Journal of Artificial Intelligence Research 79 (2024) 1047-1090 Submitted 05/2023; published 03/2024

Iterative Train Scheduling under Disruption with Maximum Satisfiability

Alexandre Lemos ALEXANDRE.LEMOS@TECNICO.ULISBOA.PT
Filipe Gouveia FILIPE.GOUVEIA@TECNICO.ULISBOA.PT
Pedro T. Monteiro PEDRO.TIAGO.MONTEIRO@TECNICO.ULISBOA.PT
Ines Lynce INES.LYNCE@TECNICO.ULISBOA.PT
Instituto Superior Técnico, Universidade de Lisboa
INESC-ID, Rua Alves Redol 9, 1000-029 Lisboa, Portugal

Abstract

This paper proposes an iterative Maximum Satisfiability (MaxSAT) approach designed to solve train scheduling optimization problems. The generation of railway timetables is known to be intractable for a single track. We consider hundreds of trains on interconnected multi-track railway networks with complex connections between trains. Furthermore, the proposed algorithm is incremental to reduce the impact of time discretization.

The performance of our approach is evaluated with the real-world Swiss Federal Railway (SBB) Crowd Sourcing Challenge benchmark and Periodic Event Scheduling Problems benchmark (PESPLib). The execution time of the proposed approach is shown to be, on average, twice as fast as the best existing solution for the SBB instances. In addition, we achieve a significant improvement over SAT-based solutions for solving the PESPLib instances.

We also analyzed real schedule data from Switzerland and the Netherlands to create a disruption generator based on probability distributions. The novel incremental algorithm allows solving the train scheduling problem under disruptions with better performance than traditional algorithms.

1. Introduction

The transport sector is expected to continue to grow over the next three decades[1]. Moreover, the reduction in CO_2 emissions agreed to in the Paris Agreement is expected to promote a shift toward the use of collective transport, notably railways. According to the UNIFE World Rail Market Study (Berger, 2018), the railway market is estimated to show an average growth of 2.7% worldwide for the 2021 to 2023 period. In particular, the European Union will invest in energy-efficient railways and continue transforming the railway into a seamless European network[2]. Hence, the size of the railways and the number of trains are expected to grow significantly in the coming years. Now, more than ever, it is essential to optimize the schedules of trains efficiently.

The 2019 Swiss Federal Railway (SBB - Schweizerische Bundesbahnen) challenge (Jordi & Mohanty, 2018) defined new data sets and constraints for Train Scheduling Optimiza-

1. For more details, see the Transport Outlook published in 2019 by the International Transport Forum.
2. More information about the European Railway Traffic Management System project is available at http://www.ertms.net/.

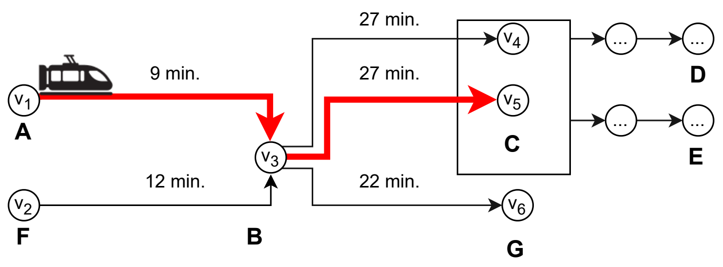


Figure 1: Part of a railway network where one train has to go from A to C. The bold red arrow represents a possible solution. Nodes and edges represent relevant points and sections of railway tracks, respectively. A relevant point of the railway network may be a stopping point in a station, a junction, or a point where the characteristics of the track change. The minimum travel time, in minutes, in each section of the railway track is depicted above the corresponding edge. The square represents a junction box, where a train can change tracks (changing from v_4 to v_5 and vice-versa), and from which a train can depart to any available track

.

tion Problems (TSOP) (Borndörfer et al., 2018). The TSOP problem can be informally described as the combination of two complementary problems: (i) a routing problem and (ii) a scheduling problem. The goal of the *routing problem* is to find the least cost route for all trains passing through pre-defined stations. The goal of the *scheduling problem* is to generate a timetable, *i.e.*, to assign departure times for each train from each station while minimizing the delay, subject to time and route constraints. To find an optimal solution, one must solve these two problems together (*scheduling* and *routing*), considering that there are two optimization criteria in TSOP: (i) minimize the delay and (ii) minimize the route cost.

Example 1 *Figure 1 shows a simplified railway network. There are two train routes from B to C, and therefore, the railway supports two trains at the same time. Train station B can receive two trains (from A and F). B and C are connection spots since the trains can depart to multiple routes. Consider that one train has to go from A to C. A possible solution to this routing problem is shown in red. Now, let us consider that the traveling time from v_1 to v_3 is 9 minutes, and from v_3 to v_4 / v_5 is 27 minutes. Assume that each v_i corresponds to stopping points at the corresponding station. Consider that the train cannot leave A before 9* AM*. Also, consider that there is a connection with another train at C. The connection requires the train to arrive at C before 9:42* AM *to ensure the passengers can interchange with the train moving to E. Therefore, a feasible solution to this scheduling problem instance is the following timetable:*

- *The train departs from A at 9* AM*;*
- *To make the connection at C, the train stops in B for only 5 minutes;*
- *The train leaves at 9:14* AM *from B, thus arriving at the destination at 9:41* AM*.*

Planning and scheduling problems have been successfully encoded into propositional logic (Lemos et al., 2020; Schreiber, 2021) in the recent past, in particular, applied to the train domain (Goerigk et al., 2013; Luteberget et al., 2018; Matos, 2018; Matos et al., 2020). These approaches have the advantage of using propositional satisfiability (SAT) solvers that are well-known for being quite efficient (Biere et al., 2009). However, none of these approaches applies SAT to solve TSOP, which is a complex problem with many constraints.

The *contribution* of this paper is two-fold: (i) a compact MaxSAT encoding for TSOP and TSOP under disruption (TSOPuD); and (ii) a novel iterative MaxSAT algorithm to solve TSOP and TSOPuD. The new encoding is shown to efficiently solve the large data set from the SBB benchmarks (Abels et al., 2020b; Jordi et al., 2019). The proposed iterative algorithm allows the reduction of the size of the problem as we only delay the trains when needed. With the proposed approach, we are able to find an optimal solution for all SBB instances within 260 seconds. Our implementation is publicly available at `https://github.com/ADDALemos/train-schedule-optimisation`.

This paper is organized as follows. Section 2 provides the required background on train scheduling and MaxSAT solving. Section 3 formally describes TSOP and the considered disruptions (TSOPuD). Section 4 presents the proposed approach, detailing the MaxSAT encoding and the proposed iterative algorithms for TSOP and TSOPuD. Section 5 analyses the evaluation of the different iterative algorithms. Furthermore, we compare our approach with the current state-of-the-art solutions. Finally, Section 6 concludes the paper and discusses possible future directions.

2. Background

In this section, we present and discuss the relevant state-of-the-art in maximum satisfiability (MaxSAT), the generation of railway timetables, and possible causes of disruptions that can affect a train schedule.

2.1 MaxSAT

A propositional formula in conjunctive normal form (CNF) is defined as a conjunction of clauses, where a clause is a disjunction of literals and a literal is either a Boolean variable x or its complement $\neg x$. Given a CNF formula, the propositional satisfiability (SAT) problem is to decide whether there is a truth assignment to the variables such that the formula is satisfied. A formula is satisfied *iff* there is at least one assignment for which all the clauses are satisfied. A clause is satisfied *iff* it contains at least one literal that is satisfied. A literal x or $\neg x$ is satisfied if it is assigned true or false, respectively. Nowadays, most SAT solvers implement conflict-driven clause learning algorithms (Marques-Silva & Sakallah, 1996; Bayardo Jr. & Schrag, 1997), which extend the well-known Davis-Putnam algorithm (Davis & Putnam, 1960) with clause learning (see Biere et al. (2009) for more details).

The MaxSAT problem is an optimization version of SAT, where the objective is to find an assignment that *maximizes* the number of satisfied clauses. A partial MaxSAT formula $\varphi = \varphi_h \cup \varphi_s$ consists of a set of hard clauses φ_h and a set of soft clauses φ_s. The objective in partial MaxSAT is to find an assignment such that all hard clauses in φ_h are satisfied, while maximizing the number of satisfied soft clauses in φ_s.

In this paper, we consider the weighted variant of partial MaxSAT where there is a function $w^{\varphi} : \varphi_s \to \mathbb{N}$ associating an integer weight to each soft clause. In this case, the objective is to satisfy all the clauses in φ_h and maximize the total weight of the satisfied clauses in φ_s.

In general, we assume that all formulas are encoded into CNF. Nevertheless, we will write some constraints in pseudo-Boolean (PB) form for the sake of readability. PB constraints are linear constraints over Boolean variables and can be written as follows: $\sum q_i x_i \text{ OP } K$, where K and q_i are integer constants, x_i are Boolean variables, and $\text{OP} \in \{<, \leq, =, \geq, >\}$. PB constraints can be easily translated into CNF (Eén & Sörensson, 2006). To encode PB constraints in CNF, in this work we use the adder encoding (Warners, 1998) and the ladder encoding (Ansótegui & Manyà, 2004).

Example 2 *Consider the following PB constraint:* $\sum_{i=0}^{2} x_i \leq 1$. *One possible CNF encoding is as follows:* $(\neg x_0 \vee \neg x_1) \wedge (\neg x_0 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$.

2.2 Railway Timetabling

The generation of railway timetables is known to be intractable for a single track (Caprara et al., 2002). Nonetheless, different methods have been proposed to solve this problem effectively (Borndörfer et al., 2018; Caimi et al., 2017; Liebchen & Möhring, 2007; Sels et al., 2016; Fischetti & Monaci, 2017). In the past, a few challenges have been organized (Artigues et al., 2018; Jordi et al., 2019). In the 2014 ROADEF/EURO challenge for the shunting yard operations, most participants submitted greedy algorithms (Artigues et al., 2018). The winner of the challenge used a mix of Integer Linear Programming (ILP) and greedy heuristics (Buljubašić et al., 2017).

Recently, the SBB challenge (Jordi et al., 2019) motivated the appearance of new approaches ranging from ILP and Answer Set Programming (ASP) (Abels et al., 2020b) to a greedy algorithm. The proposed ILP solution decomposes the problem into two subproblems: routing and scheduling. This decomposition may result in suboptimal results but drastically reduces the size of the problem at hand. The ASP solution uses a hybrid ASP solver with difference constraints (*e.g.* $u - v \leq d$ where $u, v, d \in \mathbb{Z}$). Furthermore, the optimization problem is solved with an approximation cost function that allows reducing the size of the problem. However, it may remove the optimal solution. The greedy algorithm, the winner of the challenge, solves the most *critical* conflicts first. A conflict between two trains occurs when both trains occupy the same resource simultaneously. The flexibility of the time constraints and the density of resources are used to determine the *criticality* of a conflict. The density of resources is defined by the number of trains that require the same resource at the same time. Trains with more flexible time constraints are less critical.

Matos *et al.* (2018, 2020) proposed a binary search approach that uses a SAT solver to find global minimum solutions concerning travel time and a procedure to compute a better upper bound and speed up the search process. The resulting tool is able to solve all the instances from the PESPlib benchmark (Goerigk et al., 2013) without exceeding the memory limit (64 Gb). However, it does not ensure optimality. Nevertheless, the approach was able to improve the best currently known value in 7 out of the 20 of the PESPLib benchmark instances.

2.3 Handling Disruptions in Train Scheduling

The generation of robust timetables (or how to recover timetables after a disruption) has been widely studied for different scheduling problems (Fischetti & Monaci, 2017; Lusby et al., 2018; Fang et al., 2015; Andersson, 2014; Bajestani & Beck, 2013; Geiger et al., 2019; Adenso-Díaz et al., 1999; Acuna-Agost et al., 2011). A disruption can have multiple causes, from weather influences and accidents to staffing problems (Caprara et al., 1998). Nevertheless, we can summarize all causes into three kinds of disruptions: blocking a train, blocking a track, and changes in the travel speed of a train. Blocking a train is a disruption that affects only a single train but not the track. Blocking a track is a disruption that only affects part of the railway, and the trains are free to move around the rest of the network. Finally, the changes in the travel speed of a train (slowdown) are disruptions that affect part of the track and force a train to reduce its speed. Disruptions can occur before or after a train departs from the station (Fischetti & Monaci, 2017). The disruptions that occur after a train departs require the algorithm to be run in real-time since the train is currently en route. Naturally, we have more flexibility if the train has not yet started its route. The three types of disruptions can have different causes[3]. A summary of the related work is shown in Table 1.

The different causes of *block train* disruptions are the following:

- *Logistical problems*: something unforeseen occurs in the planned routes. An example of a logistical problem is a delay in train connections with other trains or other services (*e.g.*, buses).
- *External influences*: outside forces that impact the regular operation of the railway. External influences are, for example, police investigations or fire alarms.
- *Accidents*: the different types of accidents related to trains (*e.g.*, collision, derail, run over). Accidents with trains arise during the travel of trains. Hence, the trains involved in an accident are delayed. This type of disruption does not only block a part of the railway but also the trains affected. Thus, no re-routing can reduce the delay. In this type of disruption, both the trains and their passengers are blocked. The only solution to mitigate this problem would require moving passengers to another train.
- *Staffing problems*: strikes and sickness of the crew members may compromise the safe operation of a train. Therefore, staffing problems may cause a change in the crew's assignment to a train, leading to delays in that train. In this work, we do not solve the crew's assignment problem (Hoffmann et al., 2017; Demirović et al., 2017). However, delays in trains due to strikes are considered. This cause is simulated by blocking a train for a period of time corresponding to the strike. This type of disruption can be mitigated by having redundant resources to assign in case of necessity.

3. The real-time update of the disruptions and their causes can be accessed in `https://www.rijdendetreinen.nl/en/statistics/cause`.

Table 1: Review of the different causes of disruptions of train scheduling problems considered in this work and in related work (Reference)

Cause	Disruption	Occurrence	Reference
Weather influences	*Slowdown*	Before/During	(Adenso-Díaz et al., 1999) (Acuna-Agost et al., 2011)
Logistical problems	Block train	During	(Schachtebeck & Schöbel, 2010) (Acuna-Agost et al., 2011) (Veelenturf, Kidd, Cacchiani, Kroon, & Toth, 2016)
External influences	Block train	During	(Schachtebeck & Schöbel, 2010) (Veelenturf et al., 2016)
Accidents	Block track/train	During	(Li, Shou, & Ralescu, 2014) (Corman, D'Ariano, Pacciarelli, & Pranzo, 2012) (Acuna-Agost et al., 2011) (Binder, Maknoon, & Bierlaire, 2017) (Adenso-Díaz et al., 1999) (Veelenturf et al., 2016) (Sato, Tamura, & Tomii, 2013) (Fischetti & Monaci, 2017)
Engineering work	Block track	During	(Aken, Bešinović, & Goverde, 2017) (Veelenturf et al., 2016) (Fischetti & Monaci, 2017)
Infrastructure problems	Block track	Before	(Higgins, Kozan, & Ferreira, 1996) (Adenso-Díaz et al., 1999) (Acuna-Agost et al., 2011) (Veelenturf et al., 2016) (Fischetti & Monaci, 2017)
Staffing problems	Partially considered in this paper		(Adenso-Díaz et al., 1999)
Rolling stock problems	Not considered in this paper		(Adenso-Díaz et al., 1999) (Binder et al., 2017; Sato et al., 2013) (Sato et al., 2013)

The different causes of *block track* disruptions are the following:

- *Accidents*: the different types of accidents related to tracks (*e.g.*, fallen trees). The tracks become blocked and thus may cause a delay. We may solve this delay with the re-routing of the trains.

- *Engineering work*: unexpected engineering works on the tracks. There is a need to wait or re-route the trains to avoid the tracks affected by the engineering works.

- *Infrastructure problems*: the different railway infrastructure problems (*e.g.*, signals, overhead wires) cause the need for scheduled engineering work. We know beforehand which tracks need work, and thus, we can plan ahead.

The only considered causes of *slowdown* disruptions are *weather influences*: different weather patterns (*e.g.*, rain, snow) can reduce visibility and cause slippery tracks. Hence, the weather causes a reduction in train speed on the affected tracks.

Table 2: Review of the different cost functions used.

Minimize	
Delay	(Molloy, 2020)
Weighted delay with the number of passengers	(Molloy, 2020)
Cost of changing track	(Li et al., 2014; Higgins et al., 1996; Binder et al., 2017)
Missed connections	(Schachtebeck & Schöbel, 2010)
Cost of compensation	(Zeng, Durach, & Fang, 2012; Tseng & Verhoef, 2008)
Hamming distance	(Binder et al., 2017)
Canceled trains	(Aken et al., 2017)
Additional connections for a passenger	(Sato et al., 2013)
Maximize	
Number of passengers transported	(Adenso-Díaz et al., 1999)
This paper	
Delay + Cost of changing track + No missed connections + Cost of compensation	

In this work, we do not consider the disruptions caused by rolling stock problems: a train may need replacements during its travel. This problem can be mitigated by having redundant resources in key places on the track. These resources are used to re-supply when needed.

Different cost functions are used in the literature when recovering timetables after disruptions. A summary of the different cost functions is shown in Table 2. The most straightforward cost function is the Hamming distance (Binder et al., 2017), *i.e.*, the number of variables[4] that change their assignment between the original solution and the new feasible solution. However, the Hamming distance is domain-independent, thus not a realistic cost function. The most common cost function is to minimize the delay (Molloy, 2020) or the operational cost of the delay (Zeng et al., 2012; Tseng & Verhoef, 2008; Molloy, 2020). The cost function can be even more detailed, considering the impact of a train changing tracks (Li et al., 2014; Higgins et al., 1996; Binder et al., 2017). A change of track by a train impacts the operational cost and train speed.

The cost function can also focus on the impact on the passengers (Adenso-Díaz et al., 1999). To reduce the impact of re-scheduling trains on the passengers, we can avoid the cancellation of trains (Aken et al., 2017) and reduce the number of connections the passengers require to arrive at their destination (Sato et al., 2013). This type of cost function requires *a priori* knowledge of the passenger routes.

3. Train Scheduling Optimization Problems

Next, we formalize TSOP based on the description provided by the SBB challenge (Jordi et al., 2019). A railway network $\mathcal{R}$ is characterized by a graph (V, E) where V is the set of nodes (representing stopping points in stations, or junctions[5]) and E is the set of edges (representing sections of railway tracks with the same characteristics). Each node has a label l associated. We denote $\mathcal{L}$ as the set of labels $l_v, v \in V$, where l_v is the label of node

4. The variables used to model the problem.
5. A junction node is a place where two railways join or split. The junction can also symbolize a change in the quality of the track and, therefore, a change in the maximum speed of the train.

v. The minimum travel time of any train in an edge $e \in E$ is defined as t_e^{min} (an integer value in seconds). The cost of traveling in an edge $e \in E$ is given by p_e.

Example 3 *Recall the railway network shown in Figure 1. The set of nodes is* $V = \{v_1, \ldots, v_6\}$ *and the set of edges is* $E = \{(v_1, v_3), (v_2, v_3), (v_3, v_4), (v_3, v_5), (v_3, v_6)\}$. *The labels are* l_{v_1}*=A,* l_{v_2}*=F,* l_{v_3}*=B,* l_{v_4}*=*l_{v_5}*=C, and* l_{v_6}*=G.*

Consider a set of train lines $\mathcal{T}$. Each train line has its own train and a set of possible routes that it may take. Furthermore, each train line must be scheduled on the global railway $\mathcal{R}$. A train line $\tau \in \mathcal{T}$ is characterized by: a direct acyclic subgraph (V_τ, E_τ), $V_\tau \subseteq V$, $E_\tau \subseteq E$, with all possible routes the train can take; a set of labels $\mathcal{L}_\tau$ representing the stops the train must do; the earliest (latest) arrival time $t_{\tau,v}^{earliest}$ ($t_{\tau,v}^{latest}$) at node $v \in V_\tau$ (an integer value in seconds); the minimum stopping time of the train on node $v \in V_\tau$ is $t_{\tau,v}^{stop}$ (an integer value in seconds); and the set of connections C_τ between the train in train line τ and other trains in different train lines. We formally define the connections C_τ later in this section. A route for a train in a train line τ is a sequence of connected nodes $v \in V_\tau$ that the train must stop at, such that the set of all l_v contains $\mathcal{L}_\tau$. $\mathcal{P}_\tau$ is the set of all the possible routes a train can take. Each route is denoted by P_τ^i, with $i \in [1 \ldots |\mathcal{P}_\tau|]$ and $P_\tau^i \in \mathcal{P}_\tau$.

Example 4 *Recall Figure 1. Let us consider a train* $\tau \in \mathcal{T}$ *with* $V_\tau = V$ *and* $E_\tau = E$. *A train must start in A and end in C. Hence, there are two possible routes the train can take:* $\{(v_1, v_3),(v_3, v_4)\}$ *or* $\{(v_1, v_3),(v_3, v_5)\}$ *(red bold line).*

In this work, we consider that the assignment of trains to train lines occurs beforehand[6]. Hence, we consider that each train line has only one associated train, and each train belongs to only one train line. For this reason, the concepts of trains and train lines are interchangeable. From now on, we will only use the word train to improve readability.

Considering the earliest/latest arrival time, we can define, for each train τ and node v, the entry time domain $\gamma_{\tau,v} = [t_{\tau,v}^{earliest}, t_{\tau,v}^{latest}]$. Not all nodes have associated limits on entry time. Furthermore, the entry time limits are only guidelines and not mandatory. In the most relaxed version, $\gamma_{\tau,v} = [t_{\tau,v_i}^{earliest}, t_{\tau,v_f}^{latest}]$ where v_i and v_f are the first and last nodes of the train route, respectively. However, considering the most relaxed version may lead to memory problems since the encoding grows exponentially in size with all the necessary time variables, and the space of possible solutions explodes.

Example 5 *Recall Example 1 and Figure 1. Let us consider the train is already in node* v_1*, and thus no entry time domain for* v_1 *exists. The train cannot leave* v_1 *before 9*AM*, and cannot leave* v_3 *after 9:14 in order to arrive at the destination before 9:42, considering that the train takes 27 minutes from* v_3 *to the destination. Considering that the train takes 9 minutes from* v_1 *to* v_3*,* γ_{τ,v_3} = *[9:09,9:14]. The domain for* v_4 *and* v_5 *is* $\gamma_{\tau,v_4} = \gamma_{\tau,v_5}$ = *[9:36,9:42]. The earliest arrival time requires the train to skip station B. If* $t_{\tau,v_3}^{stop} \neq 0$ *then the train must stop at* v_3*, and we can reduce the domain of* γ_{τ,v_4} *and* γ_{τ,v_5} *since it would no longer be possible for the train to arrive at* v_4 *or* v_5 *at 9:36.*

6. Solving the rolling stock problem (Budai, Maróti, Dekker, Huisman, & Kroon, 2010) is outside the scope of this paper.

A connection $c \in C_\tau$ can be of two types:

- *virtual connection* that represents the merge of trains[7];
- *true connections* that represent the transfer of passengers/cargo between two trains on different tracks.

Each connection is characterized by: two trains τ_1 and τ_2; two nodes $v_1 \in V_{\tau_1}$ and $v_2 \in V_{\tau_2}$ where the transfer will occur; and the minimum connection time is $\iota_{\tau_1,\tau_2}^{(v_1,v_2)}$.

A railway network may contain junction boxes where a train can change to a different node in the junction box (see station C in Figure 1). Such track change can be seen as a connection and can have an associated connection time, $\iota_{\tau_1,\tau_1}^{(v_1,v_2)}$.

A *solution* to the TSOP has two components: (i) the assignment of all trains to the route they take through the railway, ensuring that the train passes through a set of nodes (routing problem); and (ii) an assignment of entry times to each train at each node on the respective route (scheduling problem). In this work, we consider a complete discretization of time in seconds.

Formally, a solution σ to a TSOP instance is characterized by: the scheduled entry time $t_{\tau,v}^{entry}$ of each train τ in each node v, and the scheduled passage of each train τ by an edge e represented by a Boolean variable $x_{\tau,e}$ which takes value 1 *iff* train τ goes through edge e.

An optimal solution to a TSOP instance is a solution that (i) minimizes the delay of the trains considering the entry time limits and (ii) minimizes the route cost. Further details are presented in Section 4.

3.1 Train Scheduling Optimization Problem under Disruption

The Train Scheduling Optimization Problem under Disruption (TSOPuD) is the problem of continuously solving the TSOP problem subject to a set of disruptions. TSOPuD is characterized by:

- σ_0, the original solution to a TSOP problem instance, *i.e.*, the route of each train and corresponding schedule;
- χ, a set of disruptions.

In this work, a disruption can be of three types: *slowdown*, *block track*, and *block train* (see Section 2.3). Each disruption $\boldsymbol{\varsigma} \in \chi$, independently of its type, has a set of common characteristics. All disruptions occur at a specific time $t_{\boldsymbol{\varsigma}}$ and have a duration $d_{\boldsymbol{\varsigma}}$. If the disruption starts before the train begins its route, $t_{\boldsymbol{\varsigma}} \leq \min_{v \in V_\tau, \tau \in \mathcal{T}}(t_{\tau,v}^{entry})$, then we have the freedom to change the whole schedule (we call these types of disruptions *before*). Otherwise, we can only change the schedule of trains for nodes with entry times after $t_{\boldsymbol{\varsigma}}$ (denoted *during*).

Example 6 *Consider the railway shown in Figure 2. The traveling time of the train from v_1 to v_3 is 9 minutes, and from v_3 to v_4 / v_5 is 27 minutes. Assume the train cannot leave*

7. In a virtual connection, two different trains may share a node. In this node, the two trains are merged and create a *new* train.

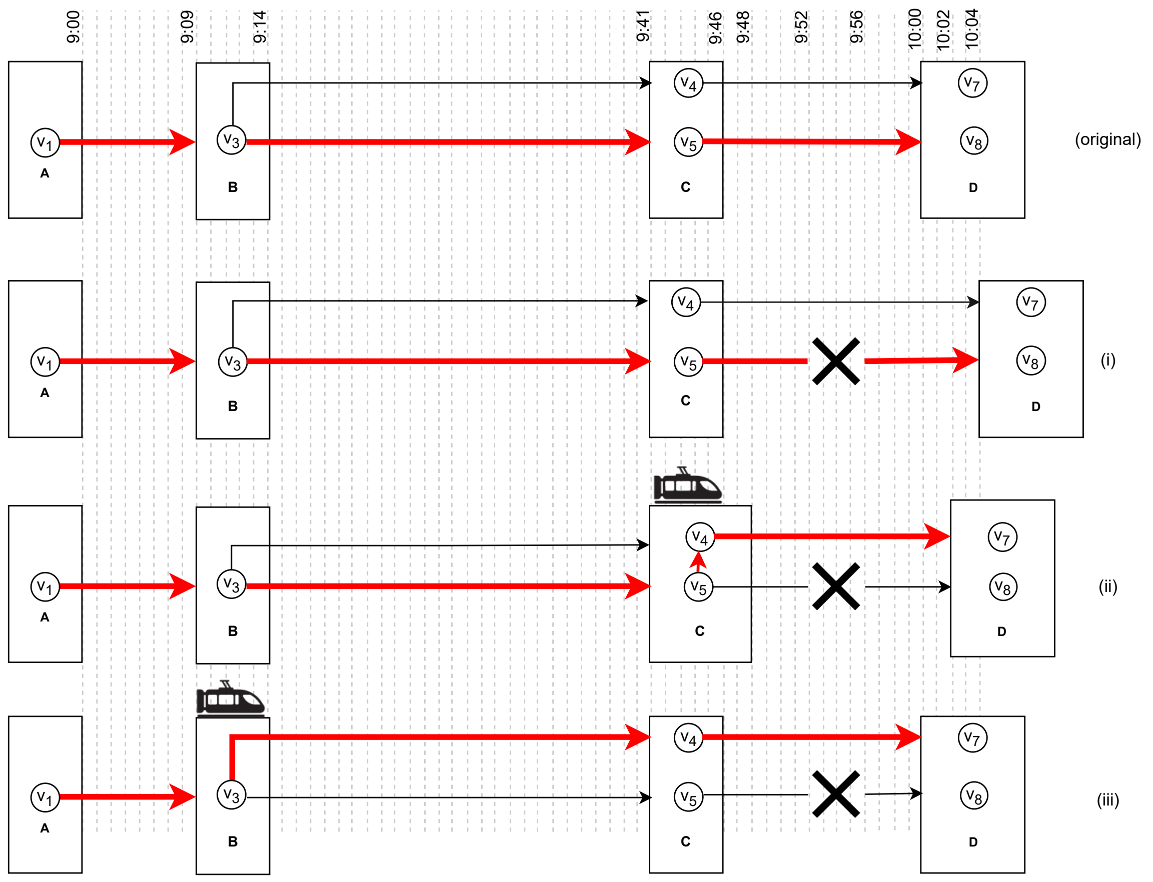


Figure 2: Schedule for a train on a railway network (original), and possible solutions after a disruption occurs in the track (v_5, v_8), blocking it for 4 minutes (cases (i), (ii), and (iii)). The bold red arrows represent the possible solutions. The dashed gray lines represent the time. The size of the rectangles depends on the duration of the train stop at the corresponding station. In case (i), the train waits for the disruption to end. In case (ii), the train switches tracks at station C and proceeds to station D using the track (v_4, v_7). In case (iii), the train switches tracks when departing from station B, using the track (v_3, v_4).

A before 9 AM *and it has a connection with another train at C. This connection requires the train to arrive at C before 9:42* AM *to ensure the passengers can switch to another train. Additionally, constraints require the train to stop for 5 minutes in C, and the minimal traveling time from C to D is 14 minutes. The expected arrival time in D is 10* AM*. A possible schedule (the original solution) is represented at the top. Finally, a disruption causes the edge (v_5, v_8) to be blocked for 4 minutes. There are three possible solutions to recover from this disruption: (i) wait for the edge to be free; (ii) change the path from (v_5, v_8) to (v_4, v_7); and (iii) change the path from (v_3, v_5), (v_5, v_8) to (v_3, v_4), (v_4, v_7). The simplest solution is the first one (i),* i.e. *to wait for the disruption to end. However, such solution causes a delay of 4 minutes. The second solution (ii) requires one change to the train path. Assuming that changing tracks takes 2 minutes, this change reduces the delay to 2 minutes. In this case, the delay is caused by the track change at station C, switching from*

v_5 to v_4. The third solution (iii) causes no delay. However, it does require more changes to the train path. Change (iii) can only occur if the disruption is discovered before *the train departs from station B.*

Additionally, the *slowdown* disruptions are characterized by the coefficient of velocity $\omega_{\varsigma,e}$ for a set of edges $e \in E_{\varsigma}$ affected by the disruption. This coefficient represents the reduction of the train velocity in edge e.

Example 7 *Consider a route (A, B) and (B, C). The train departs from A at 9*AM*, and it is expected to arrive at C at 3*PM*. The traveling time $t^{min}_{(A,B)} = t^{min}_{(B,C)} = 3$ hours. The train is in station B when the weather conditions causes $t^{min}_{(B,C)}$ to change to 6 hours ($\omega_{\varsigma,(B,C)} = 2$). Therefore, the train will arrive 3 hours late.*

The *block track* disruptions are additionally characterized by a set of edges $e \in E_{\varsigma}$ where no train can pass.

Example 8 *Consider a route (A, B) and (B, C). Furthermore, consider that there are two parallel edges between B and C referenced here as (B, C) and (B, C'). The traveling time of a train on the track $t^{min}_{(B,C)} = 3$ hours is shorter than on the track $t^{min}_{(B,C')} = 4$ hours. A fallen tree causes track (B, C) to be blocked for 3 hours. Re-routing the train to track (B, C') will reduce the delay of the train.*

The *block train* disruptions are additionally characterized by a set of trains $\tau \in \mathcal{T}_{\varsigma}$ that cannot travel.

Example 9 *Consider a route (A, B) and (B, C). The train departs from A at 9*AM *and it is expected to arrive at C at 3*PM*. The traveling time $t^{min}_{(A,B)} = t^{min}_{(B,C)} = 3$ hours. A strike causes the train to stop for 6 hours. We do not consider the rolling stock problem, and thus we cannot assign a new train. We have neither the data for train capacity (number of passengers) nor the number of trains available (stock) and their location while not working. The only solution is to wait. Therefore, in this case, the train is delayed by 6 hours.*

Finally, considering disruptions, each $e \in E$ will have a cost $taxi_e$ associated with it. This cost represents the financial implication of subsidizing a taxi for passengers traveling on edge e. This value is only considered when a threshold δ of delay is exceeded.

To determine an optimal solution for a TSOPuD instance, different cost functions can be used (see Table 2). In this work, for TSOPuD, we consider a cost function that combines the delay of the trains, the cost of changing tracks, and the compensation costs ($taxi$), where the goal is to minimize its value. This cost function is described in more detail in Section 4.1.4.

4. Overall Approach

In this section, we describe the overall solution to TSOP and TSOPuD. The proposed encodings for TSOP and TSOPuD, and the proposed algorithms are presented in detail in the subsequent subsections.

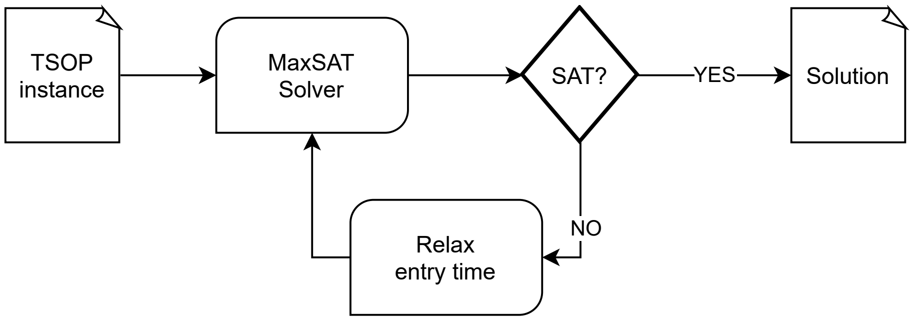


Figure 3: Overview of the solution to solve TSOP.

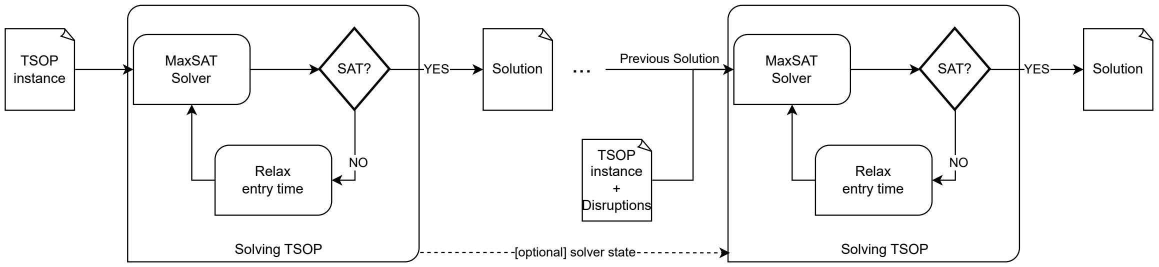


Figure 4: Overview of the solution to solve TSOPuD.

Figure 3 shows the proposed overall approach for solving TSOP. The process starts with the conversion of the TSOP instance into the MaxSAT encoding described in the following subsection. The proposed encoding defines that all trains must arrive on time, according to the TSOP instance provided. However, forcing all the trains to arrive on time may be infeasible. Therefore, a MaxSAT solver is used to determine if the instance has feasible solutions under the current time delay constraints and compute an optimal solution in terms of route penalization constraints, or determine that the instance is unsatisfiable (UNSAT), *i.e.*, there is no feasible solution. If the solver returns UNSAT, then we relax the entry time constraints, allowing a train to not arrive on time, *i.e.*, allowing delays. This relaxation process can be implemented using different strategies. The strategy choice has a direct influence on the execution time of the solver. In this work, we propose three different strategies, further described at the end of this section.

The proposed approach for solving TSOPuD is similar. The overall process is described in Figure 4. The main difference lies in the addition of disruptions that occur after the first solution. Each time a new disruption occurs, the solver needs to be called again, and the same approach for solving TSOP is followed. The input for each iteration can change. In each iteration, we have as input: the previous solution σ, the new TSOP instance with the disruptions χ, and optionally the current solver state. The usage of the previous solution and the solver state depends on the option considered. The solver state basically consists of the current search tree, variable order, lower/upper bounds, clauses learned, and variable assignments. The different options for solving TSOPuD are compared in Section 5.

4.1 MaxSAT Encoding

Our MaxSAT encoding to solve TSOP mainly uses two sets of Boolean variables:

- $r_{\tau,e}$: train τ passes through edge e as part of its chosen train route, with $\tau \in \mathcal{T}$ and $e \in E_\tau$;
- $s^t_{\tau,v}$ represents the entry of train τ in the node v at time t, with $\tau \in \mathcal{T}$, $v \in V_\tau$ and $t \in \gamma_{\tau,v}$.

To simplify the writing of the encoding, we consider the integer variables 1 as True and 0 as False.

4.1.1 Routing Constraints

In this section, we describe the routing constraints of TSOP, establishing that each train must pass through a set of nodes. To ensure that, we add the following exactly-one constraint:

$$\sum_{(v_i,v_j)\in E_\tau, l_{v_j}=m} r_{\tau,(v_i,v_j)} = 1 \quad \forall_{\tau\in\mathcal{T}, m\in\mathcal{L}_\tau} \tag{1}$$

The constraint above forces the train to travel through exactly one edge for which the destination node is a station where the train must stop. Hence, we only need to backtrack from the final destination to compute all the possible routes. Note that this is a recursive definition of a route.

Example 10 *Recall the railway network shown in Figure 1. Constraint (1) generates the constraints* $r_{\tau,(v_1,v_3)} = 1$, $r_{\tau,(v_3,v_4)} + r_{\tau,(v_3,v_5)} = 1$.

Furthermore, we add the following clauses:

$$r_{\tau,(v_i,v_j)} \implies \bigvee_{v_k\in V_\tau,(v_k,v_i)\in E_\tau} r_{\tau,(v_k,v_i)} \ \forall_{\tau\in\mathcal{T},\, (v_i,v_j)\in E_\tau, (\_,v_i)\in E_\tau} \tag{2}$$

In other words, for a train to be able to depart from some node v_i, we need to have traveled through an edge, having that node v_i as the destination, and so on, until we arrive at the starting station of the train. We can do this since the starting and ending stations of a train are fixed *a priori*.

Example 11 *Recall the railway network shown in Figure 1. Constraint (2) generates the constraints* $\neg r_{\tau,(v_3,v_4)} \vee r_{\tau,(v_1,v_3)} \vee r_{\tau,(v_2,v_3)}$, $\neg r_{\tau,(v_3,v_5)} \vee r_{\tau,(v_1,v_3)} \vee r_{\tau,(v_2,v_3)}$ *and* $\neg r_{\tau,(v_3,v_6)} \vee r_{\tau,(v_1,v_3)} \vee r_{\tau,(v_2,v_3)}$. *These three clauses ensure that if the train τ departs from node v_3 then it must have arrived at node v_3 coming from one of its possible entry routes.*

The constraint above ensures that at least one route is used by the train. Therefore, we still need to ensure that the train does not use more than one route. This is only achieved when assigning the entry times to a node (see below).

The above constraints are considered hard constraints in the proposed encoding and must be satisfied. However, we must consider that a route can have a penalization cost associated. Therefore, we define soft clauses representing the penalization of the chosen route. For each $\tau \in \mathcal{T}, (v_i, v_j) \in E_\tau$ with $p_{(v_i,v_j)} \neq 0$, where $p_{(v_i,v_j)}$ is the penalization cost of traveling in (v_i, v_j), we add the unit clause $\neg r_{\tau,(v_i,v_j)}$ with weight equal to $p_{(v_i,v_j)}$. Note that a weighted MaxSAT approach maximizes the sum of the weights of the satisfied soft clauses. By introducing these negated variables as unit soft clauses, we are maximizing the cost of the routes *not* chosen, thus minimizing the cost of the chosen routes.

4.1.2 Time Constraints

In this section, we describe the scheduling constraints of TSOP, establishing that each train must have exactly one entry time associated with each node it passes through. To ensure this, we need to define the set O_i as a set of concurrent nodes which thus cannot be visited by the same train that departs from node v_i (*e.g.*, if $(v_i, v_j), (v_i, v_k) \in E_\tau$ then $v_j, v_k \in O_i$ since a train departing from v_i cannot arrive at both v_j and v_k simultaneously). In this case, we have a choice of routing the train by any of the nodes in O_i. Formally, for each node $v_i \in V_\tau$, we define a set O_i as the set of nodes v_j such that there is an edge $(v_i, v_j) \in E_\tau$. Consider $\mathcal{O}$ as a set of Os for the railway (V_τ, E_τ). Ergo, we add:

$$\sum_{v_j \in O_i} \sum_{t \in \gamma_{\tau,v_j}} s^t_{\tau,v_j} = 1 \ \forall_{\tau \in \mathcal{T},\, O_i \in \mathcal{O}} \tag{3}$$

Example 12 *Recall the railway network shown in Figure 1. For simplicity, let us consider that we schedule the train by the minute. Since v_4 and v_5 are concurrent nodes, constraint (3) generates the following exactly-one constraint: $s^{9:09}_{\tau,v_3}+s^{9:10}_{\tau,v_3}+s^{9:11}_{\tau,v_3}+s^{9:12}_{\tau,v_3}+s^{9:13}_{\tau,v_3}+s^{9:14}_{\tau,v_3}=1$ and $s^{9:36}_{\tau,v_4}+s^{9:37}_{\tau,v_4}+s^{9:38}_{\tau,v_4}+s^{9:39}_{\tau,v_4}+s^{9:40}_{\tau,v_4}+s^{9:41}_{\tau,v_4}+s^{9:42}_{\tau,v_4}+s^{9:36}_{\tau,v_5}+s^{9:37}_{\tau,v_5}+s^{9:38}_{\tau,v_5}+s^{9:39}_{\tau,v_5}+s^{9:40}_{\tau,v_5}+s^{9:41}_{\tau,v_5}+s^{9:42}_{\tau,v_5}=1$. Note that node v_3 does not have other concurrent nodes, and therefore, we consider that there is a set O containing only v_3.*

Considering time constraints, in fact, we do not need to have the entry time for all nodes a train passes through. We can reduce the number of variables by only considering the nodes for which time constraints exist. This change does not remove any solution. Furthermore, the value t can be restricted to a subset of γ^τ_v by checking the time constraints of the train. We can propagate the time constraints and minimum traveling times to reduce the subset.

The relation between two consecutive entry times depends on the traveling time and, thus, on the route taken by the train. So, we define an auxiliary variable q^i_τ that represents the passage of train τ on the route P^i_τ, where $P^i_\tau \in \mathcal{P}_\tau$ is a possible route for train τ. The relation of this variable to the decision variables is given by:

$$q^y_\tau \leftrightarrow \bigwedge_{(v_i,v_j) \in P^y_\tau} r_{\tau,(v_i,v_j)} \qquad \forall_{\tau \in \mathcal{T}, y \in [1..|P_\tau|]} \tag{4}$$

Note that we could define this constraint recursively. However, recursively defining the path would unnecessarily generate a large number of clauses and require longer clauses. Such clauses would allow the solver to choose paths. However, the large size of the clauses would reduce the impact of unit propagation. The impact of computing the set of paths beforehand is small since, in TSOP, most trains have three or fewer possible paths. For different benchmarks, a different encoding could make sense.

Example 13 *Recall the network shown in Figure 1. There are six possible routes. For readability, we only show the clauses generated by constraint (4) for the route starting in node v_1. $q^1_\tau \leftrightarrow r_{\tau,(v_1,v_3)} \wedge r_{\tau,(v_3,v_4)}$, $q^2_\tau \leftrightarrow r_{\tau,(v_1,v_3)} \wedge r_{\tau,(v_3,v_5)}$ and $q^3_\tau \leftrightarrow r_{\tau,(v_1,v_3)} \wedge r_{\tau,(v_3,v_6)}$. Similar clauses are generated for node v_2.*

We still need to ensure that the entry time for two consecutive nodes is consistent. In other words, we need to ensure that the entry time of the second node is equal to the entry time of the first node plus the travel time on the edge between those nodes and plus the minimum stopping time of the train at the first node. The discretization of time has a significant impact on this constraint. As a result, the following constraint is added:

$$q^y_\tau \wedge s^t_{\tau,v_i} \Rightarrow \bigvee_{t' \in [t + t^{min}_{(v_i,v_j)} + t^{stop}_{\tau,v_i};\ t^{latest}_{\tau,v_j}]} s^{t'}_{\tau,v_j} \quad \forall_{\tau \in \mathcal{T},\, t \in \gamma_{\tau,v_i},\, y \in [1 \ldots |P_\tau|],\, (v_i,v_j) \in P^y_\tau} \tag{5}$$

Example 14 *Let us consider a train τ and the edge (v_3, v_4), with a minimum traveling time $t^{min}_{(v_3,v_4)} = 27$ minutes, belonging to the route P^1_τ. Furthermore, let us consider the minimum stopping time at node v_3 of $t^{stop}_{\tau,v_3} = 0$ minutes. Lastly, let us consider that the value of t^{latest}_{τ,v_4} is 9:41, representing the latest arrival time at node v_4. The constraint (5) generates the clauses: $q^1_\tau \wedge s^{9:09}_{\tau,v_3} \Rightarrow (s^{9:36}_{\tau,v_4} \vee \ldots \vee s^{9:41}_{\tau,v_4})$, $q^1_\tau \wedge s^{9:10}_{\tau,v_3} \Rightarrow (s^{9:37}_{\tau,v_4} \vee \ldots \vee s^{9:41}_{\tau,v_4})$, $q^1_\tau \wedge s^{9:11}_{\tau,v_3} \Rightarrow (s^{9:38}_{\tau,v_4} \vee \ldots \vee s^{9:41}_{\tau,v_4})$, $q^1_\tau \wedge s^{9:12}_{\tau,v_3} \Rightarrow (s^{9:39}_{\tau,v_4} \vee \ldots \vee s^{9:41}_{\tau,v_4})$, $q^1_\tau \wedge s^{9:13}_{\tau,v_3} \Rightarrow (s^{9:40}_{\tau,v_4} \vee s^{9:41}_{\tau,v_4})$, $q^1_\tau \wedge s^{9:14}_{\tau,v_3} \Rightarrow s^{9:41}_{\tau,v_4}$. This way, ensuring the entry time of the train τ in the next node has the traveling time in consideration.*

Let us consider two trains $\tau_1, \tau_2 \in \mathcal{T}$ that have the entry times t_{τ_1} and t_{τ_2} in nodes v_i and v_j, respectively. Consider the following constraint that prevents train τ_1 from being in node v_i at time t_{τ_1} if train τ_2 is in node v_2 at time t_{τ_2}, which we instantiate in a few different cases:

$$\neg s^{t_{\tau_1}}_{\tau_1,v_i} \vee \neg s^{t_{\tau_2}}_{\tau_2,v_j} \tag{6}$$

We need to add clause (6) to ensure that the two trains do not collide, considering that $v_i = v_j$ and times of staying are overlapping, *i.e.*, both trains cannot be at the same time at the same node. Hence, we add a pairwise constraint for each pair of entry time variables of different trains in the same node that would cause the trains to collide if both variables were assigned the value true. Formally, the clause (6) is added when $\tau_1, \tau_2 \in \mathcal{T}$, $v_i = v_j$, $v_i \in V_{\tau_1} \cap V_{\tau_2}$, $(v_i, v_n) \in E_{\tau_1}$, $t_{\tau_1} \in \gamma_{\tau_1,v_i}$, $t'_{\tau_1} \in \gamma_{\tau_1,v_n}$, $t_{\tau_2} \in \gamma_{\tau_2,v_j}$, $t_{\tau_1} < t_{\tau_2} < t'_{\tau_1} - t^{min}_{(v_i,v_n)}$[8].

Considering virtual connections[9], there is no need for additional constraints regarding collisions. In this case, collision constraints (two trains in the same place at the same time) are not added for trains with virtual connections on the nodes where the connection occurs. However, transferring passengers/cargo between two trains requires the addition of clause (6). In this case, we add a clause for every entry time for which the minimum connection time is not guaranteed. Note that in this case, $v_i \neq v_j$.

8. $t'_{\tau_1} - t^{min}_{(v_i,v_n)}$ is the exit time of train τ_1 at node v_i. Note that we only define the entry times for each train at each node; thus, we calculate the exit time of train τ_1 at node v_i, based on the entry time of that train at the next node v_n
9. Recall that this type of connection allows the split/merge of trains.

The earliest/latest entry time constraints for a node are initially all considered hard and are ensured by the previous constraints. When the need arises, these constraints are relaxed, and we add the following unit clause: $\neg s^{t}_{\tau,v}$ for all trains and nodes for which t violates the earliest/latest entry constraints. The weight of this constraint is proportional to the delay caused. The need to relax the time constraints is further discussed in detail in the description of the proposed algorithms.

4.1.3 Encoding Disruptions

There are three types of disruptions considered in this work. Next, we describe how these disruptions are encoded into CNF.

The *train slowdown* disruption $\boldsymbol{\varsigma}$ is encoded adding constraint (5) with a modified minimum traveling time for an edge ($\omega_{\boldsymbol{\varsigma},e} \times t^{min}_{e}$), where $\omega_{\boldsymbol{\varsigma},e}$ is the speed slowdown coefficient.

The *block train* disruption $\boldsymbol{\varsigma}$, starting at time $t_{\boldsymbol{\varsigma}}$ with duration $d_{\boldsymbol{\varsigma}}$, is encoded adding a clause :

$$\neg s^{t}_{\tau,v} \;\forall_{\tau \in \mathcal{T}_{\varsigma}, t \in [t_{\varsigma}; t_{\varsigma}+d_{\varsigma}], v \in V_{\varsigma}} \tag{7}$$

Finally, the *block track* disruption $\boldsymbol{\varsigma}$, starting at time $t_{\boldsymbol{\varsigma}}$ with duration $d_{\boldsymbol{\varsigma}}$, affecting the tracks in $E_{\boldsymbol{\varsigma}}$, is encoded adding a clause:

$$\neg s^{t}_{\tau,v_1} \vee \neg r_{\tau,(v_1,v_2)} \;\; \forall_{\tau \in \mathcal{T}_{\varsigma}, t \in [t_{\varsigma}; t_{\varsigma}+d_{\varsigma}], (v_1,v_2) \in E_{\varsigma}} \tag{8}$$

4.1.4 Cost Function for Disruptions

In this work, we aim to design a cost function that reduces the gap between the literature and the real world of train scheduling (Ministry of Land & Tourism, 2019). Our cost function can be seen as an aggregate of all cost functions described in the literature that have an impact in the real world. Furthermore, we added weights based on data from a real-world survey (Ministry of Land & Tourism, 2019).

For readability, we write the cost function using pseudo-Boolean constraints. These constraints are easily encoded into SAT (Eén & Sörensson, 2006). The cost function is as follows:

$$\sum_{\tau \in \mathcal{T}} \left[\alpha \times \sum_{v \in V_{\tau}, t \in \gamma_{\tau,v}} \left[(|t - t^{entry}_{\tau,v}|) \times s^{t}_{\tau,v} \right] \right. \tag{9}$$

$$\left. + \beta \times \sum_{e \in E_{\tau}} \left[|r_{\tau,e} - x_{\tau,e}| \right] + \sum_{(v_i,v_j) \in E_{\tau}} f(v_i,v_j) \right]$$

$$f(v_i,v_j) = \begin{cases} 0 & \text{if}(|t - t^{entry}_{\tau,v_j}|) < \delta \\ taxi_{(v_i,v_j)} & \text{otherwise} \end{cases} \tag{10}$$

Recall that $t^{entry}_{\tau,v}$ represents the scheduled entry time of train τ at node v in the previous solution, and $x_{\tau,e}$ is a Boolean variable representing the scheduled passage of train τ in track e. Consider that $f(v_i,v_j)$ is a cost function associated with the penalization of traveling from v_i to v_j, that α, β are weights for the delay and changes of tracks, and δ is the threshold for the delay from which the passenger is compensated for. We consider the values obtained from the Ministry of Land, Infrastructure, Transport and Tourism (Ministry of Land & Tourism, 2019). This cost function penalizes the delay locally (by station) and not globally since we know the schedule the train should take. Furthermore, we penalize the change of track by a train and consider the compensation for the passenger for delays larger than δ.

4.1.5 Discussion

The usage of two types of decision variables can be seen as redundant. However, it causes a reduction in the number and in size of the clauses. Particularly, they allow us to define routes with fewer clauses. The penalization of routes using only $s^t_{\tau,v}$ variables would require an unnecessarily high number of soft constraints. Finally, not all routes need $s^t_{\tau,v}$ variables (discussed further on), but most of them need $r_{\tau,e}$ variables to identify routes. All possible routes are precomputed along with the constraints used to propagate the train schedule.

4.2 Iterative Algorithm

This section describes the novel iterative algorithms proposed to solve the TSOP and TSOPuD: a simple *iterative algorithm*, a *core-guided algorithm*, and a *core-guided and propagation algorithm.* The *core guided*, and the *core-guided and propagation* algorithms are modified versions of the *iterative algorithm.* Therefore, we will start by describing the *iterative algorithm.*

This work uses a MaxSAT solver to solve routing and scheduling problems together. At first, the only soft constraints in the problem are the constraints corresponding to the penalization of the routes (see Section 4.1.1). Note that the only possible causes for unsatisfiability are the earliest/latest entry time constraints not being satisfied since, initially, they are considered hard.

Given an instance of the problem, considering the encoding previously described, using a MaxSAT solver, we obtain either a solution to the problem or the indication that the problem is unsatisfiable. In case of unsatisfiability, the most straightforward approach is to relax all time constraints, following the overview in Figures 3 and 4. In other words, we expand the domain of all entry time variables in the instance, iteratively considering increments of 30 seconds. In the worst case, we end up with the whole domain on all entry time variables. A new formula is generated with the corresponding extension of the time domain for the next iteration of our approach. During each iteration, we add 30 new entry time variables (30 seconds) to each node, the time constraints clauses are adjusted accordingly, and soft clauses representing the delay introduced with each new variable are added. This approach requires a low number of iterations, but we may have to deal with a large domain in each iteration since all entry time domains are increased. We denote this approach by *iterative.*

We consider iteratively incrementing the domain of the entry time variables until a solution is found, *i.e.*, until the formula is no longer unsatisfiable. If a solution is found, it is optimal (in terms of delay), but we may improve the quality of the solution, in terms of overall cost, by further relaxing the time constraints. Recall that there are two optimization criteria in TSOP: (i) delay and (ii) route cost. Since we are expanding the time domain at each iteration as needed, naturally, the iterative procedure described guarantees optimality on the delay criterion, *i.e.*, the solution produced has the minimum delay possible. Note that

Algorithm 1: *core-guided* Algorithm

```
  Input: An UNSAT core
  Output: A relaxed problem instance
1 for clause ∈ UNSAT do
2 |   update_domain(clause) ▷ Algorithm 2;
3 end
```

the route cost is also being optimized, however, as we are limiting the possible delays due to the domain of time variables, the route cost may not be the best (*i.e.*, better route cost may require higher delays). To ensure the overall global optimality criterion, we perform additional iterations, further relaxing the time constraints in order to improve the route penalty. This will allow searching for solutions with higher delays but lower route penalties.

There are two stopping criteria in all the algorithms proposed in this paper: (i) timeout (no information about the optimality), and (ii) we found a global optimum. Note that we can, sometimes, conclude that we have an optimal solution in terms of the delay, even if the algorithm times out. Moreover, we can stop the algorithm if we reach a set of routes for which it is mathematically impossible to compute a route with a lower penalty. This value is computed beforehand without any time constraints (*i.e.*, we solve the routing problem). This computation is done when computing all the possible routes, where each route has its cost. Furthermore, we can stop the algorithm sooner if the current solution has a delay cost that cannot be compensated by the possible gains of reducing the route cost. Note that we know that the next iteration will have a worse cost in terms of delay.

4.2.1 Core-guided Algorithm

The performance of the algorithm can be improved by only expanding the domain of the variables that are the cause of unsatisfiability. This new approach is called *core-guided.* Note that there is a trade-off between the number of iterations and the complexity of the formulae in each iteration. Even though each iteration has an overhead caused by the MaxSAT solver call, it is easier to solve smaller instances. That is the main difference between the algorithms proposed.

Given an unsatisfiable formula, many SAT solvers can be configured to provide a subset of still unsatisfiable clauses, named the unsatisfiable (UNSAT) core. The UNSAT core may be used to implement a relaxing scheme to make the formula satisfiable. The different MaxSAT solvers tested in this work use a SAT solver incrementally with assumptions (Biere et al., 2009). For this reason, we can extract the UNSAT core from the underlying SAT solver.

Core-guided algorithms have been successfully applied in the past to scheduling problems (Matos, 2018; Matos et al., 2020; Schutt, 2011; Schutt et al., 2013). In the area of train optimization, Matos et al. (2020) followed a similar approach to relax the constraints and improve the lower bound estimation in binary search. Here, we relax a problem instance by adding new variables (with an associated penalty) and constraints.

Algorithm 1 shows the pseudocode of the *core-guided* algorithm, where we start with an UNSAT core. The result is a relaxed instance that is the base for the next iteration. For each clause in the UNSAT core (line 1), we extract the entry time variables ($s^{t_1}_{\tau_1,v_1}$) present

Algorithm 2: update_domain Function

```
    Input: A clause
    Output: A set of nodes that may be affected by the domain change (nodes).
 1  for $s^{t_1}_{\tau_1,v_1} \in clause$ do
 2      $\triangleright\ t_1 \in \gamma_{\tau_1,v_1},\ \gamma_{\tau_1,v_1} := [t^{earliest}_{\tau_1,v_1}; t^{latest}_{\tau_1,v_1}]$
 3      for $s^{t_2}_{\tau_2,v_2} \in clause \wedge t_2 \leq t_1$ do
 4          $\triangleright\ t_2 \in \gamma_{\tau_2,v_2},\ \gamma_{\tau_2,v_2} := [t^{earliest}_{\tau_2,v_2}; t^{latest}_{\tau_2,v_2}]$
 5          if isConnectionConstraint(clause) then
 6              $t^{latest}_{\tau_2,v_2} = \{t_1 + \iota^{(v_1,v_2)}_{\tau_1,\tau_2}\}$ ▷ The trains are always different $\tau_1 \neq \tau_2$;
 7          else
 8              if $(v_1, v_2) \in E_{\tau_1}$ then
 9                  $t^{latest}_{\tau_2,v_2} = \{t_1 + t^{min}_{(v_1,v_2)}\}$;
10              else
11                  $t^{latest}_{\tau_2,v_2} = \{t_1 - t^{min}_{(v_2,v_1)}\}$;
12              end
13          end
14          $nodes := nodes \cup getConnectedNode(v_2)$ ▷ Get all nodes such that
             $(v_2, \_) \in E$;
15      end
16  end
```

in that clause. Each variable extracted from the UNSAT core corresponds to the entry time of train τ_1 in node v_1 at time t_1. Part of this algorithm is common to the *core-guided and propagation* algorithm proposed in this paper and thus explained in Algorithm 2. This way, we avoid duplicated code and showcase what is common to both approaches. Note that the *core-guided* algorithm does not use the output information of Algorithm 2, which will only be used by the *core-guided and propagation* algorithm.

For each two variables[10] (lines 1-3 of Algorithm 2), we compute the number of new time variables needed for each node. This computation depends on the type of clause: connection (clause (6)) or propagation (clause (5)). If the clause relates to a connection, $\tau_1 \neq \tau_2$ as we are considering two different trains. Therefore, we update γ with the entry time of the following node plus the minimum connection time. However, if the clause relates to propagation, both variables relate to the same train ($\tau_1 = \tau_2$), and the nodes v_1 and v_2 are connected via an edge. The update of γ depends on the direction of the edge. We update γ with the entry time of the following (previous) node plus (minus) the minimum travel time between them (lines 8-12 of Algorithm 2). In the end, the new domain of the entry time variables in the affected nodes is enough to solve the cause of unsatisfiability. In addition, soft clauses are added to penalize the delay of the corresponding train departure time.

10. In order to ensure we do not test redundant variables (*e.g.*, $s^{t_1}_{\tau_1,v_1}$, $s^{t_2}_{\tau_1,v_1}$ and $s^{t_2}_{\tau_1,v_1}$, $s^{t_1}_{\tau_1,v_1}$) we ensure order with the condition $t_2 \leq t_1$.

Recall that there are only two possible transitions between iterations (due to infeasibility or optimality). In this case, if we find an optimal solution, then the algorithm falls back to the default setting and increases the domain of the entry time variables by 30 seconds.

Example 15 *Consider a train that must travel through two nodes that are connected by one edge (v_1, v_2). The earliest possible entry for the train on v_1 is 1*PM*. The latest entry time on v_2 is 3*PM*. The minimum traveling time in (v_1, v_2) is 1 hour. We only need two entry time variables for each node since the possible values for v_1 and v_2 are 1*PM*/2*PM *and 2*PM*/3*PM*, respectively.*

*Consider that v_1 is the spot for a connection and that the train must depart after 3*PM *to ensure the minimum connection time. The problem becomes unsatisfiable. The* UNSAT *core contains the clause for the connection. The first iteration extends the domain of the entry time for v_1 to accommodate 3*PM *as a possible time (line 4 of Algorithm 2). However, performing this domain extension is insufficient, and the problem is still unsatisfiable. The new* UNSAT *core, in the second iteration, contains the clauses for the time propagation constraint of (v_1, v_2). Therefore, the domain of the entry time for node v_2 will be expanded (line 7 of Algorithm 2).*

Proof of Algorithm 1 and 2 by Loop Invariant

Let us consider an array representing the UNSAT core, and an *idx* index representing the position of each clause in the array.

Invariant: The subarray UNSAT$[1 : idx]$ does not contain clauses with an entry time variable $s^{t_1}_{\tau_1,v_1}$, such that $t \notin \gamma_{\tau_2,v_2}$ where $t = t_1 + c$, and c is the minimum time between v_1 and v_2. Note that τ_2, v_2, and c depend on the type of constraint (connection or propagation). For example, for propagation clauses $\tau_1 = \tau_2$ and v_2 represents the next stopping point of train τ_1, and for connection clauses $v_1 = v_2$.

Initialization: The index *idx* starts at 0 and the considered subarray contains no elements. Therefore the invariant holds trivially.

Maintenance: At each iteration of the main loop (line 1 of Algorithm 1), the index is updated $idx = idx + 1$, and Algorithm 2 is called with UNSAT$[idx]$ clause. The if statements on lines 5 and 8 of Algorithm 2 check the correct value of c to add to γ_{τ_2,v_2} in order to ensure $t_1 + c \in \gamma_{\tau_2,v_2}$. At the end of each iteration, the Algorithm 2 corrected the domain of entry time variables in clause UNSAT$[idx]$, and the subarray UNSAT$[1 : idx]$ satisfies the invariant.

Termination: The main loop on line 1 of Algorithm 1 terminates when all clauses in UNSAT have been processed, *i.e.*, $idx = \|$UNSAT$\|$. Since the subarray UNSAT$[1 : \|$UNSAT$\|] =$ UNSAT, and the maintenance step ensures the invariant over UNSAT$[1 : idx]$, the invariant holds for the whole UNSAT array.

4.2.2 Core-guided and Propagation Algorithm

The *core-guided* algorithm presented requires more iterations than the *iterative* algorithm, but each iteration implies fewer changes to the time domain. An alternative approach is to predict and avoid the next UNSAT core by propagating delays caused by the changes to the domain. This new approach is called *core-guided and propagation.*

The goal is to reduce the number of iterations by propagating the delay through the railway. This way, the algorithm mitigates the trade-off between the number of iterations and the complexity of the formulae. This algorithm is similar to Algorithm 1 and is shown in Algorithm 3. The expansion results in propagating the delay every time the domain of γ_{τ_2,v_2} (lines 5-13 of Algorithm 2) is changed. Recall that γ is the domain of the entry time variables for a train in a node. To propagate the update of the entry time domain of a node throughout the railway network, we added the cycle in lines 3-8 of Algorithm 3. Therefore, the clauses containing variables for which the entry time domain (γ) changed are iterated in this new cycle (lines 5-7). The number of clauses to iterate increases every time the domain of a variable γ increases. Naturally, the clause that causes this propagation is not added. The process ends when no more values need to be propagated.

Algorithm 3: *core-guided and propagation* Algorithm

```
Input: An UNSAT core
Output: A relaxed problem instance
1 for $unsatClause \in$ UNSAT do
2     $changed := update\_domain(unsatClause)$ ▷ Get the nodes to be propagated
      from Algorithm 2;
3     for $node \in changed$ do
4         $clauses := getClausesOfNode(node)$ ▷ This obtains all clauses relating to
          node $node$;
5         for $c \in clauses$ do
6             $changed := changed \cup update\_domain(c)$ ▷ Add the new nodes;
7         end
8     end
9 end
```

In other words, for each new variable created, we need to check the impact on the train's possible routes and propagate the delay through the railway. Note that some nodes may already support the delay caused by this procedure.

Example 16 *Recall Example 15. The* core-guided and propagation *approach solves the problem instance in just one iteration. The initial* UNSAT *core is the same. However, using the* core-guided and propagation *algorithm, all delays are propagated in the first iteration. For this reason, we add 4*PM *as the domain value of the entry time for v_2 at the same time as we add 3*PM *to the domain of v_1. The solution has a delay of 1 hour since the train enters v_2 at 4*PM.

5. Experimental Evaluation

In this section, we discuss the computational results. First, the experimental setup and evaluation considering the train scheduling problem (TSOP) are presented. Moreover, a comparison with related work is discussed. Next, the generation of disruptions and the experimental evaluation considering the train scheduling problem under disruptions (TSOPuD) are presented.

Table 3: Data set characteristics. #T, #N, and #TN corresponds to the number of trains, nodes, and nodes with time constraints, respectively.

SBB Benchmark without virtual connections									
	P1	P2	P3	P4	P5	P6	P7	P8	P9
# T	4	58	143	148	149	365	467	133	287
# N	318	4 357	8 631	9 323	9 327	38 742	51 807	22 169	34 917
# TN	49	368	932	963	965	2 887	3 700	1 061	2 009

SBB Benchmark with virtual connections									
	TS	TSE	TL1	TL1E	TL1EI	TL2	TL3	TL4	TL5
# T	131	132	447	448	448	448	448	448	448
# N	12 765	12 870	39 657	39 762	39 762	56 201	56 201	51 207	51 207
# TN	1 124	1 128	3 872	3 876	3 876	3 843	3 843	3 843	3 843

	TL6	TL7	TL8	TL9	TL10
# T	448	448	448	448	451
# N	41 156	41 156	64 983	64 983	57 068
# TN	3 843	3 843	3 843	3 843	3 882

In order to evaluate the proposed approach to TSOP and TSOPuD, we considered the SBB benchmark (Abels et al., 2020b; Jordi et al., 2019). The SBB contains 23 real-world instances divided into two data sets: the data set from CrowdAI challenge (Jordi et al., 2019) and the data set from Abels *et al.* (Abels et al., 2020b). The main difference between these data sets is that the first data set does not contain virtual connections.The characteristics of each instance of the SBB benchmark are shown in Table 3. The benchmark with the virtual connections has more trains and a larger railway network. However, we can see that in both data sets, the percentage of nodes with time constraints (TN) is small (around 9%). Therefore, there is a clear advantage of only considering the entry time variables in those nodes.

The evaluations were performed on a computer with Fedora 14, with a 2.6 GHz CPU and 128 Gb of RAM. The proposed approach was executed considering a time limit of 900 seconds. The only time limitation imposed is for the whole execution of the algorithm. The obtained results were verified by external programs provided by SBB (Abels et al., 2020a) and PESPlib (Goerigk, 1989).

5.1 Train Scheduling

In this section, we present the results regarding TSOP. We start by comparing different MaxSAT solvers in order to select the solver that will be used for the following experiments. Then, we present comparison results of the different algorithms proposed in this work. Furthermore, comparisons with related work are also presented.

5.1.1 Comparing MaxSAT solvers

All our iterative approaches rely on a MaxSAT solver. For this reason, the proposed solution is implemented with the top 5 MaxSAT solvers of both complete and incomplete weighted

Table 4: Comparison of the running time (in seconds) for different MaxSAT solvers for the SBB benchmark without virtual connections. The instances are grouped by characteristics.

Instance	P1-P3	P4	P5	P6	P7	P8	P9	**SUM**
TT-Open-WBO-Inc-20 (Nadel, 2020a)	**1.54**	**10**	**28**	**42**	70	32	**190**	**373.54**
Loandra (Berg, Demirović, & Stuckey, 2020, 2019)	1.98	11	**28**	43	78	34	205	400.98
Open-WBO-Inc-complete (Joshi, Kumar, Rao, & Martins, 2019a)	**1.54**	**10**	**28**	**42**	70	32	194	377.54
Open-WBO-Inc-satlike (Joshi, Kumar, Rao, & Martins, 2019b)	1.48	16	36	**42**	**69**	32	**190**	386.48
SATLike-cw (Zhendong Lei, 2020)	1.48	16	36	**42**	70	**31**	**190**	386.56
UWrMaxSat (Piotrow, 2020)	1.87	14	**28**	44	71	**31**	200	389.87
MaxHS (Hickey & Bacchus, 2019)	1.91	12	**28**	46	70	33	**190**	380.91
Maxino (Alviano, 2020)	1.71	15	30	45	79	**31**	200	401.71

Table 5: Comparison of the running time (in seconds) for different MaxSAT solvers for the SBB benchmark with virtual connections. The instances are grouped by characteristics.

Instance	TS*	TL1*	TL2-3	TL4-5	TL6-7	TL8-9	TL10	**SUM**
TT-Open-WBO-Inc-20 (Nadel, 2020a)	**32.5**	75.3	**149.5**	**94.5**	**69.5**	237	**260**	**918.3**
Loandra (Berg et al., 2020, 2019)	**32.5**	**75**	155	97	**69.5**	264.5	295	988.5
Open-WBO-Inc-complete (Joshi et al., 2019a)	33	75.3	150.5	96	**69.5**	239	262	925.3333
Open-WBO-Inc-satlike (Joshi et al., 2019b)	36	81	151.5	99	71.5	235.5	288	962.5
SATLike-cw (Zhendong Lei, 2020)	36	81	158.5	100.5	72	**232**	299	979
UWrMaxSat (Piotrow, 2020)	33	77	152.5	97.5	71.5	249	262	942.5
MaxHS (Hickey & Bacchus, 2019)	34.5	78	150.5	99.5	72.5	250	275	960
Maxino (Alviano, 2020)	34	80.6	156.5	103.5	73	250.5	268	966.1667

tracks of the 2020 competition (Bacchusand, Järvisalo, & Martins, 2020). As our data set is weighted, we did not test our approach with solvers from unweighted tracks.

The results show that there are no significant differences between the performance of these MaxSAT solvers with their default configurations. The difference between the worst and the best solver is only 3% more time (except for the RC2 solver by Ignatiev, Morgado, and Marques-Silva to be explained below).

Table 4 and 5 show a comparison of the running time for different MaxSAT solvers for the SBB benchmark with and without virtual connections, respectively. These results consider the *core-guided and propagation* algorithm presented in Section 4. To improve readability, we grouped the data by instance characteristics. The detailed results are available on our GitHub (`https://github.com/ADDALemos/train-schedule-optimisation`). Table 4 shows the results for the easiest instances, and thus the difference between solvers is less pronounced.

Note that we do not show the results for the RC2 (Ignatiev et al., 2019) solver. RC2 is implemented in Python, and thus we implemented a wrapper to use the solver within our C++ implementation. This wrapper makes the comparison unfair as it adds unnecessary overhead to convert the object from one language to the other. Nevertheless, if one compares only the solver time in each iteration, RC2 is still slower than TT-Open-WBO-Inc-20 – it would require some conversion to C++ that would influence the results.

SATLike-cw (Zhendong Lei, 2020) uses SATLikey (Lei & Cai, 2018) until it fails to improve the current solution in a given time limit. After that, uses TT-Open-WBO-inc-20 solver to improve the solutions further. In most instances, the TT-Open-WBO-inc-20 solver is called and thus takes longer than the TT-Open-WBO-inc-20 natively. The difference

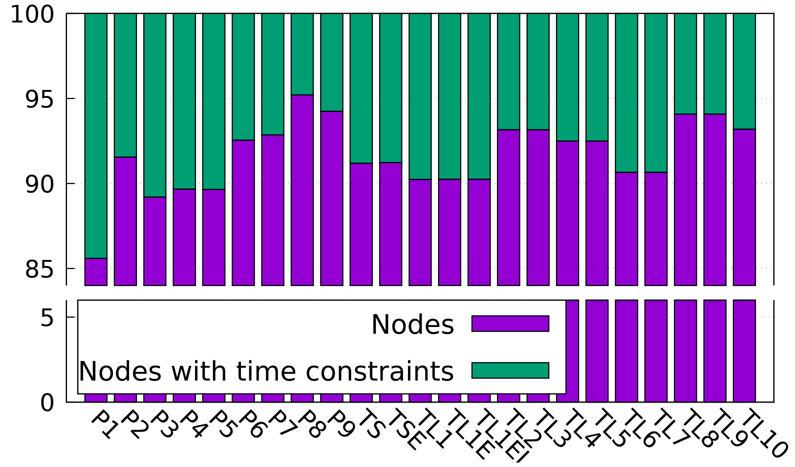


Figure 5: The percentage of nodes with time constraints for each instance in the SBB benchmark. The portion of the graph related to the percentage of nodes with time constraints of 5% to 85% is removed for better readability.

between solvers is particularly noticeable for the largest instances (or if we do not reduce the domain size of entry time variables). We tried different values for the time limit for the local search procedure to no avail. In the future, we could try parameter tuning using dedicated tools with larger data sets (Eggensperger et al., 2019).

The best-performing solver shown in Table 4 and 5 is TT-Open-WBO-Inc-20 (Nadel, 2020a), which is also the winner of the incomplete weight track of the 2020 MaxSAT competition. TT-Open-WBO-Inc-20 (Nadel, 2020a) is based on Open-WBO-Inc-complete (Joshi et al., 2019a; Joshi, Kumar, Rao, & Martins, 2020) and, actually, these are the two best solvers for these benchmarks. Therefore, for the remaining experimental evaluation, we will consider the use of the TT-Open-WBO-Inc-20 (Nadel, 2020a) as the MaxSAT solver. Note that this solver is not complete, *i.e.*, does not guarantee an optimal solution, and thus, our approach will not guarantee an optimal solution when this solver is used. Nonetheless, even considering this incomplete MaxSAT solver, our approach found an optimal solution for the SBB instances.

5.1.2 Comparing Approaches

Considering the SBB instances, Figure 5 shows the percentage of the total number of nodes with associated time constraints. We can see that, on average, only 9% of the total number of nodes deal with time constraints. In the extreme case, concerning the smallest instance, 14% of the nodes have time constraints.

Hence, it is not surprising that the results are better when we only have entry time variables for nodes with time constraints. If we consider the variables in all nodes, we can only solve instance P1. Still, this is not enough if we do not restrict the domain of the entry time variables. Without restricting the domain of the entry variables, there are still 15 timeouts out of the 23 instances. The timeouts are caused by the size of the instance. Similar problems have already been reported in the past (Lemos et al., 2020).

Table 6: The running time in seconds for the different iterative approaches for the instances with optimal cost (in terms of delay) different from 0, considering a time limit of 900 seconds. The number of iterations is shown in parentheses.

	Iterative	Core-guided	Core-guided and Propagation
P5	116 (**3**)	56 (16)	**28** (4)
TSE	56 (**2**)	42 (5)	**36** (3)
TL1E	224 (**3**)	158 (17)	**86** (5)
TL1EI	171 (**2**)	96 (6)	**68** (**2**)
TL10	- (2)	380 (14)	**260** (**7**)

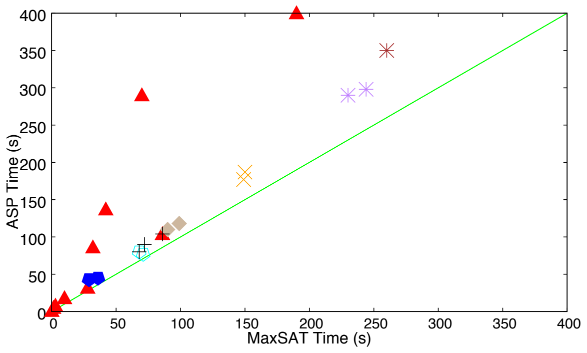
Table 7: The results for the SBB benchmark without virtual connections, considering a time limit of 900 seconds. T, M, and C represent the running time in seconds, memory in gigabytes, and cost, respectively. ExactASP was adapted from Abels et al., 2020b to have an exact cost function.

	ExactASP			ASP (Abels et al., 2020b)			Greedy (Riser, 2018)			MaxSAT			MILP (Baldi, Butun, Kantor, Middelhauve, & Suciu, 2018)		
	T(s)	M(Gb)	C	T(s)	M(Gb)	C	T(s)	M(Gb)	C	T(s)	M(Gb)	C	T(s)	M(Gb)	C
P1	1	0.2	**0**	1	**0.06**	**0**	1	0.1	**0**	**0.14**	**0.06**	**0**	0.7	**0.06**	**0**
P2	8	0.2	**0**	5	0.08	0	4	0.15	**0**	**1.5**	**0.07**	**0**	43	0.5	**0**
P3	18	0.7	**0**	8	0.3	**0**	8	0.72	**0**	**3**	**0.2**	**0**	94	2.2	**0**
P4	38	1	**0.1**	18	0.5	**0.1**	13	0.8	**0.1**	**10**	**0.5**	**0.1**	141	3	0.8
P5	64	1	**33**	32	0.5	**33**	501	1.75	37.3	**28**	**0.5**	**33**	671	5.1	237.6
P6	317	5.1	**0**	137	3.2	**0**	44	1.56	**0**	**42**	**1.2**	**0**	661	7.4	**0**
P7	580	14	**0**	290	6.31	**0**	91	1.8	**0**	**70**	**1.5**	**0**	899	9.24	**0**
P8	142	4.9	**0**	86	3.1	**0**	**31**	1.3	**0**	32	**0.8**	**0**	250	6.8	1.7
P9	-	-	-	400	7.6	**0**	360	**2**	**0**	**190**	**2**	**0**	-	-	-

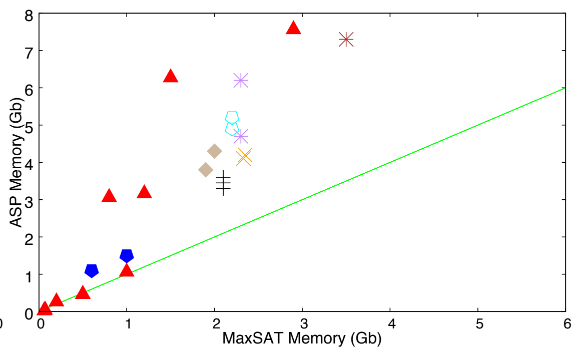
Restricting the domain of the entry variables allows for solving all instances. However, we need to iterate to solve instances that have an optimal cost in terms of delay different from 0 (there are only 5 instances). Note that for instances with optimal cost in terms of delay equal to 0, the problem can be solved within the first MaxSAT call, and no further iterations are necessary.

Table 6 shows the results of the different iterative algorithms for the 5 instances with an optimal cost in terms of delay different from 0. The *iterative* algorithm has fewer iterations needed but expands the domain unnecessarily, which causes a time-out in TL10. With the addition of a *core-guided* process, we can see clear progress in terms of running time. The best algorithm is the *core-guided and propagation* algorithm. The main reason is the fact that we avoid doing unnecessary iterations. Each call to the solver has an overhead, which may not be necessary if one can predict the problem. On average, the first call is really fast since it is easy to prove the unsatisfiability of the instance. However, the calls get slower each time the domains are increased. Therefore, for the remaining experiments we will consider the *core-guided and propagation* algorithm.

Table 7 compares the performance of the proposed solution with the related work on the SBB Crowd Sourcing Challenge benchmark. As previously mentioned in Section 2, the ASP solution uses an approximation of the cost function to deal with the size of the problem. However, this process may remove the optimal solution. To make a fair comparison, we



(a)



(b)

Figure 6: (a) Comparison of the running time (in seconds) and (b) comparison of the memory consumption, between our best solution and the best ASP approach (Abels et al., 2020b) for all SBB data sets. The same symbols/colors symbolize instances with the same overall characteristics. The only exception is the red triangles that represent the P instances (P1 to P9) and not characteristics.

changed the ASP solution (Abels et al., 2020b), to use an exact cost function. We refer to this solution as *ExactASP*.

ExactASP is slower than the approximated version but can prove the optimal solution for all instances but P9. The main difference relies on memory consumption. The approximated version can solve all instances and find the optimal solution for most of them. The difference between the approximation and the optimum is only a few seconds of delay for most instances. However, this approach cannot solve the 4 largest instances of the SBB benchmark with virtual connections.

The greedy approach (Jordi et al., 2019; Riser, 2018) is one of the fastest (the winner of the challenge) and is characterized by keeping the memory low and yet finding a solution with a good cost. The shortcoming lies in the backtracking procedure. Instances P5 and P9 are the only instances that require backtracking and, therefore, more time.

MILP (Jordi et al., 2019; Baldi et al., 2018) can prove optimality for all instances but P9. However, the decomposition removes the actual optimal solution. The choice of routes in the first stage reduces the search space, but it also removes the actual optimal solution. Choosing the route has a direct impact on the overall cost of the solution.

Our approach can solve faster and with less memory than any other approach. The main difference lies in the iterative approach with the smallest domain at the beginning. Furthermore, it is the only approach that solves the exact problem.

Figure 6a compares the running time (in seconds) of our best solution and the best ASP approach (Abels et al., 2020b) for the SBB benchmark. We can see that the MaxSAT approach is faster than the ASP counterpart for all instances. The MaxSAT approach is, on average, twice as fast as the ASP approach, even with the approximation.

Figure 6b compares the memory consumption between our best solution and the best ASP approach (Abels et al., 2020b) for SBB data sets. We can see that the great advantage obtained from the iterative nature of MaxSAT is memory handling. Notice that around the 2Gb mark, we have plenty of instances with the same number of nodes, resources, and

Table 8: #Nodes, #Edges, #Var, and #Const represent the number of nodes, edges, variables, and constraints, respectively. The direct and compact encodings are versions of the same model with and without the pre-processing step.

Instance			Matos et al. (Matos, 2018)		This Paper			
					Direct		Compact	
Name	# Nodes	# Edges	# Var (k)	# Const. (k)	# Var (k)	# Const. (k)	# Var (k)	# Const. (k)
R1L1	3 664	6 386	508	17 268	366	12 700	356	11 557
R1L2	3 668	6 544	530	18 163	385	12 607	375	11 346
R1L3	4 184	7 032	539	18 202	387	13 618	377	12 256
R1L4	4 760	8 529	688	23 749	504	19 440	494	17 496
R2L1	4 156	7 362	585	20 203	429	15 825	419	14 242
R2L2	4 204	7 564	606	21 096	449	15 892	439	14 303
R2L3	5 048	8 287	615	20 795	444	16 987	436	15 288
R2L4	7 660	13 174	990	34 762	744	29 868	736	26 881
R3L1	4 516	9 146	785	28 410	605	24 032	494	21 629
R3L2	4 452	9 252	808	29 356	625	24 978	606	21 481
R3L3	5 724	11 170	933	33 646	717	29 267	697	23 999
R3L4	8 180	15 658	1 284	46 172	986	41 793	974	37 614
R4L1	4 932	10 263	888	32 663	696	28 284	655	22 627
R4L2	5 048	10 755	940	34 763	741	30 384	700	24 307
R4L3	6 368	13 239	1 135	42 079	898	37 700	858	30 160
R4L4	8 384	17 755	1 534	57 005	1 218	52 627	1 175	45 259
BL1	2 688	7 988	536	10 702	329	6 323	326	6 260
BL2	2 606	7 488	504	10 201	304	5 822	301	5 764
BL3	3 044	9 311	603	11 855	389	7 477	382	7 402
BL4	3 816	13 502	764	13 807	595	9 429	593	9 335

trains. The only difference lies in the domain size of the entry time variables. We conclude that the iterative approach effectively reduces the memory footprint.

5.1.3 Periodic Event Scheduling

To further test our approach, we considered the PESPLib (Goerigk et al., 2013) benchmark. Periodic Event Scheduling Problems (PESP) is a generic format to describe scheduling problems. Thus to evaluate our approach with this benchmark, we converted the instances to TSOP format (see Appendix A). The characteristics of each instance of the PESP benchmark is shown in Table 8. Furthermore, we compare the number of variables and constraints of our approach with and without the pre-processing step (explained below) with Matos (2018). One can clearly see that our approach requires fewer variables and constraints (discussed later on).

To the best of our knowledge, there is no state-of-the-art tool publicly available able to solve PESP. Even though there are many SAT approaches in the literature, we choose to compare Matos (2018) for the following reasons. First, the proposed approach is self-contained. In other words, the approach does not require the implementation of customizing heuristics and pre-processing methods. Second, the approach only uses SAT and does not combine multiple tools. Third, the description of the SAT encoding is precise enough to be replicated and implemented.

When converting the PESP instances to the TSOP format, we reduce the size of the network by removing unnecessary nodes *a priori*. A similar approach was proposed

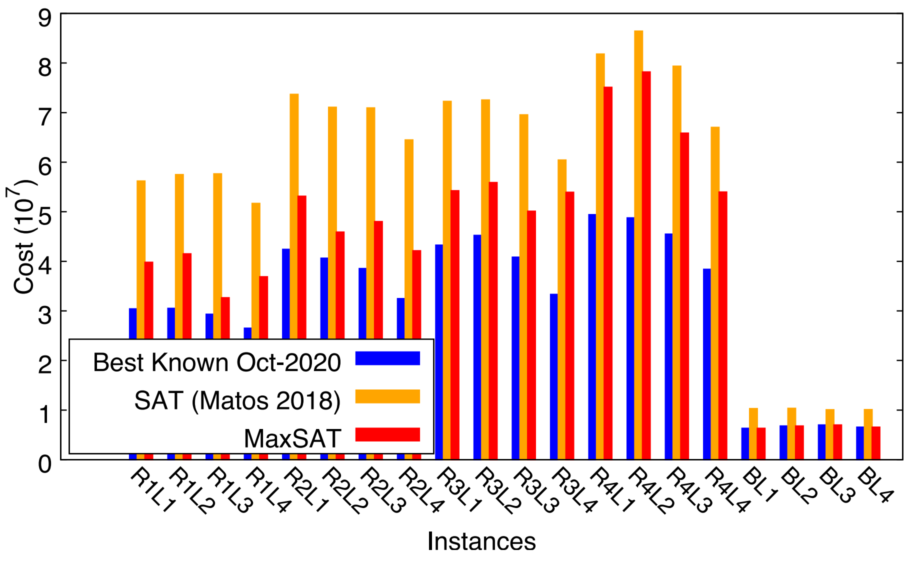


Figure 7: Comparison of the cost found by both Maximum Satisfiability (MaxSAT) approaches and the current best-known values for each instance.

by Borndörfer, Lindner, and Roth (2019). The proposed approach is able to reduce, by 35% and 12%, the size of the networks of the R and the BL instances, respectively. This pre-processing step improves the quality of the solution by 9% for the R instances.

Figure 7 compares the cost of the solution found by both MaxSAT approaches and the current best-known values for each instance. None of these values are known to be optimal. Note that we do not know the time and memory limits for which these values were found. Furthermore, not all values were found by the same tool. The best solution thus far is produced by a concurrent tool specifically designed to solve PESP (Borndörfer et al., 2019). They reduce the problem with pre-processing. The solving process is split into three phases: a SAT solver, ILP solver, and specific heuristics to guide the overall search. The SAT solver is only used to warm-start the ILP solver. Our approach is not able to find the optimal solution within the considered time limit of 900 seconds. However, due to our iterative approach, we are able to find sub-optimal solutions. Our solutions matched the current best-known cost for BL instances, which have specific characteristics. Regarding the rest of the benchmark, we are within 30% of the best value currently known.

The best SAT-based approach was proposed by Matos *et al.* (2018, 2020). Figure 8 compares the running time required to find the best solution using our MaxSAT approach with the approach of Matos (2018). The run time for our approach consists of the solving time of the MaxSAT solver plus the conversion routine and the pre-processing step. The conversion routine represents less than 1% of the whole execution time.

We are able to improve the quality of the solution by 22% and still reduce the running time, on average, by 214 seconds. This can be explained by the smaller size of our encoding. Matos (2018) encoding requires, on average, 1.4× more variables and 1.2× more constraints. This is due to the way Matos (2018) encodes cycles and constraints. They use $q_{x,i}$ variables, meaning that event x starts no later than the time i. On the other hand, we have a variable

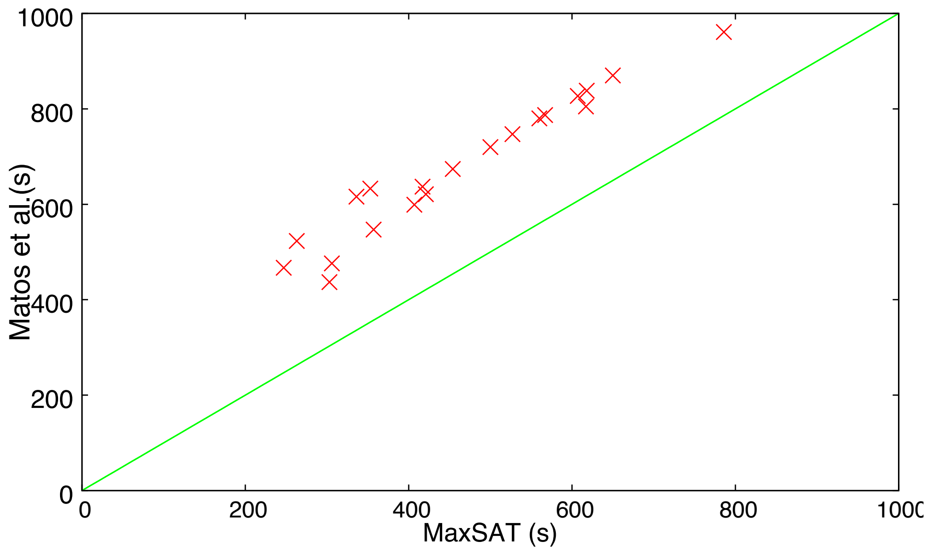


Figure 8: Comparison of the running time to find the best solution between our approach and Matos (2018).

that says the event x starts at time i. The usage of $q_{x,i}$ as a decision variable requires longer clauses. The size of the clauses is known to have a significant impact on a SAT solver performance (Heras, Larrosa, & Oliveras, 2008). Furthermore, the choice of decision variables impacts the complexity of the constraints. This choice causes the encoding to have more exactly-one constraints (20% more for the RL instances and 10% more for the BL instances).

Furthermore, Matos et al. (2020) used a binary search algorithm to optimize the solution of the SAT solver. However, their approach always considers the full time domain of the variables. We do not consider starting with the full time domain, since we have the incremental algorithm. The algorithm allows us to reduce the number of variables since not all time slots are required to solve the instances. The conversion from PESP to TSOP favors our approach since it breaks the cycles and creates room for incrementality on the entry time domain of each node.

Finally, to improve the performance, Matos et al. (2020) proposed a heuristic to approximate the lower bound. This heuristic does not ensure the optimality of the solution.

5.2 Train Scheduling under Disruptions

In this section, we present the results regarding TSOPuD. We start by describing the generation of disruptions to evaluate our approach. Then, we present the results considering recovering from disruptions.

5.2.1 Generating Disruptions

In the past, the impact of large disruptions on the overall public transportation network has been studied (Anagnostopoulos & Moosavi, 2020; Marra & Corman, 2020; Burgholzer

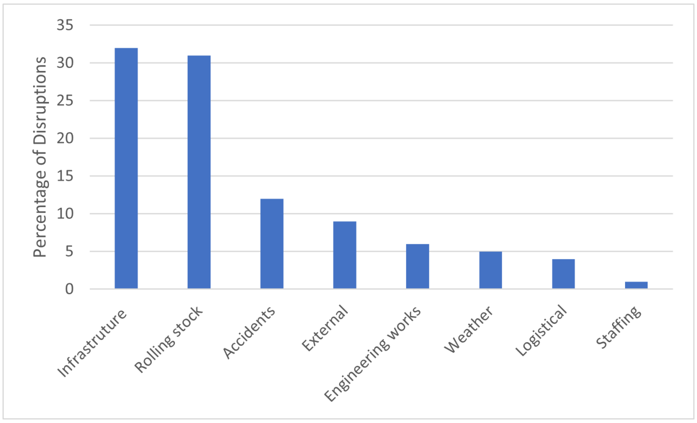


Figure 9: Percentage of disruptions per category during 2019 in Dutch railway network. The data was obtained from rijdendetreinen.nl (accessed November 2020).

et al., 2013). Marra and Corman (2020) used machine learning to identify patterns that impact the number and size of disruptions on the passenger's path. The machine learning method analyzed the real public transportation networks of Zurich. Anagnostopoulos and Moosavi (2020) studied the systemic influence and fragility of all Swiss train stations to disruptions. The goal was to restructure routes and stations in order to reduce the fragility of the schedule. Anagnostopoulos and Moosavi (2020) showed that the most influential stations (the ones used by more train routes) are less fragile than remote stations. In this work, we focus on finding the most common disruptions, their characteristics, and their causes, aiming to create a realistic benchmark to test our re-solving algorithm.

Figure 9 shows the percentage of disruptions per category during 2019 in the Dutch railway network. The disruptions in the Dutch railway network (Büchel et al., 2019) have a direct impact on all railways across Europe. We can see that our model is able to encode 68% of all disruptions that occur in railway networks (the sum of infrastructure, accidents, external, engineering works, weather, and logistical disruptions).

Our data sets and disruption scenarios were obtained from the SBB open data[11]. Figure 10 shows the percentage of trains on-time (green), delayed (orange), and canceled (red) for the SBB trains in Switzerland during one year. Furthermore, Figure 10 shows the average cumulative delay (in minutes), given that it is delayed. The most significant delay in the network that occurred in 2019 was 153 minutes. The delay in each section is computed based on the difference between the scheduled time and the actual arrival time. For this reason, the values are cumulative, and the delay at each point may have occurred in a different section. Note that a delay in a section may cause delays in the subsequent sections of the train route, hence the cumulative delays. Also, as shown in Figure 10, canceled trains are rare and thus were not considered in our solution.

We analyzed the schedule of all trains in Switzerland during 2019 to find the cause of the delay. Hence, we define the following Bayesian probabilities: $P_{station}$ is the probability of a train getting delayed arriving at station *station* knowing that the train was on time

11. The data was extracted from `https://data.sbb.ch/pages/home/`. (accessed November 2020)

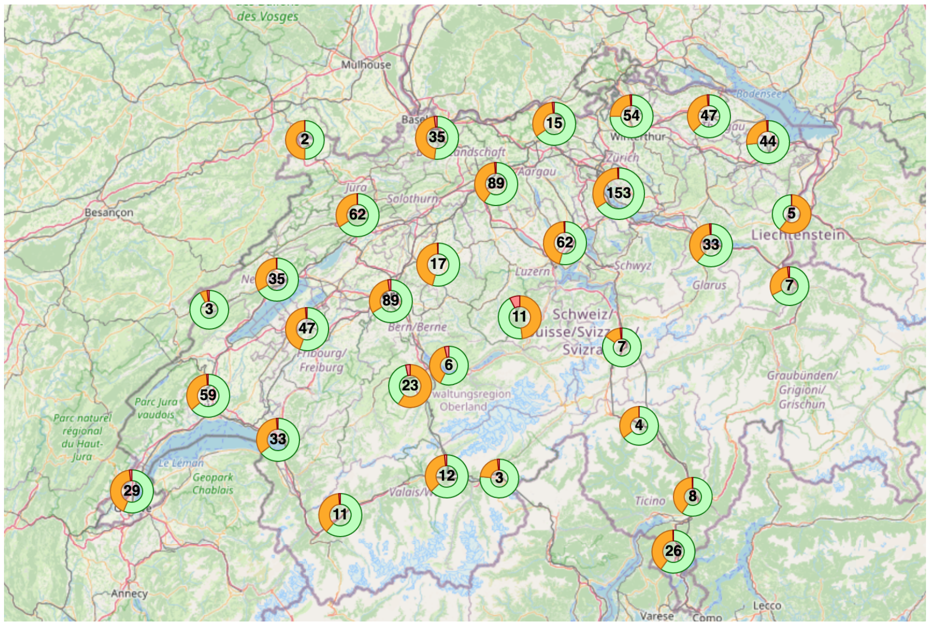


Figure 10: The percentage of trains on time (green), delayed (orange), and canceled (red) for the SBB railway in Switzerland during 2019. For each section, we show the average cumulative delay (in minutes). This Figure was extracted from `https://data.sbb.ch/pages/home/`

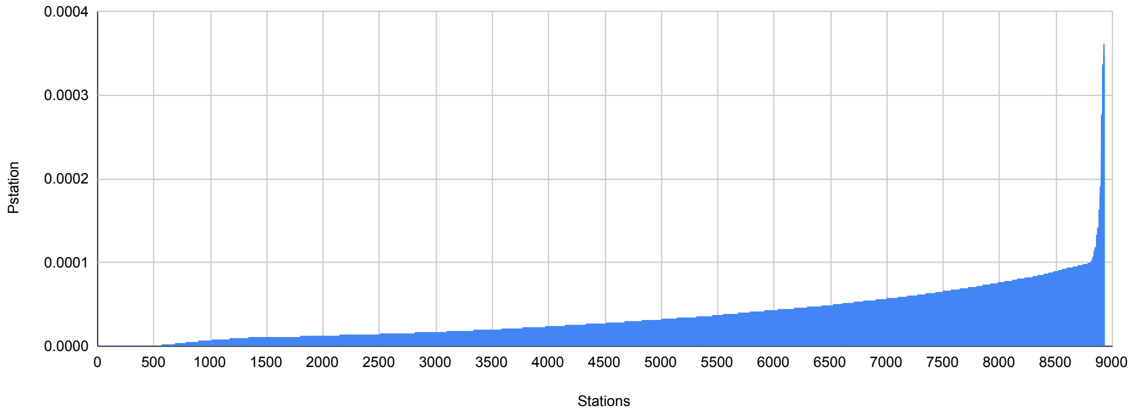
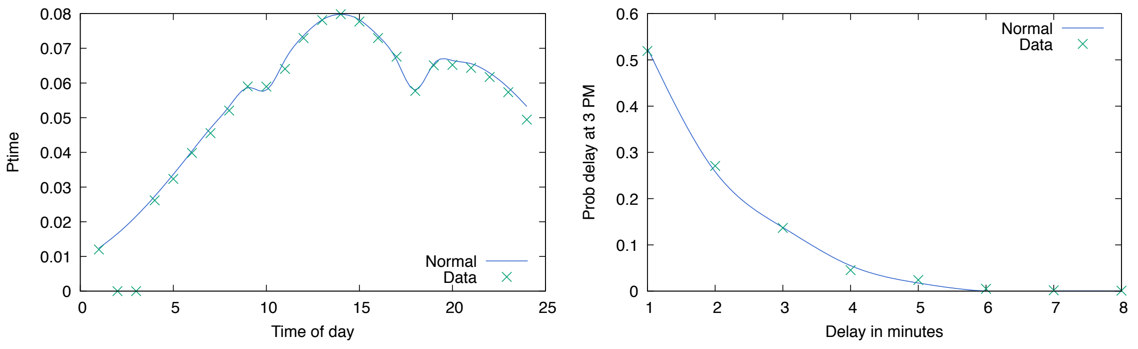


Figure 11: The probability of a train getting delayed at each station, knowing that the train was on time at the previous station. A total of 8 923 stations are considered.

at its previous station; and P_{time} is the probability of a train getting delayed at time $time$ knowing that the train was on time at $time - 1$.

We analyzed the probability $P_{station}$ of a train getting delayed knowing that it was on time at the previous station, considering 8 923 stations. Figure 11 shows the probability $P_{station}$ for the 8 923 stations, where the median value is 2.72×10^{-5} with a variance of



(a) (b)

Figure 12: (a) The probability of a train getting delayed at each time of day, knowing that the train was on time at the previous hour. Line corresponds to the multimodal normal distribution that best fits the data sets. (b) Distribution of the duration of the delay disruption at 3PM, knowing that the train is delayed. Line corresponds to the Poisson distribution with the expected rate of occurrences of 1.4 that best fits the data sets of fluctuations delays in minutes.

9.87×10^{-10}. We use these values to generate the place where the disruption occurs. Figure 12a show the values of P_{time} for each time of the day. One can see that there is a strong relationship between the time of the day and disruptions. Furthermore, we can consider that there are three peaks corresponding to the rush hours of early morning, lunchtime, and late evening. These are times when most trains get delayed. The closest fit is a multimodal normal distribution, and the coefficient of determination is above 0.99. We manually tried multiple types of distributions until we found the best fit. For each distribution, we used the Microsoft Excel Solver (Fylstra et al., 1998) to estimate the parameters. We use the multimodal distribution since the disruptions have three peaks and each peak has its own distribution. The models and data used to create are available on GitHub (`https://github.com/ADDALemos/train-schedule-optimisation`). This way, anyone can reproduce the results and improve the model by adding more data. Note that the data used is from a specific country and thus may have a specific bias from the characteristics of that country. Furthermore, we considered a whole year to reduce a possible relation to different seasons (bad weather tends to occur in the winter). Moreover, we only had data for a single year, which may cause the model to overfit.

This information is not enough to generate the complete set of disruptions. We still need to generate the duration of the disruptions. For this reason, we analyzed the duration of the disruptions that occurred at a specific hour of the day. The closest fit is a Poisson distribution for each time of day, and the coefficient of determination, on average, is above 0.99. Figure 12b shows the distribution of the duration of the delay disruption at 3PM, knowing that the train is delayed. Similar fits can be obtained for other times of the day.

To summarize, we use the probabilities described above to generate 50 different disrupted instances for each type of disruption. The disrupted instance is based on the instances from the SBB data set (23 instances). Therefore, our benchmark with disruptions is composed of two data sets depending on their type (*before*, *during*): the disruptions that occur before the train departure (2 300 instances) and the disruptions that occur after the train departure (3 450 instances). Each data set is composed of *block track* disruptions (1 150 instances), and *slowdown* disruptions (1 150 instances). Additionally, the data set corresponding to disruptions that occur after the train departure has a set of instances with *block train* disruptions (1 150 instances).

5.2.2 Recovering from Disruptions

In this section, we discuss the performance of our algorithm to recover from three different types of disruptions. Our algorithms can solve all disrupted instances with an optimal solution in less than 400 seconds. Recall that the disruption cannot lead to a solution with a better cost than the original one.

Slowdown When considering the disruption (*e.g.* bad weather) that causes a train to slow down, it is difficult to recover from the delay as the disruptions normally affect most edges. The only way to recover is to route through a faster path after or before the affected edges. However, this is rare since the original timetable is already fine-tuned and not robust. Recall that most nodes had an interval where the delay was acceptable. The goal was to reduce the entry time t ensuring $t \in \gamma_{\tau,v}$ (*i.e.*, without arriving too early) for each train τ at each node v. Recall that the maximum entry time at a node in normal conditions is $t^{latest}_{\tau,v}$. We can recover from this disruption without exceeding the maximum delay ($t^{latest}_{\tau,v}$) in 90% of all cases. When considering the *during* disruptions, we can recover by spending, on average, only 16 seconds more to solve the original problem. When considering the *before* disruptions, the algorithm takes an additional 120 seconds, on average. This can be explained by the fact that there is no need to solve the whole problem when solving the disruption while the train is traveling. On average, the number of variables is reduced by 30% considering *during* disruptions when compared to *before* disruptions.

Block track This is the only disruption that we can really improve by re-routing since there are other options. Naturally, the capacity of re-routing depends on the number of parallel routes. Figure 13 shows the execution time (in seconds) for the best algorithm to solve original instances and to recover from disruptions of the *block track* type that occur before and during the travel of the train. For the smaller instances, there is no difference between solving the original instances or the disrupted instances. In general, we need more iterations than in the original search to recover since there will always be a larger delay. This is the main reason for the execution time of disrupted instances to worsen. When considering the *during* disruptions, the number of iterations is compensated by the reduction in the size of the instance. This can be explained by the fact that we significantly reduce the size of the problem since we cannot change it anymore. Naturally, this depends on the location of the train when the disruption occurs.

Block train This disruption can be used to partially model crew problems, as we can block the train due to insufficient staff. This disruption causes a delay, which is quite difficult to recover from. Similar to the train *slowdown* disruption, re-routing the train does not improve the delay in most cases. This is the type of disruption that requires more iterations of the *core-guided and propagation* algorithm to recover. For this reason, this is also the disruption that takes longer to solve.

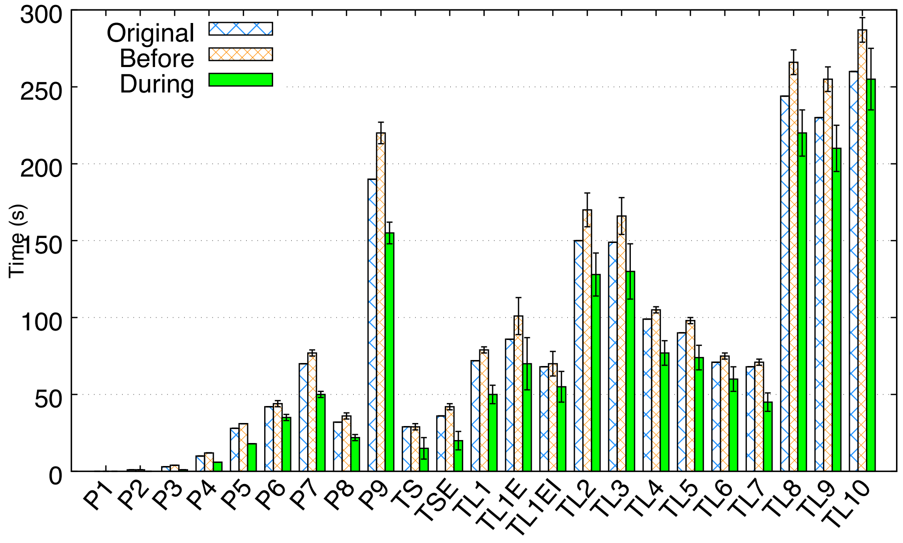


Figure 13: Comparison between execution times to find the original solution and to recover from *before* and *during* disruptions of *block track* type.

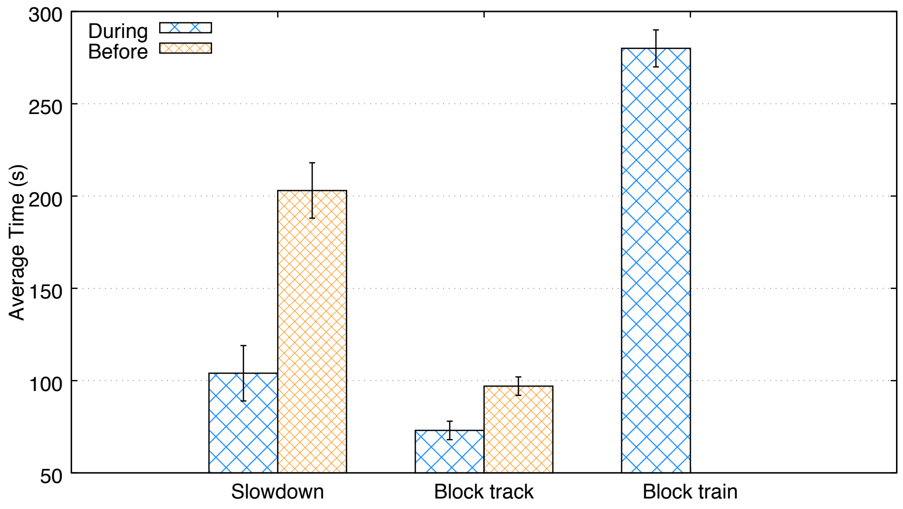


Figure 14: The average recovery time by type of disruptions.

Figure 14 shows the time spent to recover after different types of disruptions occur. *Block track* is the fastest disruption to recover as fewer iterations are needed, and there is more freedom to change the path of the train. This change allows for reducing the number of iterations.

In the worst case, the recovery procedure takes more than 62% of the total execution time. However, we can reduce this value by solving the problem incrementally.

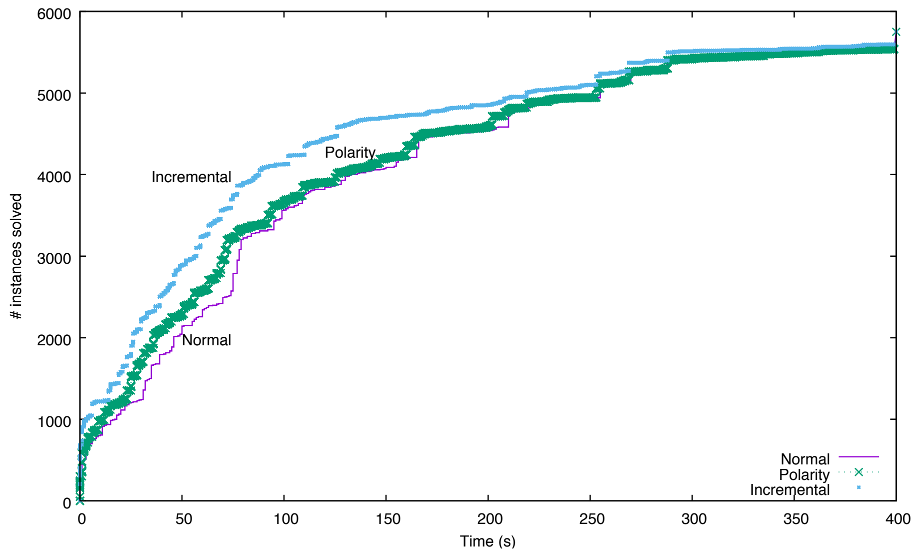


Figure 15: The number of disrupted instances solved as a function of the execution time (in seconds) for each method.

Incremental versus "from scratch" The recovery algorithm spends most of its time solving the original problem again. For this reason, we can reduce this time with two techniques: (i) setting the polarity of the variables; and (ii) solving the problem incrementally.

Setting the polarity (Nadel, 2020b) of the variables is similar to the warm-start technique used in ILP. Its impact depends on the similarity of the original solution and the new optimal solution. This technique has a larger impact when no re-routing is needed.

Solving the problem incrementally requires saving the state of the search and then restarting the search when the disruption occurs. With this approach, we only add new constraints, and therefore all the constraints previously learned are still valid. However, we need to change the cost function and update all the lower and upper bounds used in the search. Consequently, the algorithm may have explored a path in the search tree that is no longer relevant. This *per se* is not a problem but may have a small impact on the incremental search.

Figure 15 shows the number of instances solved with the respective execution time (in seconds) for each method. We can see that setting the polarity of the variables has a small impact. This impact is only effective for smaller instances with a shorter execution time. Also, incremental solving has more impact than setting the polarity of the variables. Nevertheless, the impact is negligible for instances that require more iterations in the *core-guided and propagation* algorithm and more changes to the path. On average, the gain of using incremental solving is 25%.

The performance improvements do not affect the quality of the found solution. The solution found has exactly the same cost as before. The procedure stops only when the optimal solution is found.

5.2.3 Limitations

Considering the process of recovering from disruptions, we cannot create unplanned stops outside the train path and miss connections. Hence, the disruption on a train is propagated to all connected trains. While we allow the train to stop at any node/location along its path for as long as necessary, adding stops outside the path would require knowledge of the global railway network, which is not available in the SBB benchmark.

Additionally, this work does not address disruptions caused by *rolling stock problems*, which may require train replacements during travel. To mitigate such issues, redundant resources can be placed in key locations along the track.

6. Conclusions and Future Work

This paper proposes a novel iterative MaxSAT encoding to solve the train scheduling optimization problem, that takes advantage of the relaxation of the problem.

The experiments show that the optimal solution is found for all SBB instances within 260 seconds, being, on average, twice as fast as the ASP counterpart, while avoiding the exponential growth of memory usage.

All PESPLib instances are solved within 786 seconds. Although we are not able to find an optimal solution within the considered time limit (900 seconds), our iterative approach allows finding a sub-optimal solution within the time limit. The results are a considerable improvement when compared with current MaxSAT solutions. The quality of the solution is improved by 22% compared with the current MaxSAT solutions, and the running time is reduced, on average, by 214 seconds. However, on average, the quality of the solution is 1.3× worse than the current best solution known for the instances.

Finally, we take advantage of the incremental nature of the algorithm to solve TSOPuD. The incremental algorithm performs 25% faster than the traditional approaches.

In future work, we could relax the number of possible parallel routes and add them on demand. In other words, in each iteration, we would consider longer routes. In addition, we can explore more pre-processing techniques to improve the quality of results obtained in the PESP benchmark. Finally, our approach only considers 68% of the possible disruptions. Recall that we do not consider disruptions caused either by staffing or rolling stock problems. We can extend the encoding and the algorithm in order to solve staffing (Demirović et al., 2017) and rolling stock problems.

Furthermore, we can apply the same algorithms proposed in this paper with different core solvers to see if the same performance can be achieved. The method to solve incremental TSOPuD, considering an incremental expansion of the domain of time variables, is general and thus can be applied to different dynamic problems. For this reason, we propose, as future work, to test this method with problems of different domains. In particular, problems that rely on time variables with large domains.

Acknowledgments

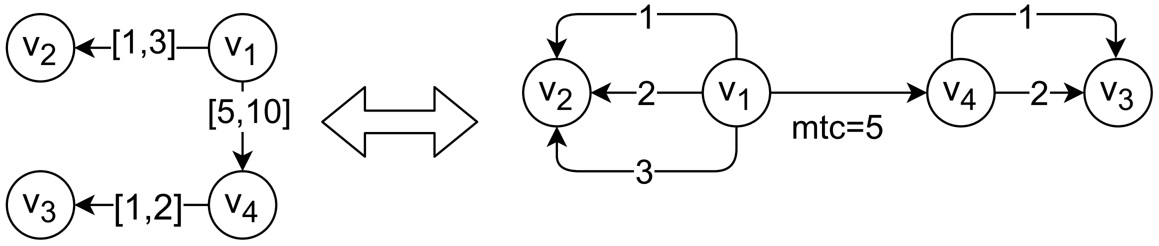


Figure 16: On the left a PESP network with 4 events, 3 constraints, and $\omega = 40$. On the right, the PESP network is converted into TSOP.

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference SFRH/BD/143212/2019 (PhD grant), DSAIPA/AI/0033/2019 (project LAIfeBlood) and UIDB/50021/2020 (INESC-ID multi-annual funding).

Appendix A. Periodic Event Scheduling Problems

A.1 Periodic Event Scheduling Problems

PESP (Borndörfer et al., 2019; Serafini & Ukovich, 1989; van Heuven van Staereling, 2018) can be formally defined as follows. Consider a directed graph $G' = (V', E')$. Each $v \in V'$ corresponds to a periodic event. All events occur with periodicity ω. Each $e \in E'$ corresponds to an order between events (e.g. the edge (a, b) tells us that event a starts before event b) and is characterized by a feasibility interval $[L_e, U_e]$, where $L_e(U_e)$ corresponds to the lower (upper) bound. The interval limits the starting time of the next event (e.g. (a, b) with an interval [1, 2] says the event b cannot start before the time of $a + 1$ but cannot start later than $a + 2$). Consider π_v ($v \in V'$) as the starting time of event v. The *goal* is to find $\pi_{v_i}, \pi_{v_j} \in [0, \omega)$ for every $e = (v_i, v_j) \in E'$ such that $(\pi_{v_j} - \pi_{v_i}) modulo\ \omega \in [L_e, U_e]$. Furthermore, we want to minimize the cost given by $\sum_{(v_i,v_j)\in E'}[((\pi_{v_j} - \pi_{v_i}) modulo\ \omega) - L_e]$.

Example 17 *Figure 16 shows an example of a graph for a PESP instance. An optimal solution to this instance is:* $\pi_{v_1} = 0, \pi_{v_2} = 1, \pi_{v_3} = 6$ *and* $\pi_{v_4} = 5$. *This solution has a cost of 0, given by* $(\pi_{v_2} - \pi_{v_1} - L_{(v_1,v_2)}) + (\pi_{v_4} - \pi_{v_1} - L_{(v_1,v_4)}) + (\pi_{v_3} - \pi_{v_4} - L_{(v_4,v_3)})$.

A.2 Converting PESP into TSOP

TSOP is richer and more complex than PESP. Note that PESP is not a domain-specific problem and therefore can be used to model different problems. PESP has less domain information than TSOP, and PESP instances are simpler. For example, PESP ignores the routing problem, i.e., a train having only one possible path. However, PESP has a large public benchmark, and thus we can use it to test the generality of our approach. PESP formalism is less restrictive and thus leaves room for multiple interpretations. This is the reason PESP can be used to model different problems. For this reason, there can exist many possible approaches to encoding PESP as TSOP. In the literature, there are many possible approaches to encode PESP as train scheduling problems in general (Serafini & Ukovich, 1989). However, these train scheduling problems have their own characteristics, which are different from the ones proposed by Jordi et al. (2019). For this reason, we propose an approach to convert to TSOP, which is described as follows. Different from TSOP, PESP is a cyclic problem. Ergo, we need to break the cycles when converting to TSOP. For example, the edges $\{(a, b), (b, a)\}$ from a PESP instance are split into two routes for different train lines. Recall that the assignment of the train to routes is done beforehand. Two train lines can imply two different trains. However, we need to force the usage of the same train, considering the initial cyclic problem, and thus we add a *virtual connection*. The first train route is a to b, and the second is b to a. The train on the second route must only depart after the train on the first route arrives. Furthermore, the two trains can be simultaneously in node b (representing the same train from the original problem).

The constraints of PESP can be divided into two types: (i) the traveling time of a train between two nodes, and (ii) the connection time between two trains. Yet, a PESP instance does not distinguish them (they are all represented as edges). Hence, we must separate the edges in different train routes and connections. To keep all constraints of the original problem, we must ensure that all nodes belong to a route. However, there are still multiple possible routes. Hence, we define a route as the longest set of edges possible while ensuring that all nodes belong to exactly one route. Note that each route must have more than one node. After defining the routes such that all nodes belong to exactly one route, the edges that connect two different routes correspond to connections between those routes.

Consider n trains in the PESP network. The route of a train τ is represented by E'_τ with $1 \leq \tau \leq n$. V'_τ represents all the nodes in the route of the train τ. Furthermore, consider $E'^{con} \subset E'$ as the set of connections in the network. Let us convert a PESP network $(V',\ E')$ into a TSOP network $(V,\ E)$. Recall that each train τ in the TSOP network has a corresponding sub-graph $(V_\tau,\ E_\tau)$. After the conversion, the nodes remain the same $(V_\tau = V'_\tau)$. The conversion of the edges takes into account the respective feasibility interval. Therefore, for each edge $e \in E'_\tau$ we add e to E_τ for each value of $t \in [L_e, U_e]$. The edges are characterized by the minimal traveling time $(t_e^{min} = t)$ and penalty $(p^e = t - L_e)$.

Now, the only part missing in the TSOP network is the connections. For each edge $(v_i, v_j) \in E'^{con}$ we add a new connection c to the set C_τ where $v_j \in V_\tau$. The connection c is characterized by two trains τ and τ' such that $v_j \in V_\tau$ and $v_i \in V_{\tau'}$. The lower bound of the interval corresponds to the minimum connection time $(\iota_{\tau',\tau}^{(v_i,v_j)})$. The upper bound U_e corresponds to the latest departure time (the arrival time in the next node) for the first train to depart.

Example 18 *Consider the PESP graph shown on the left of Figure 16. This graph represents two trains, τ_1 and τ_2, with a connection. Train τ_1 departs from v_1 and reaches v_2 within the time interval [1,3]. Train τ_2 departs from v_4 and reaches v_3 within the time interval [1,2]. Hence, we can encode the travel time of the trains as two railway networks, as shown in Figure 16. Each value of the time interval corresponds to a new edge with a different minimum traveling time (t_e^{min}). Finally, we must enforce that the departure of τ_2 and the departure of τ_1 are within the time interval [5,10]. We encode this constraint as a connection between the train on the node v_1 and the train on the node v_4. The $\iota_{\tau_1,\tau_2}^{(v_1,v_4)}$ is 5. This is the only acceptable conversion from PESP into TSOP, given that any other definition of routes would either leave independent nodes or use the same node in different* *routes. The entry time intervals are* $\gamma_{\tau_1,v_1} = [0]$ *(as it has no constraints),* $\gamma_{\tau_1,v_2} = [1,3]$ *(due to the traveling time),* $\gamma_{\tau_2,v_4} = [5]$ *(due to the connection), and* $\gamma_{\tau_2,v_3} = [6,7]$ *(due to the traveling time and connection). These are the smallest entry time intervals one can compute without solving the problem.*

References

Abels, D., Jordi, J., Ostrowski, M., Schaub, T., Toletti, A., & Wanko, P. (2020a). *SBB Train Scheduling Validator.* `https://github.com/potassco/train-scheduling-with-hybrid-asp/releases` (accessed Apr 10, 2020).

Abels, D., Jordi, J., Ostrowski, M., Schaub, T., Toletti, A., & Wanko, P. (2020b). Train scheduling with hybrid answer set programming..

Acuna-Agost, R., Michelon, P., Feillet, D., & Gueye, S. (2011). SAPI: Statistical analysis of propagation of incidents. a new approach for rescheduling trains after disruptions. *European Journal of Operational Research*, *215*(1), 227–243.

Adenso-Díaz, B., Oliva González, M., & González-Torre, P. (1999). On-line timetable rescheduling in regional train services. *Transportation Research Part B: Methodological*, *33*(6), 387 – 398.

Aken, S. V., Bešinović, N., & Goverde, R. M. (2017). Solving large-scale train timetable adjustment problems under infrastructure maintenance possessions. *Journal of Rail Transport Planning & Management*, *7*(3), 141–156.

Alviano, M. (2020). Maxino. In *Proceedings of the MaxSAT Evaluations*, pp. 21–22.

Anagnostopoulos, G., & Moosavi, V. (2020). Stationrank: Aggregate dynamics of the swiss railway. *CoRR*, *abs/2006.02781.*

Andersson, E. V. (2014). *Assessment of robustness in railway traffic timetables.* Ph.D. thesis, Linköping University Electronic Press.

Ansótegui, C., & Manyà, F. (2004). Mapping problems with finite-domain variables into problems with boolean variables. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, Vol. 3542, p. 1–15.

Artigues, C., Bourreau, E., Jost, V., Kedad-Sidhoum, S., & Ramond, F. (2018). Trains do not vanish: the ROADEF/EURO challenge 2014. *Annals OR*, *271*(2), 1091–1105.

Bacchusand, F., Järvisalo, M., & Martins, R. (Eds.). (2020). *MaxSAT Evaluation 2019 Solver and Benchmark Descriptions.* Department of Computer Science Report Series B. Department of Computer Science, University of Helsinki, Finland.

Bajestani, M. A., & Beck, J. C. (2013). Scheduling a dynamic aircraft repair shop with limited repair resources. *Journal of Artificial Intelligence Research*, *47*, 35–70.

Baldi, F., Butun, H., Kantor, I., Middelhauve, L., & Suciu, R. (2018). *MILP: Train Schedule Solver.* `https://github.com/iquadrat/sbb-train-scheduler` (accessed Jan 30, 2020).

Bayardo Jr., R. J., & Schrag, R. (1997). Using CSP look-back techniques to solve real-world SAT instances. In Kuipers, B., & Webber, B. L. (Eds.), *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI) and Ninth Innovative Applications of Artificial Intelligence Conference (IAAI)*, pp. 203–208. AAAI Press / The MIT Press.

Berg, J., Demirović, E., & Stuckey, P. (2019). Core-boosted linear search for incomplete MaxSAT. In *Proceedings of 16th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, Vol. 11494 of *Lecture Notes in Computer Science*, pp. 39–56. Springer.

Berg, J., Demirović, E., & Stuckey, P. (2020). Loandra in the 2020 maxsat evaluation. In *Proceedings of the MaxSAT Evaluations*, pp. 10–11.

Berger, R. (2018). *World Rail Market Study forecast 2018 to 2023*. The European Rail Supply Industry Association (UNIFE).

Biere, A., Heule, M., & van Maaren, H. (2009). *Handbook of satisfiability*, Vol. 185. IOS press.

Binder, S., Maknoon, Y., & Bierlaire, M. (2017). The multi-objective railway timetable rescheduling problem. *Transportation Research Part C: Emerging Technologies*, *78*, 78–94.

Borndörfer, R., Klug, T., Lamorgese, L., Mannino, C., Reuther, M., & Schlechte, T. (Eds.). (2018). *Handbook of Optimization in the Railway Industry*. Springer.

Borndörfer, R., Lindner, N., & Roth, S. (2019). A concurrent approach to the periodic event scheduling problem. In *Journal of Rail Transport Planning & Management*, p. 100175.

Büchel, B., Partl, T., & Corman, F. (2019). The disruption at rastatt and its effects on the swiss railway system. In *Proceedings of 8th International Conference on Railway Operations Modelling and Analysis (ICROMA)*, RailNorrköping, pp. 201–218. Linköping Electronic Conference.

Budai, G., Maróti, G., Dekker, R., Huisman, D., & Kroon, L. G. (2010). Rescheduling in passenger railways: the rolling stock rebalancing problem. *Journal Scheduling*, *13*(3), 281–297.

Buljubašić, M., Vasquez, M., & Gavranović, H. (2017). Two-phase heuristic for SNCF rolling stock problem. *Annals of Operations Research*, *271*(2), 1107–1129.

Burgholzer, W., Bauer, G., Posset, M., & Jammernegg, W. (2013). Analysing the impact of disruptions in intermodal transport networks: A micro simulation-based model. *Decis. Support Syst.*, *54*(4), 1580–1586.

Caimi, G., Kroon, L. G., & Liebchen, C. (2017). Models for railway timetable optimization: Applicability and applications in practice. *Journal of Rail Transport Planning & Management*, *6*(4), 285–312.

Caprara, A., Fischetti, M., & Toth, P. (2002). Modeling and solving the train timetabling problem. *Operations Research*, *50*(5), 851–861.

Caprara, A., Toth, P., Vigo, D., & Fischetti, M. (1998). Modeling and solving the crew rostering problem. *Oper. Res.*, *46*(6), 820–830.

Corman, F., D'Ariano, A., Pacciarelli, D., & Pranzo, M. (2012). Optimal inter-area coordination of train rescheduling decisions. *Transportation Research Part E: Logistics and Transportation Review*, *48*(1), 71 – 88. Select Papers from the 19th International Symposium on Transportation and Traffic Theory.

Davis, M., & Putnam, H. (1960). A computing procedure for quantification theory. *J. ACM*, *7*(3), 201–215.

Demirović, E., Musliu, N., & Winter, F. (2017). Modeling and solving staff scheduling with partial weighted maxSAT. *Annals of Operations Research*, *275*(1), 79–99.

Eén, N., & Sörensson, N. (2006). Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, *2*(1-4), 1–26.

Eggensperger, K., Lindauer, M., & Hutter, F. (2019). Pitfalls and best practices in algorithm configuration. *J. Artif. Intell. Res.*, *64*, 861–893.

Fang, W., Yang, S., & Yao, X. (2015). A survey on problem models and solution approaches to rescheduling in railway networks. *IEEE Transactions on Intelligent Transportation Systems*, *16*(6), 2997–3016.

Fischetti, M., & Monaci, M. (2017). Using a general-purpose mixed-integer linear programming solver for the practical solution of real-time train rescheduling. *European Journal of Operational Research*, *263*(1), 258 – 264.

Fylstra, D. H., Lasdon, L. S., Watson, J., & Waren, A. D. (1998). Design and use of the Microsoft Excel solver. *Interfaces*, *28*(5), 29–55.

Geiger, M. J., Kletzander, L., & Musliu, N. (2019). Solving the torpedo scheduling problem. *Journal of Artificial Intelligence Research*, *66*, 1–32.

Goerigk, M. (1989). *Verification of the PESPlib library.* http://num.math.uni-goettingen.de/ m.goerigk/pesplib/programs/verification.cpp (accessed Aug, 2020).

Goerigk, M., Schachtebeck, M., & Schöbel, A. (2013). Evaluating line concepts using travel times and robustness. *Public Transport*, *5*(3), 267–284.

Heras, F., Larrosa, J., & Oliveras, A. (2008). MiniMaxSAT: An efficient weighted Max-SAT solver. *Journal of Artificial Intelligence Research*, *31*, 1–32.

Hickey, R., & Bacchus, F. (2019). Speeding up assumption-based SAT. In *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, Vol. 11628 of *Lecture Notes in Computer Science*, pp. 164–182. Springer.

Higgins, A., Kozan, E., & Ferreira, L. (1996). Optimal scheduling of trains on a single line track. *Transportation Research Part B: Methodological*, *30*(2), 147 – 161.

Hoffmann, K., Buscher, U., Neufeld, J. S., & Tamke, F. (2017). Solving practical railway crew scheduling problems with attendance rates. *Business & Information Systems Engineering*, *59*(3), 147–159.

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

Jordi, J., & Mohanty, S. (2018). *Train Schedule Optimisation Challenge.* https://www.crowdai.org/challenges/train-schedule-optimisation-challenge (accessed Apr 15, 2020).

Jordi, J., Toletti, A., Caimi, G., & Schupbach, K. (2019). Applied timetabling for railways: Experiences with several solution approaches. In *Proceedings of 8th International Conference on Railway Operations Modelling and Analysis (ICROMA)*, RailNorrköping, pp. 1–9. Linköping Electronic Conference.

Joshi, S., Kumar, P., Rao, S., & Martins, R. (2019a). Open-wbo-inc: Approximation strategies for incomplete weighted maxsat. *J. Satisf. Boolean Model. Comput.*, *11*(1), 73–97.

Joshi, S., Kumar, P., Rao, S., & Martins, R. (2019b). Open-WBO-Inc: Approximation strategies for incomplete weighted MaxSAT. *Journal on Satisfiability, Boolean Modeling and Computation*, *11*(1), 73–97.

Joshi, S., Kumar, P., Rao, S., & Martins, R. (2020). Open-wbo-inc in maxsat evaluation 2020. In *Proceedings of the MaxSAT Evaluations*, pp. 26–27.

Lei, Z., & Cai, S. (2018). Solving (weighted) partial maxsat by dynamic local search for SAT. In Lang, J. (Ed.), *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1346–1352.

Lemos, A., Monteiro, P. T., & Lynce, I. (2020). Minimal perturbation in university timetabling with maximum satisfiability. In *Proceedings of 17th International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, Vol. 12296, pp. 317–333.

Li, X., Shou, B., & Ralescu, D. A. (2014). Train rescheduling with stochastic recovery time: A new track-backup approach. *IEEE Trans. Syst. Man Cybern. Syst.*, *44*(9), 1216–1233.

Liebchen, C., & Möhring, R. H. (2007). The modeling power of the periodic event scheduling problem: Railway timetables — and beyond. In *Algorithmic Methods for Railway Optimization*, pp. 3–40. Springer.

Lusby, R. M., Larsen, J., & Bull, S. (2018). A survey on robustness in railway planning. *European Journal of Operational Research*, *266*(1), 1 – 15.

Luteberget, B., Claessen, K., & Johansen, C. (2018). Design-time railway capacity verification using SAT modulo discrete event simulation. In *Formal Methods in Computer Aided Design (FMCAD) 2018*, pp. 1–9.

Marques-Silva, J. P., & Sakallah, K. A. (1996). GRASP - a new search algorithm for satisfiability. In Rutenbar, R. A., & Otten, R. H. J. M. (Eds.), *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, pp. 220–227. IEEE Computer Society / ACM.

Marra, A. D., & Corman, F. (2020). From delay to disruption: Impact of service degradation on public transport networks. *Transportation Research Record: Journal of the Transportation Research Board*, *2674*(10), 886–897.

Matos, G. P. (2018). Optimisation of periodic train timetables. Master's thesis, Instituto Superior Técnico, Portugal.

Matos, G. P., Albino, L. M., Saldanha, R. L., & Morgado, E. M. (2020). Solving periodic timetabling problems with SAT and machine learning..

Ministry of Land, Infrastructure, T., & Tourism (2019). *White Paper on Land, Infrastructure, Transport and Tourism in Japan.*

Molloy, K. (2020). *Artificial Intelligence in Train Scheduling Problems.* Ph.D. thesis, Manchester Metropolitan University.

Nadel, A. (2020a). On optimizing a generic function in sat. In *Formal Methods in Computer Aided Design (FMCAD).*

Nadel, A. (2020b). Polarity and variable selection heuristics for sat-based anytime maxsat. *Journal on Satisfiability, Boolean Modeling and Computation*, *12*(1), 17–22.

Piotrow, M. (2020). Uwrmaxsat: an efficient solver in maxsat evaluation 2020. In *Proceedings of the MaxSAT Evaluations*, pp. 34–35.

Riser, M. (2018). *Train Schedule Solver.* `https://github.com/iquadrat/sbb-train-scheduler` (accessed Jan 30, 2020).

Sato, K., Tamura, K., & Tomii, N. (2013). A MIP-based timetable rescheduling formulation and algorithm minimizing further inconvenience to passengers. *Journal of Rail Transport Planning & Management*, *3*(3), 38–53.

Schachtebeck, M., & Schöbel, A. (2010). To wait or not to wait—and who goes first? delay management with priority decisions. *Transportation Science*, *44*(3), 307–321.

Schreiber, D. (2021). Lilotane: A lifted SAT-based approach to hierarchical planning. *Journal of Artificial Intelligence Research*, *70*, 1117–1181.

Schutt, A. (2011). *Improving scheduling by learning.* Ph.D. thesis, The University of Melbourne.

Schutt, A., Feydy, T., Stuckey, P. J., & Wallace, M. G. (2013). Solving RCPSP/max by lazy clause generation. *Journal of Scheduling*, *16*(3), 273–289.

Sels, P., Dewilde, T., Cattrysse, D., & Vansteenwegen, P. (2016). Reducing the passenger travel time in practice by the automated construction of a robust railway timetable. *Transportation Research Part B: Methodological*, *84*, 124 – 156.

Serafini, P., & Ukovich, W. (1989). A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, *2*(4), 550–581.

Tseng, Y.-Y., & Verhoef, E. T. (2008). Value of time by time of day: A stated-preference study. *Transportation Research Part B: Methodological*, *42*(7-8), 607–618.

van Heuven van Staereling, I. (2018). Tree decomposition methods for the periodic event scheduling problem. In *18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, (ATMOS)*, Vol. 65 of *OASICS*, pp. 6:1–6:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Veelenturf, L. P., Kidd, M. P., Cacchiani, V., Kroon, L. G., & Toth, P. (2016). A railway timetable rescheduling approach for handling large-scale disruptions. *Transportation Science*, *50*(3), 841–862.

Warners, J. P. (1998). A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, *68*(2), 63–69.

Zeng, A. Z., Durach, C. F., & Fang, Y. (2012). Collaboration decisions on disruption recovery service in urban public tram systems. *Transportation Research Part E: Logistics and Transportation Review*, *48*(3), 578–590.

Zhendong Lei, S. C. (2020). Satlike-c(w): Solver description. In *Proceedings of the MaxSAT Evaluations*, p. 15.