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, ensures the necessary modelling precision. By exploiting the DDD paradigm, we develop a novel approach to train dispatching and, more in general, to job-shop scheduling. The algorithm implemented represents the first application of a *Maximum SATisfiability* problem approach to the field. In our comparisons on real-life instances of train dispatching, our restricted Time-Indexed formulation solves faster on piece-wise constant objective functions, while the Big-M approach maintains the lead on linear continuous objectives.

*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)

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 [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 real-time activity is called *train rescheduling* or *train dispatching* [2] - the latter typically also includes the possibility of rail path changes. There may be different objectives, depending on the planning phase. For instance, in the strategic and tactical phase, one is interested in producing timetables maximizing some service requirement (e.g. the number of running trains [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]. Also, we mention here some competitions that have recently been set ([5], [6], [7]) for solving real-world rail scheduling problems and that gave the impulse for developing interesting solution approaches ([8]). However, the considered problems are quite different from the one we tackle here.

From the optimization theory standpoint, train scheduling is a generalization of job-shop (with blocking and no-wait constraints, [9]) 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, 10, 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 [11]: *Big-M* formulations and *Time-Indexed* (TI) formulations (see [12] for theoretical insights). In Big-M formulations, the time a train i enters a given resource r on its path is described by a continuous variable t^{ir}. In order to represent a disjunctive constraint, one must introduce a binary variable and two constraints which contain the notorious Big-M term, which is a very large coefficient. In TI formulations the planning horizon is subdivided in time intervals and a binary variable x_p^{ir} will be 1 if train i enters resource r at time p. The fact that two trains i, j cannot occupy the same resource r implies that, if train i enters a resource r at time p then, depending on the running times of the two trains, train j cannot enter r at some times q. In turn, this translates into packing constraints of the type

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

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

1. *Oversize.* When intervals are small and many, TI formulations typically contain a very large number of variables and constraints. This fact tends to slow down the solution time in each branching node by a factor which is usually (much) larger than the reduction in the tree size. In the few comparative experiments available in the literature, the comparison is by far in favour of the Big-M alternative [13].

2. *Bad approximation.* Larger and fewer intervals result in fewer variables and constraints, but poorer approximation. As a consequence, feasible solutions in the model may prove infeasible in practice on the field, or feasible field schedules may be cut off by the TI model [14, 15].

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 [16, 17, 18, 19]). In any case, because the complete formulation would be too large to handle, all these papers resort to delayed column generation procedures [20]. 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 - [21, 22]) and/or its dual problem combined with branch-and-price techniques (see [23, 24]). 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 [19] for a medium-size station (Doncaster) with the experiments over a large junction performed in [25] with a Big-M model at a microscopic network level and an almost 10-year-old technology.

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

Dynamic Discretization Discovery (DDD) is a technique to solve TI formulations recently introduced by [26] and then further extended in [27, 28, 29, 30]. 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 [31] and the method presented in [32]. We consider for each train i and given resource r a partition of the time horizon into intervals $\lambda_1, \ldots, \lambda_n$ of different widths. We hence construct an integer linear program, called Interval Assignment Problem (IAP), with binary variables x (see the forthcoming Section 3.2). Then x_p^{ir} is 1 if and only if train i enters resource r at some time $t^{ir} \in \lambda_p$ during the 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 $t^{ir} \in \lambda_p$ and $t^{jr} \in \lambda_q$, trains are in conflict on resource r. Finally, the cost of each variable is chosen in such a way that the value of the optimal solution to the DDD formulation is a lower bound on the cost of the optimal solution to the original problem.

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

Different DDD approaches differ in the way intervals are iteratively generated, how costs are defined and how feasible solutions to the original problem are constructed from the current TI solution. Although the idea of a dynamic discovery is fairly natural, there are many possible ways to implement it and engineering the algorithm is one of the most delicate phases of its design. In the present work, we present a viable implementation of the DDD for the train scheduling problem. As we will show in more detail in the following sections, such implementation fulfils all three components of the DDD paradigm, as introduced in [15]. A crucial component of our solution methodology is the way the IAP is solved at each iteration of the DDD approach. In this work, we transform it into the problem of satisfying a family of logic clauses (SAT problem). It turned out that solving IAPs incrementally by means of an open-source SAT solver is dramatically more efficient than using a state-of-the-art MILP solver. Note that similar behaviour was observed in the work of Leutwiler and Corman [33] and Matos et al. [34]. In the former, the authors authors compared a MILP and a SAT version of specific binary programs, reporting significant improvements with the SAT formulation. The latter study combines SAT and machine learning approaches to address periodic timetabling problems, outperforming state-of-the-art algorithms, including MILP and heuristics. Formulations with disjunctive precedence constraints (such as Big-M formulations) make use of continuous variables, and cannot be directly translated into a SAT problem (in [33] this problem is solved by using a Satisfiability Modulo Theories (SMT) approach). However, with a TI formulation the MILP contains only binary variables and all the constraints can be directly translated into SAT. The objective function can also be handled in a *MaxSAT* problem – the optimization version of the SAT problem.

MaxSAT solvers implemented on top of standard SAT solvers have been very successful lately [35]. In this paper, we successfully used the RC2 algorithm [36] to solve DDD formulations of the train scheduling problem.

A preliminary version of the ideas developed in this work was presented in [37] and in [38].

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

2. Time-Indexed formulation for the Train Re-scheduling Problem

The main purpose of this paper is to show how the DDD mechanism can be exploited in order to make TI formulations competitive with the classic Big-M formulation for train scheduling. To this end, in our developments and comparisons, we focus on a basic version of the problem, which corresponds to finding a plan for a *macroscopic* approximation of the rail network [39] 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., [40, 41, 42, 43]). Accordingly, in our experiments, we will make use of macroscopic real-life instances.

We look at the operational version of the train scheduling problem, also called *train dispatching* or *train rescheduling*, and, for simplicity, we do not include the possibility of rerouting. Note that the model can be extended to cope with multiple routes, for instance as in [33, 44]. In this version of the problem, we are given a *reference timetable* (i.e. the timetable which is published either for passengers or for the railway personnel), and the position of the trains at the current time. Note that trains may be delayed with respect to the reference timetable. Additionally, two trains cannot occupy the same section of track simultaneously (i.e. have a *conflict*). Depending on the subsequent scheduling decisions, particularly those involving meet or pass locations between trains, some trains may experience a reduction in delay at their destination, while others may encounter an increase. Due to the real-time nature of the problem, a solution must be computed within a very short timeframe, typically not exceeding 10 seconds. The goal is to find a conflict-free schedule for the next hours for all trains, minimizing the sum

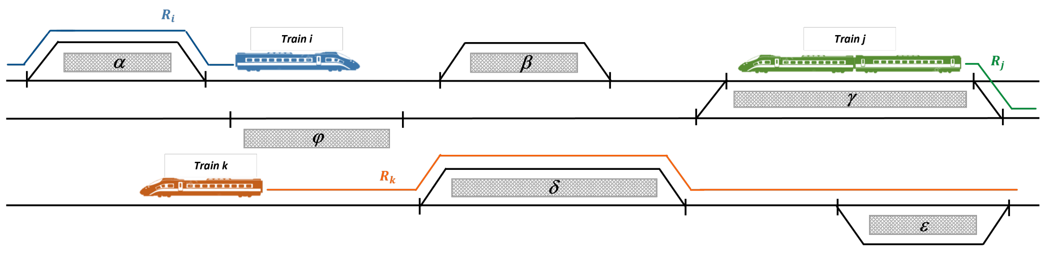


Figure 1: Schematic representation of a railway infrastructure. Stations are depicted as filled rectangles.

measure of the delays. In real-life railway systems, such a solution is typically formulated by human dispatchers, and imposed on trains by scheduling signals and switch changes, and communicated to drivers. Regardless, once a solution is implemented, trains will continue their movements, potentially deviating from the intended schedule. Consequently, a new (snapshot) instance is generated and resolved every 10 seconds.

We next introduce our main modeling choices and 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, refuelling, etc.). Tracks are further subdivided into (one or more) *tracks segments* or *block sections*, that can accommodate at most a train at a time. Tracks are either *bidirectional*, i.e. they can be traversed from in both directions or *unidirectional*. We let R be the set of all track segments.

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

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

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

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

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

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

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

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

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

ii) (*train-rail path precedences*) $t^{iq} \geq t^{ir} + l^r$, for each $i \in I$, and each distinct pair $r, q \in R_i$ with $r \prec_i q$;

iii) (*disjunctive precedences*) either $t^{jq} \geq t^{ir} + l^{rq}$ **or** $t^{ir} \geq t^{jq} + l^{qr}$, for each distinct $i, j \in I$ and each $(r, q) \in \mathcal{D}_{ij}$.

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

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

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

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

1. **Linear continuous**: $c^{ir}(t) = d^{ir}(t)$. In the Big-M formulation, this objective is implemented by introducing one continuous variable and one linear inequality for each t^{ir}.

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

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

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

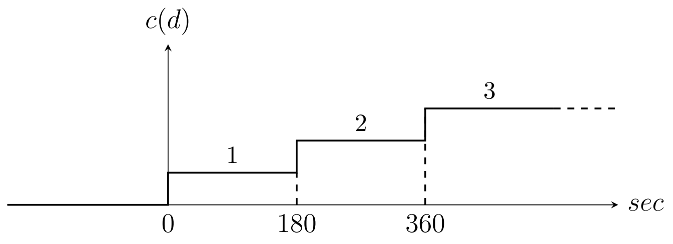


Figure 2: Example of a stepwise linear convex function considered to cost the delay (in seconds) of a generic train.

In the Big-M formulation, this objective is implemented through the introduction of one binary variable.

The use of stepwise objective functions is inspired by the official performance indicators that railway administrations use to report their punctuality, including the Norwegian railways. This can be considered to be the high-level goal of railway dispatching. Typically, small deviations are not counted in this performance indicator, and instead, punctuality is defined as the percentage of trains that were less than e.g. 5 minutes delayed.

It is also interesting to note that, since this cost function has a maximum cost per train if a train has exceeded the highest delay threshold, then any additional delay has no additional cost, and the train can hold at the station as long as necessary to reduce other trains delays. This models the real-world dispatcher behaviour of making a train cancelled in case of large traffic disruptions. In practical use, finding an optimal solution where a train has exceeded the highest delay threshold can be interpreted as a suggestion to cancel that train.

2.2. A full TI formulation

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

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

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:

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

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

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

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

3. The Dynamic Discretization Discovery method

In this section we describe the DDD paradigm (according to [15]) and how we re-interpret it in the context of the TRP. The paradigm includes:

- the construction of a sequence of (small) approximated models of the original scheduling problem, that are easier to solve than the full problem. As anticipated in the introduction, the models $D_1, D_2, \ldots$ are

TI formulations (for job-shop scheduling), where the discretization intervals have different widths. 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 with solution x_k a (possibly infeasible) schedule for the original problem. If the schedule $\Phi(x_k^*)$ is feasible for the original problem, then it is optimal.

- a heuristic mechanism which "repairs" an infeasible schedule for the original problem and returns a feasible schedule which provides an upper bound on the optimal solution value to the original problem (see, e.g. [45]).

- a dynamic discovery mechanism that allows, when the schedule is not feasible, to refine the previous model by further discretizing time intervals.

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

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

3.1. The Λ-Interval Assignment Problem

Given, for each $i \in I$ and $r \in R_i$, the *feasibility interval* $[\underline{t}^{ir}, M)$, let $\Lambda^{ir} = \{\lambda_1^{ir}, \lambda_2^{ir}, \ldots, \lambda_{n^{ir}}^{ir}\}$ 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 $\bar{c}^{ir}(\lambda_p^{ir}) = c^{ir}(h_p^{ir})$ be the cost associated with the interval λ_p^{ir}. That is,

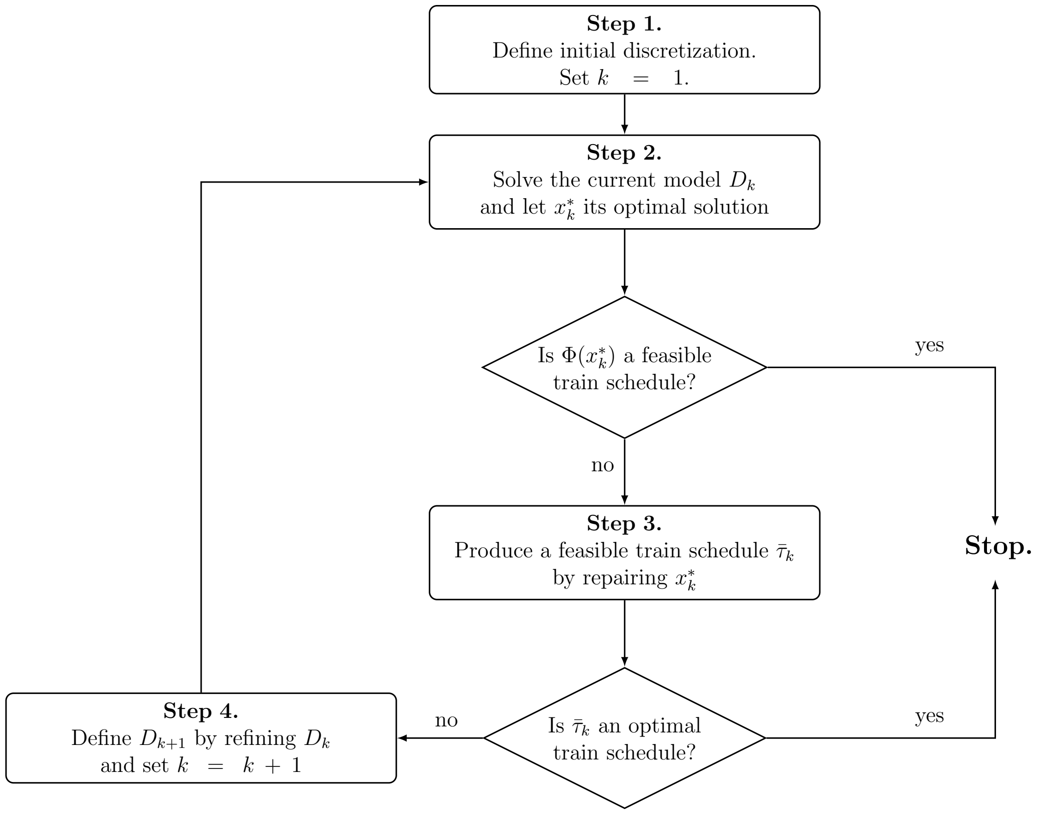

Step 1.
Define initial discretization.
Set $k = 1$.
Step 2.
Solve the current model D_k
and let x_k^* its optimal solution
Is $\Phi(x_k^*)$ a feasible train schedule?
yes
no
Step 3.
Produce a feasible train schedule $\bar{\tau}_k$ by repairing x_k^*
Stop.
Step 4.
Define D_{k+1} by refining D_k and set $k = k + 1$
no
Is $\bar{\tau}_k$ an optimal train schedule?
yes

Figure 3: Generalized flowchart of the DDD paradigm.

the cost of the interval is the cost of train i entering track segment r at the beginning of the interval. We finally let $\Lambda = \{\Lambda^{ir} : i \in I, r \in R\}$ be the set of all partitions.

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

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

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

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

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

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

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

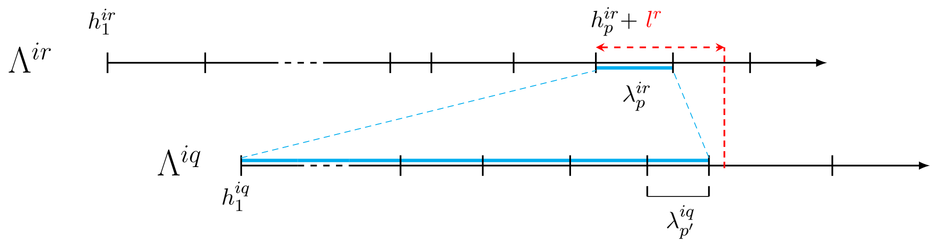
Figure 4 shows an example for the rail path incompatibility and one for the conflict incompatibility. For both (rail path and conflict) incompatibilities the following holds.

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

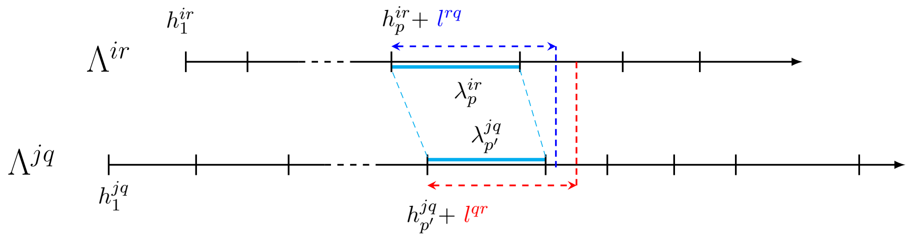
Note the difference in the definitions of interval incompatibility: in the full time-indexed formulation (TI), two intervals are incompatible exactly if the start times of the intervals are incompatible (i.e. in violation of the constraints of Definition 2). In contrast, in the IAP's definition, two intervals are incompatible if all pairs of time points in the Cartesian product of the intervals are incompatible (wrt. Definition 2).

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

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



(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).

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

We can indeed formally prove the following theorem:

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

Proof. Let $\tilde{\tau} = \{\tilde{t}^{ir} \mid i \in I, r \in R_i\}$ be a feasible schedule, and let Ψ_Λ be the function which associates the unique interval $\Psi_\Lambda(\tilde{t}^{ir}) = \lambda_p^{ir} \in \Lambda^{ir}$ such that $\tilde{t}^{ir} \in \lambda_p^{ir}$. Note that the function is well-defined, since $\tilde{t}^{ir} \in [\underline{t}^{ir}, M]$ and Λ^{ir} is a partition of $[\underline{t}^{ir}, M]$. We extend the notation by denoting with $\Psi_\Lambda(\tilde{\tau})$ the complete set of intervals $\{\Psi_\Lambda(\tilde{t}^{ir}) \mid i \in I, r \in R_i\}$. We prove that $\Psi_\Lambda(\tilde{\tau})$ is

Λ-feasible. Suppose not, then there exist two distinct intervals $\lambda_p^{ir}, \lambda_{p'}^{jq}$ which are incompatible.

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

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

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

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

□

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

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

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

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

Hence, the IAP can be formulated as follows:

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

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

Therefore, the following holds:

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

3.3. A basic MaxSAT formulation for the Interval Assignment Problem

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

A SAT problem is usually formulated in logical terms, where a Boolean variable y can take values *true* or *false*. A literal ℓ is a variable y or its negation $\overline{y}$. A clause $c = \ell_1 \vee \ell_2 \vee \dots \vee \ell_n$ contains one or more distinct variables and is satisfied if it has at least one literal assigned to *true*. A conjunctive normal form (CNF) formula $\phi = c_1 \wedge c_2 \wedge \dots \wedge c_m$ is a conjunction of clauses. Given a CNF formula, the SAT asks whether there exists an assignment to the variables that satisfies all the clauses. The optimization version of the SAT problem is the so-called *(partial) weighted MaxSAT problem*: given a CNF, we have that only a (possibly empty) subset of its clauses (*hard* clauses) must be satisfied, whereas each of the remaining (*soft*) clauses is given a weight. The goal is to find an assignment of values to the Boolean variables such that all hard clauses are satisfied and the sum of the weights of the satisfied soft clauses is maximized. Exact MaxSAT solvers have made large advances in the last decade, and are applied to industrial problems in scheduling, timetabling, decision trees, and computer hardware and software verification [35, 46].

It is straightforward to see that MILP problems are a generalization of SAT problems, but for readers who are unfamiliar with logic notation, we show here a simple transformation of a SAT problem into a MILP feasibility problem. Namely, we introduce a binary variable x_i for each Boolean variable y_i. Assigning value 1 (0) to x_i corresponds to assigning value True (False) to y_i. Next, for each clause $c = y_1 \vee y_2 \vee \ldots \vee y_k \vee \overline{y}_{k+1} \vee \ldots \vee \overline{y}_n$, 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 [47]). In the following, we show that the IAP can be converted into SAT. Indeed, we will show how to formulate the IAP as a MaxSAT problem. Consider the 0,1-LP (IAP) restated in a more compact form:

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

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

The 0,1-LP (IAP-compact) can be reduced to (partial) MaxSAT by constructing an equivalent CNF formula. In particular, each binary variable corresponds to a Boolean variable, each term in the objective function corresponds to a soft clause, and each constraint corresponds to a hard clause or a conjunction of hard clauses (see [46]).

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 with an equivalent conjunction of a set of edge packing constraints: $x_i + x_j \leq 1$, for $i, j \in Q, i \neq j$. Then, as for the constraint ii), each edge packing constraint is transformed into the clause (3).

There are, in general, multiple ways of converting various types of constraints into clause form, with different trade-offs (see [48]). 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 [49]).

3.4. A MaxSAT reformulation using lower bound variables

It is well known that reformulations of SAT and MaxSAT problems can have a huge impact on solver performance (see [48]). Even different formulations that are deemed equally strong in the linear programming sense (such as those corresponding to an affine transformation) may exhibit widely different performance characteristics. Motivated by this observation, we reformulated the MaxSAT encoding of IAP. In this reformulation, variables y_p^{ir}, corresponding to intervals $\lambda_p^{ir} \in \Lambda^{ir}$, are true not only for the selected interval but also for all preceding intervals in Λ^{ir}. This implies that interval λ_p^{ir} is selected if $y_p^{ir} = 1$ and $y_{p+1}^{ir} = 0$. The constraints for the problem are then as follows:

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

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

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

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

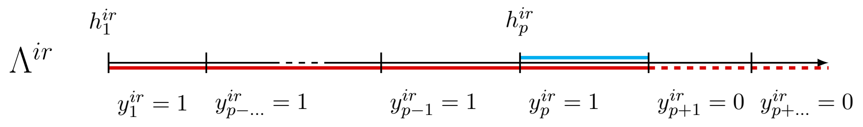


Figure 5: Lower bound variables representation for selecting interval λ_p^{ir}. All preceding intervals in Λ^{ir} are set to 1 and all subsequent intervals are set to 0.

Note that for rail path incompatibilities, this can be simplified to

$$\overline{y_p^{ir}} \vee y_{p'}^{iq}.$$

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

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

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

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

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

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

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

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

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

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:

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

One can visualize an example of a 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. Repair an infeasible schedule (Step 3)

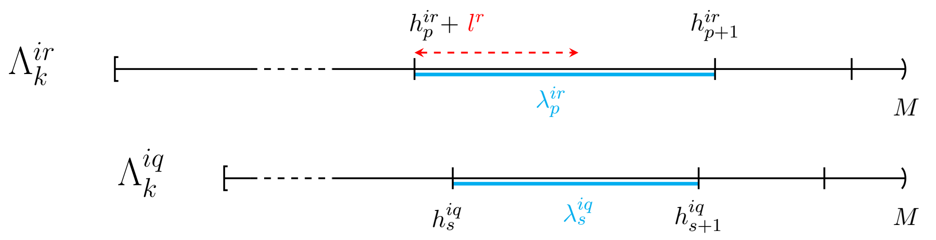
Let $\tilde{\tau} = \{\tilde{t}^{ir} : i \in I, r \in R_i\}$ be the (infeasible) schedule we want to repair to (possibly) get a feasible schedule for TRP. We build the LP program LP_0 with variables t^{ir}, lower bound constraints $t^{ir} \geq \tilde{t}^{ir} : i \in I, r \in R_i$ and all the time precedence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

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

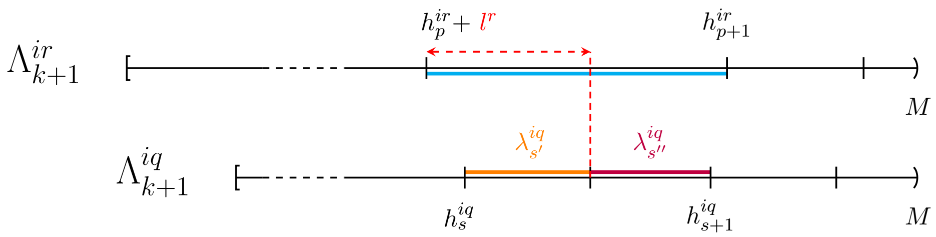
4.4. Refine the IAP (Step 4)

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

A train-rail path precedence is violated. In this case, $i = j$ and we can assume, w.l.o.g., that $r \prec_i q$. Since t^{ir} and t^{iq} violate a train-rail path precedence, then



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



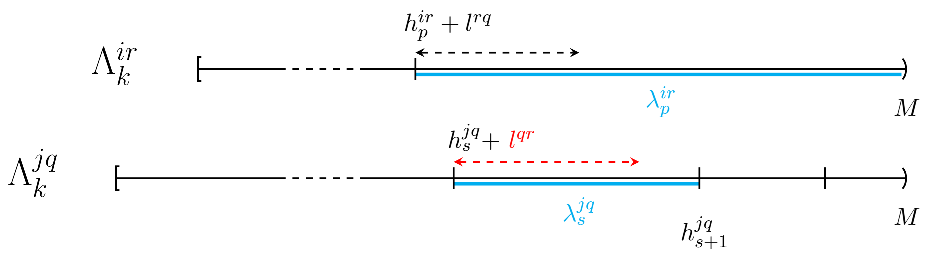
(b) Resolution of the violated train-rail path precedence shown in Figure 6a.

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

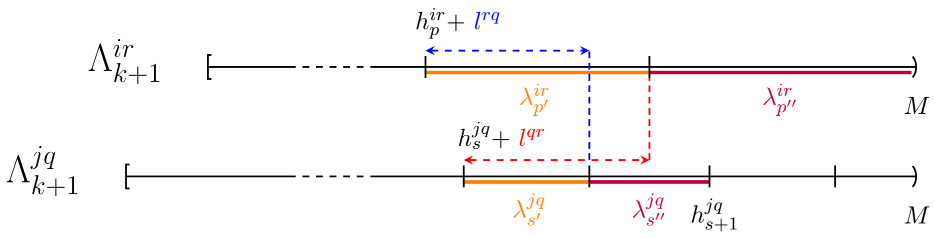
$t^{ir} + l^r > t^{iq}$. Moreover, as λ_p^{ir} and λ_s^{iq} are compatible in S_k^*, then $t^{ir} + l^r < h_{s+1}^{iq}$. Therefore, we construct Λ_{k+1} from Λ_k (and then G_{k+1} from G_k) by replacing the interval λ_s^{iq} with the two smaller intervals $\lambda_{s'}^{iq} = [h_s^{iq}, t^{ir} + l^r)$ and $\lambda_{s''}^{iq} = [t^{ir} + l^r, h_{s+1}^{iq})$.

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

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



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



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

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

breaking λ_p^{ir} into $\lambda_{p'}^{ir} = [h_p^{ir}, t^{jq} + l_{qr}^{jq})$ and $\lambda_{p''}^{ir} = [t^{jq} + l_{qr}^{jq}, h_{p+1}^{ir})$, and λ_s^{jq} into $\lambda_{s'}^{jq} = [h_s^{jq}, t^{ir} + l^{rq})$ and $\lambda_{s''}^{jq} = [t^{ir} + l^{rq}, h_{s+1}^{jq})$.

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

In both cases, once a new interval is defined, we can *propagate* the new time points discovered along the train rail path to ensure a time consistency with the intervals belonging to subsequent train track segments. With some abuse of notation in the following, given a track segment r traversed by a train i, we indicate the subsequent track segment on the rail path of i with the index $(r+1)$. Then, said $\lambda_p^{ir} = [h_p^{ir}, h_{p+1}^{ir})$ a newly generated interval and $\lambda_s^{i(r+1)} = [h_s^{i(r+1)}, h_{s+1}^{i(r+1)}) = max\{\Lambda_k^{i(r+1)} \ni \lambda_s^{i(r+1)} : h_s^{i(r+1)} < h_p^{ir} + l^r\}$, we define two new intervals of the type $\lambda_{s'}^{i(r+1)} = [h_s^{i(r+1)}, h_p^{ir} + l^r)$ and

$\lambda_{s''}^{i(r+1)} = [h_p^{ir} + l^r, h_{s+1}^{i(r+1)})$. This procedure allows us to identify a lower bound $h_{s''}^{i(r+1)}$ for $(r+1)$ that avoids the violation of the rail path constraints. The propagation is thus recursively applied on $\lambda_{s'}^{i(r+1)}$ until the final destination track segment is reached. Along with the creation of these intervals, we need to add every rail path and/or conflict incompatibility arising between the intervals in Λ_{k+1}.

Summarizing, said $G_k(V_k, E_k)$ the IAP graph at a generic iteration k, the DDD-TRP follows the steps reported below. Note that, since we apply a rail path time consistency each time we propagate a new interval, the feasibility check on the associated schedule τ_k^* can be limited to the disjunctive constraints.

DDD-TRP *k-th* iteration

```
Data: Graph $G_k(V_k := \Lambda_k, E_k)$, functions $\Phi(S) : \Lambda_k \to \mathbb{R}_+$ and $\bar{c}(S) : \Lambda_k \to \mathbb{R}_+$.
Solve the IAP (Step 2)
  Find $S^*_k$ on $G_k(V_k, E_k)$ and compute $\tau^*_k = \Phi(S^*_k)$
$\diamondsuit$ 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 $\lambda^{ir}_p \in S^*_k$ define $\lambda^{ir}_{p'} = [h^{ir}_p, t^{jq} + l^{jq}_{qr})$ and $\lambda^{ir}_{p''} = [t^{jq} + l^{jq}_{qr}, h^{ir}_{p+1})$, set $\Lambda^{ir}_k = \Lambda^{ir}_k \setminus \{\lambda^{ir}_p\} \cup \{\lambda^{ir}_{p'}, \lambda^{ir}_{p''}\}$;
        (ii)  from $\lambda^{jq}_s \in S^*_k$ define $\lambda^{jq}_{s'} = [h^{jq}_s, t^{ir} + l^{rq})$ and $\lambda^{jq}_{s''} = [t^{ir} + l^{rq}, h^{jq}_{s+1})$, set $\Lambda^{jq}_k = \Lambda^{jq}_k \setminus \{\lambda^{jq}_s\} \cup \{\lambda^{jq}_{s'}, \lambda^{jq}_{s''}\}$;
        (iii) set $E_k = E_k \cup \{\lambda^{ir}_{p'}, \lambda^{jq}_{s'}\}$.
  If $\nexists$ violated disjunctive precedence Then
    STOP. $\tau^* = \tau^*_k$ with value $c(\tau^*) = \bar{c}(S^*_k)$.
Repair the IAP solution (Step 3)
  Find $\tau_k$ by repairing $\tau^*_k$.
$\diamondsuit$ Check for solution optimality
  If $c(\tau_k) - \bar{c}(S^*_k) < \epsilon$ Then
    STOP. $\tau^* = \tau_k$ with value $c(\tau_k)$.
Refine the IAP (Step 4)
  For each newly defined interval $\lambda^{ir}_p$
      (i)   update resource assignment clique: for each $\lambda^{ir}_{p'} \in \Lambda^{ir}_k \setminus \{\lambda^{ir}_p\}$ set $E_k = E_k \cup \{\lambda^{ir}_p, \lambda^{ir}_{p'}\}$;
      (ii)  update fixed precedence clique (Definition 3): for each $\lambda^{iq}_s \in \Lambda^{iq}_k$ with $(q+1) = r$ such that $h^{iq}_s + l^{iq} > h^{ir}_{p+1}$ add a new edge $\{\lambda^{ir}_p, \lambda^{iq}_s\}$ to $E_k$;
      (iii) update conflict incompatibility (Definition 4): for each $\lambda^{iq}_s \in \Lambda^{iq}_k$ with $i \neq j$, $\{r, q\} \in \mathcal{D}$, such that $h^{ir}_{p+1} < h^{jq}_{p'} + l^{qr}$ and $h^{jq}_{p'+1} < h^{ir}_p + l^{rq}$ add a new edge $\{\lambda^{ir}_p, \lambda^{jq}_s\}$ to $E_k$;
      (iv)  propagate along the train rail path: if exists $(r+1) \in R_i$, let $\lambda^{i(r+1)}_s = max\{\Lambda^{i(r+1)}_k \ni \lambda^{i(r+1)}_s \mid h^{i(r+1)}_s < h^{ir}_p + l^r\}$, define $\lambda^{i(r+1)}_{s'} = [h^{i(r+1)}_s, h^{ir}_p + l^r)$ and $\lambda^{i(r+1)}_{s''} = [h^{ir}_p + l^r, h^{i(r+1)}_{s+1})$, set $\Lambda^{i(r+1)}_k = \Lambda^{i(r+1)}_k \setminus \{\lambda^{i(r+1)}_s\} \cup \{\lambda^{i(r+1)}_{s'}, \lambda^{i(r+1)}_{s''}\}$.
  Set $V_{k+1} = V_k$ and $E_{k+1} = E_k$
  $k = k + 1$
```

4.5. Convergence of the algorithm

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

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

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

5. Computational experiments

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

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

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

5.1. Instances

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

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

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

Secondly, we created a third set of 24 instances with increased running time for travelling from one station to the next, without adjusting the timetable accordingly, to simulate slow-downs. Such slow-downs may for example be caused by signalling equipment faults. The problem instances are available (see `https://github.com/luteberget/maxsattrainscheduling`).

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

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

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

5.2. Computational results and discussion

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

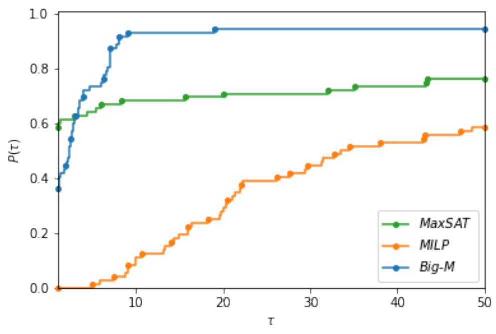
We highlight that the algorithms work by iteratively removing conflicts (infeasibility) while increasing the lower bound and using the primal repair heuristic introduced in Section 4.3 to possibly produce feasible solutions during the search. The repair heuristic is also used in the Big-M row generation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

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

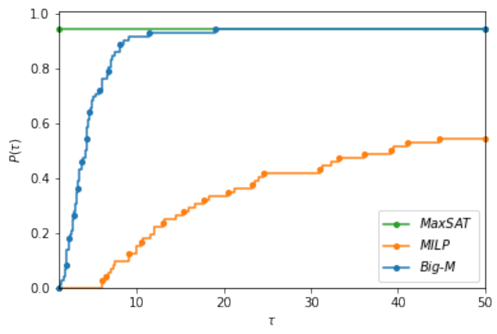
In our experiments, we observe that the number of iterations and the number of solved conflicts can vary significantly from instance to instance. Both these indicators affect the overall solution time. In fact, at each iteration k, a new IAP has to be solved and its complexity grows with the number of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

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

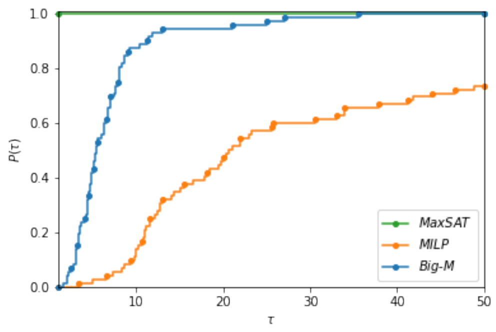
Linear continuous objective. Examining the performance of the Big-M formulation with the linear objective, it is observed that the MaxSAT solver for the DDD-TRP emerges as the fastest approach in 60% of the problems within the test sets. However, for the remaining instances, the Big-M formulation exhibits superior performance. We believe this to be a general weakness with exact weighted MaxSAT based directly on standard SAT solvers because such algorithms (including RC2) work by finding logical conflicts between subsets of components of the objective and creating cardinality constraints (linear constraints), which are costly to translate into SAT. When weights vary a lot, such as a cost of 1-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. On the contrary, the MILP solver



(a) Linear continuous objective.



(b) Linear rounded objective.



(c) Stepwise objective.

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

demonstrates superiority in handling such intricate numerical structures, successfully resolving 60 out of 72 instances. Nevertheless, despite its efficiency in solving, the MILP solver exhibits significantly slower performance, with the MaxSAT solver being on average 20 times faster for the 57 instances it successfully addressed.

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

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

		# Solved			*Avg Time (ms)*		
Obj	**Set**	**Big-M**	**MILP**	**MaxSAT**	**Big-M**	**MILP**	**MaxSAT**
Cont.	*O*	24	22	22	181	5,511	1,334
	S	22	20	19	1,285	5,893	349
	T	22	18	16	2,929	15,041	8,524
	all	**68**	**60**	**57**	**1,427**	**8,497**	**3,024**
Round.	*O*	24	22	24	132	722	34
	S	22	21	22	1,857	7,505	856
	T	22	19	22	2,902	8,501	1,787
	all	**68**	**62**	**68**	**1,587**	**5,403**	**867**
Step.	*O*	24	24	24	55	283	14
	S	24	22	24	251	5,768	49
	T	24	19	24	347	774	73
	all	**72**	**65**	**72**	**218**	**2,283**	**45**

ceeds the time limit for ten instances and consistently exhibits the highest computational time among the considered approaches.

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

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

This suggests a way to extend the DDD-TRP to a dual approach where the objective function is gradually extended from a single step to the full linear rounded objective. Simple step function objectives can be solved much faster and can provide feasible train schedules in case the exact optimal solu-

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

	Time (ms)			*Speed-up*	
Instance	**Big-M**	**MILP**	**MaxSAT**	**Big-M**	**MILP**
$\mathcal{I}_1^{AT}$	2204	8376	188	11.7x	44.6x
$\mathcal{I}_2^{AT}$	753	1997	58	12.9x	34.2x
$\mathcal{I}_8^{AT}$	7953	59085	210	37.9x	281.4x
$\mathcal{I}_{11}^{AT}$	T/O [3% gap]	108005	396	-	272.8x
$\mathcal{I}_{12}^{AT}$	14680	63967	540	27.2x	118.4x
$\mathcal{I}_1^{AS}$	10559	3901	223	47.4x	17.5x
$\mathcal{I}_2^{AS}$	474	1923	116	4.1x	16.5x
$\mathcal{I}_8^{AS}$	4099	7816	215	19.1x	36.4x
$\mathcal{I}_{11}^{AS}$	18297	11073	437	41.9x	25.3x
$\mathcal{I}_{12}^{AS}$	6137	30737	286	21.4x	107.4x

tion 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 refine the last instance generated during the previous resolution process, aiming to generate only a few new intervals. In[50] one can observe that this approach proved to be successful when extending the *Path&Cycle* formulation to cope with dynamic instances of the TRP.

6. Conclusions and future work

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

We achieved better performance using a MaxSAT solver instead of a MILP solver; we believe this is due to the way that SAT solvers can solve a sequence of incrementally constructed problem instances very efficiently. Indeed, this is crucial for many, if not most, industrial applications of SAT solvers (see [51]). Our DDD-TRP represents the first application of MaxSAT to train re-scheduling (some work has been done on periodic railway timetabling, see [52]). Going forward, we would like to investigate in more detail how MILP solvers are affected by small, incremental changes in the problem instance, and see if there is a way to make MILP solvers work efficiently with the DDD-TRP.

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

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] SBB Swiss Federal Railways - Train Schedule Optimisation Challenge, https://www.aicrowd.com/challenges/train-schedule-optimisation-challenge (2019).

[6] SBB Swiss Federal Railways - Flatland Challenge, https://www.aicrowd.com/challenges/flatland-challenge (2020).

[7] The 2023 RAS problem solving competition, https://connect.informs.org/railway-applications/new-item3/problem-solving-competition681 (2023).

[8] D. Abels, J. Jordi, M. Ostrowski, T. Schaub, A. Toletti, P. Wanko, Train scheduling with hybrid answer set programming, Theory and Practice of Logic Programming 21 (3) (2021) 317–347. doi:10.1017/S1471068420000046.

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

[10] 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.

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

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

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

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

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

[16] 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.

[17] 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.

[18] 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.

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

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

[21] 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.

[22] 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

[23] 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.

[24] 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

[25] 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.

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

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

[34] G. P. Matos, L. M. Albino, R. L. Saldanha, E. M. Morgado, Solving periodic timetabling problems with SAT and machine learning, Vol. 13, 2021, pp. 625–648. doi:10.1007/s12469-020-00244-.

[35] 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`

[36] 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`

[37] 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.

[38] 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).

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

[40] 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.

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

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

[43] 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.

[44] C. Mannino, A. Nakkerud, Optimal train rescheduling in oslo central station, Omega 116 (2023).

[45] D. M. Vu, M. Hewitt, D. D. Vu, Solving the time dependent minimum tour duration and delivery man problems with dynamic discretization discovery, European Journal of Operational Research 302 (3) (2022) 831–846. doi:https://doi.org/10.1016/j.ejor.2022.01.029.
URL `https://www.sciencedirect.com/science/article/pii/S0377221722000674`

[46] 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`

[47] 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`

[48] 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`

[49] 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`

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

[51] 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`

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

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

Appendix A. A basic Big-M formulation

For the 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 [53].

The schedule is represented by introducing, for every train $i \in I$ and every track segment $r \in R_i$, a continuous variable $t^{ir} \in \mathbb{Q}_+$ 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

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

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

where M is a large constant.

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

$$\min \sum_{i \in I} \sum_{r \in R_i} c^{ir}(t^{ir}) \quad .$$

For the step-wise objective function, instead, we need to introduce a binary variable z^{ir}_{s} for each train $i \in I$ and route $r \in R_i$ 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

$$t^{ir} \le \sum_{s\in S} d^i_s z^{ir}_s \qquad i \in I,\ r \in R_i \tag{A.5}$$

$$\sum_{s\in S} z^{ir}_s = 1 \qquad i \in I,\ r \in R_i,\ s \in S \tag{A.6}$$

And then let the objective be:

$$\min \sum_{s\in S}\sum_{i\in I}\sum_{r\in R_i} c^i_s z^{ir}_s \quad . \tag{A.7}$$

Appendix B. Formulations and solver setups for the IAP

We considered the following combinations of formulations and solvers.

Big-M. In [53] (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. A 10-minute run of the Gurobi tuning tool produced no significant difference in solving time, so we used standard Gurobi parameters.

Full TI formulation. To build the full TI formulation, we need to find reasonable values for the upper bounds of each timing variable and the resolution of the discretization. The upper bounds should not be too low and the discretization should not be too coarse to ensure that optimal solutions are not excluded. A reasonable compromise is to allow up to 1-hour delay for every event with a 10-second resolution. This would result in 360 intervals per event. Since our instances contain up to around 990 events, the full TI formulation then requires up to 356,400 binary variables.

We solved the resulting full TI MILP problems using Gurobi v9.5.0. We also solved the corresponding MaxSAT problems using the conversion described in 3.3. For at-most-one constraints, we tried both the pairwise and the bitwise encodings described in [49]. As MaxSAT solvers, we tried EvalMaxSAT 2022 and UWrMaxSat 14, both among the most highly ranked solvers in the MaxSAT Evaluation 2022 competition.

Full TI exhibited much worse performance than Big-M and DDD. For the problem instances described in Section 5, all of these TI solver setups were more than 100x slower than the Big-M and DDD approaches on average. Very few instances were solved within the time limit of 2 minutes. Since this is nowhere near competitive with the Big-M and DDD approaches, we do not report detailed results.

IAP MILP formulation (Sec. 3.2). The DDD-ALG was implemented using Gurobi v9.5.0 to solve the IAP MILP problems. A 10-minute run of the Gurobi tuning tool produced no significant difference in solving time, so we used standard Gurobi parameters.

IAP MaxSAT formulation (Sec. 3.3 and 3.4). The DDD-ALG using MaxSAT was tested in several configurations. First, we used standalone MaxSAT solvers EvalMaxSAT 2022 and UWrMaxSat 14 to solve the basic IAP MaxSAT formulation described in Sec. 3.3. We also compared with an incremental version of the same formulation where we use the IPAMIR interface[1] and incrementally change the IAP problem after each iteration of the DDD-ALG.

The lower bound formulation in Sec. 3.4 was tested using the same IPAMIR solvers, and also with a custom implementation of the RC2 MaxSAT algorithm [36] in Rust, with MiniSat v2.1.0 as the SAT solver. In this algorithm, cardinality constraints representing a lower bound on the objective value are built incrementally, and additional variables and clauses can also be added to the problem incrementally.

An overview of the MaxSAT results is shown in Table B.3. It can be seen that the solver setup with the incremental RC2 algorithm on the lower bound formulation outperformed the other setups.

Appendix C. 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: $R = \{a, b, c, d, e, f, g\}$.

[1]The IPAMIR interface for incremental MaxSAT introduced in the MaxSAT Evaluation 2022 – see `https://maxsat-evaluations.github.io/2022/incremental.html`.

Table B.3: Performance comparison of different MaxSAT solver setups for the iterated IAP problem on the stepwise objective with 3 steps.

Formulation	**Solver**	**Incremental**	**# solved**	**Avg. time (ms)**
Basic	EvalMaxSAT	No	69	1770
Basic	UWrMaxSat	No	70	2798
Basic	EvalMaxSAT	Yes	64	907
Basic	UWrMaxSat	Yes	67	1989
Lower bound	EvalMaxSAT	Yes	**72**	221
Lower bound	UWrMaxSat	Yes	**72**	344
Lower bound	RC2	Yes	**72**	**20**

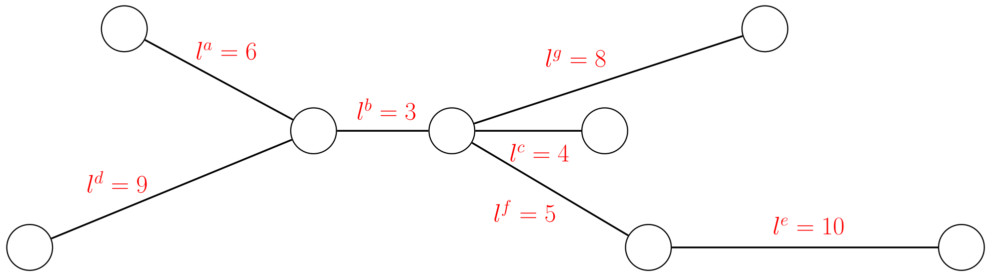


Figure C.1: Topography of the example rail network. For each track segment, we report in red the time needed by all trains to traverse it.

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 C.1 then schematically depicts the railway network topography, showing for each track segment the traversing time. Then, the graph in Figure C.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 $\bar{c}^{ir}(\lambda_p^{ir}) = c^{ir}(h_p^{ir}) = h_p^{ir}$.

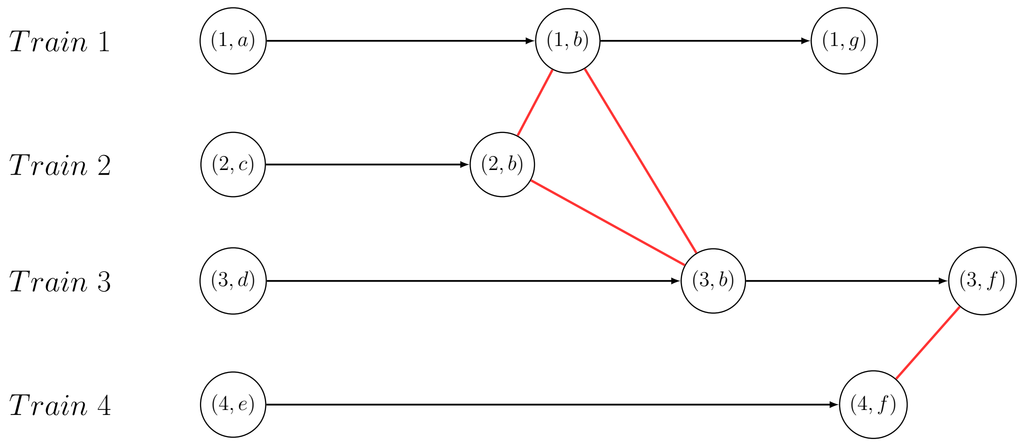


Figure C.2: Graph of the trains rail paths. Oriented black lines represent trains fixed precedence, while red edges identify the couples of potentially conflicting events.

Initialize the DDD-TRP *(Step 1).* 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 C.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 *(Step 2 and Step 4).* We set $k = 1$ and iteratively apply Step 2 and Step 4 to the example. Figure C.4a, C.4b and C.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.

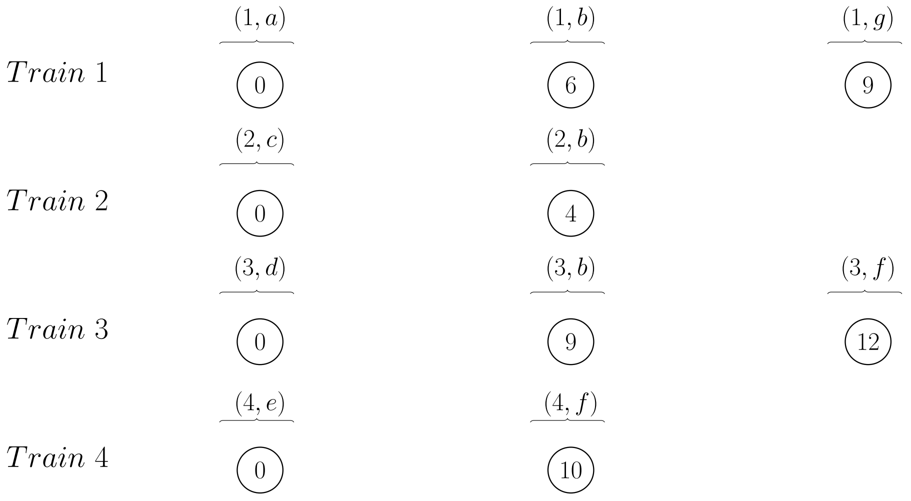
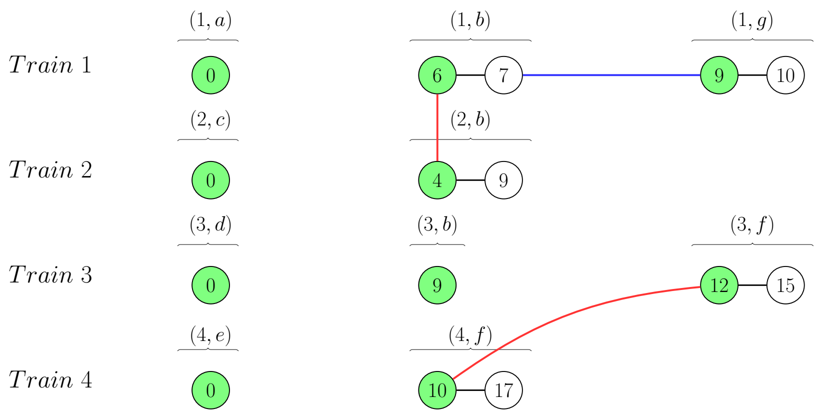
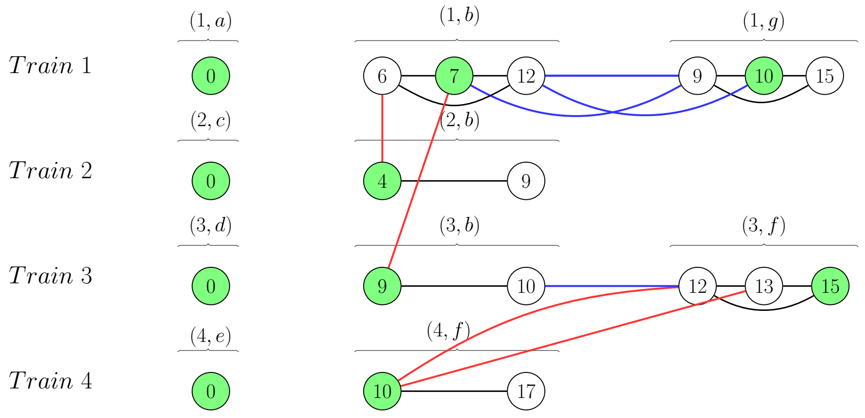


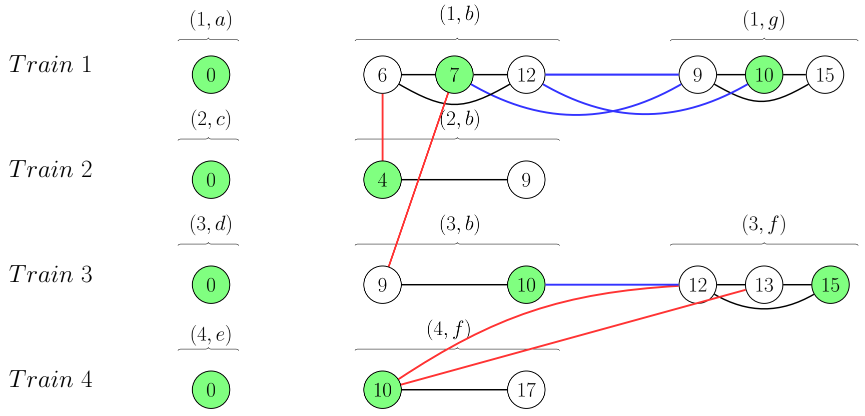
Figure C.3: Graph $G_0(V_0, E_0)$ associated with the initialization (Step 1) of the DDD-TRP.



(a) $G_1(V_1, E_1)$



(b) $G_2(V_2, E_2)$



(c) $G_3(V_3, E_3)$

Figure C.4: Graphs associated with the iterations of the DDD-TRP.

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 4)

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 $E_2 = E_1$
$k = 2$

DDD-TRP $2-nd$ iteration

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 4)
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 $c(\tau^*) = \bar{c}(S_3^*) = 56$.

Appendix D. Computational experiment full tables

Table D.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 D.1: Relevant statistics of test instances.

Table D.2, Table D.3 and Table D.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. Dashes indicate that the solver timeout of two minutes was exceeded, and the percentages in the corresponding *Time* columns show the remaining gap ($\frac{\text{ub}-\text{lb}}{\text{ub}}$) after two minutes.

The first set of columns presents, for each instance (*Inst.*), 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. The 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 D.2: Computational results for the continuous linear objective function.

	Big-M			*MaxSAT*			*MILP*			*Speed-up*	
Inst.	#It.	#Con.	Time	#It.	#Int.	Time	#It.	#Int.	Time	Big-M	MILP
$\mathcal{I}_1^{AO}$	5	10	137	231	2678	56	5	2107	596	2.5x	10.7x
$\mathcal{I}_2^{AO}$	4	16	92	222	2188	152	5	1823	640	0.6x	4.2x
$\mathcal{I}_3^{AO}$	4	9	58	136	1295	15	4	1214	196	3.9x	13.3x
$\mathcal{I}_4^{AO}$	2	6	26	106	1089	14	4	1046	157	1.9x	11.4x
$\mathcal{I}_5^{AO}$	3	7	30	104	933	11	3	878	81	2.8x	7.6x
$\mathcal{I}_6^{AO}$	3	5	23	93	532	6	3	544	48	4.1x	8.5x
$\mathcal{I}_7^{AO}$	2	5	13	90	584	6	2	450	43	2.0x	7.0x
$\mathcal{I}_8^{AO}$	8	59	1563	-	-	[8%]	-	-	[1.6%]	-	-
$\mathcal{I}_9^{AO}$	5	18	85	251	2533	92	6	2104	1014	0.9x	11.0x
$\mathcal{I}_{10}^{AO}$	5	17	64	230	2158	81	6	1761	1195	0.8x	14.7x
$\mathcal{I}_{11}^{AO}$	5	34	187	218	4187	185	6	3926	7352	1.0x	39.7x
$\mathcal{I}_{12}^{AO}$	6	37	548	562	4644	35227	7	4428	45607	0.0x	1.3x
$\mathcal{I}_1^{BO}$	4	4	39	95	804	9	4	798	75	4.2x	8.0x
$\mathcal{I}_2^{BO}$	2	1	9	55	233	3	2	205	13	2.9x	4.2x
$\mathcal{I}_3^{BO}$	3	4	14	88	432	6	4	467	107	2.5x	18.7x
$\mathcal{I}_4^{BO}$	6	10	71	128	1570	28	7	1128	393	2.5x	14.0x
$\mathcal{I}_5^{BO}$	4	3	16	107	330	5	4	319	39	3.3x	8.4x
$\mathcal{I}_6^{BO}$	2	1	8	44	208	2	2	199	12	4.5x	6.7x

Continued on next page

Table D.2 – continued from previous page

	Big-M			*MaxSAT*			*MILP*			*Speed-up*	
Inst.	#It.	#Con.	Time	#It.	#Int.	Time	#It.	#Int.	Time	Big-M	MILP
$\mathcal{I}_7^{BO}$	7	26	133	496	2763	10706	22	2858	114505	0.0x	10.7x
$\mathcal{I}_8^{BO}$	2	1	9	46	180	2	2	181	9	4.9x	4.8x
$\mathcal{I}_9^{BO}$	5	9	37	112	607	12	5	577	239	3.1x	20.1x
$\mathcal{I}_{10}^{BO}$	3	8	19	84	525	6	3	546	100	3.3x	17.7x
$\mathcal{I}_{11}^{BO}$	23	61	766	-	-	[0.2%]	-	-	[0.2%]	-	-
$\mathcal{I}_{12}^{BO}$	2	3	21	57	800	6	2	604	27	3.7x	4.9x
$\mathcal{I}_1^{AT}$	4	61	343	248	5474	446	7	5097	12691	0.8x	28.5x
$\mathcal{I}_2^{AT}$	4	52	231	359	4171	1606	8	3951	26138	0.1x	16.3x
$\mathcal{I}_3^{AT}$	4	31	91	247	2659	169	6	3024	1979	0.5x	11.7x
$\mathcal{I}_4^{AT}$	3	16	37	112	1294	19	3	1268	152	1.9x	7.9x
$\mathcal{I}_5^{AT}$	4	15	50	260	1236	444	5	1326	513	0.1x	1.2x
$\mathcal{I}_6^{AT}$	3	6	20	101	733	7	5	704	195	2.7x	27.0x
$\mathcal{I}_7^{AT}$	2	6	18	95	508	7	3	476	75	2.6x	11.2x
$\mathcal{I}_8^{AT}$	10	156	96220	-	-	[44%]	-	-	[21%]	-	-
$\mathcal{I}_9^{AT}$	4	40	122	368	3827	1200	6	3116	6749	0.1x	5.6x
$\mathcal{I}_{10}^{AT}$	4	18	57	165	1823	53	4	1709	443	1.1x	8.4x
$\mathcal{I}_{11}^{AT}$	-	-	[30%]	-	-	[79%]	-	-	[50%]	-	-
$\mathcal{I}_{12}^{AT}$	-	-	[38%]	-	-	[79%]	-	-	[54%]	-	-
$\mathcal{I}_1^{BT}$	3	21	49	127	1915	32	4	1864	525	1.5x	16.2x
$\mathcal{I}_2^{BT}$	3	6	14	60	302	4	4	353	74	3.8x	20.7x
$\mathcal{I}_3^{BT}$	3	7	18	73	499	6	4	553	120	3.3x	21.7x
$\mathcal{I}_4^{BT}$	6	36	138	-	-	[0.3%]	-	-	[0.1%]	-	-
$\mathcal{I}_5^{BT}$	4	10	32	101	787	10	6	769	429	3.3x	44.3x
$\mathcal{I}_6^{BT}$	2	3	10	52	268	2	2	279	15	4.2x	6.2x
$\mathcal{I}_7^{BT}$	7	39	220	251	2962	360	10	3264	7658	0.6x	21.3x
$\mathcal{I}_8^{BT}$	4	6	19	144	446	16	8	497	312	1.2x	19.7x
$\mathcal{I}_9^{BT}$	4	9	24	155	946	23	8	853	664	1.1x	29.0x
$\mathcal{I}_{10}^{BT}$	2	3	11	48	294	2	2	291	11	4.9x	5.0x
$\mathcal{I}_{11}^{BT}$	4	42	127	274	3159	399	8	2911	3567	0.3x	8.9x
$\mathcal{I}_{12}^{BT}$	7	35	171	-	-	[2.3%]	17	2582	43734	-	-
$\mathcal{I}_1^{AS}$	6	68	6720	-	-	[15%]	-	-	[3%]	-	-
$\mathcal{I}_2^{AS}$	7	64	3614	-	-	[20%]	-	-	[2.1%]	-	-
$\mathcal{I}_3^{AS}$	4	31	123	-	-	[4%]	11	4240	83387	-	-
$\mathcal{I}_4^{AS}$	4	20	68	193	2036	69	6	2102	1965	1.0x	28.5x
$\mathcal{I}_5^{AS}$	5	20	70	307	1741	964	6	1780	1864	0.1x	1.9x
$\mathcal{I}_6^{AS}$	2	4	13	62	543	4	2	490	27	3.2x	6.3x
$\mathcal{I}_7^{AS}$	3	8	20	112	586	6	3	650	173	3.2x	26.8x
$\mathcal{I}_8^{AS}$	-	-	[8%]	-	-	[52%]	-	-	[15%]	-	-
$\mathcal{I}_9^{AS}$	4	40	252	-	-	[9%]	8	4314	48137	-	-
$\mathcal{I}_{10}^{AS}$	5	30	107	565	2393	9505	8	2679	11298	0.0x	1.2x
$\mathcal{I}_{11}^{AS}$	-	-	[34%]	-	-	[86%]	-	-	[47%]	-	-
$\mathcal{I}_{12}^{AS}$	-	-	[31%]	-	-	[84%]	-	-	[44%]	-	-
$\mathcal{I}_1^{BS}$	3	27	111	722	2732	102049	7	2786	14524	0.0x	0.1x
$\mathcal{I}_2^{BS}$	3	6	13	49	290	3	3	353	48	5.1x	19.3x

Continued on next page

Table D.2 – continued from previous page

	Big-M			*MaxSAT*			*MILP*			*Speed-up*	
Inst.	#It.	#Con.	Time	#It.	#Int.	Time	#It.	#Int.	Time	Big-M	MILP
$\mathcal{I}_3^{BS}$	2	5	10	75	512	5	3	425	52	2.2x	10.7x
$\mathcal{I}_4^{BS}$	6	43	301	588	3495	18823	10	3116	30446	0.0x	1.6x
$\mathcal{I}_5^{BS}$	6	10	33	107	880	13	6	567	236	2.6x	18.8x
$\mathcal{I}_6^{BS}$	2	3	9	46	303	2	2	283	14	3.8x	5.7x
$\mathcal{I}_7^{BS}$	5	46	905	463	3808	6941	7	3510	23191	0.1x	3.3x
$\mathcal{I}_8^{BS}$	3	5	14	95	351	6	5	377	146	2.4x	25.9x
$\mathcal{I}_9^{BS}$	4	9	34	102	717	12	5	687	184	2.7x	15.0x
$\mathcal{I}_{10}^{BS}$	2	3	12	48	289	2	2	291	16	4.7x	6.7x
$\mathcal{I}_{11}^{BS}$	5	52	1286	520	3744	22565	8	3613	30118	0.1x	1.3x
$\mathcal{I}_{12}^{BS}$	7	30	131	-	-	[11%]	9	2040	7585	-	-

Table D.3: Computational results for the rounded linear objective function.

	Big-M			*MaxSAT*			*MILP*			*Speed-up*	
Inst.	#It.	#Con.	Time	#It.	#Int.	Time	#It.	#Int.	Time	Big-M	MILP
$\mathcal{I}_1^{AO}$	4	9	115	150	2252	35	4	1921	308	3.3x	8.8x
$\mathcal{I}_2^{AO}$	6	23	130	156	2061	28	6	2031	614	4.7x	22.2x
$\mathcal{I}_3^{AO}$	9	16	133	226	1612	33	6	1368	307	4.1x	9.5x
$\mathcal{I}_4^{AO}$	3	7	40	148	1130	18	4	1108	162	2.2x	8.9x
$\mathcal{I}_5^{AO}$	4	9	42	100	861	10	3	816	74	4.2x	7.5x
$\mathcal{I}_6^{AO}$	5	6	45	89	531	7	4	594	69	6.7x	10.2x
$\mathcal{I}_7^{AO}$	2	5	13	91	447	8	2	450	32	1.8x	4.3x
$\mathcal{I}_8^{AO}$	8	61	1354	279	6746	364	11	6578	62147	3.7x	170.9x
$\mathcal{I}_9^{AO}$	6	23	131	232	2564	79	6	1844	777	1.7x	9.8x
$\mathcal{I}_{10}^{AO}$	4	16	72	153	1675	29	7	1645	600	2.5x	20.9x
$\mathcal{I}_{11}^{AO}$	4	29	142	178	3526	98	6	3272	1331	1.4x	13.6x
$\mathcal{I}_{12}^{AO}$	7	37	725	226	3291	129	7	3322	5581	5.6x	43.4x
$\mathcal{I}_1^{BO}$	3	3	47	74	775	8	4	798	101	6.1x	13.3x
$\mathcal{I}_2^{BO}$	2	1	11	56	233	3	2	205	11	4.0x	4.1x
$\mathcal{I}_3^{BO}$	3	4	22	86	435	6	3	425	53	3.4x	8.4x
$\mathcal{I}_4^{BO}$	6	10	85	99	1342	19	3	968	89	4.4x	4.6x
$\mathcal{I}_5^{BO}$	3	2	21	73	283	2	2	225	12	8.4x	5.0x
$\mathcal{I}_6^{BO}$	2	1	10	17	172	1	2	199	12	10.1x	12.7x
$\mathcal{I}_7^{BO}$	6	23	120	119	2330	37	7	2152	1814	3.3x	49.3x
$\mathcal{I}_8^{BO}$	2	1	11	47	180	1	2	181	9	7.5x	6.2x
$\mathcal{I}_9^{BO}$	4	9	27	93	545	10	4	525	137	2.6x	13.6x
$\mathcal{I}_{10}^{BO}$	3	8	24	92	554	11	3	588	63	2.3x	5.9x
$\mathcal{I}_{11}^{BO}$	16	60	588	318	3837	146	-	-	[0.2%]	4.0x	-
$\mathcal{I}_{12}^{BO}$	2	3	22	57	757	6	2	604	31	3.5x	5.0x
$\mathcal{I}_1^{AT}$	5	74	581	196	5068	211	9	5593	6147	2.8x	29.1x
$\mathcal{I}_2^{AT}$	4	52	634	263	4157	230	7	3981	10802	2.8x	47.0x
$\mathcal{I}_3^{AT}$	4	32	161	293	3143	140	6	2692	1324	1.2x	9.5x
$\mathcal{I}_4^{AT}$	4	17	59	203	1612	47	5	1646	458	1.2x	9.7x

Continued on next page

Table D.3 – continued from previous page

	Big-M			*MaxSAT*			*MILP*			*Speed-up*	
Inst.	#It.	#Con.	Time	#It.	#Int.	Time	#It.	#Int.	Time	Big-M	MILP
$\mathcal{I}_5^{AT}$	4	15	48	186	1393	34	5	1290	373	1.4x	10.9x
$\mathcal{I}_6^{AT}$	4	7	23	108	634	7	4	642	110	3.3x	15.6x
$\mathcal{I}_7^{AT}$	3	8	22	71	468	8	3	476	75	2.9x	10.0x
$\mathcal{I}_8^{AT}$	-	-	[2.0%]	379	19371	15523	-	-	[21%]	-	-
$\mathcal{I}_9^{AT}$	6	47	453	208	3832	97	7	3724	10307	4.7x	106.6x
$\mathcal{I}_{10}^{AT}$	4	19	67	107	1870	21	3	1469	195	3.2x	9.4x
$\mathcal{I}_{11}^{AT}$	-	-	[32%]	-	-	[34%]	-	-	[46%]	-	-
$\mathcal{I}_{12}^{AT}$	-	-	[37%]	-	-	[37%]	-	-	[51%]	-	-
$\mathcal{I}_1^{BT}$	4	23	87	154	2032	39	5	1826	384	2.2x	9.8x
$\mathcal{I}_2^{BT}$	3	6	14	62	324	5	3	339	49	2.8x	9.4x
$\mathcal{I}_3^{BT}$	3	7	20	69	483	7	3	375	40	2.9x	5.9x
$\mathcal{I}_4^{BT}$	7	39	192	175	2863	94	14	4000	37913	2.0x	402.7x
$\mathcal{I}_5^{BT}$	4	10	26	101	775	9	5	609	166	2.9x	18.3x
$\mathcal{I}_6^{BT}$	2	3	11	51	268	2	2	279	14	4.6x	5.9x
$\mathcal{I}_7^{BT}$	4	41	136	190	3306	164	7	2816	1433	0.8x	8.7x
$\mathcal{I}_8^{BT}$	4	6	19	107	432	6	5	385	94	3.2x	16.0x
$\mathcal{I}_9^{BT}$	5	11	37	120	941	14	8	873	589	2.7x	42.0x
$\mathcal{I}_{10}^{BT}$	2	3	11	47	294	3	2	291	11	3.3x	3.4x
$\mathcal{I}_{11}^{BT}$	6	46	289	169	2837	78	7	2737	1293	3.7x	16.6x
$\mathcal{I}_{12}^{BT}$	7	34	137	194	1995	131	15	2146	12302	1.0x	93.8x
$\mathcal{I}_1^{AS}$	7	77	10298	351	8475	722	10	7275	77419	14.3x	107.2x
$\mathcal{I}_2^{AS}$	5	63	1571	218	5934	295	7	4883	18588	5.3x	63.1x
$\mathcal{I}_3^{AS}$	4	31	157	235	3604	104	10	4152	19989	1.5x	192.0x
$\mathcal{I}_4^{AS}$	4	19	71	158	1954	27	5	1880	701	2.6x	25.7x
$\mathcal{I}_5^{AS}$	3	15	37	146	1905	32	6	1718	996	1.2x	31.2x
$\mathcal{I}_6^{AS}$	2	4	15	56	501	5	2	490	26	3.0x	5.2x
$\mathcal{I}_7^{AS}$	3	7	26	138	656	12	4	650	146	2.2x	12.3x
$\mathcal{I}_8^{AS}$	-	-	[9%]	563	24773	54003	-	-	[14%]	-	-
$\mathcal{I}_9^{AS}$	6	46	528	191	3741	93	7	4072	18038	5.7x	194.0x
$\mathcal{I}_{10}^{AS}$	3	21	61	131	2041	27	6	2291	1676	2.3x	63.2x
$\mathcal{I}_{11}^{AS}$	-	-	[26%]	-	-	[35%]	-	-	[38%]	-	-
$\mathcal{I}_{12}^{AS}$	-	-	[33%]	-	-	[40%]	-	-	[41%]	-	-
$\mathcal{I}_1^{BS}$	4	29	198	192	2603	93	7	2674	12104	2.1x	130.4x
$\mathcal{I}_2^{BS}$	3	6	14	53	290	4	3	353	60	3.9x	16.6x
$\mathcal{I}_3^{BS}$	2	5	12	67	477	5	3	467	63	2.6x	14.0x
$\mathcal{I}_4^{BS}$	5	36	364	219	2979	133	7	2714	3413	2.7x	25.7x
$\mathcal{I}_5^{BS}$	6	10	40	107	870	15	6	567	219	2.6x	14.3x
$\mathcal{I}_6^{BS}$	2	3	12	46	303	3	2	283	16	4.0x	5.3x
$\mathcal{I}_7^{BS}$	4	47	883	202	3742	170	8	3522	9559	5.2x	56.1x
$\mathcal{I}_8^{BS}$	3	5	16	123	381	5	6	389	141	3.0x	26.0x
$\mathcal{I}_9^{BS}$	5	10	33	83	746	7	4	545	95	4.4x	12.7x
$\mathcal{I}_{10}^{BS}$	2	3	16	52	289	3	2	291	18	4.6x	5.3x
$\mathcal{I}_{11}^{BS}$	6	57	1944	201	3571	141	12	4005	29879	13.8x	212.5x
$\mathcal{I}_{12}^{BS}$	6	32	203	166	1800	105	7	2064	2992	1.9x	28.6x

Table D.4: Computational results for the stepwise linear objective function.

	Big-M			*MaxSAT*			*MILP*			*Speed-up*	
Inst.	#It.	#Con.	Time	#It.	#Int.	Time	#It.	#Int.	Time	Big-M	MILP
$\mathcal{I}_1^{AO}$	3	8	88	127	1887	19	3	1623	170	4.6x	8.7x
$\mathcal{I}_2^{AO}$	8	25	154	135	1684	25	5	1749	205	6.2x	8.3x
$\mathcal{I}_3^{AO}$	4	9	60	126	1298	16	4	1264	121	3.8x	7.7x
$\mathcal{I}_4^{AO}$	7	13	95	152	1069	16	5	1222	135	5.8x	8.3x
$\mathcal{I}_5^{AO}$	4	8	45	127	824	12	4	878	131	3.8x	10.9x
$\mathcal{I}_6^{AO}$	3	4	27	93	520	5	4	552	60	5.3x	12.1x
$\mathcal{I}_7^{AO}$	4	7	20	55	398	4	2	450	15	5.2x	4.0x
$\mathcal{I}_8^{AO}$	6	56	182	214	3514	80	5	3728	507	2.3x	6.3x
$\mathcal{I}_9^{AO}$	6	22	78	128	1874	20	5	1636	175	3.9x	8.7x
$\mathcal{I}_{10}^{AO}$	5	19	58	115	1303	15	5	1447	194	3.8x	12.6x
$\mathcal{I}_{11}^{AO}$	6	31	212	129	3000	49	4	2736	474	4.4x	9.8x
$\mathcal{I}_{12}^{AO}$	6	35	320	206	2971	86	7	3480	1379	3.7x	16.1x
$\mathcal{I}_1^{BO}$	2	2	30	39	719	4	2	726	36	8.5x	10.4x
$\mathcal{I}_2^{BO}$	3	2	14	61	242	3	2	205	9	4.5x	2.9x
$\mathcal{I}_3^{BO}$	3	4	16	50	346	3	3	341	26	6.0x	9.5x
$\mathcal{I}_4^{BO}$	3	6	43	121	1380	16	6	1096	134	2.6x	8.1x
$\mathcal{I}_5^{BO}$	3	2	17	73	293	3	2	225	13	5.5x	4.1x
$\mathcal{I}_6^{BO}$	2	1	8	17	172	1	2	199	9	10.5x	11.2x
$\mathcal{I}_7^{BO}$	6	25	100	104	2299	30	5	1744	188	3.3x	6.2x
$\mathcal{I}_8^{BO}$	2	1	11	47	180	2	2	181	10	4.7x	4.6x
$\mathcal{I}_9^{BO}$	3	6	20	96	548	10	4	563	60	2.0x	6.0x
$\mathcal{I}_{10}^{BO}$	5	10	28	64	506	6	3	546	36	5.1x	6.6x
$\mathcal{I}_{11}^{BO}$	7	33	92	112	2164	25	5	1577	158	3.7x	6.4x
$\mathcal{I}_{12}^{BO}$	2	3	25	57	730	5	2	604	24	4.8x	4.5x
$\mathcal{I}_1^{AT}$	7	92	2204	221	5017	188	8	6287	8376	11.7x	44.6x
$\mathcal{I}_2^{AT}$	6	63	753	159	3732	58	7	4559	1997	12.9x	34.2x
$\mathcal{I}_3^{AT}$	6	40	284	307	3114	114	6	2782	981	2.5x	8.6x
$\mathcal{I}_4^{AT}$	4	25	72	95	1423	19	3	1384	150	3.9x	8.1x
$\mathcal{I}_5^{AT}$	6	20	69	104	1096	11	3	1024	99	6.2x	9.0x
$\mathcal{I}_6^{AT}$	4	7	28	104	652	8	4	632	96	3.7x	12.5x
$\mathcal{I}_7^{AT}$	4	9	28	63	437	5	3	524	62	5.5x	12.1x
$\mathcal{I}_8^{AT}$	8	147	7953	187	9743	210	10	9607	59085	37.9x	281.4x
$\mathcal{I}_9^{AT}$	5	45	167	157	2567	47	4	2786	377	3.6x	8.1x
$\mathcal{I}_{10}^{AT}$	5	25	79	88	1504	17	6	1617	272	4.7x	16.0x
$\mathcal{I}_{11}^{AT}$	-	-	[3%]	207	10554	396	9	11093	108005	-	272.8x
$\mathcal{I}_{12}^{AT}$	5	153	14680	222	12754	540	8	11428	63967	27.2x	118.4x
$\mathcal{I}_1^{BT}$	4	21	107	106	1605	29	6	1926	494	3.7x	17.3x
$\mathcal{I}_2^{BT}$	3	6	16	58	277	3	3	325	34	5.5x	11.9x
$\mathcal{I}_3^{BT}$	3	7	19	72	434	5	3	375	32	4.2x	7.0x
$\mathcal{I}_4^{BT}$	5	33	103	118	2494	39	4	2116	271	2.7x	7.0x
$\mathcal{I}_5^{BT}$	4	10	33	92	702	7	4	597	90	4.5x	12.3x
$\mathcal{I}_6^{BT}$	2	3	12	51	268	4	2	279	13	2.9x	3.0x
$\mathcal{I}_7^{BT}$	7	42	169	131	2722	43	4	2112	256	4.0x	6.0x
$\mathcal{I}_8^{BT}$	3	5	13	87	372	6	4	371	60	2.1x	10.0x
$\mathcal{I}_9^{BT}$	6	10	42	67	707	6	5	579	165	6.8x	26.6x

Continued on next page

Table D.4 – continued from previous page

	Big-M			*MaxSAT*			*MILP*			*Speed-up*	
Inst.	#It.	#Con.	Time	#It.	#Int.	Time	#It.	#Int.	Time	Big-M	MILP
$\mathcal{I}_{10}^{BT}$	2	3	11	49	294	3	2	291	12	3.4x	3.8x
$\mathcal{I}_{11}^{BT}$	5	43	175	147	3083	53	6	2469	635	3.3x	12.0x
$\mathcal{I}_{12}^{BT}$	4	27	83	106	1286	15	5	1588	321	5.5x	21.4x
$\mathcal{I}_{1}^{AS}$	4	92	10559	201	6620	223	3	5257	3901	47.4x	17.5x
$\mathcal{I}_{2}^{AS}$	3	66	474	210	5181	116	4	4747	1923	4.1x	16.5x
$\mathcal{I}_{3}^{AS}$	4	38	130	144	2607	39	3	2336	411	3.4x	10.6x
$\mathcal{I}_{4}^{AS}$	4	25	69	167	1751	30	3	1410	176	2.3x	6.0x
$\mathcal{I}_{5}^{AS}$	4	16	58	103	1371	12	3	1246	165	4.7x	13.3x
$\mathcal{I}_{6}^{AS}$	2	4	16	51	495	3	2	490	27	4.6x	8.0x
$\mathcal{I}_{7}^{AS}$	3	8	22	107	558	8	4	712	90	2.8x	11.4x
$\mathcal{I}_{8}^{AS}$	4	113	4099	194	9805	215	4	6471	7816	19.1x	36.4x
$\mathcal{I}_{9}^{AS}$	6	53	193	141	2843	40	3	2902	405	4.8x	10.1x
$\mathcal{I}_{10}^{AS}$	5	27	83	110	1886	19	3	1827	160	4.5x	8.6x
$\mathcal{I}_{11}^{AS}$	3	96	18297	167	10963	437	4	6855	11073	41.9x	25.3x
$\mathcal{I}_{12}^{AS}$	3	109	6137	168	12056	286	6	10004	30737	21.4x	107.4x
$\mathcal{I}_{1}^{BS}$	4	30	105	90	1882	19	3	1842	248	5.5x	13.0x
$\mathcal{I}_{2}^{BS}$	4	8	21	48	267	2	3	339	47	9.6x	22.0x
$\mathcal{I}_{3}^{BS}$	2	5	12	71	473	6	3	425	44	2.1x	7.7x
$\mathcal{I}_{4}^{BS}$	4	33	144	149	2564	44	3	2074	330	3.3x	7.4x
$\mathcal{I}_{5}^{BS}$	5	9	38	75	721	7	4	555	93	5.2x	13.0x
$\mathcal{I}_{6}^{BS}$	2	3	13	46	303	3	2	283	21	4.0x	6.5x
$\mathcal{I}_{7}^{BS}$	5	50	535	148	3184	58	5	3130	1059	9.2x	18.2x
$\mathcal{I}_{8}^{BS}$	3	5	16	83	342	4	3	331	29	3.7x	6.8x
$\mathcal{I}_{9}^{BS}$	8	19	71	66	566	5	5	501	67	13.3x	12.5x
$\mathcal{I}_{10}^{BS}$	5	6	28	58	316	4	5	357	56	7.3x	14.6x
$\mathcal{I}_{11}^{BS}$	3	42	414	122	3076	36	3	2463	437	11.6x	12.2x
$\mathcal{I}_{12}^{BS}$	6	27	122	121	1655	28	5	1658	465	4.4x	16.6x

All the three tables give evidence that the DDD-TRP algorithm offers better performances when it uses the MaxSAT solver instead of the MILP one. In particular, on the instances that could be solved to optimality, the speed-up among the two algorithms is always greater than one (with only one exception in Table D.2) and its average value is 12.7 for the continuous linear objective function (Table D.2), 39.7 for the rounded linear objective function (Table D.3), and 18.4 for the stepwise linear objective function (Table D.4). The comparison between DDD-TRP with MaxSAT and the MILP formulation requires a more detailed analysis. In particular, for the case of the linear objective function, MILP could solve 67 of the 72 instances within the time limit, while MaxSAT only 57. On the other hand, on the instances that could be solved by both the algorithms, MaxSAT is faster than MILP on 38 (over 57) cases. For the rounded linear objective function (Table D.3), the situation is more clearly in favour of MaxSAT. Indeed, here MaxSAT could solve 3 instances more than MILP (68 vs 65 over 72). Moreover, MaxSAT results in all cases (with only one exception) to be the faster algorithm, with an average speed up on the solved instances of 3.8. The same behaviour is confirmed in the case of stepwise linear functions (Table D.4). Here both MaxSAT and MILP could solve all (but one, for MILP) the instances within the time limit but MaxSAT presents computational times always smaller than MILP (the average speed up is 7.3).

Finally, in order to illustrate the scale of SAT instances addressed by the MaxSAT RC2 algorithm, Table D.5 presents the most challenging instances (corresponding to those in Table 2). The table includes information on the number of variables and clauses in the MiniSat solver instance after all the required IAP instances have been solved, i.e. at the end of the DDD-ALG.

Table D.5: Comparison of SAT instance sizes at the convergence of the DDD-ALG using RC2 and MiniSat on the 10 hardest instances with a stepwise objective function.

Instance	**Time (ms)**	**Variables**	**Clauses**
$\mathcal{I}_1^{AS}$	249	5894	11626
$\mathcal{I}_2^{AS}$	78	4147	8025
$\mathcal{I}_8^{AS}$	186	8499	18839
$\mathcal{I}_{11}^{AS}$	419	9374	20773
$\mathcal{I}_{12}^{AS}$	553	12284	27899
$\mathcal{I}_1^{AT}$	86	4757	9428
$\mathcal{I}_2^{AT}$	63	3212	5923
$\mathcal{I}_8^{AT}$	181	9610	22198
$\mathcal{I}_{11}^{AT}$	300	9869	22619
$\mathcal{I}_{12}^{AT}$	285	10858	24791