



Late Breaking Results: Iterative Design Automation for Train Control with Hybrid Train Detection

Stefan Engels 
Chair for Design Automation
Technical University of Munich
Munich, Germany
stefan.engels@tum.de

Robert Wille 
Chair for Design Automation
Technical University of Munich
Munich, Germany
robert.wille@tum.de

Abstract—To increase the capacity of existing railway infrastructure, the *European Train Control System* (ETCS) allows the introduction of virtual subsections. As of today, the planning of such systems is mainly done by hand. Previous design automation methods suffer from long runtimes in certain instances. However, late breaking results show that these methods can highly benefit from an iterative approach. An initial implementation of the resulting method is available in open-source as part of the *Munich Train Control Toolkit* at <https://github.com/cda-tum/mtct>.

Index Terms—ETCS, iterative design automation, hybrid train detection, block signaling, virtual subsection

I. INTRODUCTION

Rail transportation plays a vital role in future and sustainable mobility. However, the existing infrastructure has only limited capacity. Building new tracks alone will not suffice to deal with the increasing demand. To increase capacity on the existing infrastructure, new train control systems have been developed, e.g., the *European Train Control System* (ETCS, [1]). However, designing those signaling systems on specific railway networks by hand is cumbersome. Hence, design automation for this emerging technology is of great interest (see, e.g., [2]–[7]).

A. Basic Principles of Train Control Systems

Most train control systems in the world rely on block signaling. For this, a railway network is divided into fixed sections, on each of which at most one train is allowed at the same time. To decide if a given section is free, the network is equipped with *Trackside Train Detection* (TTD). Hence, these blocks are also called *TTD sections*.

Train systems based on ETCS allow the train itself to report its exact position and integrity status (*Hybrid Train Detection*, [8]). Because of that, TTD hardware is no longer needed to safely report a block section as free. Hence, TTD sections can be separated into smaller *virtual subsections* (VSS) allowing for shorter train following times without the need for additional hardware. While this principle is defined within the scope of ETCS, it could also be applied to similar standards such as CTCS (China) or PTC (North America) [9].

Example 1. Consider the track depicted in Fig. 1, which consists of two TTD sections and two trains T1 and T2. Using a classical control system, T2 has to slow down (solid orange line), because it cannot enter TTD 2, which is occupied by T1. However, if the second block is separated into two VSS, T2

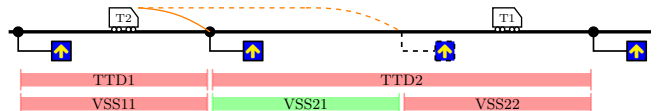


Fig. 1: Train control with hybrid train detection

can continue at the same speed (dashed orange line), because VSS 21 is already reported free, allowing for shorter headways.

B. Design Automation for Train Control Systems

To use the additional degree of freedom allowed by VSS, design automation methods are needed and different design tasks arise depending on the criterion of optimality [10]. Exemplary, we focus on the following task: Given an (infeasible) timetable and railway network, the goal is to place a minimal number of VSS to make the previously infeasible timetable realizable under new control systems using hybrid train detection.

Previous methods to solve such design tasks have been proposed in [2]–[7]. However, they do not incorporate the flexibility allowed by VSS or suffer from scalability issues. These late breaking results aim to shed light on what can be done to reduce the runtime.

II. ITERATIVE APPROACH

Without loss of generality, we base this paper on the most recent *Mixed Integer Linear Programming* (MILP) approach proposed in [7]. On every track segment, it continuously places an arbitrary number of VSS sections. Theoretically, the number of such sections is only upper bound since VSS sections have a predefined minimal length. However, an optimal solution (i.e., only using a minimal number of VSS sections) is likely to only place a small part of the VSS sections it could place in theory.

Because of this, it is promising to limit the number of VSS the solver is allowed to place, leading to a restricted model. While this might cut off the optimal (or even all) solutions, it is suspected to have a great influence on the runtime. This gives rise to an iterative approach. If the imposed limit is too tight, it is slowly increased until an (optimal) solution is found. Since the limit likely has to be updated only a few times, this approach is promising for runtime reduction.

A flow chart of the resulting iterative algorithm is presented in Fig. 2. In the following, we present only high-level aspects

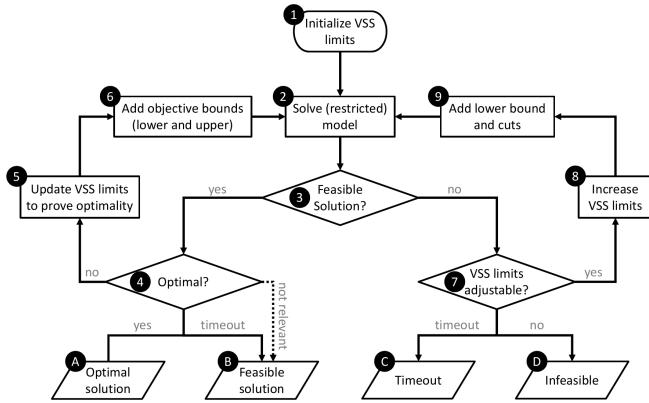


Fig. 2: Iterative approach (flowchart)

of how to update those limits. More details can be found in the implementation, which is part of the *Munich Train Control Toolkit* at <https://github.com/cda-tum/mtct>.

Assume that the restricted model is infeasible (7). Unless the above-mentioned theoretical limits are already reached (D), the restricted VSS limits have to be increased (8). Observe that any feasible solution not found within the restricted model must violate the imposed VSS limit on at least one track segment. From this, it is easy to deduce additional constraints that cut off all configurations that were already considered in the previous iteration (9). This prevents the solver from spending time on implications it has already concluded in previous iterations.

On the other hand, the above observation can be used to prove the optimality of a feasible solution (4, A). It might be necessary to increase some limits in an extra iteration in order to prove overall optimality (5). However, this is only to rule out improbable corner cases. Thus, it might be reasonable to skip this additional iteration (path indicated by a dotted line in Fig. 2) and return the current solution (B). In practice one can assume this solution to only use a minimal number of VSS sections, however a 100% certain proof is not provided.

III. EVALUATION

To evaluate the proposed method, we test it on the benchmarks provided by [7] (including real-world instances) using an AMD Threadripper 5955WX system (16 core 4GHz CPU, 128GB RAM) and the C++ API of Gurobi version 10.0.3 [11].

A. Experimental Analysis

We compare the iterative algorithm to the fully detailed method proposed in [7] with predefined routes. The respective runtimes (in seconds) are depicted in Tab. I.

If we skip the theoretical proof of optimality, we can clearly see a runtime improvement in all instances. On instances running more than 30 seconds, this was even by a factor between 8 and 34, i.e., approximately one order of magnitude. Even though the algorithm does not prove optimality with 100% certainty, the returned solution was in fact the optimal one (i.e., use a minimal number of VSS sections) on all tested instances. In particular, it is reasonable to skip the optimality

TABLE I: Experimental results (runtime in seconds)

	Algorithm: Optimality Proof:	Original	Iterative	
		yes	no	yes
Single Track	Without Station	47.4	1.4	1.4
	With Station	43.8	1.7	1.7
Highspeed Track	2 Trains	56.0	5.6	5.6
	5 Trains	730.5	87.4	87.4
Simple 2-Track Station		1.0	0.3	0.3
Simple Network		452.7	15.2	46.7
Overtake		2.1	2.0	2.0
Stammstrecke	4 Trains	1.9	1.1	2.1
	8 Trains	4.0	2.8	5.6
	16 Trains	9.1	5.1	13.8

proof, in which case the proposed method is a significant improvement on all tested instances.

On the other hand, we observe that proving the optimality of the obtained solution can take a substantial amount of the runtime for some instances. This is not unexpected, because the model obtained from step (5) can be rather large. However, on long-running instances (whose improvement is arguably more important) the runtime improvement remains approximately one order of magnitude. Moreover, we have argued that this step can be skipped in practice anyway.

B. Conclusions

With these late breaking results, we have seen that design automation methods for designing the block layout of train control systems (such as proposed in [7]) can highly benefit from an iterative approach that restricts the search space and slowly loosens those restrictions if no solution can be found. This approach has been implemented open-source as part of the *Munich Train Control Toolkit*.

Note that this method might benefit from tighter objective bounds (steps 6 and 9) using more information on the problem structure, which is a work in progress.

REFERENCES

- [1] L. Schnieder, *European Train Control System (ETCS)*. Springer Berlin Heidelberg, 2021.
- [2] S. Dillmann and R. Hähnle, “Automated planning of ETCS tracks.” *RSSRail*, 2019.
- [3] B. Luteberget, C. Johansen, and M. Steffen, “Synthesis of railway signaling layout from local capacity specifications.” *FM*, 2019.
- [4] V. Vignali, F. Cuppi, C. Lantieri, N. Dimola, T. Galasso, and L. Rapagnà, “A methodology for the design of sections block length on ETCS L2 railway networks,” *JRTPM*, 2020.
- [5] R. Wille, T. Peham, J. Przigoda, and N. Przigoda, “Towards automatic design and verification for Level 3 of the European Train Control System.” *DATE*, 2021.
- [6] T. Peham, J. Przigoda, N. Przigoda, and R. Wille, “Optimal railway routing using virtual subsections.” *RSSRail*, 2022.
- [7] S. Engels, T. Peham, and R. Wille, “A symbolic design method for ETCS Hybrid Level 3 at different degrees of accuracy.” *ATMOS*, 2023.
- [8] EEIG ERTMS Users Group, “ERTMS/ETCS hybrid train detection,” *Tech. Rep. 16E042*, 2022.
- [9] J. Pachel, *Railway Signalling Principles: Edition 2.0*. University Library Braunschweig, 2021.
- [10] S. Engels, T. Peham, J. Przigoda, N. Przigoda, and R. Wille, “Design tasks and their complexity for Hybrid Level 3 of the European Train Control System,” 2023.
- [11] Gurobi Optimization, LLC, “Gurobi Optimizer Reference Manual,” 2023.