Real Time Large Railway Network Re-Scheduling

Erik Nygren[*], Christian Eichenberger[†], Emma Frejinger[‡]

September 9, 2020

Contents

[*] erik.nygren@sbb.ch
[†] christian.markus.eichenberger@sbb.ch
[‡] emma.frejinger@unmontreal.ca

Abstract

In this paper, we describe our novel approach to tackle the Real-Time Re-Scheduling Problem for Large Railway Networks by a combination of techniques from operations research and machine learning.

The Industry State of the Art is limited to resolve re-scheduling conflicts fully automatically only at narrowly defined geographical regions due to the exponential growth of computational time for combinatorial problems. Our aim is to improve the scalability of the optimization to larger regions by fundamentally reformulating the core problem description. In our approach, we aim to use the generalization capabilities of machine learning algorithms to reformulate any new problem instance to a smaller equivalent instance, which can be solved in a much shorter time span.

This allows us to keep the rigor of the OR formulation while relying on compressed knowledge in a machine learning algorithms to vastly increasing the size of solvable problem instances. Early investigations support this assumption by showing, that large problem instances can be reduced to much smaller core problems even with the use of domain specific heuristics.

We believe that our approach is even applicable to other transportation and logistic systems outside of railway networks.

The goal of this paper is four-fold:

G1 report the problem scope reduction formulation in a formal way;

G0 motivate the approach empirically;

G2 show the validity of the approach for one specific OR solver;

G3 report our first steps in tackling automated problem scope reduction;

G4 provide an extensible playground open-source implementation `https://github.com/SchweizerischeBundesbahnen/RSP` for further research into automated problem scope reduction.

We hope that this will draw the attention of both academic and industrial researchers to find other and better approaches and collaboration across Railway companies and from different research traditions.

1 Context and Goals

1.1 Real-world Context

Switzerland has one of the worlds most dense railway network with both freight and passenger trains running on the same infrastructure. More than 1.2 million people use trains on a daily basis [1]. Besides the publicly available railway schedule, there is also the more detailed operational schedule, which maps specific trains to planned train runs and specifies what railway infrastructure will be utilised by what train at any time.

The operational schedule has to be continually re-computed due to many smaller and larger delays and disturbances that arise during operations. While many of the minor incidents (up to 1 million per day) can be fully resolved by the

rail control system, some larger disturbances require human or more advanced algorithmic interception. To limit the spread of disturbances and minimize the delay within the dynamic railway system, decisions on re-ordering or re-routing trains have to be taken to derive a new feasible operational plan. The industry state of the art systems efficiently re-compute a traffic forecast for the next two hours without resolving the conflicts. The predicted conflicts, mostly have to be resolved by humans by explicitly deciding on re-ordering or re-routing based on their experience.The massive combinatorial complexity of these microscopic re-scheduling problems, currenlty limits the application of Operations Research models to very restricted geographic areas with a limited number of trains in the given forecast horizon.

There is a common view of the current situation within system boundaries; the whole system is decomposed into disjoint geographic cells of responsibility as depicted in Figure 1. Most of them are handled by humans: human dispatchers have a view of the full system and can communicate through structured (e.g. IT system of incident messages) or informal ways (e.g. phone call with station managers or locomotive drivers).

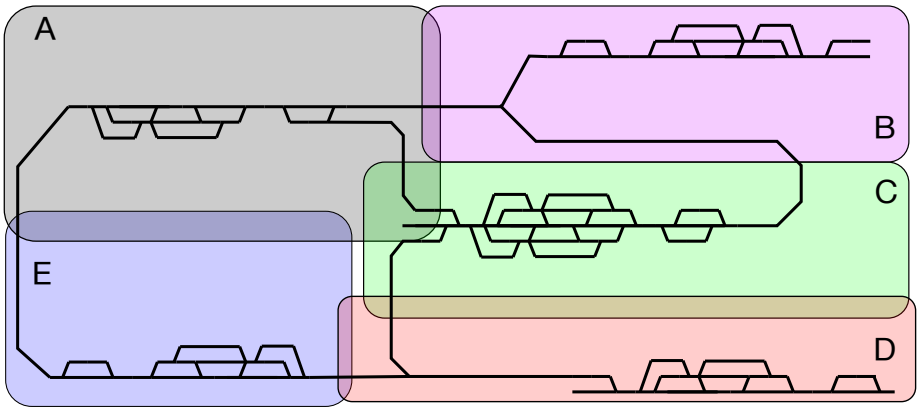

Figure 1: *Due to the vast complexity of a railway network the re-scheduling task is decomposed into smaller geographical areas (A-E), withiwin which human dispatchers optimize traffic flow. Inter-regeion coordination is mostly done by informal means of communication such as telephone.*

This situation is depicted in a schematic way in Figure 2:

> [...], a network separation approach is applied to divide the railway network into zones of manageable size by taking account of the network properties, distinguishing condensation and compensation zones. Condensation zones are usually situated near main stations, where the track topology is complex and many different routes exist. As such an area is expected to have a high traffic density, it is

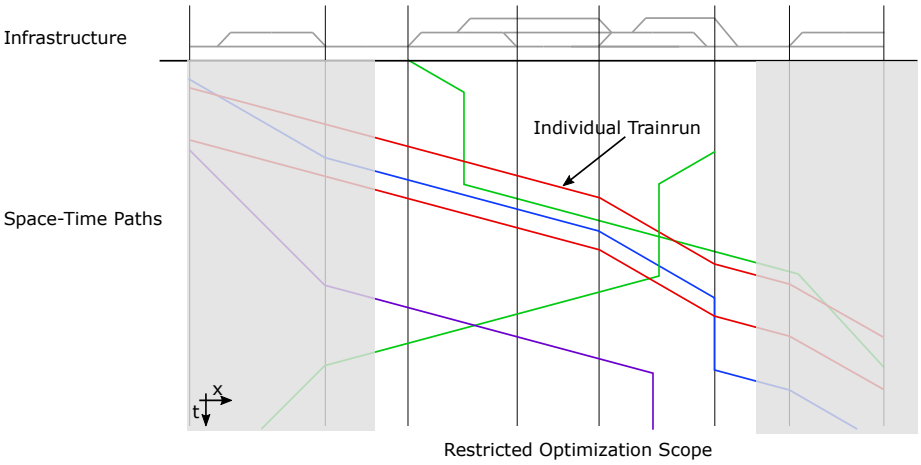

Figure 2: *Real-time rescheduling has so far been applied to restricted geographic areas only. This means that human intervention or buffer times are necessary outside of the optimization scope to guarantee conflict free traffic in the whole network. The limited geographical scope leads to a cost-function that can only locally be minimized and might have negative effects on the whole network scale.*

> also a capacity bottleneck and trains are required to travel through with maximum allowed speed and thus without time reserves. Collisions are avoided by exploiting the various routing possibilities in the station area. Conversely, a compensation zone connects two or more condensation zones and consists of a simpler topology and less traffic density. Here, time reserves should be introduced to improve timetable stability. The choice of an appropriate speed profile is the most important degree of freedom to exploit in these compensation zones. [2]

There are only a few so called condensation areas [2] (in bottleneck areas such as merging/approach areas in front of large stations or tunnels) that are operated through automatic systems. Between these dense areas, trains need to be able to compensate for temporal delays: if trains are reordered or rerouted, these decisions must be feasible in the neighboring areas.

The following Figure 3 shows the re-scheduling control loop adapted from [1, 3], as well as our simplified control loop for RSP.

Our playground implementation is a simplification (see Section 1.4 below), shown in Figure 3: we only consider an operational plan and a single delay and try to re-schedule such that we have a conflict-free plan, again that stays close to the previous operational plan. We hint at a fully automatic re-scheduling loop as follows: information from the current situation needs to be included into the operational plan by passing it to the optimizer via the Oracle; as time

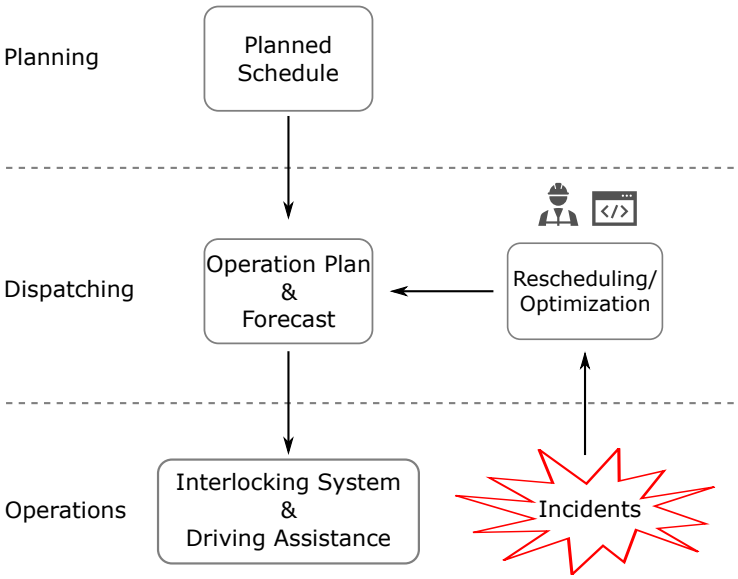

Figure 3: *Simplified schema of the rescheduling process during operations. There are three main areas involved:* planning, dispatching *and* operation *(interlocking and driving assistance). Dispatchers have access to the current situation and an automated forecast and make decisions on the level of the operational plan, which are then translated either automatically or by dedicated dispatching team members to the interlocking system; some areas are under the control of an automatic optimization system. As time goes on, new elements from the planned schedule will be introduced to the operational schedule and the planned schedule gives also constraints to re-scheduling: passenger trains must not depart earlier than communicated to customers and the service intention may define further hard or soft constraints such as pecuniar penalties for delay or train dropping. The current situation will be reflected in the operational schedule and the forecast*

moves on, new trains from the planned schedule need to be included into the operational schedule.

In reality, a distinction is made between

- operational schedule containing train routes and train ordering
- forecast containing passing times within a time horizon

The forecast outside the optimization areas needs only be conflict-free for the imminent decisions; conflicts in the future are continually resolved by human dispatchers.

As these condensation areas are supposed to grow, we think that the locality of decision-making might become a problem and some sort of coordination might be needed. Therefore, in our RSP approach, we consider how not to take space-local decisions; also we think of operational schedule and forecast as the same, being conflict-free.

1.2 Research Approach: Core idea

1.3 Research Approach: Decomposition in Space and Time

Physical railway infrastructure is expensive in building and maintenance [4]. Therefore, the existing infrastructure capacity should be exploited as best as possible. As the number of trains operating increases and condensation areas increase in number and size, it will become increasingly difficult to compensate for decisions taken within condensation zones or to define restrictions that keep the effects on the neighboring compensation zones as predictable as possible.

We will argue that

H2 it is possible to predict the affected time-space, either from the problem structure or from historic data (see below, Section 1.4.2); by this, we would over-achieve goal **G3**.

H1 such a prediction allows for a speed-up of the OR model (see below, Section 1.4.1); by this, we achieve goal **G2**.

To tackle this problem, our approach is to combine Operations Research and Domain-specific Learning to get the best of both worlds: an "Oracle" is able to predict the "impact" of a delay, with or without a knowledge base learnt by training; we hope the Oracle could predict which trains and which departures are or could be affected by the delay based on past decisions. This piece of information from the Oracle then helps the solver to constrain the search space or at least drive its search more efficiently (driving the branching process).

This approach is shown in Figure 4: the Oracle predicts restrictions in time and space, which are passed to the solver.

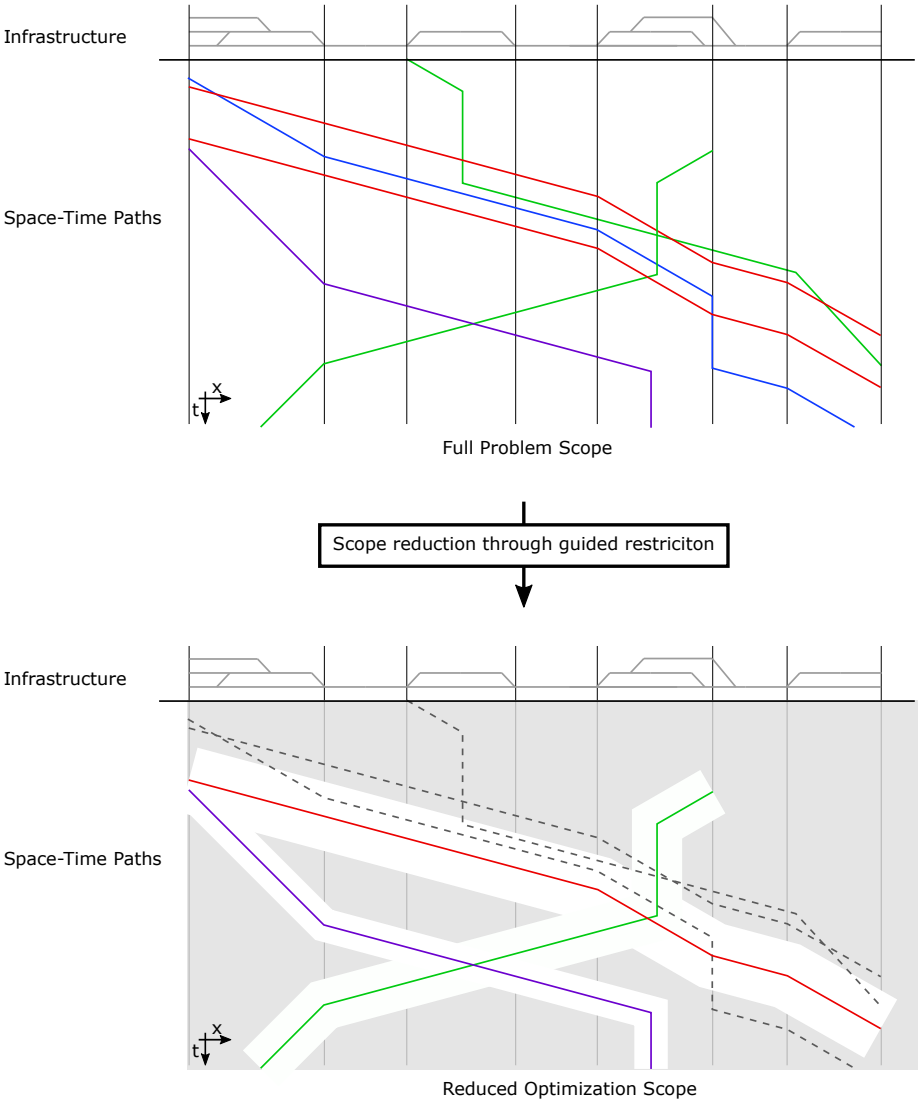

Figure 4: The Oracle gives degrees of freedom in time and space.

1.4 Research Approach: a Synthetic Playground

We now give a short introduction to our playground implementation (G4) and its limitation with respect to real-world features.

Synthetic Infrastructure and Simplified Resource Model We use the FLATland toolbox [5] to generate a grid world infrastructure consisting of 2D square cells; the infrastructure defines the possible movements to the 4 neighbor cells mirroring railway infrastructure (switches etc.)

Synthetic Timetable and Train Dynamics

- Every train has one single source and target, no intermediate stops, as provided by the FLATland toolbox [5].
- Every train has a constant speed (which may be different from train to train), as provided by the FLATland toolbox [5].
- The schedule (departure and passing times at all cells) is generated such that the sum of travel times of all trains is minimized within a chosen upper bound; this the generated timetable might not have a realistic structure.
- There are no time reserves in the schedule, trains cannot catch up.
- There is no distinction between published timetable and operational schedule, we only have an operational timetable; in reality, trains must not depart earlier than published.
- There are no connections or vehicle tours (turnrounds).

Synthetic route alternatives We use shortest paths (in reality, in particular in the case of disturbances affecting a whole area, we might need a different scheme knowing the parts that cannot be taken); path cycles are not allowed.

Simple Disturbance Model We simulate one train a being stopped for d discrete time steps at some time t and call this a malfunction $M = (t, d, a)$; in reality, the delay might not be known or only probabilities can be assumed. In reality, update information also comes in batches and we would need to consider multiple delays in the same update interval. This setup is shown in Figure 5.

1.4.1 Hypothesis H1

Hypothesis 1 is a sanity hypothesis: if we do not have a big speed-up with a perfect oracle, the whole approach must be dismissed. If no speed-up can be found we want to identify the reason and document this to gain further insight and adjust our research aim.

Consider Figure 6: The initial schedule $S0$ and a malfunction M are passed to an OR solver which is given unbounded time to solve the model to optimality; the resulting re-schedule is S.

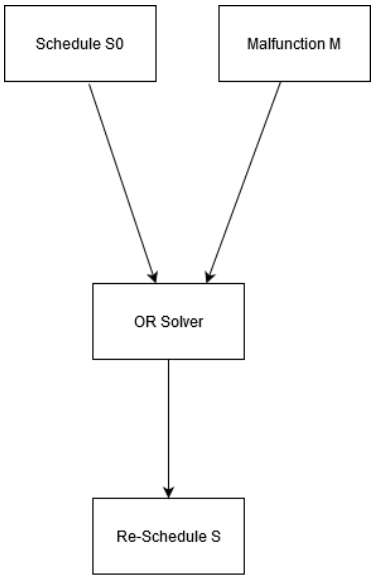

Figure 5: Simple non-iterative re-scheduling with single delay.

If we compare the initial schedule $S0$ and the re-schedule S, we can take the differences ($Delta\_perfect$):

- every time difference at the same node opens up timing flexibility: the nodes are kept fixed, but time is flexible
- every node difference opens up routing flexibility: both nodes and times are kept flexible

We will describe this in more detail below in Section 2.2.3.

Now we want to see whether the OR solver f, given the perfect information coming through the red arrows, allows for a speed-up, i.e. whether the time t_S' needed for the solver with the restriction is much smaller than the time t_S for the full problem without the restriction

The rationale of this hypothesis is non-general and asymmetric:

- *non-general*: if we show the speed-up for one particular OR solver, this does not mean that the information can be exploited by every other OR solver for speed-up. However, we conjecture that general setup may be applicable to other OR solvers and models, where the exact content of the Oracle's "hint" passed to the solver might differ. We might strengthen this conjecture by comparing with
- *asymmetric*: If we cannot show the speed-up for one particular solver, this does not exclude that the approach might work with a different OR solver and its particular shape of "perfect information". We might need to consider a different OR solver for our exposition.

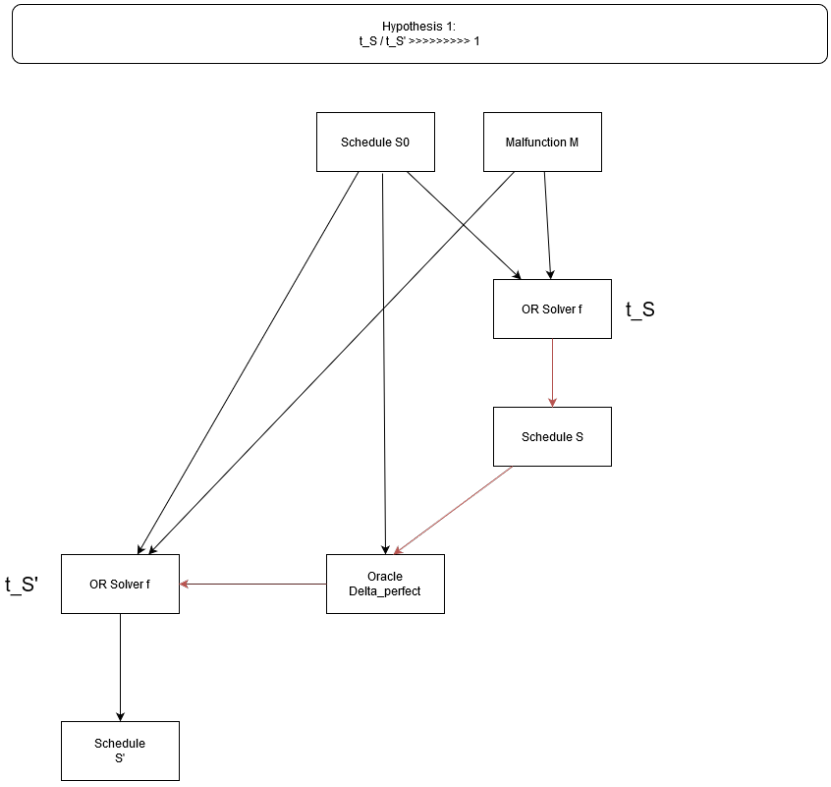

Figure 6: Rationale of hypothesis H1.

1.4.2 Hypothesis H2

Hypothesis 2 asks whether we can build an oracle that provides a considerable speed-up without access to perfect information, only from the current situation.

Consider Figure 7: Our Oracle *Omega_realistic* now has only access to

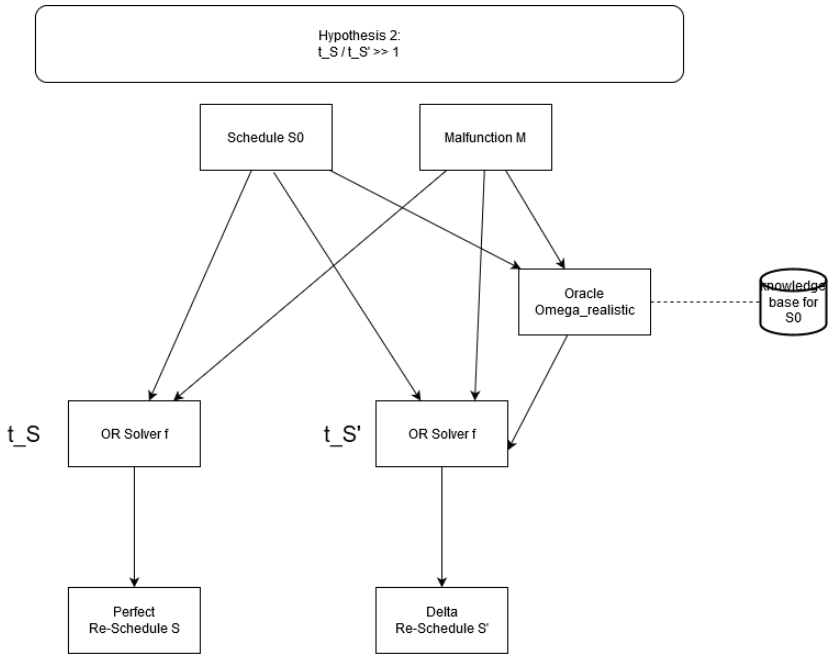

Figure 7: Rationale of hypothesis H2.

the current information and possibly its knowledge base, not to the perfect re-schedule for the current situation. Can it provide hints to the OR solver that allows for a speed-up $t\_S/t\_S' >> 1$?

We will not build such an *Omega_realistic*, but report on some first ideas which illustrate the limitation of a geographic decomposition in our synthetic setting in Section 3. We hope this motivates researchers to invest in this task.

2 The Pipeline for H1

2.1 Overview

We now give a more detailed view of the pipeline for hypothesis H1 with respect to Section 1.4.1. We refer to Figure 8: the pipeline decomposes into four top-level stages:

Experiment Planning This stage produces experiment parameters in a grid-like search: it takes a set of *parameter ranges* $(l_1, u_1, n_1), (l_2, u_2, n_2) \ldots$ and

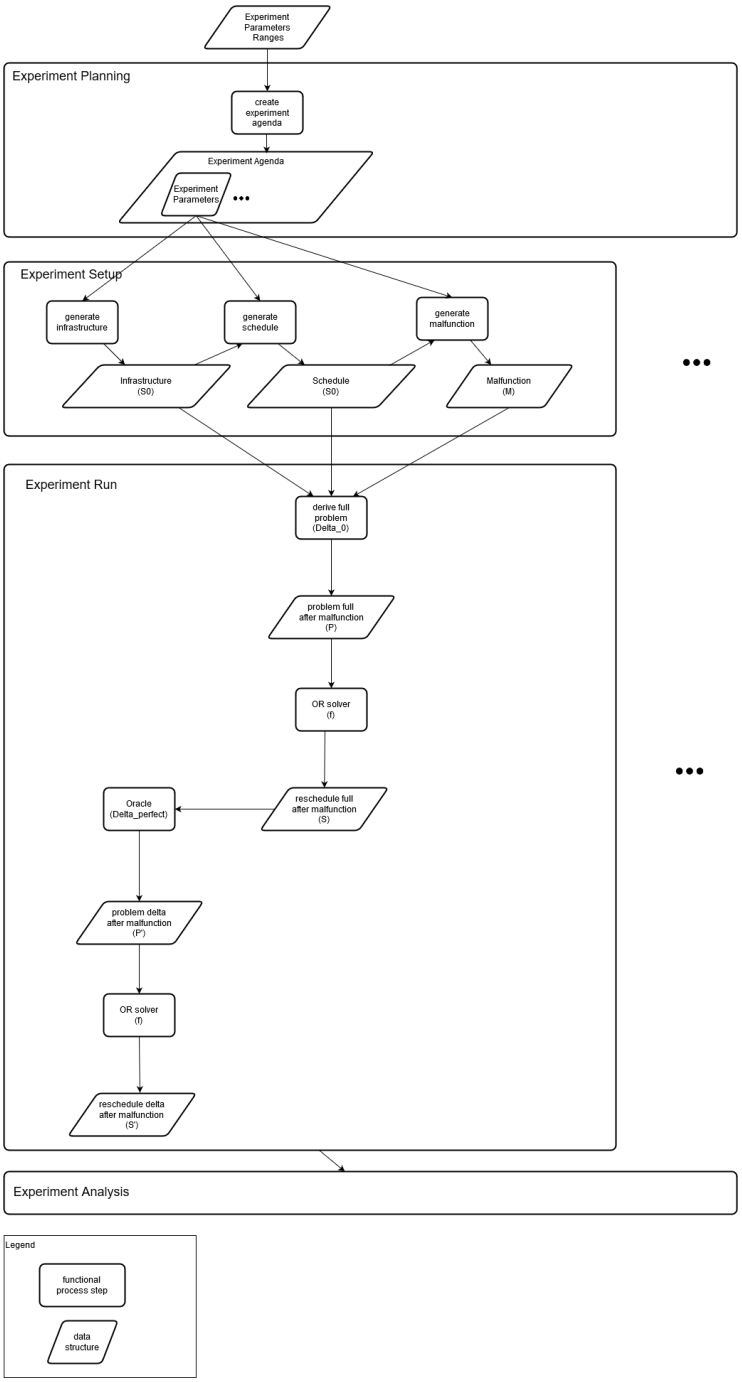

Figure 8: Pipeline for hypothesis H1 as functional diagram with intermediate data structures at two levels. The top level consists of four stages.

produces the Cartesian product $R_1 \times R_2 \times \ldots$ of the expanded parameter ranges $R_i = \bigcup_{\ell=0,\ldots,n_i-1} \left\{ l_i + \ell \cdot \frac{u_i - l_i}{n_i} \right\}$ of size $n_1 \cdot n_2 \cdot \ldots$ (we assume $u_i - l_i$ to be a multiple of n_i). We will call this roll-out $R_1 \times R_2 \times \cdots$ an *experiment agenda* and its elements $(p_1, p_2, \ldots) \in R_1 \times R_2 \times \cdots$ *experiment parameters*.

Experiment Setup consists of *infrastructure generation* for the given experiment parameters, *schedule generation* and *malfunction generation*. We will cover this stage in detail in Sections 2.2 and 2.2.1, respectively.

Experiment Run corresponds to the content of Figure 6. We will cover these data structures and functional process steps in Sections 2.2.2 and 2.2.3.

Experiment Analysis This stage produces the plots that help to verify hypothesis H1. We will give more details in Section 5.

2.2 Infrastructure generation

We now describe our railway infrastructure and how it is generated, mimicking a natural setup.

The infrastructure consists of a grid of cells. Each cell consists of a distinct tile type which defines the movement possibilities of the agent through the cell. There are 8 basic tile types, which describe a rail network. As a general fact in railway network when on navigation choice must be taken at maximum two options are available. Figure 9 (top) gives an overview of the eight basic types. These can be rotated in steps of 45° and mirrored along the North-South of East-West axis. The bottom row shows the corresponding translation into a graph structure: squares represent cells, black dots represent an entry to pin to the depicted cell and a grey dot represents the pin in the opposite direction into the neighboring cell. Formally, the *railway infrastructure* is a tuple $(\mathcal{C}, \mathcal{V}, \mathfrak{c}, \mathcal{E})$,

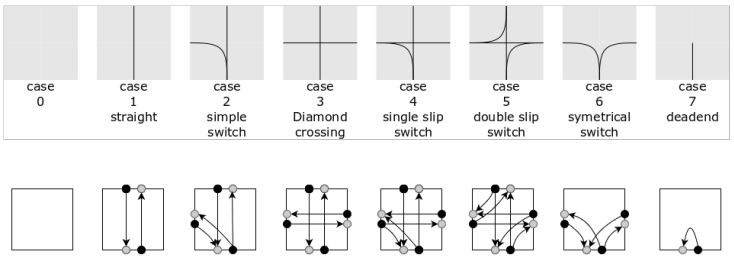

Figure 9: Eight basic cell types of a FLATland grid (top) and their translation into a directed graph (bottom).

where

- $\mathcal{C}$ is a set of cells,

- $\mathcal{V}$ is the set of pins by which the cell can be entered (they are positioned to the north, east, south or east of the cell),
- $\mathfrak{c} : \mathcal{V} \to \mathcal{C}$ which associates to each "pin" the cell it enters (there are at most 4 pins for every cell),
- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, the possible directed transitions in the grid;

furthermore, we require $(\mathcal{V}, \mathcal{E})$ to be an acyclic graph.

TODO Erik describe infrastructure generation: placement of stations and lines (FLATland)

2.2.1 Schedule and Malfunction Generation

A *railway service intention* consists of set $\mathcal{A}$ of trains or agents; each train a has a source $\sigma(a) \in \mathcal{V}$ and a some target in $\tau(a) \subseteq \mathcal{V}$ and a speed $v(a) \in [0, 1]$ and a release time r, which specifies how long a cell remains blocked after a train has left it. A *schedule* is a pair (P, A) consisting of

1. a function P assigning to each train line the path it takes through the network, and
2. an assignment A of arrival times to each train line at each node on their path.

A path is a sequence of nodes, pair-wise connected by edges. We write $v \in p$ and $(v, v') \in p$ to denote that node v or edge (v, v') are contained in path $p = (v_1, ..., v_n)$, that is, whenever $v = v_i$ for some $1 \leq i \leq n$, or this and additionally $v' = v_{i+1}$, respectively. A path $P(a) = (v_1, ..., v_n)$ for $a = (S, L, e, l, w) \in \mathcal{A}$ has to satisfy

$$v_i \in S \text{ for } 1 \leq i \leq n, \tag{1}$$

$$(v_j, v_{j+1}) \in L \text{ for } 1 \leq j \leq n - 1 \tag{2}$$

$$in(v_1) = 0 \text{ and } out(v_n) = 0, \tag{3}$$

where *in* and *out* give the in- and out-degree of a node in graph (S, L), respectively. Intuitively, conditions (1) and (2) enforce paths to be connected and feasible for the train line in question and Condition (3) ensures that each path is between a possible start and end node. An assignment A is a partial function $\mathcal{A} \times V \to \mathbf{N}$, where $A(a, v)$ is undefined whenever$v \notin P(a)$. Now we require schedules to satisfy

speed each agent has a constant *minimum running time*, we require

$$A(a, v') - A(a, v) \geq v(a)^{-1} \tag{4}$$

for all trains $a \in \mathcal{A}$ and all traversed edges $(v, v')inP(a)$;

resource exclusion cells must not be used concurrently and not before the release time after the leaving the cell has elapsed, i.e.

$$A(a, v') + r \leq A(a', u) \text{ or } A(a', u') + r \leq A(a, v) \tag{5}$$

for all $a, a' \subseteq \mathcal{A}$, $a \neq a'$,$(v, v') \in P(a)$,$(u, u') \in P(a')$ with $r(v) = r(u)$. Notice that the exit time corresponds to the entry time in the next cell/edge and should not be confused with the release time of the cell, which comes r time steps after the exit time.

With these definitions, we are now equipped to describe how we generate service intentions and schedules.

TODO Erik describe railway service intention generation (placement of agents in FLATland)

For schedule generation, we use the OR solver model from Section 2.2.2 with a different objective function, namely minimizing

$$\sum_{a \in \mathcal{A}, v, v' \in P(a), in(v)=0, out(v)=0} A(a, v') - A(a, v) \tag{6}$$

and restricting

$$\max_{a \in \mathcal{A}, v \in P(a)} A(a, v) \leq U, \tag{7}$$

where we use the following heuristic for the upper bound,

$$U = \delta \cdot \left(w + h + \frac{|\mathcal{A}|}{|S|} \right) \tag{8}$$

and $\delta = 8$, to find a schedule S for a given service intention. This objective function is intended to mimick a green wave behaviour of real-world train scheduling (schedules are constructed such that there are only planned stops for passenger boarding and alighting); however, this also keeps us from introducing time reserves in the schedule and trains will not be able to catch up in our synthetic world.

Our experiment parameters contain the following parameters for malfunction generation:

- $m_{earliest} \in \mathbb{N}$: the earliest time step the malfunction can happen;
- $m_{duration} \in \mathbb{N}$: how much the train will be delayed;
- $m_{agent} \in \mathcal{A}$: the train that is concerned.

Given the schedule, we derive our malfunction $M = (m_{time\_step}, m_{duration}, m_{agent})$ where

$$m_{time\_step} = \min \left\{ \min_{v \in P(m_{agent})} A(m_{agent}, v) + m_{earliest}, \max_{v \in P(m_{agent}, v)} A(m_{agent}, v) \right\},$$

which ensures that the agent malfunction happens while the train is running.

2.2.2 General Train Scheduling Problem

We will introduce the abstract model from [6], which is more gerneral than we actually need for our pipeline for H1; this will allow us to review the synthetic assumptions of Section 1.4 in a more formal setting. We already mentioned in the previous Section 2.2.1 that we use this more general model with a dedicated objective to generate the input schedule for our pipeline; in the next Section 2.2.3, we will show how this solver model can be used for re-scheduling.

According to [6], the *(general) train scheduling problem* is formalized as a tuple $(N, \mathcal{A}, C, F)$ having the following components

- N stands for the railway network (V, E, R, m, a, b), where
  - (V, E) is a directed graph,
  - R is a set of resources,
  - $m : E \to \mathbb{N}$ assigns the minimum travel time of an edge,
  - $a : \mathbb{R} \to 2^E$ associates resources with edges in the railway network, and
  - $b : R \to \mathbb{N}$ gives the time a resource is blocked after it was accessed by a train line.
- $\mathcal{A}$ is a set of train lines to be scheduled on network N. Each train in $\mathcal{A}$ is represented as a tuple (S, L, e, l, w), where
  - (S, L) is an acyclic subgraph of (V, E),
  - $e : S \to \mathbb{N}$ gives the earliest time a train may arrive at a node,
  - $l : S \to \mathbb{N} \cup \{\infty\}$ gives the latest time a train may arrive at a node, and
  - $w : L \to \mathbb{N}$ is the time a train has to wait on an edge.

  Note that all functions are total unless specified otherwise.
- C contains connections requiring that a certain train line a' must not arrive at node n' before another train line a has arrived at node n for at least α and at most ω discrete time steps. More precisely, each connection in C is of form $(t, (v, v'), t', (u, u'), \alpha, \omega, n, n')$ such that $a = (S, L, e, l, w) \in \mathcal{A}$ and $a' = (S', L', e', l', w') \in \mathcal{A}$, $a \neq a'$,$(v, v') \in L$,$(u, u') \in L'$,$\{\alpha, \omega\} \subseteq \mathbb{Z} \cup \{\infty, -\infty\}$, and either $n = v$ or $n = v'$, as well as, either $n' = u$ or $n' = u'$.
- Finally, F contains collision-free resource points for each connection in C. We represent it as a family $(F_c)_{c \in C}$. Connections removing collision detection are used to model splitting (or merging) of trains, as well as reusing the whole physical train between two train lines. More importantly, this allows us to alleviate the restriction that subgraphs for train lines are acyclic, as we can use two train lines forming a cycle that are connected via such connections. Refer to [6] for more details.

In this setting, edges can be interpreted as geographic sections with a speed profile; however, the model has no reference to the underlying geography (coordinates etc.); also, resources have no location – they can be interpreted as track sections that need to be reserved, but they may also be gates or sideway tracks that need to be reserved while travelling the edge.

We now show how our simple scheduling model of Section 2.2.1 can be embedded into this more general framework. Let $(\mathcal{A}, \sigma, \tau, v, r)$ be a service intention in a railway infrastructure $(\mathcal{C}, \mathcal{V}, \mathfrak{c}, \mathcal{E})$. Then, $(N, \tilde{\mathcal{A}}, C, F)$ with

- $\tilde{\mathcal{A}}$ consists of a tuple (S, L, e, l, w) for each $a \in \mathcal{A}$ where Let
  - $S = \{v : v \in P_a\}$
  - $L = \{(v_1, v_2) : (v_1, v_2) \in P_a\}$
  - $e(v) = \min_{p:\sigma(a)\text{–}v \text{ path}} |p| \cdot v(a)^{-1}$
  - $l(v) = \max_{p:v\text{–}\tau(a) \text{ path}} U - |p| \cdot v(a)^{-1}$
  - $w(e) = v(a)^{-1}$ [1].

  for a set P_a of $\sigma(a)$–$\tau(a)$ paths in $\mathcal{E}$.
- $N = (V, E, R, m, a, b)$ where
  - $V = \bigcup_{a\in\mathcal{A}} V_a$
  - $E = \bigcup_{a\in\mathcal{A}} E_a$
  - $R = \mathcal{C}$
  - $m((v_1, v_2, a)) = 0$
  - $a(c) = \{e = (v_1, v_2, a) : \mathfrak{c}(v_1) = c\}$
  - $b(c) = r$
- $C = \emptyset$
- $F = \emptyset$

is a train scheduling problem. Some remarks on this transformation:

- We only have one resource per edge, i.e. we only reserve the train's own track.
- The earliest and latest windows are designed such that they represent the earliest possible time the train can reach the vertex, respectively, the latest possible time the train must pass in order to be able to reach the target within the time limit.

[1] This does not respect the intended semantics of the general model. Therefore, it would be better to use $S = \{(v, a) : v \in P_a\}$, $L = \{((v_1, a), (v_2, a)) : (v_1, v_2) \in P_a\}$, $w(e) = 0$ and $m(((v_1, a), (v_2, a))) = v(a)^{-1}$.

The time windows given by e and l can be computed by Algorithm 1 and 2, respectively. They propagate the minimum running forward from an initial set of earliest and backwards from an initial set of latest constraints. An illustration can be found in Figure 10 and 11, respectively. The time windows are thus given by

$$e \leftarrow \mathit{propagate\_earliest}(\{\, e(\sigma(a)) = 0 \,\}, (S, L), \{\, \sigma(a) \,\}, v(a)^{-1})$$

and

$$l \leftarrow \mathit{propagate\_latest}(\{\, e(\tau) = U : \tau \in S, \mathit{out}(\tau) = 0 \,\}, (S, L), \{\, \tau \in S : \mathit{out}(\tau) = 0 \,\}, v(a)^{-1}).$$

Finally, we can remove nodes v with empty time window $e(v) > l(v)$.

Algorithm 1 *propagate_earliest*

Input: e, (S, L), F, mrt s.t. $F \subseteq \mathrm{dom}(e)$
Output: e

```
1:  Open ← F
2:  for v ∈ Open do
3:      for v' ∈ S − F : (v, v') ∈ L do
4:          if v' ∉ dom(e) then
5:              e(v') ← ∞
6:          end if
7:          e(v) ← min{ e(v'), e(v) + mrt }
8:          Open ← Open ∪ { v' }
9:      end for
10:     Open ← Open − { v }
11: end for
```

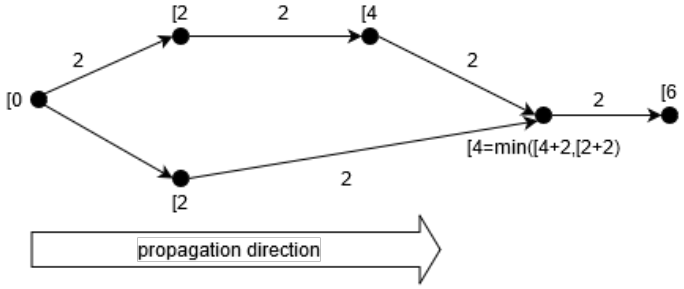

Figure 10: Illustration of *propagate_earliest* for an agent a with speed $v(a) = \frac{1}{2}$ (i.e. $mrt = 2$).

After this discussion of how our simple instances are translated into the more general train scheduling problem, we define solutions: As above, the *solution to a train scheduling problem* $(N, \mathcal{A}, C, F)$ is a pair (P, A) satisfying conditions

Algorithm 2 *propagate_latest*

Input: l, (S, L), F, mrt s.t. $F \subseteq \mathrm{dom}(l)$
Output: l

```
1: Open ← F
2: for v ∈ Open do
3:     for v' ∈ S − F : (v', v) ∈ L do
4:         if v' ∉ dom(l) then
5:             l(v') ← −∞
6:         end if
7:         l(v) ← max{ l(v'), l(v) − mrt }
8:         Open ← Open ∪ { v' }
9:     end for
10:    Open ← Open − { v }
11: end for
```

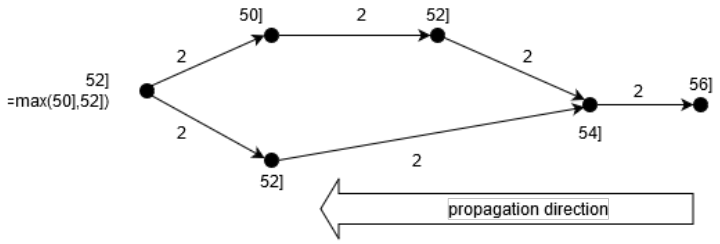

Figure 11: Illustration of *propagate_latest* for an agent a with speed $v(a) = \frac{1}{2}$ (i.e. $mrt = 2$) and for upper bound $U = 56$.

(1)–(3). In addition, given path function P, an assignment A has to satisfy the conditions in (9) to (13):

$$A(a, v_i) \geq e(v_i) \tag{9}$$

$$A(a, v_i) \leq l(v_i) \tag{10}$$

$$A(a, v_j) + m((v_j, v_{j+1})) + w((v_j, v_{j+1})) \leq A(a, v_{j+1}) \tag{11}$$

for all $a = (S, L, e, l, w) \in \mathcal{A}$ and $P(a) = (v_1, ..., v_n)$ such that $1 \leq i \leq n$,$1 \leq j \leq n-1$, either

$$A(a, v') + b(r) \leq A(a', u) \text{ or } A(a', u') + b(r) \leq A(a, v) \tag{12}$$

for all $r \in R$, $a, a' \subseteq \mathcal{A}$, $a \neq a'$,$(v, v') \in P(a)$,$(u, u') \in P(a')$ with $\{(v, v'), (u, u')\} \subseteq a(r)$ whenever for all $(a, (x, x'), a', (y, y'), \alpha, \omega, n, n') \in C$ such that $(x, x') \in P(a)$,$(y, y') \in P(a')$, we have $(a, (v, v'), a', (u, u'), r) \notin F_c$, and finally

$$\alpha \leq A(t', n') - A(t, n) \leq \omega \tag{13}$$

for all $(a, (v, v'), a', (u, u'), \alpha, \omega, n, n') \in C$ if $(v, v') \in P(a)$ and $(u, u') \in P(a')$. Intuitively, conditions (9), (10) and (11) ensure that a train line arrives at nodes neither too early nor too late and that waiting and traveling times are accounted for. Furthermore, Condition (12) resolves conflicts between two train lines that travel edges sharing a resource, so that one train line can only enter after another has left for a specified time span. This condition does not have to hold if the two trains use a connection that defines a collision-free resource point for the given edges and resource. Finally, Condition (13) ensures that train line a connects to a'at node n and n', respectively, within a time interval from α to ω. Note that this is only required if both train lines use the specific edges specified in the connections. Furthermore, note that it is feasible that n and n' are visited but no connection is required since one or both train lines took alternative routes.

It is easy to see that any solution to the (simplified) *railway service intention* is also a solution to the general train scheduling problem after transformation, hence the given transformation is indeed an embedding, carrying solutions to solutions.

TODO Christian should we introduce removal of edges not reached in time here?
TODO Christian should we introduce notation for sources $sources(V, E) = Set * v \in V : in(v) = 0$ and sinks $sinks(V, E) = Set * v \in V : out(v) = 0$?
TODO Christian make footnote for generalizing to multiple sources.

2.2.3 Re-scheduling Full and Delta

Re-scheduling Full
We first describe full the re-scheduling problem as a general train scheduling

problem. Let $N = (V, E, R, m, a, b)$ be the network of train scheduling problem, (P, A) be a solution to it and let $M = (m_{time\_step}, m_{duration}, m_{agent})$ be a malfunction.

Informally, the *re-scheduling problem* is the train scheduling problem such that

- all decisions up to and including m_{time_step} are fixed from the solution. If the train is on an edge at m_{time_step}, then it has to use the same edge, reflecting that a train on a straight segment has to continue and the switch position cannot be changed once the train is on the switch;
- the train corresponding to m_{agent} is delayed by $m_{duration}$.

We here have two options:

1. we can constrain the original scheduling up to the malfunction
2. we can remove everything up to the malfunction and constrain only the last decision before the malfunction

In order to keep the exposition simple and since this is also in the spirit of a re-scheduling loop with moving time horizon, we adopt the second option in the following exposition.[2]

We now describe the transformations Δ_0 for a single train in Algorithm 3: If the train is already done at the malfunction time step, it can be removed from the problem; if the train has not started yet at the malfunction time step, we keep its start node fixed from $S0$ and do not start earlier than in $S0$ and use *propagate* (described below) to derive the constraints.

TODO Christian in case 2 (not started yet), we should respect the malfunction delay? Problem here and in source code!

Before describing the third case of the malfunction happening while the train is running, we describe *propagate* (Algorithm 4): We first propagate earliest and latest, then truncate time windows to c; we need to propagate latest again since truncation might spoil the semantics of $l(v)$ being the latest possible passing time to reach the target in time. Finally, we remove nodes that are not reachable in time (because of an empty time window) or cannot be reached from one of the nodes F_v that must be visited.

We now describe the remaining case of Algorithm 3, namely when the malfunction happens while the train is running. We refer to Algorithm 5: we first determine the edge (v_1, v_2) the train is on when the malfunction happens; we then fix the time for v_1 as in $S0$ and set the earliest for v_2, possibly delayed. Then, we use *propagate* to tighten the search space. Notice that *propagate* will here remove the nodes on the scheduled path before v_1 (since the forward propagation will not reach these nodes, they will be removed as the time window will be empty).

[2]The implementation follows option 1. Should we adapt the implementation?

Algorithm 3 $Delta_0$ for train a

Input: (S, L), (P, A), $M = (m_{time\_step}, m_{duration}, m_{agent})$, mrt, U, c
Output: $(S, L), e, l$

if $\max_{v \in S} A(v) \leq m_{time\_step}$ **then**
  $S \leftarrow \emptyset$, $L \leftarrow \emptyset$
else if $\min_{v \in S} A(v) > m_{time\_step}$ **then**
  $v_1 \leftarrow \arg\min_{v \in S} A(v)$
  $e(v_1) \leftarrow A(v_1)$
  for $\tau \in S : out(\tau) = 0$ **do**
    $l(\tau) = U$
  end for
  $F_v \leftarrow F_e \leftarrow \{ v_1 \}$
  $F_l \leftarrow \{ \tau \in S : out(\tau) = 0 \}$
  $e, l, (S, L) \leftarrow propagate(e, l, (S, L), F_e, F_l, F_v, mrt, U, c)$
else
  $e, l, (S, L) \leftarrow Delta\_0\_running((S, L), (P, A), M, mrt, U, c)$
end if

Algorithm 4 *propagate*

Input: $e, l, (S, L), F_e, F_l, F_v, mrt, U, c$
Output: $e, l, (S, L)$

for $v \in F_v$ **do**
  $S \leftarrow \{ v' \in S :$ there is a v–v' path or a v'–v path in $L \}$
  $L \leftarrow \{ (v, v') \in L : v, v' \in S \}$
end for
$e \leftarrow propagate\_earliest(e, (S, L), F_e, mrt)$
$l \leftarrow propagate\_latest(l, (S, L), F_l, mrt)$
if $c < \infty$ **then**
  for $v \in S - F_e$ **do**
    $l(v) \leftarrow \min\{ l(v), e(v) + c \}$
  end for
  $l \leftarrow propagate\_latest(l, (S, L), F_l, mrt)$
end if
$S \leftarrow \{ v \in S : e(v) \leq l(v) \}$, $L \leftarrow \{ (v, v') \in L : v, v' \in S \}$

Algorithm 5 *Delta_0_running* for running train a

Input: (S, L), (P, A), $M = (m_{time\_step}, m_{duration}, m_{agent})$, mrt, U, c
Output: $e, l, (S, L)$

$(v_1, v_2) \leftarrow (v_1, v_2) \in L$ s.t. $A(v_1) \leq m_{time\_step}$ and $A(v_2) > m_{time\_step}$
$e(v_1) \leftarrow A(v_1)$, $l(v_1) \leftarrow A(v_1)$
if a corresponds to m_{agent} **then**
    $e_1(v_2) \leftarrow A(v_1) + mrt + m_{duration}$
else
    $e_1(v_2) \leftarrow A(v_1) + mrt$
end if
$F_e \leftarrow \{ v_1, v_2 \}$, $F_l \leftarrow \{ v_1 \} \cup \{ \tau \in S : out(\tau) = 0 \}$
for $\tau \in S - \{ v_1 \} : out(\tau) = 0$ **do**
    $l(\tau) \leftarrow U$
end for
$e, l, (S, L) \leftarrow propagate(e, l, (S, L), F_e, F_l, \{ v_1, v_2 \}, mrt, U, c)$

Re-scheduling Delta

Now we define the perfect oracle as outlined above. Let (P_{S0}, A_{S0}) be the solution to the original train scheduling problem and let $N_S = (V_S, E_S, R_S, m_S, a_S, b_S)$ be the network of the re-scheduling problem for an agent and let (P_S, A_S) be a solution to the re-scheduling problem. Algorithm 6 defines the perfect oracle: we keep fixed all times and nodes that are common to S_0 and S, respectively. Everything that is not reachable in path ($F = \Delta_P$) or time ($F_e = F_l = \Delta_A$)) will be removed by *propagate*. Notice that we have $v_1 \in \subseteq \Delta_A \subseteq \Delta_P$ and $v_2 \in \Delta_P$

TODO Christian does $Delta_{perfect}$ work if the agent has already terminated or has not started at malfunction time?

TODO Christian Formalize Delta_0/Delta_perfect of the whole problem for all agents. Proposition: "Delta_perfect ¡= Delta_0", i.e. every solution to Delta_perfect is a solution to Delta_0

Objective for Re-scheduling

We chose to to minimize a linear combination of delay with respect to the initial schedule $S0$ and penalizing diverging route segments (only the first edge of each such segment). This reflects the idea of not re-routing trains without the need of avoiding delay; if re-scheduling is applied in an iterative loop, this should help to avoid "flickering" of decisions. Formally, the objective is to minimize over solutions $S = (P_S, A_S)$ the sum

$$\sum_{a \in \mathcal{A}} \delta \left(S(a, \tau(a)) - S0(a, \tau(a)) \right) + \rho \cdot |\{ v \in \mathcal{V}(S, a) - \mathcal{V}(S0, a) : (v', v) \in P_{S0}(a) \}| \tag{14}$$

Algorithm 6 $Delta_{perfect}$

Input: (S, L), (P_{S_0}, A_{S_0}), (P_S, A_S), $M = (m_{time\_step}, m_{duration}, m_{agent})$, mrt, U, c

Output: $e, l, (S_1, L_1)$

$\Delta_A \leftarrow \{\, v : A_S(v) = A_{S_0}(v) \,\}$
$\Delta_P \leftarrow \{\, v : v \in P_S, v \in P_{S_0} \,\}$
$S_1 \leftarrow \{\, v : v \in P_{S_0} \,\} \cup \{\, v : v \in P_S \,\}$
$L_1 \leftarrow \{\, (v, v') \in P_S \text{ or } (v, v') \in P_{S_0} : v, v' \in S_1 \,\}$
for $v \in \Delta_A$ **do**
    $l(v) \leftarrow e(v) \leftarrow A_{S_0}(v)$
end for
for $v \in S_1 - \Delta_A : out(v) = 0$ **do**
    $l(v) \leftarrow U$
end for
$(v_1, v_2) \leftarrow (v_1, v_2) \in L_1$ s.t. $A_{S_0}(v_1) \leq m_{time\_step}$ and $A_{S_0}(v_2) > m_{time\_step}$
if $v_2 \notin \Delta_A$ **then**
    if a corresponds to m_{agent} **then**
        $e_1(v_2) \leftarrow A_{S_0}(v_1) + mrt + m_{duration}$
    else
        $e_1(v_2) \leftarrow A_{S_0}(v_1) + mrt$
    end if
end if
$e, l, (S_1, L_1) \leftarrow propagate(e, l, (S_1, L_1), \Delta_A \cup \{\, v_2 \,\}, \Delta_A, \Delta_P, mrt, U, c)$

where the *delay model* is step-wise linear,

$$\delta(t) = \begin{cases} \infty & \text{if } t \geq \delta_{cutoff}, \\ \lfloor t/\delta_{step} \rfloor \cdot \delta_{penalty} & \text{else,} \end{cases} \tag{15}$$

for hyper-parameters $\delta_{step}, \delta_{cutoff}, \delta_{penalty}$ and the second term of (14) penalizes re-routing with respect to the initial schedule S0 by weight ρ for every the first edge deviating from the original schedule $S0$.

TODO Christian discuss harmonization with FLUX

3 Ideas H2

Heuristic ideas ML Ideas

TODO Christian pseudo-code naive prediction

Algorithm 7 *transmissionchains*

Input: $\mathcal{A}$, (P_S, A_S), $\mathcal{A}^* \subseteq \mathcal{A}$
Output: $\mathcal{A}$

1: **for** $(S, L, e, l, w) \in \mathcal{A} - \mathcal{A}^*$ **do**
2: $S \leftarrow \{\, v : v \in P_S \,\}$
3: $L \leftarrow \{\, (v, v') \in P_S : v, v' \in S_1 \,\}$
4: $e \leftarrow \{\, (v, P_S(v) : v \in S \,\}$
5: $l \leftarrow \{\, (v, P_S(v) : v \in S \,\}$
6: $w \leftarrow w|_L$
7: **end for**

Reduce the strictness of the oracle by freeing up all the variables of changed trains. I.e. if a train has some change somewhere, all the parameters of the train are available or change such as alternative routes and different times. This is detailled in Algorithm 8, where $\mathcal{A}^*$ denotes the set of changed trains and $\mathcal{A}$ are the trains as of the full re-scheduling problem.

4 Discussion/Related Work/Literature: show links to various Research Approaches

ML delay propagation / recourse / stability of iterative and batch processing simulation OR / heuristics decomposition approaches real-time rescheduling

Algorithm 8 $Delta_{notsoperfect}$

Input: $\mathcal{A}$, (P_S, A_S), $\mathcal{A}^* \subseteq \mathcal{A}$
Output: $\mathcal{A}$

1: **for** $(S, L, e, l, w) \in \mathcal{A} - \mathcal{A}^*$ **do**
2: $S \leftarrow \{ v : v \in P_S \}$
3: $L \leftarrow \{ (v, v') \in P_S : v, v' \in S_1 \}$
4: $e \leftarrow \{ (v, P_S(v) : v \in S \}$
5: $l \leftarrow \{ (v, P_S(v) : v \in S \}$
6: $w \leftarrow w|_L$
7: **end for**

5 Results H1

5.1 Problem Size

In order to show the speed-up postulated by H1, we want to investigate computation times over a wide range of "problem sizes". We saw in Section 3 on a qualitative, illustrative level that the impact of a malfunction varies greatly for different schedules within the same infrastructure, different malfunctions (agent and malfunction time) within the same schedule. In other words, the "size" of the problem is not derivable in an easy direct way from the experiment parameters, and we need a measure for the problem size.

We now define problem size in a semi-formal way in terms of predictors and postdictors: We call *predictor* a function that takes a rescheduling problem as input and predicts the runtime approximately correctly at some level α with probability $1 - \beta$ (i.e. α controls the precision of the prediction and β controls the probability the the prediction is wrong at the given precision). Similarly, a *postdictor* takes the result of rescheduling (including the original problem and solver statistics) and postdicts the runtime for parameters α, β. The number of variables in the solver model is a potential predictor, whereas the number of solver of choices during solver run is a potential postdictor (formally, the runtime itself is a postdictor).

TODO Christian Do we really need the formalization? Only if we want to relate speed-up to problem size! Argue why we want to do this. Do we need in order to show experimental asymptotics? We can derive experiment design without this measure and not relate speed-up to problem size. Introduce later in chapter?

TODO Christian Figures:

- nb of agents x (time_full_after_malfunction, time_delta_after_malfunction) –¿ variance

- nb of cells x (time_full_after_malfunction, time_delta_after_malfunction) –¿ variance
- nb of affected trains x (time_full_after_malfunction, time_delta_after_malfunction) –¿ variance

TODO Christian - Speed-Up: Discussion of multiple predictors/postdictors

5.2 Experiment Design

In light of the findings of the previous paragraphs, we design our experiments in a hierarchical fashion on multiple levels

1. infrastructure
2. schedule
3. malfunction
4. solver runs

where at each level we generate multiple instances for a fixed instance of the previous levels (i.e. we generate multiple infrastructures for the same parameters, we generate multiple schedules for the same infrastructure etc.). The lowest level is a sanity level to investigate the variance of the solver for the same problem.

TODO Christian Numbers, relate to problem size? Get rid of experiment agenda entirely and work on a collection of this folder structure?

5.3 Speed-Up

TODO Christian Show relative speed-up (experiment_id x speed_up) and absolute runtimes of Delta perfect (experiment_id x time_delta_after_malfunction and time_full_after_malfunction x time_delta_after_malfunction. Three plots.

5.4 ASP Model Characteristics

TODO Christian Potassco notebook ratios, large vs. complex

6 Early Results H2

References

[1] Rcs – traffic management for europe's densest rail network. controlling and monitoring with swiss precision. `https://company.sbb.ch/content/dam/`

sbb/de/pdf/sbb-konzern/sbb-als-geschaeftspartner/RCS/SBB_RCS_Broschuere_EN.pdf.sbbdownload.pdf. Accessed: 2020-07-15.

[2] Gabrio C. Caimi. *Algorithmic decision suport for train scheduling in a large and highly utilised railway network*. PhD thesis, ETH Zurich, Aachen, 2009.

[3] The swiss way to capacity optimization for traffic management. https://www.globalrailwayreview.com/wp-content/uploads/SBB-RCS-Whitepaper-NEW.pdf. Accessed: 2020-07-15.

[4] smartrail 4.0: Ein modernisierungsprogramm der schweizer bahnbranche. https://www.smartrail40.ch/index.asp?inc=programm/uebersicht.asp. Accessed: 2020-07-15.

[5] Flatland challenge. multi agent reinforcement learning on trains. https://www.aicrowd.com/challenges/flatland-challenge. Accessed: 2020-07-15.

[6] Dirk Abels, Julian Jordi, Max Ostrowski, Torsten Schaub, Ambra Toletti, and Philipp Wanko. Train scheduling with hybrid answer set programming. *CoRR*, abs/2003.08598, 2020.