


# Accelerating logic-based Benders decomposition for railway rescheduling by exploiting similarities in delays

**Journal Article****Author(s):**

Leutwiler, Florin; Bonet Filella, Guillem; [Corman, Francesco](#) 

**Publication date:**

2023-02

**Permanent link:**

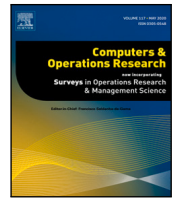
<https://doi.org/10.3929/ethz-b-000580901>

**Rights / license:**

[Creative Commons Attribution 4.0 International](#)

**Originally published in:**

Computers & Operations Research 150, <https://doi.org/10.1016/j.cor.2022.106075>



# Accelerating logic-based Benders decomposition for railway rescheduling by exploiting similarities in delays

Florin Leutwiler, Guillem Bonet Filella, Francesco Corman\*

*Institute for Transport Planning and Systems, ETH Zürich, 8092 Zürich, Switzerland*

## ARTICLE INFO

### Keywords:

Rescheduling  
Benders decomposition  
Precomputation  
Similarity

## ABSTRACT

The operation of a railway system is subject to unpredictable delays or disruptions. Operators control the railway system to minimize losses in performance. Real-time rescheduling is the adaptation of a railway schedule to any unforeseen delay or disturbance and recovers an optimal system state. In this work we propose the extension of an existing Benders decomposition scheme used so far for timetabling, to the case of railway rescheduling. We show how to increase its computational speed by a factor 2, by considering libraries of Benders cuts computed for other instances, to be reused in the solution. We show how including extra cuts has to balance a speedup potential, with a general slowdown due to optimization problems of increased sizes. We show that, if delays in an instance of rescheduling are in fact unknown, but come from a known statistical distribution, we can use a similarity measure to identify a-priori the most promising libraries of Benders cuts, which lead to speedups up to 20%.

## 1. Introduction

In the operation of a railway, unpredictable events and delays are unavoidable and continuously perturb the system. Railway systems are operated based on a timetable, that is carefully designed by solving a timetabling problem, and aims to maximize stability and performance of the railway system during the operation. Railway operators continuously monitor the railway system and identify deviations between plan and reality. In fact, in case an unforeseen event or delay perturbs the railway system, the initially designed timetable is often no longer a suitable plan of operation; degradation of performance, such as delay propagation, and, in worst cases, cancellations and short turning are the result.

The adaptation of the offline planned timetable to an ever-changing situation is termed rescheduling. Despite much rescheduling is still done by hand, automated tools have been developed. Such tools require the definition of a measure of system performance (related to delays), and suitable mathematical modeling of the constraints of the railway system. Typical actions to be considered in rescheduling include retiming, reordering and rerouting of trains over the infrastructure. The most advanced optimization algorithms can compute an optimal or near-optimal adaptation of the original timetable, considering the current perturbed state of the railway system. While these approaches have proven their academic value in an increasing literature, real-life applications of such sort are in general limited (Borndörfer et al., 2017). The problems of timetabling and rescheduling (we refer to both of

them as railway scheduling) are similar, and are in fact both NP-hard problems (e.g., Mascis and Pacciarelli (2002)). At the same time the problem of rescheduling is faced during the real-time operation of the railway system, where the available time for computations is strongly limited. As a consequence, the size of possible optimization problems for rescheduling is bounded to local areas with rather limited size.

In this work, we propose a technique to find faster the solution to a rescheduling problem, by exploiting similarities of delay in instances of rescheduling, and precomputation.

We use a logic Benders decomposition for scheduling, that has been introduced in Leutwiler and Corman (2022) and propose a way to improve it for rescheduling. We exploit the fact that in a daily repeating timetable, rescheduling problems for the same area and the same time of the day are all variations derived from the same timetable, and vary only by their input delays, i.e., the real-time delay of trains. Moreover, input delays are in general similar, i.e., they can be statistically characterized and described by a sufficiently large amount of samples. From the academic literature, we see that knowledge of the statistical distribution of delay, and similarity has not been extensively used.

We propose to precompute features on a set of delay instances, and use insight generated in this way during real-time use, to an unknown delay. We specifically use logic Benders cuts, generated in the solution process of Benders decomposition, as output of the precomputation, which can be included (reused) in the solution process of another

\* Corresponding author.

E-mail address: [francesco.corman@ivt.baug.ethz.ch](mailto:francesco.corman@ivt.baug.ethz.ch) (F. Corman).

instance. We assume that some logic Benders cuts that have been generated while solving one instance of rescheduling with a particular situation of input delay, are useful for the computation of a solution to a different instance, where input delays are different but similar to each other. The reused cuts decrease the amount of iterations and the amount of constraints in the master problem of a Benders decomposition; both reduce the total time spent solving the master problem, which is a substantial part of the computational burden in a Benders decomposition, resulting in an overall computational speedup. Logic Benders cuts for the reuse can be precomputed ahead of time and saved into libraries, such that we can reduce the amount of computation necessary at the moment of operation, when time is crucial. Two measures of similarity, applied to the situation of delay, are used to identify which libraries and cuts are actually useful for the reuse. We show that using precomputed cuts from the most similar instance leads to a speedup up to ~20%. In general, the potential of reuse is even larger, up to a factor of 2.5, compared to a decomposition scheme with no reuse of cuts; and even larger, when compared to a centralized solution by a commercial solver.

This paper is structured as following. We review related literature in Section 2 to highlight the contributions of this work. In Section 3, we introduce the railway rescheduling problem addressed. We extend a disjunctive formulation for our rescheduling problem, similar to Leutwiler and Corman (2022), and apply the logic-based Benders decomposition of Leutwiler and Corman (2022) in Section 4. We describe the details of the proposed approach in Section 5. Numerous experiments are provided on real-world examples in Section 7. We conclude in Section 8.

## 2. Related work

### 2.1. Railway rescheduling

The literature on railway rescheduling shows a variety of scientific publications. Comprehensive overviews can be found in reviews such as Cacchiani et al. (2014) or Corman and Meng (2014).

From a high-level perspective we can categorize the literature in three aspects. One is granularity. In coarse granular models, i.e., macroscopic models, the network is abstracted into nodes and lines representing stations and connections between the stations respectively (e.g. Veelenturf et al. (2016) and Dollevoet et al. (2017)). Instead, fine granular models, i.e., microscopic models, consider the network at the level of detail of a safety system (e.g., Corman et al. (2014), Pellegrini et al. (2015) and Samà et al. (2017)). In those models, conflict free movements of trains over the network as well as routing of trains can be explicitly modeled. We further focus for our work on this second stream of works.

In the aspect of deviation from the planned operations, and information available in real-time about it, we can differentiate the literature in publications working on disturbances and those considering disruptions. Disturbances are usually considered as delays of trains in the magnitude of few minutes (e.g., Corman et al. (2012)). Disturbances are common, and their empirical statistics can be collected and described by probability distributions. Disruptions are usually considered as events with more severe effects on the network, e.g., delays above 30 min (Corman et al., 2012) or the unavailability of parts of the network for several hours (Cacchiani et al., 2012). Disruptions are typically rare and large; and often dealt with as single cases, as scenarios (where the probability of occurrence cannot be estimated well).

In the aspect of solution approaches, the literature shows a comprehensive variety of methods. The variety ranges from Branch&Bound approaches (e.g., D'Ariano et al. (2007)), over Integer (e.g., Caimi et al. (2012)) and Mixed-Integer Programming (e.g., Pellegrini et al. (2015)) solved by commercial solvers, to a large variety of heuristics (e.g., Corman et al. (2014)) or collaborative solutions (e.g., D'Ariano and Hemelrijk (2006)), to name a few.

### 2.2. Decomposition in rescheduling

As we use a decomposed approach, we quickly review a variety of decomposition approaches that have been proposed for railway scheduling (either timetabling or rescheduling). In general, we can classify these decompositions into hierarchical and decentralized structures.

In hierarchical decompositions (e.g., Lamorgese et al. (2016), Lamorgese and Mannino (2019), Keita et al. (2020), Corman et al. (2014), Leutwiler and Corman (2022), Luan et al. (2020) and Cacchiani et al. (2008)), the original scheduling problem is separated into multiple partial optimization problems distributed over several hierarchical layers. In a hierarchical structure, solutions for the optimization on each layer are passed downwards on the hierarchical structure by additional constraints, penalties or by fixing some of the variables of the subordinate optimization problems. This coordination over the different layers happens top-down. Coordination in the opposite direction is usually achieved by means of additional constraints, penalties or heuristics (determining how to proceed, in case of conflicting partial solutions from the subordinate layers).

In decentralized decompositions (e.g., Perrachon et al. (2020), Liu et al. (2019) and Bretas et al. (2019)), the original scheduling problem is separated into multiple partial and hierarchically equal optimization problems. The partial optimization problems are solved independently, and an iterative coordination steers them towards a global feasible solution.

### 2.3. Precomputation and data-driven approaches

A few approaches proposed the precomputation in rescheduling, i.e., conduct computations in advance, well before the day of operation, save relevant result actions indexed in some feature space, and are very quick to apply the action for a situation with the same features. In Caimi et al. (2012), speed profiles for trains are precomputed, which are then, during the real-time application, selected and assembled to a schedule considering the current delays of the railway. In Van Thielen et al. (2018), a dynamic impact zone is created for delay situations in which rescheduling actions are necessary. The dynamic impact zone reduces the size of the rescheduling problem, increasing the computational speed of rescheduling. The dynamic impact zone is computed in real-time, based on precomputed scenarios of possible delay propagation and emerging resource conflicts. In Ghasempour and Heydecker (2020), precomputation is done within a machine learning approach, where an algorithm of approximate dynamic programming uses learned costs of different decision. The approach is shown able to control traffic at a single junction.

### 2.4. Contributions

We consider railway rescheduling on a microscopic model, to handle small disturbances on a large and heavily utilized railway network. We formulate the rescheduling problem as a disjunctive program (Balas, 1998), similar to Leutwiler and Corman (2022), encompassing the decisions of retiming, reordering and also rerouting. We extend the hierarchical logic-Benders decomposition of Leutwiler and Corman (2022) to this case. The contributions of this work are:

(1) We show that the reuse of logic Benders cuts is valid, when dealing with rescheduling instances that differ only by their input delay. We propose a modification for precomputed logic Benders cuts, which allows for a broader reuse of precomputed logic Benders cuts. Benders cuts for the reuse can be precomputed offline and stored in a library of cuts.

(2) We propose a lazy-constraint approach for dynamically reusing logic Benders cuts, including cuts in the master problem only in case they are violated by the incumbent master solution. In this way, we can consider larger libraries of precomputed cuts, without an unnecessary

slow down of the solution process (which would happen if too many cuts are directly included in the master problem). We show how the perfect reuse leads to solving an instance twice as fast as no-reuse.

(3) We propose similarity measures to estimate a-priori, based on the input delays, the worth of a logic Benders cut in terms of expected computational speedup when reused. In this way we can reuse only the cuts that are most promising, avoiding to extend a problem by too many cuts. Overall, this achieves a significant speedup of the entire solution process; if a sufficiently similar instance exists in the training set, a 20% faster computation is achieved on average.

### 3. Microscopic railway rescheduling

We consider the problem of railway rescheduling on a microscopic representation of the railway infrastructure. This computes an adaptation of an existing (offline) timetable, which considers the real-time system state (i.e., actual delays of trains). The solution is a new schedule, which obeys the safety regulations of the railway, kinematics of trains and the railway infrastructure. A solution must be available within seconds or few minutes to enable real-time control of the railway system.

This problem has been addressed by many others, and our description of the problem is rather standard in this sense (e.g., D’Ariano et al. (2007) and Pellegrini et al. (2015)). We optimize over times, routes and orders, to minimize total delays of trains, when some input delays arise in the network. We represent times by continuous variables, and routing and ordering decisions as discrete decision. As in Leutwiler and Corman (2022), we generalize decisions to non-binary sets, and to larger sets, to properly model rerouting actions. Formally, the *discrete decision* is to select one *choice*, out of a finite set of possible choices, specific to the decision. Each choice results in a specific (set of) constraints to be satisfied by the solution.

We model the railway network at the level of blocks, i.e., sections of several hundred meters of tracks. A single operation is the passing of a train over a block, which is associated to an entry (*start*) event, and an exit (*end*) event. An instance is described by a list of trains, their planned timetable, the infrastructure, and some input delays.

The planned timetable results in temporal specifications on events in the problem that are identified as relevant by the operators of the railway (like arrivals/departures at/from stations). Such specifications prescribe a latest time for arrivals, an earliest time for departures, and possibly a minimal/maximal duration between two relevant events for passenger transfers. For an arrival of train after the latest time, there is an *arrival delay*.

The description of the infrastructure defines the dedicated infrastructure for the operation of trains, individually for each train. Dynamics of trains are abstracted into minimal traveling times for each individual block and train. The route of a train is the consecutive sequence of blocks from the train origin to its destination. In between origin and destination, the route may contain routing areas, where different alternative sequences of blocks are available to travel over the network. Each alternative sequence in a routing area provides a connection between the same starting and ending block. For each routing area of a train, a routing decision must be made to select one alternative sequence, that is used by the train in the final schedule. Routing decisions for all routing areas must be made to conclude with a final schedule.

Shared infrastructure gives rise to resource conflicts, i.e., the potential concurrent use of the same block by two trains. For each resource conflict amongst a pair of trains, caused by the respective temporal and infrastructural limitations, an ordering decision has to be made. For a final schedule, an order has to be decided for each resource conflict, i.e., each pair of trains in conflict.

We consider the real-time delay of trains as *input delay* in our problem. In the literature, delays are most commonly considered only on the entrance of a train into the network (e.g., Pellegrini et al. (2015)

and D’Ariano et al. (2007)). In our approach, we do not limit the occurrence of input delay to the first event of the train, i.e., the entrance into the network, but consider the possibility of an input delay at any relevant event of the train, which corresponds to a departure from a station. The input delay on a relevant event is a variation to the earliest departure time given by a temporal specification.

The solution of rescheduling is a schedule which, by suitable update of times, orders and routes for all running trains, minimizes the sum of all arrival delays over all relevant arrival events of the problem, given some statistically characterized input delays. We ignore early arrivals in this work.

### 4. A decomposed disjunctive formulation of rescheduling

#### 4.1. A disjunctive formulation of rescheduling

We formulate the microscopic railway rescheduling problem (MRR) as a disjunctive optimization problem (Balas, 1998), likewise to the disjunctive formulation of railway timetabling in Leutwiler and Corman (2022). A train  $d$  represents a sequence of operations  $(d, b)$ , where  $b$  indicates the related block. We represent the start time of operation  $(d, b)$  by the continuous variable  $t_{db} \in \mathbb{R}_+$ . Time variables for the end of an operation are redundant as jobs cannot be paused or interrupted in MRR.

We use precedence relations to model the constraints of MRR. A precedence relation  $((d, b), (q, p))$  is a linear inequality, constraining  $t_{qp}$  to occur at least  $f_{db,qp}$  time units after  $t_{db}$ . Similar to Leutwiler and Corman (2022) we can characterize precedence relations as *fixed* or *selectable*. The former are standard conjunctive constraints, which must be satisfied in any case by a solution of MRR; the latter model choices of reordering or rerouting.

We model minimal travel times and minimal/maximal durations between relevant events by precedence relations in a set  $A_f$ , which are fixed.

With a time origin  $t_0 = 0$  we model absolute timing constraints as precedence relations. A set  $A_h$  contains (fixed) precedence relations representing earliest departure times  $\tau_{b,i}$  at relevant events, with respect to the origin of times  $t_0$ , updated by (possibly null) input delay  $\mu_i$ . In the set  $A_h$  we consider exactly one precedence relation for each relevant event that is a departure.

A set  $A_\delta$  contains (fixed) precedence relations, which model the latest arrival time  $\tau_{ub,i}$  at relevant events. We use those constraints to determine the arrival delay  $\delta_{db}$  of a relevant event  $(d, b)$ .

For the discrete decisions of MRR we use the modeling technique of Leutwiler and Corman (2022). The set  $A_s$  contains (selectable) precedence relations that are constraints on events, which must hold upon the choices for the discrete decisions. Selectable precedence relations  $A_s$  are grouped into choice sets  $W_c$ , where precedence relations of the same choice set can only be jointly selected. Choice sets represent the choices of a discrete decision. The choice sets again are grouped into a decision set  $D_l$ . The set  $D_l$  is the set of all choice sets related to the same decision  $l$ . Given  $D_l$  we can model a discrete decision  $l$  by the following disjunctive constraint,

$$\bigvee_{W_c \in D_l} \bigwedge_{((d,b),(q,p)) \in W_c} (t_{qp} - t_{db} \geq f_{db,qp}). \quad (1)$$

To model the fact that ordering decisions may depend on routing decisions, we use the technique of auxiliary variables as in Leutwiler and Corman (2022). The auxiliary variables replace the original variables in the precedence constraints of an ordering decision, which prescribe either order. Auxiliary variables are only constrained to be equal to the corresponding original variables, if the associated routing is chosen.

For simplicity we reduce the notion of an operation  $(d, b)$  of train  $d$  on block  $b$ , to  $i$  in the remainder of this paper. With  $L$  as the set of

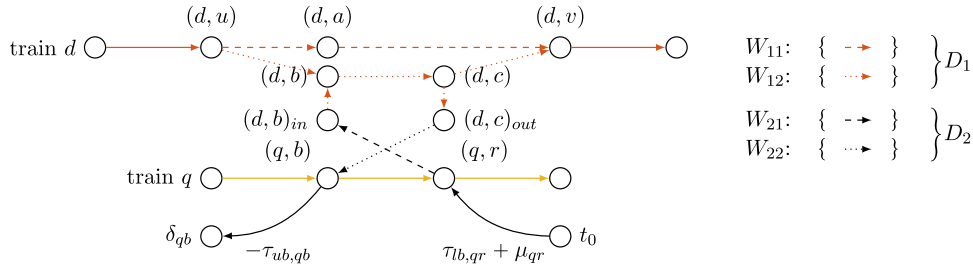


Fig. 1. Example of the generalized disjunctive graph.

all decisions in an instance of MRR, we can formulate the MRR as the following disjunctive program,

$$\begin{aligned}
 \min \quad & \sum_{(i,0) \in A_\delta} \delta_i \\
 \text{s.t.} \quad & t_j - t_i \geq f_{ij} & (i,j) \in A_f \\
 & t_i - t_0 \geq \tau_{lb,i} + \mu_i & (0,i) \in A_h \\
 & t_0 - t_i \geq -\tau_{ub,i} - \delta_i & (i,0) \in A_\delta \\
 & \bigvee_{W_c \in D_l} \bigwedge_{(i,j) \in W_c} (t_j - t_i \geq f_{ij}) & l \in L \\
 & t_i \in \mathbb{R}_+ \forall t_i, \quad \delta_i \in \mathbb{R}_+ \forall \delta_i
 \end{aligned} \tag{2}$$

where the objective is to minimize the sum of arrival delays  $\delta_i$  at all arrival events in the problem. We can further simplify the constraints  $A_\delta$  to  $\delta_i - t_i \geq -\tau_{ub,i}$ , using  $t_0 = 0$ .

Constraints of Problem (2) can be represented in the generalized disjunctive graph of Leutwiler and Corman (2022). We will later use the generalized disjunctive graph in Section 5 to modify precomputed logic Benders cuts for the reuse. The generalized disjunctive graph for Problem (2) is defined by the tuple  $G = (V, A_f \cup A_h \cup A_\delta, A_s)$ .  $V$  is the set of nodes, where each node represents a variable  $t_i$  or  $\delta_i$  of Problem (2).  $A_f, A_h$  and  $A_\delta$  are fixed arcs in the graph with length  $f_{ij}, \tau_{lb,i} + \mu_i$  and  $-\tau_{ub,i}$  respectively, representing the fixed precedence relations of Problem (2).  $A_s$  are selectable arcs with length  $f_{ij}$  representing the selectable precedence relations. Selectable arcs are grouped into the choice sets of Problem (2) and such choices sets are grouped into the decision sets of Problem (2). A selection  $\theta \subseteq A_s$  on  $G$  is a set of selectable arcs such that  $G(\theta) = (V, A_f \cup A_h \cup A_\delta \cup \theta)$  is a standard directed graph. The selection  $\theta$  is complete if it contains for each decision  $l \in L$  at least one choice set, i.e.,  $\forall l \in L, \exists W_c \in D_l$  s.t.  $W_c \subseteq \theta$ ; else the selection is partial. For simplicity we further denote the generalized disjunctive graph simply as the disjunctive graph.

In Fig. 1, we give an illustrative example of the generalized disjunctive graph (similar to example in Leutwiler and Corman (2022)) for a scenario of two trains  $d$  and  $q$ , with train  $d$  passing a routing area with two routing alternatives; one of those alternatives leads to a resource conflict with train  $q$ . Nodes  $(d,b)_{in}$  and  $(d,c)_{out}$  represent auxiliary variables. We consider in the example  $(q,b)$  an arrival event with a latest arrival time  $\tau_{ub,qb}$  and  $(q,r)$  a departure event with an earliest departure  $\tau_{lb,qr}$  and an input delay of  $\mu_{qr}$

#### 4.2. A decomposition of rescheduling

We apply to Problem (2) the Benders decomposition of Leutwiler and Corman (2022) to decompose the centralized problem  $C$ , i.e., Problem (2), into a master problem  $\mathcal{M}$  and a subproblem  $S$ . Benders decomposition works iteratively, where at each iteration the master problem is first solved, and its solution is imposed to the subproblem. If the subproblem cannot find a feasible solution given the master solution, a proof of infeasibility can be generated. In this case, Benders feasibility cuts are generated from the proof of infeasibility and sent back to be included in the successive iterations of the master. The procedure continues until a feasible solution for the subproblem is found.

The problem presented in this paper has some differences to Leutwiler and Corman (2022) in the way it models delays. To be coherent with the decomposition of Leutwiler and Corman (2022), the subproblem must be a problem of feasibility only. We partition the precedence relations of Problem (2) such that all  $A_\delta$  and therefore all variables  $\delta_i$  are considered as constraints and variables of the master problem, and the subproblem is a problem of feasibility only. According to Leutwiler and Corman (2022), we decompose Problem (2) by partitioning  $A_f, A_h, A_\delta$  and  $L$  into  $A_{M,f}, A_{M,h}, A_\delta$  and  $L_M$  for the master; and  $A_{S,f}, A_{S,h}$  and  $L_S$  for the subproblem. In this separation, we restate that  $A_\delta$  is only present in the master. Consequently,  $A_M := A_{M,f} \cup A_{M,h} \cup A_\delta \cup (\bigcup_{L_M} \bigcup_{W_c} (i,j))$  and  $M := \{t_i, t_j \mid (i,j) \in A_M\} \cup \{\delta_i \mid (i,0) \in A_\delta\}$  are the precedence relations and variables of the master; and  $A_S := A_{S,f} \cup A_{S,h} \cup (\bigcup_{L_S} \bigcup_{W_c} (i,j))$  and  $S := \{t_i, t_j \mid (i,j) \in A_S\}$  are the precedence relations and variables of the subproblem. With a set of Benders cuts  $B_R$ , which we additionally consider (reuse) in the master problem, we define the complete master problem of our Benders decomposition for Problem (2) at the iteration  $\alpha$  of the Benders decomposition scheme, i.e.,  $\mathcal{M}^\alpha(B_R)$ , as the following disjunctive problem

$$\begin{aligned}
 \min \quad & \sum_{(i,0) \in A_\delta} \delta_i \\
 \text{s.t.} \quad & t_j - t_i \geq f_{ij} & (i,j) \in A_{M,f} \\
 & t_i - t_0 \geq \tau_{lb,i} + \mu_i & (0,i) \in A_{M,h} \\
 & \delta_i - t_i \geq -\tau_{ub,i} & (i,0) \in A_\delta \\
 & \bigvee_{W_c \in D_l} \bigwedge_{(i,j) \in W_c} (t_j - t_i \geq f_{ij}) & l \in L_M \\
 & \beta^r & r \in B^\alpha \cup B_R \\
 & t_i \in \mathbb{R}_+ \forall t_i \in M, \quad \delta_i \in \mathbb{R}_+ \forall \delta_i \in M,
 \end{aligned} \tag{3}$$

where  $\beta^r$  has the form of the logic Benders cut as introduced in Leutwiler and Corman (2022),  $B^\alpha$  is the set of all Benders cuts iteratively generated from analyzing the subproblem until iteration  $\alpha$ . In general, we will select  $B_R$  by a suitable procedure, from a larger set (a library) of Benders cuts  $\mathcal{B}$ . Once a Benders cut is included in the problem, i.e., either in  $B^\alpha$  or  $B_R$ , it remains considered in the problem until the end.

We define the subproblem  $S^\alpha$  at iteration  $\alpha$  of the decomposition scheme, given  $\tilde{t}_i^\alpha$  of master solution  $\mathcal{O}^\alpha := \{\tilde{t}_i^\alpha, i \in M\}$ , to be,

$$\begin{aligned}
 \min \quad & 0 \\
 \text{s.t.} \quad & t_i = \tilde{t}_i^\alpha & \forall i \in M_S \\
 & t_j - t_i \geq f_{ij} & (i,j) \in A_{S,f} \\
 & t_i - t_0 \geq \tau_{lb,i} + \mu_i & (0,i) \in A_{S,h} \\
 & \bigvee_{W_c \in D_l} \bigwedge_{(i,j) \in W_c} (t_j - t_i \geq f_{ij}) & l \in L_S \\
 & t_i \in \mathbb{R}_+ \forall t_i \in S
 \end{aligned} \tag{4}$$

where  $M_S := M \cap S$  are those variables appearing in  $\mathcal{M}^\alpha$  and  $S^\alpha$  and that are fixed in the subproblem to the solution of the master problem  $\mathcal{O}^\alpha$ ; in accordance to Benders decomposition.

In Fig. 2, we illustrate a possible decomposition of the example given in Fig. 1 by illustrating the disjunctive graphs of the master and

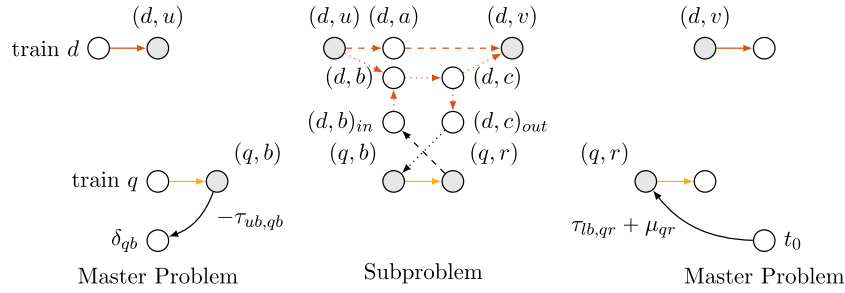


Fig. 2. Example of a decomposition on the generalized disjunctive graph.

subproblem. We define for the example  $A_{M,f} = A_f \setminus \{(q,b), (q,r)\}$ ,  $A_{M,h} = A_h$ ,  $L_M = \{\}$  and  $A_{S,f} = \{(q,b), (q,r)\}$ ,  $A_{S,h} = \{\}$ ,  $L_S = \{1, 2\}$ . Variables appearing in both, master and subproblem, i.e., variables  $M_S$ , are shaded in gray in Fig. 2.

### 5. Reusing logic Benders cuts

#### 5.1. Preliminaries

We reuse logic Benders cuts to accelerate the solution process of a rescheduling problem as we extend the master problem  $\mathcal{M}$  of an instance  $\mathcal{R}'$  by some logic Benders cuts  $B_R$  from a larger library  $\mathcal{B}$  of precomputed Benders cuts. The library of cuts has been computed by solving another instance(s)  $\mathcal{R}$  of MRR, which describes the same infrastructure, timetable and traffic as  $\mathcal{R}'$ , but under different input delays.

In case instances of MRR differ only by the input delay  $\mu_i$  we show in Section 5.2 that optimality remains in case logic Benders cuts are reused among these instances, and the reuse of logic Benders cuts is a valid process.

Formally we define an instance  $\mathcal{R}$  of MRR by the tuple  $\mathcal{R} := (A_f, A_h, A_\delta, A_s, L)$ . We denote as  $\partial(\mathcal{R})$  the class of all instances which differ from an instance  $\mathcal{R}$  of MRR only by input delay, and use the same decomposition. Those instances obviously contain the same number of events, precedence relations and decisions.

As discussed in Section 1, most railways are operated on a daily repeating schedule. Every day during the same time of the day, the same trains can be found on the railway network, and in fact the instances of MRR solved throughout the days are always considering the same trains, timetable and infrastructure. With our approach, we exploit this basic property of rescheduling problems. We are moreover able to quantify a similarity of specific realizations of input delays (and therefore of instances), based on the fact that we can statistically characterize the input delays.

In Section 6.2, we will introduce a measure to identify, within this class of instances (and therefore class of cuts), instances which are more similar, and others which are less similar. The measure is based on the feature space of the respective sets  $A_h$  and  $A'_h$ , which describe the input delays  $\mu_i$  and  $\mu'_i$ . We will show that with a measure to quantify the similarity between instances, we can estimate the computational benefit of particular Benders cuts for their reuse.

#### 5.2. Modification of a logic Benders cut to preserve validity

We consider the reuse of a logic Benders cut as valid, in case optimality is preserved. We can show that with an appropriate modification, the reuse of a cut between instances of MRR, which only differ in input delay, is guaranteed not to cut off any feasible solution, which implicitly preserves the optimality.

In Section 4.2, we applied the logic Benders decomposition (Leutwiler and Corman, 2022) to the problem of rescheduling. In Leutwiler and Corman (2022), a logic Benders cut has been introduced, which can be used for our decomposition of Problem (2) in Section 4.2.

In principle, the logic Benders cut summarizes the constraints of the subproblem for the master problem, such that a solution of the master problem is consistent with the constraints of the subproblem and allows for a feasible solution in the subproblem.

Regarding the decomposition in Section 4.2, let us assume that  $p$  is a path (of fixed arcs) on the disjunctive graph of the subproblem  $S^\alpha$  and between two variables of  $M_S$  (one of them possibly  $t_0$ ). Then, a solution of the master problem must satisfy the sum of all constraints (arcs) along such path, to make sure that a solution to the subproblem exists. In general, a path  $p$  on the disjunctive graph of the subproblem  $S^\alpha$  contains selectable arcs and the existence of the path depends upon the selection of particular arcs. In Leutwiler and Corman (2022) it was shown that we can derive a set of paths (here denoted by  $\mathcal{P}$ ) between variables of  $M_S$ , for which we are guaranteed that any solution of the subproblem satisfies the constraints along at least one path  $p \in \mathcal{P}$ ; the set is derived by a proof of infeasibility on the subproblem. As all paths in  $\mathcal{P}$  are between variables of  $M_S$ , a solution of the master problem must satisfy the constraints along at least one path in  $\mathcal{P}$  as well. Otherwise, the subproblem has no feasible solution for the considered solution of the master problem. From the set of paths  $\mathcal{P}$ , the following logic Benders cut has been introduced in (Leutwiler and Corman, 2022),

$$\beta = \bigvee_{p \in \mathcal{P}} (t_{e(p)} - t_{s(p)} \geq l_p). \tag{5}$$

In the cut (5),  $s(p)$  and  $e(p)$  are the start and end node of path  $p$  respectively (both variables of  $M_S$ ) and  $l_p$  is the length of the path; the constraint  $t_{e(p)} - t_{s(p)} \geq l_p$  is the result of summing up all left and right hand-sides of constraints along  $p$  and summarizes the constraints along  $p$  for variables of  $M_S$ .

Between two instances of MRR  $\mathcal{R}$  and  $\mathcal{R}' \in \partial(\mathcal{R})$  only constraints of  $A_h$  change; it holds that  $A_{S,s} = A'_{S,s}$  and  $L_S = L'_S$ , i.e., selectable arcs and decisions remain identical. Therefore, for any set of paths  $\mathcal{P}$  on the disjunctive graph of  $S^\alpha$  (of  $\mathcal{R}$ ), where any solution of  $S^\alpha$  satisfies the constraints along at least one path  $p \in \mathcal{P}$ , the same holds for the set  $\mathcal{P}$  considered on the disjunctive graph of  $S'^\alpha$  (of  $\mathcal{R}'$ ) and solutions of  $S'^\alpha$ . Constraints  $A_h$  and  $A'_h$  of  $\mathcal{R}$  and  $\mathcal{R}'$  may differ because they show a different right hand-side  $\mu_i$  (resp.  $\mu'_i$ ), such that a path  $p$  may have different length considered on the disjunctive graph of  $S^\alpha$  or  $S'^\alpha$ . To reuse a logic Benders cut (in the form of Eq. (5)) generated for  $\mathcal{R}$  on  $\mathcal{R}'$ , we adjust all right hand-sides in the linear constraints of the logic Benders cut (5) according to  $\mu'_i$  of  $S'^\alpha$  to

$$\beta' = \bigvee_{p \in \mathcal{P}} (t_{e(p)} - t_{s(p)} \geq l'_p) \tag{6}$$

where  $l'_p$  is the length of path  $p$  considered on the disjunctive graph of  $S'^\alpha$ . With the modification of  $\beta$  to  $\beta'$  and the fact that selectable arcs and decisions remain identical between  $\mathcal{R}$  and  $\mathcal{R}'$ , we are guaranteed that  $\beta'$  is a valid cut for  $\mathcal{R}'$  and optimality remains under the reuse of cuts.

**Algorithm 1:** Lazy-Constraint Reuse of logic Benders Cuts

---

```

input : Master  $\mathcal{M}$ , Subproblem  $S$ , Library  $B$ 
output:  $\mathcal{O}_C$ 
init :  $\alpha = 0$ ,  $B^\alpha = \emptyset$ ,  $\mathcal{O}_C = \emptyset$ ,  $B_R = \emptyset$ ,
1 while  $\mathcal{O}_C = \emptyset$  do
2   do
3      $\mathcal{O}_M^\alpha \leftarrow \text{solve}(\mathcal{M}^\alpha(B_R))$ 
4      $B_V \leftarrow \text{getViolatedCuts}(\mathcal{O}_M^\alpha, B)$ 
5      $B_R \leftarrow B_R \cup B_V$ 
6   while  $B_V \neq \emptyset$ 
7      $\mathcal{O}_S^\alpha, B_S^\alpha \leftarrow \text{SMT}_{\text{Agg}}(S^\alpha(\mathcal{O}_M^\alpha))$ 
8     if  $\mathcal{O}_S^\alpha = \emptyset$  then
9        $B^{\alpha+1} \leftarrow B^\alpha \cup B_S^\alpha$ 
10    else
11       $\mathcal{O}_C \leftarrow \mathcal{O}_M^\alpha \cup \mathcal{O}_S^\alpha$ 
12     $\alpha \leftarrow \alpha + 1$ 

```

---

## 6. Proposed approach

### 6.1. A lazy-constraint approach

In this section we propose an implementation for the reuse of logic Benders cuts. We first propose a lazy-constraint approach for the iterative extension of the master problem in our decomposition by precomputed Benders cuts. The idea is to include only cuts, which are violated by the incumbent solution of the master problem and ignore the rest. This allows to keep the master problem smaller and with only cuts which potentially provide progress.

Given a library  $B$  of precomputed and modified logic Benders, which can be potentially reused, it is difficult to estimate in advance, which of these cuts will bring a computational benefit if reused. In case the master problem is extended by a precomputed cut, which does not affect any optimal solution or is dominated by those already in the master problem, the cut will add no computational benefit. Instead, the additional constraint will only decrease the computational performance of solving the master problem, as the master will increase in size. With the design of a lazy-constraint approach we aim to prevent such phenomena and exclude cuts with no computational benefit from the reuse.

We reuse logic Benders cuts in a lazy manner, to avoid including cuts with no computational benefit. That is, given a library of cuts for reuse, we evaluate in an iterative manner, which cuts are violated by the incumbent master solution and only add those cuts to the master problem. After every extension by violated cuts, we compute a new master solution and check again for further violated cuts. The lazy-constraint procedure is integrated in the entire process of the Benders decomposition as only after no further violated cuts are found in the library, the subproblem is queried for feasibility, to possibly generate further, new logic Benders cuts.

The complete lazy-constraint approach is illustrated in Algorithm 1, where  $\text{SMT}_{\text{Agg}}$  (see Appendix) is the algorithm of Leutwiler and Corman (2022) by which we address the subproblem. The expression  $\mathcal{O}_C$  denotes a solution to the centralized problem; as far as there is an infeasibility in the subproblem, a centralized solution cannot be found and the algorithm iteratively proceeds. In an iteration of Algorithm 1, Lines 3–5 adds violated constraints  $B_V$  from the library  $B$  in a lazy manner. The master problem is solved again in Line 3 with an iteratively growing set of cuts  $B_R$  considered, for every non-empty set of violated cuts  $B_V$  in Line 4, until no further violated logic Benders cut is found in the library  $B$ . In Line 7 the subproblem is analyzed using the  $\text{SMT}_{\text{Agg}}$  algorithm of Leutwiler and Corman (2022), which either returns a set of new Benders cuts to be added to the master problem,

or a feasible solution for the subproblem. Lines 8–11 either add any new Benders cut to the master problem or generate a feasible solution for the centralized problem, if the subproblem is feasible.

With the lazy-constraint approach we define two types of logic Benders cuts in reuse:

**Definition 1 (Useless Logic Benders Cut).** We denote a logic Benders cut  $\beta$  as useless, if  $\beta \in B$  but  $\beta \notin B_R$  after the termination of Algorithm 1. In other terms, in no iteration the cut was found to be violated by the incumbent master solution.

**Definition 2 (Useful Logic Benders Cut).** We denote a logic Benders cut  $\beta$  as useful, if  $\beta \in B$  and  $\beta \in B_R$  after the termination of Algorithm 1. In other terms, in at least one iteration the cut was violated by the incumbent master solution.

Useless logic Benders cuts are cuts that provide no computational benefit when reused; if such cuts are not excluded from the reuse, these cuts can significantly decrease the computational speed of solving the master problem.

With the lazy-constraint approach we can further define a perfect set of logic Benders cuts as:

**Definition 3 (Perfect Set of Logic Benders Cuts).** We denote a set of logic Benders cuts  $B$  as perfect if  $B^\alpha = \emptyset$  after the termination of Algorithm 1.

The perfect set of logic Benders cuts leads to a termination of Algorithm 1 after the first full iteration between master problem and subproblem. In case of a perfect set of cuts, no further cuts must be generated from the subproblem to determine an optimal solution for the centralized problem. All necessary cuts are within the perfect set. From a computational point of view, this is the ideal situation.

### 6.2. Identifying the computational benefit of reusable logic Benders cuts

With the lazy-constraint approach (Algorithm 1) we are able to exclude cuts with no computational benefit from inclusion in the master. Unfortunately, we are not guaranteed that useful cuts will actually lead to a computational speedup, in case they are reused. An excessive amount of cuts, despite being useful, actually decreases the performance of decomposition in case reused. In other terms, two processes are contrasting each other: the master getting larger and slower as more cuts are included, and the master getting faster (and the entire scheme having less iterations) in case the best cuts are added. In this case, a method is necessary to not only identify the useful cuts, but actually prioritize those few, which add a computational benefit to the entire solution process.

We assume that precomputed Benders cuts add the most computational benefit for reuse, in case the instance, which has been used to precompute cuts, shows similar input delays of trains as the instance for which cuts are being reused. As such we expect that for an instance  $\mathcal{R}' \in \partial(\mathcal{R})$ , cuts computed when solving  $\mathcal{R}$  add the most computational benefit, if for each  $(i, j) \in A'_h$  of  $\mathcal{R}'$  there exists a  $(i, j) \in A_h$  of  $\mathcal{R}$  where  $\mu_i \approx \mu'_j$ .

We propose in this section two different measures to quantify the similarity in input delay  $\mu$ . To this purpose, we summarize the situation of updated starting times of relevant events, including the input delays, of an instance  $\mathcal{R}$  as we write all right hand-side  $\tau_{l_b,i} + \mu_i$ ,  $(0, i) \in A_h$  in a vector  $v_{\mathcal{R}}$ . By definition of  $\partial(\mathcal{R})$ , all vectors  $v_{\mathcal{R}}$  and  $v_{\mathcal{R}'}$  we consider in reuse, have the same dimension. For a pair of instances  $\mathcal{R}$  and  $\mathcal{R}' \in \partial(\mathcal{R})$ , we quantify similarity in input delay by the euclidean distance and the Sørensen–Dice coefficient (Dice, 1945) on the respective vectors  $v_{\mathcal{R}}$  and  $v_{\mathcal{R}'}$ . The euclidean distance is defined as

$$e_{v_{\mathcal{R}}, v_{\mathcal{R}'}} = \|v_{\mathcal{R}} - v_{\mathcal{R}'}\|. \quad (7)$$

**Table 1**  
Overview of original timetabling instance.

Time horizon	Subproblems	Trains	Blocks	Routing alternatives	Resource conflicts
08:00–09:00	40	80	626	790	3240

In case  $e_{v_{\mathcal{R}}, v_{\mathcal{R}'}} = 0$ ,  $v_{\mathcal{R}}$  and  $v_{\mathcal{R}'}$  are identical. The Sørensen–Dice coefficient is a measure to quantify the similarity of vectors and can be computed as,

$$s_{v_{\mathcal{R}}, v_{\mathcal{R}'}} = \frac{2 |v_{\mathcal{R}} \cdot v_{\mathcal{R}'}|}{|v_{\mathcal{R}}|^2 + |v_{\mathcal{R}'}|^2}. \quad (8)$$

In case  $s_{v_{\mathcal{R}}, v_{\mathcal{R}'}} = 1$ , the vectors  $v_{\mathcal{R}}$  and  $v_{\mathcal{R}'}$  are identical. Values of  $s_{v_{\mathcal{R}}, v_{\mathcal{R}'}}$  between 0 and 1 indicate a measure of similarity between  $v_{\mathcal{R}}$  and  $v_{\mathcal{R}'}$ .

## 7. Computational experiments

In this section we conduct comprehensive experiments to study the reuse of logic Benders cuts. Compared to the theory above where only one subproblem is considered, we conduct our experiments on a decomposition with 40 different subproblems. The extension of the theory to the case of multiple subproblems is straightforward. The decomposition is chosen such that each subproblem can be treated individually without dependencies to other subproblems. The decomposition itself is a geographical decomposition, identical to the decomposition in [Leutwiler and Corman \(2022\)](#). The commercial solver Gurobi ([Gurobi Optimization, 2021](#)) is used to solve the master problem.

All experiments were run on the Euler cluster of ETH Zurich on a Intel(R) Xeon(R) CPU E3-1585L v5 @ 3.00 GHz processor with 4 GB of RAM.

### 7.1. Instances and delay scenarios

We consider for our computational experiments the railway traffic within the triangle of the cities Zurich, Luzern and Chur in Switzerland between 08:00 and 09:00 in the morning; [Table 1](#) provides an overview in numbers.

For our experiments we consider a typical Monte Carlo scheme, where instances  $\mathcal{R} = (A_f, A_h, A_\delta, A_s, L)$  differ only by the input delay  $\mu$  considered in the constraints in  $A_h$ . The input delay is generated with the goal of having sufficiently different patterns of input delay to understand the potential and limit of the approach. The procedure goes as follows.

First, we randomly determine the trains which are experiencing an input delay. Instead of using a probability for the likelihood of a train being delayed, we first decide, based on a uniform random distribution, how many out of all trains experience a delay (ranging from none, to all). We then randomly select such number of trains out of all trains. In this way, we achieve an equal probability for the number of delayed trains over our generated MRR instances. Given the trains experiencing a delay, we determine for each such train a single relevant event that is a departure, where a delay should occur. The event is randomly selected out of all departure events related to the train. At last, we define the value of input delay  $\mu_i$  for each selected relevant event  $i$ . We sample the value of the input delay from a Weibull distribution according to [Corman et al. \(2011\)](#) (see parameters in [Table 2](#)). We do not consider negative input delays in this work, such that we resample from the Weibull distribution as long as we receive a negative delay value.

For the experiments, we randomly generate two sets of instances, with the same procedure and same statistical properties. A first set of 100 instances is the *test set* and is used to evaluate the performance of our algorithm, with/without reuse of cuts. A second set of 1000

**Table 2**  
Weibull parameters ([Corman et al., 2011](#)).

Scale	Shape	Shift
395	2.5	-315

instances is the *training set* and is solved in advance by the normal Benders decomposition (i.e. without considering any library of cuts for reuse) to generate a large library  $\bar{\mathcal{B}}$  of logic Benders cuts. From this latter large library, we identify the actual library  $\mathcal{B}$  available for reuse in [Algorithm 1](#).

### 7.2. Computational benefit of Benders cuts

In a series of experiments we show the computational benefit of cuts by taking a perfect set of cuts and changing the proportion of perfect cuts against non-perfect (random) cuts in such set. We designed two experiments, one with a library of fixed size (i.e., when adding a non-perfect cut, we remove one perfect cut), and one with a library of increasing size (i.e. the perfect cuts are always considered, but increasingly many non-perfect cuts are included).

We create for each of our test instances a perfect set (library)  $\mathcal{B}^*$  of logic Benders cuts for the reuse. We create such sets by applying, for each test instance, [Algorithm 1](#) multiple times and iteratively collecting and reapplying the generated cuts, until a perfect set has been found. We have to reapply generated cuts because [Algorithm 1](#) is rather sensitive to  $\mathcal{B}$ . Therefore, for different cuts considered in input  $\mathcal{B}$ , different cuts might be generated as  $\mathcal{B}^\alpha$  from the subproblem analysis. To generate  $\mathcal{B}^*$ , we start for each instance with an empty set of generated cuts  $\mathcal{B}_{gen} = \emptyset$ , which we use as library  $\mathcal{B}$  for [Algorithm 1](#). After the first termination of [Algorithm 1](#), we can retrieve from the algorithm the set  $\mathcal{B}^\alpha$ , i.e., the set of all Benders cuts generated by the subproblems during this first run of the algorithm. Such set  $\mathcal{B}^\alpha$  is not necessarily perfect yet. In case  $\mathcal{B}^\alpha$  is reused, it can occur that other cuts might be needed as the master problem results in different intermediate optimal solutions, due to the reused cuts. To create a perfect set, we set  $\mathcal{B}_{gen} = \mathcal{B}_{gen} \cup \mathcal{B}^\alpha$ , i.e., we incrementally extend the set of generated cuts by  $\mathcal{B}^\alpha$  and consider this extended set  $\mathcal{B}_{gen}$  as input  $\mathcal{B}$  for [Algorithm 1](#). This will consider all previously generated cuts when we run [Algorithm 1](#) again.

We repeat this process until no further cuts are generated from any subproblem during [Algorithm 1](#), i.e.,  $\mathcal{B}^\alpha = \emptyset$ . Finally, when  $\mathcal{B}^\alpha = \emptyset$ , the set of cuts  $\mathcal{B}_{gen}$  is a perfect set of cuts, but it might contain useless cuts. We avoid useless cuts in the perfect library  $\mathcal{B}^*$  by taking only the useful cuts from  $\mathcal{B}_{gen}$ . We can get the useful cuts which are in  $\mathcal{B}_{gen}$ , by considering the latest run of [Algorithm 1](#) where  $\mathcal{B}^\alpha = \emptyset$ . In this run of [Algorithm 1](#), after termination, the set  $\mathcal{B}_R$  corresponds exactly to those cuts of  $\mathcal{B}_{gen}$  that are useful, i.e., that have been violated at some point during [Algorithm 1](#).

A perfect sets of cuts generated by this procedure is not necessarily minimal, but as we use  $\mathcal{B}_R$  at the end of the procedure,  $\mathcal{B}^*$  is guaranteed not contain any useless cuts.

We consider in this subsection a series of libraries for each instance of the testing set, all of the size of the corresponding perfect library  $|\mathcal{B}^*|$ , where an increasing amount of cuts from the original perfect library  $\mathcal{B}^*$  are substituted with other (random) cuts from other delay cases, randomly chosen from  $\bar{\mathcal{B}}$ . We assume those latter have on average low computational benefits.

In [Fig. 3](#) we illustrate the performance on average over all test instances for an increasing proportion of random cuts in a library  $\mathcal{B}$  of fixed size. The three figures report respectively from top to bottom: the computational time, normalized by the case with no reuse, i.e.,  $\mathcal{B} = \emptyset$ ; the amount of iterations until the solution is found; and the amount of Benders cuts  $|\mathcal{B}_R|$  actually included in the master from the library  $\mathcal{B}$ . The top two plots report the lazy-constraint approach (red), the direct



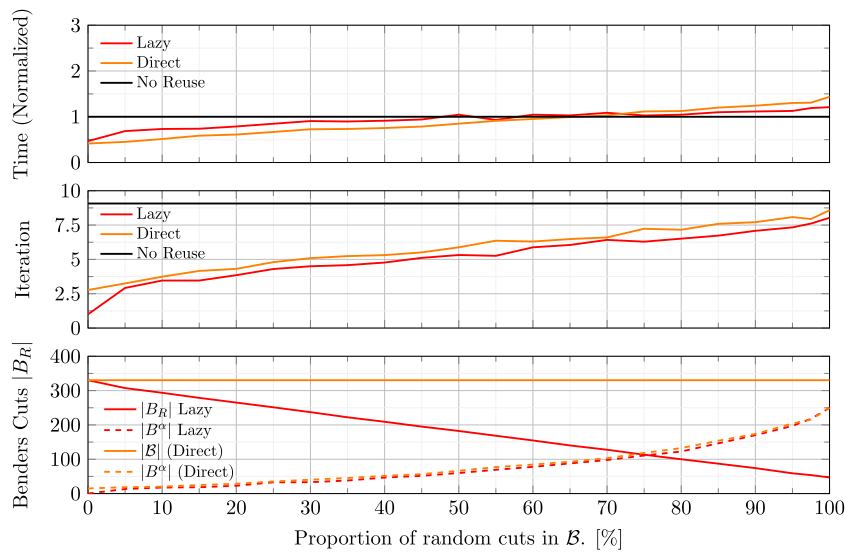


Fig. 3. Time, iteration and cuts for direct reuse, lazy-constraint vs. no reuse.

reuse (orange) and the case of no reuse (black). The bottom plot reports in solid lines the total amount of cuts reused from the library, i.e.,  $|B_R|$ , and in dotted lines the total amount of cuts newly generated in the iterative process. In red is the lazy-constraint approach, in orange the direct reuse.

In the top plot of Fig. 3, we can see that both, the lazy-constraint approach as well as the direct reuse, show a speedup around 2 for the perfect cuts (the left end of the plot). With an increasing substitution of perfect cuts by random cuts, the computational performance decreases. Above 65% of random cuts, the reuse performs worse than the normal decomposition where no reuse is considered. An explanation for the decrease in performance is given in the middle and bottom plot of Fig. 3. We can see that with an increasing proportion of random cuts, the necessary iterations till convergence increase, almost up to the number of the decomposition with no reuse, i.e., when  $\mathcal{B} = \emptyset$ . The increasing amount of iterations is caused by the lack of useful cuts in the library  $\mathcal{B}$ , due to the increasing number of random cuts. In other words, the random cuts are not resulting in less iterations, and instead only increase the size of the master problem. The lack of useful cuts is shown in the bottom plot of Fig. 3, where for the lazy-constraint approach the number of cuts reused ( $B_R$ ) from the library  $\mathcal{B}$  continuously decreases for higher proportions of random cuts, while the number of newly generated cuts  $B^\alpha$  (dashed line) increases.

Furthermore, at 100% random cuts, lazy-constraint still finds that some of the random cuts are useful, see bottom plot, and includes these in the master problem. The computational performance in such case lies above 1, i.e., it is slower than no reuse. We thus empirically find that even cuts determined as useful can result in a slow down of the decomposition scheme.

A further interesting point in Fig. 3 is the fact that our perfect sets of Benders cuts are perfect when using the lazy-constraint approach, but not perfect under the direct reuse. We see this empirically as in the middle plot of Fig. 3, the average number of iterations for direct reuse is around 2.5 instead of 1, as in the lazy-constraint approach. This is caused by the fact that Gurobi, which is used to solve the master problem in both cases, computes different optimal solutions depending on whether cuts are added iteratively, i.e., in a lazy manner, or all together. Therefore it is possible that in the direct reuse a minimal amount of further iterations and logic Benders cuts are necessary to converge.

### 7.3. Issues from excessive amount of Benders cuts reused

With a second series of experiments we aim to study the decreasing performance in experiments of Section 7.2, for large proportions of

random cuts. We want to estimate whether this is caused by the lack of computationally beneficial cuts or the presence of random but useful cuts. Random cuts might distract the decomposition scheme from converging quickly. If the latter case applies, we clearly must take care in generating the library  $\mathcal{B}$  for actual reuse.

In this second series of experiments we consider libraries, where on top of the complete perfect set of Benders cuts  $B^*$  for our test instances (which are definitely useful), an increasing amount of additional random cuts is added. Compared to the experiments of Section 7.2, the libraries considered increase in size from  $|B^*|$  to  $\sim 40|B^*|$ ; and are always a superset of the perfect sets  $B^*$  for the individual test instances.

Fig. 4 reports the computational results for variable library sizes, in an analogous manner to Fig. 3. Starting from the top plot, we can see again the performance increase of a factor 2 for the reuse of just the perfect set of cuts (i.e., the left end of the plot is at 0.5). We can further see, that with an increasing amount of additional random cuts in the library, the computational performance of both, direct and lazy-constraint reuse, decreases. The bottom plot of Fig. 4 (with logarithmic y-axis), explains such decrease, due to a significant increase in the size of the master problem. The plots in Fig. 4 further illustrate well, the two main effects of excluding useless cuts from reuse as in the lazy-constraint approach. On the one hand, the absence of useless cuts leads to a smaller master as shown in the bottom plot of Fig. 4. On the other hand, as the middle plot shows, the absence of useless cuts leads to a reduced number of iterations in the decomposition scheme. Both lead to a significant computational speedup, as shown in the top plot. Also, including useless cuts in the library does not have a systematic effect on the amount of iterations required until convergence of the scheme (the lines in the middle plot are mostly horizontal).

Overall, we empirically conclude from Fig. 4 that reusing an excessive amount of Benders cuts, even if useful, negates the overall benefits of reusing computationally beneficial logic Benders cuts, such as the perfect cuts  $B^*$ . Furthermore, we can see in Fig. 4 that also the lazy-constraint approach, which considers only useful cuts, shows a slow down in computational performance. This is due to random, but useful cuts, which are included, and increase the size of the master without causing a desired decrease in iterations of the scheme. As a result, they slow down the computational performance of the solution process.

The above clearly confirms the importance of carefully selecting the cuts in the library for the reuse, beyond their usefulness, but based on (some measure of) their computational benefit to the entire solution process, which is the topic of the next subsection.

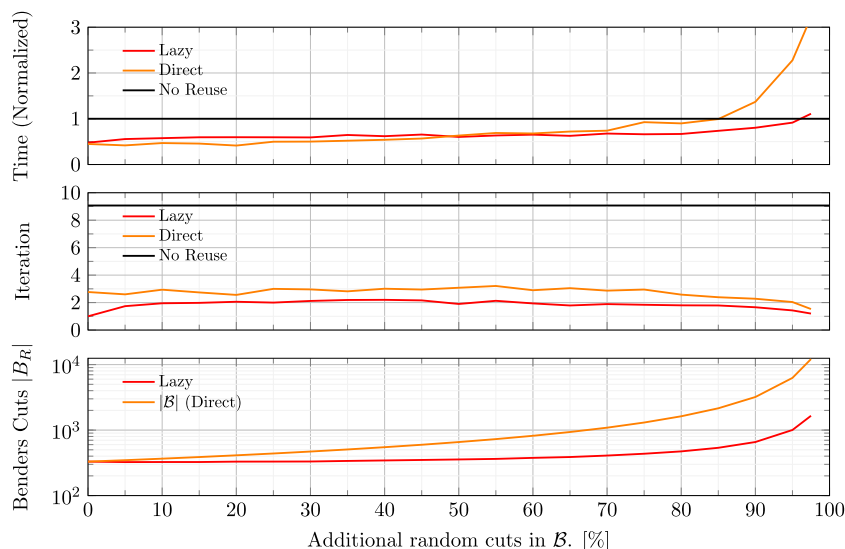


Fig. 4. Time, iteration and cuts for direct reuse, lazy-constraint vs. no reuse.

Table 3  
Potential when reusing cuts from libraries of logic Benders cuts.

Approach	Perfect library	Best library	Avg. library	No library	Gurobi
Time [s]	32.30	33.19	94.31	81.86	201.24
Normalized time	0.45	0.46	1.16	1	3.92
Iterations [-]	1 (5.85 <sup>a</sup> )	4.04 (8.01 <sup>a</sup> )	7.60 (12.96 <sup>a</sup> )	9.07	-
Total Cuts [-]	345.21	307.64	359.01	382.40	-

<sup>a</sup>Total Master Solves, including Lazy-Constraint.

#### 7.4. Reuse of cuts from libraries of similar instances

In this section we analyze the reuse of logic Benders cuts under real-world conditions. In this case, it is unlikely that a perfect set of precomputed cuts is available. We thus create a series of experiments where we pair the libraries computed each by a single training instance (out of the 1000 training instances), with our 100 test instances. This results in 100'000 pairs, matching a single training library (i.e., the cuts computed on a single training instance) and a single test instance (to apply the library). Which such “Training Library - Test Instance” pairs, we want to understand under which conditions logic Benders cuts result in a computational benefit. The experiments in this section are exclusively computed using our lazy-constraint approach.

In Table 3 we report computational results for all pairs of training library - test instance, as well as the ideal performance using the perfect library; and two further benchmarks for a general comparison in performance. The columns of the Table 3 are as follows.

The Perfect Library repeats the ideal performance from the previous sections, in case the perfect set of Benders cuts is reused. The Best Library considers, for each test instance, the computational performance using the “best” library, i.e., the one, out of the 1000 libraries computed by the 1000 instances of the training set, which results in the smallest computational time. The Average (Avg.) Library reports the computational performance on average over all pairs of training library - test instance. This would correspond of taking a random training instance and reusing its cuts on a random test instance. As benchmarks, we report in Table 3 also the computational results of solving the test instances with the normal logic Benders decomposition of Leutwiler and Corman (2022) as No Library, and the solution of the centralized Problem (2), directly computed by the commercial solver Gurobi (Gurobi Optimization, 2021). All approaches in Table 3 report an optimal solution within a tolerated optimality gap of 1% (on solutions of the master problem), which is considered sufficient for practical applications.

Overall, Table 3 clearly shows the advantage of decomposition over the centralized benchmark of Gurobi. Further, we can see that with the reuse, in case of the best library, we can achieve a speedup of a factor similar to the perfect library, empirically proving the potential of reusing logic Benders cuts also under real-world conditions.

Table 3 indicates that a speedup by the reuse of precomputed logic Benders cuts is not guaranteed in case of an arbitrary reuse of cuts, i.e., for the average library it is 16% slower. This restates the importance of selecting carefully the cuts for reuse.

The performances in different cases of decomposition in Table 3 can be explained by the number of iterations, in particular solves of the master problem, and total amount of Benders cuts in the master problem. Due to the lazy-constraint approach, the number of times the master problem is solved is higher than number of iterations done by the decomposition scheme in cases of reuse. In Table 3, the perfect library and the best library show significantly less iterations (and slightly less master solves) than the no library case, which explains the observed speedup in both cases. Also, in case of the lazy-constraint, most of the master solves are generally performed in the first iteration of the Benders scheme. In the first iteration, the master is usually small in terms of additional Benders cuts and thus rather efficient to be solved, compared to a master problem after several iterations. In other terms, master solves in the No Library case are in general more complex. The reason for the rather unsatisfying performance of the average pairs in Table 3 has been already commented in Section 7.2. Random cuts, which most of training cuts seem to be, are more harmful than useful, and tend to slow down the solution procedure.

In Fig. 5 we report a histogram of the computational performance (x-axis, normalized by no reuse), for all pairs of training library - test instance. This describes the empirical probability for the performance of pairs. The expected value of this histogram is the performance of the average library, i.e., 1.16. In the histogram of Fig. 5 the total probability for a pair to result in a computational benefit is 48.2%, while the probability of a computational degradation is 51.2%. Therefore, a random cut is slightly more likely to result in a slow down,

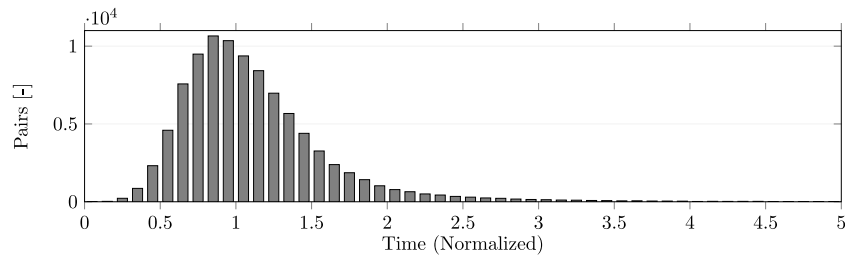


Fig. 5. Histogram of computational times for pairs of training library - test instance.

than a speedup. In the histogram of Fig. 5, 26.2% of the pairs show a computational time of 0.8 or less; we will consider a normalized computation time of 0.8, which corresponds to 20% faster computation, to be a significant enough speedup for practical considerations.

The variability in Fig. 5 illustrates the existence of cuts with high and low computational benefit in the reuse. Clearly, the pairs of best libraries reported in Table 3 relate to pairs that are on the left in Fig. 5, i.e., with a normalized computational time  $< 1$ . These are the pairs, i.e., the libraries, we wish to identify efficiently in a real-life application to improve computational speed of the decomposition process.

#### 7.5. A measure to estimate speedup resulting from a library

In the previous two sections, we analyzed the computational benefits of libraries of logic Benders cuts, and the impact of the excessive reuse of cuts. We concluded that for a real-world application, an estimation of the computational benefit of (a library of) logic Benders cuts is necessary, to limit the reuse of logic Benders cuts to those with a high computational benefit only. In Section 6.2, we proposed two measures to estimate such computational benefit, at the level of libraries generated from a single instance. Those measures describe the similarity of input delay between the instance determining the library, and the target instance we want to solve. The ultimate goal is to identify a-priori the libraries, which lead to a performance as good as the best library, which is reported in Table 3; and in general, the libraries which approximate a performance comparable to using the perfect set of logic Benders cuts.

To estimate the accuracy of our similarity measures in indicating the computational benefit of a library of logic Benders cuts, we analyze the euclidean distance and the Sørensen–Dice coefficients on the pairs of training library - test instance from Section 7.4. In particular, we analyze the relation between the speedup (i.e., how smaller the computational time is; normalized by the computational time when no cut is reused) achieved by a particular training library - test instance, and the similarity measures. We evaluate our measures for each pair using the training instance, from which the library has been generated; and the test instance, on which the library has been reused. We consider libraries generated by only a single instance.

In Fig. 6 we report a histogram of values on the Sørensen–Dice and euclidean measures, over the 100'000 pairs of Section 7.4. In Fig. 6, we overlay in red the average normalized computational times, which are computed as average over the pairs in the respective individual bin. The top plot in Fig. 6 reports on the Sørensen–Dice measure. The plot empirically confirms the ability of the measure to discriminate between libraries which have different computational benefit. Specifically, while the majority of pairs in Fig. 6 show no computational benefits, above a Sørensen–Dice value of 0.9998 the libraries being reused clearly bring a computational benefit. The red line decreases to the right, below 0.8, for a maximum speedup of 0.5 (twice as fast) for Sørensen–Dice values close to 1. Such an observed speedup matches with the reported speedup in case of a set of perfect cuts (0.46) in Section 6.1, i.e., cuts generated from an instance with a Sørensen–Dice value of exactly 1. The bottom plot of Fig. 6 reports the case

of the euclidean distance, and in general confirms a similar ability to discriminate libraries with high computational benefit from those of no benefit. In this case, values of the euclidean distance below 700 are connected with a decrease in normalized computational time below 0.8. In both plots of Fig. 6, fluctuations in the computational times are caused by sample approximations, and low numbers of samples for the extreme cases.

The Sørensen–Dice measure shows a slightly better discriminating power than euclidean. In case of Sørensen–Dice, 1.38% of all pairs are categorized into bins with an average normalized computational time below 0.8. In comparison, for the euclidean distance, only 1.17% of all pairs were categorized into bins with a computational time below 0.8. Note as a reference that up to 26.2% of the pairs have a speedup of 0.8 or better (see Fig. 5).

The histograms reported in Fig. 6 give further an estimate on the statistics of the similarity measures over instances. From our experiments, reported in Fig. 6, out of the 100 test instances, only 23 instances have a matching library, inside the training set of 1000 instances, where the training instances have a measure above 0.9998 in Sørensen–Dice and below 700 in euclidean. This calls for a more diversified set of training instances, either achieved by an even larger training set, or by a different sampling of the domain of input delays. For instance, one can target coverage of the delay domain, by Sobol sampling; and not match the density of the probability, like the simple Monte Carlo scheme we used. For an actual real-life application, thus, we would expect to increase the set of instances for the precomputation of logic Benders cuts. This would increase the probability of finding good cuts for a wide range of instances, ahead of time, as it is required in reality.

Finally, we analyze the benefits of the similarity measure for computational benefit of cuts under real-world conditions. For this analysis we first focus on 23 testing instances, for which we have available highly similar training instances (libraries), i.e., those with at least one library showing a measure above 0.9998 in Sørensen–Dice or below 700 in the euclidean distance. We report in Fig. 7 a similar analysis as in Fig. 4, with the top plot showing the normalized computational time, the middle plot the iterations, and the bottom plot the amount of cuts considered. Instead of reporting the performance of the best library (in terms of speedup) over the entire training set, which in reality would be a-priori unknown, we report the average performance over the increasing set of the most similar libraries.

Specifically, we rank the instances by the similarity measure, and we run the algorithms with each library from the best  $k$  ranked ones. We then take the average performance over this set. We consider an increasing  $k$ , and analyze how the performance changes. The x-axes of those three plots are the same, and describe a logarithmically increasing set of training library - test instance pairs  $k$  considered from 1 (i.e., only the best ranked), up to 1000 (i.e. all libraries available).

We rank the libraries separately for Sørensen–Dice (decreasing), reported in all plots as orange solid line; and euclidean (increasing), reported in all plots as red, dashed line. There are actually very little differences between the two measures. We also report a random selection of libraries, i.e., without a similarity measure (reported in black).

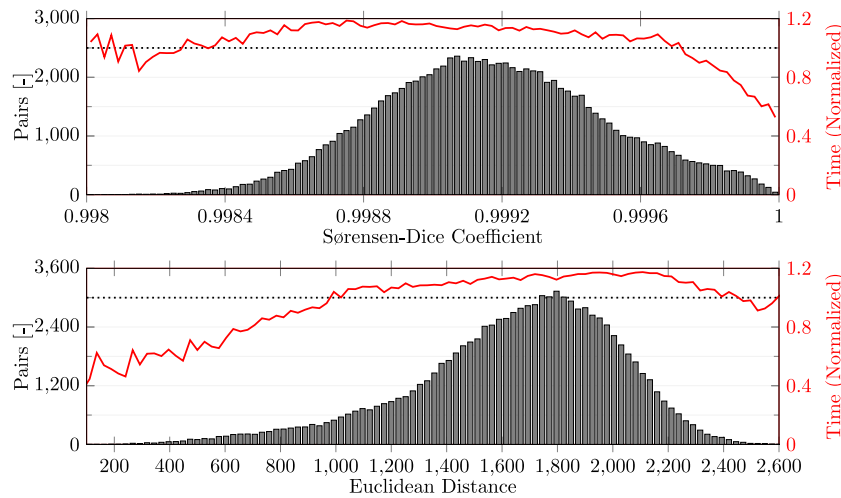


Fig. 6. Histogram of similarity measures over pairs of training library - test instance, overlaid with computational time (red).

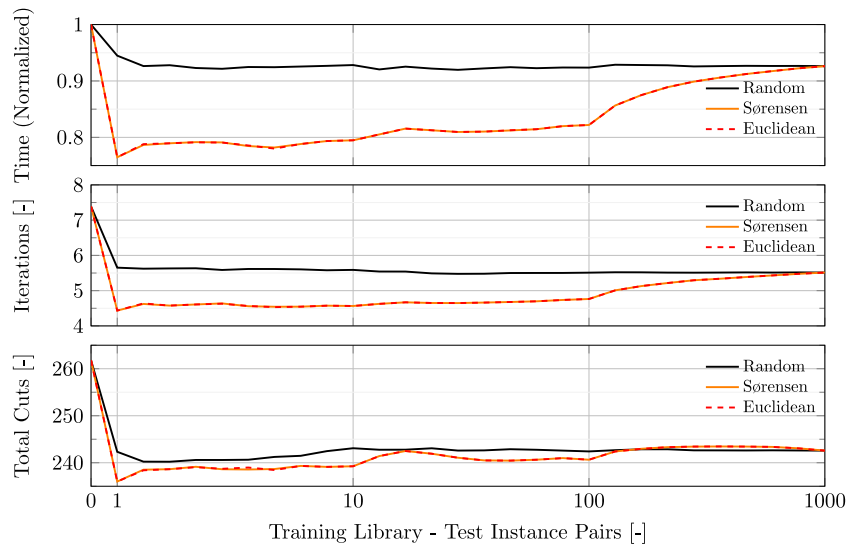


Fig. 7. Computational performance on an increasing selection of training library- test instance pairs considered. reported on test instances with pairs of high similarity measures only.

Table 4  
Computational time (Normalized) over an increasing number of libraries.

Sørensen	Number of libraries				
	1	10	100	500	1000
$1 \geq \geq 0.9998$	0.8091 (3%)	0.7484 (23%)	0.7505 (23%)	0.7379 (23%)	0.7391 (23%)
$0.9998 > \geq 0.9989$	1.1265 (73%)	1.2364 (76%)	1.2102 (76%)	1.1942 (76%)	1.2310 (77%)
$0.9989 > \geq 0.998$	1.0995 (24%)	0.6583 (1%)	1.3006 (1%)	0.9582 (1%)	-

(\*): Percentage share of test instances, with the best library in that category.

If our similarity measures are able to identify the most valuable libraries, we expect for a small set of training library - test instance pairs considered, the best performance. Increasing the amount of pairs, we expect the performance benefits to decrease to a larger average normalized computational time. This is due to the fact that the increasingly added instances have a smaller similarity and therefore result in a decreasing performance on average.

In Fig. 7 we can indeed see such effect, of sharp decrease (i.e. the most similar is actually good), and rebound (i.e. the less similar are

actually not good, going towards the right along the x-axis) for testing instances with similar libraries. In the plots describing computational time and the iterations, this phenomenon is particularly visible: the improvement in performance considering the most similar library achieves a value well below 0.8, on average. We conclude that our measures are able to indicate libraries with high computational value.

We extend the analysis of computational benefits for the reuse under real-world conditions to all testing instances in Table 4. We exclusively focus on the Sørensen–Dice measure as the euclidean has in general shown identical results. In Table 4 we report the average normalized

computational time over testing instances. We consider a reduced set of libraries (with increasing size of 1, 10, 100, 500 and 1000) from the training instances, which are randomly selected from the full set of 1000 libraries available. The idea is to understand the benefit of a larger set of libraries. We consider for each testing instance and each reduced set of libraries the performance of the most similar library in the reduced set of libraries, with respect to the Sørensen–Dice measure. In other terms, we take the single most similar out of 1, 10, 100, 500 and 1000, and we use that as library of reference. We report computational time (normalized by no reuse), further differentiated by the Sørensen–Dice measure of the most similar library (to distinguish cases where the most similar library is indeed highly similar, or less). In brackets, we report the percentage of test instances and corresponding most similar library into the three similarity classes. For the case of library with a unitary size, we repeat the test 10 times and take the average.

In Table 4 we can see that, if only one library is considered per test instance, only few test instances show high similarity to that library (i.e.,  $\geq 0.9998$ ); 73% of the instances show medium similarity and 24% of instances show a low similarity. The medium and low similarity of libraries reflects in the average computational times, which are for medium similarity and low similarity clearly  $> 1$ . When considering 10 or more libraries per testing instance, we can find (as discussed earlier) 23% of the instances with a highly similar library, and for almost all remaining test instances a library with medium similarity. The test instances with highly similar libraries show a computational speedup of  $\sim 25\%$ , empirically showing the ability of Sørensen–Dice to identify beneficial libraries.

Further, we can see in Table 4 that little computational improvements are made by increasing the set of libraries considered from 10 to 1000. We explain such behavior by the fact that Sørensen–Dice is an imperfect estimator; and seems not able to differentiate the very best, when sufficiently many libraries of similar quality are available. Moreover, the results in Table 4 can be explained by the delay in the different test instances. The 23% of instances with a highly similar library are instances with very common delays, which drastically increases the probability of a highly similar library. The remaining instances are instances with uncommon, and large, input delay, drastically reducing the probability of highly similar libraries. In that case, even 1000 training instances are insufficient to produce a highly similar library.

## 8. Conclusion

In this paper, we introduce an approach to enable a faster solution process for railway rescheduling, considering precomputation and statistical measures of similarity for the input instances. We start from adapting the logic Benders decomposition for timetabling of Leutwiler and Corman (2022) to the purpose of real-time rescheduling, where computational time is strongly limited. We propose the reuse of pre-computed logic Benders cuts to accelerate the iterative decomposition process, exploring the similarity between rescheduling problems dealing with the same railway network, traffic and timetable, and varying input delay. With a modification on precomputed Benders cuts we are able to reuse any precomputed cut, as far as the instance (i.e. trains, infrastructure, planned timetable) is the same. For the reuse of cuts, we propose a lazy-constraint approach, which only includes in the master those Benders cuts, that provide progress in the iterative decomposition process. We propose two measures that are able to identify the potential computational benefit from a library of precomputed Benders cuts, based on the similarity of the input delay of the training (which generated the library) and test instance. This allows to avoid an excessive amount of logic Benders cuts considered for reuse, and prevents large amounts of constraints being included in the master problem of the Benders decomposition.

With the lazy-constraint approach we are able to exclude useless precomputed cuts from the reuse and avoid to add those to the master problem. This prevents an exponential growth of computational time (twice as slow, when 95% additional cuts are considered) due an increasing size of the master problem, when direct reuse would be considered. The reuse of the best logic Benders cuts under a lazy-constraint scheme can find a solution twice as fast as the normal Benders decomposition without precomputation and reuse of cuts. Compared to a benchmark with a commercial solver on a centralized approach, we achieve an overall speedup up to a factor of 5.

With a further analysis, we are able to show that similarity in input delay is a promising estimator for the computational benefit of reusing cuts from a library. Both proposed similarity measures, for the considered delay statistics, are able to identify the 1% of instances which result in a speedup. The reuse of Benders cuts from libraries from instances with similar input delay achieved a comparable acceleration as in the ideal case of reuse with the perfect logic Benders cuts. This underlines the value of reusing logic Benders cuts in a real-world application.

The computational results underline the practical benefits of findings in this paper. In reality, rescheduling actions from railway dispatchers are expected between within 3 (D'Ariano et al., 2007) up to 10 (Lamorgese and Mannino, 2015) minutes. With the novel methodology of this paper, we are able to solve instances previously only solvable in over 3 min, in around 1 min of computational time; this emphasizes the practical benefits of the proposed methodology.

Future research should clearly include additional studies on identifying the potential of precomputed cuts and larger libraries. In our case, if libraries with a sufficiently high similarity exist, the speedup is perceivable. Thus, a detailed sampling of the actual delay domain (which depends on the instance and operations, in general) has to cover the possible delays with a sufficiently fine-grained detail. It is advisable for a real-world application, where more sophisticated data structures and very large libraries are acceptable, to increase the number of training instances, and to use more sophisticated sampling schemes. While we were able to propose two valuable measures for determining the similarity of instances, it remains an open question for future research, whether other features can determine the similarity of instances towards usability of Benders cuts, for a higher computational speedup.

## CRedit authorship contribution statement

**Florin Leutwiler:** Conceptualization, Methodology, Implementation, Writing. **Guillem Bonet Filella:** Methodology, Implementation. **Francesco Corman:** Conceptualization, Methodology, Writing.

## Data availability

The authors do not have permission to share data.

## Acknowledgments

The authors thank the colleagues from SBB for the useful discussions and support during the implementation. This project was supported by the ETH Zürich Foundation.

## Appendix. SMT<sub>Agg</sub> - generation and aggregation of logic Benders cuts

In this appendix we repeat details of Algorithm SMT<sub>Agg</sub> in Leutwiler and Corman (2022). We use SMT<sub>Agg</sub> in this paper to evaluate the feasibility of the subproblem  $S^\alpha$  (Problem (4)) in the decomposition of Section 4.2. In case of infeasibility, Algorithm SMT<sub>Agg</sub> creates a logic Benders cut as in Leutwiler and Corman (2022). Algorithm SMT<sub>Agg</sub> is based on Satisfiability Modulo Theories (SMT), which is a combination

**Algorithm 2:** SMT

---

```

input :  $S^\alpha$ 
output:  $\mathcal{O}^\alpha, I^\alpha, \beta^\alpha$ 
init :  $\Phi \leftarrow S^\alpha, G^\alpha \leftarrow S^\alpha, \theta = \emptyset$ 
1 while true do
2    $\text{confl} \leftarrow \text{UnitPropagation}(\Phi, \theta)$ 
3   if ! $\text{confl}$  then
4      $\text{confl} \leftarrow \text{Evaluate}(G^\alpha(\theta))$ 
5   if ! $\text{confl}$  then
6     if  $\theta = \text{complete}$  then
7        $\mathcal{O}^\alpha \leftarrow G^\alpha(\theta)$ 
8       return  $(\mathcal{O}^\alpha, \emptyset, \emptyset)$ 
9        $\theta \leftarrow \theta \cup \text{Decide}()$ 
10    else
11      if  $\text{confl} = \text{Unsatisfiable}$  then
12         $I^\alpha \leftarrow \text{AnalyzeIP}(\text{confl})$ 
13         $\beta^\alpha \leftarrow \text{BendersCut}(I^\alpha)$ 
14        return  $(\emptyset, I^\alpha, \beta^\alpha)$ 
15      else
16         $\text{Analyze}(\text{confl})$ 
17         $\text{Backtrack}(\text{confl})$ 

```

---

of Satisfiability (SAT) solving (Davis et al., 1962) and a first-order logic (De Moura et al., 2007). Furthermore, concepts of Asín et al. (2008) are used in  $\text{SMT}_{\text{Agg}}$  for the discovery of infeasibility proofs.

Algorithm  $\text{SMT}_{\text{Agg}}$  is an aggregation of logic Benders cuts, such that if  $\text{SMT}_{\text{Agg}}$  is invoked on a subproblem, the algorithm computes not a single, but multiple logic Benders cuts. By the aggregation, the total number of iterations till convergence in the Benders scheme is reduced. Inside  $\text{SMT}_{\text{Agg}}$ , Algorithm SMT (Algorithm 2) of Leutwiler and Corman (2022) is invoked multiple times, with different initial conditions to discover multiple logic Benders cuts.

In Algorithm SMT (Algorithm 2),  $\Phi$  are constraints of SAT used to model the decision and choice sets of  $S^\alpha$ .  $I^\alpha$  is an infeasibility proof for  $S^\alpha$  used to derive the logic Benders cut  $\beta^\alpha$ . Algorithm 2 proceeds by extending iteratively an initially empty selection  $\theta$  through new choice sets  $W_c$  (line 9), until the selection is complete (line 6) or an unsatisfiable constraint has been found (line 11).  $\text{Decide}$  in line 9 selects, by some SAT heuristics, new choice sets  $W_c$  to extend  $\theta$ . If  $\theta$  is a complete selection (line 6), the algorithm returns a feasible solution of the subproblem, derived by the disjunctive graph of the subproblem (line 7). After every extension of  $\theta$  in line 9, unit propagation (Marques-Silva and Sakallah, 1999) is performed (line 2) and  $\theta$  is extended by additional choice sets, that are implied through the constraints in  $\Phi$ . If after such extension of  $\theta$ , a violated constraint ( $\text{confl}$ ) is found in  $\Phi$ , such constraint is either generally unsatisfiable (line 11) and an infeasibility proof, together with a logic Benders cut can be derived (line 12 and 13); or the constraint can be satisfied by a different  $\theta$  and selection must be adjusted by first analyzing the violated constraints (line 16) and then removing appropriate choice sets from  $\theta$  (line 17).

$\text{SMT}_{\text{Agg}}$  (Algorithm 3) invokes Algorithm 2 multiple times with different master solutions imposed to the subproblem  $S^\alpha$  to generate multiple logic Benders cut in a single iteration of the Benders scheme. Algorithm 3 start by applying Algorithm 2 on the subproblem  $S^\alpha$ , which has imposed the latest master solution  $\mathcal{O}_M^\alpha$  (line 3). In case Algorithm 2 discovers an infeasibility proof and a logic Benders cut,  $S^\alpha$  is modified; the master problem is extended by the cut. This means that the next iterations will result in a solution  $\mathcal{O}_M^\alpha$  which will satisfy the latest discovered Benders cut  $\beta_i^\alpha$  (line 4). Algorithm 2 is reapplied to the modified version of the subproblem  $S'^\alpha$  to discover further Benders cuts. Eventually, Algorithm 2 in line 4 will return a feasible solution and Algorithm 3 terminates and returns all aggregated cuts.

**Algorithm 3:**  $\text{SMT}_{\text{Agg}}$ , Benders cut aggregation scheme for a subproblem.

---

```

input :  $S^\alpha$ 
output:  $\{\beta_1^\alpha, \beta_2^\alpha, \dots\}$ 
init :  $i = 1, S'^\alpha = S^\alpha$ 
1 while  $\mathcal{O}^\alpha = \emptyset$  do
2    $\mathcal{O}^\alpha, I_i^\alpha, \beta_i^\alpha \leftarrow \text{SMT}(S'^\alpha)$ 
3   if  $I_i^\alpha \neq \emptyset$  then
4      $S'^\alpha \leftarrow \text{Modify}(S'^\alpha, I_i^\alpha)$ 
5      $i \leftarrow i + 1$ 
6 return  $\{\beta_1^\alpha, \beta_2^\alpha, \dots\}$ 

```

---

**References**

- Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., 2008. Efficient generation of unsatisfiability proofs and cores in SAT. In: Lect. Notes Comput. Sci.. In: LNAI, vol. 5330, pp. 16–30.
- Balas, E., 1998. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Appl. Math.* 89 (1–3), 3–44. [http://dx.doi.org/10.1016/S0166-218X\(98\)00136-X](http://dx.doi.org/10.1016/S0166-218X(98)00136-X).
- Borndörfer, R., Klug, T., Lamogese, L., Mannino, C., Reuther, M., Schlechte, T., 2017. Recent success stories on integrated optimization of railway systems. *Transp. Res. C* 74, 196–211.
- Bretas, A., Mendes, A., Chalup, S., Jackson, M., Clement, R., Sanhueza, C., 2019. Modelling railway traffic management through multi-agent systems and reinforcement learning. In: 23rd Int. Congr. Model. Simul. - Support. Evidence-Based Decis. Mak. Role Model. Simulation, MODSIM 2019 (December). pp. 291–297.
- Cacchiani, V., Caprara, A., Galli, L., Kroon, L., Maróti, G., Toth, P., 2012. Railway rolling stock planning: Robustness against large disruptions. *Transp. Sci.* 46 (2), 217–232.
- Cacchiani, V., Caprara, A., Toth, P., 2008. A column generation approach to train timetabling on a corridor. *4or* 6 (2), 125–142.
- Cacchiani, V., Huisman, D., Kidd, M., Kroon, L., Toth, P., Veelenturf, L., Wagenaar, J., 2014. An overview of recovery models and algorithms for real-time railway rescheduling. *Transp. Res. B* 63, 15–37.
- Caimi, G., Fuchsberger, M., Laumanns, M., Lüthi, M., 2012. A model predictive control approach for discrete-time rescheduling in complex central railway station areas. *Comput. Oper. Res.* 39 (11), 2578–2593.
- Corman, F., D'Ariano, A., Pacciarelli, D., Pranzo, M., 2012. Optimal inter-area coordination of train rescheduling decisions. *Transp. Res. E* 48 (1), 71–88.
- Corman, F., D'Ariano, A., Pacciarelli, D., Pranzo, M., 2014. Dispatching and coordination in multi-area railway traffic management. *Comput. Oper. Res.* 44, 146–160.
- Corman, F., D'Ariano, A., Pranzo, M., Hansen, I.A., 2011. Effectiveness of dynamic reordering and rerouting of trains in a complicated and densely occupied station area. *Transp. Plan. Technol.* 34 (4), 341–362.
- Corman, F., Meng, L., 2014. A review of online dynamic models and algorithms for railway traffic management. *IEEE Trans. Intell. Transp. Syst.* 9 (3), 1–11.
- D'Ariano, A., Hemelrijk, R., 2006. Designing a multi-agent system for cooperative train dispatching. *IFAC Proc. Vol. 12 (PART 1)*.
- D'Ariano, A., Pacciarelli, D., Pranzo, M., 2007. A branch and bound algorithm for scheduling trains in a railway network. *European J. Oper. Res.* 183 (2), 643–657.
- Davis, M., Logemann, G., Loveland, D., 1962. A machine program for theorem-proving. *Commun. ACM* 5, 394–397.
- De Moura, L., Dutertre, B., Shankar, N., 2007. A tutorial on satisfiability modulo theories. In: Lect. Notes Comput. Sci.. In: LNCS, vol. 4590, pp. 20–36.
- Dice, L.R., 1945. Measures of the amount of ecologic association between species. *Ecology* 26 (3), 297–302.
- Dollevoet, T., Huisman, D., Kroon, L.G., Veelenturf, L.P., Wagenaar, J.C., 2017. Application of an iterative framework for real-time railway rescheduling. *Comput. Oper. Res.* 78, 203–217.
- Ghasempour, T., Heydecker, B., 2020. Adaptive railway traffic control using approximate dynamic programming. *Transp. Res. C* 113, 91–107.
- Gurobi Optimization, L., 2021. Gurobi optimizer reference manual.
- Keita, K., Pellegrini, P., Rodriguez, J., 2020. A three-step benders decomposition for the real-time Railway Traffic Management Problem. *J. Rail Transp. Plan. Manag.* 13 (July 2019), 100170.
- Lamogese, L., Mannino, C., 2015. An exact decomposition approach for the real-time train dispatching problem. *Oper. Res.* 63 (1), 48–64.
- Lamogese, L., Mannino, C., 2019. A non-compact formulation for job-shop scheduling problems in traffic management. *Oper. Res.* 67 (6), 1503–1782.
- Lamogese, L., Mannino, C., Piacentini, M., 2016. Optimal train dispatching by Benders'-Like reformulation. *Transp. Sci.* 50 (3), 910–925.
- Leutwiler, F., Corman, F., 2022. A logic-based benders decomposition for microscopic railway timetable planning. *European J. Oper. Res.* 303 (2), 525–540.

- Liu, J., Chen, L., Roberts, C., Li, Z., Wen, T., 2019. A multi-agent based approach for railway traffic management problems. In: 2018 Int. Conf. Intell. Rail Transp. ICIRT 2018. IEEE.
- Luan, X., De Schutter, B., Meng, L., Corman, F., 2020. Decomposition and distributed optimization of real-time traffic management for large-scale railway networks. *Transp. Res. B* 141, 72–97.
- Marques-Silva, J.P., Sakallah, K.A., 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Comput.* 48 (5), 506–521.
- Mascis, A., Pacciarelli, D., 2002. Job-shop scheduling with blocking and no-wait constraints. *European J. Oper. Res.* 143 (3), 498–517.
- Pellegrini, P., Marlière, G., Pesenti, R., Rodriguez, J., 2015. RECIFE-MILP: An effective MILP-based heuristic for the real-time railway traffic management problem. *IEEE Trans. Intell. Transp. Syst.* 16 (5), 2609–2619.
- Perrachon, Q., Chevrier, R., Pellegrini, P., 2020. Experimental study on the viability of decentralized railway traffic management. *WIT Trans. Built Environ.* 199, 337–344.
- Samà, M., D'Ariano, A., Corman, F., Pacciarelli, D., 2017. A variable neighbourhood search for fast train scheduling and routing during disturbed railway traffic situations. *Comput. Oper. Res.* 78, 480–499.
- Van Thielen, S., Corman, F., Vansteenwegen, P., 2018. Considering a dynamic impact zone for real-time railway traffic management. *Transp. Res. Part B Methodol.* 111, 39–59.
- Veelenturf, L.P., Kidd, M.P., Cacchiani, V., Kroon, L.G., Toth, P., 2016. A railway timetable rescheduling approach for handling large-scale disruptions. *Transp. Sci.* 50 (3), 841–862.