabling problem, all trains \( r \in R \) have an application train path and a time domain, \( \pm m_r \) minutes, within which the operator requires the final train path to be. That is, the train paths have to respect timing constraints, interaction constraints, safety regulation constraints and commercial constraints.

There are various objective functions that can be used, e.g. timetable robustness, customer satisfaction or expected passenger travel time. In this paper the objective is to minimize the sum of run times of all trains. This objective function is chosen as operators often want short run times, and the objective also promotes efficient capacity utilization as trains spend as little time as possible on the tracks. For a further discussion on the problem definition and constraints we refer to Gestrelius et al. (2015).

2.2. The train timetabling MILP

The journey of a train \( r \) through the geography is defined by continuous time variables, \( t_{rg}, g \in L \cup S \), and binary stop variables, \( y_{rs}, s \in S \). A time variable represents the time when the train enters a link or a station. Time variables are constrained by the commercial requirements, and also by the technical limitations of the train and the infrastructure, and by the stop variables. Trains can only stop at certain stations, and if a train stops at a station the minimum running time on the preceding and succeeding links must be increased due to deceleration and acceleration. Further, crossings at single track section stations, and overtakings in general, require that the first train that arrives to the interaction geography stops. The interaction of two trains is modelled using binary interaction variables, \( y_{gr}^{ij} \), for all geographical points \( g \) where trains \( r \) and \( j \) may cross or overtake each other. The interaction variable takes value 1 if the trains cross or overtake each other at geography \( g \), and 0 otherwise. Trains moving in opposite directions only overlap once, namely when they cross, and the binary interaction variables therefore capture the train interaction completely. Trains moving in the same direction can overlap and overtake each other many times, and additional binary variables are required to model train orders on links and stations, and overlaps at stations.

Binary variables often prolong the execution times of optimization algorithms. The commercial constraints forcing the train paths to be within a pre-defined time domain reduce the number of interaction possibilities for trains, and thereby also the number of binary variables in the problem. However, the number of binary variables in the MILP still grows with the number of trains and geographical points, and the execution times become unmanageably long when trying to solve the commercial timetabling problem for large geographical areas and many trains. If all binary variables are fixed, the timetabling problem is reduced to a linear program that can be efficiently solved. Therefore, in the proposed heuristics only a few binary variables are unrestricted in each iteration, resulting in optimization problems that are readily tractable by CPLEX. Note that fixing binary variables greatly reduces the search space, and the optimal solution to the reduced problem may not be the optimal solution to the total,
unreduced, problem. Also, the fixing of binary variables may render the problem infeasible, which is why a recovery algorithm releasing binary variables is required.

3. Heuristics

3.1. Incremental fix and release heuristic

The incremental fix and release heuristic attempts to generate a solution to the total problem \( P \) consisting of trains \( R \) by adding a sequence of train batches \( (B_n)_{n\in\mathbb{N}} \) to the timetable. Note that the train batches are disjunctive, and that their union is \( R \). Let the subscript \( n \) denote the order of addition. That is, let \( B_1 \) be the first set of trains to be added, \( B_2 \) the second one, and so on. In each iteration of the algorithm, a new batch of trains is added to a timetable comprising all trains from previously added batches. The previously added trains are partially fixed; they are not fixed in time, but rather their stops and interactions with other fixed trains have been decided. Let \( F \) be the set of fixed trains, then the variables that have been fixed are \( y_{rs}, s\in S, r\in F \) and \( y^g_r, g\in G, r\in F \). Note that for the first iteration there are no fixed trains, \( F = \emptyset \). The first step is therefore simply to solve the timetable problem for a large geographical area but with only a few trains, namely the trains in \( B_1 \).

Let \( B_i \) be the batch of new trains to be added. The timetable problem to be solved comprises trains \( r \in B_i \cup F \) and we call it the reduced timetable problem, \( RP(B_i \cup F) \). If a solution is found to \( RP(B_i \cup F) \), all stop and interaction variables for trains \( r \in B_i \) are fixed based on their value in the solution, and the trains in \( B_i \) are added to \( F \). The algorithm then moves on to adding the trains in batch \( B_{i+1} \). The procedure of adding and fixing trains is continued until a feasible timetable for all trains is returned, or until a reduced problem \( RP(B_a \cup F) \) is infeasible. If a reduced problem is infeasible the recovery algorithm described below is executed. If the recovery algorithm returns a feasible timetable the stop and interaction binaries are re-fixed according to the new feasible timetable, and the iterative addition of train batches is continued by adding train batch \( B_{a+1} \). If the recovery algorithm does not return a feasible solution, the total timetable problem is infeasible, and the algorithm stops.

The full incremental fix and release heuristic is shown in Algorithm 1.

3.1.1. Recovery algorithm

The aim of the recovery algorithm is to return a feasible timetable for the trains \( B_a \cup F \). The input is an infeasible reduced timetable problem \( RP(B_a \cup F) \). The base assumption is that if no binary variables were fixed in \( RP(B_a \cup F) \), then it would be feasible (otherwise the original timetable problem is infeasible). The first step of the algorithm is therefore to release the binaries of all trains that have an interaction with a train \( r \in B_a \). Let \( K(r) \) denote the set of trains that interact with train \( r \), and let \( I \) denote the set of trains in \( F \) that interact with a train in \( B_a \), i.e. \( I = \{r: r \in K(i) \cap F, i \in B_a \} \). The binary variables of all trains in \( I \) are then released, \( = F \setminus I \), and an attempt is made to solve the new problem, \( RP(B_a \cup I \cup F) \). We call the new optimization problem the 1-released problem. If a feasible solution is obtained for the 1-released problem, the timetable is returned to the main algorithm. On the other hand, if the 1-released problem is also infeasible, all trains in \( F \) interacting with a train in \( I \) are released. All newly released trains are added to \( I \), and an attempt is made to solve the new problem. We call this the 2-released problem. The procedure of releasing interacting trains and solving the new problem is repeated until a feasible n-released problem (and solution) is found and a feasible timetable can be returned to the main algorithm, or until all trains have been released and the entire, non-reduced, timetable problem is proven infeasible. Note that as more and more trains are released, the problem approaches the original timetable problem, albeit possibly with fewer trains. The execution time of the recovery algorithm may therefore become unmanageably long.

3.2. Improvement heuristic (IH)

Assume that a feasible timetable, \( E \), has been generated by IFRH. If the timetable is suboptimal, which is generally the case, the improvement heuristic (IH) can be used to try to improve the solution quality. IH starts by fixing all stop and interaction binaries to their value in \( E \). The heuristic then releases the fixed variables of a set of trains. The new timetable problem can either be solved to optimality or an execution time limit may be used. Once

\[ \text{Algorithm 1: Incremental fix and release heuristic} \]

\[
\begin{align*}
1. & \text{Input: } P, B_1, \ldots, B_n, F \\
2. & \text{Output: } E \\
3. & \text{Initialization: } B = B_1, F = \emptyset, \text{solve } RP(B \cup F) \\
4. & \text{while } B_a \cup F \text{ is infeasible} \\
5. & \quad \text{do } K(r) = \{s \mid s \text{ interacts with } r \text{ and } s \in B_a \}, I = \{r \mid r \text{ interacts with a train in } B_a \} \\
6. & \quad \text{release binary variables of all trains in } I \\
7. & \quad \text{solve } RP(B_a \cup I \cup F) \\
8. & \text{if } \text{solution found} \\
9. & \quad \text{State } E = \text{return solution} \\
10. & \text{else } \text{return infeasible} \\
11. & \text{end while} \\
12. & \text{return infeasible} \\
\end{align*}
\]

\[ \text{Algorithm 2: Recovery algorithm} \]

\[
\begin{align*}
1. & \text{Input: } P, B_a, F \\
2. & \text{Output: } E \\
3. & \text{Initialization: } K(r) = \{s \mid s \text{ interacts with } r \text{ and } s \in B_a \}, I = \{r \mid r \text{ interacts with a train in } B_a \} \\
4. & \text{while } B_a \cup F \text{ is infeasible} \\
5. & \quad \text{do } K(r) = \{s \mid s \text{ interacts with } r \text{ and } s \in B_a \}, I = \{r \mid r \text{ interacts with a train in } B_a \} \\
6. & \quad \text{release binary variables of all trains in } I \\
7. & \quad \text{solve } RP(B_a \cup I \cup F) \\
8. & \text{if } \text{solution found} \\
9. & \quad \text{State } E = \text{return solution} \\
10. & \text{else } \text{return infeasible} \\
11. & \text{end while} \\
12. & \text{return infeasible} \\
\end{align*}
\]
the solution is returned all released variables are fixed to their new values and another set of trains is released. The IH algorithm is outlined in Algorithm 2 below.

Algorithm 1: Incremental fix and release heuristics (IFRH)

**RP** is the reduced problem and **Y** is the solution to the reduced problem. An infeasible solution is denoted as an empty set, i.e. if \( Y = \emptyset \) then **Y** is infeasible. **F** = set of fixed trains. **I** = set of released trains. \( (B_n)_{n \in \mathbb{N}} \) = sequence of train batches to be added.

1) \( F = \emptyset. I = \emptyset. Y = \emptyset. \)
2) For all batches \( B_n \) in \( (B_n)_{n \in \mathbb{N}} \):
   a) Use CPLEX to solve \( RP(B_n \cup F) \) to get a new solution **Y**.
   b) If \( Y = \emptyset \): //Run recovery algorithm
      i) Find the trains to be released: \( I_a = \{r: r \in K(i) \cap F, i \in B_n\} \).
      ii) Release all trains in \( I_a: F = F \setminus I_a, I = I \cup I_a \).
      iii) Use CPLEX to solve \( RP(B_n \cup I \cup F) \) to get a new solution **Y**.
      iv) While \( Y = \emptyset \) and \( I_a \neq \emptyset \):
         A. Identify more trains to release: \( I_a = \{r: r \in K(i) \cap F, i \in I\} \).
         B. Release all trains in \( I_a: F = F \setminus I_a, I = I \cup I_a \).
         C. Use CPLEX to solve \( RP(B_n \cup I \cup F) \) to get a new solution **Y**.
         v) If **Y** = \( \emptyset \): the total problem is infeasible, stop algorithm.
   c) Fix all interaction and stop variables for trains in \( B_n \cup I \) to the value they have in solution **Y**. Set \( F = F \cup I \cup B_n, I = \emptyset. \)
3) Return solution **Y**.

Algorithm 2: Improvement heuristics (IH)

**P** is the total problem and **Y** is a solution to the total problem. **E** is the initial feasible solution. **R** = set of trains in \( P. A \) = set of trains that have not yet been released. \( c_r \) = cost of train \( r \) in current solution. \( n \) = number of trains to release in each iteration.

1) \( A = R. Y = E. \)
2) While \( A \neq \emptyset. \)
   a) Fix all variables to the values they have in **Y**.
   b) Sort all trains \( r \in A \) in decreasing order of \( c_r \). Let \( L \) denote the set of \( n \) trains with the highest cost.
   c) Release the binary variables of all trains in \( L: A = A \setminus L \).
   d) Use CPLEX to find a solution to the problem with the released trains, starting from the existing solution **Y**. Get a, possibly new, solution **Y**.
3) Return solution **Y**.

4. Experiments

Experimental runs were carried out in order to test the performance of the heuristics. The performance of IFRH was compared with solving the MILP by stand-alone optimization using CPLEX 12.2. The performance of IH was compared with stand-alone optimization in CPLEX 12.2, and also with the CPLEX improvement heuristic called Polishing.

4.1. Test case

IFRH and IH were tested on Hallsberg operational control area. Hallsberg operational control area is one of 8 operational control areas in Sweden, and one of the main lines pass through it. The problem has 158 stations and 199
links (when both tracks are counted as separate links for double tracks). A timetable was generated for each day of
the first week of February 2014. The first two days are Saturday and Sunday, and about 300 trains are operated each
day (small problems). The other days are week days, and about 600 trains are operated each day (large problems).
The small problems have approximately 1400 possible interactions each, and the large approximately 5500 possible
interactions. The train paths from the yearly timetable were used as applications, and a time range of ±15 minutes
was used as the commercial time domain limit. The objective function was to minimize the sum of run times of all
trains. Train data and geographical data were exported from the planning tool used by the timetable constructors,
TrainPlan. The generated timetables therefore have the same level of detail as the timetables constructed in the
yearly timetable process in Sweden.

4.2. Optimization set-up

CPELX 12.2 was run on a Linux workstation with 16 Intel(R) Xeon(R) CPU X5560 2.80 GHz to solve the
optimization problems. The default CPLEX settings were used but with repeat presolve being set to 3, thus allowing
cuts and new root cuts. For IFRH, the number of trains in each batch was set to ten. The batches were generated by
sorting the trains by their departure time, and then adding the first ten trains to the first batch, the next ten trains to
the second batch, and so on. A new batch of trains was added as soon as a feasible solution had been found for
the previous batch. That is, the relative MIP gap tolerance was set to 100% in CPLEX. For the stand-alone optimization,
a relative MIP gap tolerance of 100% was used and an execution time limit of 2 hours.

When it comes to IH, 5 trains were released in each iteration. A relative MIP gap tolerance of 0.1% and an
execution time limit of 3 minutes after an initial solution had been found were used in each iteration of IH. When
improving the timetable using stand-alone optimization or Polishing, a relative MIP gap tolerance of 0.1% and an
execution time limit of 10 minutes were used. All three methods were started from the timetable generated by IFRH.

4.3. Results

Table 1 shows how many seconds it took for each method to return a valid timetable for the different problem
instances. For the small problems of nearly 300 trains both methods returned a solution within about 1 minute. For
the large problems with nearly 600 trains, stand-alone optimization never returned a timetable within the 2 hour
execution time limit, while IFRH returned a valid timetable within 30 minutes for all instances. The number of times
the recovery algorithm was called is also shown in Table 1. The 1-released problem was always feasible in the
recovery runs, so further releasing of variables was never required.

<table>
<thead>
<tr>
<th>Instance data</th>
<th>IFRH</th>
<th>Stand-alone optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day</td>
<td>Trains (#)</td>
<td>Interactions (#)</td>
</tr>
<tr>
<td>1</td>
<td>297</td>
<td>1378</td>
</tr>
<tr>
<td>2</td>
<td>284</td>
<td>1544</td>
</tr>
<tr>
<td>3</td>
<td>560</td>
<td>5042</td>
</tr>
<tr>
<td>4</td>
<td>598</td>
<td>5525</td>
</tr>
<tr>
<td>5</td>
<td>607</td>
<td>5878</td>
</tr>
<tr>
<td>6</td>
<td>611</td>
<td>5967</td>
</tr>
<tr>
<td>7</td>
<td>609</td>
<td>5919</td>
</tr>
</tbody>
</table>
Figures 1(a) to 1(g) show how IH (black), stand-alone optimization (dark grey) and CPLEX Polishing (light grey) improve the timetable solution returned by IFRH. The objective function value is shown on the y-axis and the execution time on the x-axis. Note that the y-axis does not start at 0. Each point represents a returned timetable. Stand-alone optimization generates few solutions during the first 10 minutes, and often gives the smallest timetable improvement. CPLEX Polishing and IH are generally comparable. CPLEX Polishing is better for the small problem instances Day 1 and Day 2 and for the large problem instances Day 3 and Day 4, while IH is better for the large problem instances Day 5 and Day 6. For small problem instances, IH is comparable with Polishing for the first 1.5 minutes, but then IH finishes as all trains have been re-optimized once, while Polishing continues for another 8.5 minutes and subsequently finds better timetables.

Fig. 1. Comparison of improvement heuristic (black), CPLEX stand-alone optimization (dark grey) and CPLEX Polishing (light grey).
5. Conclusions

This paper introduces two MILP-based heuristics for a commercial train timetabling problem, the \textit{incremental fix and release heuristic}, and an \textit{improvement heuristic}. The core idea behind the two heuristics is to add trains to the timetable in batches. The heuristics perform better than stand-alone optimization for large problem instances of approximately 600 trains and 5500 interactions. A feasible timetable is always returned within 30 minutes, and after running the improvement heuristic the optimality gaps are less than 5%. Stand-alone optimization does not return a feasible timetable within 2 hours for the large problem instances.

The results show that constructive heuristics are useful for solving commercial train timetabling problems for large geographical areas and many trains. This may be because the timetable problems defined by our data set, or maybe also by real world timetable data in general, have some inherent structure that mitigates the negative effects of local decisions.

Naturally, trains in early batches are likely to get better train paths. However, this is in line with the current practice, where a timetable for high priority trains is constructed before adding the lower priority trains. If priority batches are not to be used, designing good batches is an issue that requires more research. The aim is to find batches that limit the number of iterations and infeasible partial solutions, while at the same time ensuring fast execution times of each iteration. Likewise, further research on deciding which sets of trains to release in the improvement heuristic would be beneficial. Designing a more recursive IH where trains are released more than once would also be interesting.

A weakness of the IFRH is that it reduces to stand-alone optimization in case of an infeasible original problem. It would be useful to further develop the recovery algorithm, and also potentially letting the recovery algorithm cancel trains if it fails to quickly regain feasibility.

Acknowledgements

This work has been funded by Trafikverket (the Swedish Transport Administration), grant TRV 2014/5233.

References


Gestrelius, S., Aronsson, M., Bohlin, M., 2015. On the uniqueness of operation days and delivery commitment generation for train timetables. 6th International Conference on Railway Operations Modelling and Analysis (RailTokyo2015).


