

Why Start Over: Incremental MaxSMT for Operational Railway Scheduling

Thomas Dubach   

IVT Transport Systems Group, ETH Zurich, Switzerland

Florian Flükiger-Fuchs  

IVT Transport Systems Group, ETH Zurich, Switzerland

Bernardo Martin-Iradi   

IVT Transport Systems Group, ETH Zurich, Switzerland

Abstract

Railway timetables are planned periodically, but operational planners continuously need to deviate from this base: extra trains are added, or infrastructure changes are made, requiring local updates to the existing schedule. We study this process as a sequential extra train scheduling problem, in which requests arrive one at a time and must be routed and timed against the existing timetable while minimising deviation from it. Commonly, each admitted train is fixed before the next request arrives, and all accumulated solver knowledge is discarded between calls.

We propose an incremental MaxSAT framework over Integer Difference Logic (MaxSMT) that retains accumulated solver state across successive extra-train additions, rather than solving from scratch after each request. We evaluate the approach on the Rhätische Bahn network using different solving strategies and compare against a MIP baseline. Retained solver state substantially reduces solve time for successive additions, increasingly so as more extra trains are admitted, and is an order of magnitude faster than MIP given the same setup. Beyond solving time, we show that jointly re-optimising over all admitted extra trains yields better cumulative objective values, and that our approach achieves this while prioritising adjustments in the timetable region affected by the new request.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Applied computing → Transportation; Theory of computation → Integer programming

Keywords and phrases MaxSAT, operational railway timetabling, incremental solving

Digital Object Identifier 10.4230/OASICS.ATMOS.2026.4

Funding *Thomas Dubach*: ETH Mobility Initiative – Project LeRaBe

1 Introduction

Periodic timetables are the backbone of modern railway operations. By repeating at regular intervals, they support efficient vehicle and crew rostering, consistent passenger information, and the predictability on which connecting services depend [21, 15]. Yet the periodic structure is an idealisation: in practice, railway operations continuously require local deviations from the periodic base. Extra services are added to accommodate seasonal demand, or infrastructure works impose temporary capacity restrictions. These adaptations are a routine part of the operational planning process, which runs from the moment the yearly timetable is fixed until the operating day [1, 3].

This paper addresses one part of this *operational timetabling* process: given an existing periodic timetable and a sequence of extra train requests, produce a locally adapted aperiodic schedule that accommodates each request while minimising aggregate delays from the periodic base. Each request must be routed and timed on a microscopic infrastructure model without creating resource occupation overlaps with the context trains or with previously admitted extra trains.



© Thomas Dubach, Florian Flükiger-Fuchs, and Bernardo Martin-Iradi;
licensed under Creative Commons License CC-BY 4.0

26th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2026).

Editors: Valentina Cacchiani and Stefan Funke; Article No. 4; pp. 4:1–4:19



OpenAccess Series in Informatics

OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This operational planning process has a structure that existing solution methods do not exploit. Requests arrive and are processed sequentially, each producing a locally revised schedule that serves as the reference for the next. When a new request concerns a region of the network already affected by earlier requests, the knowledge developed while solving those earlier problems remains relevant: the infrastructure has not changed, and the combinations of scheduling decisions that were infeasible before are still infeasible now. A solver that is reset between successive requests must rediscover this knowledge from scratch; a solver that retains it can apply it directly to the new problem.

We exploit this structure through an incremental Maximum Satisfiability (MaxSAT) framework over Integer Difference Logic (IDL), which we will call MaxSMT for the remainder of this paper. Building on the SAT+IDL architecture of [14], we add a MaxSAT optimisation layer that minimises timetable delays. The solver is never reset between successive extra train additions. Instead, it carries over its accumulated state: (1) the learnt cuts, which record which combinations of scheduling decisions are proven infeasible, (2) its lower bound proof, which certifies the minimum objective cost reached so far, and (3) the branching decision heuristic, which orients the search toward solutions close to the one previously found.

MIP warm-starting differs: a standard MIP solver can carry over incumbent solutions between solves, but it discards its learnt cuts, its dual bound proof, and the branch-and-bound search tree on each invocation.

The paper makes the following contributions:

- A MaxSMT formulation for operational timetabling on a microscopic infrastructure model, combining Integer Difference Logic for continuous event-time variables with soft clauses penalising delays from a periodic reference timetable.
- An incremental solving architecture in which the full accumulated solver state is retained across successive extra train additions rather than discarded between solves.
- Empirical evidence that retained solver state in MaxSMT substantially reduces median total solve time, with the gain growing as the number of additions increases, while remaining an order of magnitude faster than a state-of-the-art commercial solver under the same solving strategy.
- A finding that allowing joint re-optimisation over the full set of extra train requests, rather than committing to the schedule of each train after a request, yields strictly better cumulative objective values, and that the MaxSMT solver achieves this without sacrificing its speed advantage.
- A finding that the MaxSMT solver, guided by retained search state from the prior solve, tends to concentrate revisions in the timetable region affected by each new request.

The remainder of the paper is structured as follows. Section 2 reviews the relevant literature on operational timetabling and MaxSAT in timetabling, then identifies the research gap this work addresses. Section 3 describes the event-activity model and the MaxSMT encoding. Section 4 presents the incremental solving architecture and the solving strategies evaluated. Section 5 describes the experimental setup on the Rhätische Bahn network, a metre-gauge railway in south-eastern Switzerland. Section 6 reports and analyses the results.

2 Literature Review

We begin by establishing the incremental process of the operational railway planning and situating the present work within the existing timetabling literature. We then review MaxSAT approaches to train scheduling. We conclude by identifying the gap that the present work addresses.

2.1 Operational Timetabling as an Incremental Process

In practice, railway timetables are not constructed in a single pass but emerge through successive refinement across planning horizons of decreasing length and increasing detail [1, 2, 3]. At each stage, the output of the previous stage serves as a starting point that is locally adapted rather than reconstructed from scratch. At the operational layer, this means adjusting an existing timetable to accommodate new constraints or requests (e.g. non-periodic extra services, track possessions, or changed demand) while minimising delays from the base schedule. Each new request triggers a local revision of the schedule constructed so far, making the operational planning process inherently incremental.

Several papers have studied this adjustment problem; we discuss the two most directly relevant. Kloster et al. [12] study an interactive planning workflow in which a route planner makes successive modifications to a yearly timetable, such as adding an extra train service, and calls a solver after each change to restore conflict-freeness. Van Aken et al. [23] address the Train Timetable Adjustment Problem (TTAP): given an existing timetable and a set of planned infrastructure possessions, find an adjusted timetable that minimises cancellations and deviations from the original. Both papers share the delay-minimisation objective that the present work also adopts.

Kloster et al. solve a MIP with lazy constraint generation, operating on a mesoscopic model in which the periodic timetable is projected onto a single operating day; solver state is maintained as constraints are added during lazy generation, but is discarded between successive planning calls. Van Aken et al. extend the PESP formulation with cancellation variables and a row-generation approach for station capacity, operating on a macroscopic network model. Their approach starts with an initial periodic solution and produces a single adjusted timetable that must also be periodic.

2.2 MaxSAT in Timetabling

A growing body of work has established MaxSAT as a competitive solution method for railway timetabling problems where the objective is to find a conflict-free schedule minimising delay from a reference or ideal timetable. Croella et al. [5] apply MaxSAT via the RC2 algorithm to a Dynamic Discretisation Discovery model for real-time train rescheduling. Lemos et al. [13] develop an iterative MaxSAT encoding for scheduling under disruptions. Gouveia et al. [10] address a non-periodic timetabling variant that combines route choice, train cancellations, and delays from an ideal timetable. Matos et al. [18] combine MaxSAT with reinforcement learning to optimise periodic timetables. Fuchs et al. [9] use a binary-search MaxSMT approach to optimize the stability in periodic timetables.

Across this body of work, incremental reuse of solver state takes two forms. Within a single optimisation run, Croella et al. design their variable encoding so that refining the time discretisation only adds new constraints and never invalidates existing ones, which makes the accumulated clause database directly reusable across iterations. Across separate solve invocations for adapted problem instances, only Lemos et al. investigate explicit state retention: they carry the search state into each re-solve triggered by a new disruption, and report an average 25 % speed-up. Unlike Croella et al., however, the search space expansion method they use to handle problem scale arising from discretisation is incompatible with solver state reuse.

2.3 Research Gap

This paper targets the operational layer: given a valid periodic timetable and a sequence of extra train requests, the task is to produce a locally adapted aperiodic schedule that accommodates each request while minimising the delay from the periodic base. The literature reviewed above leaves this problem unaddressed from two complementary directions.

The papers closest to this work at the *problem level* [12, 23] treat each planning call as an independent optimisation: previously adjusted trains are fixed before each new solve, and all solver state is discarded between calls. For Kloster et al., this is an inherent consequence of lazy constraint generation: to handle a request in a different region of the timetable, the solver must restart from scratch to discover the constraints relevant to that region, so no conflict structure accumulated in one call can transfer to the next. Neither paper exploits the incremental structure of the operational planning process at the solver level.

The papers closest to this work at the *methodology level* [5, 13, 10, 18, 9] do not demonstrate the benefits of retaining solver state across a growing sequence of problem additions under an encoding that is complete from the outset. Lemos et al. [13] retain state across re-solves but cannot do so under the search space expansion they require to handle problem scale. The incremental MaxSAT literature establishes that state retention across related instances can yield measurable performance gains [20, 22], but this has not been shown on a practical combinatorial optimisation problem such as operational timetabling.

The present work addresses both gaps simultaneously by encoding the full constraint structure within the MaxSMT solver from the outset, on a microscopic infrastructure model. Because the encoding is complete, learnt cuts accumulate across the entire constraint space with each solve, and successive extra train additions can draw on this full body of accumulated knowledge. We demonstrate empirically that retained solver state outperforms both a warm-start MIP and a cold-start MaxSMT baseline.

3 Problem and Model

We consider the problem of scheduling a set of *extra train requests* against an existing periodic microscopic timetable. A valid periodic timetable for a single operating day is given as input: as supplied, its trains occupy the infrastructure without conflict, and their published arrival times define the reference from which delays are penalised. We refer to these as *context* trains. Context trains are not held to the input schedule: their routes through the microscopic infrastructure may be reassigned, while keeping the sequence of stations served fixed. The event times may move within hard windows around the published arrival and departure times. Departures may only be delayed since passengers plan around them, while arrivals may shift in either direction. Because routes and event times may change, resource occupation must be resolved anew across all trains whenever the timetable is revised, and the conflict-freeness of the input does not carry over.

A set of extra train requests is to be scheduled on top of this context. Each request specifies a sequence of stations together with a desired departure and arrival time at each serviced station; the extra train must be routed through the network so that it serves the requested stations in the given order, without conflicting with context trains or with other extra trains already admitted to the schedule. Its time windows are considerably wider than those of the context trains, since an extra service carries no published time that passengers already rely on. The planning task is to find, for every train, context and extra alike, a route and a set of event times that are jointly feasible, while minimising the aggregate delays from the reference arrival times.

3.1 Event-Activity Model

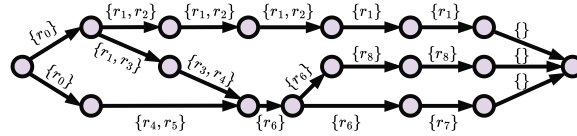
We model the train scheduling problem using the *Event-Activity Network (EAN)* framework; we refer to existing work [8] for the full derivation and give only a brief account here.

The network is a directed graph $G = (\mathcal{E}, \mathcal{A})$ whose components correspond to individual trains, with nodes and edges defined as follows:

Events $e \in \mathcal{E}$ represent scheduled timestamps such as arrivals, departures, or passing times at infrastructure elements. Each event e carries a timing variable $t_e \geq 0$ and hard time bounds $lb_e \leq t_e \leq ub_e$.

Activities $a = (e, e') \in \mathcal{A}$ are directed edges representing precedence relations between pairs of events belonging to a single train, such as running times over track sections or dwell times at stations. Each activity imposes a duration constraint $lb_a \leq t_{e'} - t_e \leq ub_a$ and is associated with a set of resources $R(a) \subseteq \mathcal{R}$. A resource $r \in R(a)$ is occupied during the interval $[t_e, t_{e'} + \delta(r, a)]$, where $\delta(r, a)$ is a release time that enforces the required separation between successive trains. The same resource can appear in the resource sets of multiple activities, whenever those activities must reserve the same physical infrastructure element.

An example EAN for a single train is shown in Figure 1. A feasible solution requires: *route selection*, a single source-to-sink path for each train; *temporal feasibility*, event timings respecting all event and activity bounds along the selected path; and *resource conflict resolution*, no resource occupied by more than one train at any point in time.



■ **Figure 1** The EAN for a single train [8]. Events (nodes) and activities (links) carry time bounds as described in the main text; each activity's resource set $R(a)$ is shown alongside it.

3.2 Encoding with SMT

All constraints in the event-activity model can be reformulated as either clauses over Boolean variables or difference constraints of the form $t_j - t_i \geq l_{ij}$, the latter activated conditionally on the route selection; we refer to [8, 9] for the complete formulation. Event times are integer variables in units of 1 s.

The IDL theory represents the active difference constraints as a directed weighted graph [4]: each timing variable t_i is a node, and each active constraint $t_j - t_i \geq l_{ij}$ adds a directed edge from t_j to t_i with weight l_{ij} . A partial assignment is timing-feasible if and only if this graph contains no positive-length cycle. The theory propagator detects such cycles incrementally as the SAT solver makes Boolean assignments; when a cycle is found, the activation literals of its constituent edges form a *theory lemma*, a clause asserting that this combination of active constraints is infeasible, which is added to the SAT solver's clause database and guides the subsequent search.

4 Algorithm

The central contribution is an incremental solving architecture in which the accumulated MaxSMT solver state is retained and reused across successive extra train additions. As background, we first introduce the core-guided MaxSAT algorithm that underlies the optimisation layer in Section 4.1. Then, Section 4.2 describes the encoding of the delay objective over IDL timing variables; Section 4.3 presents the architecture and the soundness argument for state retention; Section 4.4 defines the six solving strategies compared empirically in Section 6.

4.1 Core-Guided MaxSAT

On top of the SAT+IDL solver we build an optimisation layer using the RC2 algorithm [11], a core-guided MaxSAT solver [19, 17]. MaxSAT partitions constraints into *hard clauses*, which must hold in every solution, and *soft clauses*, each carrying a weight equal to the cost of its violation. The objective is to find a satisfying assignment to the hard clauses that minimises the total weight of violated soft clauses.

RC2 iteratively calls the SAT solver. All soft clauses are initially assumed to hold. If this is infeasible, the SAT solver returns an *UNSAT core*: a subset of soft clauses that cannot simultaneously hold together with the hard clauses. The minimum weight among the clauses in the core is a certificate that the objective must increase by at least that amount, thereby tightening the lower bound. RC2 then encodes the relaxation of the core as a cardinality constraint which asserts that at least one clause in the core may be violated, and permanently adds it to the clause database. Then it probes the solver again at the updated bound. This repeats until the solver returns a feasible solution, which is then provably optimal.

4.2 Timing Objective in MaxSAT

The combination of a MaxSAT solver with an IDL theory lets us encode an objective function directly over the continuous timing variables in the theory, rather than solely over Boolean variables. Thus, though the objective function remains discrete, scheduling itself can be done to the precision-level of the IDL theory. To our knowledge, this combination has not previously been applied in railway scheduling.

The objective minimises the weighted sum of delays from reference times: for context trains, delay from their published arrival times; for extra trains, delay from the preferred arrival time at each serviced station. Both are encoded identically. For each objective event e with timing variable t_e and reference time t_e^* , we introduce n Boolean activation literals $l_1^e, \dots, l_n^e$ together with the following conditionally activated difference constraints:

$$l_k^e \rightarrow t_e \leq t_e^* + k \cdot 60 \text{ s}, \quad k = 1, \dots, n.$$

Each literal l_k^e is added as a soft clause carrying weight 60, so violating it contributes one minute to the objective. The encoding is tightened by adding the implication chain

$$l_k^e \rightarrow l_{k+1}^e, \quad k = 1, \dots, n-1,$$

as hard clauses. The chain is valid because $t_e \leq t_e^* + k \cdot 60 \text{ s}$ implies $t_e \leq t_e^* + (k+1) \cdot 60 \text{ s}$, so setting any literal to true forces all later ones to the same value.

All soft clauses carry the same weight, across events and across the delay minutes of a single event, so the objective is step-wise linear in total delay and indifferent to its distribution; different weights would express other distributional preferences without altering the encoding.

The granularity of 60s per soft clause matches the precision of the published timetable: delays below one minute are not observable to passengers or reflected on public departure boards, so the first minute of delay incurs no penalty. Granularity also controls the encoding size: a finer discretisation would produce more accurate cost distinctions but also lead to a proportionally larger clause database. 60s balances these considerations and matches the objective discretisation used in prior MaxSAT timetabling work [5].

4.3 Incremental Architecture

When a new extra train request is added to an existing feasible schedule, the new solve begins from the full state accumulated during the previous one, rather than discarding it. In the MaxSMT framework, four components make up this retained state:

Conflict clauses. When the SAT solver reaches a contradictory variable assignment, it analyses the chain of decisions that led to the contradiction and derives a *conflict clause*: a constraint that rules out that combination of assignments in any future search.

Theory lemmas. The clauses generated by the IDL propagator when it detects a positive-length cycle in the constraint graph (Section 3.2). They are analogous to conflict clauses, but encode infeasibility of timing combinations rather than propositional contradictions.

Core clauses. When RC2 identifies an UNSAT core among the soft clauses, it encodes the resolution of that core as a cardinality constraint and adds it permanently to the clause database.

Search statistics. The SAT solver maintains an activity score for each variable that reflects how often it appeared in recent conflicts, and uses these scores to decide which variable to branch on next. It also records the last assignment used for each variable. When the solver is not reset, both the activity scores and the saved assignments carry over: the solver begins the new search already biased toward the variables and assignments that were informative in the previous one.

The reason all four components remain valid across solving iterations is that the instance only becomes more constrained between successive solves: each new extra train addition introduces new variables and new constraints, but never removes or relaxes any constraint present in a prior iteration. This monotone strengthening property guarantees soundness. Conflict clauses and theory lemmas are derived by resolution and cycle detection, respectively; both are valid for any instance that includes the constraints used to derive them, and since those constraints are never removed, both remain valid after strengthening. A core witnesses a lower bound on the objective. When the constraints are tightened, the set of satisfying assignments can only shrink, so any previously identified core remains infeasible and its associated lower bound remains valid. Finally, search statistics are heuristic: the solver remains complete regardless of the activity scores and saved variable values it inherits, so carrying them over can only affect efficiency, not correctness.

None of these components is novel in isolation; the contribution lies in their composition within the operational timetabling setting, where the instance changes only locally between successive solves. The claim is that clauses and cores discovered while scheduling earlier extra trains recur when scheduling later ones, and need not be re-derived from scratch.

Here, MIP warm-starting differs. Gurobi retains incumbent solutions between solves but keeps neither the branch-and-bound tree nor any learned cuts, so both the knowledge about infeasibility and the progress toward proving the lower bound are lost on each invocation. The MaxSMT solver retains both, which is the basis for the empirical comparison in Section 6.

■ **Table 1** The four sequential solving strategies, formed by combining the treatment of previously admitted extra trains with the treatment of the solver state between solves. Flexible-persistent is our proposed method; Commit-reset most closely matches the workflow of Kloster et al. [12].

		Solver state between solves	
		Retained	Discarded
Admitted extra trains	Remain free	Flexible-persistent	Flexible-reset
	Committed	Commit-persistent	Commit-reset

4.4 Solving Strategies

Extra train requests may arrive all at once or one at a time, both being operationally realistic. When requests arrive sequentially, there are two independent choices. The first is a problem-level choice: whether an extra train, once scheduled, remains completely free to be rescheduled when later requests arrive (*Flexible*), or is committed and thereafter treated exactly as a context train (*Commit*) keeping route flexibility but with reduced time windows. The second is a solver-level choice: whether the accumulated solver state is retained across solves (*persistent*), as described in Section 4.3, or fully discarded before each solve (*reset*). Combining the two axes gives the four sequential strategies in Table 1.

In addition, we evaluate two *Batch* strategies in which all requests are encoded simultaneously and solved in a single invocation. In the *all-extra-trains* variant, the solver is warm-started from the periodic solution. In the *one-shot* variant, the full periodic timetable is encoded together with all requests and solved from scratch.

The Flexible and Batch strategies yield the same globally optimal objective value: because all extra trains remain free throughout, each solve jointly optimises over the full set of requests, regardless of delivery mode or state-management choice. The Commit strategies trade this guarantee for a smaller search space. Committing each newly scheduled train before the next request arrives removes degrees of freedom and reduces solve time, but forfeits the ability to make joint adjustments when a later request would benefit from revising an earlier decision, potentially leading to a worse objective value.

5 Experimental Setup

All MaxSMT experiments use a custom implementation of RC2 [11] on top of Mini-Card v1.2 [16, 6] extended with IDL theory propagation [14]. The MIP baseline uses Gurobi v13.0.2. The methods are used within the OpenBus toolbox [7]. Both run on a server using 4 cores of an AMD EPYC 7763 CPU allocated 16 GB of RAM. Each experiment is repeated multiple times; run-to-run variation arises from non-deterministic encoding order and server load rather than from solver randomness, as MaxSMT runs deterministically.

The comparison is set up to favour the MIP wherever a choice arises. We use the dedicated formulation of the event-activity model of Fuchs et al. [8], developed for this problem class and competitive on instances of comparable size. Both solvers receive the same problem data, preprocessing, and model strengthenings from that work, and the MIP additionally uses all four cores, whereas MaxSMT is single-threaded. Because Gurobi has been shown to handle discretised objectives in railway scheduling less well than a MaxSAT solver [5], the MIP is also given a continuous relaxation of the discrete objective function: it halts as soon as it finds a feasible solution whose continuous objective value is at most the optimal discretised MaxSMT objective, and therefore never certifies optimality of the discretised problem.

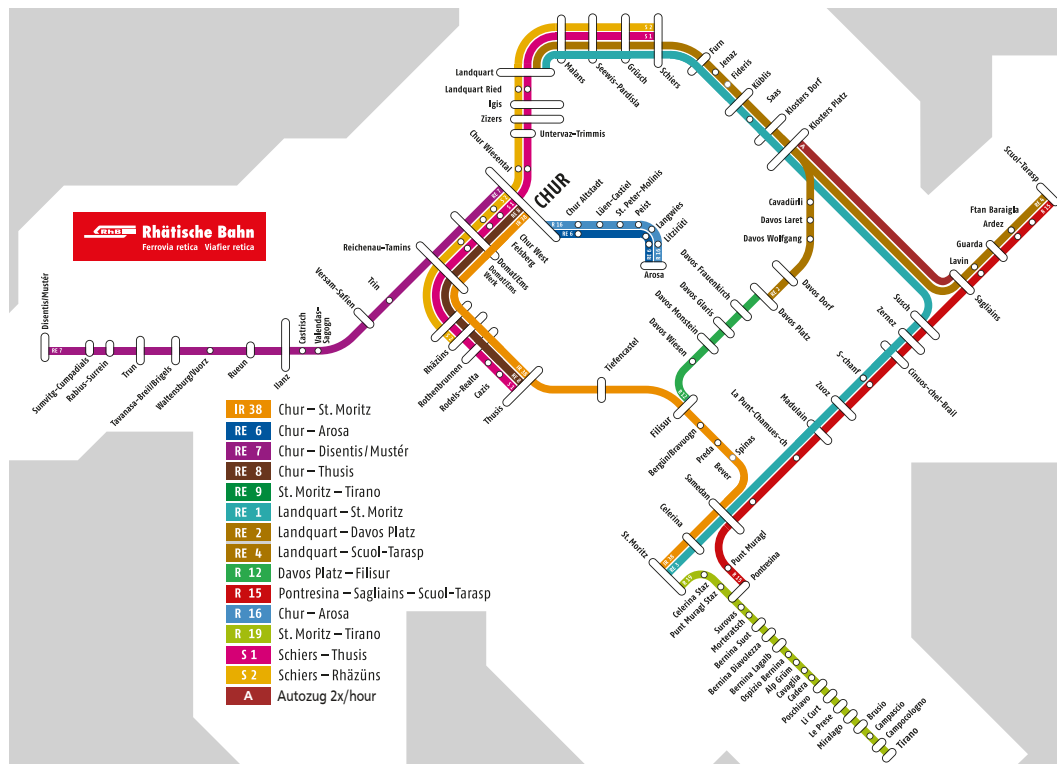


Figure 2 Services in the RhB network. The 14 passenger services run bi-directionally once an hour; the motorail (Autozug) runs twice an hour in each direction. The 240 trains make up the context trains for the experiments.¹

Network and Context Trains. The experiments use infrastructure from the Rhätische Bahn (RhB), a metre-gauge network in the canton of Graubünden, Switzerland, shown in Figure 2. A perfectly periodic solution for the base timetable is provided as input: each train in the timetable is scheduled once an hour in each direction over 8 hours, with the schedule being periodic in both route choice and timings.

The arrival times at stations define the reference from which delays are penalised. Context trains may deviate from their published times within a window of $[0, +5 \text{ min}]$ at station departures and $[-5 \text{ min}, +5 \text{ min}]$ at station arrivals; windows at intermediate events are propagated from these bounds, given minimal travel times [8].

For tractability and interpretability, Sections 6.1–6.3 use only the Chur–Arosa section: the two lines (RE6, R16) visible in the centre of Figure 2, consisting of 8 passing loops connected by single-track corridors. The infrastructure is small enough to visualise the timetable in full, while the single-track structure and two trains per hour per direction produce a high density of resource occupation that prevents trivial scheduling of additional services. Section 6.4 extends to the full RhB network to assess scalability.

Extra Train Requests. Extra train requests are generated by sampling a desired departure time uniformly at random from the 8-hour window and choosing a direction of travel at random. For the Chur–Arosa experiment, all 10 requests are for the same service type: a

¹ Reproduced from: <https://www.rhb.ch/de/streckennetz/>

■ **Table 2** Instance dimensions for the two experimental networks. *Base* is the periodic timetable before any extra train is added; *final* is the instance after all extra train requests have been admitted. Routing decisions count the Boolean route-branching variables; conflicts count pairs of activities competing for the same resource, each introducing an ordering decision.

Network	Trains		Events		Routing dec.		Conflicts	
	Base	Final	Base	Final	Base	Final	Base	Final
Chur–Arosa	32	42	6432	8012	1696	2226	3976	33 255
Full RhB	224	254	110 688	118 613	19 392	21 138	98 035	242 755
<i>Added per extra train request (min–max)</i>								
Chur–Arosa			158		53		837–4339	
Full RhB			125–409		36–91		51–13970	

train with one intermediate stop making three scheduled calls in total. For the full-network experiment, 30 requests are generated across 5 service types, with the service type chosen at random for each request, along with the direction and desired departure time. In both cases, the full set of requests, including their order of addition, is sampled once and held fixed across all runs and all solver configurations, so that differences in solve time reflect solver behaviour rather than instance variability.

Each request specifies a sequence of stations and a desired departure and arrival time at each serviced station, with a time window of $[-3 \text{ min}, +90 \text{ min}]$ around the desired time. The desired arrival times are derived from the sampled departures based on the minimal dwell time at the station. In the Flexible strategies, these windows apply to every admitted extra train throughout, so a train scheduled at an earlier step can still be re-timed anywhere within its original window; in the Commit strategies, each newly scheduled is treated as a context train in subsequent solves, with the context-train windows above applied around the times just scheduled for it. Its reference times in the objective remain those of the original request, so the cumulative objective values reported for the Flexible and Commit strategies measure the same quantity.

Instance Size. Table 2 reports the dimensions of both instances. The full RhB instance contains 254 trains once all requests are admitted, against 160 trains with distinct schedules in Kloster et al. [12] and 135 train lines in the largest instance of Van Aken et al. [23]. Both papers work on mesoscopic and macroscopic models respectively, so the number of events, routing decisions, and resource conflicts per train is considerably smaller than under the microscopic model used here: their largest reported instance comprises 4411 events, where ours has 118 613 events. Though not reported in the two papers, the number of routing decisions and resource conflicts will scale accordingly.

6 Results

We show in Section 6.1 that state retention reduces solve time for both solvers, but not in the same way: MaxSMT benefits consistently across steps, whereas the MIP advantage erodes at the hardest steps, where the warm-start incumbent diverges from optimality. Section 6.2 shows that leaving the time windows of previously scheduled extra trains free between solves yields better cumulative objective values than when reducing them. Section 6.3 shows that MaxSMT reaches these objective values while leaving most of the existing timetable unchanged. Section 6.4 confirms that the conclusions hold on the full RhB network.

■ **Table 3** Median total solve time in seconds and factor relative to the baseline (Flexible-persistent, MaxSMT; 9.3 s), across $n = 50$ runs per condition. *Flexible* keeps all admitted extra trains free, while *Commit* makes each newly scheduled extra train a context train before the next solve; *persistent* retains the solver state between steps, *reset* discards it. *Batch: all-extra-trains* warm-starts from the periodic solution and solves all requests at once. *Batch: one-shot* encodes the full periodic timetable together with all requests and solves once.

Strategy	MaxSMT		MIP (Gurobi)	
	Median (s)	Factor	Median (s)	Factor
Flexible-persistent	9.3	1.0×	559.4	60.2×
Flexible-reset	44.3	4.8×	1742.3	187.3×
Commit-persistent	3.1	0.33×	44.5	4.8×
Commit-reset	5.0	0.54×	36.9	4.0×
Batch: all-extra-trains	14.5	1.6×	577.7	62.1×
Batch: one-shot	14.7	1.6×	1425.8	153.3×

6.1 Solve Time: Persistent vs. Reset

We compare the Flexible-persistent and Flexible-reset strategies for both solvers and report Batch results for comparison with sequential solving. Per-step solve times for the Flexible strategies are shown in Figure 3; aggregate timings for all strategies appear in Table 3.

For MaxSMT, retaining solver state reduces the median total time by a factor of around $5\times$ relative to the Flexible-reset strategy. The benefit is not uniform across steps: early steps, where little state has accumulated, show smaller gains; later steps, where the accumulated conflict structure is richer, show larger ones. This is consistent with the state-retention argument in Section 4.3: the value of retained conflict clauses, theory lemmas, and RC2 core constraints increases as the schedule becomes denser with each addition.

For MIP, the persistent strategy is around $3\times$ faster than resetting the solver state. The MIP therefore also benefits from state retention, though not robustly across all steps. In steps 7 and 9 (the hardest steps as the problem has grown), the benefits of warm-starting the MIP are relatively smaller: the median persistent solve time at step 7 is 29.8 s against 32.4 s for reset, and at step 9, 513.5 s against 570.2 s. These two steps dominate the MIP aggregate solve time and erode the advantage of state retention.

The Batch strategies confirm that the advantage of state retention is not an artefact of sequential delivery. The all-extra-trains batch, which warm-starts from the periodic solution and encodes all requests simultaneously, takes 14.5 s for MaxSMT and 577.7 s for MIP. The Flexible-persistent strategy for MaxSMT is thus measurably faster than Batch, whereas Flexible-persistent MIP and Batch MIP are roughly comparable. Flexible-persistent MaxSMT therefore outperforms both its MIP counterpart and its own Batch baseline.

6.2 Solution Quality: Flexible vs. Commit

Committing previously scheduled extra trains between steps reduces the degrees of freedom in each subsequent solve, and one would therefore expect it to reduce solve time. The per-step solve times for the Commit-persistent and Commit-reset strategies are shown in Figure 4, and the last two rows of Table 3 report the aggregate figures.

The solve-time reduction from committing is substantial for both solvers. Comparing Commit-persistent with Flexible-persistent with MaxSMT, the median total time drops from 9.3 s to 3.1 s, a factor of approximately $3\times$. Comparing the same for the MIP, the reduction

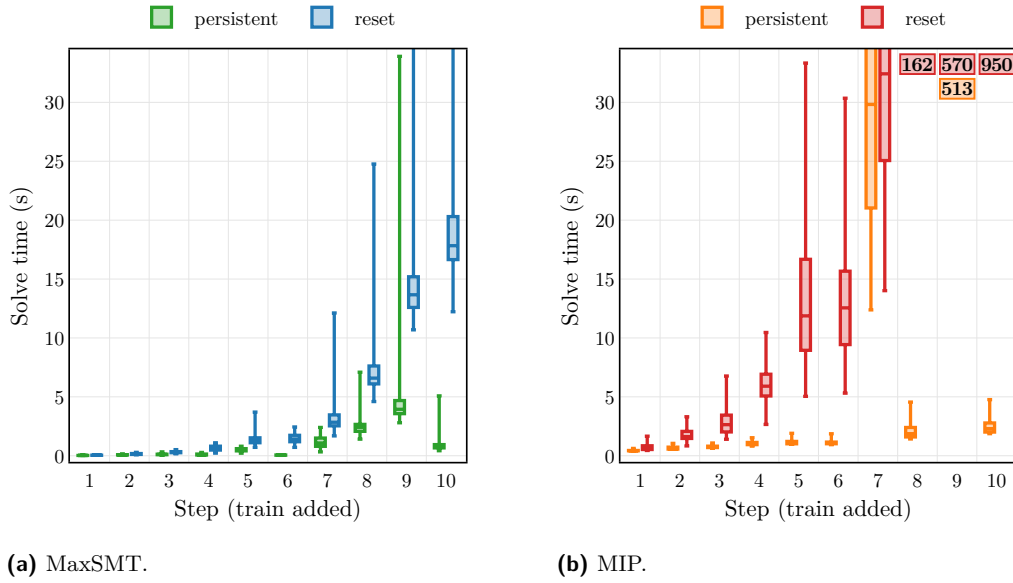


Figure 3 Per-step solve time for the Flexible-persistent and Flexible-reset strategies, for MaxSMT (left) and MIP (right), across $n = 50$ runs. Boxes show the median and interquartile range; whiskers extend to the minimum and maximum. Off-the-chart median values are reported in the boxes at the top of the chart.

Table 4 Cumulative objective value (total delay, in minutes) for the Flexible and Commit strategies. Steps 1–8 are identical; the Commit strategies incur a penalty of 26 min at step 9, which persists to step 10.

Strategy	Step (train added)									
	1	2	3	4	5	6	7	8	9	10
Flexible (min)	1	8	20	26	37	39	53	104	188	219
Commit (min)	1	8	20	26	37	39	53	104	214	245
Gap (min)	0	0	0	0	0	0	0	0	26	26

is larger: from 559.4s to 44.5s, a factor of approximately $12\times$. The MIP Commit-reset strategy (36.9s) is slightly faster than Commit-persistent (44.5s). This reverses the pattern seen in the Flexible case and shows that warm-starting the MIP can become an impediment.

The speed gain from committing comes at a cost to solution quality. Because a committed train cannot be retimed as freely when a new request arrives, the solver loses the ability to make joint adjustments that would reduce the aggregate delay. The effect is visible in Table 4: the cumulative objective values for the Commit and Flexible strategies are identical through step 8, but at step 9 the Commit strategies incur an additional penalty of 26 minutes, which persists through step 10.

6.3 Solution Structure: Locality Bias in MaxSMT

Beyond aggregate solve times and objective values, the two solvers differ in which parts of the timetable they modify when a new extra train request arrives. Figure 5 shows the timetable produced after the seventh step overlaid on the timetable from the sixth step (in grey) for MaxSMT and MIP, side by side, in the Flexible-persistent strategy.

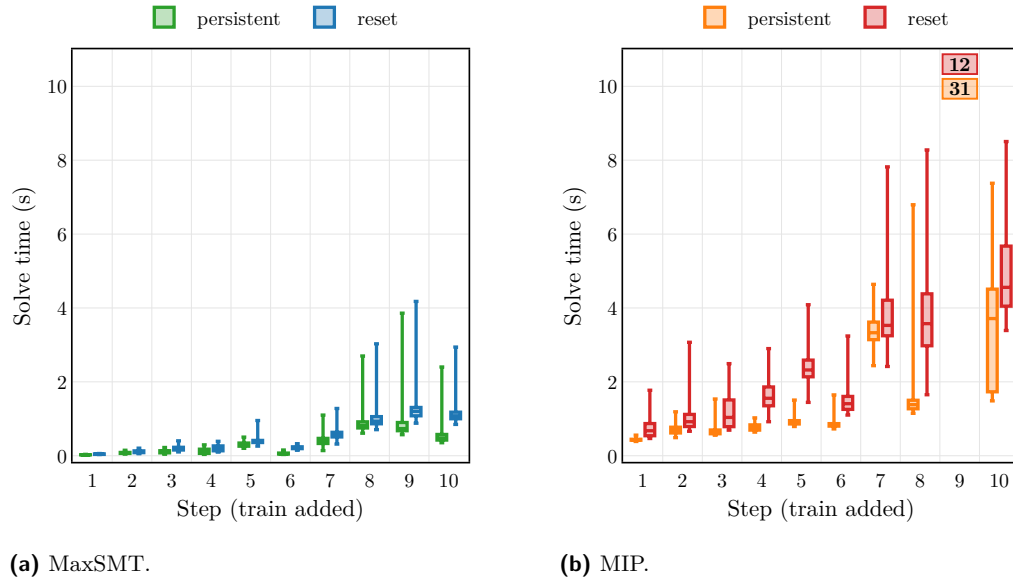


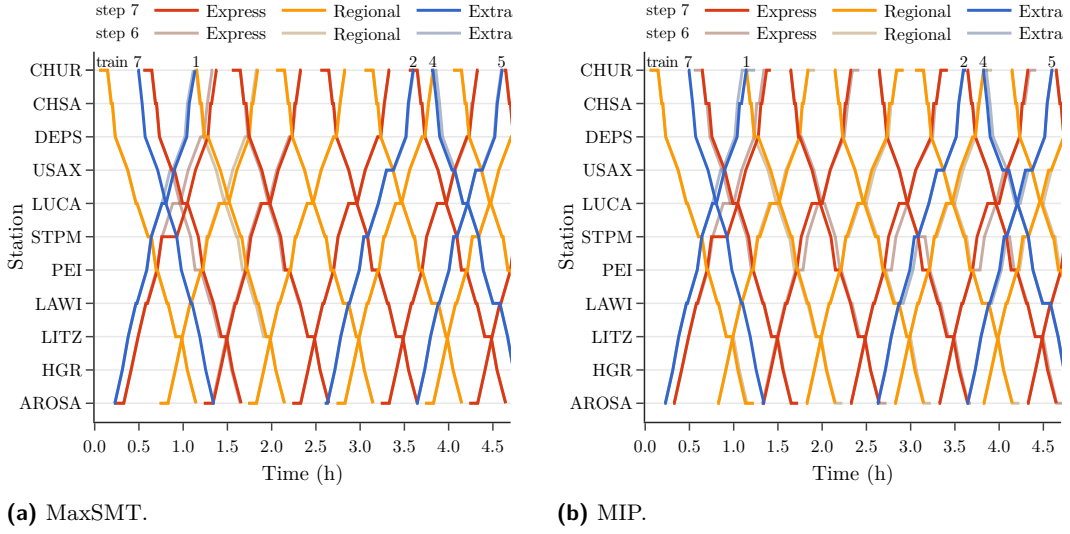
Figure 4 Per-step solve time for the Commit-persistent and Commit-reset strategies, for MaxSMT (left) and MIP (right), across $n = 50$ runs. Boxes show the median and interquartile range; whiskers extend to the minimum and maximum. Off-the-chart median values are reported in the boxes at the top of the chart.

The MaxSMT solution concentrates modifications to the region affected by the new request, with a slight adjustment to another previous extra train. Outside the displayed time window, no further changes are visible. The MIP solution, by contrast, redistributes event times across the full timetable horizon, with changes far from the new train both within and beyond the displayed extract. See the Appendix for the same plot over the full horizon in Figure 7. Across all trains present at both the sixth and seventh step, the MIP changes 230 route branching decisions and shifts 67 station arrival times by more than one minute, while MaxSMT changes 114 route branching decisions and shifts only 12 station arrival times by more than one minute: about half the route modifications and a fifth of the timing changes.

The difference does not reflect a sacrifice of optimality. Both solutions are optimal, and the 60s objective granularity leaves many assignments at the same cost. The MaxSMT solver begins each solve from the variable assignments and activity scores of the previous one, so within this set it tends to return the solution nearest the prior schedule; the MIP solver reconstructs its search tree and has no preference among equally good feasible points. Operationally, the smaller revision is the more useful one: a planner reviewing the updated timetable after a MaxSMT solve sees a change small enough to inspect, and trains running periodically and without conflict keep their periodic timings. Nothing in the formulation enforces this, however: it is a tendency arising from state retention rather than a property of the encoding.

6.4 Scalability: Larger Network

To assess whether the conclusions from the Chur–Arosa experiment hold at larger scale, we repeat the experiment on the full RhB network containing 224 context trains (excludes the RE6 Chur–Arosa express service, as this train only runs occasionally in practice). We schedule 30 extra train requests drawn from the five service types in Table 5, with the



■ **Figure 5** Space–time diagrams for the Chur–Arosa section after the seventh extra train addition (coloured lines) overlaid on the sixth-step solution (grey). MaxSMT (left) concentrates revisions to the region affected by the new request and a slight adjustment to one other extra train; the remainder of the timetable is unchanged. MIP (right) redistributes event times across the full planning horizon. The full-horizon overlay is shown in Figure 7 in the Appendix.

assignment to steps again fixed across all 21 runs. Aggregate solve timings for all strategies appear in Table 6. For interpretability, per-step figures and reported timings refer to the run with the median total solve time across the 21 repetitions in this section.

The patterns observed for Chur–Arosa are preserved and amplified, as shown in Figure 6a. Flexible-persistent MaxSMT remains the fastest strategy, completing all 30 additions in a median of 11.7 min, while Flexible-reset takes nearly 30× as long. State retention thus becomes increasingly valuable as the problem grows. Flexible-persistent also outperforms the Batch strategies by a factor of 2×.

The per-step comparison between Flexible-persistent MaxSMT and MIP is shown in Figure 6b and shows how warm-starting behaves at scale. Recall that the MIP is not required to prove optimality of the discretised problem: it terminates as soon as it finds a feasible solution whose continuous objective value does not exceed the optimal discretised MaxSMT objective (Section 5). On this larger and sparser network, most extra train additions interact weakly with the existing schedule and each other, and the warm-started MIP finds feasible solutions near the warm-start incumbent quickly. Nevertheless, the median total time for Flexible-persistent MIP is 117.7 min, dominated by the few steps where the warm-start incumbent is far from optimality and MIP must fall back on branch-and-bound to find a solution of sufficient quality. Step 24, the longest such step, also stands out in the objective: it is the only step at which the Flexible and Commit strategies diverge, with Commit incurring a penalty of 4 min. Here, the new request conflicts with a previously scheduled extra train and requires joint re-optimisation to resolve. This single step is enough for the Batch strategy to outperform Flexible-persistent MIP in solution time overall. In contrast, Flexible-persistent MaxSMT handles the same step in 67.5 s, within its normal range of 0.1 s to 95.4 s per step; the accumulated solver state already encodes the relevant information from earlier solves, for both feasibility and optimality.

Flexible-reset MIP and Batch one-shot MIP could not be completed within the 10h time limit or exceeded available memory, respectively, and are omitted.

7 Conclusion and Outlook

This paper presented an incremental MaxSMT framework for operational timetabling, scheduling a sequence of extra train requests against an existing periodic timetable while minimising delay from the periodic base. The central contribution is an architecture in which the accumulated solver state is retained across successive extra train additions.

The experimental results on the Rhätische Bahn network support three conclusions.

First, state retention reduces solve time for both solvers, with the benefit growing as the sequence of additions lengthens and the schedule becomes denser. For MIP, the advantage is present but erodes at the hardest steps, where the warm-start incumbent diverges from optimality and branch-and-bound is required; at network scale this erosion is severe enough that solving everything in one batch outperforms incrementally solving overall.

Second, the performance gap between MaxSMT and MIP widens as the sequence of additions grows. The asymmetry is in what each solver keeps: MaxSMT retains its lower bound proof and knowledge about infeasibilities across invocations, whereas MIP discards both on each call.

Third, leaving the time windows of previously scheduled extra trains free between solves yields strictly better cumulative objective values than when reducing them: the Commit strategies incur an objective penalty that the Flexible strategies avoid by exploiting the freedom to jointly revise earlier decisions. The MaxSMT solver achieves this without sacrificing its speed advantage, concentrating revisions to the timetable region of each new request and producing interpretable, minimal changes for a planner to review.

Future work. The incremental architecture as presented relies on the monotone-add-only structure of the problem: solver state is retained because the constraint set never shrinks. This assumption is satisfied by extra train requests but excludes settings where earlier decisions must be retracted, for example, if a request is cancelled after admission or if a track possession modifies the feasibility of previously scheduled services. Recent work on incremental MaxSAT solving [20] provides mechanisms for handling such retractions while preserving accumulated state; extending the present architecture to support constraint relaxation would open the framework to a broader class of operational timetabling scenarios.

The tendency of the MaxSMT solver to revise solutions locally arises from retained search heuristics rather than from the objective. Making it explicit would require a secondary criterion penalising deviation from the base timetable. In a core-guided setting this means a second weight level over a soft-clause set. This would give planners direct control over the stability of successive schedule revisions.

■ **Table 5** Service type for each extra train request in the full-network experiment.

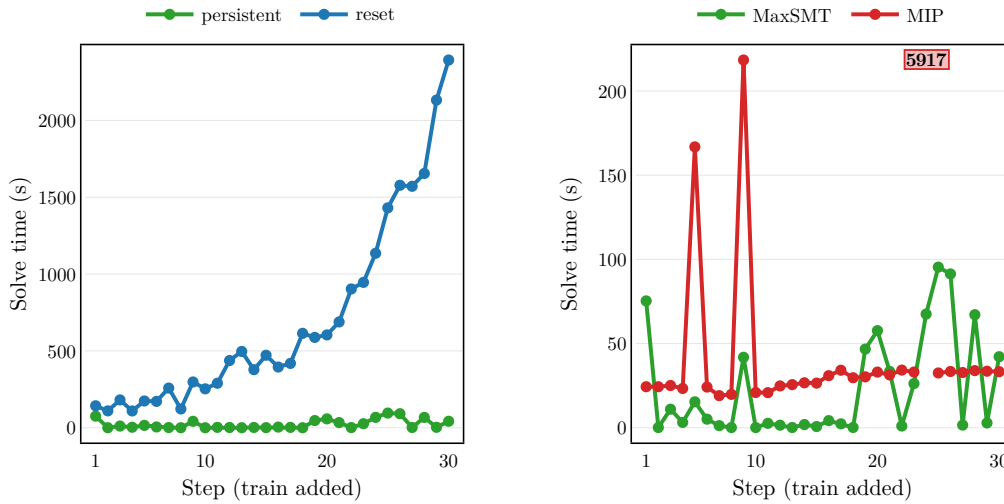
Service type	Steps
Chur – Samedan	3, 4, 7, 15, 16, 18, 19, 20, 26, 28, 30
Samedan – Scuol-Tarasp	2, 5, 6, 9, 10, 12, 14, 21, 22, 25, 27
Landquart – Davos	1, 23, 24
Chur – Illanz	8, 13, 17
Chur – Arosa	11, 29

■ **Table 6** Median total solve time in minutes and factor relative to the baseline (Flexible-persistent, MaxSMT; 11.7 min), across $n = 21$ runs per condition. *Flexible* keeps all admitted extra trains free, while *Commit* makes each newly scheduled extra train a context train before the next solve; *persistent* retains the solver state between steps, *reset* discards it. *Batch: all-extra-trains* warm-starts from the periodic solution and solves all requests at once. *Batch: one-shot* encodes the full periodic timetable together with all requests and solves once.

Strategy	MaxSMT		MIP (Gurobi)	
	Median (min)	Factor	Median (min)	Factor
Flexible-persistent	11.7	1.0×	117.7	10.1×
Flexible-reset	349.1	29.9×	— ^a	—
Commit-persistent	7.2	0.61×	31.6	2.7×
Commit-reset	106.9	9.1×	178.5	15.3×
Batch: all-extra-trains	22.0	1.9×	108.0	9.2×
Batch: one-shot	26.9	2.3×	— ^b	—

^a MIP timed out at 10 h on all runs.

^b MIP exceeded available memory on all runs.



(a) Flexible-persistent versus -reset for MaxSMT. (b) MaxSMT versus MIP for Flexible-persistent.

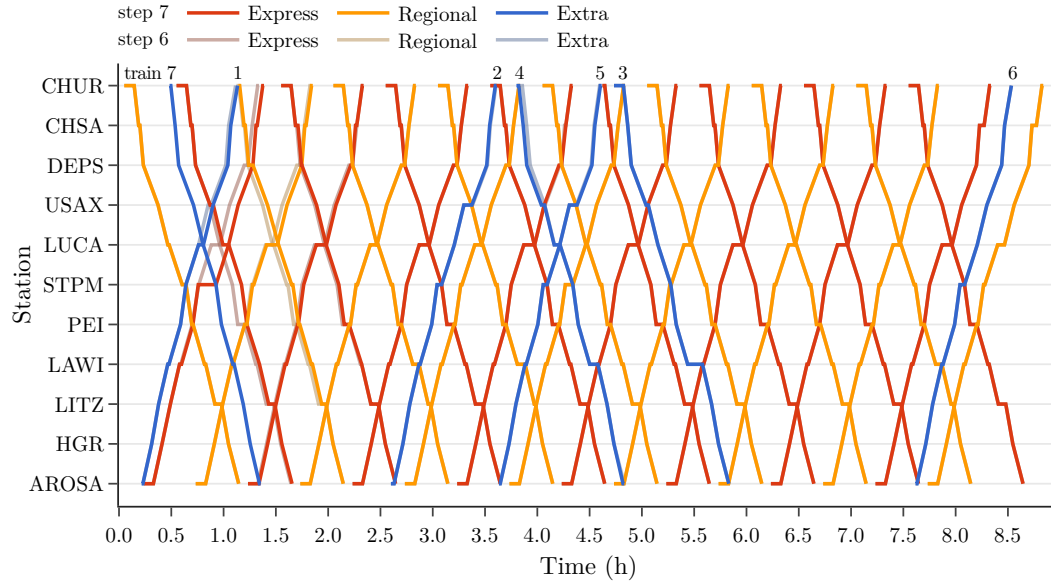
■ **Figure 6** Per-step solve time comparing persistent and reset strategies for MaxSMT (left) and comparing MaxSMT with MIP with the Flexible-persistent strategy (right). Shown are the timings for the run with median total solve time across $n = 21$ runs. Off-the-chart values are reported in the boxes at the top of the chart.

References

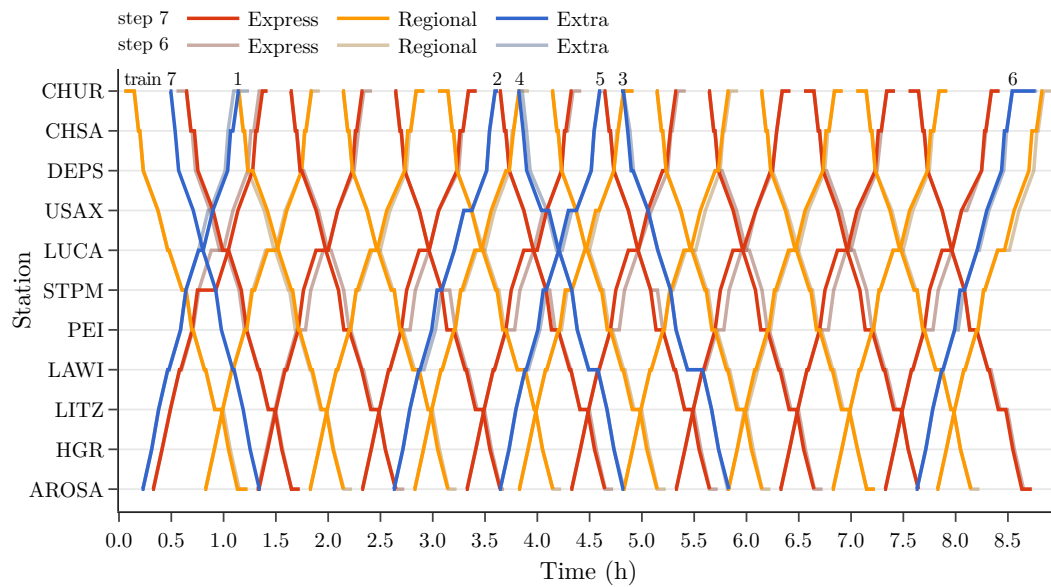
- 1 Gabrio Caimi, Leo Kroon, and Christian Liebchen. Models for railway timetable optimization: Applicability and applications in practice. *Journal of Rail Transport Planning & Management*, 6(4):285–312, January 2017. doi:10.1016/j.jrtpm.2016.11.002.
- 2 SBB Infrastruktur / SBB Cargo. So entsteht der Fahrplan, April 2014. URL: https://blog.sbbcargo.com/wp-content/uploads/2014/11/A2_Plakat_Fahrplan_Infrastruktur_dt_Web.pdf.
- 3 Pedro José Correia Duarte, Stéphane Dauzère-Pérès, Dennis Huisman, Carlo Mannino, Giorgio Sartor, Norman Weik, and Philipp Widmann. 50 years of OR in railway timetabling and rolling stock planning. *EURO Journal on Transportation and Logistics*, 14:100155, January 2025. doi:10.1016/j.ejtl.2025.100155.
- 4 Scott Cotton and Oded Maler. Fast and Flexible Difference Constraint Propagation for DPLL(T). In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006*, pages 170–183, Berlin, Heidelberg, 2006. Springer. doi:10.1007/11814948_19.
- 5 Anna Livia Croella, Bjørnar Luteberget, Carlo Mannino, and Paolo Ventura. A MaxSAT approach for solving a new Dynamic Discretization Discovery model for train rescheduling problems. *Computers & Operations Research*, 167:106679, July 2024. doi:10.1016/j.cor.2024.106679.
- 6 Niklas Eén and Niklas Sörensson. MiniSat Page, September 2013. URL: <http://minisat.se/>.
- 7 Florian Fuchs and Francesco Corman. An Open Toolbox for Integrated Optimization of Public Transport. In *2019 6th International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pages 1–7, June 2019. doi:10.1109/MTITS.2019.8883335.
- 8 Florian Fuchs, Thomas Dubach, Jan Lordieck, Francesco Corman, and Bernardo Martin-Iradi. Handling conflicts in microscopic railway timetabling at scale: A parallelized approach. *Journal of Rail Transport Planning & Management*, 39:100600, September 2026. doi:10.1016/j.jrtpm.2026.100600.
- 9 Florian Fuchs, Bernardo Martin-Iradi, and Francesco Corman. Optimizing periodic stability in railway timetables: A microscopic model for networks with a macroscopic comparison. *Journal of Rail Transport Planning & Management*, 36:100555, December 2025. doi:10.1016/j.jrtpm.2025.100555.
- 10 Filipe Gouveia, Luís Albino, and Ricardo L. Saldanha. A MaxSAT Approach for the Train Timetabling Problem with Route Choice and Other Features. In José Valente de Oliveira, João Leite, João Rodrigues, João Dias, and Pedro Cardoso, editors, *Progress in Artificial Intelligence*, pages 179–191, Cham, 2026. Springer Nature Switzerland. doi:10.1007/978-3-032-05179-0_14.
- 11 Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. RC2: an Efficient MaxSAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):53–64, September 2019. doi:10.3233/SAT190116.
- 12 Oddvar Kloster, Bjørnar Luteberget, Carlo Mannino, and Giorgio Sartor. An Optimization-Based Decision Support Tool for Incremental Train Timetabling. *Operations Research Forum*, 4(3):65, August 2023. doi:10.1007/s43069-023-00243-2.
- 13 Alexandre Lemos, Filipe Gouveia, Pedro T. Monteiro, and Inês Lynce. Iterative Train Scheduling under Disruption with Maximum Satisfiability. *Journal of Artificial Intelligence Research*, 79:1047–1090, March 2024. doi:10.1613/jair.1.14924.
- 14 Florin Leutwiler and Francesco Corman. A logic-based Benders decomposition for microscopic railway timetable planning. *European Journal of Operational Research*, 303(2):525–540, December 2022. doi:10.1016/j.ejor.2022.02.043.
- 15 Christian Liebchen and Rolf H. Möhring. The Modeling Power of the Periodic Event Scheduling Problem: Railway Timetables — and Beyond. In Frank Geraets, Leo Kroon, Anita Schoebel, Dorothea Wagner, and Christos D. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, pages 3–40, Berlin, Heidelberg, 2007. Springer. doi:10.1007/978-3-540-74247-0_1.

- 16 Mark H. Liffiton and Jordyn C. Maglalang. A Cardinality Solver: More Expressive Constraints for Free. In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing — SAT 2012*, pages 485–486, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-31612-8_47.
- 17 Ruben Martins, Saurabh Joshi, Vasco Manquinho, and Inês Lynce. Incremental Cardinality Constraints for MaxSAT. In Barry O’Sullivan, editor, *Principles and Practice of Constraint Programming*, pages 531–548, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-10428-7_39.
- 18 Gonalo P. Matos, Lu s M. Albino, Ricardo L. Saldanha, and Ernesto M. Morgado. Solving periodic timetabling problems with SAT and machine learning. *Public Transport*, 13(3):625–648, October 2021. doi:10.1007/s12469-020-00244-y.
- 19 Antonio Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-Guided MaxSAT with Soft Cardinality Constraints. In Barry O’Sullivan, editor, *Principles and Practice of Constraint Programming*, pages 564–573, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-10428-7_41.
- 20 Andreas Niskanen, Jeremias Berg, and Matti J rvisalo. Incremental Maximum Satisfiability. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum f r Informatik. doi:10.4230/LIPIcs.SAT.2022.14.
- 21 P. Serafini and W. Ukovich. A Periodic Job Shop Model. *IFAC Proceedings Volumes*, 22(14):89–94, September 1989. doi:10.1016/S1474-6670(17)54331-1.
- 22 Xujie Si, Xin Zhang, Vasco Manquinho, Mikol   Janota, Alexey Ignatiev, and Mayur Naik. On Incremental Core-Guided MaxSAT Solving. In Michel Rueher, editor, *Principles and Practice of Constraint Programming*, volume 9892, pages 473–482. Springer International Publishing, Cham, 2016. Series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-319-44953-1_30.
- 23 Sander Van Aken, Nikola Be sinovi , and Rob M. P. Goverde. Designing alternative railway timetables under infrastructure maintenance possessions. *Transportation Research Part B: Methodological*, 98:224–238, April 2017. doi:10.1016/j.trb.2016.12.019.

A Full Space-Time diagram



(a) MaxSMT.



(b) MIP.

Figure 7 Space-time diagrams for Chur–Arosa case study after the seventh extra train addition (coloured lines) overlaid on the sixth-step solution (grey). MaxSMT (top) revises only the region directly affected by the new request and an adjustment to one other extra train; the remainder of the timetable is unchanged. MIP (bottom) redistributes event times across the full planning horizon.