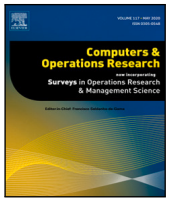
Computers & Operations Research 167 (2024) 106679

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor



A MaxSAT approach for solving a new Dynamic Discretization Discovery model for train rescheduling problems

Anna Livia Croella [a,*], Bjørnar Luteberget [b], Carlo Mannino [b,c], Paolo Ventura [d]

[a] Sapienza University of Rome, Rome, Italy
[b] SINTEF, Oslo, Norway
[c] University of Oslo, Oslo, Norway
[d] Istituto di Analisi dei Sistemi ed Informatica (IASI) del CNR, Rome, Italy

ARTICLE INFO

Keywords:
Train rescheduling
Dynamic discretization discovery
Maximum SATisfiability problem

ABSTRACT

Train scheduling is a critical activity in rail traffic management, both off-line (timetabling) and on-line (dispatching). Time-Indexed formulations for scheduling problems are stronger than other classical formulations, like Big-M. Unfortunately, their size grows usually very large with the size of the scheduling instance, making even the linear relaxation hard to solve. Moreover, the approximation introduced by time discretization can lead to solutions which cannot be realized in practice. Dynamic Discretization Discovery (DDD), recently introduced by Boland et al. (2017) for the continuous-time service network design problem, is a technique to keep at bay the growth of Time-Indexed formulations and their response times and, at the same time, ensures the necessary modelling precision. By exploiting the DDD paradigm, we develop a novel approach to train dispatching and, more in general, to job-shop scheduling. The algorithm implemented represents the first application of a *Maximum SATisfiability* problem approach to the field. In our comparisons on real-life instances of train dispatching, our restricted Time-Indexed formulation solves faster on piece-wise constant objective functions, while the Big-M approach maintains the lead on linear continuous objectives.

1. Introduction

When trains move through a railway network, they occupy a sequence of rail resources, such as track segments, platforms, stations, etc. Roughly speaking, train scheduling amounts to establishing the time in which each train enters and exits the resources encountered on its path (Cacchiani and Toth, 2018). Train scheduling is central in various phases of traffic planning. and in strategic and tactical planning - performed from 5 years to months or even to few hours before the actual movements. In the operational phase, when trains finally move through the railway, the original schedule must be adjusted in real-time to factor in erratic train delays, small or large network disruptions, missing crews, etc. This real-time activity is called *train rescheduling* or *train dispatching* (Lamorgese et al., 2018) - the latter typically also includes the possibility of rail path changes. There may be different objectives, depending on the planning phase. For instance, in the strategic and tactical phase, one is interested in producing timetables maximizing some service requirement (e.g. the number of running trains Lamorgese et al. (2018)), or some robustness measures (Cacchiani and Toth, 2018; Croella et al., 2022; Fischetti and Monaci, 2009). At the operational level, in contrast, one typically wants to minimize some measure of the deviation of the actual schedule from the official timetable (Lamorgese and Mannino, 2015). Also, we mention here some competitions that have recently been set (SBB Swiss Federal Railways, 2019, 2020, The 2023 RAS Problem Solving Competition, 2023) for solving real-world rail scheduling problems and that gave the impulse for developing interesting solution approaches (Abels et al., 2021). However, the considered problems are quite different from the one we tackle here.

From the optimization theory standpoint, train scheduling is a generalization of job-shop (with blocking and no-wait constraints, Mascis and Pacciarelli, 2002) and thus of machine scheduling, and as such it can benefit from a vast body of literature. Even if we limit ourselves to the scientific literature specifically devoted to train scheduling, the number of studies has grown exponentially in the past decades. In our literature discussion, we will focus on a few of the most relevant papers, and refer the interested reader to recent surveys (Cacchiani and Toth, 2018; Fang and Yao, 2015; Lamorgese et al., 2018).

* Corresponding author.
E-mail addresses: annalivia.croella@uniroma1.it (A.L. Croella), bjornar.luteberget@sintef.no (B. Luteberget), carlo.mannino@sintef.no (C. Mannino), paolo.ventura@iasi.cnr.it (P. Ventura).

https://doi.org/10.1016/j.cor.2024.106679
Received 13 June 2023; Received in revised form 16 April 2024; Accepted 22 April 2024
Available online 26 April 2024

There are of course different approaches to train scheduling, but here we are interested in those based on MILP, which are the most adopted in the literature (Lamorgese et al., 2018). The main issue that such approaches must face is how to represent the fact that two trains cannot occupy the same track segment (or other pairs of incompatible railway resources) simultaneously. Then, in any feasible schedule, one of the conflicting trains must use the contended resource before the other train, and this gives rise to a disjunctive constraint on the scheduling variables. Mainly, two classes of MILP models are adopted in the literature on train scheduling (Harrod, 2012): *Big-M* formulations and *Time-Indexed* (TI) formulations (see Queyranne and Schulz, 1994 for theoretical insights). In Big-M formulations, the time a train i enters a given resource r on its path is described by a continuous variable t^{ir}. In order to represent a disjunctive constraint, one must introduce a binary variable and two constraints which contain the notorious Big-M term, which is a very large coefficient. In TI formulations the planning horizon is subdivided in time intervals and a binary variable x_p^{ir} will be 1 if train i enters resource r at time p. The fact that two trains i, j cannot occupy the same resource r implies that, if train i enters a resource r at time p then, depending on the running times of the two trains, train j cannot enter r at some times q. In turn, this translates into packing constraints of the type

$$x_p^{ir} + x_q^{jr} \leq 1. \tag{1}$$

When used in a branch-and-bound approach, Big-M formulations typically return poor bounds, which in turn causes large branching trees. In contrast, TI formulations return better bounds and smaller trees. However, depending on the width of the interval in the discretized horizon, TI formulations have two main drawbacks:

1. *Oversize*. When intervals are small and many, TI formulations typically contain a very large number of variables and constraints. This fact tends to slow down the solution time in each branching node by a factor which is usually (much) larger than the reduction in the tree size. In the few comparative experiments available in the literature, the comparison is by far in favour of the Big-M alternative (Mannino and Mascis, 2009).
2. *Bad approximation.* Larger and fewer intervals result in fewer variables and constraints, but poorer approximation. As a consequence, feasible solutions in the model may prove infeasible in practice on the field, or feasible field schedules may be cut off by the TI model (Harrod, 2011; Boland and Savelsbergh, 2019).

For the above reasons, there are indeed not too many examples in the literature of TI formulations devoted to train scheduling. Most of them are devoted to the off-line version (i.e. timetabling, see Cacchiani and Toth, 2018 for a survey) and only very few to dispatching (such as Bettinelli et al., 2017; Caimi et al., 2012; Meng and Zhou, 2014; Reynolds et al., 2020). In any case, because the complete formulation would be too large to handle, all these papers resort to delayed column generation procedures (Desaulniers et al., 2006). The idea is to start with only a subset of the variables (and constraints), solve this restricted problem and then iteratively identify and add missing variables and/or constraints, and solve again. The process is iterated until some conditions are satisfied, and typically the final instance is much smaller than the full TI instance. An alternative search path is solving the Integer Linear Program (ILP) relaxation of the full TI formulations, both through the exploitation of Lagrangian relaxation models (for example, with alternating direction method of multipliers algorithms - Gao and Niu, 2021; Zhan et al., 2021) and/or its dual problem combined with branch-and-price techniques (see Lusby et al., 2011, 2013). In any case, despite the complexity and ingenuity of the most recent solution algorithms, TI formulations do not seem to perform better than a standard Big-M formulation, solved by a standard commercial solver. For instance, compare the behaviour of a recent TI approach presented in Reynolds et al. (2020) for a medium-size station (Doncaster) with the experiments over a large junction performed in Pellegrini et al. (2014) with a Big-M model at a microscopic network level and an almost 10-year-old technology.

On the other hand, it would be very handy to be able to solve large instances of train scheduling with TI rather than Big-M formulations. Indeed, TI formulations are more flexible than the Big-M counterparts in expressing and manipulating complicated constraints and non-linear objectives as the ones occurring in the train dispatching framework.

Dynamic Discretization Discovery (DDD) is a technique to solve TI formulations recently introduced by Boland et al. (2017) and then further extended in Hewitt (2019), Marshall et al. (2021), Scherr et al. (2020) and Vu et al. (2020). DDD was developed to cope with the size and approximation issues above discussed. The main idea applied in this paper is similar to the bucket discretization described in Dash et al. (2012) and the method presented in Wang and Regan (2002). We consider for each train i and given resource r a partition of the time horizon into intervals $\lambda_1, \ldots, \lambda_n$ of different widths. We hence construct an integer linear program, called Interval Assignment Problem (IAP), with binary variables x (see the forthcoming Section 3.2). Then x_p^{ir} is 1 if and only if train i enters resource r at some time $t^{ir} \in \lambda_p$ during the pth interval (that is, not necessarily at the beginning of the interval, as it is in the full TI formulation). As a consequence, the packing constraint (1) is introduced only if, for every choice of $t^{ir} \in \lambda_p$ and $t^{jr} \in \lambda_q$, trains are in conflict on resource r. Finally, the cost of each variable is chosen in such a way that the value of the optimal solution to the DDD formulation is a lower bound on the cost of the optimal solution to the original problem.

In the DDD approach, intervals and associated variables are generated iteratively, by further subdividing intervals generated in previous iterations. At the end of each iteration k, the optimal solution x_k^* to the current 0,1 formulation is calculated. If we can associate x_k^* with a feasible schedule t_k, and the cost of t_k is not larger than the cost of x_k^*, then we are done. Otherwise, we iterate.

Different DDD approaches differ in the way intervals are iteratively generated, how costs are defined and how feasible solutions to the original problem are constructed from the current TI solution. Although the idea of a dynamic discovery is fairly natural, there are many possible ways to implement it and engineering the algorithm is one of the most delicate phases of its design. In the present work, we present a viable implementation of the DDD for the train scheduling problem. As we will show in more detail in the following sections, such implementation fulfils all three components of the DDD paradigm, as introduced in Boland and Savelsbergh (2019). A crucial component of our solution methodology is the way the IAP is solved at each iteration of the DDD approach. In this work, we transform it into the problem of satisfying a family of logic clauses (SAT problem). It turned out that solving IAPs incrementally by means of an open-source SAT solver is dramatically more efficient than using a state-of-the-art MILP solver. Note that similar behaviour was observed in the work of Leutwiler and Corman (2022) and Matos et al. (2021). In the former, the authors compared a MILP and a SAT version of specific binary programs, reporting significant improvements with the SAT formulation. The latter study combines SAT and machine learning approaches to address periodic timetabling problems, outperforming state-of-the-art algorithms, including MILP and heuristics. Formulations with disjunctive precedence constraints (such as Big-M formulations) make use of continuous variables, and cannot be directly translated into a SAT problem (in Leutwiler and Corman, 2022 this problem is solved by using a Satisfiability Modulo Theories (SMT) approach). However, with a TI formulation the MILP contains only binary variables and all the constraints can be directly translated into SAT. The objective function can also be handled in a *MaxSAT* problem — the optimization version of the SAT problem.

MaxSAT solvers implemented on top of standard SAT solvers have been very successful lately (Bacchus et al., 2019). In this paper, we successfully used the RC2 algorithm (Ignatiev et al., 2019) to solve DDD formulations of the train scheduling problem.

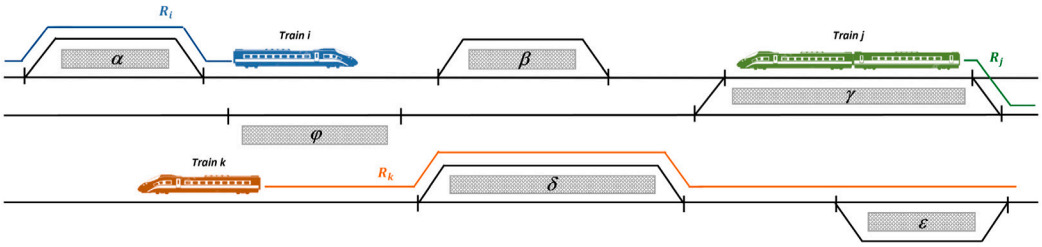


Fig. 1. Schematic representation of a railway infrastructure. Stations are depicted as filled rectangles.

A preliminary version of the ideas developed in this work was presented in Croella et al. (2021) and in Croella (2022).

In the end, by our version of the DDD approach plus the transformation into a SAT problem of the associated binary programs, we were able to obtain speed-ups over our efficient implementation of the Big-M formulation. In particular, this is true for linear rounded or stepwise objective functions. On the other hand, for the case of linear objective functions, the Big-M formulation still has, on average, better computational performances.

2. Time-indexed formulation for the train re-scheduling problem

The main purpose of this paper is to show how the DDD mechanism can be exploited in order to make TI formulations competitive with the classic Big-M formulation for train scheduling. To this end, in our developments and comparisons, we focus on a basic version of the problem, which corresponds to finding a plan for a *macroscopic* approximation of the rail network (Hansen and Pachl, 2014) in which stations are collapsed into simple nodes and routing is omitted. Note that this is also a usual practice in manual train scheduling, and also the master problem in advanced decomposition approaches (see, e.g., Bešinović et al., 2016; Lamorgese et al., 2016, 2017; Schlechte et al., 2011). Accordingly, in our experiments, we will make use of macroscopic real-life instances.

We look at the operational version of the train scheduling problem, also called *train dispatching* or *train rescheduling*, and, for simplicity, we do not include the possibility of rerouting. Note that the model can be extended to cope with multiple routes, for instance as in Leutwiler and Corman (2022) and Mannino and Nakkerud (2023). In this version of the problem, we are given a *reference timetable* (i.e. the timetable which is published either for passengers or for the railway personnel), and the position of the trains at the current time. Note that trains may be delayed with respect to the reference timetable. Additionally, two trains cannot occupy the same section of track simultaneously (i.e. have a *conflict*). Depending on the subsequent scheduling decisions, particularly those involving meet or pass locations between trains, some trains may experience a reduction in delay at their destination, while others may encounter an increase. Due to the real-time nature of the problem, a solution must be computed within a very short timeframe, typically not exceeding 10 s. The goal is to find a conflict-free schedule for the next hours for all trains, minimizing the sum measure of the delays. In real-life railway systems, such a solution is typically formulated by human dispatchers, and imposed on trains by scheduling signals and switch changes, and communicated to drivers. Regardless, once a solution is implemented, trains will continue their movements, potentially deviating from the intended schedule. Consequently, a new (snapshot) instance is generated and resolved every 10 s.

We next introduce our main modelling choices and formalism.

2.1. Problem definition

Networks, trains and rail paths. In Fig. 1 we show a schematic section of a railway infrastructure which comprises a number of sub-networks, called lines. Each line is schematized as an alternating sequence of stations and interconnecting tracks. Stations are special groups of tracks where trains can stop and perform some activities (such as embarking/disembarking passengers, refuelling, etc.). Tracks are further subdivided into (one or more) *tracks segments* or *block sections*, that can accommodate at most a train at a time. Tracks are either *bidirectional*, i.e. they can be traversed from in both directions or *unidirectional*. We let R be the set of all track segments.

In the example, train i travels from west to east reaching station β, while train j is departing from station γ running west. Note that j can either proceed on its current line and reach station β and station α, or change to the lower line and proceed to φ.

If train j continues on the current track, at some point, it will encounter train i. Since train i and j are running in opposite directions, they are called *crossing trains*. Trains running in the same directions and on the same sequence of tracks are called *trailing trains*.

In the Train Re-scheduling Problem (TRP) addressed in this paper, we are given a set of trains I and we assume their path across the network is given.

Definition 1. A *rail path* is the sequence of contiguous track segments of the line traversed by a train from its origin to its destination. The direction of the train depends on the ordering of its track segments, either west-bound or east-bound.

For each train $i \in I$, we let R_i be the ordered set of track segments of its rail path. We therefore have that $R = \cup_{i \in I} R_i$. Moreover, we assume that each track segment is traversed only once. Hence, for $r, q \in R_i$, we use the notation $r \prec_i q$ if r precedes q in the rail path of i. For $r \in R_i$, we also let $l_i^r \in \mathbb{Q}_+$ be the minimum amount of time i needs to traverse track segment r (*running time* of i on r). For the sake of clarity of the exposition and without loss of generality, in the following, we assume l^r to be integer. Moreover, if no confusion arises, in the following we use l^r for l_i^r. Trains may also stop in any station r, and we include the value of this given *wait* or *dwell* time in the amount l^r.

Next, for each train $i \in I$ and each track segment $r \in R_i$ of its rail path, we let $\underline{t}^{ir}$ denote the earliest time i can enter a track segment. This parameter is either the one specified in the reference timetable, suitably augmented in its first track segment when i is delayed, or it is derived by using the minimum running times on the track segments.

Schedule and conflicts. A *train schedule* is a vector $t^i \in \mathbb{Q}^{|R_i|}$, where component t^{ir} denotes the time when i enters track segment $r \in R_i$. The *(full) schedule* will be thus the vector $\tau = \{t^{ir} \mid i \in I, r \in R_i\}$. For physical or safety reasons, certain track segments cannot be occupied by two trains simultaneously. In particular, for each pair of trains i, j let us denote by $\mathcal{D}_{ij} \subseteq R_i \times R_j$ the set of rail paths pairs $(r \in R_i, q \in R_j)$ such that either i enters r before j enters q or j enters q before i enters r. Let l_{ij}^{rq} be the minimum amount of time that, for safety and business rules, train i needs to wait for occupying the track segment r after train j used track segment q (again, if no confusion arises, we use l^{rq} for l_{ij}^{rq}). Hence, we either have $t^{jq} \geq t^{ir} + l^{rq}$ or $t^{ir} \geq t^{jq} + l^{qr}$, respectively.

Definition 2. A schedule $\tau = \{t^{ir} \mid i \in I, r \in R_i\}$ is feasible if it satisfies the following constraints:

(i) (*lower bound constraints*) $t^{ir} \geq \underline{t}^{ir}$, for each $i \in I$, and $r \in R_i$;

(ii) (*train-rail path precedences*) $t^{iq} \geq t^{ir} + l^r$, for each $i \in I$, and each distinct pair $r, q \in R_i$ with $r \prec_i q$;
(iii) (*disjunctive precedences*) either $t^{jq} \geq t^{ir} + l^{rq}$ **or** $t^{ir} \geq t^{jq} + l^{qr}$, for each distinct $i, j \in I$ and each $(r, q) \in \mathcal{D}_{ij}$.

If a schedule violates (iii) for a $(r, q) \in \mathcal{D}_{ij}$, then we say that it contains a *conflict* (associated with (r, q)); otherwise the schedule is said to be *conflict-free*. We assume that any movement (i.e. a train entering a track segment) must happen before time M, where M is a suitably large integer number, i.e., $t^{ir} \ll M$ for each $i \in I$, and $r \in R_i$.

Objective function. We say that a schedule is optimal if it minimizes the delays of trains along their path. We consider separable cost functions. For each $i \in I$, $r \in R_i$, the delay of train i entering track segment r at time t is defined as:

$$d^{ir}(t) = \max(0, t - \underline{t}^{ir}).$$

Then, the cost function is defined as $\Sigma_{i \in I} \Sigma_{r \in R_i} c^{ir}(t^{ir})$, and we consider three cases:

1. **Linear continuous**: $c^{ir}(t) = d^{ir}(t)$. In the Big-M formulation, this objective is implemented by introducing one continuous variable and one linear inequality for each t^{ir}.
2. **Linear rounded**: $c^{ir}(t) = \lfloor d^{ir}(t)/Q \rfloor$, where Q is a constant. We use $Q = 180$, i.e., 3 min. This objective is similar to the stepwise function described below, except that it does not have a maximum cost.
   In the Big-M formulation, this objective is implemented by introducing one integer variable and one linear constraint for each t^{ir}.
3. **Stepwise**: Each train has its finite sequence of *steps* valid for specific delay ranges. For example (see also Fig. 2):

$$c^{ir}(t) = \begin{cases} 3 & 360 < d^{ir}(t) \\ 2 & 180 < d^{ir}(t) \leq 360 \\ 1 & 0 < d^{ir}(t) \leq 180 \\ 0 & d^{ir}(t) = 0 \end{cases}$$

   In the Big-M formulation, this objective is implemented through the introduction of one binary variable.
   The use of stepwise objective functions is inspired by the official performance indicators that railway administrations use to report their punctuality, including the Norwegian railways. This can be considered to be the high-level goal of railway dispatching. Typically, small deviations are not counted in this performance indicator, and instead, punctuality is defined as the percentage of trains that were less than e.g. 5 min delayed.
   It is also interesting to note that, since this cost function has a maximum cost per train if a train has exceeded the highest delay threshold, then any additional delay has no additional cost, and the train can hold at the station as long as necessary to reduce other trains delays. This models the real-world dispatcher behaviour of making a train cancelled in case of large traffic disruptions. In practical use, finding an optimal solution where a train has exceeded the highest delay threshold can be interpreted as a suggestion to cancel that train.

2.2. A full TI formulation

For ease of reference, we now present a full TI formulation for the TRP. For each $i \in I$ and $r \in R_i$, we call *feasibility interval* the time interval $[\underline{t}^{ir}, M)$ in which train i can enter track segment r, i.e. its planning horizon. Let w be the time-intervals width, we divide each feasibility interval into sub-intervals of the type:

$$[p,\ p + w) \text{ with } p \in \Pi^{ir} = \{\underline{t}^{ir}, \underline{t}^{ir} + w, \underline{t}^{ir} + 2w, \dots, \underline{t}^{ir} + \left\lfloor \frac{M - \underline{t}^{ir}}{w} \right\rfloor \cdot w\}.$$

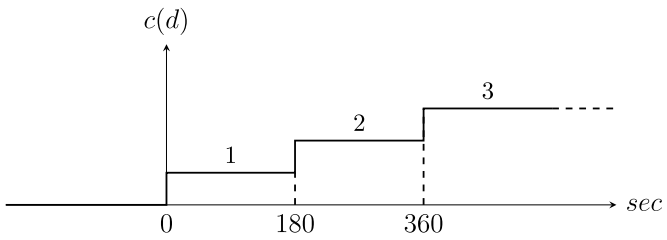


Fig. 2. Example of a stepwise linear convex function considered to cost the delay (in seconds) of a generic train.

We then let $\Pi = \{\Pi^{ir} : i \in I, r \in R_i\}$ and define the following set of binary variables:

$$x_p^{ir} = \begin{cases} 1 & \text{if train } i \text{ enters } r \text{ at time } p \\ 0 & \text{otherwise} \end{cases} \qquad i \in I,\ r \in R_i,\ p \in \Pi^{ir}.$$

The full TI formulation is hence given as follows:

$$\begin{aligned} \min \quad & \textstyle\sum_{i \in I} \sum_{r \in R_i} \sum_{p \in \Pi^{ir}} c^{ir}(p) \cdot x_p^{ir} \\ s.t. \\ (1) \quad & \textstyle\sum_{p \in \Pi^{ir}} x_p^{ir} = 1, && i \in I,\ r \in R_i \\ (2) \quad & x_p^{ir} + x_{p'}^{jq} \leq 1, && p \in \Pi^{ir} \text{ incompatible with } p' \in \Pi^{jq} \\ & && \Pi^{ir}, \Pi^{jq} \in \Pi \text{ and } (i, r, p) \neq (j, q, p') \\ & x_p^{ir} \in \{0, 1\} && i \in I,\ r \in R_i,\ p \in \Pi^{ir}. \end{aligned} \tag{TI}$$

The first group of constraints states that x defines a (full) train schedule, whereas the second imposes schedule feasibility. In particular, looking at Definition 2, there is a packing constraint of type (2)

- between two variables x_p^{ir} and $x_{p'}^{iq}$, with $r \prec_i q$, if and only if $p' < p + l^r$ (they violate condition 2.*ii*);
- between two variables x_p^{ir} and $x_{p'}^{jq}$, with $(r, q) \in \mathcal{D}_{ij}$, if and only if $p' < p + l^{rq}$ and $p < p' + l^{qr}$ (they violate condition 2.*iii*).

Note that condition i of Definition 2 is trivially satisfied by taking a planning horizon equal to the feasibility interval $[\underline{t}^{ir}, M)$. We highlight that TI models allow to express complicated (non-linear) objectives: any c function introduced in the previous section can be translated into the sum of the costs realized by each chosen interval.

3. The dynamic discretization discovery method

In this section we describe the DDD paradigm (according to Boland and Savelsbergh, 2019) and how we re-interpret it in the context of the TRP. The paradigm includes:

- the construction of a sequence of (small) approximated models of the original scheduling problem, that are easier to solve than the full problem. As anticipated in the introduction, the models $D_1, D_2, \dots$ are TI formulations (for job-shop scheduling), where the discretization intervals have different widths. The value of an optimal solution x_k^* to the kth model in the sequence provides a lower bound on the optimal solution value to the original problem.
- a function Φ which associates with solution x_k a (possibly infeasible) schedule for the original problem. If the schedule $\Phi(x_k^*)$ is feasible for the original problem, then it is optimal.
- a heuristic mechanism which "repairs" an infeasible schedule for the original problem and returns a feasible schedule which provides an upper bound on the optimal solution value to the original problem (see, e.g. Vu et al., 2022).
- a dynamic discovery mechanism that allows, when the schedule is not feasible, to refine the previous model by further discretizing time intervals.

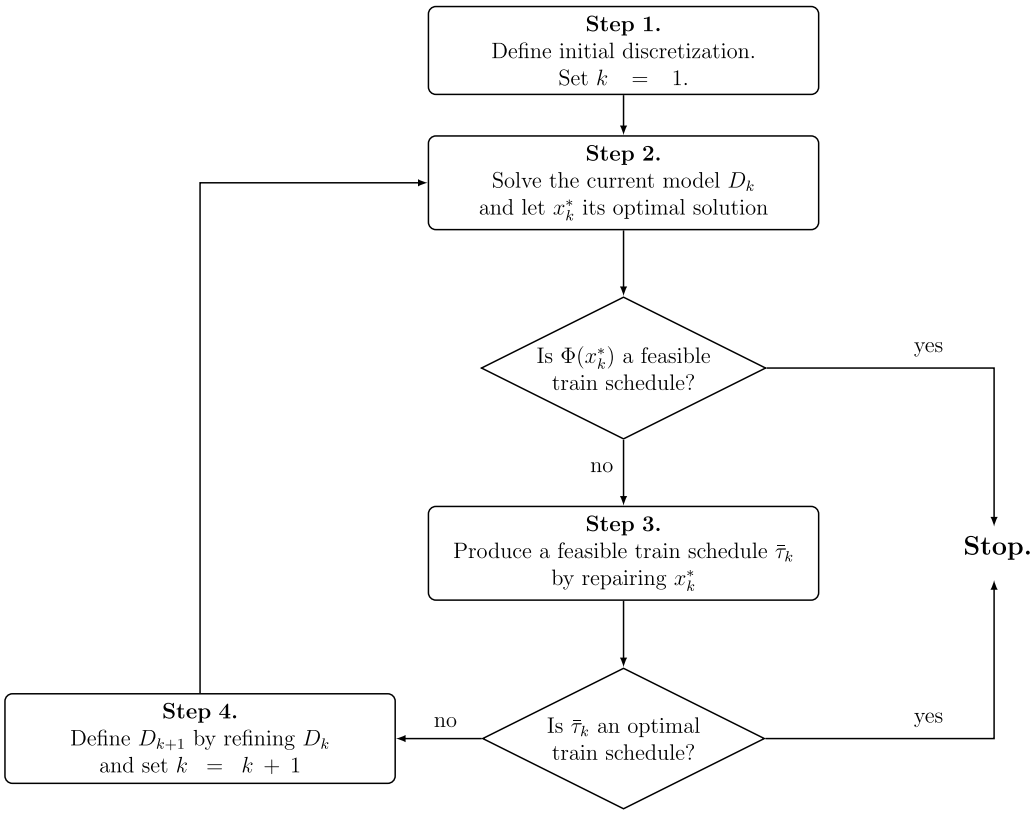


Fig. 3. Generalized flowchart of the DDD paradigm.

A schematic representation of the DDD paradigm is given in Fig. 3. Observe that the optimal solution x_k^* of the current partial model D_k provides a lower bound for the original problem. If the associated schedule $\Phi(x_k^*)$ is not feasible for the original TRP, at step 3 we apply a heuristic mechanism for "repairing" the solution, i.e. for generating a feasible solution $\bar{\tau}_k$ by exploiting $\Phi(x_k^*)$. In this way, an upper bound on the optimal value of the solution is also provided. We then refine the time discretization at step 4, making x_k^* infeasible for the new partial model D_{k+1}. The process is halted when the optimal solution is found or, possibly, if the optimality gap is smaller than a given threshold. We remark that several factors, ranging from the way the new refinements are generated to the efficiency of the algorithm used to solve the partial models, affect the performance of the DDD paradigm.

We are now ready to introduce our approximated model, namely the *Interval Assignment Problem* (IAP), whose optimal solution defines a lower bound for the optimal solution of the TRP.

3.1. The Λ-Interval Assignment Problem

Given, for each $i \in I$ and $r \in R_i$, the *feasibility interval* $[\underline{t}^{ir}, M)$, let $\Lambda^{ir} = \{\lambda_1^{ir}, \lambda_2^{ir}, \dots, \lambda_{n^{ir}}^{ir}\}$ be a partition of the feasibility interval such that $\lambda_p^{ir} = [h_p^{ir}, h_{p+1}^{ir})$, for $1 \le p < n^{ir}$, $h_1^{ir} = \underline{t}^{ir}$, and $\lambda_{n^{ir}}^{ir} = [h_{n^{ir}}^{ir}, M)$. Also, let $\bar{c}^{ir}(\lambda_p^{ir}) = c^{ir}(h_p^{ir})$ be the cost associated with the interval λ_p^{ir}. That is, the cost of the interval is the cost of train i entering track segment r at the beginning of the interval. We finally let $\Lambda = \{\Lambda^{ir} : i \in I, r \in R\}$ be the set of all partitions.

Note first that, for $\lambda \in \Lambda^{ir}$, $t^{ir} \in \lambda$ implies that t^{ir} satisfies the lower bound condition (2.(i)). Now, let $i \in I$ and $r, q \in R_i$ be distinct track segments, with $r \prec_i q$.

Definition 3 (*Rail Path Incompatible Intervals*)**.** Two distinct intervals $\lambda_p^{ir} \in \Lambda^{ir}$ and $\lambda_{p'}^{iq} \in \Lambda^{iq}$ are *rail path incompatible* if condition (2.(ii)) is violated for every $t^{ir} \in \lambda_p^{ir}$ and every $t^{iq} \in \lambda_{p'}^{iq}$. That is, assuming $r \prec_i q$, for every $t^{ir} \in \lambda_p^{ir}$ and every $t^{iq} \in \lambda_{p'}^{iq}$, we have $t^{iq} < t^{ir} + l^r$.

Corollary 1. *The two intervals $\lambda_p^{ir} = [h_p^{ir}, h_{p+1}^{ir})$ and $\lambda_{p'}^{iq} = [h_{p'}^{iq}, h_{p'+1}^{iq})$, with $r, q \in R_i$ and $r \prec_i q$, are rail path incompatible if and only if $h_p^{ir} + l^r > h_{p'+1}^{iq}$.*

In other words, if there is no way for train i to enter r during interval λ_p^{ir} and to enter q during time interval $\lambda_{p'}^{iq}$.

Definition 4 (*Conflict Incompatible Intervals*)**.** Let $i, j \in I$ be two distinct trains, and let $r \in R_i$, and $q \in R_j$. We say that the intervals λ_p^{ir} and $\lambda_{p'}^{jq}$ are *conflict incompatible* if condition (2.(iii)) is violated for every $t^{ir} \in \lambda_p^{ir}$ and every $t^{jq} \in \lambda_{p'}^{jq}$. That is, for every $t^{ir} \in \lambda_p^{ir}$ and every $t^{jq} \in \lambda_{p'}^{jq}$, we have both $t^{ir} < t^{jq} + l^{qr}$ and $t^{jq} < t^{ir} + l^{rq}$.

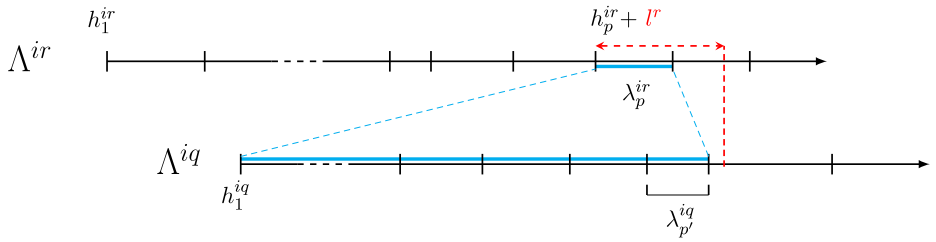
Namely, train i cannot enter track r in interval λ_p^{ir} if train j is entering track q in time interval $\lambda_{p'}^{jq}$.

Corollary 2. *Two intervals $\lambda_p^{ir} = [h_p^{ir}, h_{p+1}^{ir})$ and $\lambda_{p'}^{jq} = [h_{p'}^{jq}, h_{p'+1}^{jq})$, with $i \neq j$, $r \in R_i$ and $q \in R_j$, are conflict incompatible if and only if $h_{p'}^{jq} + l^{qr} > h_{p+1}^{ir}$* **and** *$h_p^{ir} + l^{rq} > h_{p'+1}^{jq}$.*

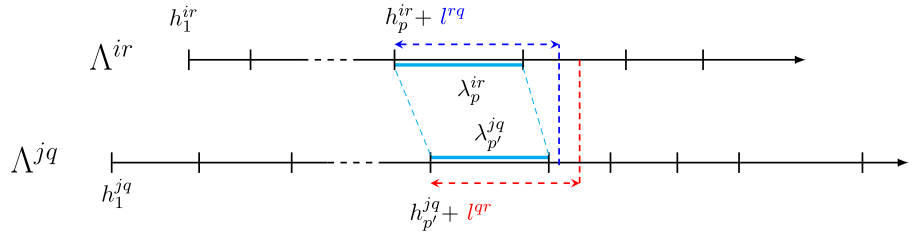
Fig. 4 shows an example for the rail path incompatibility and one for the conflict incompatibility. For both (rail path and conflict) incompatibilities the following holds.

Remark 1. Let λ, λ' be two incompatible intervals and let $\bar{\lambda}, \bar{\lambda}'$ be two intervals such that $\bar{\lambda} \subseteq \lambda$ and $\bar{\lambda}' \subseteq \lambda'$. Then $\bar{\lambda}$ and $\bar{\lambda}'$ are incompatible intervals.

Note the difference in the definitions of interval incompatibility: in the full time-indexed formulation (TI), two intervals are incompatible exactly if the start times of the intervals are incompatible (i.e. in



(a) Example of *rail path incompatibility*. We consider a train i and two of its track segments, r, q, with $r \prec_i q$. Intervals $\lambda_p^{ir} \in \Lambda^{ir}$ and $\lambda_{p'}^{iq} \in \Lambda^{iq}$ are (rail path) incompatible, since $h_p^{ir} + l^r > h_{p'+1}^{iq}$.



(b) Example of *conflict incompatibility*. We consider two distinct trains i and j traversing respectively track segments r and q. Intervals $\lambda_p^{ir} \in \Lambda^{ir}$ and $\lambda_{p'}^{jq} \in \Lambda^{jq}$ are (conflict) incompatible, since $h_{p'}^{jq} + l^{qr} > h_{p+1}^{ir}$ and $h_p^{ir} + l^{rq} > h_{p'+1}^{jq}$.

Fig. 4. Example of intervals that are *rail path incompatible* 4(a) and *conflict incompatible* 4(b).

violation of the constraints of Definition 2). In contrast, in the IAP's definition, two intervals are incompatible if all pairs of time points in the Cartesian product of the intervals are incompatible (wrt. Definition 2).

In an attempt to construct a feasible schedule τ, we will first find a set of partitions Λ, then assign an interval $\lambda^{ir} \in \Lambda^{ir}$ for each $i \in I$ and $r \in R_i$, and finally choose $t^{ir} \in \lambda^{ir}$. This motivates the following definition:

Definition 5 (Λ-*Interval Assignment*)**.** Let Ω be the set of all pairs (i, r) with $i \in I$ and $r \in R_i$. A Λ-*interval assignment* S is a function $S : \Omega \to \Lambda$ that assigns each couple $(i, r) \in \Omega$ an interval $S(i, r) \in \Lambda^{ir}$. Moreover, we say that S is *feasible* if $S(i, r)$ and $S(j, q)$ are not incompatible according to Definitions 3 and 4, for each (i, r) and $(j, q) \in \Omega$.

Given a set of partitions Λ, we define the IAP as the problem of finding a Λ-*feasible assignment* S of minimum cost $\bar{c}(S) = \sum_{i\in I} \sum_{r\in R_i} \sum_{\lambda_p^{ir}\in\Lambda^{ir}\cap S} \bar{c}^{ir}(\lambda_p^{ir})$.

We can indeed formally prove the following theorem:

Theorem 1. *If the TRP admits an optimal solution* $\bar{\tau} = \{\bar{t}^{ir} \mid i \in I, r \in R_i\}$*, then the IAP admits an optimal solution* S^**, and* $\bar{c}(S^*) \le c(\bar{\tau})$*, where* $c(\bar{\tau}) = \sum_{i\in I} \sum_{r\in R_i} c^{ir}(\bar{t}^{ir})$.

Proof. Let $\tilde{\tau} = \{\tilde{t}^{ir} \mid i \in I, r \in R_i\}$ be a feasible schedule, and let Ψ_Λ be the function which associates the unique interval $\Psi_\Lambda(\tilde{t}^{ir}) = \lambda_p^{ir} \in \Lambda^{ir}$ such that $\tilde{t}^{ir} \in \lambda_p^{ir}$. Note that the function is well-defined, since $\tilde{t}^{ir} \in [\underline{t}^{ir}, M]$ and Λ^{ir} is a partition of $[\underline{t}^{ir}, M]$. We extend the notation by denoting with $\Psi_\Lambda(\tilde{\tau})$ the complete set of intervals $\{\Psi_\Lambda(\tilde{t}^{ir}) \mid i \in I, r \in R_i\}$. We prove that $\Psi_\Lambda(\tilde{\tau})$ is Λ-feasible. Suppose not, then there exist two distinct intervals $\lambda_p^{ir}, \lambda_{p'}^{jq}$ which are incompatible.

If $i = j$ then the intervals are rail path-incompatible. Assume $r \prec_i q$. Then we must have that $t^{iq} < t^{ir} + l^r$ for every $t^{ir} \in \lambda_p^{ir}$ and every $t^{iq} \in \lambda_{p'}^{iq}$, a contradiction since $\tilde{\tau}$ is feasible and thus $\tilde{t}^{iq} \ge \tilde{t}^{ir} + l^r$.

If $i \neq j$ then the intervals are conflict-incompatible. We arrive at a contradiction again by using a similar argument as above.

So, $\Psi_\Lambda(\bar{\tau})$ is Λ-feasible. Now, since c is non-decreasing, we have that:

$$\bar{c}(\Psi_\Lambda(\bar{\tau})) := \sum_{i\in I} \sum_{r\in R_i} \bar{c}^{ir}(\Psi_\Lambda(\bar{t}^{ir})) \le \sum_{i\in I} \sum_{r\in R_i} c(\bar{t}^{ir}) = c^{ir}(\bar{\tau}).$$

□

Let Φ be the function that assigns to each time interval $[h, h^+)$ the time instant $t = h$. Moreover, we extend such a definition to a set of intervals $S \subseteq \Lambda$, so to let $\tau = \Phi(S)$ be the schedule $\{t^{ir} = h_p^{ir} \mid \lambda_p^{ir} = [h_p^{ir}, h_{p+1}^{ir}) \in S\}$. Therefore, because of Theorem 1, if S^* is a Λ-feasible set of intervals of minimum cost, with respect to every given partition Λ, and $\tau^* = \Phi(S^*)$ is feasible, then τ^* is also optimal for the TRP.

3.2. A 0,1-LP for the Interval Assignment Problem

We now present a 0,1 Linear Program (0,1-LP) for the IAP. We are given the set of partitions $\Lambda = \{\Lambda^{ir} : i \in I, r \in R_i\}$, and we want to find a complete interval assignment S of non-incompatible intervals of minimum cost $\bar{c}(S)$. To this end, we introduce the following set of binary variables:

$$x_p^{ir} = \begin{cases} 1 & \text{if interval } \lambda_p^{ir} \in S \\ 0 & \text{otherwise} \end{cases} \qquad i \in I,\ r \in R_i,\ \lambda_p^{ir} \in \Lambda^{ir}\ .$$

Hence, the IAP can be formulated as follows:

$$\begin{array}{lll} \min & \sum_{i\in I} \sum_{r\in R_i} \sum_{\lambda_p^{ir}\in\Lambda^{ir}} \bar{c}^{ir}(\lambda_p^{ir}) \cdot x_p^{ir} & \\ s.t. & & \\ (1) & \sum_{\lambda_p^{ir}\in\Lambda^{ir}} x_p^{ir} = 1, & i \in I,\ r \in R_i \\ (2) & x_p^{ir} + x_{p'}^{jq} \le 1, & \lambda_p^{ir} \in \Lambda^{ir} \text{ incompatible with } \lambda_{p'}^{jq} \in \Lambda^{jq} \\ & & \Lambda^{ir}, \Lambda^{jq} \in \Lambda \text{ and } (i, r, p) \neq (j, q, p') \\ & x_p^{ir} \in \{0, 1\} & i \in I,\ r \in R_i,\ \lambda_p^{ir} \in \Lambda^{ir}. \end{array} \tag{IAP}$$

Constraints (1) ensure that x is the incidence vector of a complete interval assignment S, whereas constraints (2) guarantee that the intervals in S are mutually non-incompatible. Also observe that if all the intervals λ_p^{ir} have unit length, then the formulation (IAP) reduced to a full TI formulation for the TRP with interval width $w = 1$ (see formulation (TI) given in Section 2). In turn, if, as assumed, all involved constants are integers (e.g. number of seconds), then the full TI formulation is exact.

Therefore, the following holds:

Observation 1. *If the set of partitions Λ is such that $h_{p+1}^{ir} = h_p^{ir} + 1$, for all $i \in I$, $r \in R_i$, $p \in \{1, \dots, n^{ir} - 1\}$, then exactly one of the following claims holds: (i) IAP and TRP are both infeasible; (ii) the value of an optimal solution of the IAP equals the value of an optimal solution of the corresponding TRP.*

3.3. A basic MaxSAT formulation for the Interval Assignment Problem

Although ILP (and MILP) formulations are commonly used for solving train scheduling problems, the IAP formulation uses only binary variables, which opens up the possibility of translating the problem into a Boolean satisfiability (SAT) problem.

A SAT problem is usually formulated in logical terms, where a Boolean variable y can take values *true* or *false*. A literal ℓ is a variable y or its negation $\bar{y}$. A clause $c = \ell_1 \vee \ell_2 \vee \dots \vee \ell_n$ contains one or more distinct variables and is satisfied if it has at least one literal assigned to *true*. A conjunctive normal form (CNF) formula $\phi = c_1 \wedge c_2 \wedge \dots \wedge c_m$ is a conjunction of clauses. Given a CNF formula, the SAT asks whether there exists an assignment to the variables that satisfies all the clauses. The optimization version of the SAT problem is the so-called *(partial) weighted MaxSAT problem*: given a CNF, we have that only a (possibly empty) subset of its clauses (*hard* clauses) must be satisfied, whereas each of the remaining (*soft*) clauses is given a weight. The goal is to find an assignment of values to the Boolean variables such that all hard clauses are satisfied and the sum of the weights of the satisfied soft clauses is maximized. Exact MaxSAT solvers have made large advances in the last decade, and are applied to industrial problems in scheduling, timetabling, decision trees, and computer hardware and software verification (Bacchus et al., 2019; Li and Manyà, 2021).

It is straightforward to see that MILP problems are a generalization of SAT problems, but for readers who are unfamiliar with logic notation, we show here a simple transformation of a SAT problem into a MILP feasibility problem. Namely, we introduce a binary variable x_i for each Boolean variable y_i. Assigning value 1 (0) to x_i corresponds to assigning value True (False) to y_i. Next, for each clause $c = y_1 \vee y_2 \vee \dots \vee y_k \vee \bar{y}_{k+1} \vee \dots \vee \bar{y}_n$, where the first k literals are positive variables whereas the remaining are negated, we introduce the constraint

$$x_1 + \dots + x_k + (1 - x_{k+1}) + \dots + (1 - x_n) \geq 1 \tag{2}$$

Then the CNF formula is satisfiable if and only if there exists a binary vector x which satisfies all constraints (2).

The opposite direction, converting MILPs to SAT problems, does not have a corresponding simple transformation (though several special cases have been studied extensively, see Roussel and Manquinho, 2021). In the following, we show that the IAP can be converted into SAT. Indeed, we will show how to formulate the IAP as a MaxSAT problem. Consider the 0,1-LP (IAP) restated in a more compact form:

$$\begin{array}{lll} \min & \sum_{i \in V} w_i x_i & \\ s.t. & & \\ (i) & \sum_{i \in Q} x_i = 1, & Q \in \mathcal{Q} \\ (ii) & x_i + x_j \leq 1, & \{i,j\} \in E \\ & x_i \in \{0,1\} & i \in V. \end{array} \qquad \text{(IAP-compact)}$$

There are two types of constraints: partitioning constraints (i), and (edge) packing constraints (ii). Note that $\mathcal{Q}$ is a set of subsets of V, and E is a set of unordered pairs of V.

The 0,1-LP (IAP-compact) can be reduced to (partial) MaxSAT by constructing an equivalent CNF formula. In particular, each binary variable corresponds to a Boolean variable, each term in the objective function corresponds to a soft clause, and each constraint corresponds to a hard clause or a conjunction of hard clauses (see Li and Manyà, 2021).

First, for $i \in V$, we associate a Boolean variable y_i with each binary variable x_i, and its negation $\bar{y}_i$ with $1 - x_i$.

- Each term $w_i x_i$ in the objective function of (IAP-compact) is represented by a soft clause $c = y_i$ (a unit disjunction) with weight w_i.
- Each edge packing constraint (ii) is first represented as the equivalent edge covering constraint: $(1 - x_i) + (1 - x_j) \geq 1$, to which corresponds the clause

$$\bar{y}_i \vee \bar{y}_j. \tag{3}$$

- Each partitioning constraint (i) is first transformed into the conjunction of a covering constraint $\sum_{i \in Q} x_i \geq 1$ and a packing constraint $\sum_{i \in Q} x_i \leq 1$. The covering constraint is immediately reduced into the clause $\bigvee_{i \in Q} y_i$. As for the packing constraint, we first replace it with an equivalent conjunction of a set of edge packing constraints: $x_i + x_j \leq 1$, for $i, j \in Q, i \neq j$. Then, as for the constraint (ii), each edge packing constraint is transformed into the clause (3).

There are, in general, multiple ways of converting various types of constraints into clause form, with different trade-offs (see Björk, 2011). Especially for the at-most-one constraint and its generalization, at-most-k, many different algorithms and techniques have been studied. The pairwise encoding (3) is presented here for simplicity, but any of the alternative at-most-one encodings can be used (see Prestwich, 2021).

3.4. A MaxSAT reformulation using lower bound variables

It is well known that reformulations of SAT and MaxSAT problems can have a huge impact on solver performance (see Björk, 2011). Even different formulations that are deemed equally strong in the linear programming sense (such as those corresponding to an affine transformation) may exhibit widely different performance characteristics. Motivated by this observation, we reformulated the MaxSAT encoding of IAP. In this reformulation, variables y_p^{ir}, corresponding to intervals $\lambda_p^{ir} \in \Lambda^{ir}$, are true not only for the selected interval but also for all preceding intervals in Λ^{ir}. This implies that interval λ_p^{ir} is selected if $y_p^{ir} = 1$ and $y_{p+1}^{ir} = 0$. The constraints for the problem are then as follows:

- To ensure that there is exactly one interval $\lambda_p^{ir} \in \Lambda^{ir}$ s.t. $y_p^{ir} = 1$ and $y_{p+1}^{ir} = 0$, we need for each p and each Λ^{ir},

$$\overline{y_{p+1}^{ir}} \vee y_p^{ir} \tag{4}$$

- To exclude incompatible intervals λ_p^{ir} and $\lambda_{p'}^{jq}$, we need

$$\overline{y_p^{ir}} \vee y_{p+1}^{ir} \vee \overline{y_{p'}^{jq}} \vee y_{p'+1}^{jq} \tag{5}$$

Note that for rail path incompatibilities, this can be simplified to $\overline{y_p^{ir}} \vee y_{p'}^{iq}$.

This reformulation offers the advantage that iteratively subdividing intervals, as described in the algorithmic framework below, can be done without invalidating any constraints. Specifically, we start from the partition

$$\Lambda^{ir} = \left\{ \lambda_1^{ir}, \dots, \lambda_p^{ir}, \dots, \lambda_n^{ir} \right\}$$

and we then subdivide interval p into two new intervals in positions p and $p+1$ to get

$$\tilde{\Lambda}^{ir} = \left\{ \tilde{\lambda}_1^{ir}, \dots, \tilde{\lambda}_p^{ir}, \tilde{\lambda}_{p+1}^{ir}, \dots, \tilde{\lambda}_{n+1}^{ir} \right\}$$

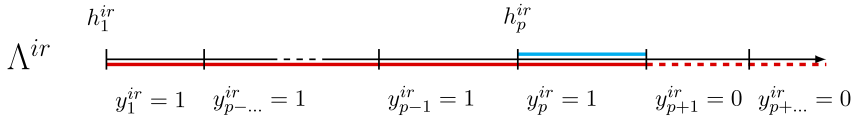


Fig. 5. Lower bound variables representation for selecting interval λ_p^{ir}. All preceding intervals in Λ^{ir} are set to 1 and all subsequent intervals are set to 0.

Now, Eq. (4) for λ_{p+1}^{ir} becomes, in the new partition $\tilde{\Lambda}^{ir}$,

$\overline{\tilde{y}_{p+2}^{ir}} \vee \tilde{y}_p^{ir}$,

and Eq. (5) for λ_p^{ir} and $\lambda_{p'}^{jq}$ becomes

$\overline{\tilde{y}_p^{ir}} \vee \tilde{y}_{p+2}^{ir} \vee \overline{\tilde{y}_{p'}^{jq}} \vee \tilde{y}_{p'+1}^{jq}$.

These equations remain valid within the new partition $\tilde{\Lambda}^{ir}$ but are redundant. However, keeping these redundant constraints can be beneficial when solving a sequence of IAP problems, as detailed in Section 4 below. This approach ensures that only the task of *adding* constraints is required (i.e., we do not need to remove or modify any constraints) when moving from one iteration to the next. This enables the solver to start the solving process using the valid derived constraints and lower bounds from the previous iteration, providing potential performance benefits (see Fig. 5).

4. The algorithmic framework

In the following, we describe our implementation of the *Step 1*, *Step 2*, and *Step 3* of the DDD paradigm, exploiting the definition of the IAP that we gave in the previous section.

We propose an algorithm, named DDD-TRP, that iteratively solves a IAP instance D_k defined on a partition set $\Lambda_k = \{\Lambda_k^{ir} : i \in I, r \in R_i\}$. Then, we prove that the DDD-TRP terminates after a finite number of steps, returning the optimal solution of the original TRP.

Note that, for each iteration k, the set Λ_k is always *more refined* with respect to the partition set Λ_{k-1} defined at the previous iteration.

We can also associate with each set of partitions Λ_k a *IAP graph* $G_k(V_k, E_k)$. In particular, we have that $V_k := \bigcup_{i\in I, r\in R_i} \Lambda_k^{ir}$, with $\Lambda_k^{ir} = \{\lambda_1^{ir}, \lambda_2^{ir}, \ldots, \lambda_{n^{ir}}^{ir}\}$, while the set E_k contains an edge of the type $\{\lambda_p^{ir}, \lambda_{p'}^{jq}\}$ for each incompatible couple of intervals in Λ_k (with i, j and r, q not necessarily distinct).

4.1. Initialize the DDD-TRP (Step 1)

We define the initial problem D_0 of the IAP by just considering, for each track segment used by each train, its feasibility interval $\lambda_1^{ir} = [\underline{t}^{ir}, M)$. Thus, we set $\Lambda_0 = \{\Lambda_k^{ir} = \{\lambda_1^{ir}\} \ : \ i \in I, r \in R_i\}$.

4.2. Solve the IAP (Step 2)

Observe that, for all $i \in I$ and $r \in R_i$, the set of nodes of $\Lambda_k^{ir} \in \Lambda_k$ defines a clique in the graph G_k. We denote such cliques as *resource assignment cliques*. Moreover, we can identify a second type of clique: given two consecutive track segments r and q traversed by a train i, such that $r \prec_i q$, for each λ_p^{ir} in the considered partition Λ_k, we can write a single *fixed precedence clique* constraint defined by the rail path incompatibilities (see Definition 3) as follows:

$$x_p^{ir} + \sum_{\lambda_{p'}^{iq} | h_{p'}^{iq} < h_p^{ir} + l^r} x_{p'}^{iq} \leq 1 \qquad i \in I,\ r, q \in R_i \text{ with } r \prec_i q,\ \lambda_p^{ir} \in \Lambda_k^{ip}. \tag{6}$$

One can visualize an example of a fixed precedence clique in Fig. 4(a). Here all intervals of Λ^{iq} highlighted in cyan belong to the clique defined by λ_p^{ir}.

Resuming, the IAP D_k associated with Λ_k, and solved at Step 2 of the DDD-TRP, reduces to find a minimum weighted set S_k^* of the IAP graph G_k that intersects:

- each resource assignment clique exactly once,
- each fixed precedence clique at least once,
- and each incompatible couple of intervals at least once.

4.3. Repair an infeasible schedule (Step 3)

Let $\bar{\tau} = \{\bar{t}^{ir} : i \in I, r \in R_i\}$ be the (infeasible) schedule we want to repair to (possibly) get a feasible schedule for TRP. We build the LP program LP_0 with variables t^{ir}, lower bound constraints $t^{ir} \geq \bar{t}^{ir} : i \in I, r \in R_i$ and all the time precedence constraints associated with the train-rail path (i.e. with Constraints 2.ii). We let the objective be the sum of the delays. Then, let $\bar{\tau}_0 = \bar{t}_0^{ir} : i \in I, r \in R_i$ be the optimal solution of LP_0 and $j := 1$.

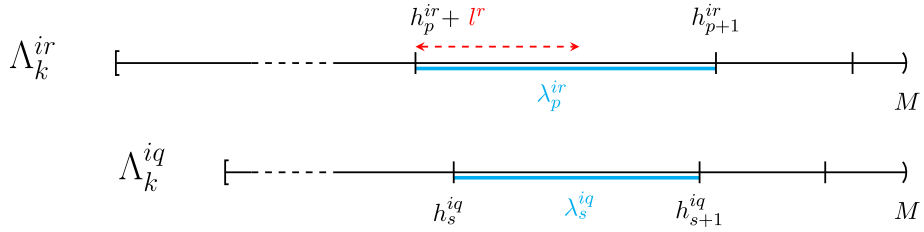
At the jth iteration, if $\bar{\tau}_{j-1} = \{\bar{t}_{j-1}^{ir} : i \in I, r \in R_i\}$ is feasible for the TRP, we are done. Otherwise, we build the linear program LP_j from LP_{j-1} by (a) replacing the lower bound constraints with the constraints $t^{ir} \geq \bar{t}_{j-1}^{ir} : i \in I, r \in R_i$ and (b) including one additional time precedence constraint. Indeed, since $\bar{\tau}_{j-1}$ is infeasible and all train-rail path precedence constraints are satisfied, there exists at least one disjunctive constraint of type 2.iii which is violated by $\bar{\tau}_{j-1}$. We add the one with the smallest left-hand side, i.e. the "first violated" in chronological order. Then we generate the new linear program by adding one of the two terms of the disjunction, namely the one that produces the smallest increase in the objective function of TRP. We solve again and obtain a new schedule $\bar{\tau}_{j+1}$ and iterate until the schedule is feasible or the current linear program is infeasible. Note that since the number of disjunctive constraints is finite, the repairing mechanism always terminates.

4.4. Refine the IAP (Step 4)

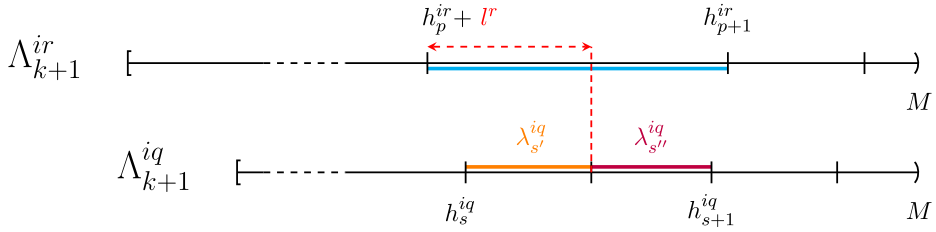
Let Λ_k be the set of partitions defined at iteration k of the DDD-TRP, G_k be the corresponding IAP graph, and let S_k^* be the optimal set of G_k (i.e. the optimal solution of the IAP D_k associated with Λ_k). Now, as already noticed, if $\tau_k^* = \Phi(S_k^*)$ is a feasible schedule then it defines an optimal solution to the TRP and we are done. So, assume τ_k^* is not feasible. This means that S_k^* contains two intervals, say $\lambda_p^{ir} = [h_p^{ir}, h_{p+1}^{ir})$ and $\lambda_s^{jq} = [h_s^{jq}, h_{s+1}^{jq})$ that are compatible (since S_k^* is Λ_k-feasible) but such that $t^{ir} = h_p^{ir}$ and $t^{jq} = h_s^{jq}$ violate either constraint (ii) or constraint (iii) of Definition 2. We consider separately the two cases.

A train-rail path precedence is violated. In this case, $i = j$ and we can assume, w.l.o.g., that $r \prec_i q$. Since t^{ir} and t^{iq} violate a train-rail path precedence, then $t^{ir} + l^r > t^{iq}$. Moreover, as λ_p^{ir} and λ_s^{iq} are compatible in S_k^*, then $t^{ir} + l^r < h_{s+1}^{iq}$. Therefore, we construct Λ_{k+1} from Λ_k (and then G_{k+1} from G_k) by replacing the interval λ_s^{iq} with the two smaller intervals $\lambda_{s'}^{iq} = [h_s^{iq}, t^{ir} + l^r)$ and $\lambda_{s''}^{iq} = [t^{ir} + l^r, h_{s+1}^{iq})$.

Observe that, in Λ_{k+1}, the intervals λ_p^{ir} and $\lambda_{s'}^{iq}$ are incompatible and, as a consequence, the optimal set S_{k+1}^* (hence the optimal schedule τ_{k+1}^*) that will be calculated at the next iteration of the DDD-TRP cannot contain both λ_p^{ir} and $\lambda_{s'}^{iq}$ ($t^{ir} = h_p^{ir}$ and $t^{iq} = h_s^{iq}$). This is equivalent to adding a vertex for the new interval $\lambda_{s''}^{iq}$ and an edge $\{\lambda_p^{ir}, \lambda_{s''}^{iq}\}$ in the set E_{k+1}. In other words, we are inserting a term $x_{s''}^{iq}$ to the fixed precedence clique associated with x_p^{ir}. Concurrently, we also add a new fixed precedence clique relative to the interval $\lambda_{s''}^{iq}$ itself. Fig. 6 shows how starting from a violated train-rail path precedence (Fig. 6(a)), new intervals $\lambda_{s'}^{iq}$ and $\lambda_{s''}^{iq}$ are generated (Fig. 6(b)).

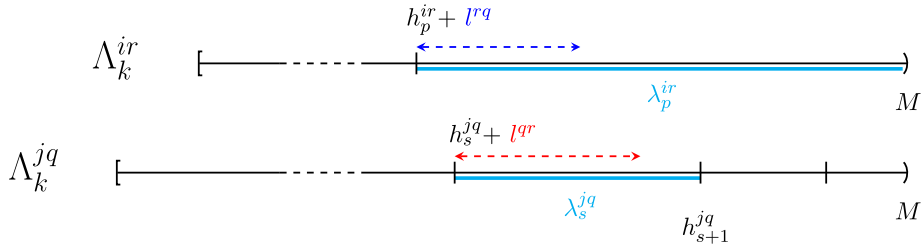


(a) Violated train-rail path precedence between two track segments traversed by train i, with $r \prec_i q$.

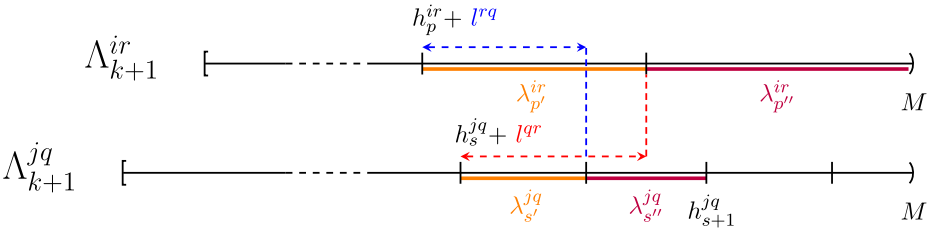


(b) Resolution of the violated train-rail path precedence shown in Figure 6a.

Fig. 6. Example of resolution of a violated train-rail path precedence at Step 3 of the DDD-TRP.



(a) Violated disjunctive precedence between two different trains i and j traversing the non-shareable track segments r and q.



(b) Resolution of the disjunctive precedence shown in Figure 7a.

Fig. 7. Example of resolution of a violated disjunctive precedence at Step 3 of the DDD-TRP.

A disjunctive precedence is violated. In this case, we have that $i \neq j$, $t^{jq} < t^{ir} + l^{rq}$ and $t^{ir} < t^{jq} + l^{qr}$. Since λ_p^{ir} and λ_s^{jq} are compatible in S_k^*, we have that $t^{ir} + l^{rq} < h_{s+1}^{jq}$ and $t^{jq} + l^{qr} < h_{p+1}^{ir}$. We obtain Λ_{k+1} from Λ_k by breaking λ_p^{ir} into $\lambda_{p'}^{ir} = [h_p^{ir}, t^{jq} + l_{qr}^{jq})$ and $\lambda_{p''}^{ir} = [t^{jq} + l_{qr}^{jq}, h_{p+1}^{ir})$, and λ_s^{jq} into $\lambda_{s'}^{jq} = [h_s^{jq}, t^{ir} + l^{rq})$ and $\lambda_{s''}^{jq} = [t^{ir} + l^{rq}, h_{s+1}^{jq})$.

Again here, we observe that the newly generated intervals $\lambda_{p'}^{ir}$ and $\lambda_{s'}^{jq}$ are now incompatible. This implies adding an edge $\{\lambda_{p'}^{ir}, \lambda_{s'}^{jq}\}$ to E_{k+1} since $\tau_{k+1}^* = \Phi(S_{k+1}^*)$ cannot contain the couple $(t^{ir} = h_p^{ir}, t^{jq} = h_s^{jq})$. Also, note that other conflicts and rail path incompatibilities may be generated by the definition of the new intervals. Consequently, the new incompatibilities should be added to the set E_{k+1}. In Fig. 7 we visually present the generation of the new intervals.

In both cases, once a new interval is defined, we can *propagate* the new time points discovered along the train rail path to ensure a time consistency with the intervals belonging to subsequent train track segments. With some abuse of notation in the following, given a track segment r traversed by a train i, we indicate the subsequent track segment on the rail path of i with the index $(r+1)$. Then, said $\lambda_p^{ir} = [h_p^{ir}, h_{p+1}^{ir})$ a newly generated interval and $\lambda_s^{i(r+1)} = [h_s^{i(r+1)}, h_{s+1}^{i(r+1)}) = max\{\Lambda_k^{i(r+1)} \ni \lambda_s^{i(r+1)} : h_s^{i(r+1)} < h_p^{ir} + l^r\}$, we define two new intervals of the type $\lambda_{s'}^{i(r+1)} = [h_s^{i(r+1)}, h_p^{ir} + l^r)$ and $\lambda_{s''}^{i(r+1)} = [h_p^{ir} + l^r, h_{s+1}^{i(r+1)})$. This procedure allows us to identify a lower bound $h_{s''}^{i(r+1)}$ for $(r+1)$ that avoids the violation of the rail path constraints. The propagation is thus recursively applied on $\lambda_{s'}^{i(r+1)}$ until the final destination track segment is reached. Along with the creation of these intervals, we need to add every rail path and/or conflict incompatibility arising between the intervals in Λ_{k+1}.

Summarizing, said $G_k(V_k, E_k)$ the IAP graph at a generic iteration k, the DDD-TRP follows the steps reported below. Note that, since

we apply a rail path time consistency each time we propagate a new interval, the feasibility check on the associated schedule τ_k^* can be limited to the disjunctive constraints.

DDD-TRP k-th iteration

Data: Graph $G_k(V_k := \Lambda_k, E_k)$, functions $\Phi(S) : \Lambda_k \to \mathbb{R}_+$ and $\bar{c}(S) : \Lambda_k \to \mathbb{R}_+$.

Solve the IAP (Step 2)

Find S_k^* on $G_k(V_k, E_k)$ and compute $\tau_k^* = \Phi(S_k^*)$

◇Check for solution optimality

For each $i, j \in I$, $i \neq j$, and $r \in R_i$, $q \in R_j$ with $\{r, q\} \in \mathcal{D}$
If $t^{ir}, t^{jq} \in \tau_k^*$ violate a disjunctive precedence (Definition 2.(iii)) **Then**

(i) from $\lambda_p^{ir} \in S_k^*$ define $\lambda_{p'}^{ir} = [h_p^{ir}, t^{jq} + l_{qr}^{jq})$ and $\lambda_{p''}^{ir} = [t^{jq} + l_{qr}^{jq}, h_{p+1}^{ir})$, set $\Lambda_k^{ir} = \Lambda_k^{ir} \setminus \{\lambda_p^{ir}\} \cup \{\lambda_{p'}^{ir}, \lambda_{p''}^{ir}\}$;
(ii) from $\lambda_s^{jq} \in S_k^*$ define $\lambda_{s'}^{jq} = [h_s^{jq}, t^{ir} + l^{rq})$ and $\lambda_{s''}^{jq} = [t^{ir} + l^{rq}, h_{s+1}^{jq})$, set $\Lambda_k^{jq} = \Lambda_k^{jq} \setminus \{\lambda_s^{jq}\} \cup \{\lambda_{s'}^{jq}, \lambda_{s''}^{jq}\}$;
(iii) set $E_k = E_k \cup \{\lambda_{p'}^{ir}, \lambda_{s'}^{jq}\}$.

If $\nexists$ violated disjunctive precedence **Then**
Stop. $\tau^* = \tau_k^*$ with value $c(\tau^*) = \bar{c}(S_k^*)$.

Repair the IAP solution (Step 3)

Find τ_k by repairing τ_k^*.

◇Check for solution optimality

If $c(\tau_k) - \bar{c}(S_k^*) < \epsilon$ **Then**
Stop. $\tau^* = \tau_k$ with value $c(\tau_k)$.

Refine the IAP (Step 4)

For each newly defined interval λ_p^{ir}

(i) update *resource assignment clique*: for each $\lambda_{p'}^{ir} \in \Lambda_k^{ir} \setminus \{\lambda_p^{ir}\}$ set $E_k = E_k \cup \{\lambda_p^{ir}, \lambda_{p'}^{ir}\}$;
(ii) update *fixed precedence clique* (Definition 3): for each $\lambda_s^{iq} \in \Lambda_k^{iq}$ with $(q+1) = r$ such that $h_s^{iq} + l^{iq} > h_{p+1}^{ir}$ add a new edge $\{\lambda_p^{ir}, \lambda_s^{iq}\}$ to E_k;
(iii) update *conflict incompatibility* (Definition 4): for each $\lambda_s^{iq} \in \Lambda_k^{iq}$ with $i \neq j$, $\{r, q\} \in \mathcal{D}$, such that $h_{p+1}^{ir} < h_{p'}^{jq} + l^{qr}$ and $h_{p'+1}^{jq} < h_p^{ir} + l^{rq}$ add a new edge $\{\lambda_p^{ir}, \lambda_s^{jq}\}$ to E_k;
(iv) *propagate* along the train rail path: if exists $(r+1) \in R_i$, let $\lambda_s^{i(r+1)} = max\{\Lambda_k^{i(r+1)} \ni \lambda_s^{i(r+1)} \mid h_s^{i(r+1)} < h_p^{ir} + l^r\}$, define $\lambda_{s'}^{i(r+1)} = [h_s^{i(r+1)}, h_p^{ir} + l^r)$ and $\lambda_{s''}^{i(r+1)} = [h_p^{ir} + l^r, h_{s+1}^{i(r+1)})$, set $\Lambda_k^{i(r+1)} = \Lambda_k^{i(r+1)} \setminus \{\lambda_s^{i(r+1)}\} \cup \{\lambda_{s'}^{i(r+1)}, \lambda_{s''}^{i(r+1)}\}$.

Set $V_{k+1} = V_k$ and $E_{k+1} = E_k$
$k = k + 1$

4.5. Convergence of the algorithm

We now show that the DDD-TRP converges to the optimal solution in a finite number of steps. We remark that our approach for solving the TRP can be also seen as a dual procedure with both row and column generations. At each iteration k, a restricted relaxation of the basic problem is solved, if the solution cannot be converted into a feasible solution for the original formulation (at least) a row is added to the constraints groups and (at least) a new variable is generated and added to the problem D_{k+1}. Otherwise, Theorem 1 ensures that a solution of the same value can be obtained for the TRP. In particular, we prove the following:

Theorem 2. *The* DDD-TRP *terminates providing the optimal solution of the TRP or proving that the problem is infeasible.*

Proof. Theorem 1 shows that at each iteration k of the DDD-TRP, the value of the optimal solution S_k^* defines a lower bound on the value of the optimal solution of the TRP. At the first iteration of the algorithm, if the heuristic mechanism fails to produce a feasible solution at step 3, we can conclude that the problem is infeasible. Otherwise, each partition Λ^{ir} is defined by one interval (see Section 4.1). As shown in Section 4.4, at each iteration k, we add at least one interval to the current set Λ_k. Since the running time values l_t^r are assumed to be integral, the maximum number of intervals that can be defined for each resource $r \in R_i$ is then M. Therefore, Observation 1 implies that, after at most $\sum_{i \in I} |R_i| M$ iterations, the DDD-TRP terminates either providing the optimal solution for the TRP. □

5. Computational experiments

In this section, we report the results of the computational experiments conducted to assess the performance of the DDD approach in the train re-scheduling context and compare it with alternative approaches, especially the Big-M approach, which is the main alternative competitor.

We solve the DDD-TRP using both a MILP solver and a MaxSAT solver to compare their effectiveness.

The test set consists of 72 real-life instances derived from two single-track railroad networks, later named *Line A* and *Line B*, of the Norwegian railway. More details are given below.

5.1. Instances

The instances considered refer to portions of a physical railway network infrastructure, comprising stations and single-tracks, and including junctions between different lines. The lengths of tracks may vary considerably. A set of trains with different speed classes traverses the network. For each of them, we are given a desired timetable. At a given instant in time, the state of the network is provided by the current position of all trains and their deviations from the scheduled departure times. Note that when taking a snapshot of the network at a particular instant, a train may be *on time*, i.e. its arrival and departure times adhere to the timetable schedule, or it may be affected by a delay. Besides, a train can be either positioned at a station (i.e. *in station*) or on an open line track (i.e. *in connection*). In the second case, the delay refers to the time at which the train entered the last track segment traversed, i.e. the one it is occupying at that moment. We do not consider the possibility of accelerating or decelerating the rolling stock, therefore we consider the train speeds, and consequently the train travel time, as fixed.

Line A is a 124 km long line for passenger trains and includes 30 stations and 33 track segments. The A-instances present on average a set of 20 trains with an average of 19 track segments each. Line B is smaller (115 km, 20 stations and 25 tracks) and is crossed by commuters and freight trains. The B-instances present, on average, 11 trains and 15 track segments for each scheduled path. See Table D.1 in Appendix D for more details about the original 24 test instances. We emphasize that, since the instances of Line A include in most of cases more trains than those belonging to Line B, the number of potential conflicting track segments can be significantly higher for them, thus they will produce more complex models.

The snapshots were extracted from the real-time train information system of the Norwegian railway. Most of the time, most of the trains are running on time, which makes the re-scheduling problem easy: simply follow the prescribed timetable. So, a random sampling of snapshots would not be an interesting benchmark. Real-time train re-scheduling optimization systems will have harder challenges and more value to the dispatchers when many trains have large delays. When large delays happen, there are many more possible trade-offs to make between delays on different trains, and the optimization search tree becomes much larger. To simulate a more difficult setting, we first modified all 24

instances to have a mandatory dwelling time in the station equal to the timetable dwelling time. This removes the possibility for trains to catch up with their delay by shorter dwelling times. Secondly, we created a third set of 24 instances with increased running time for travelling from one station to the next, without adjusting the timetable accordingly, to simulate slow-downs. Such slow-downs may for example be caused by signalling equipment faults. The problem instances are available (see https://github.com/luteberget/maxsattrainscheduling).

In the following, we denote the original instance set as O, the set of instances with mandatory dwelling time added to stations as S, and the set of instances obtained by adding extra time to tracks as T. We will refer to the instances with the notation I_n^l, where, by $l \in \{AO, BO, AS, BS, AT, BT\}$ we indicate the line (A,B) and the test set (O,S,T) to which they belong, and by the subscript $n \in \{1, \ldots, 12\}$ the instance number.

Objective functions. Following the normal practice in the railway industry, we minimize the delays of trains only at their final destination stations. We ran the computational experiments using the objective functions defined in Section 2. To model the three functions, in the Big-M formulation we introduced one continuous variable and one linear inequality for each t^{if} (**Linear continuous**), one integer variable and one linear inequality for each t^{if} (**Linear rounded**), and one binary variable per step (**Stepwise**).

The DDD-TRP implements each of these cost functions by simply evaluating them for each new binary variable that is added to the problem, and adding the corresponding objective component for the binary variable.

5.2. Computational results and discussion

This section reports the computational results for the Gurobi Big-M model, the Gurobi IAP MILP model, and the MaxSAT incremental RC2 DDD model. A timeout of 2 min was used.

We highlight that the algorithms work by iteratively removing conflicts (infeasibility) while increasing the lower bound and using the primal repair heuristic introduced in Section 4.3 to possibly produce feasible solutions during the search. The repair heuristic is also used in the Big-M row generation algorithm, also there taking lower bounds from the optimal solution of the relaxed problem solved at every iteration. On some instances, both for the DDD and Big-M implementations, the heuristic causes the solver to terminate earlier by closing the gap. On instances where the solvers timed out, the heuristic produced a feasible solution in all cases, and the remaining optimality gap is reported in the tables of Appendix D.

For a complete list of all tested combinations of formulations and solvers, we direct interested readers to Appendix B.

In our experiments, we observe that the number of iterations and the number of solved conflicts can vary significantly from instance to instance. Both these indicators affect the overall solution time. In fact, at each iteration k, a new IAP has to be solved and its complexity grows with the number of nodes and edges defined in G_k. We observe that the DDD-TRP solved with a MaxSAT approach requires a higher number of iterations before it finishes. This can be explained by two causes: first, the number includes both refinements of the IAP graph G_k (when the SAT solver finds a feasible solution), and refinements of the objective function (when the SAT solver reports an infeasible problem), due to how the RC2 algorithm works. Secondly, we implemented only the least amount of refinement of the graph G_k required, omitting step 3-(iv) of DDD-TRP. Because the SAT solver handles incremental additions to its problem instance well, we believe this to give only a negligible performance impact (including step 3-(iv)). Both the SAT and MILP versions of the DDD-TRP spend most of their time in the solving phase and only a very small fraction of time (<3%) on building, conflicts checking and refining the IAPs.

Table 1
Number of instances solved and average computational times for the three algorithms, aggregated by objective types and test sets.

Obj	Set	# Solved			Avg time (ms)		
		Big-M	MILP	MaxSAT	Big-M	MILP	MaxSAT
Cont.	O	24	22	22	181	5,511	1334
	S	22	20	19	1285	5,893	349
	T	22	18	16	2929	15,041	8524
	All	**68**	**60**	**57**	**1427**	**8,497**	**3024**
Round.	O	24	22	24	132	722	34
	S	22	21	22	1857	7,505	856
	T	22	19	22	2902	8,501	1787
	All	**68**	**62**	**68**	**1587**	**5,403**	**867**
Step.	O	24	24	24	55	283	14
	S	24	22	24	251	5,768	49
	T	24	19	24	347	774	73
	All	**72**	**65**	**72**	**218**	**2,283**	**45**

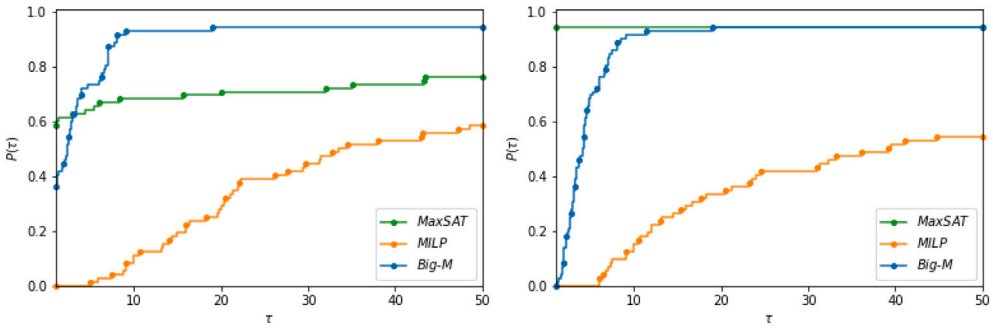
Fig. 8 shows the performance profiles of the three algorithms for each of the objective types — linear continuous, linear rounded, and stepwise, respectively. Aggregated statistics for the test instances are provided in Table 1. For a comprehensive overview of computational results, please refer to Tables D.2-D.4 within Appendix D.

Linear continuous objective. Examining the performance of the Big-M formulation with the linear objective, it is observed that the MaxSAT solver for the DDD-TRP emerges as the fastest approach in 60% of the problems within the test sets. However, for the remaining instances, the Big-M formulation exhibits superior performance. We believe this to be a general weakness with exact weighted MaxSAT based directly on standard SAT solvers because such algorithms (including RC2) work by finding logical conflicts between subsets of components of the objective and creating cardinality constraints (linear constraints), which are costly to translate into SAT. When weights vary a lot, such as a cost of 1-s delay for one train in conflict with a delay of 10 min for another train, or when conflicts span over a large subset of the objective components, representing this trade-off exactly as SAT constraints is computationally expensive. On the contrary, the MILP solver demonstrates superiority in handling such intricate numerical structures, successfully resolving 60 out of 72 instances. Nevertheless, despite its efficiency in solving, the MILP solver exhibits significantly slower performance, with the MaxSAT solver being on average 20 times faster for the 57 instances it successfully addressed.

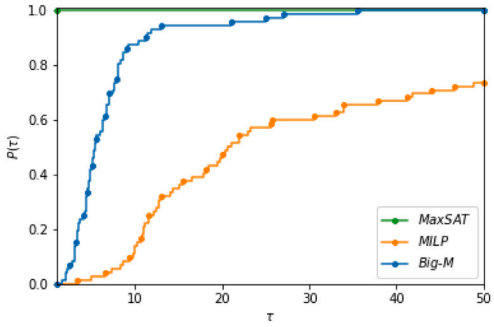
Linear rounded objective. When we change the objective function to the stepwise and linear rounded forms, the picture changes completely in favour of the MaxSAT. In Table 1, it is observed that when employing the linear rounded objective, all the problem instances successfully solved by Big-M were also resolved by DDD-TRP using the MaxSAT approach (68 out of 72 instances). Notably, the MaxSAT approach demonstrated superior speed, typically ranging from 2x to 10x faster. Furthermore, the MILP solver exceeds the time limit for ten instances and consistently exhibits the highest computational time among the considered approaches.

Stepwise objective. In the case of the stepwise objective function, computation times are notably lower overall, with all instances solved within the timeout threshold for both the Big-M and MaxSAT approaches. In contrast, the MILP solver exceeds the timeout for seven instances out of 72. The MaxSAT DDD-TRP is the fastest approach over all instances, with a speed-up of 2x-20x over the Big-M.

In Table 2 we report a comparison of computation times (in milliseconds) for the stepwise function, focusing on the ten most challenging instances in our set, which demand the most time for the Big-M model to solve. We can see how employing the MaxSAT solver with a stepwise function significantly impacts computation times.



(a) Linear continuous objective. (b) Linear rounded objective.



(c) Stepwise objective.

Fig. 8. Performance profiles based on the CPU times of solved instances.

Table 2
Comparison of the computation times obtained by the Big-M MILP and MaxSAT approaches on the 10 hardest instances with a stepwise objective function.

Instance	Time (ms)			Speed-up	
	Big-M	MILP	MaxSAT	Big-M	MILP
I_1^{AT}	2204	8 376	188	11.7×	44.6×
I_2^{AT}	753	1 997	58	12.9×	34.2×
I_8^{AT}	7953	59 085	210	37.9×	281.4×
I_{11}^{AT}	T/O [3% gap]	108 005	396	–	272.8×
I_{12}^{AT}	14 680	63 967	540	27.2×	118.4×
I_1^{AS}	10 559	3 901	223	47.4×	17.5×
I_2^{AS}	474	1 923	116	4.1×	16.5×
I_8^{AS}	4099	7 816	215	19.1×	36.4×
I_{11}^{AS}	18 297	11 073	437	41.9×	25.3×
I_{12}^{AS}	6137	30 737	286	21.4×	107.4×

This suggests a way to extend the DDD-TRP to a dual approach where the objective function is gradually extended from a single step to the full linear rounded objective. Simple step function objectives can be solved much faster and can provide feasible train schedules in case the exact optimal solution takes too long to compute in a real-time setting. The model resolution speed is crucial when applied to the dynamic solution of train dispatching problems. In this case, a new instance is generated from the field every ten seconds or so (Lamorgese et al., 2018). Typically instances only slightly change over time; therefore, one can refine the last instance generated during the previous resolution process, aiming to generate only a few new intervals. In Mannino and Sartor (2020) one can observe that this approach proved to be successful when extending the *Path&Cycle* formulation to cope with dynamic instances of the TRP.

6. Conclusions and future work

This paper demonstrates how the dynamic discretization paradigm can be adapted to make TI formulations competitive with the Big-M formulations in the train dispatching field. When the objective function is piecewise constant, our approach outperforms the Big-M formulation on all the problem instances in our set of real-world models and data.

We achieved better performance using a MaxSAT solver instead of a MILP solver; we believe this is due to the way that SAT solvers can solve a sequence of incrementally constructed problem instances very efficiently. Indeed, this is crucial for many, if not most, industrial applications of SAT solvers (see Kochemazov et al., 2021). Our DDD-TRP represents the first application of MaxSAT to train re-scheduling (some work has been done on periodic railway timetabling, see Reisch, 2021). Going forward, we would like to investigate in more detail how MILP solvers are affected by small, incremental changes in the problem instance, and see if there is a way to make MILP solvers work efficiently with the DDD-TRP.

It follows a very natural road map for future studies and developments. First, adapt the approach to handle dynamic problems and re-optimization. The refinement of the interval between one iteration and the next can be seen as a decomposition, so one can ask how to fit previously generated resolution cuts to new partial models. Also, fixed variable values retrieved in the preprocessing (solving) phase (or previously calculated bounds) can be exploited for those elements that are not directly "involved" in the conflicts solved at a generic iteration k. Second, develop techniques to limit the generation of intervals at each iteration. It can be shown that only a small subset of the intervals defined for the last IAP solved is sufficient to find an overall feasible (and thus optimal) solution. Third, it is possible to speed up the algorithm by selecting different and/or multiple time points in the intervals. Currently, we obtain the optimal solution by assigning each interval its lower end, but we can also make different choices and possibly

obtain a faster yet feasible solution. Fourth, the described methodology applies to every job-shop scheduling problem, so it is logical to extend it to cope with other contexts, such as industrial production or project scheduling.

CRediT authorship contribution statement

Anna Livia Croella: Writing – review & editing, Writing – original draft, Visualization, Data curation, Conceptualization. **Bjørnar Luteberget:** Writing – review & editing, Writing – original draft, Software, Data curation, Conceptualization. **Carlo Mannino:** Writing – review & editing, Writing – original draft, Methodology, Formal analysis, Conceptualization. **Paolo Ventura:** Writing – review & editing, Writing – original draft, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors confirm that the data supporting the findings of this study are available within the article and its supplementary materials. The data do not violate the protection of human subjects, or other valid ethical, privacy, or security concerns.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cor.2024.106679.

References

Abels, D., Jordi, J., Ostrowski, M., Schaub, T., Toletti, A., Wanko, P., 2021. Train scheduling with hybrid answer set programming. Theory Pract. Log. Program. 21 (3), 317–347. http://dx.doi.org/10.1017/S1471068420000046.

Bacchus, F., Järvisalo, M., Martins, R., 2019. MaxSAT evaluation 2018: New developments and detailed results. J. Satisf. Boolean Model. Comput. 11 (1), 99–131. http://dx.doi.org/10.3233/SAT190119.

Bešinović, N., Goverde, R.M., Quaglietta, E., Roberti, R., 2016. An integrated micro–macro approach to robust railway timetabling. Transp. Res. B 87, 14–32.

Bettinelli, A., Santini, A., Vigo, D., 2017. A real-time conflict solution algorithm for the train rescheduling problem. Transp. Res. B 106, 237–265.

Björk, M., 2011. Successful SAT encoding techniques. J. Satisf. Boolean Model. Comput. 7 (4), 189–201. http://dx.doi.org/10.3233/sat190085.

Boland, N., Hewitt, M., Marshall, L., Savelsbergh, M., 2017. The continuous-time service network design problem. Oper. Res. 65 (5), 1303–1321.

Boland, N.L., Savelsbergh, M.W., 2019. Perspectives on integer programming for time-dependent models. Top 27 (2), 147–173.

Cacchiani, V., Toth, P., 2018. Robust train timetabling. In: Handbook of Optimization in the Railway Industry. Springer, pp. 93–115.

Caimi, G., Fuchsberger, M., Laumanns, M., Lüthi, M., 2012. A model predictive control approach for discrete-time rescheduling in complex central railway station areas. Comput. Oper. Res. 39 (11), 2578–2593.

Croella, A.L., 2022. Real-Time Train Scheduling: Reactive and Proactive Algorithms for Safe and Reliable Railway Networks (Ph.D. thesis). Sapienza University of Rome, Department of Computer, Control, and Management Engineering Antonio Ruberti (DIAG), Italy.

Croella, A.L., Mannino, C., Ventura, P., 2021. Dynamic discretization discovery for the train scheduling problem. In: RailBeijing 2021, the 9th International Conference on Railway Operations Modelling and Analysis (ICROMA), Beijing, China, November 3 - 7, 2021, Conference Proceedings.

Croella, A.L., Sasso, V., Lamorgese, L., Mannino, C., Ventura, P., 2022. Disruption management in railway systems by safe place assignment. Transp. Sci. 56, http://dx.doi.org/10.1287/trsc.2021.1107.

Dash, S., Günlük, O., Lodi, A., Tramontani, A., 2012. A time bucket formulation for the traveling salesman problem with time windows. INFORMS J. Comput. 24 (1), 132–147.

Desaulniers, G., Desrosiers, J., Solomon, M.M., 2006. Column Generation, vol. 5, Springer Science & Business Media.

Fang, W., Yao, X., 2015. A survey on problem models and solution approaches to rescheduling in railway networks. IEEE Trans. Intell. Transp. Syst. 16, 2997–3016. http://dx.doi.org/10.1109/TITS.2015.2446985.

Fischetti, M., Monaci, M., 2009. Light robustness. In: Robust and Online Large-Scale Optimization. Springer, pp. 61–84.

Gao, R., Niu, H., 2021. A priority-based ADMM approach for flexible train scheduling problems. Transp. Res. C 123, 102960. http://dx.doi.org/10.1016/j.trc.2020.102960.

Hansen, I.A., Pachl, J., 2014. Railway Timetabling & Operations. Eurailpress, Hamburg.

Harrod, S., 2011. Modeling network transition constraints with hypergraphs. Transp. Sci. 45 (1), 81–97.

Harrod, S., 2012. A tutorial on fundamental model structures for railway timetable optimization. Surv. Oper. Res. Manag. Sci. 17 (2), 85–96.

Hewitt, M., 2019. Enhanced dynamic discretization discovery for the continuous time load plan design problem. Transp. Sci. 53 (6), 1731–1750.

Ignatiev, A., Morgado, A., Marques-Silva, J., 2019. RC2: an efficient MaxSAT solver. J. Satisf. Boolean Model. Comput. 11 (1), 53–64. http://dx.doi.org/10.3233/SAT190116.

Kochemazov, S., Ignatiev, A., Marques-Silva, J., 2021. Assessing progress in SAT solvers through the lens of incremental SAT. In: Li, C., Manyà, F. (Eds.), Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings. In: Lecture Notes in Computer Science, Vol. 12831, Springer, pp. 280–298. http://dx.doi.org/10.1007/978-3-030-80223-3_20.

Lamorgese, L., Mannino, C., 2015. An exact decomposition approach for the real-time train dispatching problem. Oper. Res. 63 (1), 48–64.

Lamorgese, L., Mannino, C., Natvig, E., 2017. An exact micro–macro approach to cyclic and non-cyclic train timetabling. Omega 72, 59–70.

Lamorgese, L., Mannino, C., Pacciarelli, D., Krasemann, J.T., 2018. Train dispatching. Handb. Optim. Railw. Ind. 265–283.

Lamorgese, L., Mannino, C., Piacentini, M., 2016. Optimal train dispatching by Benders' like reformulation. Transp. Sci. 50 (3), 910–925.

Leutwiler, F., Corman, F., 2022. A logic-based benders decomposition for microscopic railway timetable planning. European J. Oper. Res..

Li, C.M., Manyà, F., 2021. MaxSAT, hard and soft constraints. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (Eds.), Handbook of Satisfiability - Second Edition. In: Frontiers in Artificial Intelligence and Applications, Vol. 336, IOS Press, pp. 903–927. http://dx.doi.org/10.3233/FAIA201007.

Lusby, R.M., Larsen, J., Ehrgott, M., Ryan, D.M., 2013. A set packing inspired method for real-time junction train routing. Comput. Oper. Res. 40 (3), 713–724. http://dx.doi.org/10.1016/j.cor.2011.12.004, URL https://www.sciencedirect.com/science/article/pii/S0305054811003595, Transport Scheduling.

Lusby, R., Larsen, J., Ryan, D., Ehrgott, M., 2011. Routing trains through railway junctions: A new set-packing approach. Transp. Sci. 45, 228–245. http://dx.doi.org/10.1287/trsc.1100.0362.

Mannino, C., Mascis, A., 2009. Optimal real-time traffic control in metro stations. Oper. Res. 57 (4), 1026–1039.

Mannino, C., Nakkerud, A., 2023. Optimal train rescheduling in Oslo central station. Omega 116.

Mannino, C., Sartor, G., 2020. An exact (re) optimization framework for real-time traffic management. optim. Online.

Marshall, L., Boland, N., Savelsbergh, M., Hewitt, M., 2021. Interval-based dynamic discretization discovery for solving the continuous-time service network design problem. Transp. Sci. 55 (1), 29–51.

Mascis, A., Pacciarelli, D., 2002. Job-shop scheduling with blocking and no-wait constraints. European J. Oper. Res. 143 (3), 498–517.

Matos, G.P., Albino, L.M., Saldanha, R.L., Morgado, E.M., 2021. Solving periodic timetabling problems with SAT and machine learning. 13, (3), pp. 625–648. http://dx.doi.org/10.1007/s12469-020-00244-,

Meng, L., Zhou, X., 2014. Simultaneous train rerouting and rescheduling on an N-track network: A model reformulation with network-based cumulative flow variables. Transp. Res. B 67, 208–234.

Pellegrini, P., Marlière, G., Rodriguez, J., 2014. Optimal train routing and scheduling for managing traffic perturbations in complex junctions. Transp. Res. B 59, 58–80.

Prestwich, S.D., 2021. CNF encodings. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (Eds.), Handbook of Satisfiability - Second Edition. In: Frontiers in Artificial Intelligence and Applications, Vol. 336, IOS Press, pp. 75–100. http://dx.doi.org/10.3233/FAIA200985.

Queyranne, M., Schulz, A.S., 1994. Polyhedral Approaches to Machine Scheduling. Citeseer.

Reisch, J., 2021. Railway Timetable Optimization (Ph.D. thesis). Freie Universität Berlin, URL http://dx.doi.org/10.17169/refubium-30524.

Reynolds, E., Ehrgott, M., Maher, S.J., Patman, A., Wang, J.Y., 2020. A multicommodity flow model for rerouting and retiming trains in real-time to reduce reactionary delay in complex station areas. Optim. Online.

Roussel, O., Manquinho, V.M., 2021. Pseudo-Boolean and cardinality constraints. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (Eds.), Handbook of Satisfiability - Second Edition. In: Frontiers in Artificial Intelligence and Applications, Vol. 336, IOS Press, pp. 1087–1129. http://dx.doi.org/10.3233/FAIA201012.

2019. SBB swiss federal railways - train schedule optimisation challenge. https://www.aicrowd.com/challenges/train-schedule-optimisation-challenge.

2020. SBB swiss federal railways - flatland challenge. https://www.aicrowd.com/challenges/flatland-challenge.

Scherr, Y.O., Hewitt, M., Saavedra, B.A.N., Mattfeld, D.C., 2020. Dynamic discretization discovery for the service network design problem with mixed autonomous fleets. Transp. Res. B 141, 164–195.

Schlechte, T., Borndörfer, R., Erol, B., Graffagnino, T., Swarat, E., 2011. Micro–macro transformation of railway networks. J. Rail Transp. Plan. Manage. 1 (1), 38–48.

2023. The 2023 RAS problem solving competition. https://connect.informs.org/railway-applications/new-item3/problem-solving-competition681.

Vu, D.M., Hewitt, M., Boland, N., Savelsbergh, M., 2020. Dynamic discretization discovery for solving the time-dependent traveling salesman problem with time windows. Transp. Sci. 54 (3), 703–720.

Vu, D.M., Hewitt, M., Vu, D.D., 2022. Solving the time dependent minimum tour duration and delivery man problems with dynamic discretization discovery. European J. Oper. Res. 302 (3), 831–846. http://dx.doi.org/10.1016/j.ejor.2022.01.029, URL https://www.sciencedirect.com/science/article/pii/S0377221722000674.

Wang, X., Regan, A.C., 2002. Local truckload pickup and delivery with hard time window constraints. Transp. Res. B 36 (2), 97–112.

Zhan, S., Wong, S., Shang, P., Peng, Q., Xie, J., Lo, S., 2021. Integrated railway timetable rescheduling and dynamic passenger routing during a complete blockage. Transp. Res. B 143, 86–123. http://dx.doi.org/10.1016/j.trb.2020.11.006, URL https://www.sciencedirect.com/science/article/pii/S0191261520304264.