Deadlock Detection in Real-Time Railway Dispatching Using SMT

Bjørnar Luteberget[0000−0002−3444−6209]* and Carlo Mannino[0000−0003−2974−9165]

SINTEF Digital, Oslo, Norway
bjornar.luteberget@sintef.no

Abstract. Congestion and delays on the railway are handled by teams of dispatchers who modify schedules and routes of trains in a region to minimize overall delay. Reducing delays locally always runs the risk of creating complications for other stations, trains, or regions. In some cases, trains become deadlocked, i.e., a group of trains mutually blocking each other's paths, which is a highly complex and costly situation. Traffic Management Systems (TMSs) can use special-purpose deadlock algorithms to check, before committing to a dispatching decision, whether it will cause a deadlock.
The scientific literature describes two main types of mathematical models for deadlocks: transition system models and precedence graph models. We have developed a novel combination of applying a Satisfiability Modulo Theories solver to the precedence graph model. We compare this to deadlock detection algorithms based on transition systems and on Mixed-Integer Linear Programming solvers, and demonstrate our method's performance on real-world benchmark instances. Scaling up deadlock detection to handle more complex networks enables dispatchers to make quick and confident decisions, whether manual or algorithm-assisted.

Keywords: Railway deadlock detection · SMT solver · Train dispatching · Precedence graph · Traffic management system

1 Introduction

Efficient railway operations require centrally coordinated train movements, but even detailed plans need adjustment when delays happen. Dispatchers perform this real-time coordination aided by Traffic Management Systems (TMSs), which show real-time train positions and allow timetable modifications. Some TMSs also provide decision support by computing optimal timetable changes to limit delay propagation. Whether these modifications are made manually or by an algorithm, determining the optimal choices is a computationally hard problem [1], and in practice it is necessary to limit the scope of decisions in time and space [2]. Although such locally optimal decisions are often also globally near-optimal, disregarding global effects risks creating a *deadlock*, where some set of trains are mutually blocking each other's paths. Deadlocks are costly to resolve. In European railway networks dominated by passenger traffic, deadlocks are relatively

uncommon, though they can happen [3]. In US networks dominated by long freight trains, deadlocks are a more pressing issue. Indeed, specialized deadlock detection algorithms are implemented in an automatic dispatching tool deployed by Union Pacific, the largest class 1 American railroad [2,4,5]. Because deadlock detection is less computationally demanding than full schedule optimization, a specialized deadlock checker can cover larger regions, longer time horizons, or more trains than the optimizer, helping dispatchers make re-scheduling decisions with greater confidence.

Detecting (potential) deadlocks in job-shop scheduling has received significant interest from the combinatorial optimization community [6,7,8,9]. For railway deadlocks specifically, there are two main types of mathematical model. Transition systems models represent the position of all trains at some point in time and define allowable transitions in this state space [10,11]. In contrast, precedence graph models describe each train's trajectory independently, imposing only a partial ordering of transitions as required by shared resources [5]. This is also the main modeling approach used in schedule optimization [2,12,13].

Mathematical models for railways are typically solved using mixed-integer linear programming (MILP) solvers, but we have previously shown that SAT solvers for deadlock transition system models can significantly outperform MILP [11]. For precedence graph models, however, there is no compact SAT representation, but it is possible to efficiently generate constraints as needed. Such incremental constraint generation has been studied in an Integer Programming context [13,5], and in an SMT context for delay management [14]. In this paper, we show how the precedence graph approach to deadlock detection can also be solved with an SAT-based SMT solver, and that this framework significantly outperforms the MILP algorithm from [5], currently used in a state-of-the-art industrial application at Union Pacific [2].

2 Background

2.1 The Bound-to-Deadlock Problem

A train occupies a set of one or more *blocks*. We call such a set of blocks a *position*. Consider a set of trains T, and define, for each train $t \in T$ a *movement graph* $G^t = (N^t, A^t)$, where the nodes N^t are a set of positions, and the arcs A^t are atomic movements $(a, b) \in A^t$, representing that the train can move from position a to b, by reserving and traversing the blocks between one signal and the next. We assume that each G^t is acyclic. An example of a fixed-block infrastructure and a movement graph for a train is shown in Fig. 1.

In the real-time setting, each train is at a given position $\bar{a}^t \in N^t$. The movement graph also contains a final node $\emptyset$ representing the train finishing. We assume that G^t contains at least one path from $\bar{a}^t$ to $\emptyset$. If it is possible to successfully dispatch all the trains, then we must be able to do the following. For each train t, we select a *path*, i.e., a sequence of positions $\bar{a}^t, \pi_1^t, \pi_2^t, \ldots, \emptyset \in N^t$ that forms a path in the graph G^t, starting from $\bar{a}^t$ and ending in $\emptyset$. Let

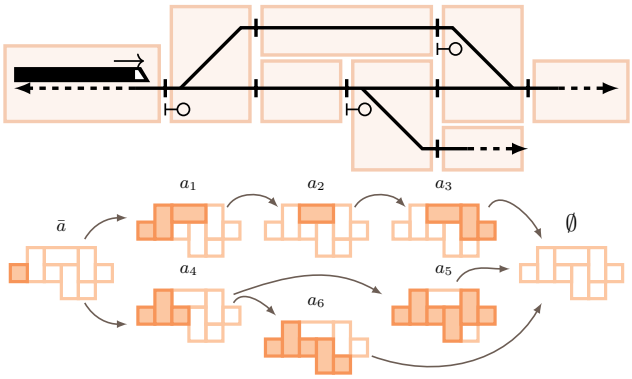

Fig. 1. Top: a train is traveling on the railway from the dashed line on the left to either of the dashed lines on the right. Bottom: a movement graph corresponding to the situation above.

$\mathcal{N} = \{(t, a) \mid t \in T,\, a \in N^t\}$ be the set of all pairs of trains and positions. Then, we also select a *schedule*, defining the order in which the positions will be reached, and thus the order in which the movements will be performed. We define a schedule as a strict partial order $\prec$ over the set $\mathcal{N}$ with these additional restrictions:

(R1) Following the path of each train $t \in T$, we must have $(t, \bar{a}^t) \prec (t, \pi_1^t) \prec (t, \pi_2^t) \prec \ldots \prec (t, \emptyset)$.

(R2) Consider any pair of distinct trains $t_1, t_2 \in T$ and pair of corresponding positions $a_1 \in N^1$ and $a_2 \in N^2$, such that both a_1 and a_2 are part of the selected path for the trains t_1 and t_2, respectively. If the two positions have a non-empty intersection, i.e., if $a_1 \cap a_2 \neq \emptyset$, then the trains cannot be in this pair of positions simultaneously. Note that because the intersection is non-empty, neither a_1 nor a_2 is a final position. So, let b_1 denote the successor position of a_1 in the path of train t_1, and correspondingly b_2 for a_2.
To avoid the trains t_1 and t_2 being, respectively, in positions a_1 and a_2 simultaneously, then either train t_1 must reach position b_1 before t_2 reaches a_2, or vice versa. If a_1 is the initial position of train t_1 (i.e., $a_1 = \bar{a}^{t_1}$), then we must have $(t_1, b_1) \prec (t_2, a_2)$ (and correspondingly for a_2). Otherwise, we must have either

$$(t_1, b_1) \prec (t_2, a_2) \qquad \text{or} \qquad (t_2, b_2) \prec (t_1, a_1).$$

We denote by $\mathcal{C}$ the set of all such conflicting pairs $\{(t_1, a_1), (t_2, a_2)\}$.

If we can find a path for each train and a corresponding schedule $\prec$, then we can successfully dispatch the trains by ordering movements (i.e., pairs of successive positions) according to $\prec$.

Definition 1. *A set of trains T is* Live *if there exists a path for each train such that there also exists a corresponding schedule.*

If no combination of paths and schedule $\prec$ exists, then some subset of the trains will always eventually be mutually blocking each other's paths. The trains are **bound-to-deadlock** and we say that T is DEAD.

Definition 2. *A set of trains T is* DEAD *(bound-to-deadlock) if it is not* LIVE.

2.2 SAT and SMT

The Boolean satisfiability problem (SAT) asks whether there exists an assignment of truth values to Boolean variables $\mathbf{x} = \langle x_1, x_2, \ldots \rangle$ that satisfies a given propositional logic formula (cf. [15]). A *literal* is either a variable x_i or its complement $\neg x_i$. A *clause* is a disjunction of literals, for example:

$$x_1 \vee x_2 \vee \neg x_3$$

Satisfiability Modulo Theories (SMT) is a framework for extending SAT solvers to handle formulas more expressive than propositional logic. In this paper, we will make use of a specific type of formula involving a set of integer variables $y_1, y_2, \ldots$, and difference constraints on the form $y_1 - y_2 \geq k$, for a given constant $k \in \mathbb{Z}$. Now, we define an *atom* to be either a difference constraint, a negated difference constraint, or a literal (as before). We re-define a *formula* to be a set of clauses of atoms. A clause of atoms could be, for example:

$$x_1 \vee \neg x_2 \vee (y_3 - y_4 \geq k)$$

This class of formulas is called satisfiability modulo integer difference logic (IDL). IDL is sufficient for our scheduling application because the constraints only involve pairwise ordering of events, and IDL admits a graph-based feasibility characterization (Proposition 1) that our algorithm exploits directly. IDL feasibility can be determined by an extended SAT solver that represents each difference constraint by an auxiliary Boolean variable, and checks whether the active difference constraints constitute a satisfiable set of IDL constraints. To determine whether a set of IDL constraints resulting from the assignment $\mathbf{x}$ are satisfiable, we can define a weighted directed graph $\mathcal{G}(\mathbf{x})$ whose nodes are the integer variables $y_1, y_2, \ldots$, and, for each constraint $y_2 - y_1 \geq k$, has an arc from y_1 to y_2 with weight k.

Proposition 1. *A set of IDL constraints are satisfiable iff the graph $\mathcal{G}(\mathbf{x})$ has no cycles where the sum of weights of the edges is positive. Proof:* See [16].

Remark 1. For any assignment $\mathbf{x}$ to the Boolean variables such that the resulting set of IDL constraints is unsatisfiable, we can identify a cycle in $\mathcal{G}(\mathbf{x})$ and eliminate that assignment in the SAT solver by adding a corresponding clause.

This would result in constraints such as:

$$\neg(y_2 - y_1 \geq 1) \ \vee \ \neg(y_3 - y_2 \geq 1) \ \vee \ \neg(y_1 - y_3 \geq 1)$$

In our experiments below, we use a SAT solver based on MiniSat [17], and we check IDL consistency using an incremental graph algorithm as described in [16].

3 IDL Formula for Deadlock Detection

Intuitively, we encode the deadlock detection problem as a logical formula that asks: is it possible to find a route for each train and an ordering of train movements such that no two trains need to occupy the same block at the same time? The SAT solver searches over route choices and conflict resolutions, while the IDL theory enforces that the resulting movement ordering is consistent.

More precisely, we build a formula of satisfiability modulo IDL which is satisfiable if and only if T is LIVE. For each combination of train $t \in T$ and position $a \in N^t$, we define a Boolean variable x_a^t representing whether the position a may be in the path of train t, and an integer variable $u_a^t \in \mathbb{Z}$, which will be used to represent the schedule. Note that we force the assignment $x_{\bar{a}^t}^t = \text{TRUE}$ for the initial positions $\bar{a}^t$. Also, for each conflict $\{(t_1, a_1), (t_2, a_2)\} \in \mathcal{C}$, we define two Boolean variables c_1, c_2 representing which of the two trains goes first.

To make sure that the selected positions, i.e., the positions a for which x_a^t is assigned TRUE, contains at least one path from $\bar{a}^t$ to $\emptyset$, we have the clauses:

$$\neg x_a^t \vee \bigvee_{(a,b)\in N^t} x_b^t \qquad \forall t \in T, \quad \forall a \in N^t \setminus \{\emptyset\} \tag{1}$$

To ensure that the schedule respects the train path (R1), we have the clauses:

$$\neg x_a^t \vee \neg x_b^t \vee (u_b^t - u_a^t \geq 1) \qquad \forall t \in T, \quad \forall (a,b) \in N^t. \tag{2}$$

For any conflict that may appear in the trains' paths, we need to choose which train goes first:

$$\neg x_{a_1}^{t_1} \vee \neg x_{a_2}^{t_2} \vee c_1 \vee c_2 \qquad \forall \{(t_1, a_1), (t_2, a_2)\} \in \mathcal{C} \tag{3}$$

Note that if $a_1 = \bar{a}^{t_1}$, then we force the assignment $c_1 = \text{TRUE}$ (and correspondingly for a_2 and c_2). Finally, to ensure that the schedule satisfies (R2), we have the clauses:

$$\neg c_1 \vee \neg x_{b_1}^{t_1} \vee \left(u_{a_2}^{t_2} - u_{b_1}^{t_1} \geq 1\right) \qquad \forall \{(t_1, a_1), (t_2, a_2)\} \in \mathcal{C}, \quad \forall (a_1, b_1) \in N^{t_1} \tag{4}$$

Note that, because the conflicts are unordered, we also implicitly have the corresponding clauses with indices 1 and 2 swapped. Let F denote the conjunction of all the clauses (1)-(4).

Proposition 2. *The formula F is satisfiable if and only if T is* LIVE.

Proof (sketch). If F is satisfiable, consider any assignment to the x, u, and c variables satisfying (1)-(4). For each train t, the subset of nodes $\bar{N}^t \subseteq N^t$ for which x_a^t is assigned TRUE contains at least one path from $\bar{a}^t$ to $\emptyset$. Let any such path be the selected path $\bar{a}^t, \pi_1^t, \pi_2^t, \ldots, \emptyset$ for train t. Then, define $\prec$ such that $(t_1, a_1) \prec (t_2, a_2)$ iff $u_{a_1}^{t_1} < u_{a_2}^{t_2}$. Then, $\prec$ is clearly a strict partial order. Furthermore, $\prec$ must include the elements required by (R1) because of (2), and (R2) because of (3)-(4). This shows that T is LIVE.

Conversely, if T is LIVE, take any valid combination of paths and schedule $\prec$. Set $x_a^t = \text{TRUE}$ iff a is in the path of train t, and let u_a^t be the index of (t, a) in a topological ordering of $\prec$. For each conflict, set $c_1 = \text{TRUE}$ iff $(t_1, b_1) \prec (t_2, a_2)$ (and vice versa for c_2); otherwise $c_1 = c_2 = \text{FALSE}$. This assignment satisfies F.

Table 1. Performance evaluation on the problem instances from [10] and [5]. The n_r and n_t columns show the number of routes and trains, respectively. "Steps" show the number of transition system steps, "Iter." show the number of constraint generation iterations.

Instance				MILP ticks (reported in [10,5])		MILP cycles (reported in [5])		SAT transition system [11]		SMT cycles (this paper)	
#	Result	n_r	n_t	Steps	Time (s)	Iter.	Time (s)	Steps	Time (s)	Iter.	Time (s)
T11	Dead	62	2	27	17.60		N/A	3	0.00	650	0.01
T12	Dead	62	4	39	>60.00		N/A	10	0.10	2217	0.07
T13	Dead	62	4	39	>60.00		N/A	10	0.01	1262	0.04
T14	Live	62	4	39	3.27		N/A	6	0.00	103	0.04
T15	Dead	46	4	42	>60.00		N/A	6	0.01	48	0.00
T16	Live	62	5	50	5.33		N/A	5	0.01	50	0.00
T17	Live	62	4	50	43.11		N/A	6	0.01	170	0.00
T18	Dead	62	4	50	>60.00		N/A	6	0.02	234	0.00
T19	Dead	62	5	51	>60.00		N/A	6	0.01	132	0.00
T20	Dead	70	5	57	>60.00		N/A	8	0.05	7	0.00
C11	Dead	274	5	114	>60.00	3	0.69	7	0.09	652	0.02
C12	Live	35	5	21	0.08	0	0.02	4	0.00	3	0.00
C13	Dead	257	6	68	0.48	0	0.13	8	0.77	1	0.00
C14	Live	317	6	77	1.79	3	1.19	4	0.01	26	0.01
C15	Dead	102	6	32	0.20	0	0.05	7	0.02	18	0.00
C16	Live	142	6	34	0.54	0	0.16	4	0.00	14	0.00
C17	Live	137	6	70	1.86	3	0.24	6	0.01	71	0.00
C18	Live	325	7	54	3.46	0	6.35	5	0.08	38	0.06
C19	Live	219	9	108	4.50	29	10.90	7	0.02	117	0.01
C20	Live	837	9	194	14.73	1	3.93	4	0.07	378	0.09

4 Performance Results

We measured the running time of the algorithm on a set of real-world problem instances supplied by Siemens Mobility Italy. Computational experiments were performed on an AMD Ryzen 9 7900X machine running Ubuntu 22.04. Algorithms were implemented using the Rust programming language.

An overview of the results is shown in Table 1. We compare our approach ("SMT cycles") and a previous SAT-based algorithm ("SAT transition system") to the reported performance from [10,5] (because the source code used in those papers was not available to us). Even though these results are produced by different hardware, solver libraries, etc., the marked difference in running times between the algorithms gives an indication that the SAT and SMT approaches are significantly faster, and thus potentially scalable to larger-scale problem instances. The implementation of the SAT-based algorithms, and the problem instances used in the comparison, are available online at `https://github.com/luteberget/deadlockrail`.

As described in Sect. 2.2, the solving process consists of assigning Boolean variables, identifying cycles, and adding the corresponding clauses (cf. Remark 1). We observe that the cycles eliminated by the SMT algorithm are short: 92% of them are of length less than 10, and the most common length is 4 (see Fig. 2). This might be an underlying reason for why eliminating cycles in $\mathcal{G}(\mathbf{x})$ is effective as a deadlock detection algorithm.

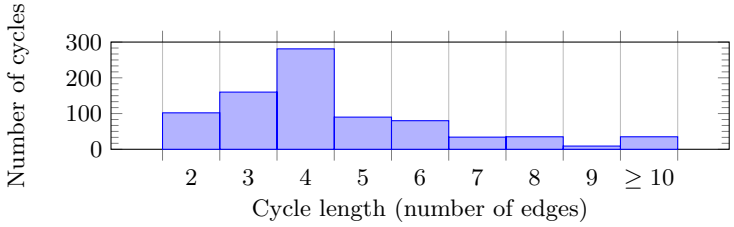

Fig. 2. Histogram of the length of the cycles (in terms of the number of edges of the graph $\mathcal{G}(\mathbf{x})$) arising during a run through the solving of all the instances from [5].

5 Conclusions

In practice, deadlock checking is invoked immediately before committing to a dispatching decision, so computational efficiency is essential. Our model assumes a centralized view of trains in a region and a complete movement graph under fixed-block signaling. Deadlocks arising from constraints not captured in the movement graph (e.g., crew scheduling, energy limitations, or platform assignments) are outside the scope of this model.

The performance of our SAT-based algorithm indicates that exploring SAT and SMT-based algorithms for other operations research problems can be a fruitful avenue for future research. We have shown that our algorithm for deadlock detection significantly outperforms the industrial state-of-the-art on real-world problem instances. Deadlock detection has an important role in scaling up and increasing the performance of decision support in TMSs, which, in turn, contributes to more efficient railway operations.

Acknowledgements. Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the Europe's Rail Joint Undertaking. Neither the European Union nor the granting authority can be held responsible for them. The projects "FP1-MOTIONAL" and "FP5-TRANS4M-R" are supported by the Europe's Rail Joint Undertaking and its members.

References

1. Lu, Q., Dessouky, M., Leachman, R.C.: Modeling train movements through complex rail networks. ACM Transactions on Modeling and Computer Simulation (TOMACS) **14**(1), 48–75 (2004). https://doi.org/10.1145/974734.974737
2. Boccia, M., Dal Sasso, V., Lamorgese, L., Mannino, C., Ventura, P.: Optimizing train dispatching for the Union Pacific Railroad. Tech. rep., Siemens Mobility, Rome, Italy (2025), available at https://arxiv.org/abs/2503.19535

3. Pachl, J.: Avoiding deadlocks in synchronous railway simulations. In: 2nd International Seminar on Railway Operations Modelling and Analysis. 2007, Hannover, Germany (2007). https://doi.org/10.24355/dbbs.084-200704030200-0
4. Dal Sasso, V., Lamorgese, L., Mannino, C., Tancredi, A., Ventura, P.: Easy cases of deadlock detection in train scheduling. Operations Research **70**(4), 2101–2118 (2022). https://doi.org/10.1287/opre.2022.2283
5. Dal Sasso, V., Lamorgese, L., Mannino, C., Onofri, A., Ventura, P.: A 0, 1 linear programming approach to deadlock detection and management in railways. Transportation Science **59**(1), 187–205 (2025). https://doi.org/10.1287/trsc.2024.0521
6. Arbib, C., Italiano, G.F., Panconesl, A.: Predicting deadlock in store-and-forward networks. Networks **20**(7), 861–881 (1990). https://doi.org/10.1002/net.3230200705
7. Li, F., Sheu, J.B., Gao, Z.Y.: Deadlock analysis, prevention and train optimal travel mechanism in single-track railway system. Transportation research part B: methodological **68**, 385–414 (2014). https://doi.org/10.1016/j.trb.2014.06.014
8. Oriolo, G., Russo Russo, A.: Avoiding deadlocks via weak deadlock sets. Networks **86**(1), 48–56 (2025). https://doi.org/10.1002/net.22275
9. Pachl, J.: Deadlock avoidance in railroad operations simulations. Tech. rep., Institute of Railway Systems Engineering and Traffic Safety, Technische Universität Braunschweig (2011), paper No. 11-0175, 90th Annual Meeting of the Transportation Research Board, Washington DC
10. Dal Sasso, V., Lamorgese, L., Mannino, C., Onofri, A., Ventura, P.: The tick formulation for deadlock detection and avoidance in railways traffic control. Journal of Rail Transport Planning & Management **17**, 100239 (2021). https://doi.org/10.1016/j.jrtpm.2021.100239
11. Luteberget, B.: Improving online railway deadlock detection using a partial order reduction. Electronic Proceedings in Theoretical Computer Science **348**, 110–127 (Oct 2021). https://doi.org/10.4204/eptcs.348.8
12. Lamorgese, L., Mannino, C., Pacciarelli, D., Krasemann, J.T.: Train dispatching. In: Handbook of optimization in the railway industry, pp. 265–283. Springer (2018). https://doi.org/10.1007/978-3-319-72153-8_12
13. Lamorgese, L., Mannino, C.: A noncompact formulation for job-shop scheduling problems in traffic management. Operations Research **67**(6), 1586–1609 (2019). https://doi.org/10.1287/opre.2018.1837
14. Leutwiler, F., Corman, F.: A logic-based Benders decomposition for microscopic railway timetable planning. European Journal of Operational Research **303**(2), 525–540 (2022). https://doi.org/10.1016/j.ejor.2022.02.043
15. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Handbook of satisfiability, pp. 133–182. IOS Press (2021). https://doi.org/10.3233/FAIA200987
16. Cotton, S., Maler, O.: Fast and flexible difference constraint propagation for DPLL(T). In: International Conference on Theory and Applications of Satisfiability Testing. pp. 170–183. Springer (2006). https://doi.org/10.1007/11814948_19
17. Eén, N., Sörensson, N.: An extensible SAT-solver. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 502–518. Springer (2003). https://doi.org/10.1007/978-3-540-24605-3_37