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*

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, ensure 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 results faster on piece-wise constant objective functions, while the Big-M approach maintains the lead on linear continuous objectives.

*Corresponding author

Email addresses: annalivia.croella@uniroma1.it (Anna Livia Croella), bjornar.luteberget@sintef.no (Bjørnar Luteberget), carlo.mannino@sintef.no (Carlo Mannino), paolo.ventura@iasi.cnr.it (Paolo Ventura)

Preprint, June 14, 2023

Keywords: Train Rescheduling, Dynamic Discretization Discovery, Maximum SATisfiability problem

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. Train scheduling is central in various phases of traffic planning. In strategic and tactical planning - performed from 5 years to months or even to few hours before the actual movements - train scheduling amounts to establishing the timetable trains should follow in their trip. Roughly speaking, train scheduling amounts to establishing the time in which each train enters and exits the resources encountered on its path [1]. 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 on-line activity is called *train rescheduling* or *train dispatching* [2] - the latter typically also including 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 [2]), or some robustness measures [1, 3, 4]. At the operational level, in contrast, one typically wants to minimize some measure of the deviation of the actual schedule from the official timetable [2].

From the optimization theory standpoint, train scheduling is a generalization of job-shop (with blocking and no-wait constraints, [5]) 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 [1, 6, 2].

There are of course different approaches to train scheduling, but here we are interested in those based on Mixed Integer Linear Program (MILP),

which are the most adopted in the literature [2]. 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 [7]: *Big-M* formulations and *Time-Indexed* (TI) formulations (see [8] 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 $t_{i,r}$. In order to represent a disjunctive constraint, one must introduce a binary variable and two constraints which contains the notorious Big-M term, that 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 size 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 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 favor of the Big-M alternative [9].

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 field, or feasible field schedules may be cut off by the TI model [10, 11].

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 [1] for a survey) and only very few to dispatching (such as [12, 13, 14, 15]). In any case, because the complete formulation would be too large to handle, all these papers resort to delayed column generation procedures [16]. 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 - [17, 18]) and/or its dual problem combined with branch-and-price techniques (see [19, 20]). 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 [15] for a medium-size station (Doncaster) with the experiments over a large junction performed in [21] with a Big-M model at 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 [22] and then further extended in [23, 24, 25, 26]. 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 [27] and the method presented in [28]. 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 sizes. 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 $t_{i,r} \in \lambda_p$ during the p-th 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 $t_{i,r} \in \lambda_p$ and $t_{j,r} \in \lambda_q$, 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 the three components of the DDD paradigm, as introduced in [11]. 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 such system by means of an open source SAT solver is dramatically more efficient than solving the original IAP by a state-of-the-art MILP solver. Note that a similar behaviour was observed in [29] where the authors compare a MILP and a SAT version of certain binary programs, and experience significant improvements with the latter. 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 [29] Leutwiler and Corman resolve this problem 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 [30]. In this paper, we successfully used the RC2 algorithm [31] to solve DDD formulations of the train scheduling problem.

A preliminary version of the ideas developed in this work were presented

in [32] and in [33].

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.

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 [34] 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., [35, 36, 37, 38]). Accordingly, in our experiments, we will make use of macroscopic real-life instances.

We look at the on-line version of the train scheduling problem, also called *train dispatching* or *train rescheduling*, and we do not consider the possibility of rerouting. Note that the model can be extended to cope with multiple routes, for instances as in [29, 39]. 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. Depending on the next scheduling decisions, some trains may reduce their delays at destination, while others may increase it. The target is to schedule trains for the next few hours so as to minimize the cost of delays. We next introduce our main modeling choices and the formalism.

2.1. Problem Definition

Networks, trains and rail paths. In Figure 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, refuel, etc.). Tracks are further subdivided in (one or more) *tracks segments* or *block sections*, that can accommodate at most a train at a time.

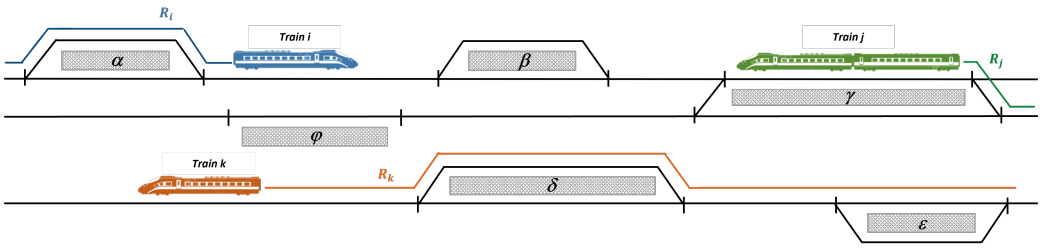


Figure 1: Schematic representation of a railway infrastructure. Stations are depicted as filled rectangles.

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 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 $r \prec_i q$ ($r \preceq_i q$) if r (strictly, resp.) precedes q in the rail path of i.

For $r \in R_i$, we also let $l_i^r \in \mathbb{R}_+$ be the minimum amount of time i needs to traverse track segment r (*running time* of i on r). 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 $t^i \in \mathbb{R}^{|R_i|}$, where component t^{ir} denotes the time when i enters track segment $r \in R_i$. The *(full) schedule* will be thus the vector $t = \{t^{ir} \mid i \in I, r \in R_i\}$. 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 $t = \{t^{ir} \mid i \in I, r \in R_i\}$ 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 precedence*) $t^{iq} \geq t^{ir} + l^r$, for each $i \in I$, and $r, q \in R_i$ with $r \prec_i q$;

iii) (*disjunctive precedence*) 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 *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 real number, i.e., $t^{ir} \ll M$ for each $i \in I$, and $r \in R_i$.

Objective function. Following the normal practice in railway industry, we define as optimal a schedule that minimizes the train delays at their final destination stations and we consider separable cost functions. For each $i \in I$, let t^{if} be the time train i enters its final station. As defined before, we let

$\underline{t}^{if}$ be the wanted arrival time for i at final destination. Then, the delay of $i \in I$ is

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

Then, the cost function is defined as $\Sigma_{i \in I}\, c^{if}$, and we consider three cases:

1. **Linear continuous**: simply summing the delays, namely $c^{if} = d(t^{if})$, for each train $i \in I$. In the Big-M formulation, this objective is implemented by introducing one continuous variable and one linear inequality for each t^{if}.

2. **Linear rounded**: $c^{if} = \lfloor d(t^{if})/Q \rfloor$, for each train $i \in I$, where Q is a constant. We use $Q = 180$, i.e., 3 minutes. 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^{if}.

3. **Stepwise**: Each train has its own finite sequence of *steps* valid for specific delay ranges. For example (see also Figure 2):

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

   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 the punctuality is defined as the percentage of trains that were less than e.g. 5 minutes delayed.

   It is also interesting to note that since this cost function has a maximum cost per train, it means that as soon as a train has exceeded the highest delay threshold, making it additionally delayed has no additional cost,

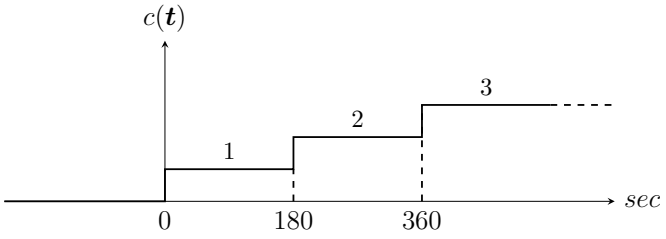


Figure 2: Example of a stepwise linear convex function considered to cost the delay (in seconds) of a generic train.

meaning that it can be left standing in a station for as long as desirable to make other trains achieve lower delays. This mechanism simulates the real-world dispatcher behavior of making a train partially 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 in sub-intervals of the type:

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

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:

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

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 constraints 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 [11]) and how we re-interpret it in the context of the TRP.

The DDD approach involves:

- 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 $D_1, D_2, \ldots$ are TI formulations (for job-shop scheduling), where the discretization periods have different sizes. The value of an optimal solution x_k^* to the k-th model in the sequence provides a lower bound on the optimal solution value to the original problem.
- a function Φ which associates a schedule (for the original problem) with solution x_k. If the schedule $\Phi(x_k^*)$ is feasible for the original problem, then it is optimal.
- a dynamic discovery mechanism that allows, when the schedule is not feasible, to refine the previous model by further discretizing time periods.

A generalized scheme of the paradigm DDD and its fundamental three steps can be seen in Figure 3.

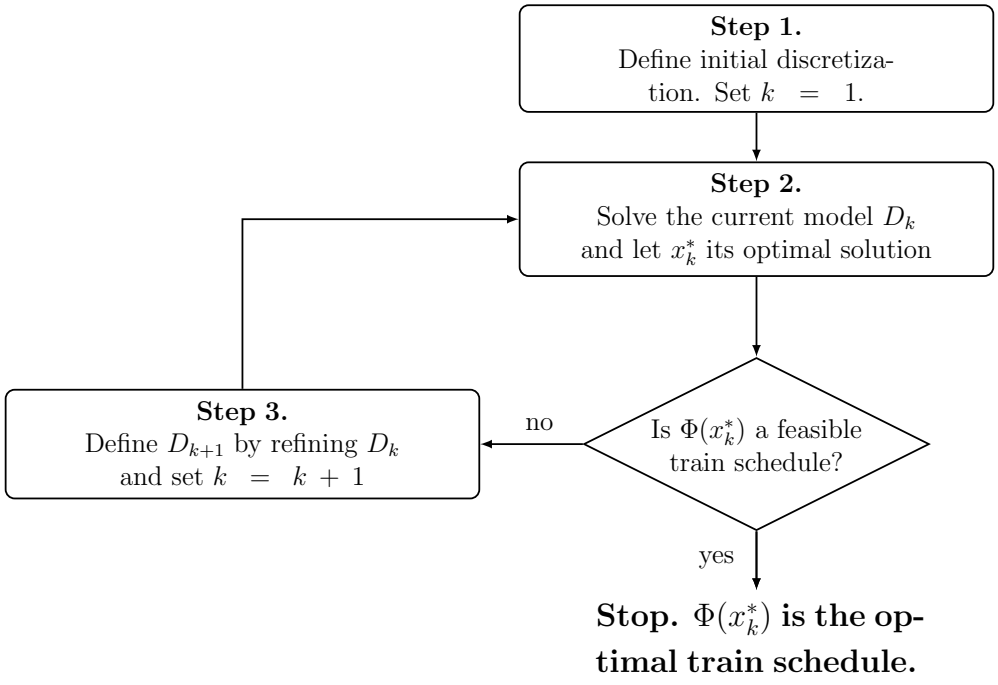


Figure 3: Generalized flowchart of the DDD paradigm.

Observe that, the optimal solution x_k^* of the current partial model D_k provides a lower bound for the original problem. If $\Phi(x_k^*)$ is not feasible for the original TRP, then we refine the time discretization at step 3 making x_k^* infeasible for the new partial model D_{k+1}. Additional mechanisms can be added to the method, such as those repairing a currently infeasible solution. Also, one can make use of models that provide both an upper and a lower bound for the optimal solution. We remark that a number of parameters, ranging from the way the new refinements are generated to the efficiency of the algorithm used to solve the partial models, affects the implementation of the DDD paradigm.

We now present a combinatorial problem, called 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 $\Lambda^{ir} = \{\lambda_1^{ir}, \lambda_2^{ir}, \ldots, \lambda_{n^{ir}}^{ir}\}$ be a partition of the feasibility interval such that

$\lambda_p^{ir} = [h_p^{ir}, h_{p+1}^{ir})$, for $1 \leq p < n^{ir}$, $h_1^{ir} = \underline{t}^{ir}$, and $\lambda_{n^{ir}}^{ir} = [h_{n^{ir}}^{ir}, M)$. Also, let $\bar{c}(\lambda_p^{ir}) = c(h_p^{ir})$ 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 $h_p^{ir} + l^r > h_{p'+1}^{iq}$.

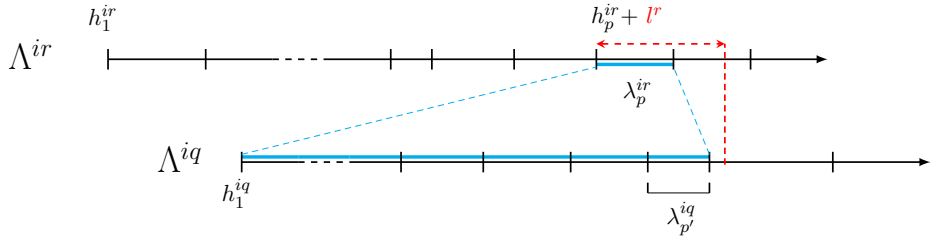
In other words, if there is no way for train i to enter r during interval λ_p^{ir} and to enter q during time interval $\lambda_{p'}^{iq}$.

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 $t^{jq} < t^{ir} + l^{rq}$.

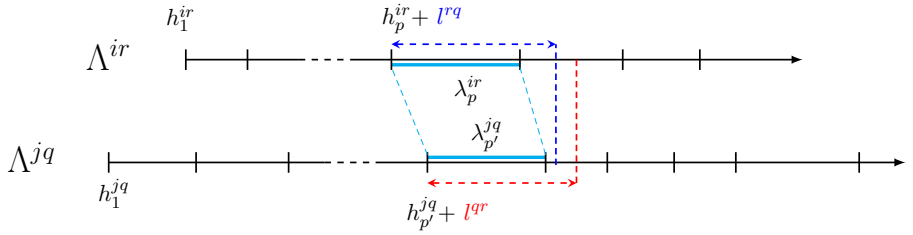
Namely, train i cannot enter track r during interval λ_p^{ir} if train j is entering track q in time interval $\lambda_{p'}^{jq}$.

Corollary 2. Given 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 $q \in R_j$, they are said *conflict incompatible* if and only if both $h_{p'}^{jq} + l^{qr} > h_{p+1}^{ir}$ and $h_p^{ir} + l^{rq} > h_{p'+1}^{jq}$.

Figure 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.



(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 $h_p^{ir} + l^{rq} > h_{p'+1}^{jq}$.

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

Remark 1. Let λ, λ' be two incompatible intervals, 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 violation of the constraints of Definition 2). In contrast, in the IAP's definition, two intervals are incompatible if all pairs in the Cartesian product of the intervals are incompatible (wrt. Definition 2).

In an attempt to construct a feasible schedule t, 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 Definition 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 $\bar{c}(S) = \sum_{\lambda \in S} \bar{c}(\lambda)$.

We can indeed formally prove the following theorem:

Theorem 1. *If the TRP admits an optimal solution t^*, then the IAP admits an optimal solution S^*, and $\bar{c}(S^*) \leq c(t^*)$.*

Proof. Let $\tilde{t}$ 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 $\Psi_\Lambda(\tilde{t})$ the complete set of intervals $\{\Psi_\Lambda(\tilde{t}^{ir}) \mid i \in I, r \in R_i\}$. We prove that $\Psi_\Lambda(\tilde{t})$ 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 $\tilde{t}$ is feasible and thus $\tilde{t}^{iq} \geq \tilde{t}^{ir} + l^r$.

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

So, $\Psi(t^*)$ is Λ-feasible. Now, since c is non-decreasing, we have that:

$$\bar{c}(\Psi(t^*)) := \sum_{i \in I} \sum_{r \in R_i} \bar{c}(\Psi(t^{ir*})) \leq \sum_{i \in I} \sum_{r \in R_i} c(t^{ir*}) = c(t^*).$$

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 Lemma 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:

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

Constraints (1) ensure that x is the incidence vector of a complete interval assignment S, whereas constraints (2) ensure 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 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 $i \in I$, $r \in R_i$, $p \in \{1, \ldots, n^{ir} - 1\}$, 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 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 for 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 $\overline{y}$. A clause $c = l_1 \vee l_2 \vee \ldots \vee l_n$ 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 $\phi = c_1 \wedge c_2 \wedge \ldots \wedge c_m$ 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 [30, 40].

It is straight-forward 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 $c = y_1 \vee y_2 \vee \ldots \vee y_k \vee \overline{y}_{k+1} \vee \ldots \vee \overline{y}_n$, where the first k literals are positive variables whereas the remaining are negated, we introduce the constraint

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

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 [41]). In the following, we will 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:

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

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 an hard clause or a conjunction of hard clauses (see [40]).

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

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

- 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 by 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 [42]). 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 [43]).

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:

$$x_p^{ir} \ + \ \sum_{\lambda_{p'}^{iq} \mid h_{p'}^{iq} < h_p^{ir} + l^r} x_{p'}^{iq} \leq 1 \qquad i \in I, \ r, q \in R_i \text{ with } r \prec_i q, \ \lambda_p^{ir} \in \Lambda_k^{ip} \ . \tag{4}$$

One can visualize an example of fixed precedence clique in Figure 4a. 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.

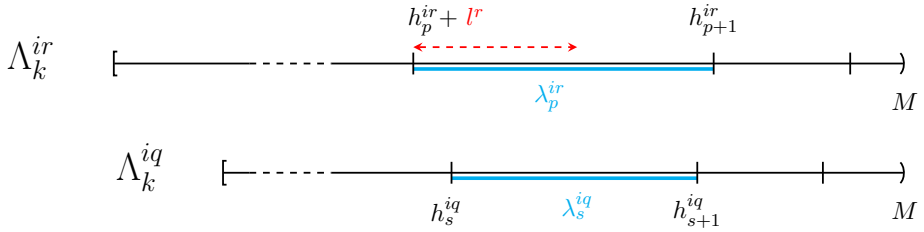
4.3. Refine the IAP (Step 3)

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.

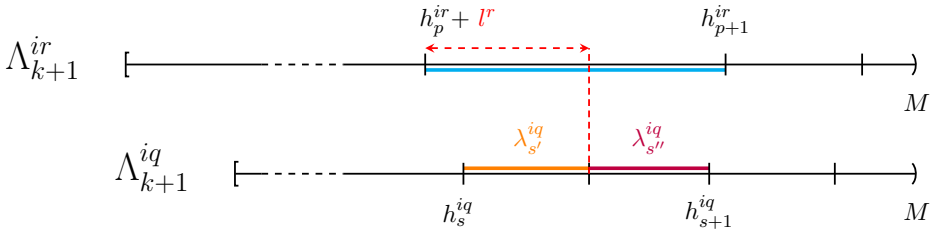
A train-rail path precedence is violated. In this case, $i = j$ and we can assume, w.l.o.g., that $r \prec_i q$. Since t^{ir} and t^{iq} violate a train-rail path precedence, then $t^{ir} + l^r > t^{iq}$. Moreover, as λ_p^{ir} and λ_s^{iq} are compatible in S_k^*, then $t^{ir} + l^r < h_{s+1}^{iq}$. 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 to x_p^{ir}. Concurrently, we also add a new fixed precedence clique relative to the interval $\lambda_{s''}^{iq}$ itself. Figure 5 shows how, starting from a violated train-rail path precedence (Figure 5a), new intervals $\lambda_{s'}^{iq}$ and $\lambda_{s''}^{iq}$ are generated (Figure 5b).

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 $\lambda_{s''}^{jq} = [t^{ir} + l^{rq}, h_{s+1}^{jq})$.



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

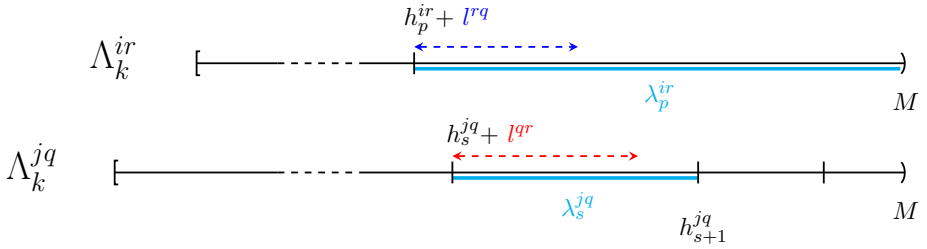


(b) Resolution of the violated train-rail path precedence shown in Figure 5a.

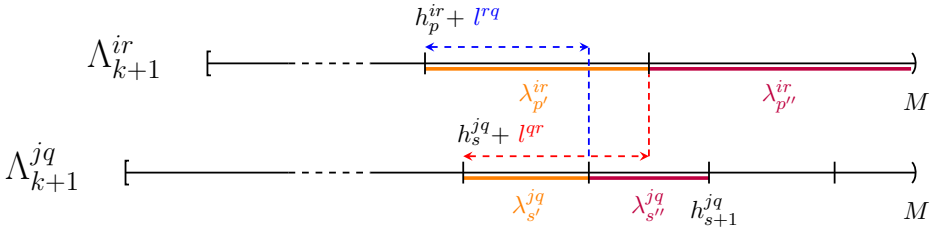
Figure 5: Example of resolution of a violated train-rail path precedence at Step 3 of the DDD-TRP.

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 Figure 6 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.



(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 6a.

Figure 6: Example of resolution of a violated disjunctive precedence at Step 3 of the DDD-TRP.

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 $\tau_k^* = \Phi(S^*)$

$\diamondsuit$***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 constraint **Then**

STOP. $\tau^* = \tau_k^*$ with value $w(\tau^*) = \bar{c}(S_k^*)$.

Refine the IAP (Step 3)

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 $E_{k+1} = E_k$

$k = k + 1$

4.4. 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 to solve the TRP can be also seen as a primal-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, Lemma 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. Lemma 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, each partition Λ^{ir} is defined by one interval (see Section 4.1). As shown in Section 4.3, at each iteration k, we add at least one interval to the current set Λ_k. Then, 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 or proving that the problem is infeasible.

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 rescheduling context and compare it with the main alternative competitor, the Big-M approach. For a complete description of this approach, we refer the reader to [44]. The full Big-M model is also reported in the Appendix A. In [44] (and in these experiments) the Big-M model is solved by row generation. In particular, once the current model is solved and a tentative schedule is generated, we check if it contains conflicts. If this is the case, specific conflict elimination constraints are added to the model, and the process iterates until a conflict-free schedule is generated. We solve the DDD-TRP using both a MILP solver and a MaxSAT solver to compare their effectiveness.

The test set consists of 24 real-life instances derived from two single-track railroad networks, later named *Line A* and *Line B*, of the Norwegian railway.

The network is small enough that all algorithms can be used to solve all the 24 instances. We created an additional 48 test instances by modifying the original 24 instances to make them more difficult. The results on these harder instances give an indication of the computational performance in situations that may occur in practice, though rarely.

5.1. Instances

The instances considered refer to portions of a physical railway network infrastructure composed by stations and tracks. 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 means of 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 C.1 in Appendix C for more details about the original 24 test instances. We emphasize that, since the instances of Line A include in most of the 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. Online train re-scheduling optimization systems will have both 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 their delay by shorter dwelling times. Secondly, we created a third set of 24 instances with increased running time for traveling 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 online (see `https://github.com/luteberget/maxsattrainscheduling`).

In the following, we will refer to the instances with the notation $\mathcal{I}_n^l$, where by $l \in \{AO, BO, AS,\ BS, AT, BT\}$ we denote the line (A,B) to which they belong, whether they are the original instance (O), have extra time added to stations (S) or tracks (T), and by the subscript $n \in \{1, ..., 12\}$ we indicate the instance number.

Objective functions. We ran the computational experiments using the objective functions defined in Section 2. In order 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

We use an x64 Intel Core i7-7700HQ CPU with 32 GB of RAM running Windows 10. The algorithms are implemented (see `https://github.com/luteberget/maxsattrainscheduling`) in Rust and uses Gurobi v9.5.0 to solve the Big-M and IAP models, and the RC2 algorithm [31] (with MiniSat v2.1.0 as the SAT backend) to solve MaxSAT problems. Both solvers were run with their default settings (note that this means that MiniSat runs single-threaded while Gurobi is able to make use of 4 processor cores). A timeout of 2 minutes was used. We highlight that all algorithms work by iteratively removing conflicts (infeasibilty) while increasing the lower bound. This also means that, when they time out, neither of the algorithms have produced any feasible (conflict-free) solution.

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 an 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 (over 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 of the IAPs.

See Tables C.2-C.4 in Appendix C for computational results on the three algorithms for each of the objective types – linear continuous, linear rounded, and stepwise – respectively.

Linear continuous objective. Looking at the Big-M formulation performance when using the linear objective, we can see how use a MaxSAT solver for the DDD-TRP is not a successful approach (see Table C.2). 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 second delay for one train in conflict with a delay of 10 minutes 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. Such detailed numerical structures, on the other hand, seem to be handled better by the MILP solver, which, however, turns out to be much slower: the MaxSAT solver reaches a timeout of two minutes for 15 instances, while the MILP solver is able to find a solution for three of them; nevertheless, the MaxSAT solver is on average 20 times faster than the MILP solver for 57 instances.

Table 1: Comparison of the computation times (in milliseconds) obtained by the the Big-M , MILP and MaxSAT approaches on the 10 hardest instances in our set

	Time (ms)			*Speed-up*	
Instance	**Big-M**	**MILP**	**MaxSAT**	**Big-M**	**MILP**
$\mathcal{I}_{12}^{AS}$	2624	T/O	553	4.7x	-
$\mathcal{I}_{11}^{AT}$	1574	T/O	300	5.2x	-
$\mathcal{I}_{8}^{AS}$	1594	T/O	186	8.6x	-
$\mathcal{I}_{12}^{AT}$	1451	T/O	285	5.1x	-
$\mathcal{I}_{1}^{AS}$	1086	T/O	249	4.4x	-
$\mathcal{I}_{11}^{AS}$	917	T/O	419	2.2x	-
$\mathcal{I}_{2}^{AS}$	596	T/O	78	7.6x	-
$\mathcal{I}_{8}^{AT}$	930	99960	181	5.1x	553.5x
$\mathcal{I}_{1}^{AT}$	367	14738	86	4.3x	171.4x
$\mathcal{I}_{2}^{AT}$	272	3863	63	4.3x	61.0x

Linear rounded objective. On the other hand, when we change the objective function to the stepwise and linear rounded forms, the picture changes completely in favor of the DDD-TRP. The results for the linear rounded objective function show that all of the problem instances that could be solved by Big-M were also solved by DDD-TRP with the MaxSAT approach (68 out of 72 instances), typically between 2x and 10x faster (see Table C.3). We also see that the MILP solver reaches the timeout for 10 instances and is always the slowest approach.

Stepwise objective. Table 1 reports for the stepwise function a comparison of the computation times (in milliseconds) obtained by the the Big-M, MILP and MaxSAT approaches on the 10 hardest instances in our set.

For the stepwise objective function (see also Table C.4 for all details), the computation times are much lower in general, and all the instances were brought below the timeout threshold for both the Big-M and MaxSAT approaches, whereas the MILP solver reaches timeout for seven instances. The MaxSAT DDD-TRP is faster than Big-M by 3x-10x on all instances, and also outperforms the MILP solver on all instances.

5.3. Discussion

We see that rounding the objective function (i.e., changing from the linear continuous objective to the linear rounded objective) does not do significant changes to the performance of the Big-M formulation, but makes a large difference to the DDD-TRP. On the other hand, the stepwise function makes all instances faster for all algorithms. We believe that this is due to the upper bound on the cost of every single train, making very congested situations resolvable by effectively parking a train at a station for a longer time.

The fact that the stepwise function makes such a difference in computation times suggests a way to extend the DDD-TRP to a primal-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 [2]. Typically instances only slightly change over time; therefore, one can massage the last instance generated at the previous resolution process and hope to produce only a few new intervals. In[45] 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 out-performs 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 are able to 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 [46]). Our DDD-TRP represents the first application of MaxSAT to train re-scheduling (some work has been done on periodic railway timetabling, see [47]). 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. Actually, the refinement of the interval between an 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. In fact, it can be shown that only a small subset of the intervals defined for the last IAP solved is actually 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 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.

Data Availability Statement

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.

Disclosure statement

The authors report there are no competing interests to declare.

References

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

[2] L. Lamorgese, C. Mannino, D. Pacciarelli, J. T. Krasemann, Train dispatching, Handbook of Optimization in the Railway Industry (2018) 265–283.

[3] A. L. Croella, V. Sasso, L. Lamorgese, C. Mannino, P. Ventura, Disruption management in railway systems by safe place assignment, Transportation Science 56 (01 2022). doi:10.1287/trsc.2021.1107.

[4] M. Fischetti, M. Monaci, Light robustness, in: Robust and online large-scale optimization, Springer, 2009, pp. 61–84.

[5] A. Mascis, D. Pacciarelli, Job-shop scheduling with blocking and no-wait constraints, European Journal of Operational Research 143 (3) (2002) 498–517.

[6] W. Fang, X. Yao, A survey on problem models and solution approaches to rescheduling in railway networks, IEEE Transactions on Intelligent Transportation Systems 16 (2015) 2997–3016. doi:10.1109/TITS.2015.2446985.

[7] S. Harrod, A tutorial on fundamental model structures for railway timetable optimization, Surveys in Operations Research and Management Science 17 (2) (2012) 85–96.

[8] M. Queyranne, A. S. Schulz, Polyhedral approaches to machine scheduling, Citeseer, 1994.

[9] C. Mannino, A. Mascis, Optimal real-time traffic control in metro stations, Operations Research 57 (4) (2009) 1026–1039.

[10] S. Harrod, Modeling network transition constraints with hypergraphs, Transportation Science 45 (1) (2011) 81–97.

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

[12] A. Bettinelli, A. Santini, D. Vigo, A real-time conflict solution algorithm for the train rescheduling problem, Transportation Research Part B: Methodological 106 (2017) 237–265.

[13] G. Caimi, M. Fuchsberger, M. Laumanns, M. Lüthi, A model predictive control approach for discrete-time rescheduling in complex central railway station areas, Computers & Operations Research 39 (11) (2012) 2578–2593.

[14] L. Meng, X. Zhou, Simultaneous train rerouting and rescheduling on an n-track network: A model reformulation with network-based cumulative flow variables, Transportation Research Part B: Methodological 67 (2014) 208–234.

[15] E. Reynolds, M. Ehrgott, S. J. Maher, A. Patman, J. Y. Wang, A multi-commodity flow model for rerouting and retiming trains in real-time to reduce reactionary delay in complex station areas, Optimization Online (2020).

[16] G. Desaulniers, J. Desrosiers, M. M. Solomon, Column generation, Vol. 5, Springer Science & Business Media, 2006.

[17] R. Gao, H. Niu, A priority-based admm approach for flexible train scheduling problems, Transportation Research Part C: Emerging Technologies 123 (2021) 102960. doi:10.1016/j.trc.2020.102960.

[18] S. Zhan, S. Wong, P. Shang, Q. Peng, J. Xie, S. Lo, Integrated railway timetable rescheduling and dynamic passenger routing during a complete blockage, Transportation Research Part B: Methodological 143 (2021) 86–123. doi:https://doi.org/10.1016/j.trb.2020.11.006.
URL https://www.sciencedirect.com/science/article/pii/S0191261520304264

[19] R. Lusby, J. Larsen, D. Ryan, M. Ehrgott, Routing trains through railway junctions: A new set-packing approach, Transportation Science 45 (2011) 228–245. doi:10.1287/trsc.1100.0362.

[20] R. M. Lusby, J. Larsen, M. Ehrgott, D. M. Ryan, A set packing inspired method for real-time junction train routing, Computers & Operations Research 40 (3) (2013) 713–724, transport Scheduling. doi:https://doi.org/10.1016/j.cor.2011.12.004.
URL https://www.sciencedirect.com/science/article/pii/S0305054811003595

[21] P. Pellegrini, G. Marlière, J. Rodriguez, Optimal train routing and scheduling for managing traffic perturbations in complex junctions, Transportation Research Part B: Methodological 59 (2014) 58–80.

[22] N. Boland, M. Hewitt, L. Marshall, M. Savelsbergh, The continuous-time service network design problem, Operations research 65 (5) (2017) 1303–1321.

[23] M. Hewitt, Enhanced dynamic discretization discovery for the continuous time load plan design problem, Transportation Science 53 (6) (2019) 1731–1750.

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

[25] Y. O. Scherr, M. Hewitt, B. A. N. Saavedra, D. C. Mattfeld, Dynamic discretization discovery for the service network design problem with mixed autonomous fleets, Transportation Research Part B: Methodological 141 (2020) 164–195.

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

[27] S. Dash, O. Günlük, A. Lodi, A. Tramontani, A time bucket formulation for the traveling salesman problem with time windows, INFORMS Journal on Computing 24 (1) (2012) 132–147.

[28] X. Wang, A. C. Regan, Local truckload pickup and delivery with hard time window constraints, Transportation Research Part B: Methodological 36 (2) (2002) 97–112.

[29] F. Leutwiler, F. Corman, A logic-based benders decomposition for microscopic railway timetable planning, European Journal of Operational Research (2022).

[30] F. Bacchus, M. Järvisalo, R. Martins, Maxsat evaluation 2018: New developments and detailed results, J. Satisf. Boolean Model. Comput. 11 (1) (2019) 99–131. doi:10.3233/SAT190119.
URL `https://doi.org/10.3233/SAT190119`

[31] A. Ignatiev, A. Morgado, J. Marques-Silva, RC2: an efficient maxsat solver, J. Satisf. Boolean Model. Comput. 11 (1) (2019) 53–64.

doi:10.3233/SAT190116.
URL https://doi.org/10.3233/SAT190116

[32] A. L. Croella, C. Mannino, P. Ventura, 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, 2021.

[33] A. L. Croella, 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 (2022).

[34] I. A. Hansen, J. Pachl, Railway timetabling & operations, Eurailpress, Hamburg (2014).

[35] N. Bešinović, R. M. Goverde, E. Quaglietta, R. Roberti, An integrated micro–macro approach to robust railway timetabling, Transportation Research Part B: Methodological 87 (2016) 14–32.

[36] L. Lamorgese, C. Mannino, M. Piacentini, Optimal train dispatching by benders' like reformulation, Transportation Science 50 (3) (2016) 910–925.

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

[38] T. Schlechte, R. Borndörfer, B. Erol, T. Graffagnino, E. Swarat, Micro–macro transformation of railway networks, Journal of Rail Transport Planning & Management 1 (1) (2011) 38–48.

[39] C. Mannino, A. Nakkerud, Optimal train rescheduling in oslo central station, Omega 116 (2022).

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

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

[42] M. Björk, Successful SAT encoding techniques, J. Satisf. Boolean Model. Comput. 7 (4) (2011) 189–201. doi:10.3233/sat190085.
URL https://doi.org/10.3233/sat190085

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

[44] L. Lamorgese, C. Mannino, An exact decomposition approach for the real-time train dispatching problem, Operations Research 63 (1) (2015) 48–64.

[45] C. Mannino, G. Sartor, An exact (re) optimization framework for real-time traffic management, optimization on line (2020).

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

[47] J. Reisch, Railway timetable optimization, Ph.D. thesis, Freie Universität Berlin (2021).
URL http://dx.doi.org/10.17169/refubium-30524

Appendix A. A basic Big-M formulation

For sake of clarity, we write here the basic Big-M formulation which is used in the experiments for comparisons. Note that we limit our model to the macroscopic representation of the network adopted in this paper and we neglect routing. All details about the model and the delayed row generation algorithm can be found in [44].

The schedule is represented by introducing, for every train $i \in I$ and every track segment $r \in R_i$, a continuous variable $t^{ir} \in \mathbb{R}$ denoting the time when i enters track segment r. We now introduce constraints and additional variables to model the feasibility conditions of Definition 2. In particular, conditions 2.i) and 2.ii) translate immediately into the following two sets of constraints

$$t^{ir} \geq \underline{t}^{ir}, \qquad i \in I,\, r \in R_i \tag{A.1}$$

$$t^{iq} \geq t^{ir} + l^r, \qquad i \in I,\, r, q \in R_i \text{ with } r \prec_i q \tag{A.2}$$

As for the conflict-free condition 2.iii), we model it by using the classic "Big-M trick". For each $i, j \in I$, $i < j$, and each $(r, q) \in \mathcal{D}_{ij}$, we introduce a binary variable $y^{ij}_{rq} \in \{0, 1\}$. We let $y^{ij}_{rq} = 1$ if i "wins" the conflict and t must satisfy $t^{jq} \geq t^{ir} + l^{rq}$; and we let $y^{ij}_{rq} = 0$ if j wins the conflict and t must satisfy $t^{ir} \geq t^{jq} + l^{qr}$. Then the *conflict resolution constraints* write as

$$t^{jq} \geq t^{ir} + l^{rq} - M(1 - y^{ij}_{rq}) \qquad (r, q) \in \mathcal{D}_{ij}, \tag{A.3}$$

$$t^{ir} \geq t^{jq} + l^{qr} - M y^{ij}_{rq} \qquad (r, q) \in \mathcal{D}_{ij}, \tag{A.4}$$

where M is a large constant.

Objective Function. In case of the linear cost objective function c, we simply want to find:

$$\min c^T t \quad .$$

For the step-wise objective function, instead, we need to introduce a binary variable z^i_s for each train $i \in I$ and each threshold d^i_s, for $s \in S = \{1, 2, \ldots, q\}$. Assume the costs of threshold violation are monotonically increasing, with $0 = c^i_1 < c^i_2 < \cdots < c^i_q$. Then we add the constraints

$$t^{if} \leq \sum_{s \in S} d_s^i z_s^i \qquad i \in I \tag{A.5}$$

$$\sum_{s \in S} z_s^i = 1 \qquad i \in I, s \in S \tag{A.6}$$

where t^{if} denotes the time train i arrives at its final destination $f \in R^i$.

And then let the objective be:

$$\min \sum_{s \in S} c_s^i z_s^i \quad . \tag{A.7}$$

Appendix B. An illustrative example

In this section we present an example application showing how the IAP graph is iteratively built by the DDD-TRP.

We are given four trains: $I = \{1, 2, 3, 4\}$ running on a railway network portion composed by seven distinct track segments: $R = \{a, b, c, d, e, f, g\}$. Trains rail paths, i.e. the ordered sequences of track segments occupied by each train, are defined as follows: $R_1 : \{a, b, g\}$, $R_2 : \{c, b\}$, $R_3 : \{d, b, f\}$ and $R_4 : \{e, f\}$. All trains have the same characteristics and priorities, therefore the time needed to traverse the track segments does not depend on the rolling stock and the train service. We have that $l = \{l^a, l^b, l^c, l^d, l^e, l^f, l^g\} = \{6, 3, 4, 9, 10, 5, 8\}$. All trains depart from their origin at time 0. For clarity of the example and w.l.o.g., the event associated with the entry of trains into stations is omitted in the following. Figure B.1 then schematically depicts the railway network topography, showing for each track segment the traversing time. Then, the graph in Figure B.2 presents the trains given rail paths (oriented black lines) and the potentially conflicting pairs of non-shareable track segments (red lines) defined by the set: $\mathcal{D} = \{\{1b, 2b\}, \{1b, 3b\}, \{2b, 3b\}, \{3f, 4f\}\}$. We simply consider the cost function in terms of scheduled departure times and for each interval λ_p^{ir} we set $\bar{c}(\lambda_p^{ir}) = c^{ir}(h_p^{ir}) = h_p^{ir}$.

Initialize the IAP. We build the initial IAP graph $G_0(V_0, E_0)$ considering for each $i \in I$ and $r \in R_i$ an initial partition composed only by the interval $\lambda_1^{ir} = \{[\underline{t}^{ir}, M)\} = \{[0, 30)\}$. We thus associate with each interval a node in V_0. Each node is referred as v_h^{ir}, where h is the lower bound h_p^{ir} of the represented interval λ_p^{ir}, and labelled with h. We initialize $E_0 = \emptyset$. In Figure

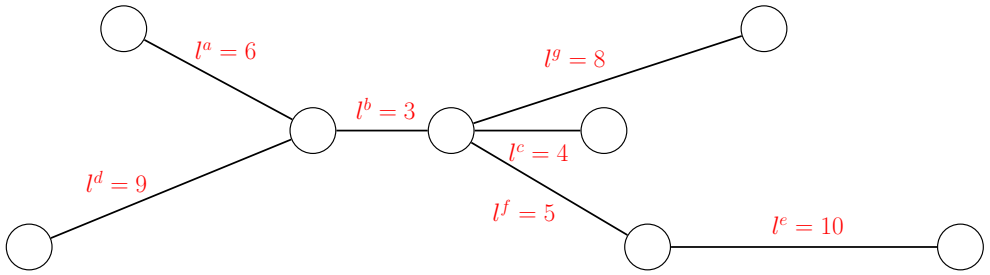


Figure B.1: Topography of the example rail network. For each track segments we report in red the time needed by all trains to traverse it.

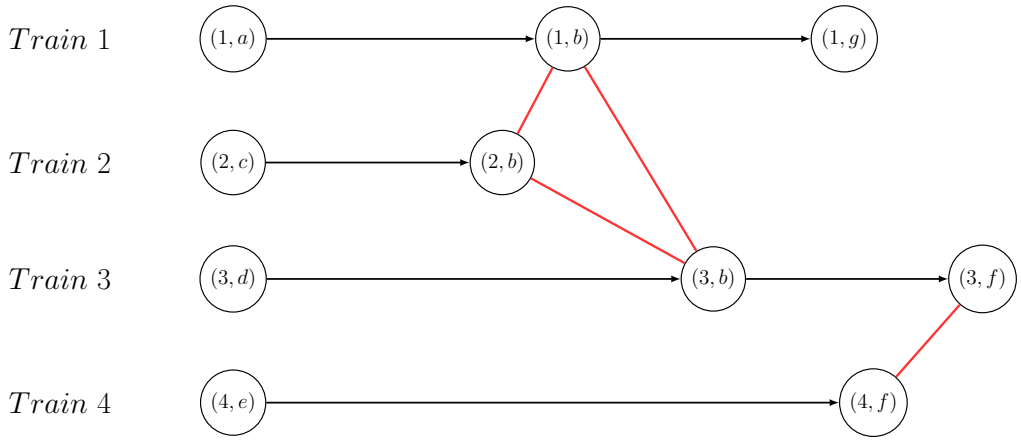


Figure B.2: Graph of the trains rail paths. Oriented black lines represents trains fixed precedence, while red edges identify the couples of potentially conflicting events.

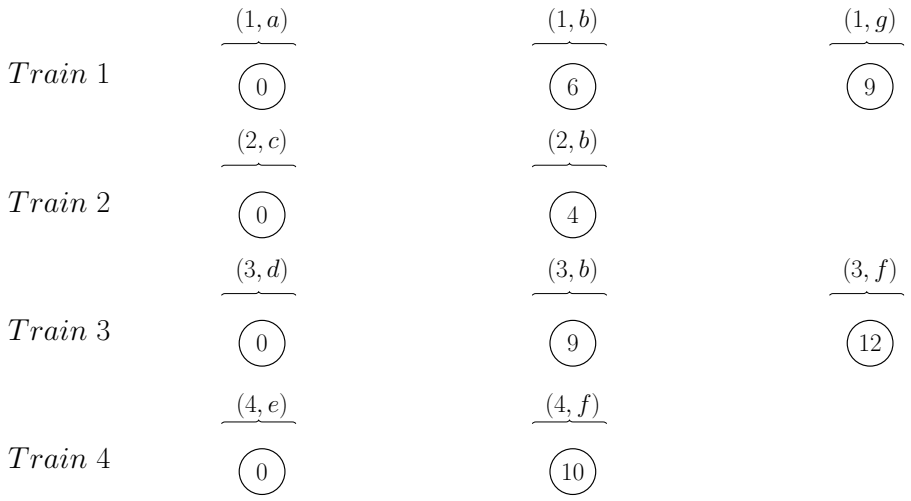
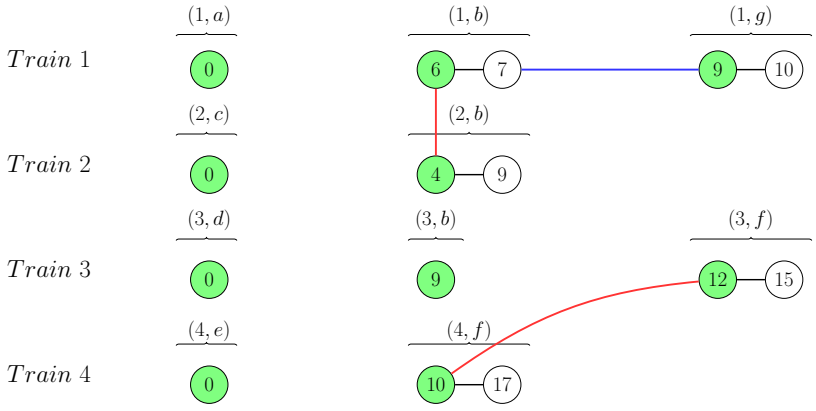


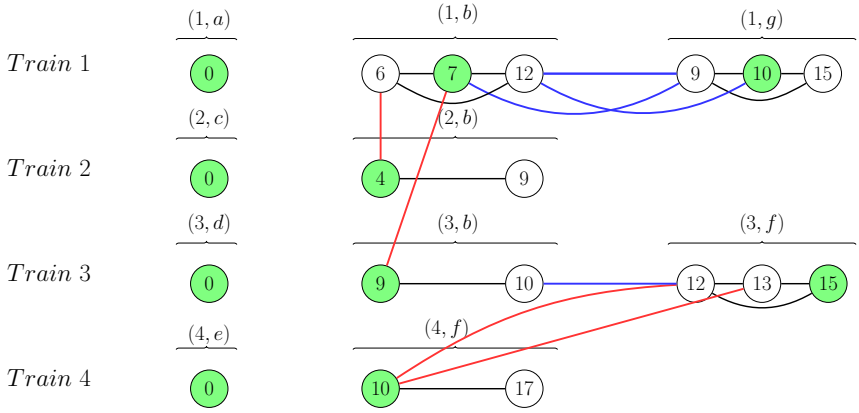
Figure B.3: Graph $G_0(V_0, E_0)$ associated with the initialization of the DDD-TRP applied to the proposed example

B.3 we report the IAP graph G_0, where each train corresponds to a different layer in the graph. Note that at the beginning of the DDD-TRP we have only isolated nodes since we define a partition set with no rail path or conflict incompatibilities (see Definition 3 and Definition 4).

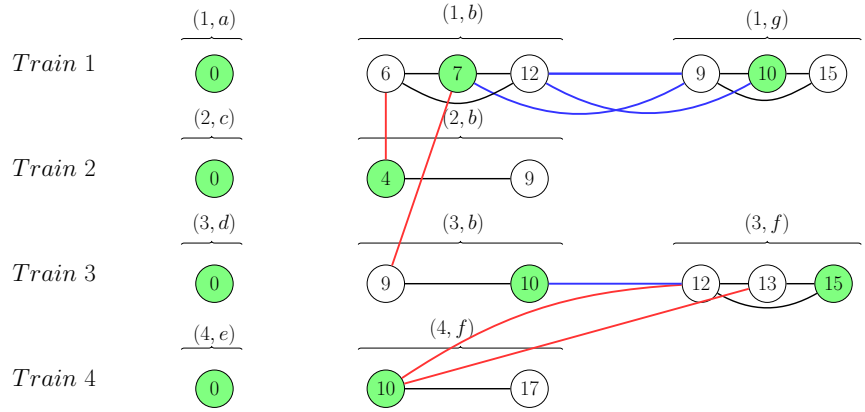
Apply the DDD-TRP. We now set $k = 1$ and apply to the example the proposed algorithm. Figure B.4a, B.4b and B.4c respectively report the IAP graphs at the end of each iteration $k = \{1, 2, 3\}$. In particular, for each train i we visually group the nodes by track segments. We draw the edges belonging to the *resource assignment clique* in black and the one belonging to the *fixed precedence clique* in blue, while the arcs referring to the disjunctive conflicts are colored in red. Finally, the nodes belonging to the optimal set computed at *Step 2* are depicted filled in light green.
Note that at termination, the set S_3^* in the IAP graph $G_3(V_3, E_3)$ does not contain any adjacent node. The solution is therefore a stable set for the graph, namely the one having the minimum cost.



(a) $G_1(V_1, E_1)$



(b) $G_2(V_2, E_2)$



(c) $G_3(V_3, E_3)$

Figure B.4: Graphs associated with the iterations of the DDD-TRP when applied to the proposed example.

DDD-TRP $1-st$ iteration

Data: $G_1(V_1, \emptyset)$, Φ, $\bar{c}$.

Solve the IAP (Step 2)

$S_1^* = V_1 = \{v_0^{1a}, v_6^{1b}, v_9^{1g}, v_0^{2c}, v_4^{2b}, v_0^{3d}, v_9^{3b}, v_{12}^{3f}, v_0^{4f}, v_{10}^{4f}\}$, $\tau_1^* = \{0, 6, 9, 0, 4, 0, 9, 12, 0, 10\}$

$\diamondsuit$***Check for solution optimality***

We find $t^{1b} = 6$ and $t^{2b} = 4$ such that $6 < 4 + 3 = 7$ and $4 < 6 + 3 = 9$ hence:

(i) we define $\lambda_1^{1b} = [6, 7)$, $\lambda_2^{1b} = [7, 30)$, and add v_7^{1b} to V_1;

(ii) we define $\lambda_1^{2b} = [4, 9)$, $\lambda_2^{2b} = [9, 30)$, and add v_9^{2b} to V_1;

(iii) we add edge $\{v_6^{1b}, v_4^{2b}\}$ to E_1.

We find $t^{3f} = 12$ and $t^{4f} = 10$ such that $12 < 10 + 5 = 15$ and $10 < 12 + 5 = 17$ hence:

(i) we define $\lambda_1^{3f} = [12, 15)$, $\lambda_2^{3f} = [15, 30)$, and add v_{15}^{3f} to V_1;

(ii) we define $\lambda_1^{4f} = [10, 17)$, $\lambda_2^{4f} = [17, 30)$, and add v_{17}^{4f} to V_1;

(iii) we add edge $\{v_{12}^{3f}, v_{10}^{4f}\}$ to E_1.

Refine the IAP (Step 3)

We select the new interval $\lambda_2^{1b} = [7, 30)$:

(i) we add edge $\{v_6^{1b}, v_7^{1b}\}$ to E_1;

(ii) we add edge $\{v_7^{1b}, v_9^{1g}\}$ to E_1;

(iv) we propagate $\lambda_2^{1b} = [7, 30)$ on track segment g: we define $\lambda_1^{1g} = [9, 10)$, $\lambda_2^{1g} = [10, 30)$, and add v_{10}^{1g} to V_1.

We select the new interval $\lambda_2^{2b} = [9, 30)$:

(i) we add edge $\{v_4^{2b}, v_9^{2b}\}$ to E_1.

We select the new interval $\lambda_2^{3f} = [15, 30)$:

(i) we add edge $\{v_9^{3f}, v_{15}^{3f}\}$ to E_1.

We select the new interval $\lambda_2^{4f} = [17, 30)$:

(i) we add edge $\{v_{17}^{3f}, v_{10}^{4f}\}$ to E_1.

We select the new interval $\lambda_2^{1g} = [10, 30)$:

(i) we add edge $\{v_9^{1g}, v_{10}^{1g}\}$ to E_1.

We set $V_2 = V_1$ and $E_2 = E_1$

$k = 2$

Data: $G_2(V_2, E_2)$, Φ, $\bar{c}$.

Solve the IAP (Step 2)

$S_2^* = \{v_0^{1a}, v_7^{1b}, v_{10}^{1g}, v_0^{2c}, v_4^{2b}, v_0^{3d}, v_9^{3b}, v_{15}^{3f}, v_0^{4f}, v_{10}^{4f}\}$, $\tau_2^* = \{0, 7, 10, 0, 4, 0, 9, 15, 0, 10\}$

$\diamondsuit$***Check for solution optimality***

We find $t^{1b} = 7$ and $t^{3b} = 9$ such that $7 < 9 + 3 = 12$ and $9 < 7 + 3 = 10$ hence:

(i) we define $\lambda_2^{1b} = [7, 12)$, $\lambda_3^{1b} = [12, 30)$, and add v_{12}^{1b} to V_2;

(ii) we define $\lambda_2^{3b} = [9, 10)$, $\lambda_3^{3b} = [10, 30)$, and add v_{10}^{3b} to V_2;

(iii) we add edge $\{v_7^{2b}, v_9^{2b}\}$ to E_2.

Refine the IAP (Step 3)

We select the new interval $\lambda_3^{1b} = [12, 30)$:

(i) we add edges $\{v_6^{1b}, v_{12}^{1b}\}$ and $\{v_7^{1b}, v_{12}^{1b}\}$ to E_2;

(ii) we add edges $\{v_{12}^{1b}, v_9^{1g}\}$ and $\{v_{12}^{1b}, v_{10}^{1g}\}$ to E_2;

(iv) we propagate t$\lambda_3^{1b} = [12, 30)$ on track segment g: we define $\lambda_2^{1g} = [10, 15)$, $\lambda_3^{1g} = [15, 30)$, and add v_{15}^{1g} to V_2.

We select the new interval $\lambda_3^{3b} = [10, 30)$:

(i) we add edges $\{v_9^{3b}, v_{10}^{3b}\}$ to E_2;

(ii) we add edges $\{v_{10}^{3b}, v_{12}^{3f}\}$ to E_2;

(iv) we propagate $\lambda_3^{3b} = [10, 30)$ on track segment f: we define $\lambda_2^{3f} = [10, 13)$, $\lambda_3^{1g} = [13, 15)$, and add v_{13}^{3f} to V_2.

We select the new interval $\lambda_3^{1g} = [15, 30)$:

(i) we add edges $\{v_9^{1g}, v_{15}^{1g}\}$ and $\{v_{10}^{1g}, v_{15}^{1g}\}$ to E_2.

We select the new interval $\lambda_3^{3f} = [13, 15)$:

(i) we add edges $\{v_{13}^{3f}, v_{12}^{3f}\}$ and $\{v_{13}^{3f}, v_{15}^{3f}\}$ to E_2;

(iii) we add edge $\{v_{13}^{3f}, v_{10}^{4f}\}$ to E_2.

We set $V_3 = V_2$ and $E_3 = E_2$
$k = 3$

DDD-TRP $3-rd$ iteration

Data: $G_3(V_3, E_3)$, Φ, $\bar{c}$.

Solve the IAP (Step 2)

$S_3^* = \{v_0^{1a}, v_7^{1b}, v_{10}^{1g}, v_0^{2c}, v_4^{2b}, v_0^{3d}, v_{10}^{3b}, v_{15}^{3f}, v_0^{4f}, v_{10}^{4f}\}$, $\tau_3^* = \{0, 7, 10, 0, 4, 0, 10, 15, 0, 10\}$

$\diamondsuit$***Check for solution optimality***

No violated disjunctive precedence

STOP. $\tau^* = \tau_3^*$ with value $\bar{c}(S_3^*) = c(\tau^*) = 56$.

Appendix C. Computational experiment full tables

Table C.1 reports some details about the original 24 test instances. For each of them, we present the number of trains ($|I|$), the total number of tracks ($|R|$), the average number of track segments per train ($\mathbb{E}[R_i]$), the total number of conflicting tracks pairs ($|\mathcal{D}|$), the number of trains having a positive delay at the "snapshot time" ($\#Dly$) and their average delay in seconds ($\mathbb{E}[d]$).

Instance	$\|I\|$	$\|R\|$	$\mathbb{E}[R_i]$	$\|\mathcal{D}\|$	# Dly	$\mathbb{E}[d]$
$\mathcal{I}_1^A$	28	33	21.46	5861	4	1200
$\mathcal{I}_2^A$	25	33	19.00	3735	9	499
$\mathcal{I}_3^A$	17	33	21.06	2061	4	376
$\mathcal{I}_4^A$	14	33	19.21	1175	6	286
$\mathcal{I}_5^A$	12	33	19.25	876	6	286
$\mathcal{I}_6^A$	8	33	18.62	307	4	406
$\mathcal{I}_7^A$	8	33	12.62	126	7	115
$\mathcal{I}_8^A$	28	33	19.21	4877	8	1640
$\mathcal{I}_9^A$	21	33	19.10	2616	7	715
$\mathcal{I}_{10}^A$	17	33	18.18	1632	9	391
$\mathcal{I}_{11}^A$	30	33	19.83	6162	6	116
$\mathcal{I}_{12}^A$	28	33	21.71	6393	5	370
$\mathcal{I}_1^B$	18	25	17.33	2022	2	228
$\mathcal{I}_2^B$	5	25	14.80	87	3	5789
$\mathcal{I}_3^B$	5	25	16.80	136	4	93
$\mathcal{I}_4^B$	18	25	16.61	1952	8	6643
$\mathcal{I}_5^B$	5	25	17.80	136	1	157
$\mathcal{I}_6^B$	6	25	12.67	93	3	61
$\mathcal{I}_7^B$	22	25	15.73	2527	11	4144
$\mathcal{I}_8^B$	5	25	13.40	72	4	110
$\mathcal{I}_9^B$	7	25	13.57	150	6	896
$\mathcal{I}_{10}^B$	8	25	11.38	130	7	697
$\mathcal{I}_{11}^B$	22	25	16.14	2674	8	7064
$\mathcal{I}_{12}^B$	14	25	16.21	970	3	242

Table C.1: Relevant statistics of test instances.

Table C.2, Table C.3 and Table C.4 show the computational results obtained by applying the DDD-TRP, using both a MILP and a MaxSAT solver, and by solving the Big-M formulation of the considered instances. Rows presenting a T/O indicate that a timeout of two minute was exceeded.

The first set of columns presents, for each instance (*Instance*), the number of iterations performed for the Big-M formulation (# *It.*), the number of violated disjunctive constraints identified in the optimality check (# *Con.*), and the time in milliseconds (*Time*) needed to find the optimal solution for the Big-M formulation. Further columns describe the performance obtained by the DDD-TRP performed with MaxSAT and MILP solvers: columns # *It.* and # *Int.* show the number of iterations, (i.e., the number of solver calls) and the total number of time intervals added (i.e., nodes in the IAP graph G_k), respectively; whereas column *Time* shows the time in milliseconds to find the optimal solution. Final set of columns (*Speed-up*) shows the relative speed-up between solvers by taking the Big-M and MILP times divided by the MaxSAT time (i.e., a speed-up > 1 means that the MaxSAT resolution was faster).

Table C.2: Algorithms computational results for the continuous linear objective function.

	Big-M			*MaxSAT*			*MILP*			*Speed-up*	
Instc	#It.	#Con.	Time	#It.	#Int.	Time	#It.	#Int.	Time	Big-M	MILP
$\mathcal{I}_1^{AO}$	5	10	120	231	2439	35	5	2107	687	3.5x	19.7x
$\mathcal{I}_2^{AO}$	4	16	90	205	2271	53	5	1823	788	1.7x	15.0x
$\mathcal{I}_3^{AO}$	4	9	43	129	1288	11	4	1214	180	3.9x	16.5x
$\mathcal{I}_4^{AO}$	2	6	20	102	1091	7	4	1046	156	2.7x	20.9x
$\mathcal{I}_5^{AO}$	3	7	23	113	969	9	3	878	82	2.6x	9.3x
$\mathcal{I}_6^{AO}$	3	5	19	93	570	3	3	544	43	6.0x	13.4x
$\mathcal{I}_7^{AO}$	2	5	8	90	593	4	2	450	35	2.0x	8.5x
$\mathcal{I}_8^{AO}$	8	59	1227	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_9^{AO}$	5	18	76	262	2431	81	6	2104	1009	0.9x	12.5x
$\mathcal{I}_{10}^{AO}$	5	17	62	218	2059	55	6	1761	1731	1.1x	31.5x
$\mathcal{I}_{11}^{AO}$	5	34	242	314	4474	1043	6	3926	6694	0.2x	6.4x
$\mathcal{I}_{12}^{AO}$	6	37	378	540	4676	16379	7	4428	44344	0.0x	2.7x
$\mathcal{I}_1^{BO}$	4	4	31	95	804	4	4	798	80	7.6x	19.8x
$\mathcal{I}_2^{BO}$	2	1	7	55	232	2	2	205	15	3.6x	7.8x
$\mathcal{I}_3^{BO}$	3	4	13	89	432	2	4	467	97	5.8x	43.5x
$\mathcal{I}_4^{BO}$	6	10	61	138	1438	18	7	1128	470	3.4x	26.2x
$\mathcal{I}_5^{BO}$	4	3	14	107	330	3	4	319	32	5.4x	12.7x
$\mathcal{I}_6^{BO}$	2	1	6	44	208	1	2	199	9	7.5x	11.5x
$\mathcal{I}_7^{BO}$	7	26	160	533	2586	11630	20	2706	64391	0.0x	5.5x
$\mathcal{I}_8^{BO}$	2	1	7	46	180	1	2	181	10	9.2x	13.1x
$\mathcal{I}_9^{BO}$	5	9	36	108	600	4	5	577	252	8.4x	59.6x

Continued on next page

Table C.2 – continued from previous page

	Big-M			*MaxSAT*			*MILP*			*Speed-up*	
Instc	#It.	#Con.	Time	#It.	#Int.	Time	#It.	#Int.	Time	Big-M	MILP
$\mathcal{I}_{10}^{BO}$	3	8	18	87	577	7	3	546	128	2.5x	18.0x
$\mathcal{I}_{11}^{BO}$	22	61	1688	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_{12}^{BO}$	2	3	17	57	792	5	2	604	25	3.2x	4.6x
$\mathcal{I}_{1}^{AT}$	5	62	364	218	5501	320	8	5139	11043	1.1x	34.5x
$\mathcal{I}_{2}^{AT}$	4	52	306	387	4452	2548	8	3951	11616	0.1x	4.6x
$\mathcal{I}_{3}^{AT}$	4	31	117	241	2672	151	6	3024	3438	0.8x	22.7x
$\mathcal{I}_{4}^{AT}$	3	16	35	115	1334	14	3	1268	185	2.6x	13.6x
$\mathcal{I}_{5}^{AT}$	4	15	59	251	1267	315	6	1400	1246	0.2x	4.0x
$\mathcal{I}_{6}^{AT}$	3	6	18	119	792	6	5	704	259	2.9x	41.5x
$\mathcal{I}_{7}^{AT}$	2	6	11	98	497	4	3	476	88	2.8x	22.0x
$\mathcal{I}_{8}^{AT}$	10	154	26064	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_{9}^{AT}$	4	39	144	391	3535	2874	5	3116	4281	0.1x	1.5x
$\mathcal{I}_{10}^{AT}$	4	18	59	176	1729	48	5	1737	980	1.2x	20.3x
$\mathcal{I}_{11}^{AT}$	-	-	T/O	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_{12}^{AT}$	-	-	T/O	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_{1}^{BT}$	3	21	46	126	1897	24	4	1864	529	1.9x	22.5x
$\mathcal{I}_{2}^{BT}$	3	6	14	60	291	2	4	353	67	9.3x	44.2x
$\mathcal{I}_{3}^{BT}$	3	7	20	64	510	3	4	553	129	6.7x	44.1x
$\mathcal{I}_{4}^{BT}$	6	36	179	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_{5}^{BT}$	4	10	26	99	748	11	6	769	582	2.4x	52.3x
$\mathcal{I}_{6}^{BT}$	2	3	7	44	297	1	2	279	14	5.9x	12.0x
$\mathcal{I}_{7}^{BT}$	7	39	397	222	2940	178	10	3264	9676	2.2x	54.4x
$\mathcal{I}_{8}^{BT}$	4	6	20	126	462	5	8	497	411	3.9x	81.5x
$\mathcal{I}_{9}^{BT}$	4	9	25	123	882	11	8	853	1051	2.2x	94.6x
$\mathcal{I}_{10}^{BT}$	2	3	8	48	294	1	2	291	10	6.9x	8.5x
$\mathcal{I}_{11}^{BT}$	4	42	133	181	2959	119	8	2911	3901	1.1x	32.8x
$\mathcal{I}_{12}^{BT}$	7	36	233	-	-	T/O	17	2618	68363	-	-
$\mathcal{I}_{1}^{AS}$	6	68	1052	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_{2}^{AS}$	7	64	1611	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_{3}^{AS}$	4	31	180	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_{4}^{AS}$	4	20	68	238	1995	193	6	2102	3211	0.3x	16.6x
$\mathcal{I}_{5}^{AS}$	4	19	61	268	1744	367	6	1780	3561	0.2x	9.7x
$\mathcal{I}_{6}^{AS}$	2	4	14	59	566	5	2	490	29	3.0x	6.2x
$\mathcal{I}_{7}^{AS}$	3	8	22	127	617	9	3	650	177	2.6x	20.6x
$\mathcal{I}_{8}^{AS}$	14	183	59400	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_{9}^{AS}$	4	40	272	-	-	T/O	8	4314	104751	-	-
$\mathcal{I}_{10}^{AS}$	5	30	148	550	2511	6436	8	2679	13204	0.0x	2.1x
$\mathcal{I}_{11}^{AS}$	-	-	T/O	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_{12}^{AS}$	-	-	T/O	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_{1}^{BS}$	3	27	114	690	2662	98667	7	2786	37884	0.0x	0.4x
$\mathcal{I}_{2}^{BS}$	3	6	19	47	276	1	3	353	58	15.1x	46.3x
$\mathcal{I}_{3}^{BS}$	2	5	7	76	533	3	3	425	64	2.5x	22.6x
$\mathcal{I}_{4}^{BS}$	6	43	324	504	3155	11349	9	3088	24349	0.0x	2.1x
$\mathcal{I}_{5}^{BS}$	6	10	35	85	719	5	6	567	316	7.6x	68.3x

Continued on next page

Table C.2 – continued from previous page

	Big-M			*MaxSAT*			*MILP*			*Speed-up*	
Instc	#It.	#Con.	Time	#It.	#Int.	Time	#It.	#Int.	Time	Big-M	MILP
$\mathcal{I}_6^{BS}$	2	3	7	50	302	1	2	283	16	5.6x	12.4x
$\mathcal{I}_7^{BS}$	5	46	406	489	3682	6322	7	3510	25814	0.1x	4.1x
$\mathcal{I}_8^{BS}$	3	5	12	89	340	3	5	377	94	4.1x	33.3x
$\mathcal{I}_9^{BS}$	4	9	25	87	661	4	5	687	205	6.6x	53.4x
$\mathcal{I}_{10}^{BS}$	2	3	8	48	303	1	2	291	16	6.7x	12.9x
$\mathcal{I}_{11}^{BS}$	5	54	406	526	3804	13018	8	3613	51810	0.0x	4.0x
$\mathcal{I}_{12}^{BS}$	7	30	255	-	-	T/O	8	1998	5180	-	-

Table C.3: Algorithms computational results for the rounded linear objective function.

	Big-M			*MaxSAT*			*MILP*			*Speed-up*	
Instc	#It.	#Con.	Time	#It.	#Int.	Time	#It.	#Int.	Time	Big-M	MILP
$\mathcal{I}_1^{AO}$	4	9	94	151	2240	21	4	1843	243	4.5x	11.5x
$\mathcal{I}_2^{AO}$	5	20	112	175	2010	23	7	2075	764	4.9x	33.2x
$\mathcal{I}_3^{AO}$	6	12	86	190	1505	20	7	1418	290	4.3x	14.3x
$\mathcal{I}_4^{AO}$	3	8	31	145	1223	12	4	1108	127	2.6x	11.0x
$\mathcal{I}_5^{AO}$	3	7	29	100	875	7	5	1140	288	4.3x	42.3x
$\mathcal{I}_6^{AO}$	3	5	21	75	545	5	4	594	53	4.4x	10.8x
$\mathcal{I}_7^{AO}$	2	5	9	89	460	3	2	450	22	3.0x	7.1x
$\mathcal{I}_8^{AO}$	8	59	800	325	7047	357	-	-	T/O	2.2x	-
$\mathcal{I}_9^{AO}$	6	21	107	175	2025	22	9	2244	1431	4.9x	65.3x
$\mathcal{I}_{10}^{AO}$	5	19	64	167	1883	23	6	1687	535	2.8x	23.3x
$\mathcal{I}_{11}^{AO}$	6	32	150	153	3427	57	6	3240	1163	2.6x	20.3x
$\mathcal{I}_{12}^{AO}$	7	40	287	222	3684	121	8	3364	2147	2.4x	17.7x
$\mathcal{I}_1^{BO}$	3	3	27	74	765	3	3	768	50	8.5x	15.5x
$\mathcal{I}_2^{BO}$	2	1	7	56	232	1	2	205	10	5.7x	7.8x
$\mathcal{I}_3^{BO}$	3	4	12	88	435	2	3	425	32	5.3x	13.7x
$\mathcal{I}_4^{BO}$	5	9	62	99	1334	11	3	968	70	5.7x	6.4x
$\mathcal{I}_5^{BO}$	3	2	10	73	289	2	2	225	12	6.4x	8.1x
$\mathcal{I}_6^{BO}$	2	1	6	17	172	1	2	199	10	7.7x	12.5x
$\mathcal{I}_7^{BO}$	6	23	101	104	2107	15	15	2684	8217	6.6x	535.9x
$\mathcal{I}_8^{BO}$	2	1	6	47	180	1	2	181	9	5.2x	7.6x
$\mathcal{I}_9^{BO}$	3	6	15	101	569	5	7	699	344	3.1x	69.7x
$\mathcal{I}_{10}^{BO}$	3	8	16	87	567	5	3	588	45	3.0x	8.4x
$\mathcal{I}_{11}^{BO}$	20	60	1122	287	3809	99	-	-	T/O	11.3x	-
$\mathcal{I}_{12}^{BO}$	2	3	15	56	743	4	2	604	24	4.1x	6.4x
$\mathcal{I}_1^{AT}$	7	76	348	198	5279	183	13	6011	11904	1.9x	65.0x
$\mathcal{I}_2^{AT}$	4	52	247	227	3953	127	8	4031	6634	1.9x	52.4x
$\mathcal{I}_3^{AT}$	7	37	217	215	2915	68	9	3034	4567	3.2x	67.6x
$\mathcal{I}_4^{AT}$	4	17	52	213	1679	30	6	1646	721	1.7x	24.3x
$\mathcal{I}_5^{AT}$	4	16	47	119	1201	11	7	1386	732	4.4x	68.8x
$\mathcal{I}_6^{AT}$	4	7	17	108	662	5	4	642	106	3.6x	22.3x
$\mathcal{I}_7^{AT}$	3	8	21	99	494	4	4	574	125	5.2x	30.5x

Continued on next page

Table C.3 – continued from previous page

	Big-M			*MaxSAT*			*MILP*			*Speed-up*	
Instc	#It.	#Con.	Time	#It.	#Int.	Time	#It.	#Int.	Time	Big-M	MILP
$\mathcal{I}_8^{AT}$	12	163	38216	349	19155	17796	-	-	T/O	2.1x	-
$\mathcal{I}_9^{AT}$	7	50	407	240	3557	100	9	3632	10200	4.1x	101.9x
$\mathcal{I}_{10}^{AT}$	4	19	56	120	1791	17	4	1537	261	3.3x	15.2x
$\mathcal{I}_{11}^{AT}$	-	-	T/O	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_{12}^{AT}$	-	-	T/O	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_1^{BT}$	4	22	63	157	1936	33	6	1940	601	1.9x	18.2x
$\mathcal{I}_2^{BT}$	3	6	17	62	291	2	4	353	62	10.9x	39.9x
$\mathcal{I}_3^{BT}$	2	5	9	55	494	2	4	517	108	3.8x	48.2x
$\mathcal{I}_4^{BT}$	7	39	227	207	3717	124	18	4252	90000	1.8x	728.4x
$\mathcal{I}_5^{BT}$	5	10	31	112	743	7	5	609	172	4.7x	25.8x
$\mathcal{I}_6^{BT}$	2	3	8	50	314	1	2	279	12	5.3x	8.3x
$\mathcal{I}_7^{BT}$	7	42	342	196	3357	124	12	3088	4896	2.8x	39.4x
$\mathcal{I}_8^{BT}$	3	5	11	106	465	3	5	385	97	3.1x	28.5x
$\mathcal{I}_9^{BT}$	4	9	23	102	830	8	7	815	576	2.9x	75.2x
$\mathcal{I}_{10}^{BT}$	2	3	38	47	294	2	2	291	15	18.2x	7.2x
$\mathcal{I}_{11}^{BT}$	6	46	310	158	2938	74	9	2861	2670	4.2x	36.3x
$\mathcal{I}_{12}^{BT}$	6	30	167	208	1939	130	18	2452	23152	1.3x	178.7x
$\mathcal{I}_1^{AS}$	8	79	1404	243	7357	329	-	-	T/O	4.3x	-
$\mathcal{I}_2^{AS}$	6	64	545	167	5235	170	-	-	T/O	3.2x	-
$\mathcal{I}_3^{AS}$	5	31	206	237	3750	96	15	4438	69993	2.1x	726.6x
$\mathcal{I}_4^{AS}$	3	16	51	182	2174	28	5	1872	1255	1.8x	44.1x
$\mathcal{I}_5^{AS}$	3	15	56	161	1635	22	8	1692	1775	2.5x	80.9x
$\mathcal{I}_6^{AS}$	2	4	14	74	625	3	2	490	21	4.1x	6.3x
$\mathcal{I}_7^{AS}$	3	8	20	142	669	7	3	650	92	2.8x	12.8x
$\mathcal{I}_8^{AS}$	13	186	59496	573	25879	37959	-	-	T/O	1.6x	-
$\mathcal{I}_9^{AS}$	6	45	424	173	3856	65	9	4082	36053	6.5x	556.2x
$\mathcal{I}_{10}^{AS}$	3	20	63	117	1776	15	6	2217	1771	4.3x	119.5x
$\mathcal{I}_{11}^{AS}$	-	-	T/O	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_{12}^{AS}$	-	-	T/O	-	-	T/O	-	-	T/O	-	-
$\mathcal{I}_1^{BS}$	3	27	115	206	2671	107	7	2626	9726	1.1x	91.0x
$\mathcal{I}_2^{BS}$	3	6	14	52	276	2	3	353	47	7.2x	23.8x
$\mathcal{I}_3^{BS}$	2	5	11	83	551	3	3	425	39	3.3x	12.0x
$\mathcal{I}_4^{BS}$	6	40	313	184	2880	73	7	2860	4078	4.3x	55.7x
$\mathcal{I}_5^{BS}$	6	10	36	78	715	5	6	567	196	6.6x	35.6x
$\mathcal{I}_6^{BS}$	2	3	10	51	303	3	2	283	20	3.7x	7.5x
$\mathcal{I}_7^{BS}$	5	50	351	244	3785	199	8	3538	15329	1.8x	77.0x
$\mathcal{I}_8^{BS}$	3	5	15	96	340	2	6	389	131	6.3x	54.5x
$\mathcal{I}_9^{BS}$	5	10	27	80	666	4	7	757	274	6.2x	63.6x
$\mathcal{I}_{10}^{BS}$	2	3	8	52	303	1	2	291	12	6.8x	10.3x
$\mathcal{I}_{11}^{BS}$	6	57	487	199	3780	156	10	3899	15290	3.1x	97.9x
$\mathcal{I}_{12}^{BS}$	6	31	190	141	1840	78	8	2084	5419	2.4x	69.9x

Table C.4: Algorithms computational results for the stepwise linear objective function.

	Big-M			*MaxSAT*			*MILP*			*Speed-up*	
Instc	#It.	#Con.	Time	#It.	#Int.	Time	#It.	#Int.	Time	Big-M	MILP
$\mathcal{I}_1^{AO}$	4	9	99	127	1858	13	3	1623	127	7.5x	9.6x
$\mathcal{I}_2^{AO}$	3	13	58	86	1578	11	5	1749	203	5.4x	19.0x
$\mathcal{I}_3^{AO}$	4	9	56	98	1289	8	4	1264	114	6.8x	13.8x
$\mathcal{I}_4^{AO}$	4	10	49	138	1042	7	5	1130	142	7.0x	20.1x
$\mathcal{I}_5^{AO}$	3	7	28	89	780	4	3	816	62	6.6x	14.7x
$\mathcal{I}_6^{AO}$	4	5	26	98	517	4	4	552	43	6.3x	10.5x
$\mathcal{I}_7^{AO}$	4	7	15	54	406	2	2	450	20	7.0x	9.0x
$\mathcal{I}_8^{AO}$	5	50	140	200	4143	71	7	3798	817	2.0x	11.5x
$\mathcal{I}_9^{AO}$	6	22	76	136	1722	14	5	1636	156	5.6x	11.4x
$\mathcal{I}_{10}^{AO}$	5	19	47	141	1427	14	5	1447	161	3.3x	11.3x
$\mathcal{I}_{11}^{AO}$	6	34	201	126	2896	32	6	2984	702	6.2x	21.6x
$\mathcal{I}_{12}^{AO}$	6	34	246	184	3178	80	7	3158	1323	3.1x	16.6x
$\mathcal{I}_1^{BO}$	2	2	21	20	673	1	2	726	63	14.7x	44.6x
$\mathcal{I}_2^{BO}$	2	1	6	56	232	2	2	205	13	3.2x	6.9x
$\mathcal{I}_3^{BO}$	3	4	13	49	345	1	3	341	23	9.6x	17.3x
$\mathcal{I}_4^{BO}$	3	6	30	105	1490	12	7	1120	145	2.5x	12.1x
$\mathcal{I}_5^{BO}$	3	2	10	73	291	3	2	225	10	3.9x	3.8x
$\mathcal{I}_6^{BO}$	2	1	8	17	172	1	2	199	11	10.5x	15.7x
$\mathcal{I}_7^{BO}$	5	22	65	131	2105	20	12	2388	933	3.3x	47.7x
$\mathcal{I}_8^{BO}$	2	1	9	47	180	1	2	181	10	13.3x	16.1x
$\mathcal{I}_9^{BO}$	3	6	14	101	575	9	5	605	77	1.6x	9.0x
$\mathcal{I}_{10}^{BO}$	5	10	25	81	617	3	3	546	32	7.4x	9.3x
$\mathcal{I}_{11}^{BO}$	7	33	73	137	2835	26	17	2315	1592	2.8x	60.8x
$\mathcal{I}_{12}^{BO}$	2	3	20	57	724	3	2	604	22	7.7x	8.7x
$\mathcal{I}_1^{AT}$	8	87	367	139	4652	86	9	6153	14738	4.3x	171.4x
$\mathcal{I}_2^{AT}$	6	69	272	181	3674	63	9	4657	3863	4.3x	61.0x
$\mathcal{I}_3^{AT}$	4	37	133	178	2857	44	8	2872	2148	3.0x	49.1x
$\mathcal{I}_4^{AT}$	4	18	73	75	1294	8	3	1500	145	8.9x	17.8x
$\mathcal{I}_5^{AT}$	7	20	77	157	1172	13	4	1294	162	5.9x	12.3x
$\mathcal{I}_6^{AT}$	4	7	19	100	634	4	4	632	79	4.5x	18.4x
$\mathcal{I}_7^{AT}$	4	9	27	73	439	4	3	524	41	7.5x	11.4x
$\mathcal{I}_8^{AT}$	11	150	930	166	9020	181	13	8609	99960	5.1x	553.5x
$\mathcal{I}_9^{AT}$	6	50	213	139	2679	32	8	3154	1213	6.6x	37.4x
$\mathcal{I}_{10}^{AT}$	5	22	75	95	1588	12	6	1617	247	6.4x	21.0x
$\mathcal{I}_{11}^{AT}$	8	166	1574	224	11609	300	-	-	T/O	5.2x	-
$\mathcal{I}_{12}^{AT}$	9	196	1451	216	11471	285	-	-	T/O	5.1x	-
$\mathcal{I}_1^{BT}$	4	21	105	110	1519	19	8	1942	785	5.5x	41.2x
$\mathcal{I}_2^{BT}$	3	6	16	58	275	3	3	325	38	5.8x	13.5x
$\mathcal{I}_3^{BT}$	2	5	12	55	447	4	4	533	84	2.7x	19.4x
$\mathcal{I}_4^{BT}$	4	28	81	97	2382	20	6	2230	516	4.1x	25.9x
$\mathcal{I}_5^{BT}$	4	8	32	127	790	7	5	655	125	4.3x	17.0x
$\mathcal{I}_6^{BT}$	2	3	8	46	305	1	2	279	11	5.2x	7.4x
$\mathcal{I}_7^{BT}$	7	42	182	105	2337	41	5	2178	385	4.4x	9.3x
$\mathcal{I}_8^{BT}$	3	5	14	87	388	4	4	371	88	3.4x	22.3x
$\mathcal{I}_9^{BT}$	6	10	62	70	754	6	6	583	251	9.8x	39.9x

Continued on next page

Table C.4 – continued from previous page

	Big-M			*MaxSAT*			*MILP*			*Speed-up*	
Instc	#It.	#Con.	Time	#It.	#Int.	Time	#It.	#Int.	Time	Big-M	MILP
$\mathcal{I}_{10}^{BT}$	2	3	7	49	294	2	2	291	10	4.0x	5.8x
$\mathcal{I}_{11}^{BT}$	5	38	188	121	2637	34	11	2707	1834	5.6x	54.5x
$\mathcal{I}_{12}^{BT}$	6	28	118	93	1320	10	5	1346	194	12.0x	19.7x
$\mathcal{I}_{1}^{AS}$	13	129	1086	244	6693	249	-	-	T/O	4.4x	-
$\mathcal{I}_{2}^{AS}$	11	117	596	181	5071	78	-	-	T/O	7.6x	-
$\mathcal{I}_{3}^{AS}$	5	44	132	215	3074	65	5	2886	1513	2.0x	23.2x
$\mathcal{I}_{4}^{AS}$	4	25	66	118	1592	14	7	1974	960	4.7x	68.4x
$\mathcal{I}_{5}^{AS}$	4	16	61	101	1388	12	5	1528	396	5.1x	33.3x
$\mathcal{I}_{6}^{AS}$	2	4	14	51	536	3	2	490	25	4.4x	7.6x
$\mathcal{I}_{7}^{AS}$	3	8	18	93	586	4	4	712	80	4.5x	20.0x
$\mathcal{I}_{8}^{AS}$	13	218	1594	222	8838	186	-	-	T/O	8.6x	-
$\mathcal{I}_{9}^{AS}$	7	54	182	146	2995	29	7	3450	1983	6.2x	67.9x
$\mathcal{I}_{10}^{AS}$	5	27	79	83	1615	10	7	2047	541	8.1x	55.8x
$\mathcal{I}_{11}^{AS}$	9	159	917	230	11802	419	-	-	T/O	2.2x	-
$\mathcal{I}_{12}^{AS}$	15	258	2624	227	13945	553	-	-	T/O	4.7x	-
$\mathcal{I}_{1}^{BS}$	4	30	96	112	1875	22	4	2040	563	4.4x	25.9x
$\mathcal{I}_{2}^{BS}$	5	9	27	58	282	1	3	339	34	19.6x	24.8x
$\mathcal{I}_{3}^{BS}$	2	5	10	81	651	3	3	425	38	3.1x	12.2x
$\mathcal{I}_{4}^{BS}$	4	33	107	108	2378	22	10	2776	3321	4.8x	150.1x
$\mathcal{I}_{5}^{BS}$	5	9	34	65	671	3	3	521	42	10.9x	13.6x
$\mathcal{I}_{6}^{BS}$	2	3	8	50	302	1	2	283	15	5.5x	10.1x
$\mathcal{I}_{7}^{BS}$	6	52	198	153	3147	54	6	3236	1649	3.7x	30.5x
$\mathcal{I}_{8}^{BS}$	3	5	16	68	329	2	3	331	26	7.2x	11.8x
$\mathcal{I}_{9}^{BS}$	8	19	71	69	500	2	5	501	68	28.7x	27.6x
$\mathcal{I}_{10}^{BS}$	5	6	25	53	318	1	5	357	44	22.1x	39.2x
$\mathcal{I}_{11}^{BS}$	6	56	232	113	2840	27	8	3347	2433	8.5x	89.4x
$\mathcal{I}_{12}^{BS}$	7	31	147	98	1599	13	7	1874	977	11.8x	78.1x