An Iterative Model Reduction Method and a Constraint Programming Formulation for the Train Dispatching Problem

László Kálmán Trautsch[0000−0002−1589−3265] and
Bence Kovari[0000−0003−1555−640X]

Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Műegyetem rkp. 3., Budapest, H-1111, Hungary
trautschl@edu.bme.hu, kovari@aut.bme.hu

Abstract. This paper addresses the train dispatching problem, which involves rescheduling and rerouting delayed trains on a shared railway network to minimize total delay and avoid resource conflicts. Dispatching optimization can reduce overall lateness, increase network throughput, and improve the utilization of railway infrastructure. We propose an iterative model reduction method that reduces the complexity of problem instances while ensuring that no optimal solution is eliminated from the search space. A series of model transformations are introduced that tighten operation start-time bounds, merge compatible operations, prune dominated subpaths, remove redundant resource requirements, and linearize train operation graphs. The proposed reduction method can be applied either as a preprocessing step or dynamically within other solution methods. In this paper, we apply the reduction as preprocessing and propose a compact constraint programming formulation based on the reformulated model. The solution method is evaluated on the DISPLIB 2025 benchmark library, which consists of instances derived from real-world use cases. The computational results show that the proposed model reduction method can substantially simplify large problem instances and improve performance. The complete solution method was able to find high-quality feasible solutions for most benchmark instances and achieved the fourth-highest overall score in the final phase of the DISPLIB 2025 competition.

Keywords: Train Dispatching · Model Reduction · Constraint Programming

1 Introduction

Train dispatching problems arise frequently in railway systems, particularly when unexpected delays or disruptions require rapid routing and scheduling adjustments to minimize overall delay and maintain service quality. Effective dispatching decisions are essential for reliable and efficient railway traffic management, especially in dense networks where even minor disruptions can propagate quickly.

Consequently, there is considerable interest in both academia and industry in developing robust and efficient methods for real-time train dispatching.

Accordingly, train dispatching and railway rescheduling have been studied for decades, as reflected already in the survey of Cordeau et al. [1]. In the real-time setting, a major line of work models dispatching as a combined routing and scheduling problem under infrastructure conflicts. D'Ariano et al. [2] proposed a branch-and-bound method for train scheduling in a railway network, and D'Ariano et al. [3] further considered conflict resolution together with train speed coordination for real-time timetable perturbations.

Subsequent work addressed several important variants of the problem. Mannino and Mascis [4] studied optimal real-time traffic control in metro stations, while Corman et al. [5] compared centralized and distributed approaches for rescheduling trains in multiple dispatching areas. More recently, optimization-based methods have been complemented by learning-based approaches: Agasucci et al. [6] applied deep reinforcement learning to the train dispatching problem. Exact and hybrid methods also show continuing progress, for example through column generation for real-time dispatching [7], constraint-based in-station dispatching [8], and hybrid optimization for complex railway stations [9].

Several studies have addressed the computational difficulty of railway scheduling and rescheduling by reducing the set of routing or sequencing decisions before or during optimization. Zwaneveld et al. [10] formulated train routing through stations as a node packing problem and used preprocessing to remove superfluous routing alternatives. Vansteenwegen et al. [11] used dominance-based preprocessing to discard train route alternatives in a maintenance-rescheduling setting. Other approaches reduce the problem by decomposition, for example by separating line and station decisions in real-time dispatching [12]. Decomposition principles for large-scale railway scheduling are reviewed by Leutwiler and Corman [13].

In this paper, we address the train dispatching problem by proposing an iterative model reduction method that can significantly reduce problem complexity. The method combines several optimality-preserving transformations, including start-time bound tightening, operation merging, dominated-subpath pruning, redundant-resource deletion, and operation graph linearization. Rather than reducing only the set of admissible routes or decomposing the problem into smaller subproblems, the proposed method applies iterative reductions directly to the operation-graph representation of the problem, deriving a reduced model for optimization. Based on the reformulated model, we also develop a compact constraint programming formulation. We evaluate the proposed method on the publicly available instances of the DISPLIB 2025 library [14], which represent challenging real-world dispatching scenarios.

The remainder of this paper is organized as follows. Section 2 provides an overview of the train dispatching problem addressed in this paper. Section 3 describes the proposed iterative model reduction method, and Section 4 introduces the constraint programming formulation based on the reformulated model. Section 5 presents the computational results on the DISPLIB 2025 benchmark

library. Finally, Section 6 concludes the paper and outlines directions for future work.

2 Problem Definition

The train dispatching problem, as defined by the DISPLIB 2025 benchmark library [14], involves routing and scheduling a set of trains, each represented by a directed acyclic graph called an operation graph. The nodes of this graph represent train operations, while the edges represent immediate precedence relations between operations. For each train, alternative routes correspond to alternative entry-to-exit paths in its operation graph. Operations represent possible states or activities of a train, typically involving the occupation of one or more infrastructure resources. Resources represent infrastructure elements that can be occupied by at most one train at a time, such as physical track sections or block sections imposed by railway safety regulations. Each operation can have a specified minimum duration, lower and upper bounds on allowable start times, and may require exclusive access to one or more resources. The notation used in this paper and the input data are presented in Table 1.

Table 1. Notation for the input data

Symbol	Definition
I	Set of trains.
O	Set of all operations.
$O_i \subset O$	Set of operations of train i. $\{O_i\}_{i \in I}$ forms a partition of O.
$B, E \subset O$	Sets of entry and exit operations, respectively.
$S_o, P_o \subset O$	Sets of immediate successors and predecessors of operation o, respectively.
$\alpha_o, \beta_o \in \mathbb{Z}_{\geq 0}$	Lower and upper bounds on the start time of operation o, respectively.
$\delta_o \in \mathbb{Z}_{\geq 0}$	Minimum duration of operation o.
R	Set of resources.
$R_o \subseteq R$	Set of the resources required by operation o.
$\lambda_{r,o} \in \mathbb{Z}_{\geq 0}$	Release time of resource r required by operation o, i.e., the additional time that must elapse after o has ended before r can be used by an operation of another train.
C	Set of cost components.
$K_o \subset O$	Set of operations potentially conflicting with operation o, i.e., operations that may require a common resource with o.
$\mu_c \in \mathbb{Z}_{\geq 0}$	Delay threshold of cost component c.
$\omega_c, \nu_c \in \mathbb{Z}_{\geq 0}$	Linear and constant cost weights of cost component c, respectively.
$\phi : C \to O$	$\phi(c)$ is the operation associated with cost component c.

An operation graph contains exactly one entry operation, which has no incoming edges, and exactly one exit operation, which has no outgoing edges. A feasible solution to a problem instance is a chronologically ordered sequence of start events for selected operations across all trains. This global sequence must

respect the structure of the operation graphs, forming a complete path from entry to exit operations for each train, while also satisfying the start-time and minimum-duration bounds of all selected operations.

In addition, resource requirements impose further restrictions: no two trains may simultaneously occupy the same resource. If operation o occupies resource r, then an operation of another train can use r only after o has ended and the additional release time $\lambda_{r,o}$ has elapsed. An operation is considered to end when the next selected operation of the same train starts.

Feasible solutions are evaluated based on an objective function that minimizes delays and operational costs. The input data contains a set of cost components, each specifying a delay threshold for an associated operation, as well as linear and constant cost weights associated with the delay. A cost component is associated with one operation, while each operation may have any number of cost components. A cost component c is triggered if the associated operation $\phi(c)$ starts at or after its delay threshold μ_c. In that case, it contributes a constant cost ν_c and a linear cost proportional to the delay beyond the threshold, with weight ω_c. The objective function is defined in (1), where $t_{\phi(c)}$ is the assigned start time of the operation associated with cost component c, and H denotes the Heaviside step function, which takes the value 0 for negative arguments and 1 otherwise.

$$\min \sum_{c \in C} (\omega_c \cdot \max(0, t_{\phi(c)} - \mu_c) + \nu_c \cdot \mathrm{H}(t_{\phi(c)} - \mu_c)) \tag{1}$$

3 Model Reduction

To reduce problem size, we propose an iterative model reduction method that tightens lower bounds, merges operations, prunes dominated subpaths, and removes redundant resource requirements. We apply this method as a preprocessing step, but the transformations are designed so that they can also be applied efficiently during the execution of solver algorithms. To further reduce problem complexity, we also reformulate the model by linearizing the operation graphs and introducing path variables. This section presents these model reduction steps. For brevity, we omit the proofs that these transformations neither remove nor introduce any optimal solutions.

First, the start-time lower bound of each operation o is tightened by forward propagation, based on the earliest feasible end time among its immediate predecessors: $(\alpha_o \leftarrow \max(\alpha_o, \min_{o' \in P_o}(\alpha_{o'} + \delta_{o'})))$

3.1 Operation Merging

Certain operations with identical resource requirements can be merged into a single operation. The original indices of the merged operations are stored, allowing the solution to be reconstructed.

Sequential Operation Merging can be applied to a pair of operations o and o' that require identical resources ($R_o = R_{o'}$) if $S_o = \{o'\}$, $P_{o'} = \{o\}$ and o

does not have any resource requirements that can extend beyond those of operation o' ($\forall r \in R_o : \lambda_{r,o} \leq \lambda_{r,o'} + \delta_{o'}$). The delay threshold of any cost component c associated with o' is reduced by the minimum duration of o ($\mu_c \leftarrow \mu_c - \delta_o$), and the associated operation of c is updated to o ($\phi(c) \leftarrow o$). The minimum duration of operation o is then increased by the minimum duration of o' ($\delta_o \leftarrow \delta_o + \delta_{o'}$). For every resource r required by operation o, the associated release time is updated to that of o' ($\lambda_{r,o} \leftarrow \lambda_{r,o'}$). If operation o' is an exit operation of a train, o becomes the new exit ($E \leftarrow (E \cup \{o\}) \setminus \{o'\}$). Operation o acquires all immediate successors of o' and o' is removed from the immediate successors of o. ($S_o \leftarrow (S_o \cup S_{o'}) \setminus \{o'\}$, $P_{o'} \leftarrow \emptyset$, $\forall o'^{+} \in S_{o'} : P_{o'^{+}} \leftarrow (P_{o'^{+}} \cup \{o\}) \setminus \{o'\}$, $S_{o'} \leftarrow \emptyset$)

Parallel Operation Merging can be applied to a pair of operations o and o' that satisfy $P_o = P_{o'}$ or $S_o = S_{o'}$, $\alpha_o = \alpha_{o'}$, $\delta_o = \delta_{o'}$, $R_o = R_{o'}$, and $\forall r \in R_o : \lambda_{r,o} = \lambda_{r,o'}$, and for which each cost component associated with o has a corresponding cost component associated with o' whose parameters match ($\forall c \in C : (\phi(c) = o) \Rightarrow (\exists c' \in C : (\phi(c') = o') \wedge (\mu_c = \mu_{c'}) \wedge (\omega_c = \omega_{c'}) \wedge (\nu_c = \nu_{c'}))$). Similarly, each cost component associated with o' should also have a corresponding cost component associated with o. Since this merging requires the two operations to have similar properties, merging o' into o only requires updating the predecessor and successor relations. Operation o acquires all immediate predecessors and successors of o' ($P_o \leftarrow P_o \cup P_{o'}$, $S_o \leftarrow S_o \cup S_{o'}$, $P_{o'} \leftarrow \emptyset$, $S_{o'} \leftarrow \emptyset$), and the sets of immediate predecessors and successors of the adjacent operations are updated accordingly. ($\forall o^{-} \in P_o : S_{o^{-}} \leftarrow (S_{o^{-}} \cup \{o\}) \setminus \{o'\}$, $\forall o^{+} \in S_o : P_{o^{+}} \leftarrow (P_{o^{+}} \cup \{o\}) \setminus \{o'\}$)

3.2 Subpath Pruning

Certain subpaths in the operation graphs may be dominated by alternative subpaths and can therefore be eliminated. To reduce computational complexity, we examine only linear sequences of operations, meaning sequences in which every operation, except the first and the last, has exactly one immediate successor, or every operation, except the first and the last, has exactly one immediate predecessor.

Let $\pi = (o_1, \ldots, o_k)$ and $\pi' = (o'_1, \ldots, o'_\ell)$ be two sequences of operations corresponding to subpaths of an operation graph, with $o_1 = o'_1$, $o_k = o'_\ell$, and $k, \ell \geq 2$. We say that π dominates π' if there exists an integer $m \geq 1$ and decompositions of π and π' into m consecutive segments, $\pi = \pi^{(1)} \cdots \pi^{(m)}$ and $\pi' = \pi'^{(1)} \cdots \pi'^{(m)}$, such that for each $j = 1, \ldots, m$, the segment $\pi^{(j)}$ dominates the segment $\pi'^{(j)}$. Here, a segment $\pi^{(j)} = (o_1^{(j)}, \ldots, o_{k_j}^{(j)})$ dominates segment $\pi'^{(j)} = (o_1'^{(j)}, \ldots, o_{\ell_j}'^{(j)})$ if the following conditions hold:

- The total minimum duration of segment $\pi^{(j)}$ is less than or equal to that of segment $\pi'^{(j)}$. ($\sum_{i=1}^{k_j} \delta_{o_i^{(j)}} \leq \sum_{i=1}^{\ell_j} \delta_{o_i'^{(j)}}$)
- Segment $\pi^{(j)}$ does not need to start later than segment $\pi'^{(j)}$. ($\alpha_{o_1^{(j)}} \leq \alpha_{o_1'^{(j)}}$)

- Segment $\pi^{(j)}$ does not contain operations that require additional resources, or the same resources with larger release times, than any operation in segment $\pi'^{(j)}$. $(\forall o \in \pi^{(j)}, \forall o' \in \pi'^{(j)}, \forall r \in R_o : (r \in R_{o'}) \wedge (\lambda_{r,o} \leq \lambda_{r,o'}))$
- Operations $o_1^{(j)}$ and $o_1'^{(j)}$ are the only operations in segments $\pi^{(j)}$ and $\pi'^{(j)}$ that may be associated with cost components. $(\forall c \in C : \phi(c) \in \pi^{(j)} \cup \pi'^{(j)} \Rightarrow \phi(c) \in \{o_1^{(j)}, o_1'^{(j)}\})$
- Each cost component associated with operation $o_1^{(j)}$ has a corresponding cost component associated with operation $o_1'^{(j)}$ whose parameters are at least as restrictive. $(\forall c \in C : (\phi(c) = o_1^{(j)}) \Rightarrow$ $(\exists c' \in C : (\phi(c') = o_1'^{(j)}) \wedge (\mu_c \geq \mu_{c'}) \wedge (\omega_c \leq \omega_{c'}) \wedge (\nu_c \leq \nu_{c'})))$

If a sequence of operations π' is dominated by another sequence π, and every operation in π', except the first and the last, has exactly one immediate successor $(\forall n \in \{2, \ldots, \ell-1\} : S_{o'_n} = \{o'_{n+1}\})$, then the first internal edge of π' is removed from the graph, since any path containing that edge must follow a dominated subpath $(S_{o'_1} \leftarrow S_{o'_1} \setminus \{o'_2\}, \quad P_{o'_2} \leftarrow P_{o'_2} \setminus \{o'_1\})$. The reverse case is handled analogously: if every internal operation has exactly one immediate predecessor, the last internal edge of π' is removed.

After the pruning steps, any operation o that becomes unreachable ($S_o = \emptyset$ or $P_o = \emptyset$) is removed from the graph, as well as from the successor and predecessor sets of all other operations. $(\forall o' \in O : P_{o'} \leftarrow P_{o'} \setminus \{o\}, S_{o'} \leftarrow S_{o'} \setminus \{o\})$

3.3 Resource Deletion

A resource r that is required only by the operations of a single train $(\exists i \in I : \forall o \in O \setminus O_i : r \notin R_o)$ cannot induce conflicts and is therefore removed. If a resource r is always required together with another resource r' $(\forall o \in O : r \in R_o \Rightarrow r' \in R_o)$, and for every operation o that requires r we have $\lambda_{r,o} \leq \lambda_{r',o}$, then r is redundant and therefore removed. $(\forall o \in O : R_o \leftarrow R_o \setminus \{r\})$

3.4 Iterative Model Reduction

The previously presented model reduction transformations are applied iteratively and hierarchically. After each merging transformation, the affected operations are re-examined for further merging and possible tightening of start-time lower bounds. Similarly, whenever a dominated path is pruned or an unreachable operation is removed, the affected part of the graph is re-examined for newly dominated paths, newly unreachable operations, further merging, and further lower-bound tightening. Resource deletions may trigger further resource deletions, path pruning, operation merging, and lower-bound tightening.

As part of preprocessing, four topological traversals are performed on each operation graph. The first traversal performs lower-bound tightening, the second applies all eligible merging transformations, the third prunes dominated paths and removes unreachable operations, and the fourth deletes redundant resources. During each traversal, every reduction may immediately trigger further applicable reductions, and this effect may propagate through the entire graph.

3.5 Path Variable Assignment

In this step, the possible paths in the operation graph of each train are decomposed into sets of subpaths, and a unique path variable is assigned to each set to indicate whether the subpaths in that set are selected in the solution. The procedure begins by assigning a path variable to every operation and every possible succession between operations. An operation is selected if and only if it has exactly one selected incoming edge and one selected outgoing edge, except for entry and exit operations, which are always selected. Based on this, equality relationships between sums of variables are recorded to identify variables that must always be equal, and such variables are then replaced by a single path variable. As a result, the entry and exit operations of the graph are assigned the same path variable, and a single variable can indicate the selection of multiple operations and successions across multiple subpaths rather than only one operation or succession.

3.6 Operation Graph Linearization

Finally, to reduce problem complexity, the operation graph of each train is linearized into a single chain of aggregated operations. Each aggregated operation a is assigned a set G_a of original operations associated with it, and a set S_a^{agg} containing at most one immediate successor aggregated operation. Starting from the entry operation, the graph is processed iteratively by maintaining the set of currently reachable but not yet processed operations. In each step, operations from this set whose predecessors have all already been processed are grouped into a new aggregated operation, which becomes the immediate successor of the previous aggregated operation, and the set of reachable operations is updated.

After the operation graph linearization, the path variables ensure that, for each train, a single valid path of original operations is selected in the solution, and that the constraints defined for the selected original operations are enforced on the aggregated operations containing them. In this way, the subsequent model formulation requires fewer variables and constraints, and these become more strongly linked, while the solution space remains unchanged.

4 Constraint Programming Model

After applying the model reduction transformations, we obtain a reformulated model with linearized operation graphs, which form the basis of the proposed compact constraint programming model. This linearization allows operation conflict constraints to be formulated based on the start times of successor aggregated operations, thereby reducing the number of decision variables and linking the corresponding constraints more directly. In this section, we introduce the decision variables and constraints of the CP formulation. Table 2 summarizes the additional notation and decision variables used in the formulation. Throughout this section, the superscript "agg" denotes the aggregated-operation counterpart of notation previously defined for operations; in particular, B^{agg}, E^{agg}, P_a^{agg}, and S_a^{agg} denote the corresponding sets for aggregated operations.

Table 2. Notation used in the CP formulation

Symbol	Definition
A	Set of aggregated operations.
$G_a \subset O$	Set of original operations associated with aggregated operation a. $\{G_a\}_{a \in A}$ forms a partition of O.
$\eta_{o,o'} \in \mathbb{Z}_{\geq 0}$	The maximum resource release time required between the end of operation o and the start of operation $o' \in K_o$, if o precedes o'.
$\psi : C \to A$	$\psi(c)$ is the aggregated operation associated with cost component c.
X	Set of path decision variables. Each variable corresponds to one or more subpaths in the operation graph of a train. For $x \in X$, x is 1 if all subpaths associated with x are traversed by the train, 0 otherwise.
$t_a \in \mathbb{Z}_{\geq 0}$	Start time decision variable of aggregated operation a.
$q_{a,a'} \in \{0, 1\}$	Binary decision variable equal to 1 if aggregated operation a must finish before aggregated operation a' starts, and 0 otherwise.
$y_c \in \{0, 1\}$	Binary decision variable equal to 1 if cost component c is triggered, and 0 otherwise.
$z_c \in \mathbb{Z}_{\geq 0}$	Decision variable equal to the linear cost induced by cost component c when it is triggered.
$\chi^{\text{op}} : O \to X$	$\chi^{\text{op}}(o)$ is the path variable associated with operation o.
$\chi^{\text{suc}} : O \times O \to X$	$\chi^{\text{suc}}(o, o')$ is the path variable associated with the succession between operation o and o', where $o \in O$ and $o' \in S_o$.

4.1 Constraints and Objective

Correct operation assignments are enforced through constraints on the path variables. Constraints 2, 3, and 4 ensure path consistency and that exactly one path is selected from the entry operation to the exit operation of each train.

$$\chi^{\text{op}}(o) = 1 \quad \forall o \in B \tag{2}$$

$$\chi^{\text{op}}(o) = \sum_{o^- \in P_o} \chi^{\text{suc}}(o^-, o) \quad \forall o \in O \setminus B \tag{3}$$

$$\chi^{\text{op}}(o) = \sum_{o^+ \in S_o} \chi^{\text{suc}}(o, o^+) \quad \forall o \in O \setminus E \tag{4}$$

The correct start times and minimum durations of operations on selected subpaths are enforced by Constraints 5 and 6, while Constraints 7 ensure that aggregated operations containing no selected operations have zero duration.

$$\chi^{\text{op}}(o) \Rightarrow \alpha_o \leq t_a \leq \beta_o \quad \forall a \in A,\, o \in G_a \tag{5}$$

$$t_a \geq t_{a^-} + \sum_{o \in G_{a^-}} \delta_o \cdot \chi^{\text{op}}(o) \quad \forall a \in A \setminus B^{\text{agg}},\, a^- \in P_a^{\text{agg}} \tag{6}$$

$$\left(\neg \bigvee_{o \in G_a} \chi^{\text{op}}(o)\right) \Rightarrow t_a = t_{a^+} \quad \forall a \in A \setminus E^{\text{agg}},\, a^+ \in S_a^{\text{agg}} \tag{7}$$

Constraints 8 enforce a precedence decision between each pair of aggregated operations that contain potentially conflicting selected operations.

$$(\chi^{\text{op}}(o) \wedge \chi^{\text{op}}(o')) \Rightarrow q_{a,a'} + q_{a',a} = 1 \quad \forall a, a' \in A,\, o \in G_a,\, o' \in G_{a'} \cap K_o \tag{8}$$

Constraints 9 enforce that if an aggregated operation a precedes another aggregated operation a', then a' cannot start before the end time of a plus the maximum release time associated with potentially conflicting pairs of selected operations in a and a'. The end time of a is the start time of its successor aggregated operation a^+.

$$\begin{aligned}(q_{a,a'} \wedge \chi^{\text{op}}(o) \wedge \chi^{\text{op}}(o')) \Rightarrow t_{a'} \geq t_{a^+} + \eta_{o,o'} \\ \forall a \in A \setminus E^{\text{agg}},\, a' \in A,\, a^+ \in S_a^{\text{agg}},\, o \in G_a,\, o' \in G_{a'} \cap K_o\end{aligned} \tag{9}$$

No aggregated operation may precede the entry aggregated operation of a train, and the exit aggregated operation of a train may not precede any other aggregated operation. This is enforced by Constraints 10.

$$q_{a,a'} = 0 \quad \forall (a, a') \in (E^{\text{agg}} \times A) \cup (A \times B^{\text{agg}}) \tag{10}$$

Constraints 11 ensure that if the operation associated with a cost component c is selected, then either the start time of the associated aggregated operation remains below the delay threshold or the variable y_c takes the value 1, indicating that cost component c is triggered. Constraints 12 enforce that if c is triggered, then the corresponding cost variable z_c is equal to the weighted delay beyond the threshold.

$$(\neg y_c \wedge \chi^{\text{op}}(\phi(c))) \Rightarrow t_{\psi(c)} < \mu_c \quad \forall c \in C \tag{11}$$

$$y_c \Rightarrow z_c = \omega_c \cdot (t_{\psi(c)} - \mu_c) \quad \forall c \in C \tag{12}$$

The objective of the problem is to minimize the total cost induced by the cost components. Each triggered cost component c contributes a constant cost ν_c and a linear delay cost given by the decision variable z_c, as defined in (13).

$$\min \sum_{c \in C} (\nu_c \cdot y_c + z_c) \tag{13}$$

4.2 Deadlock Prevention

In our CP model, certain configurations can lead to circular-wait deadlocks that are not excluded by the constraints presented above. Specifically, if certain resource release times are zero, the model may allow a set of operations to start simultaneously even if each of these operations depends on another to release a required resource first. This creates a cyclic dependency that is feasible in the CP model but contradicts the problem definition.

To eliminate the possibility of such deadlocks, every zero release time is replaced by a small positive value $\epsilon < 1$, thereby enforcing a strict ordering between successive uses of the same resource, without introducing additional variables and constraints that would substantially increase the complexity of the model. Since all original constants and variable domains are integer-valued, a suitable value of ϵ can always be chosen so that the set of intended optimal solutions is preserved. Specifically, we set $\epsilon = 0.001$ and scale all constants and variable domains by $1/\epsilon$ so that they remain integer-valued. We also modify the constraints related to the cost components to ensure that these small delays do not affect the objective value.

We also employ auxiliary constraints that directly rule out two-train deadlock configurations and infeasible precedence relations between two trains. In particular, if $q_{a,a'} = 1$ for some aggregated operations a and a', then the opposite ordering cannot be assigned to the corresponding earlier and later aggregated operations of the two trains, as well as for the immediate predecessors and successors of a and a', so the relevant $q_{\hat{a}',\hat{a}}$ variables are set to 0. These constraints are not required for correctness, but can improve computational performance in practice. The constraints can be applied only if the operation graph of each train has been linearized.

5 Experimental Results

5.1 Experimental Setup

The algorithm was tested on the instances of the publicly available benchmark library developed for the DISPLIB 2025 competition [14]. The library contains 112 different real-world problem instances of varying difficulty, ranging from 4 to 505 trains, 113 to 52,741 operations, and 17 to 50,910,450 potential conflicts. The instances are divided into 13 categories, based on how they were produced.

The solution method was implemented in Python, using the CP-SAT solver [15] for the constraint programming model. Experiments were run on a machine with an AMD Ryzen 9 9950X processor (4.3 GHz base clock, up to 5.7 GHz boost) and 192 GB of RAM. In accordance with the DISPLIB 2025 competition rules [14], we limited each run to 8 CPU threads, 32 GB of RAM, and a 10-minute wall-time per instance. For each tested configuration, a single run of the solution method was performed on each problem instance of the dataset.

5.2 Results

The results of the solution method, aggregated by instance category, are presented in Table 3. To assess the effect of the iterative model reduction method, we compare the results obtained with a CP formulation based on the original model to those obtained with the presented CP formulation based on the preprocessed model. The comparison shows that the iterative model reduction method substantially improves the computational performance of the solution method, most notably in the number of feasible solutions found. Without preprocessing, feasible solutions are obtained for only 55 of the 112 problem instances, whereas with preprocessing the solution method finds feasible solutions for 100 instances. The table also reports the average solution time, with a value of 600 s assigned to instances that were not solved to optimality within the time limit or for which the solution method exhausted the available memory. This occurred frequently without iterative model reduction, contributing to the low number of feasible solutions found.

Table 3. Computational results aggregated by instance category

	CP without iterative reduction				CP with iterative reduction			
Category	Feas. sol.	Opt. sol.	Opt. gap avg	t (s) avg	Feas. sol.	Opt. sol.	Opt. gap avg	t (s) avg
smi_close	9/9	7/9	13%	144.2	9/9	9/9	0%	17.9
smi_headway	12/12	6/12	21%	336.1	12/12	10/12	12%	123.5
nor1_critical	10/10	10/10	0%	0.3	10/10	10/10	0%	0.1
nor1_full	10/10	9/10	3%	229.8	10/10	10/10	0%	128.0
nor2	5/5	5/5	0%	8.7	5/5	5/5	0%	5.6
nor3	5/5	5/5	0%	13.0	5/5	5/5	0%	5.4
nor4_large	0/5	0/5	100%	600.0	2/5	0/5	97%	600.0
nor4_small	0/5	0/5	100%	600.0	5/5	0/5	75%	600.0
nor5_large	0/5	0/5	100%	600.0	0/5	0/5	100%	600.0
nor5_small	0/5	0/5	100%	600.0	5/5	0/5	99%	600.0
swi	4/9	3/9	67%	440.1	5/9	5/9	44%	381.8
wab_large	0/16	0/16	100%	600.0	16/16	0/16	100%	600.0
wab_small	0/16	0/16	100%	600.0	16/16	0/16	91%	600.0
Total	55/112	45/112	55%	383.1	100/112	54/112	49%	335.8

The presented solution method with iterative model reduction achieved competitive results in the DISPLIB 2025 competition [14]. It obtained the fourth-highest overall score among the 15 participating teams and achieved 58 best-known solutions. These results further support the practical effectiveness of the proposed model reduction method.

We also evaluate the effect of the iterative model reduction method directly on model size in Table 4. On average over all instances, the preprocessing reduces the number of operations by 45%, potential conflicts by 53%, CP variables by 50%, and CP constraints by 22%, while requiring only an average of 2.7 s of additional preprocessing time. This confirms that the proposed reductions substantially simplify the resulting CP model at a relatively low computational cost. The largest reduction is observed in the number of potential conflicts, reaching

84% on average in one instance category. In contrast, the relatively smaller reduction in the number of CP constraints is explained by the fact that the solution method with iterative reduction can use the auxiliary deadlock-prevention constraints, whereas these cannot be applied without operation graph linearization.

Table 4. Effect of iterative reduction on model size

Category	Without iterative reduction				With iterative reduction				
	Op. avg	Conf. avg	CP var. avg	CP constr. avg	Op. reduct. avg	Conf. reduct. avg	CP var. reduct. avg	CP constr. reduct. avg	Extra prep. t (s) avg
smi_close	1318	13069	28368	63395	74%	84%	83%	69%	0.0
smi_headway	1413	13092	28559	63493	70%	81%	67%	54%	0.0
nor1_critical	464	1169	3089	6772	38%	41%	37%	14%	0.0
nor1_full	6304	257835	526005	1176102	38%	45%	44%	7%	0.9
nor2	1586	10447	23498	53596	43%	45%	43%	12%	0.2
nor3	1290	9636	21308	48504	35%	37%	36%	5%	0.1
nor4_large	27825	2336507	4720977	10532750	49%	61%	60%	20%	13.0
nor4_small	17238	765103	1559799	3481042	49%	61%	60%	20%	3.8
nor5_large	43416	3949489	7974124	17671460	50%	67%	66%	25%	26.1
nor5_small	26945	1500954	3048549	6770802	50%	66%	66%	25%	7.7
swi	20333	13328455	26679682	54533588	34%	58%	57%	27%	2.9
wab_large	5802	126197	261547	593399	36%	37%	36%	7%	0.4
wab_small	3422	41976	89315	204905	35%	35%	33%	4%	0.1
Total	9094	1503324	3021082	6335064	45%	53%	50%	22%	2.7

6 Conclusions and Future Work

In this paper, we presented an iterative model reduction method for the train dispatching problem, together with a constraint programming formulation based on the reformulated model. We introduced a series of model transformations that tighten lower bounds, merge operations, prune dominated subpaths, remove redundant resources, and linearize the operation graphs while preserving optimal solutions. We applied the iterative model reduction as a preprocessing step and formulated a compact CP model based on the resulting linearized operation graphs. Computational results on the DISPLIB 2025 benchmark library show that the proposed model reduction method can substantially simplify large train dispatching instances and improve performance in practice. The presented solution method was able to find high-quality feasible solutions for most benchmark instances.

As part of future work, we aim to introduce additional pruning transformations to further reduce the size of problem instances. We also plan to develop a branch-and-bound solution method built on the proposed iterative model reduction, where reductions are applied dynamically after each branching decision in order to narrow the search space.

Acknowledgements The project supported by the Doctoral Excellence Fellowship Programme (DCEP) is funded by the National Research Development and Innovation Fund of the Ministry of Culture and Innovation and the Budapest University of Technology and Economics.

References

1. Cordeau, J.-F., Toth, P., Vigo, D.: A survey of optimization models for train routing and scheduling. Transportation Science **32**(4), 380–404 (1998)
2. D'Ariano, A., Pacciarelli, D., Pranzo, M.: A branch and bound algorithm for scheduling trains in a railway network. European Journal of Operational Research **183**(2), 643–657 (2007)
3. D'Ariano, A., Pranzo, M., Hansen, I.A.: Conflict resolution and train speed coordination for solving real-time timetable perturbations. IEEE Transactions on Intelligent Transportation Systems **8**(2), 208–222 (2007)
4. Mannino, C., Mascis, A.: Optimal real-time traffic control in metro stations. Operations Research **57**(4), 1026–1039 (2009)
5. Corman, F., D'Ariano, A., Pacciarelli, D., Pranzo, M.: Centralized versus distributed systems to reschedule trains in two dispatching areas. Public Transport **2**(3), 219–247 (2010)
6. Agasucci, V., Grani, G., Lamorgese, L.: Solving the train dispatching problem via deep reinforcement learning. Journal of Rail Transport Planning & Management **26**, 100394 (2023)
7. Schälicke, M., Nachtigall, K.: Solving the real-time train dispatching problem by column generation. Transportation Science **59**(3), 587–602 (2025)
8. Schutt, A., Cardellini, M., Dekker, J.J., Harabor, D., Maratea, M., Vallati, M.: Constraint-Based In-Station Train Dispatching. In: Proceedings of the 31st International Conference on Principles and Practice of Constraint Programming (CP 2025), LIPIcs, vol. 340, pp. 33:1–33:24 (2025)
9. Song, Y., Chen, B., Ye, W., Wang, T.: An efficient hybrid method for dispatching trains at complex railway stations. Computers & Industrial Engineering **210**, 111512 (2025)
10. Zwaneveld, P.J., Kroon, L.G., van Hoesel, S.P.M.: Routing trains through a railway station based on a node packing model. European Journal of Operational Research **128**(1), 14–33 (2001).
11. Vansteenwegen, P., Dewilde, T., Burggraeve, S., Cattrysse, D.: An iterative approach for reducing the impact of infrastructure maintenance on the performance of railway systems. European Journal of Operational Research **252**(1), 39–53 (2016).
12. Lamorgese, L., Mannino, C.: An exact decomposition approach for the real-time train dispatching problem. Operations Research **63**(1), 48–64 (2015).
13. Leutwiler, F., Corman, F.: A review of principles and methods to decompose large-scale railway scheduling problems. EURO Journal on Transportation and Logistics **12**, 100107 (2023).
14. Kloster, O., Luteberget, B., Mannino, C., Sartor, G.: DISPLIB: a library of train dispatching problems. arXiv preprint arXiv:2509.12254 (2025). https://arxiv.org/abs/2509.12254
15. Perron, L., Didier, F.: CP-SAT. Version 9.12. Google (2025). https://developers.google.com/optimization/cp/cp_solver/