TPLP **21** (3): 317–347, 2021. © The Author(s), 2020. Published by Cambridge University Press
doi:10.1017/S1471068420000046 First published online 27 April 2020

Train Scheduling with Hybrid Answer Set Programming[*]

DIRK ABELS and JULIAN JORDI
SBB, Switzerland

MAX OSTROWSKI
Potassco Solutions, Germany

TORSTEN SCHAUB
Potassco Solutions, Germany and University of Potsdam, Germany
Simon Fraser University, Canada and Griffith University, Australia

AMBRA TOLETTI
SBB, Switzerland

PHILIPP WANKO
Potassco Solutions, Germany and University of Potsdam, Germany
(*e-mail:* wanko@cs.uni-potsdam.de)

submitted 29 August 2019; accepted 20 March 2020

Abstract

We present a solution to real-world train scheduling problems, involving routing, scheduling, and optimization, based on Answer Set Programming (ASP). To this end, we pursue a hybrid approach that extends ASP with difference constraints to account for a fine-grained timing. More precisely, we exemplarily show how the hybrid ASP system *clingo*[DL] can be used to tackle demanding planning and scheduling problems. In particular, we investigate how to boost performance by combining distinct ASP solving techniques, such as approximations and heuristics, with preprocessing and encoding techniques for tackling large-scale, real-world train-scheduling instances.

KEYWORDS: constraints, knowledge representation and nonmonotonic reasoning, logic programming methodology and application

1 Introduction

Densely-populated railway networks transport millions of people and carry millions of tons of freight daily, and this traffic is expected to increase even further. Hence, for using a railway network to capacity, it is important to schedule trains in a flexible and global way. This is, however, far from easy since the generation of railway timetables is already known to be intractable for a single track (Caprara *et al*. 2002). Moreover, hundreds of train lines on a densely connected railway network lead to complex interdependencies due to connections between trains and resource conflicts.

[*] This is a substantially extended and revised version of Abels *et al*. (2019). This work was partially funded by DFG grants SCHA 550/9 and 11.

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

We take up this challenge and show how to address real-world train scheduling with hybrid Answer Set Programming (ASP; Lifschitz 1999). Our hybrid approach allows us to specifically account for the different types of constraints induced by routing, scheduling, and optimization. While we address paths and conflicts with regular ASP, we use difference constraints (over integers) to capture fine timings. Similarly, to boost (multi-objective) optimization, we study approximations of delay functions of varying granularity. This is complemented by domain-specific heuristics aiming at improving feasibility checking. Moreover, we introduce preprocessing techniques to reduce the problem size and search space and provide redundant constraints improving propagation. This lifts our approach to a level that allows us to create high-quality train schedules spanning 6 h for a portion of Switzerland within minutes.

In the current practice, train schedules evolve by only minor adjustments year over year. The basic structure of the schedule has been shown to be more or less feasible in practice. The current scheduling tools offer limited support for conflict detection and no support for automatic conflict resolution. When additional trains are ordered by railway companies during the year, or when the capacity is reduced because of maintenance work, it is up to the professional experience and ingenuity of the planner to find a producible schedule or to reduce the number of services if no feasible solution can be found. In the first step, the solution described in this paper is intended to be used by the industrial partner in a decision support system providing planners with conflict-free schedules for adding additional trains into an existing schedule structure. The planner only has to enter the commercial requirements for the additional train, everything else shall be taken care of by the system. In later steps, it is hoped that our solution can be scaled to eventually generate complete, conflict-free schedules from scratch for the whole country and also continually optimize them during operations to account for deviations and disruptions.

We implement our approach with the hybrid ASP system *clingo*[DL] (Janhunen *et al.* 2017), an extension of *clingo* (Gebser *et al.* 2019) with difference constraints. To begin with, we introduce in Section 3, a dedicated formalization of the train-scheduling problem. This is indispensable to master the complex interdependencies of the problem. We present our solution in terms of hybrid ASP encodings, including a detailed description of preprocessing, encoding, and heuristic techniques in Section 4. Finally, we evaluate our approach on real-word instances in Section 5.

2 Background

We rely on a basic acquaintance with ASP. The syntax of our logic programs follows the one of *clingo* (Gebser *et al.* 2015b); its semantics is detailed in Gebser *et al.* (2015a).

The system *clingo*[DL] extends the input language of *clingo* by (theory) atoms representing *difference constraints*. That is, atoms of the form

`&diff{` u-v `}` `<=` d,

where u, v are symbolic terms and d is a numeral term, represent difference constraints such as "$u - v \leq d$," where u, v serve as integer variables and d stands for an integer.[1] For

[1] Strictly speaking, we had to distinguish the integer from its representation.

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

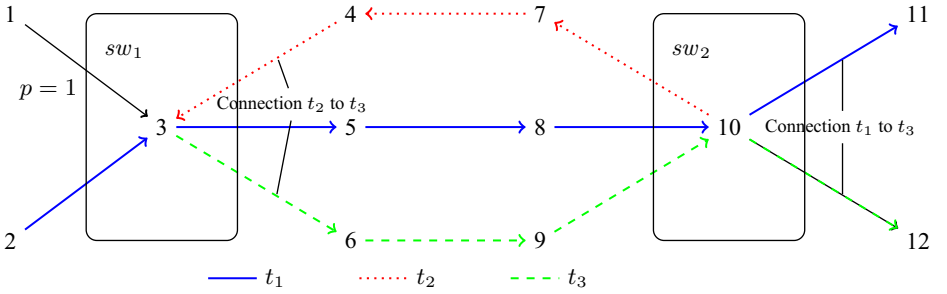


Fig. 1. Routing of three train lines through a railway network.

instance, assume that "`&diff{e(T)-b(T)} <= D`" stands for the condition that the time between the end and the beginning of a task `T` must be less or equal than some duration D. This may get instantiated to "`&diff{e(7)-b(7)} <= 42`" to require that `e(7)` and `b(7)` take integer values such that "$\texttt{e(7)} - \texttt{b(7)} \leq 42$." Note that u, v can be arbitrary terms. We exploit this below and use instances of pairs like `(T,N)` to denote integer variables. Among the alternative semantic couplings between (theory) atoms and constraints offered by *clingo*[DL] (cf. Janhunen *et al.* 2017; Gebser *et al.* 2016), we follow the *defined, non-strict* approach (i) tolerating theory atoms in rule heads and (ii) enforcing their corresponding constraints only if the atoms are derivable. Hence, if a theory atom is false, its associated constraint is ignored. This approach has the advantage that we only need to consider difference constraints occurring in the encoding and not their complements. Obviously, one great benefit of using such constraints is that their variables are not subject to grounding.

For boosting performance, we take advantage of *clingo*'s heuristic directives of form

`#heuristic` a`:`B`. [`w`,sign]`

where a is an atom and B is a body; w is a numeral term and `sign` a heuristic modifier, indicating how the solver's heuristic treatment of a should be changed whenever B holds. Whenever a is chosen by the solver, `sign` enforces that it becomes either true or false depending on whether w is positive or negative, respectively. See Gebser *et al.* (2013, 2015b) for a comprehensive introduction to heuristic modifiers in *clingo*.

3 Real-world train scheduling

3.1 Problem introduction

The train-scheduling problem can essentially be divided into three distinct tasks: routing, conflict resolution, and scheduling.

First, train lines are routed through a railway network. One such network is given in Figure 1. By and large, it is a directed graph containing nodes 1 through 12 with edges in between, for instance, $(2, 3)$ and $(10, 12)$. Given this directed graph, we assign paths through the network to three train lines, viz., t_1, t_2, and t_3. Each train line is assigned an acyclic subgraph capturing its travel trajectory. In our example, the subgraphs for t_1, t_2, and t_3 are indicated by solid, dotted and dashed edges, respectively. Note that $(10, 12)$ belongs to the subgraph of both t_1 and t_3. We see that t_1 has several different

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

start nodes, viz., 1 and 2, and end nodes, viz., 11 and 12, whereas t_2 and t_3 have no alternative routes. One possible solution to the routing task in Figure 1 is represented by edges colored blue, red, and green marking valid paths for t_1, t_2, and t_3, respectively

Second, edges in the network are assigned resources representing, for example, the physical tracks or junctions that can only be passed by a single train at once. Whenever two train lines share edges assigned the same resource, a decision has to be made which train line passes first. In our example, we assume that each edge is assigned a resource representing the physical track. Furthermore, we have two junctions, sw_1 and sw_2, that are assigned five edges each. More precisely, sw_1 is assigned to $(1,3),(2,3),(3,5),(3,6)$, and $(4,3)$, and sw_2 to $(8,10),(9,10),(10,11),(10,12)$, and $(10,7)$. The junctions highlight a common structural property of train-scheduling instances on which we rely heavily to reduce the amount of decisions that have to be made to resolve resource conflicts. Let us focus on sw_2 and the resource conflicts of the three train lines in edges $(8,10),(9,10),(10,7),(10,11)$, and $(10,12)$. Instead of deciding each pair of edges with shared resources individually, we can decide which train line enters a set of edges assigned sw_2 first. More precisely, we decide in which order t_1, t_2, and t_3 enter $\{(8,10),(10,11),(10,12)\}$, $\{(10,7)\}$, and $\{(9,10),(10,12)\}$, respectively. This is possible because train lines using the same resource cannot overtake each other once they are within these sets of edges. We call such sets *resource areas*. Note that while it is obvious in our example what said areas are (they are equal to the full set of assigned edges for each resource and train line), this may not be the case in general. For instance, there are many alternative paths in complex train stations, and thus shared resources might be visited several times enabling train lines to overtake one another.

Finally, a schedule has to be created that assigns each train line an arrival time at all nodes in its path. A valid schedule has to respect a variety of timing constraints, ranging from earliest and latest arrival times at nodes, traveling and waiting times on edges, resource conflicts between train lines, to connections between train lines. Figure 2 shows the earliest and latest arrival times for nodes of the valid paths of Figure 1. This is indicated by the light blue, red, and green areas for train lines t_1, t_2, and t_3, respectively. For instance, t_2 may arrive at node 10 at the earliest at time point 0 and at the latest at time point 360 or t_1 at node 5 between 360 and 660. Furthermore, Figure 2 shows a valid schedule for train lines t_1, t_2, and t_3 as indicated by the blue, red, and green lines, respectively. This schedule results from the decisions that t_2 and t_3 enter sw_1 before t_1 and, conversely, that t_1 enters sw_2 before t_3. Each resource conflict is resolved by a timing constraint indicating that the following train line enters the resource only after the preceding train line has left it plus a safety period. In our example, this safety period, as well as the travel time for each edge, is 60 s. This is reflected in the schedule since t_2 starts at 0 in node 10 and proceeds to node 7 at 60, and t_1 only enters sw_1 at node 2 a minute after t_3 has left it at node 6.

In our example, there are three connections. Our concept of connections captures transfer of passengers and cargo in various contexts, as well as the transfer of physical trains between train lines. The latter enables us to express cyclic train movements as can be seen with the connection of t_2 to t_3 on edges $(4,3)$ and $(3,6)$. We call such a connection *collision-free* since resource conflicts are disregarded on all connected edges that share the same resource, and the connection furthermore requires both train lines to arrive at node 3 at the exact same time point. This connection expresses that the

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

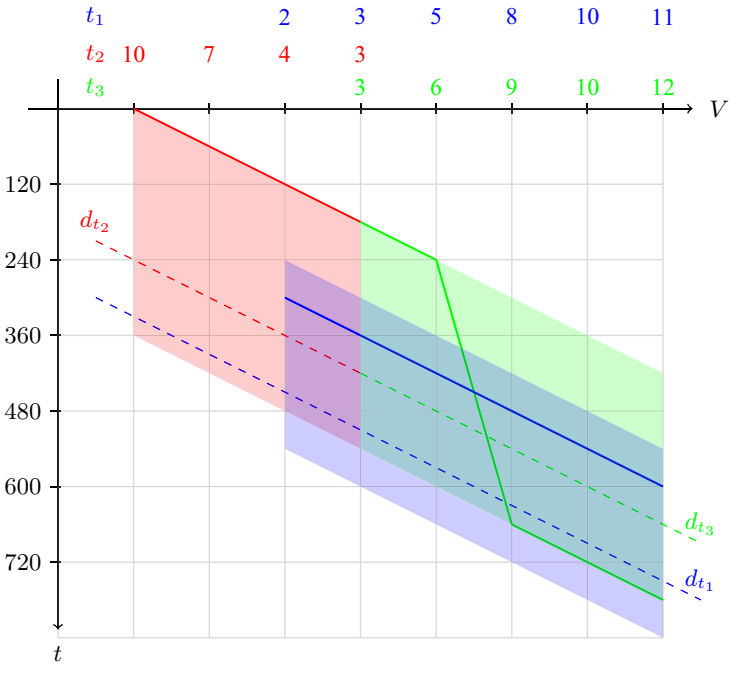


Fig. 2. Scheduling of three train lines.

same physical train is used for train lines t_2 and t_3 and seamlessly transitions at node 3.[2] The results are reflected in the connected paths in Figure 1 and the same arrival time at node 3 in Figure 2. The two other connections from t_1 to t_3 on either edges $(10, 11)$ and $(10, 12)$ or $(10, 12)$ and $(10, 12)$, on the other hand, capture a possible transfer of passengers or cargo from train lines t_1 to t_3. The connection requires t_3 not to arrive at 12 before t_1 has arrived by at least 1 min at 10 so that a transfer is possible. Since this connection does not disable resource conflicts, this can only be achieved if t_1 precedes t_2 through sw_2 which makes t_2 wait between nodes 6 and 9 until t_1 has left.

After obtaining a valid routing and scheduling, the solution is evaluated regarding the delay of the train lines and the quality of the paths they have taken. The former is calculated by summing up the delay of each train line at each node in their paths. For that purpose, a time point is defined after which the train line is considered delayed at a node. In Figure 2, these time points are indicated by lines d_{t_1}, d_{t_2}, and d_{t_3}. For instance, t_3 is delayed at nodes 9, 10, and 12, thus accumulating a penalty of $120+120+120 = 360$. For the quality of the routes, penalties are assigned to edges and accumulated for every train line traveling that edge. Such penalties may indicate tracks that can manage less workload, are in need of maintenance, or are known to be a detour, and therefore to be avoided if possible. In our example, only edge $(1, 3)$ is assigned a penalty of one, other edges are assumed to have a penalty of zero; therefore, our routing is optimal and accumulates no routing penalties.

[2] For simplicity, we assume that collision-free connections are already given with a transitive closure of all adjacent edges of the same resource and all possible other collision-free connections of other trains of that resource.

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

3.2 Problem formalization

We formalize the train-scheduling problem as a tuple (N, T, 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,
  - $a : R \to 2^E$ 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.
- T is a set of train lines to be scheduled on network N.
  Each train in T 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, and we use seconds as time unit.
- C contains connections requiring that a certain train line t' must not arrive at node n' before another train line t has arrived at node n for at least α and at most ω s.

  More precisely, each connection in C is of form $(t, (v, v'), t', (u, u'), \alpha, \omega, n, n')$ such that $t = (S, L, e, l, w) \in T$ and $t' = (S', L', e', l', w') \in T$, $t \neq t'$, $(v, v') \in L$, $(u, u') \in L'$, $\{\alpha, \omega\} \subseteq \mathbb{Z} \cup \{\infty, -\infty\}$, 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 $(F_c)_{c \in C}$.

  Each resource point in F_c is of form $(t, (v, v'), t', (u, u'), r)$ and expresses that two train lines t and t' are allowed to share the same resource r on edges (v, v') and (u, u'). 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. This can be observed in Figure 1, where t_2 and t_3 are connected like so.

  We suppose the following conditions

$$
\begin{aligned}
(t, (n, v), t', (u, u'), r) &\in F_c, \text{ if } (n, v) \in L \cap a(r),\\
(t, (v', n'), t', (u, u'), r) &\in F_c, \text{ if } (v', n') \in L \cap a(r),\\
(t, (v, v'), t', (m, u), r) &\in F_c, \text{ if } (m, u) \in L' \cap a(r), \text{ and}\\
(t, (v, v'), t', (u', m'), r) &\in F_c, \text{ if } (u', m') \in L' \cap a(r).
\end{aligned}
$$

  These assumptions ensure that adjacent edges sharing the same resource are always collision-free resource points for that connection, because otherwise the con-

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

nections could not be made if resources span over several edges. To illustrate this, consider two train lines $t_a = (\{1,2\},\{(1,2)\},e,l,w)$ and $t_b = (\{2,3,4\},\{(2,3),(3,4)\},e',l',w')$ that reuse one physical train. All edges have a minimal travel time of 6 s and the same resource r with a block time of 10 s. To facilitate a seamless transition from one train line to another, a connection $(t_a,(1,2),t_b,(2,3),0,0,2,2)$ is used where both train lines blend into each other. As both train lines ought to use the same physical train, a collision-free point $(t_a,(1,2),t_b,(2,3),r)$ is needed to allow the connection at the given edges. This collision-free point alone is not sufficient, since t_b would have to wait for 4 s at edge $(3,4)$ for the release of the resource r, blocked by t_a at edge $(1,2)$. This is due to the fact that the blocked time of 10 s is larger than the minimal travel time of 6 s. Therefore, the collision-free point $((t_a,(1,2),t_b,(3,4),r))$ is necessary to correctly execute the connection as intended, that is, modeling one physical train among multiple train lines.

For our example in Figures 1 and 2, the train-scheduling problem is defined as

- $V = \{1,\ldots,12\}$,
  - $E = \{(1,3),(2,3),\ldots,(10,11),(10,12)\}$,
  - $R = \{sw_1, sw_2\} \cup \{r_e \mid e \in E\}$,
  - $m(e) = 60$ and $a(r_e) = \{e\}$ for $e \in E$,
  - $a(sw_1) = \{(1,3),(2,3),(4,3),(3,5),(3,6))\}$,
  - $a(sw_2) = \{(8,10),(10,7),(9,10),(10,11),(10,12)\}$,
  - $b(r) = 60$ for $r \in R$,
- $T = \{t_1,t_2,t_3\}$ with
  - $t_1 = (S_1,L_1,e_1,l_1,w_1)$,
  - $t_2 = (S_2,L_2,e_2,l_2,w_2)$,
  - $t_3 = (S_3,L_3,e_3,l_3,w_3)$,

  where (S_1,L_1), (S_2,L_2), and (S_3,L_3) are nodes of edges and edges themselves that are solid, dotted, or dashed in Figure 1, respectively, and
  - e_1,l_1,e_2,l_2,e_3,l_3 give the upper and lower coordinates of the colored areas in Figure 2,
  - $w_1(e) = w_2(e) = w_3(e) = 0$ for $e \in E$,
- $C = \{c_1,c_2,c_3\}$ with
  - $c_1 = (t_2,(4,3),t_3,(3,6),0,0,3,3)$,
  - $c_2 = (t_1,(10,12),t_3,(10,12),60,\infty,10,12)$,
  - $c_3 = (t_1,(10,11),t_3,(10,12),60,\infty,10,12)$,

  and
- $F = (F_{c_1},F_{c_2},F_{c_3})$ where $F_{c_1} = \{(t_2,(4,3),t_3,(3,6),sw_1)\}$ and $F_{c_2} = F_{c_3} = \emptyset$.

A solution (P,A) to a train-scheduling problem (N,T,C,F) is a pair 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.

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

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') is contained in path $p = (v_1, \ldots, v_n)$, that is, whenever $v = v_i$ for some $1 \leq i \leq n$ or this and additionally $v' = v_{i+1}$, respectively.

A path $P(t) = (v_1, \ldots, v_n)$ for $t = (S, L, e, l, w) \in T$ 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 $T \times V \to \mathbb{N}$, where $A(t, v)$ is undefined whenever $v \notin P(t)$. Given path function P, an assignment A has to satisfy the conditions in (4)–(8):

$$A(t, v_i) \geq e(v_i), \tag{4}$$

$$A(t, v_i) \leq l(v_i), \tag{5}$$

$$A(t, v_j) + m((v_j, v_{j+1})) + w((v_j, v_{j+1})) \leq A(t, v_{j+1}), \tag{6}$$

for all $t = (S, L, e, l, w) \in T$ and $P(t) = (v_1, \ldots, v_n)$ such that $1 \leq i \leq n, 1 \leq j \leq n-1$,

either

$$A(t, v') + b(r) \leq A(t', u) \quad \text{or} \quad A(t', u') + b(r) \leq A(t, v), \tag{7}$$

for all $r \in R, \{t, t'\} \subseteq T, t \neq t', (v, v') \in P(t), (u, u') \in P(t')$ with $\{(v, v'), (u, u')\} \subseteq a(r)$ whenever for all $(t, (x, x'), t', (y, y'), \alpha, \omega, n, n') \in C$ such that $(x, x') \in P(t), (y, y') \in P(t')$, we have $(t, (v, v'), t', (u, u'), r) \notin F_c$,

and finally

$$\alpha \leq A(t', n') - A(t, n) \leq \omega, \tag{8}$$

for all $(t, (v, v'), t', (u, u'), \alpha, \omega, n, n') \in C$ if $(v, v') \in P(t)$ and $(u, u') \in P(t')$.

Intuitively, conditions (4)–(6) 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 (7) 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 (8) ensures that train line t connects to t' at nodes 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.

For our solution in Figure 2, we have

- $P(t_1) = (2, 3, 5, 8, 10, 11)$, $P(t_2) = (10, 7, 4, 3)$, and $P(t_3) = (3, 6, 9, 10, 12)$, and
- $A(t_1, 2) = 300, \ldots, A(t_1, 11) = 600$,
  $A(t_2, 10) = 0, \ldots, A(t_2, 3) = 180$, and
  $A(t_3, 3) = 180, A(t_3, 6) = 240, A(t_3, 9) = 660, \ldots, A(t_3, 12) = 780$.

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

To determine the quality of a solution, both the aggregated delay of all train lines and the quality of the paths through the network are taken into account. For that purpose, we consider two functions: the delay function d and route penalty function rp. Given a train line $t = (S, L, e, l, w) \in T$ and a node $s \in S$, $d(t, s) \in \mathbb{N}$ returns the time point after which the train line t is considered late at node s. Note that $e(s) \leq d(t, s) \leq l(s)$. Given an edge $e \in E$, $rp(e) \in \mathbb{N}$ is the penalty a solution receives for each train line traveling edge e. With this, the quality of a solution (P, A) is determined via the following pair:

$$\big(\textstyle\sum_{((t,v),a)\in A} \max\{(a - d(t, v)), 0\}/60,\ \sum_{e\in\{u|p\in P,u\in p,e\in E\}} rp(e)\big). \tag{9}$$

Since delay is the more important criteria, optimization of the quality amounts to lexicographic minimization considering delay first and route penalty second. As mentioned, our example has an accumulated delay of 360 and 0 route penalty and therefore a quality of $(360/60, 0) = (6, 0)$.

4 An ASP-based solution to real-world train scheduling

In this section, we present our hybrid ASP-based approach for solving the train-scheduling problem. It relies on dedicated preprocessing techniques to reduce the problem size as well as additional constraints and domain-specific heuristics to reduce the search space and improve solving performance. This constitutes a significant improvement over our previous approach (Abels *et al.* 2019) that, while similar in principle, does not scale to the largest instances available. We start by presenting said preprocessing techniques that mainly exploit redundancy in the resource distribution in the railway network. Following that, we describe the actual hybrid encoding that makes use of this preprocessing and furthermore reduces the amount of timing constraints by using a compressed representation of the graph. Finally, we present optional constraints and domain-specific heuristics aiming at further improving solving performance.

4.1 Preprocessing

We present two techniques that reduce the complexity of the problem at hand. While *resource subsumption* detects redundant resources that can safely be removed, *resource areas* are used to simplify conflict resolution on resource conflicts.

4.1.1 Resource subsumption

An analysis of real-world instances revealed that resources are often contained within others, thus posing redundant constraints. In a preprocessing step, we detect such subsumed resources and remove them from the problem specification. Given a railway network (V, E, R, m, a, b), a resource $r_1 \in R$ is subsumed by another resource $r_2 \in R$, if $a(r_1) \subseteq a(r_2)$ and $b((v, v')) \leq b((u, u'))$ for all $(v, v') \in a(r_1)$ and $(u, u') \in a(r_2)$, and there is no collision-free resource point $(t, (v, v'), t', (u, u'), r_2) \in F_c$ for any $c \in C$.[3] Intuitively, subsumed resources are contained within another resource that induces the

[3] If several resources are exactly the same, we keep one of them.

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

same or stronger timing constraints due to higher or equal blocked time and no conflict-free resource points in the overlapping area. In our example, we can safely remove resources $\{r_{(10,7)}, r_{(10,11)}, r_{(8,10)}, r_{(9,10)}, r_{(10,12)}\}$ as each of them is subsumed by sw_2. Note that we cannot remove any resources covered by sw_1, as it is used in the collision-free resource point $(t_2, (4,3), t_3, (3,6), sw_1)$ of c_1. The collision-free resource point disables conflict resolution on sw_1 for the trains t_1 and t_2; hence, conflicts for other resources topologically contained within sw_1 are not redundant and have to be taken into account individually.

4.1.2 Resource areas

As seen in our example, resources covering several edges can be exploited by identifying areas, for which it is enough to determine the order of train lines passing through (rather than doing pair-wise conflict resolution on edges, as done in Abels *et al.* 2019). We call them *resource areas*. Every train line has its own set of resource areas. It is required that every possible path through the resource area contains only edges assigned the original resource. More precisely, a resource area A_r^t over resource $r \in R$ and train line $t = (S, L, e, l, w)$ is a maximal set of edges $A_r^t \subseteq a(r) \cap L$ such that there is no path $p = (v_1, \ldots, v_n)$ in (S, L) with $(u, u') \in p$ but $(u, u') \notin a(r)$ between two edges $\{(v, v_1), (v_n, v')\} \in A_r^t$. Intuitively, this means that a resource area can only be occupied by one train at a time independently of their chosen paths through the resource area. Thus, other train lines using the same resource may only enter once the entire area is free. Note that there may exist several resource areas for a resource and a train.

We define a *resource coverage* $\mathcal{A}_r^t$ as a set of resource areas over resource r and train line t such that $\bigcup_{A \in \mathcal{A}_r^t} A = a(r) \cap L$ and $A \cap A' = \emptyset$ for $\{A, A'\} \subseteq \mathcal{A}_r^t$. A resource coverage $\mathcal{A}$ over sets of resources R and train lines T is defined as $\mathcal{A} = \bigcup_{r \in R, t \in T} \mathcal{A}_r^t$. Intuitively, a resource coverage distributes resource areas so that every edge is covered and no edge is in two resource areas per resource and train line.

In our example, we have the resource coverage

$$\begin{aligned}\mathcal{A} = &\{\{(v,v')\}^t_{r_{(v,v')}} \mid t \in T, r_{(v,v')} \in R\} \cup \\ &\{\{(1,3),(2,3),(3,5)\}^{t_1}_{sw_1}, \{(8,10),(10,11),(10,12)\}^{t_1}_{sw_2}\} \cup \\ &\{\{(4,3)\}^{t_2}_{sw_1}\}, \{(10,7)\}^{t_2}_{sw_2}\} \cup \\ &\{\{(3,6)\}^{t_3}_{sw_1}, \{(9,10),(10,12)\}^{t_3}_{sw_2}\},\end{aligned}$$

where T and R are the train lines and resources of the example in Figure 1, respectively.

We have three nonsingleton sets in the coverage. Given the train lines t_1, t_3, and resource sw_2 from our example, we have to decide whether t_1 enters $\{(8,10),(10,11),(10,12)\}^{t_1}_{sw_2}$ first or t_3 enters $\{(9,10),(10,12)\}^{t_3}_{sw_2}$ first. Without the use of resource areas, we would need to make a decision for every element in the cross product $\{(8,10),(10,11),(10,12)\}^{t_1}_{sw_2} \times \{(9,10),(10,12)\}^{t_3}_{sw_2}$, leaving us with six decisions instead of one. Note that there is no unique resource coverage for every graph, but different maximal subsets could be chosen.

To determine resource areas, we use the greedy algorithm in Figure 3 for every resource r and train line $t = (S, L, e, l, w)$. We incrementally create a resource area A by adding yet unused edges (v, v') to it in line 4. The function $isRA(A, r, S, L)$ in line 5 checks

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

Input : Resource r and train line $t = (S, L, e, l, w)$.
Output : A resource coverage A_r^t.

1 $A_r^t \leftarrow \emptyset$;
2 **while** $\bigcup_{A \in A_r^t} A \neq a(r) \cap S$ **do**
3 $\quad A \leftarrow \emptyset$;
4 $\quad$ **foreach** $(v, v') \in (a(r) \cap S) \setminus \bigcup_{A \in A_r^t} A$ **do**
5 $\quad\quad$ **if** *isRA*$(A \cup \{(v, v')\}, r, S, L)$ **then**
6 $\quad\quad\quad A \leftarrow A \cup \{(v, v')\}$;
7 $\quad$ **end**
8 **end**

Fig. 3. Algorithm to compute resource coverage.

that no path in (S, L) between edges in A contains edges not assigned r. Intuitively, this means that once a train line entered a resource area, it is not able to leave it and reenter it again. If this is the case, (v, v') is added to A in line 6. This is repeated until we cannot add further edges to the current resource area and then restart the procedure creating a new area in lines 2 and 3 until we achieve full resource coverage.

We outline how we use resource areas to derive solutions fulfilling Condition (7) in Section 4.3.

4.2 Fact format

A train-scheduling problem (N, T, C, F) with $N = (V, E, R, m, a, b)$ is represented by the facts

`tl(`t`).` `edge(`t`,`v`,`v'`).` `m((`v, v'`),`$m((v, v'))$`).` `w(`t`,(`v, v'`),`$m((v, v'))$`).`

for each $t = (S, L, e, l, w) \in T$ and $(v, v') \in L$.

For every $s \in S$, we add

`e(`t`,`s`,`$e(s)$`).` `l(`t`,`s`,`$l(s)$`).`

Additionally, either

`start(`t`,`s`).` or `end(`t`,`s`).`

is added, if $in(s) = 0$ or $out(s) = 0$ in (S, L), respectively. We assign unique terms to each train line for identifiability.

For example, the facts

```
tl(t1).        edge(t1,1,3).   m((1,3),60).   w(t1,(1,3),0).
e(t1,1,240).   l(t1,1,540).
start(t1,1).
```

express that train line `t1` may travel between nodes `1` and `3` taking at least 60 s, waiting on this edge for 0 s, and arrives between time points 240 and 540 at node `1`, which is a possible start node.

Furthermore, we add

`resource(`r`,(`v`,`v'`)).` `b(`r`,`$b(r)$`).`

for $r \in R$ and $(v, v') \in a(r)$. Akin to train lines, resources are assigned unique terms to distinguish them.

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

For example, facts

`resource(sw1,(1,3)).` `resource(sw1,(4,3)).` `b(sw1,60).`

assign resource `sw1` to edges $(1,3)$ and $(4,3)$ and the resource is blocked for 60 s after a train line has left it.

Finally, we add

`connection(`c`,`t`,(`v`,`v'`),`t'`,(`u`,`u'`),`α`,`ω`,`n`,`n'`).`

for all $(t,(v,v'),t',(u,u'),\alpha,\omega,n,n') \in C$, where c acts as an identifier, and

`free(`c`,`t`,(`v`,`v'`),`t'`,(`u`,`u'`),`r`).`

for all $(t,(v,v'),t',(u,u'),r) \in F_c$.

For instance, the transfer of the physical train from train lines `t2` to `t3` at node 3 is encoded by `connection(1,t2,(4,3),t3,(3,6),0,0,3,3).` This requires both train lines to be at node `3` at the exact same time. Thus, we have to additionally provide the fact `free(1,t2,(4,3),t3,(3,6),sw1)` to allow for a shared use of the resource, so that the train lines can connect. This can be done safely since in reality only one train exists for both train lines, which rules out collisions.

For a resource area $A \in \mathcal{A}_r^t$, we generate facts

`ra(`t`,`r`,`a`,(`v`,`v'`)).`

for $(v,v') \in A$, resource $r \in R$, and train line $t = (S,L,e,l,w) \in T$. We assign a unique term a to distinguish resource areas for the same resource and train line. Furthermore, for every resource area $A \in \mathcal{A}_r^t$, we add

`e_ra(`t`,`r`,`a`,`e`).` `l_ra(`t`,`r`,`a`,`l`).`

where $e = min\{e(v)) \mid (v,v') \in A\}$ and $l = max\{l(v') \mid (v,v') \in A\}$ to represent the earliest entry and latest exit times for that resource area. In our example, the earliest entry time for resource area $\{(1,3),(2,3),(3,5)\}_{sw_1}^{t_1}$ is 0, while the latest exit time is 660.

Given delay and route penalty functions d and rp, we add

`potlate(`t`,`s`,`u`,`p`).` `penalty(`m`,`$rp(m)$`).`

for $t = (S,L,e,l,w) \in T, s \in S, m \in L$ with $\{u,p\} \subseteq \mathbb{N}, d(t,s) < u \leq l(t,s)$ to evaluate a solution.

The collection of facts for our example instance can be found in Listing 9 in the Appendix.

4.3 Encoding

In the following, we describe the general problem encoding. We separate it into three parts handling path constraints, conflict resolution, and scheduling.

```
1 1 { visit(T,V)       : start(T,V)   } 1 :- tl(T).
2 1 { route(T,(V,V')) : edge(T,V,V') } 1 :- visit(T,V),
3                                           not end(T,V).
4 visit(T,V') :- route(T,(V,V')).
```

Listing 1. *Encoding of path constraints.*

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

The first part of the encoding in Listing 1 covers routing. First, exactly one valid start node is chosen for each train line to be visited (line 1). From a node that is visited by a train line and is not an end node, an edge in the relevant subgraph is chosen as the next route (line 3). The new route in turn leads to a node being visited by the train line (line 4). This way, each train line is recursively assigned a valid path. Since those paths begin at a start node, finish at an end node, and are connected via edges valid for the respective train lines, conditions (2) and (3) are ensured.

```
5  enter_ra(T,R,A,V) :-
6     ra(T,R,A,(V,V')), route(T,(V,V')),
7     1 #sum {1 : edge(T,V'',V), not ra(T,R,A,(V'',V));
8             1 : start(T,V)},
9     not route(T,(V'',V)) : ra(T,R,A,(V'',V)).
10 leave_ra(T,R,A,V') :-
11    ra(T,R,A,(V,V')), route(T,(V,V')),
12    1 #sum {1 : edge(T,V',V''), not ra(T,R,A,(V',V''));
13            1 : end(T,V')},
14    not route(T,(V',V'')) : ra(T,R,A,(V',V'')).
15
16 free_a(T,T',R,A,A') :- free(I,T,(V,V'),T',(U,U'),R),
17                       ra(T,R,A,(V,V')), ra(T',R,A',(U,U')),
18                       connection(I,T,(X,X'),T',(Y,Y'),_,_,_,_),
19                       route(T,(X,X')), route(T',(Y,Y')).
20
21 shared(T,T',R,A,A') :- e_ra(T,R,A,E), e_ra(T',R,A',E'),
22                       not free_a(T,T',R,A,A'),
23                       l_ra(T,R,A,L), b(R,B), T != T',
24                       E <= E', E' < L+B.
25 shared(T',T,R,A',A)  :- shared(T,T',R,A,A').
26
27 { seq(T,T',R,A,A') } :-  shared(T,T',R,A,A'), T < T'.
28   seq(T',T,R,A',A)   :-  shared(T,T',R,A,A'),
29                          not seq(T,T',R,A,A').
```

Listing 2. *Encoding of conflict resolution.*

The next part of the encoding shown in Listing 2 detects and resolves resource conflicts. Basically, a resource conflict exists, if two train lines each have an edge in their respective subgraph that is assigned the same resource, and time intervals overlap in which the train lines enter and leave the edges in question, extended by the time the resource is blocked. We improve upon this edge-centric conflict resolution via resource areas. In lines 5–14, we first compute entry and exit nodes leading in and out of a resource area (depending on the chosen route). A conflict is possible, if two train lines each have a resource area in their respective subgraph that is assigned the same resource, and the time intervals overlap in which the train lines may enter and leave the areas in question, extended by

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

the time the resource is blocked (lines 21–25). We resolve the conflict by making a choice which train line passes through this area first (lines 27 and 29).

As a collision-free resource point $(t, (v, v'), t', (u, u'), r)$ does not trigger a resource conflict, we prevent the involved resource areas A_r^t and $A_r^{t'}$ with $(v, v') \in A_r^t$ and $(u, u') \in A_r^{t'}$ from creating conflicts by deriving collision-free resource areas in lines 16–19. It is safe to do so, as it is necessary for a collision-free connection that all edges adjacent to (v, v') or (u, u') with the same resource are also collision-free resource points (recall Section 3.2). We exclude these areas from the conflict resolution in line 22.

Note that we require much less decisions for resolving resource conflicts by using resource areas. It is possible that several or even all edges in two resource areas with the same resource induce a conflict. Naively, a decision would have to be made for all possible combination of those edges. Instead of having this quadratic blowup, we only have to resolve the resource conflict between entire resource areas. For our example, this reduces the number of decisions necessary to resolve all possible conflicts from 15 to 5.

To represent the arrival time of a train line $t = (S, L, e, l, w)$ at node $v \in S$, we project the graph (S, L) to a possibly smaller version of it using a topological ordering. In essence, instead of assigning an arrival time to each possible node a train line could visit, we assign the arrival time to the progress the train line has made relative to its subgraph. For that purpose, we utilize the *height* of a node that indicates the maximum number of edges the train line has to travel to reach this node, (or, in other words, in an acyclic graph, the height of a node is the length of the longest path from any root node to this node). Obviously, several nodes can have the same height whenever they are on parallel paths through the subgraph. Formally, the height of a node $v \in S$ for train line t is defined as follows:

$$h^t(v) = \begin{cases} 0, & \text{if } in(v) = 0, \\ \max\{h^t(v')) \mid (v', v) \in L\} + 1, & \text{otherwise,} \end{cases}$$

The arrival times are now represented using integer variables designated by pairs of form $(t, h^t(v))$ indicating arrival time of train line t at height $h^t(v)$[4] This reduces the number of variables and difference constraints needed whenever trains are routed over nodes with the same height. For the running example, nodes 1 and 2 from train line t_1 now collapse to one variable $(t_1, 0)$, as only one of the two nodes can be used in a final routing of the train line. Instead of introducing arrival times for nodes 1 and 2, we only introduce an arrival time $(t_1, 0)$ where $0 = h^{t_1}(1) = h^{t_1}(2)$, independent of the chosen path. The number of integer variables for the arrival times of t_1 is therefore reduced from 8 to 6 in contrast to using node names such as `(t,v)`.

Note that we use ground terms in the remainder of this paper to describe our rules, while their actual encoding uses variables. Also, we sometimes mix mathematical notation for train lines and nodes (*italic*) and their identifying terms in the encoding (`typewriter`) and use them interchangeably.

[4] Cf. Section 2 on using pairs as identifiers.

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

```
30 h(T,V,0)    :- start(T,V).
31 h(T,V,M+1) :- edge(T,_,V),
32                M = #max {N : h(T,V',N), edge(T,V',V)}, M != #inf.
33
34 min_e(T,N,M) :- h(T,_,N),
35                  M != #sup, M = #min {E : h(T,V,N), e(T,V,E)}.
36 max_l(T,N,M) :- h(T,_,N),
37                  M != #inf, M = #max {L : h(T,V,N), l(T,V,L)}.
38 &diff{ 0-(T,N) } <= -E :- min_e(T,N,E).
39 &diff{ (T,N)-0 } <=  L :- max_l(T,N,L).
40 &diff{ 0-(T,N) } <= -E :- e(T,V,E), visit(T,V), h(T,V,N),
41                           min_e(T,N,M), M<E.
42 &diff{ (T,N)-0 } <=  L :- l(T,V,L), visit(T,V), h(T,V,N),
43                           max_l(T,N,M), M>L.
44
45 time(T,(V,V'),(N,N'),D) :- edge(T,V,V'), h(T,V,N), h(T,V',N'),
46                             D=#sum{ M,m : m((V,V'),M);
47                                     W,w : w(T,(V,V'),W) }.
48 min_time(T,(N,N'),M) :-
49    edge(T,V,V'), h(T,V,N), h(T,V',N'),
50    not edge(T,U,U') : h(T,U,N), h(T,U',0), 0 != N';
51    not edge(T,U,U') : h(T,U,0), h(T,U',N'), 0 != N;
52    M = #min {D : time(T,_,(N,N'),D)}.
53 &diff{ (T,N)-(T,N') } <= -D :- min_time(T,(N,N'),D).
54 &diff{ (T,N)-(T,N') } <= -D :-
55    route(T,(V,V')), h(T,V,N), h(T,V',N'),
56    time(T,(V,V'),(N,N'),D),
57    D > #max {#inf; M : min_time(T,(N,N'),M)}.
58
59 &diff{ (T,N)-(T',N') } <= -B :-
60    seq(T,T',R,A,A'), shared(T,T',R,A,A'),
61    h(T,V,N), h(T',U,N'),
62    leave_ra(T,R,A,V), enter_ra(T',R,A',U), b(R,B).
63
64 &diff{ (T,N)-(T',N') } <= -M :-
65    connection(I,T,(V,V'),T',(U,U'),M,_,X,Y),
66    h(T,X,N), h(T',Y,N'), route(T,(V,V')), route(T',(U,U')).
67 &diff{ (T',N')-(T,N) } <= M  :-
68    connection(I,T,(V,V'),T',(U,U'),_,M,X,Y), M != #inf,
69    h(T,X,N), h(T',Y,N'), route(T,(V,V')), route(T',(U,U')).
```

Listing 3. *Encoding of scheduling.*

Listing 3 displays the encoding of scheduling via difference constraints. An atom `h(`t`,`v`,`n`)` assigns a node v its height $n = h^t(v)$ for train line t and is computed in lines 30 and 32. Note that we use pairs (t, h) as names for integer variables whose values indicate

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

the arrival time of train t at height h. The next part of the encoding ensures that earliest and latest arrival times, viz., $e(v)$ and $l(v)$, for a train line $t = (S, L, e, l, w)$ on a node $v \in S$ are respected by setting lower and upper bounds for the integer variable $(t, h^t(v))$ (lines 40–43). Even though, the lower and upper bounds of variable (t, h) depend on the node that is actually visited as there possibly exist several nodes $v \in S$ on alternative paths with the same height h (the same maximum path length from a root node), we can take advantage of the height-based naming scheme by setting the lower and upper bounds of variable (t, h) to $\min\{e(v) \mid v \in S, h^t(v) = h\}$ and $\max\{l(v) \mid v \in S, h^t(v) = h\}$, respectively, independently of the chosen route. For all nodes with a greater time for the earliest arrival (lesser time for the latest arrival), an additional constraint is derived depending on whether such a node is visited. This restricts the search space in two ways. First, constraints for earliest and latest arrival times are independent of routing for nodes with a minimal earliest or maximal latest arrival time, second, regardless of routing, a constraint is added constituting the lower bound of the earliest (upper bound for the latest) arrival time at a certain height. Lines 34–37 compute said minimal and maximal arrival times for a train line at a certain height. Difference constraints representing these upper and lower bounds are derived in lines 38 and 39 and are independent of the chosen route. More precisely, for train line t at height h, we derive

$$\texttt{\&diff\{0-}(t,h)\texttt{\}} \texttt{ <= } -e \quad \text{and} \quad \texttt{\&diff\{0-}(t,h)\texttt{\}<= } -l,$$

where $e = \min\{e(v) \mid v \in S, h^t(v) = h\}$ and $l = \max\{l(v) \mid v \in S, h^t(v) = h\}$. For nodes v, where $e(v) = e$ and $l(v) = l$, we therefore ensure $e(v) \leq (t, h^t(v)) \leq l(v)$ and in turn condition (4) and (5). In case train line t visits a node v for which either $e(v) > e$ or $l(v) < l$, we have to additionally derive

$$\texttt{\&diff\{0-}(t,h^t(v))\texttt{\}} \texttt{ <= } -e(v) \quad \text{or} \quad \texttt{\&diff\{}(t,h^t(v))\texttt{-0\}} \texttt{ <= } l(v),$$ respectively.

This again ensures $e(v) \leq (t, h^t(v)) \leq l(v)$ and in turn conditions (4) and (5) to hold.

The next part of the encoding guarantees the travel and waiting time between two nodes $(v, v') \in L$ for a train line $t = (S, L, e, l, w)$. We call adjacent heights $(n, n+1) = (h^t(v), h^t(v'))$ *exclusively connected*, if there is no edge $(u, u') \in L$ such that $h^t(u) = n$ and $h^t(u') \neq n+1$ or $h^t(u') = n+1$ and $h^t(u) \neq n$. For these heights $(n, n+1)$, the minimum of the travel plus waiting time can be set independently of the chosen route. An exclusively connected pair $(n, n+1)$ represents a set of edges $\{(v_1, v_2), \ldots, (v_{m-1}, v_m)\} \subseteq L$ of which at most one edge can be visited by a train line. The minimum of travel plus waiting time of these edges can therefore be guaranteed independently of routing, as it constitutes a lower bound. For any edge $(v, v') \in L$, where either the travel plus waiting time is greater than the minimum across all edges for exclusively connected heights $(h^t(v), h^t(v'))$, or $(h^t(v), h^t(v'))$ is not exclusively connected, the constraint for minimal distance between (t, v) and (t, v') depends on the routing. In the worst case, no exclusively connected heights exist, and all constraints are dependent on the routing. In our running example, edges $(1, 3)$ and $(2, 3)$ for train line t_1 result in the same pair of exclusively connected heights $(0, 1)$. This means that the condition $(t_1, 0) + 60 \leq (t_1, 1)$ holds independently of routing as 60 is the minimum of travel plus waiting time for these edges. If edge $(2, 3)$ had a waiting time $w((2, 3)) = 10$, the condition $(t_1, 0) + 70 \leq (t_1, 1)$ needs to hold additionally, if edge $(2, 3)$ is visited. Note that the route-independent constraint only

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

poses a lower bound on the relation of the two arrival times and is therefore subsumed by this new constraint. Given a train line $t = (S, L, e, l, w)$, lines 45–52, first, compute the travel plus waiting time $time^t_{(v,v')} = m((v,v')) + w((v,v'))$ for all edges $(v,v') \in L$, second, the minimum of travel plus waiting time $min_time^t_{(n,n+1)} = \min\{time^t_{(v,v')} \mid (v,v') \in L, h^t(v) = n, h^t(v') = n+1\}$ for all exclusively connected heights $(n, n+1)$, and finally, the difference constraint atom

$$\texttt{\&diff\{}(t,n)\texttt{-}(t,n+1)\texttt{\}<=}\ -min_time^t_{(n,n+1)}$$

is derived in line 53. For any edge $(v,v') \in L$, where either $time^t_{(v,v')} > min_time^t_{(h^t(v),h^t(v'))}$ holds, or $(h^t(v), h^t(v'))$ are not exclusively connected, the difference constraint atom

$$\texttt{\&diff\{}(t,h^t(v))\texttt{-}(t,h^t(v'))\texttt{\}}\ \texttt{<=}\ -time^t_{(v,v')}$$

is derived, whenever the train line is actually routed over edge (v,v') (lines 54–57). For all exclusively connected heights $(n, n+1)$, the condition $(t,n) + min_time^t_{(n,n+1)} \leq (t, n+1)$ is provided. This satisfies condition (6) for all edges $(v,v') \in L$, where $h^t(v) = n$, $h^t(v') = n+1$ and $min_time^t_{(n,n+1)} = m((v,v')) + w((v,v'))$. As the minimum time is a lower bound, no constraints are violated if these edges are not visited. For all visited edges $(v,v') \in L$ where $h^t(v) = n$, $h^t(v') = n+1$ and $min_time^t_{(n,n+1)} < m((v,v')) + w((v,v'))$ or $(h^t(v), h^t(v'))$ is not exclusively connected, condition $(t,n) + m(v,v') + w((v,v')) \leq (t, n+1)$ for all visited edges (v,v') has to hold. This ensures condition (6). As for the earliest and latest arrival times above, by using a formulation based on heights, we may either gain a lower bound or provide timing constraints independently of the routing, thus restricting the search space.

The rule in lines 59–62 in Listing 3 utilizes conflict detection and resolution from Listing 2. We derive the difference constraint atom

$$\texttt{\&diff\{}(t,h^t(v))\texttt{-}(t',h^{t'}(u))\texttt{\}}\ \texttt{<=}\ -b(r)$$

expressing that t leaves the resource area A^t_r using a node v before t' enters $A^{t'}_r$ via a node u, given the blocked time $b(r)$ of a resource r and the decision that train line t on resource area A^t_r takes precedence over t' on resource area $A^{t'}_r$. As all edges in these resource areas have the same resource r, and once inside a resource area a train cannot leave the resource area again, the order of passing all edges is the same. Therefore, this constraint supersedes all constraints $(t, h^t(v)) + b(r) \leq (t, h^{t'}(u'))$ for all edges $(v,v') \in A^t_r$, $(u,u') \in A^{t'}_r$, and captures condition (7).

Finally, lines 64–69 handle connections $(c, t, (v,v'), t', (u,u'), \alpha, \omega, v'', u'') \in C$. Difference constraint atoms

$$\texttt{\&diff\{}(t,h^t(v''))\texttt{-}(t',h^{t'}(u''))\texttt{\}}\ \texttt{<=}\ -\alpha \quad \text{and}$$
$$\texttt{\&diff\{}(t',h^{t'}(u''))\texttt{-}(t,h^t(v''))\texttt{\}}\ \texttt{<=}\ \omega$$

are derived (lines 66 and 69) guaranteeing $\alpha \leq (t', h^{t'}(u'')) - (t, h^t(v'')) \leq \omega$, and thus condition (8), if (v,v') is visited by train line t and (u,u') is visited by t'.

4.4 Optimization

As mentioned above, we use instances of `potlate`/4 to indicate when a train line is considered late at a node and how to penalize its delay. For this purpose, we choose sets

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

$D_{t,v} \subseteq \mathbb{N}$ whose elements act as thresholds for arrival time of train line t at node v. Given delay function d, $d(t,v) \leq u \leq l(v)$ for every $u \in D_{t,v}$, train line $t = (S, L, e, l, w) \in T$ and $v \in S$. We create facts `potlate(`t`,`v`,`u`,`$u-u'$`)` for $u, u' \in D_{t,v}$ with $u' < u$ such that there is no $u'' \in D_{t,v}$ with $u' < u'' < u$. We add `potlate(`t`,`v`,`u`,`$u - d(t,v)$`)` for $u = \min(D_{t,v})$. Intuitively, we choose the penalty of a potential delay as the difference to the previous potential delay, or, if there is no smaller threshold, the difference to the time point after which the train is considered delayed. This way, the sum of penalties amounts to a lower bound on the train line's actual delay in seconds. For example, for $D_{t,v} = \{6, 10, 14\}$ and $d(t,v) = 5$, we create facts `potlate(`t`,`v`,6,1)`, `potlate(`t`,`v`,10,4)`, and `potlate(`t`,`v`,14,4)`. Now, if t arrives at v at 12, it is above thresholds 6 and 10 and should receive a penalty of 5. This penalty is a lower bound on the actual delay of 7, and we know that the value has to be between 5 and 9 since the next threshold adds a penalty of 4. This method approximates the exact objective function in (9) in two ways. First, we do not divide by 60 and penalize in minutes since this would lead to rounding problems. Second, our penalty only gives a lower bound to the actual delay if thresholds are more than 1 s apart. While our method allows us to be arbitrarily precise in theory, in practice, creating a threshold for each possible second of delay leads to an explosion in size. We employ two schemes for generating sets $D_{t,v}$ given $t = (S, L, e, l, w) \in T$, $v \in S$ and delay function d.

4.4.1 Binary approximation

Binary approximation detects if a train is a second late and penalizes it by one, therefore, only the occurrence of a delay is detected while its amount is disregarded.

We set $D_{t,v} = Bin_{t,v} = \{d(t,v) + 1\}$.

4.4.2 Linear approximation

Linear approximation evenly distributes thresholds m s apart across the time span in which a delay might occur. Here, if train line t arrives at or after $n \times m + d(t,v)$ at v, we know that the real delay is between $n \times m$ and $(n+1) \times m$ for $n \in \mathbb{N} \setminus \{0\}$. We also add $Bin_{t,v}$ to detect solutions without delay.

Accordingly, we set $D_{t,v} = Bin_{t,v} \cup Lin^m_{t,v}$ with $Lin^m_{t,v} = \{y \in \mathbb{N} \mid y = x \times m + d(t,v), x \in \mathbb{N} \setminus \{0\}, y \leq l(v)\}$.

4.4.3 Encoding

```
70  { late(T,V,D,W) : visit(T,V) } :- potlate(T,V,D,W).
71
72  next(T,V,D,D') :- potlate(T,V,D,_), potlate(T,V,D',_), D<D',
73                    not potlate(T,V,D'',_) : potlate(T,V,D'',_),
74                    D''>D, D''<D'.
75  :- not late(T,V,D,_), late(T,V,D',_), next(T,V,D,D').
76
77  topo_late(T,N,D,W) :- late(T,V,D,W), h(T,V,N).
```

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

```
70  &diff{ 0-(T,N) } <= -D :- topo_late(T,N,D,W).
71  &diff{ (T,M)-0 } <=  N :- not topo_late(T,M,D,W),
72                            potlate(T,V,D,W),
73                            N=D-1, visit(T,V), h(T,V,M).
74
75  #minimize{ W@1,T,N,D : topo_late(T,N,D,W) }.
76  #minimize{ P@0,T,E   : penalty(E,P), route(T,E) }.
```

Listing 4. *Delay and routing penalty minimization.*

Given thresholds $D_{t,v}$ for all train lines and nodes and the corresponding instances of predicate `potlate`/4, Listing 4 shows the implementation of the delay minimization. The basic idea is to use regular atoms to choose whether a train line is delayed on its path for every potential delay (line 70 in Listing 4). In lines 72–75, we order the given thresholds for every train line and node. By enforcing downward adjacent thresholds to hold as well (being late for 4 min implies being late for 3 min which in turn implies being late for 2 min, etc.), we immediately exclude solutions where this semantic condition fails, thus improving propagation. Originally, the semantics of the `late` atoms (being late for x min) is only given via the difference constraint atom introduced later. By directly encoding the implicit ordering of natural numbers using regular ASP atoms, we leverage the efficient Boolean constraint solving techniques of the ASP solver. Line 77 adjusts the `late` atoms to the topological ordering. In our example, this means that train line t_1 being late 1 min on node 11 uses the same variable as being late 1 min on node 12 thus reducing the amount of variables that are needed to capture delay. Finally, we derive difference constraint atoms expressing this information (lines 79–82), and ultimately using the regular atoms in a standard minimize statement (line 84). In detail, for every atom `potlate(`t`,`v`,`u`,`w`)`, an atom `late(`t`,`v`,`u`,`w`)` can be chosen if t visits v. For every `late(`t`,`v`,`u`,`w`)`, we derive `topo_late(`t`,`$h_t(v)$`,`u`,`w`)`. If `topo_late(`t`,`$h_t(n)$`,`u`,`w`)` is chosen to be true, a difference constraint atom `&diff{0-(`t`,`$h_t(v)$`)}<=` $-u$ is derived expressing $(t, h_t(v)) \geq u$ and, therefore, that t is delayed at the visited node v at threshold u. Otherwise, `&diff{(`t`,`$h_t(v)$`)-0}<=` $u-1$ becomes true so that $(t, h_t(v)) < u$ holds. The difference constraint atoms ensure that if the truth value of a `late` atom is decided, the schedule has to reflect this information. The minimize statement then sums up and minimizes the penalties of the `late` atoms that are true.

Finally, line 85 shows the straightforward encoding of the routing penalty minimization. The minimize statement merely collects the paths of the train lines, sums up their penalties, and minimizes this sum. This is done on a lower level than delay minimization leading to lexicographic optimization minimizing delay first and route penalty second.

4.5 Optional constraints and heuristics

While the above encoding can be used as is, additional constraints can be added to further restrict the search space in the hope of improving solving performance. To this end, we rely on the fact that *clingo*[DL] consists of a Boolean search engine (*clingo*) and a dedicated difference constraints propagator. While some atoms are purely Boolean, others have a semantics in terms of difference constraints. In the following, we transfer the knowledge represented in these difference constraints (such as transitivity and acyclicity) back to the logic program, thereby improving the search in the ASP part of the problem.

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

Finally, a domain-specific heuristic is represented that prefers sequences of train lines reducing the likelihood of conflicts.

4.5.1 Resource overlap I

Resource areas with different resources overlap if both contain at least one common edge. If two train lines are each routed over such an overlap, the sequence of entering the respective pairs of resource areas has to be the same since it is not possible for trains to overtake each other throughout both overlapping resource areas. We exploit this by detecting such overlaps and using them to add constraints that remove solution candidates for which the sequences do not match, thus reducing the search space. More precisely, given two resources r and r', train lines t and t', and resource areas $A \in \mathcal{A}_r^t$, $A' \in \mathcal{A}_{r'}^t$, $B \in \mathcal{A}_r^{t'}$, and $B' \in \mathcal{A}_{r'}^{t'}$, if train line t leaves resource area A before train t' enters its resource area B, t visits an edge $(v, v') \in A \cap A'$ and t' visits and edge from $(u, u') \in B \cap B'$ (i.e. both trains use the overlapping areas), then train t also has to leave resource area A' before train t' enters resource area B'.

As an example, we use a straight line of three tracks, where nodes are ordered adjacently from left to right. Consider train lines $t_a = (S, L, e, l, w)$, $t_b = (S, L, e', l', w')$, and $t_c = (S, L', e'', l'', w'')$ over nodes $S = \{1, 2, 3, 4\}$ and edges $L = \{(1, 2), (2, 3), (3, 4)\}$, and $L' = \{(4, 3), (3, 2), (2, 1)\}$. Furthermore, let $a(r_l) = \{(1, 2), (2, 3), (3, 2), (2, 1)\}$ and $a(r_r) = \{(2, 3), (3, 4), (4, 3), (3, 2)\}$, meaning that r_l forms a resource area covering the left and middle part of the graph while r_r forms a resource area covering the middle and right part of the graph. The resources overlap in the middle on edges $(2, 3)$ and $(3, 2)$. Consider train line t_a entering resource area $\{(1, 2), (2, 3)\}$ via edge $(1, 2)$, using resource r_l, before train line t_b enters this edge; it immediately follows that t_a also has to enter edge $(2, 3)$ before train line t_b. Given that edge $(2, 3)$ is also in the resource area $\{(2, 3), (3, 4)\}$, it is impossible for train line t_b to overtake t_a, and the sequence of entering these resource areas is the same. The same holds for train lines t_a and t_c. Once t_a enters resource area $\{(1, 2), (2, 3)\}$ before t_c enters resource area $\{(3, 2), (2, 1)\}$, it follows that t_a also has to enter resource area $\{(2, 3), (3, 4)\}$ before train line t_c can enter resource area $\{(4, 3), (3, 2)\}$, as both trains cannot get past each other because their respective resource areas overlap.

```
86  ra_overlap(T,R,A,R',A') :-
87      ra(T,R,A,(V,V')), ra(T,R',A',(V,V')), R != R'.
88  :- seq(T,T',R,A,A'), not seq(T,T',R',B,B'),
89      ra_overlap(T,R,A,R',B), ra_overlap(T',R,A',R',B'),
90      1 #sum {1 : route(T,(V,V')),
91                  ra(T,R,A,(V,V')),   ra(T,R',B,(V,V')),
92                  route(T',(U,U')),
93                  ra(T',R,A',(U,U')), ra(T',R',B',(U,U'))},
94      1 #sum {1 : shared(T,T',R',B,B');
95              1 : decided(T,T',R',B,B');
96              1 : decided(T',T,R',B',B)}.
```

Listing 5. *Optional overlap constraints (part I).*

In Listing 5, we first compute pairs of resource areas that overlap in lines 86 and 87 by deriving atoms `ra_overlap(`t`,`r`,`a`,`r'`,`a'`)` representing that for a train line t, resource areas $A \in \mathcal{A}_r^t$ and $A' \in \mathcal{A}_{r'}^t$ share at least one edge for distinct resources r and r'. Note

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

that a and a' are used as identifiers for resource areas A and A', respectively. The integrity constraint in lines 88–96 enforces that train lines routed over an overlap have the same sequence over both pairs of resource areas. Lines 90–93 require that both train lines visit at least one edge that is shared by the overlapping resource areas. We ensure that the sequence is only enforced, if a resource conflict is possible between these two train lines in lines 94–96. The predicate `decided` is used to extend this constraint to other cases, where we can exploit overlap to reduce the search space. We examine such a case in the following section.

4.5.2 Resource overlap II

We improve the optional overlap constraint from Listing 5 in Listing 6 by additionally considering sequences over resource areas which can be determined before search.

```
 97  decided(T,T',R,A,A') :-
 98     l_ra(T,R,A,L), e_ra(T',R,A',E), T != T', L < E.
 99  decided(T,T',R,A,A') :-
100     ra(T,R,A,(V,V')),   set(T,(V,V')),  l(T,V',L), T != T',
101     ra(T',R,A',(U,U')), set(T',(U,U')), e(T',U,E), L < E.
102  seq(T,T',R,A,A')      :- decided(T,T',R,A,A').
```

Listing 6. *Optional overlap constraints (part II).*

Given a resource r, the sequence how two different trains $t = (S, L, e, l, w)$ and $t' = (S', L', e', l', w')$ leave and enter their respective resource areas A_r^t and $A_r^{t'}$ is already decided if either

1. the latest entry time of train line t into resource area A_r^t is before the earliest entry time of train line t' into resource area $A_r^{t'}$ (cf. lines 97 and 98), or
2. there exist two edges $(v, v') \in A_r^t$ and $(u, u') \in A_r^{t'}$ that are included in every possible path of train line t and t', respectively, where the latest entry time $l(v')$ is before the earliest entry time $e'(u)$ (cf. lines 99–101).

For the latter case, we use precomputed atoms `set(`t`,(`v`,`v'`))` to denote edges (v, v') that are already set to be in every path of a train line t. Reconsider our previous example with train lines $t_a = (S, L, e, l, w)$ and $t_b = (S, L, e', l', w')$ where $S = \{1, 2, 3, 4\}$ and $L = \{(1, 2), (2, 3), (3, 4)\}$. Given that $l(3) < e'(1)$, it is already decided that train line t_a leaves resource area $\{(1, 2), (2, 3)\}$ before t_b enters this resource area. By deriving this information in Listing 6, Listing 5 ensures that the same sequence is used for resource area $\{(2, 3), (3, 4)\}$.

No matter which case holds, we derive the atom `decided(`t`,`t'`,`r`,`a`,`a'`)` to state that train line t leaves resource area A_r^t before train line t' enters resource area ${A'}_r^{t'}$. Converting these atoms into sequences in line 102 enhances propagation of the rules in Listing 5, as these already decided sequences can be used in the integrity constraint in line 88 to derive the truth value of additional sequences. Note that since the additional instances of the `seq`/5 predicate serialize resource areas where the time intervals do not overlap, solutions of the encodings in Listings 2 and 3 remain unchanged.

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

4.5.3 Sequence acyclicity

Whenever more than two train lines have to be serialized on a resource, the resulting sequence can never be cyclic. In more detail, given a resource r and three train lines t_1, t_2, and t_3, if t_1 leaves resource area $A_r^{t_1}$ before t_2 enters resource area $A_r^{t_2}$, and t_2 leaves resource area $A_r^{t_2}$ before t_3 enters resource area $A_r^{t_3}$, it follows that t_1 leaves area $A_r^{t_1}$ before t_3 enters area $A_r^{t_3}$ as well. This is guaranteed by the constraints in condition (7) and realized by the difference constraint atoms in Listing 3, lines 59–62.

```
103  :- seq(T,T',R,A,A'), seq(T',T'',R,A',A''),
104     shared(T,T'',R,A,A''), not seq(T,T'',R,A,A'').
```

Listing 7. *Optional acyclicity constraints.*

While this condition would be eventually derived by the difference constraints propagator, we immediately restrict the search space by providing this explicit constraint, realized in Listing 7. Furthermore, by expressing this timing-related condition in terms of regular ASP atoms, we can leverage the propagation mechanism of state-of-the-art ASP solving technology, which is currently faster and more sophisticated compared to the one used in the difference constraints propagator.

4.5.4 Sequence heuristic

```
105  #heuristic seq(T,T',R,A,A') :
106     shared(T,T',R,A,A'),
107     e_ra(T,R,A,E), e_ra(T',R,A',E'),
108     l_ra(T,R,A,L), l_ra(T',R,A',L'). [E'-E - (L-L'),sign]
```

Listing 8. *Heuristic that orders conflicting train lines by their possible arrival times.*

The heuristic in Listing 8 attempts to order conflicting train lines by their possible arrival times at the areas where the conflict is located. In essence, we analyze how the time intervals of the train lines are situated and prefer their sequence accordingly. Given two train lines t and t' with intervals $[e, l]$ and $[e', l']$ at the conflicting areas, respectively, we calculate $s = e' - e - (l - l')$ to determine whether t should be scheduled before t'. If s is positive, the preferred sign of the sequence atom is also positive, thus preferring t to go before t'; if it is negative, the opposite is expressed. In detail, $e' - e$ is positive if t' may arrive later than t thus making it more likely that t can go first without delaying t'. Similarly, if $l - l'$ is negative, t' may leave later, suggesting t to go first. If the results of both expressions have the same sign, one interval is contained in the other and if the difference is positive, the center of the interval of t is located earlier than the center of the interval of t'. For example, in Figure 2, we see that t_1 and t_2 share resource sw_1 in node 3 and the time intervals in which they potentially arrive at those edges are $[300, 600]$ and $[180, 540]$, respectively. Given that $180 - 300 - (600 - 540) = -180$, we prefer t_2 to be scheduled before t_1, which in the example is clearly the correct decision, since t_2 precedes t_1 without delaying t_1.

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

5 Experiments

We evaluate our train-scheduling solution using the hybrid ASP system *clingo*[DL] v1.1, which is built upon the API of *clingo* 5.3.[5] As benchmark set, we use 25 real-world instances crafted by domain experts at Swiss Federal Railways (SBB). The instances capture parts of the railway network between three Swiss cities, namely, Zürich, Chur, and Luzern, and vary in number of train lines, size and depths of railway network, and timing constraints. The biggest instances contain the whole railway network and up to 467 train lines taken from long distance, regional, suburban, and freight traffic between these three cities. Thus, we tackle instances with approximately 6 h of the full train schedule on a railway network covering approximately 200 km. For brevity, we omit slight grounding and propagation optimizations in the encoding presented in Section 4; the full encoding and instance set is available at https://github.com/potassco/train-scheduling-with-hybrid-asp/. We use the best optimization configuration determined in Abels *et al.* (2019). In detail, we use two threads, one running with a model-guided optimization strategy iteratively producing models of descending cost until the optimum is found, and the other with a core-guided optimization strategy Andres *et al.* (2012) relying on successively identifying and relaxing unsatisfiable cores until a model is obtained. Furthermore, the delay minimization is modeled by setting $D_{t,v} = Bin_{t,v} \cup Lin_{t,v}^{180}$ for each train line t at node v; thus, there are six thresholds in total, one if there is any delay at all, and five with a distance of 180 s in between with the last one being at 15 min. We use lexicographic optimization minimizing delay first and route penalty second.

As a preliminary step, we benchmark each domain-specific heuristic constraints individually, and removed the ones that did not yield any improvement compared to *clingo*[DL]'s default setting.[6] This might be either due to the fact that the search space was too drastically reduced and as such less impact can be achieved by the branching heuristics, or that the switch to lexicographic optimization made some heuristics obsolete.

In total, we examined four optional encoding parts, one domain-specific heuristic and three optional constraints, and all their combinations. More precisely, we consider the sequence heuristic (HS), resource overlap I (OL1), resource overlap II (OL2), and sequence acyclicity (AC), and all seven possible combinations (given that OL1 is contained in OL2). We compare the results to *clingo*[DL]'s default settings without optional encoding parts and domain-specific heuristic (DEF).

All benchmarks ran on Linux with a Xeon E3-1260L quad-core 2.9 GHz processors and 32 GB RAM; each run limited to 3 h and 32 GB RAM. For all our experiments, we asked *clingo*[DL] to return one optimal solution, and we then validate feasibility and quality via an external tool provided by SBB.[7]

Table 1 shows the average runtime (T) over all 25 instances, which includes grounding and solving, grounding time (GT), choices (CH), and conflicts (CO) for individual domain-specific heuristics constraints; their combinations are given in Table 2. The best results in a row are marked bold. Note that for all configurations and instances, *clingo*[DL] was

[5] Both are development versions available on https://github.com/potassco/clingoDL,clingo/ with commit hashs `#f1a185e` and `#f8a51134`, respectively.
[6] In particular, we dropped two heuristics proposed in Abels *et al.* (2019) since they did not improve solving performance with the new resource area and height-based encoding.
[7] https://github.com/potassco/train-scheduling-with-hybrid-asp/.

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

Table 1. *Average runtime* (T), *grounding time* (GT), *choices* (CH), *and conflicts* (CO) *for individual domain-specific heuristics and optional constraints*

CONF	T	GT	CH	CO
DEF	550	**45**	242,516,740	9135
HS	162	**45**	51,933,286	7363
OL1	118	53	13,003,796	4038
OL2	**105**	62	**8,377,540**	**3530**
AC	435	46	116,873,540	8755

Table 2. *Average runtime* (T), *grounding time* (GT), *choices* (CH) *and centering conflicts* (CO) *for combinations of domain-specific heuristics and optional constraints*

CONF	T	GT	CH	CO
HS/OL1	72	53	2,877,997	3097
HS/OL2	77	63	2,097,689	2606
HS/AC	146	**46**	29,245,280	6814
OL1/AC	99	55	5,934,253	3905
OL2/AC	93	64	3,618,248	3293
HS/OL1/AC	**71**	55	1,884,225	2706
HS/OL2/AC	76	64	**1,486,261**	**2525**
VBS	68	53	2,383,874	2560

able to find an optimal solution, thus, no time- or memory-outs blur average time as a means for comparing the performance of the different configurations. Grounding the largest instances within the time limit is enabled by our preprocessing and the height-based naming scheme for the integer variables, due to their significant impact on reducing grounding size.

We clearly see in Table 1 that OL2 by itself improves solving performance the most, drastically reducing choices and conflicts compared to DEF and displaying the lowest average runtime overall. This is closely followed by OL1. The heuristic HS still clearly improves the solving performance while optional constraint AC only slightly improves upon the default. We can observe that grounding time is higher for all optional constraints, especially for OL2. This is to be expected as additional rules and integrity constraint are added to the encoding. The improvement in solving performance through the reduced search space vastly outweighs the decrease in grounding performance though.

As for the composite configurations in Table 2, we observe that all of them are an improvement over the individual heuristics and optional constraints. The largest combinations also yield the best performance overall. Here, we see that, while HS/OL2/AC traverses the search space most effectively (yielding least choices and conflicts), HS/OL1/AC has a slightly better runtime performance due to lower grounding time. This shows, first, that the combination HS/OL1/AC alleviates some weaknesses compared to the individual

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

Table 3. *Detailed information and best results for all benchmark instances*

INS	#TL	#R	#SR	#RTL	#RA	#EC	#RAC	#VNN	#VHN	T	AQU	QU
P1	4	115	63	908	134	537	36	318	294	0	(0,0)	(0,0)
P2	58	530	205	15,180	3132	21,090	2875	4409	4371	4	(0,0)	(0,0)
P3	143	539	205	28,754	6082	64,766	9158	8759	8662	7	(0,0)	(0,0)
P4	148	553	207	34,461	6810	106,749	13,745	9425	9114	9	(0,1)	(0,1)
P5	149	553	207	34,482	6814	107,109	13,802	9430	9119	11	(9,6)	(27,6)
P6	365	567	210	175,722	25,021	912,840	44,908	38,941	28,099	53	(0,0)	(0,0)
P7	467	567	210	232,827	33,934	1,591,476	82,562	52,060	38,119	89	(0,0)	(0,0)
P8	133	528	195	102,456	12,454	1,086,507	34,996	21,559	12,277	43	(0,0)	(0,0)
P9	287	568	192	159,316	18,862	4,180,212	103,225	34,488	18,513	179	(0,0)	(0,0)
P9R	83	505	188	77,716	7541	1,998,885	28,572	14,579	6591	44	(0,0)	(0,0)
P9RI	83	505	188	77,716	7541	1,316,502	18,616	14,579	6591	35	(0,0)	(0,0)
TS	131	915	303	42,590	8026	263,278	25,036	12,199	10,695	16	(0,0)	(0,0)
TSE	132	915	303	43,011	8105	274,020	25,934	12,303	10,786	18	(≈ 0,0)	(6,0)
TL1	447	920	301	126,271	24,068	896,199	87,094	37,163	32,533	55	(0,0)	(0,0)
TL1E	448	920	301	126,692	24,147	907,059	88,003	37,267	32,624	67	(≈ 0,0)	(11,0)
TL1EI	448	920	301	126,692	24,147	187,393	20,612	37,267	32,624	41	(≈ 0,0)	(11,0)
TL2	448	952	272	183,865	29,531	494,446	30,961	51,707	32,781	116	(≈ 0,0)	(≈ 0,0)
TL3	448	952	272	183,865	29,531	488,645	30,737	51,707	32,781	111	(0,0)	(0,0)
TL4	448	940	271	165,909	28,442	359,927	28,115	47,595	32,762	87	(≈ 0,0)	(1,0)
TL5	448	940	271	165,909	28,442	357,267	28,045	47,595	32,762	86	(0,0)	(0,0)
TL6	448	925	290	131,691	25,058	197,830	21,663	38,494	32,752	41	(≈ 0,0)	(≈ 0,0)
TL7	448	925	290	131,691	25,058	197,148	21,544	38,494	32,752	42	(0,0)	(0,0)
TL8	448	955	272	216,830	31,805	871,732	40,851	58,722	32,781	177	(0,0)	(0,0)
TL9	448	955	272	216,830	31,805	863,326	40,535	58,722	32,781	173	(0,0)	(0,0)
TL10	451	952	274	185,912	29,687	1,216,327	69,972	52,184	33,088	177	(≈ 0,75)	(10,75)
			34%		83%		92%		25%			

configurations, as OL1 alone was slower than OL2. And second, that both composite configurations might prove useful in the future since for harder instances the improvement in terms of the search might outweigh the decrease in grounding performance. The last row VBS displays the virtual best solver that averages the best performance regarding runtime among all configurations. We see that the performance of VBS is close to the best configurations and as such choosing one among HS/OL1/AC, HS/OL2/ACand HS/OL1 should be a good choice as default configuration for most instances.

In the following, we analyze the 25 real-world instances in detail and highlight the best results that we could achieve using the different configurations of our train-scheduling solution. This information is presented in Table 3. Row INS contains the names of the 25 instances. Instance names starting with P are the nine instances originally used in Abels *et al.* (2019) that were published by SBB; they only contain part of the railway network between Zürich, Chur, and Luzern. Two instances among them are marked with R and are reduced versions of the largest instance, viz., P9. Instances starting with T, on the other hand, cover the whole test area between the three cities. Furthermore, instance names containing S and L cover a shorter (about 2 h) or longer time span (about 6 h), respectively, which is reflected in the amount of train lines. With our problem formulation,

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

one can express different settings of the train-scheduling problem. Normally, all train lines are considered as new and to be taken into account evenly. Another problem variation is the rescheduling after adding or changing one extra train line. This is achieved by closely restricting earliest and latest arrival times around the old schedule for existing and unchanged train lines, while the extra train line is given more freedom. Instances containing an E are such rescheduling instances. Similarly, most instances have a uniform time span between the point after which a train line is delayed and the latest possible arrival time. Unlike this, for names including I, this time span is different for each train line, that is, different train lines have different capacities for delay. Instances of the same class mostly vary in the depths of the railway network, that is, how many alternative routes are available to each train line, but also slightly in number of train lines, resources and timing constraints.

The following nine columns analyze structural attributes of the instances, and how much preprocessing and encoding techniques reduce the instance size and search space. Column #TL shows the number of train lines for each instance. Column #R gives the number of resources, and Column #SR the number of subsumed resources. Subsumed resources can be observed in all instances and constitute on average 34%, which can then be safely removed. Column #RTL sums up resources on each edge in each subgraph of all train lines, which essentially constitutes the basis for resource conflicts using the edge-based conflict resolution. Column #RA then shows into how many resource areas those individual edges could be distributed. The difference from #RTL to #RA is on average 83%, thus drastically reducing resource entities that might induce conflicts. The following columns #EC and #RAC display the number of resource conflicts based on single edges and resource areas, respectively. They highlight the benefit of introducing resource areas. We reduce the number of resource conflicts, and thus the decisions that have to be made to serialize the train lines, on average by 92%. Finally, #VNN and #VHN shows the number of integer variables needed using node- and height-based naming scheme, respectively. We save 25% of variables on average switching to the height-based naming scheme.

In the last three columns, we present the virtual best results regarding runtime across all configurations presented in Tables 1 and 2. In more detail, we show the runtime per instance in Column T, and the approximated and exact quality as a pair of delay penalty in minutes and route penalty in Columns AQU and Column QU, respectively. The approximated quality amounts to the value returned by the ASP system, which is optimal for all instances, while the exact quality is returned by a tool of SBB. We were able to solve all instances with a maximum time of about 3,min. We calculated solutions without any delay for all instances where this was possible. For instances where this was not the case, experts at SBB deemed the incurred delay as close to optimal. Of course, we are not able to prove true optimality in all cases since we use an approximation of the delay function. Note that, for example, TL10, we have a rather high difference between approximated delay and actual delay of about 11,min. This is due to the fact that the ASP system only considers a lower bound on delay. If the approximated optimal value is for example 3 s, this could mean that there are three instances of delay. The amount of delay though could be as much as $3 \times 179 = 537$ s since the next threshold is at 180 s in our configuration of the approximation. Completely avoiding this drawback of the approximation is currently only possible by introducing a threshold for every second, which deteriorates solving performance. We judge our current solution to be a good

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

trade-off between solving performance and quality, since the quality of the results is sanctioned by Swiss Federal Railways and we are able to solve all real-world instances within minutes.

6 Discussion

At its core, train scheduling is similar to classical scheduling problems that were already tackled by ASP. Foremost, job shop scheduling (Taillard 1993) is also addressed by *clingo*[DL] and compared to other hybrid approaches in Janhunen *et al.* (2017); solutions based on Satisfiability Modulo Theory, Constraint Programming (CP), and Mixed Integer Linear Programming (MILP) are given in Janhunen *et al.* (2011), Bofill *et al.* (2012), Baptiste *et al.* (2012), and Liu *et al.* (2012). In fact, job shop scheduling can be seen as a special case of our setting, in which train paths are known beforehand. Solutions to this restricted variant using MILP and CP are presented in Oliveira and Smith (2000) and Rodriguez (2007). The difference to our setting is twofold: first, resource conflicts are not known beforehand since we take routing and scheduling simultaneously into account. Second, our approach encompasses a global view of arbitrary precision, that is, we model all routing and scheduling decisions across hundreds of train lines down to inner-station conflict resolution and allow for expressing complex connections that also accommodate cyclic train movements. Furthermore, using hybrid ASP with difference constraints gives us inherent advantages over pure ASP and MILP. First, we show in Gebser *et al.* (2016) that ASP is not able to solve most shop scheduling instances since grounding all integer variables leads to an explosion in problem size. We avoid this bottleneck by encapsulating scheduling in difference constraints and, hence, avoid grounding integer variables. Second, while difference constraints are less expressive than linear constraints in MILP, they are sufficient for expressing the timing constraint needed for train scheduling while being solvable in polynomial time. Finally, routing and conflict resolution require Boolean variables and disjunctions for which ASP has effective means.

In this work, we contribute a flexible and holistic ASP-based encoding of the train-scheduling problem based on a precise formalization. On the one hand, since we produce timetables from scratch, our train-scheduling approach can be characterized as tactical scheduling (Törnquist 2006). On the other hand, without changing encoding or problem formalization, we can easily accommodate problem variations like rescheduling without losing performance, as we show for three instances in our empirical evaluation.

As seen in Section 5, all instances available to us can be solved within minutes. These instances represent sections of a test area in Switzerland, with the largest one capturing a complex, interconnected railway network spanning in length about 200 km and involving up to 467 train lines with local, regional, and cargo railway traffic among them. This is possible in part due to dedicated preprocessing techniques, such as resource subsumption and resource areas, that exploit structural redundancies within the railway network as well as graph compressing methods, which drastically reduce the size of the problem representation. Furthermore, we reduce the search space and leverage advanced propagation mechanisms of our state-of-the-art ASP solving technology by transferring implicit knowledge about difference constraints into the logic program. Some of these techniques may also be candidates to improve performance for other scheduling problems like the related job shop scheduling.

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

To advance our approach, multi-shot solving (Gebser *et al*. 2019) can be used to incrementally increase train line intervals or dynamically replan schedules by adding or removing train lines. While our hybrid ASP approach already tackles some real-world instances that are on the larger side, scalability, for example, to the entirety of Switzerland is still an issue. For that purpose, we are looking into decomposition techniques, so that smaller areas using our approach can be combined to solutions for entire countries.

References

Abels, D., Jordi, J., Ostrowski, M., Schaub, T., Toletti, A. and Wanko, P. 2019. Train scheduling with hybrid ASP. In *Proceedings of the Fifteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'19)*, M. Balduccini, Y. Lierler and S. Woltran, Eds. Lecture Notes in Artificial Intelligence, Vol. 11481. Springer-Verlag, 3–17.

Andres, B., Kaufmann, B., Matheis, O. and Schaub, T. 2012. Unsatisfiability-based optimization in clasp. In *Technical Communications of the Twenty-eighth International Conference on Logic Programming (ICLP'12)*, A. Dovier and V. Santos Costa, Eds., Vol. 17. Leibniz International Proceedings in Informatics (LIPIcs), 212–221.

Baptiste, P., Pape, C. L. and Nuijten, W. 2012. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*, Vol. 39. Springer Science+Business Media.

Bofill, M., Palahí, M., Suy, J. and Villaret, M. 2012. Solving constraint satisfaction problems with sat modulo theories. *Constraints 17,* 3, 273–303.

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

Gebser, M., Harrison, A., Kaminski, R., Lifschitz, V. and Schaub, T. 2015a. Abstract Gringo. *Theory and Practice of Logic Programming 15,* 4–5, 449–463. https://arxiv.org/abs/1507.06576.

Gebser, M., Kaminski, R., Kaufmann, B., Lindauer, M., Ostrowski, M., Romero, J., Schaub, T. and Thiele, S. 2015b. *Potassco User Guide*, 2nd ed. University of Potsdam.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T. and Wanko, P. 2016. Theory solving made easy with clingo 5. In *Technical Communications of the Thirty-Second International Conference on Logic Programming (ICLP'16)*, M. Carro and A. King, Eds. OpenAccess Series in Informatics (OASIcs), Vol. 52. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2:1–2:15.

Gebser, M., Kaminski, R., Kaufmann, B. and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming 19,* 1, 27–82.

Gebser, M., Kaufmann, B., Otero, R., Romero, J., Schaub, T. and Wanko, P. 2013. Domain-specific heuristics in answer set programming. In *Proceedings of the Twenty-Seventh National Conference on Artificial Intelligence (AAAI'13)*, M. des Jardins and M. Littman, Eds. AAAI Press, 350–356.

Janhunen, T., Kaminski, R., Ostrowski, M., Schaub, T., Schellhorn, S. and Wanko, P. 2017. Clingo goes linear constraints over reals and integers. *Theory and Practice of Logic Programming 17,* 5-6, 872–888.

Janhunen, T., Liu, G. and Niemelä, I. 2011. Tight integration of non-ground answer set programming and satisfiability modulo theories. In *Proceedings of the First Workshop on Grounding and Transformation for Theories with Variables (GTTV'11)*, P. Cabalar, D. Mitchell, D. Pearce and E. Ternovska, Eds. 1–13.

Lifschitz, V. 1999. Answer set planning. In *Proceedings of the International Conference on Logic Programming (ICLP'99)*, D. de Schreye, Ed. MIT Press, 23–37.

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

Liu, G., Janhunen, T. and Niemelä, I. 2012. Answer set programming via mixed integer programming. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'12)*, G. Brewka, T. Eiter and S. McIlraith, Eds. AAAI Press, 32–42.

Oliveira, E. and Smith, B. 2000. A job-shop scheduling model for the single-track railway scheduling problem. LU SCS RR 21, School of Computer Studies, University of Leeds.

Rodriguez, J. 2007. A constraint programming model for real-time train scheduling at junctions. *Transportation Research Part B: Methodological 41,* 2, 231–245.

Taillard, E. 1993. Benchmarks for basic scheduling problems. *European Journal of Operational Research 64,* 2, 278–285.

Törnquist, J. 2006. Computer-based decision support for railway traffic scheduling and dispatching: A review of models and algorithms. In *Proceedings of Fifth Workshop on Algorithmic Methods and Models for Optimization of Railways (ATMOS'05)*, L. G. Kroon and R. H. Möhring, Eds. OpenAccess Series in Informatics (OASIcs), Vol. 2. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Appendix: Example instance

```
tl(t1). tl(t2). tl(t3).

edge(t1,1,3).  edge(t1,2,3).   edge(t1,3,5).  edge(t1,5,8).
edge(t1,8,10). edge(t1,10,11). edge(t1,10,12).
edge(t2,10,7). edge(t2,7,4).   edge(t2,4,3).
edge(t3,3,6).  edge(t3,6,9).   edge(t3,9,10). edge(t3,10,12).

e(t1,1,240).  l(t1,1,540).  e(t1,2,240).  l(t1,2,540).
e(t1,3,300).  l(t1,3,600).  e(t1,5,360).  l(t1,5,660).
e(t1,8,420).  l(t1,8,720).  e(t1,10,480). l(t1,10,780).
e(t1,11,540). l(t1,11,840). e(t1,12,540). l(t1,12,840).
start(t1,1).  start(t1,2).  end(t1,11).   end(t1,12).

e(t2,10,0).   l(t2,10,360). e(t2,7,60).  l(t2,7,420).
e(t2,4,120).  l(t2,4,480).  e(t2,3,180). l(t2,3,540).
start(t2,10). end(t2,3).

e(t3,3,180).  l(t3,3,540). e(t3,6,240).  l(t3,6,600).
e(t3,9,300).  l(t3,9,660). e(t3,10,360). l(t3,10,720).
e(t3,12,420). l(t3,12,780).
start(t3,3).  end(t3,12).

potlate(t1,1,451,1).  potlate(t1,2,451,1). penalty((1,3),1).
potlate(t2,10,241,1). potlate(t3,3,421,1).

resource(sw1,(1,3)).   resource(sw1,(2,3)).
resource(sw1,(4,3)).   resource(sw1,(3,5)).
resource(sw1,(3,6)).   resource(sw2,(8,10)).
resource(sw2,(9,10)).  resource(sw2,(10,7)).
resource(sw2,(10,11)). resource(sw2,(10,12)).
```

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

```
connection(1,t2,(4,3),t3,(3,6),0,0,3,3).
free(1,t2,(4,3),t3,(3,6),sw1).
connection(2,t1,(10,11),t3,(10,12),60,#inf,10,12).
connection(3,t1,(10,12),t3,(10,12),60,#inf,10,12).

set(t1,(3,5)).   set(t1,(5,8)). set(t1,(8,10)).
set(t2,(10,7)). set(t2,(7,4)). set(t2,(4,3)).
set(t3,(3,6)).   set(t3,(6,9)). set(t3,(9,10)).
set(t3,(10,12)).

resource(r(1,3),(1,3)). resource(r(2,3),(2,3)).
resource(r(3,5),(3,5)). resource(r(5,8),(5,8)).
resource(r(7,4),(7,4)). resource(r(4,3),(4,3)).
resource(r(3,6),(3,6)). resource(r(6,9),(6,9)).

ra(t1,sw1,0,(1,3)).     ra(t1,r(1,3),0,(1,3)).
ra(t1,sw1,0,(2,3)).     ra(t1,r(2,3),0,(2,3)).
ra(t1,sw1,0,(3,5)).     ra(t1,r(3,5),0,(3,5)).
ra(t1,r(5,8),0,(5,8)). ra(t1,sw2,0,(8,10)).
ra(t1,sw2,0,(10,11)).   ra(t1,sw2,0,(10,12)).
ra(t2,sw2,0,(10,7)).    ra(t2,r(7,4),0,(7,4)).
ra(t2,sw1,0,(4,3)).     ra(t2,r(4,3),0,(4,3)).
ra(t3,sw1,0,(3,6)).     ra(t3,r(3,6),0,(3,6)).
ra(t3,r(6,9),0,(6,9)). ra(t3,sw2,0,(9,10)).
ra(t3,sw2,0,(10,12)).

potlate(t1,3,511,1).   potlate(t1,5,571,1).
potlate(t1,8,631,1).   potlate(t1,10,691,1).
potlate(t1,11,751,1). potlate(t1,12,751,1).
potlate(t2,7,301,1).   potlate(t2,4,361,1).
potlate(t2,3,421,1).   potlate(t3,6,481,1).
potlate(t3,9,541,1).   potlate(t3,10,601,1).
potlate(t3,12,661,1).

l_ra(t1,sw1,0,660).     l_ra(t1,r(1,3),0,600).
l_ra(t1,r(2,3),0,600). l_ra(t1,r(3,5),0,660).
l_ra(t1,r(5,8),0,720). l_ra(t1,sw2,0,840).
l_ra(t2,sw2,0,420).     l_ra(t2,r(7,4),0,480).
l_ra(t2,sw1,0,540).     l_ra(t2,r(4,3),0,540).
l_ra(t3,sw1,0,600).     l_ra(t3,r(3,6),0,600).
l_ra(t3,r(6,9),0,660). l_ra(t3,sw2,0,780).
e_ra(t1,sw1,0,240).     e_ra(t1,r(1,3),0,240).
e_ra(t1,r(2,3),0,240). e_ra(t1,r(3,5),0,300).
e_ra(t1,r(5,8),0,360). e_ra(t1,sw2,0,420).
e_ra(t2,sw2,0,0).       e_ra(t2,r(7,4),0,60).
e_ra(t2,sw1,0,120).     e_ra(t2,r(4,3),0,120).
```

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press

```
e_ra(t3,sw1,0,180).     e_ra(t3,r(3,6),0,180).
e_ra(t3,r(6,9),0,240). e_ra(t3,sw2,0,300).

b(sw1,60).     b(sw2,60).     b(r(1,3),60). b(r(2,3),60).
b(r(3,5),60). b(r(5,8),60). b(r(7,4),60). b(r(4,3),60).
b(r(3,6),60). b(r(6,9),60).

w(t1,(1,3),0).  w(t1,(2,3),0).   w(t1,(3,5),0).  w(t1,(5,8),0).
w(t1,(8,10),0). w(t1,(10,11),0). w(t1,(10,12),0).
w(t2,(10,7),0). w(t2,(7,4),0).   w(t2,(4,3),0).
w(t3,(3,6),0).  w(t3,(6,9),0).   w(t3,(9,10),0). w(t3,(10,12),0).

m((1,3),60).   m((2,3),60).   m((3,5),60).   m((5,8),60).
m((8,10),60).  m((10,11),60). m((10,12),60). m((10,7),60).
m((7,4),60).   m((4,3),60).   m((3,6),60).   m((6,9),60).
m((9,10),60).
```

Listing 9. *Facts representing example instance (Figures 1 and 2).*

https://doi.org/10.1017/S1471068420000046 Published online by Cambridge University Press