Flaspland: A Testbed for Routing and Scheduling in Answer Set Programming (Preliminary Report)

Ryan Murphy[1], Amadé Nemes[1], Javier Romero[1] and Torsten Schaub[1,2]

[1]University of Potsdam, Germany
[2]Potassco Solutions GmbH, Germany

Abstract

We present *flaspland*, an Answer Set Programming (ASP) testbed for tackling complex routing and scheduling problems within the Flatland railway simulation environment. To establish a solid theoretical foundation, we provide a rigorous formalization of the Flatland environment, detailing its grid-based topology and navigational dynamics. Building upon this, we propose and analyze three distinct graph-based representations to effectively encode the underlying constraints. Crucially, we introduce a twofold, path-driven ASP encoding scheme that decouples spatial routing from temporal scheduling. Our empirical results demonstrate that these specialized representations outperform off-the-shelf multi-agent pathfinding approaches. By providing a testbed for real-world railway scheduling, *flaspland* facilitates the evaluation of declarative approaches to qualitative and quantitative temporal constraints.

1. Introduction

The ability of Answer Set Programming (ASP [1]) to easily express reachability makes it a prime candidate for addressing routing problems, such as multi-agent path finding [2], phylogenetic inference [3], and wire routing [4]. However, this advantage is challenged once routing is combined with scheduling to enforce deadlines and durations, since fine-grained quantitative temporal reasoning cannot be feasibly represented in ASP.

To address this, we propose railway scheduling as a rich and scalable benchmark for exploring routing and scheduling problems in ASP and related fields. We specifically advocate for Flatland [5], a grid-based railway simulation environment developed by major European railway operators, which enables scalable experimentation with complex railway scenarios. The core task is to solve collision-free routing and scheduling problems, which are essentially multi-agent pathfinding problems (MAPF; [6]) augmented with quantitative constraints on train arrival and departure times, durations, energy, and speed.

The railway network is modeled as a set of connected cells using matrix notation, as shown in Figure 1. The traversability of a cell is defined by its specific track type (see Figure 2), which dictates valid transitions and may be oriented in 90-degree increments. The overall objective is to navigate all trains to their destinations, respecting strict time windows and preventing collisions. Navigation is performed via discrete actions, whose executability is constrained by the train's current direction and the underlying track configuration.

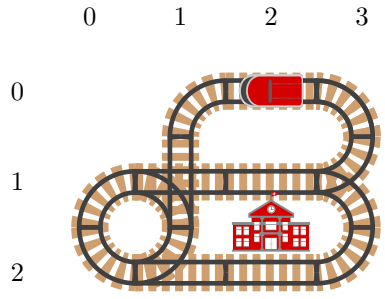


Figure 1: Visualization of a simple Flatland scenario with one train station and one train

Beyond quantitative scheduling constraints, such railway networks present structural challenges that differentiate it from standard MAPF: (i) Track types strictly limit valid cell transitions, requiring precise modeling of graph connectivity. (ii) A train's ability to move depends on

Joint Proceedings of the Workshops and Doctoral Consortium of the 42nd International Conference on Logic Programming (ICLP 2026) co-located with the Federated Logic Conference (FLoC 2026).

ryan.murphy@uni-potsdam.de (R. Murphy); balazs.amade.nemes@uni-potsdam.de (A. Nemes); javier.romero.davila@uni-potsdam.de (J. Romero); torsten.schaub@uni-potsdam.de (T. Schaub)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-03

CEUR-WS.org/Vol-4311/ASPOCP_regular_04.pdf

its current orientation, adding another layer of complexity to path planning. (iii) The network topology often features canalized passages with limited switching points, restricting navigational choices and creating bottlenecks.

In what follows, we provide a formal account of Flatland, detailing its environment, invariants, and path definitions. Although Flatland's tasks are intuitively well understood, a rigorous formalization is necessary to establish the correspondence between railway grids and their graph-based representations, and ultimately to ensure the correctness and completeness of the ASP encodings. We then propose alternative graph-based representations of railway networks and establish their correspondence to the original grid environment. To facilitate empirical evaluation, we introduce the *flaspland* toolkit, which bridges ASP and the Flatland environment. Finally, we present a novel two-fold, path-driven ASP encoding scheme. Our preliminary empirical evaluation demonstrates that these specialized representations offer significant scalability improvements over off-the-shelf MAPF approaches.

2. Flatland formalization

To begin with, we present some crosscutting concepts. Flatland considers a set $D = \{n, e, s, w\}$ of cardinal *directions* and a set T of *track types*.

We define the following operations on directions to compute their opposite and 90-degree rotations:

1. $\overline{(\cdot)} : D \to D$ where $\overline{(\cdot)} = \{\overline{n} \mapsto s, \overline{s} \mapsto n, \overline{e} \mapsto w, \overline{w} \mapsto e\}$
2. $(\cdot)^r : D \to D$ where $(\cdot)^r = \{n^r \mapsto e, e^r \mapsto s, s^r \mapsto w, w^r \mapsto n\}$
3. $(\cdot)^l : D \to D$ where $(\cdot)^l = \{n^l \mapsto w, w^l \mapsto s, s^l \mapsto e, e^l \mapsto n\}$

Clearly, we have $\overline{\overline{d}} = d$, $d^{2r} = d^{2l} = \overline{d}$, and $d^{4r} = d^{4l} = d$ for $d \in D$.



Figure 2: Basic track types

Flatland's general track types are shown in Figure 2.[1] Type 0 is reserved for empty cells. Specific track types are derived by rotating these general types in 90-degree increments. For example, the simple switch at position $(2, 1)$ in Figure 1 corresponds to Type 2 rotated by $90°$, denoted as $(2, 90)$. Each track type $t \in T$ provides a set of *transitions*, captured by the relation $H_t \subseteq D \times D$ satisfying for all $d, d' \in D$:

1. $(d, d') \in H_t$ iff $(\overline{d'}, \overline{d}) \in H_t$
2. $(d, \overline{d}) \notin H_t$
3. $|\{(d, d'') \in H_t \mid d'' \in D\}| \leq 2$ and $|\{(d'', d) \in H_t \mid d'' \in D\}| \leq 2$

That is, every transition has a complementary one, no transition represents an immediate about-face, and there may be no more than two transitions for the same incoming and outgoing direction. For instance, we have $H_{(2,90)} = \{(e, e), (w, w), (e, n), (s, w)\}$.

Notation. *The directions D, track types T, and transitions $(H_t)_{t \in T}$ are constant across all Flatland environments and are therefore assumed fixed in the remainder of this work.*

Grids. In Flatland, a railway network is an $n \times m$ grid of cells C. We associate each *cell* $c \in C$ with a coordinate position and a track type, via the total functions:

1. $p : C \to [0, n-1] \times [0, m-1]$

[1] We omit track type 7, the dead end, to align with current versions of Flatland, which no longer produce environments with this type of track.

2. $t : C \to T$

We require p to be a bijective function. For the cell with the simple switch,[2] we have $p(c_{2,1}) = (2, 1)$ and $t(c_{2,1}) = (2, 90)$. The relation $H_{t(c)}$ provides the set of transitions for a cell $c \in C$, and we call c *trivial* whenever $H_{t(c)} = \emptyset$.

The cell adjacent to cell $c \in C$ in the direction $d \in D$ is defined by the partial function

3. $a : C \times D \to C$

such that

$$p(a(c,d)) = \begin{cases} (i-1,j) & \text{if } p(c) = (i,j),\ d = n, \text{ and } (i,j),(i-1,j) \in [0,n-1] \times [0,m-1] \\ (i,j+1) & \text{if } p(c) = (i,j),\ d = e, \text{ and } (i,j),(i,j+1) \in [0,n-1] \times [0,m-1] \\ (i+1,j) & \text{if } p(c) = (i,j),\ d = s, \text{ and } (i,j),(i+1,j) \in [0,n-1] \times [0,m-1] \\ (i,j-1) & \text{if } p(c) = (i,j),\ d = w, \text{ and } (i,j),(i,j-1) \in [0,n-1] \times [0,m-1] \end{cases}$$

Function a is partial because of border cells. If $a(c,d) \neq a(c,d')$ then $d \neq d'$ for all $c \in C$ and $d, d' \in D$. Moreover, we have $a(c,d) = c'$ iff $a(c', \overline{d}) = c$ for all $c, c' \in C$ and $d \in D$. For instance, we have $a(c_{2,1}, w) = c_{2,0}$ and $a(c_{2,0}, e) = c_{2,1}$, whereas $a(c_{2,1}, s)$ is undefined.

A non-trivial cell $c \in C$ is connected to another non-trivial cell $c' \in C$ in direction $d \in D$, written as $c \overset{d}{\hookrightarrow} c'$, when the following conditions hold:

1. $a(c,d) = c'$
2. $(d', d) \in H_{t(c)}$ for some $d' \in \{d, d^l, d^r\}$
3. $(d, d'') \in H_{t(c')}$ for some $d'' \in \{d, d^l, d^r\}$

Note that Condition 3 is equivalent to $(d'', \overline{d}) \in H_{t(c')}$. Moreover, we have $c \overset{d}{\hookrightarrow} c'$ iff $c' \overset{\overline{d}}{\hookrightarrow} c$ for all $c, c' \in C$ and $d \in D$. Continuing our example, we get $c_{2,1} \overset{w}{\hookrightarrow} c_{2,0}$ since $a(c_{2,1}, w) = c_{2,0}$, $(s, w) \in H(t(c_{2,1}))$, and $(w, n) \in H(t(c_{2,0}))$, and hence $c_{2,0} \overset{e}{\hookrightarrow} c_{2,1}$, too.

A *transit pair* (c, d) combines a non-trivial cell $c \in C$ and direction $d \in D$ such that $(d', d) \in H_{t(c)}$ for some $d' \in D$. The cell $c_{2,1}$ gives rise to three transit pairs: $(c_{2,1}, e)$, $(c_{2,1}, w)$, and $(c_{2,1}, n)$, corresponding to the three possible outgoing directions. We denote the set of all transit pairs for a grid of cells C by $\mathcal{T}(C)$. We require for each transit pair (c, d) that $c \overset{d}{\hookrightarrow} c'$ for some $c' \in C$. This condition ensures that each outgoing track leads to an incoming track on the neighboring cell.

We define a *transit path* as a sequence $((c_i, d_i))_{i=0}^n$ of distinct transit pairs with $n > 0$ such that for all $0 \leq i < n$:

1. $c_i \overset{d_i}{\hookrightarrow} c_{i+1}$
2. $(d_i, d_{i+1}) \in H_{t(c_{i+1})}$

Transit paths are non-trivial, as they comprise at least two transit pairs. We call $((c_i, d_i))_{i=0}^n$ a transit path between (c_0, d_0) and (c_n, d_n). Condition 2 can be seen as an instantiation of Condition 3 in the definition of connectedness with the outgoing direction of the subsequent cell. Any portion of a transit path is itself also a transit path. While a cell may be revisited, each visit must be associated with a different outgoing direction. Also, all but the last direction capture transitions between cells; the final one is an arbitrary choice among all outgoing directions of the last cell.

For illustration, consider the following transit path between $c_{0,1}$ and $c_{1,2}$ in Figure 1:

$$((c_{0,1}, s), (c_{1,1}, s), (c_{2,1}, w), (c_{2,0}, n), (c_{1,0}, e), (c_{1,1}, e), (c_{1,2}, e)) \tag{1}$$

Note that cell $c_{1,1}$ is traversed twice, once southwards and once eastwards.

In practice, Flatland environments are always strongly connected. To capture this formally, we require that for any two cells $c, c' \in C$, there is a transit path connecting some transit pairs (c, d) and (c', d') with $d, d' \in D$. Hence, transit paths between cells are not necessarily reversible.

Notation. *We associate a Flatland grid with its set C of cells and leave functions p, t, and a implicit.*

[2] The subscript in $c_{2,1}$ is just to ease readability.

Trains. Consider a set Z of *trains*, each characterized by a start and goal position, an initial orientation, an earliest departure time, and a latest arrival time, expressed by the functions:

1. $s : Z \to C$ and $g : Z \to C$ such that $s(z) \neq g(z)$ for all $z \in Z$
2. $o : Z \to D$ such that $(o(z), d) \in H_{s(z)}$ for some $d \in D$ and for all $z \in Z$
3. $\delta : Z \to \mathbb{N}$ and $\alpha : Z \to \mathbb{N}$ such that $\alpha(z) > \delta(z)$ for all $z \in Z$

The provisos stipulate that the starting and goal positions differ for each train, that the initial orientation of a train is always valid, corresponding to an incoming direction for its starting track, and that time is discrete, with the arrival time strictly exceeding the departure time.

Notation. *We associate functions s, g, o, δ, and α with the set Z of trains and leave them implicit.*

A *Flatland scenario* is a pair (C, Z), comprising a set C of cells and a set Z of trains. A train drive is a transit path that tolerates repeated transit pairs to mimic waiting, meets the train's initial orientation and its start and goal position, and respects the earliest departure time. Accordingly, we define a *train drive* for a train $z \in Z$ and some $m, n \in \mathbb{N}$ in a Flatland scenario (C, Z) as a sequence $((c_i, d_i))_{i=m}^{n}$ of transit pairs such that

1. for all $m \leq i < n$ either
   a) $c_i \overset{d_i}{\hookrightarrow} c_{i+1}$ and $(d_i, d_{i+1}) \in H_{t(c_{i+1})}$, or
   b) $(c_i, d_i) = (c_{i+1}, d_{i+1})$
2. $(c_m, d_m) = (s(z), o(z))$
3. $c_n = g(z)$ and $c_i \neq g(z)$ for all $m \leq i < n$
4. $m \geq \delta(z)$

Numbers m and n reflect the actual departure and arrival times of train z. The scheduled arrival time, $\alpha(z)$, is considered separately in delay calculations. Train drives are non-trivial since $s(z) \neq g(z)$.

Actions. Trains have four possible actions:[3] move forward (f), turn left (l), turn right (r), and wait (w). A transit pair (c, d) and one of its transitions $(d^i, d) \in H_{t(c)}$ induce the action a where

1. $a = f$ if
   a) $H_{t(c)} \cap (\{d^i\} \times D) = \{(d^i, d)\}$ or
   b) $H_{t(c)} \supset \{(d, d), (d, d')\}$ for $d^i = d$ and some $d' \in \{d^l, d^r\}$
2. $a = l$ if $H_{t(c)} \supset \{(d^r, d), (d^r, d')\}$ for $d^i = d^r$ and some $d' \in \{\overline{d}, d^r\}$
3. $a = r$ if $H_{t(c)} \supset \{(d^l, d), (d^l, d')\}$ for $d^i = d^l$ and some $d' \in \{\overline{d}, d^l\}$

No navigational choice is required in (1a) since there is only one alternative when entering from direction d^i. A second analogous entry point may exist (e.g., track types 6 and 8), but the choice of action f remains unambiguous. The three other cases, (1b), (2) and (3), are analogous in that they all require choosing d as the outgoing direction from two possible alternatives, using action f, l or r, respectively. Transit pairs induce unique actions, either f, l, or r. Two identical transit pairs form a w[ait] action.

Given a train drive $((c_i, d_i))_{i=m}^{n}$ for train $z \in Z$ in a Flatland scenario (C, Z), we define a *plan* for train z in (C, Z) as a sequence $(a_i)_{i=m}^{n-1}$ of actions such that for $m \leq i < n$

1. $a_i = w$ if $(c_i, d_i) = (c_{i+1}, d_{i+1})$ and
2. a_i is induced by (c_i, d_i) otherwise.

i	a_i	(c_i, d_i)
0	f	$(c_{0,2}, w)$
1	f	$(c_{0,1}, s)$
2	f	$(c_{1,1}, s)$
3	f	$(c_{2,1}, w)$
4	f	$(c_{2,0}, n)$
5	f	$(c_{1,0}, e)$
6	$\boldsymbol{f}$	$(c_{1,1}, e)$
7	w	$(c_{1,2}, e)$
8	f	$(c_{1,2}, e)$
9	$\boldsymbol{r}$	$(c_{1,3}, s)$
10	f	$(c_{2,3}, w)$
11		$(c_{2,2}, w)$

Table 1: A plan from $c_{0,2}$ to the train station in $c_{2,2}$ in Figure 1

[3] Flatland provides a fifth action, *do nothing*, which we consider superfluous and therefore omit from this formalization.

A sample plan for a train z is given in Table 1, depicting its journey from the starting cell $s(z) = c_{0,2}$ to the destination station at $g(z) = c_{2,2}$ (as shown in Figure 1). This plan is based on the initial conditions $o(z) = n$ and $\delta(z) = 0$. The plan requires only two navigational decisions, at steps 6 and 9, with all other movements being deterministic f actions, except for the wait at step 7. The decision at step 6 is made at cell $c_{1,1}$, where the train has two possible transits: one continuing east, via (e, e), and another turning south, via (e, s), both of which are members of $H_{t(c_{1,1})}$. The choice of outbound direction e is reflected by the inclusion of $(c_{1,1}, e)$ in the train drive. Similarly, when transiting $c_{1,3}$ eastwards at step 9, the train may go north or south, since both (e, n) and (e, s) belong to $H_{t(c_{1,3})}$. The transit pair $(c_{1,3}, s)$ reflects the southbound trajectory of the train.

Collisions. In Flatland, trains appear upon departure and disappear once they reach their destination. Two distinct drives $\big((c_i, d_i)\big)_{i=m}^{n}$ and $\big((c'_i, d'_i)\big)_{i=m'}^{n'}$ have

1. a *vertex conflict* if $c_i = c'_i$ for some $i \in [\max(m, m'), \min(n, n')]$
2. an *edge conflict* if $c_{i+1} = c'_i$ and $c_i = c'_{i+1}$ for some $i, i+1 \in [\max(m, m'), \min(n, n')]$

Solution. A *solution* to a Flatland scenario (C, Z) is a family $(\pi_z)_{z \in Z}$ of plans π_z for each train $z \in Z$ such that all drives underlying $(\pi_z)_{z \in Z}$ are pairwisely vertex and edge conflict-free

The delay of a plan π_z for a train $z \in Z$ is defined as $delay(\pi_z) = \max(0, n - \alpha(z))$ where n is the actual arrival time of train z with plan π_z A *delay optimal* solution to a Flatland scenario (C, Z) is a solution $(\pi_z)_{z \in Z}$ to (C, Z) with a minimum value of $\sum_{z \in Z} delay(\pi_z)$

3. Graph representations

We now consider alternative graph representations for capturing railway networks in Flatland. We must concern ourselves with the behavior of movement around switches. The adjacency of two cells is a necessary condition for transition, but not a sufficient one. Furthermore, the direction of the train must be taken into consideration. For instance, cell $c_{2,1}$ has three neighboring cells in Figure 1. However, only a train facing east can go from $c_{2,1}$ to its adjacent cells $c_{1,1}$ or $c_{2,2}$, respectively, and its neighboring cell $c_{2,0}$ can only be reached by trains facing south or west. In what follows, we consider translations of grid environments to alternative graph structures which account for such additional qualities.

3.1. Directed graph with edge transition functions

Given a grid C, we define the graph $\mathcal{A}(C) = (V, E)$ as

1. $E = \bigcup_{c \in C, d \in D} \{(c, c') \mid c \overset{d}{\hookrightarrow} c'\}$
2. $V = \{c, c' \mid (c, c') \in E\}$

The directed graph $\mathcal{A}(C)$ contains no trivial cells from C. To accommodate the additional structure needed to capture transitions, we associate with each graph $\mathcal{A}(C) = (V, E)$ two partial functions on its set of edges:

1. $d : E \to D$ where $d((c, c')) = d$ if $c \overset{d}{\hookrightarrow} c'$ for $(c, c') \in E$
2. $p : E \to 2^D$ where $p((c, c')) = \{d'' \mid (d'', d) \in H_{t(c)}\}$ if $c \overset{d}{\hookrightarrow} c'$ for $(c, c') \in E$.

Function d reflects the direction of a directed edge, while p gives the necessary arriving directions when leaving a node.

Figure 3(a) shows the result of translating our example grid from Figure 2. First of all, note that the trivial cell $c_{0,0}$ was dropped from the graph. To illustrate the purpose of both functions, consider node $c_{2,1}$ in Figure 3(a) along with function d and p applied to its incoming and outgoing edges, respectively.

$$d((c_{2,0}, c_{2,1})) = e \qquad\qquad p((c_{2,1}, c_{2,0})) = \{s, w\} \tag{2}$$

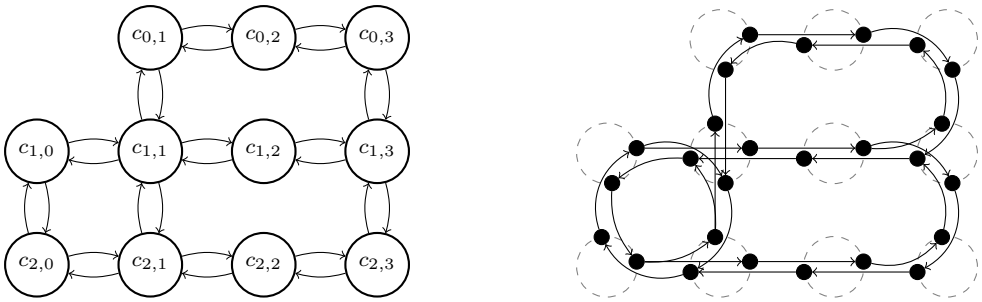


Figure 3: Representation of our example as a directed graph with (a) edge transition functions and (b) sub-nodes.

$$d((c_{1,1}, c_{2,1})) = s \qquad p((c_{2,1}, c_{2,2})) = \{e\} \tag{3}$$
$$d((c_{2,2}, c_{2,1})) = w \qquad p((c_{2,1}, c_{1,1})) = \{e\} \tag{4}$$

The interplay of both functions determines whether an incoming and outgoing edge allow for a transition over a node at hand. More precisely, this is the case for a node c, whenever $d((c', c)) \in p((c, c''))$ for some connected nodes c', c''. Four combinations of the mappings above satisfy this condition for $c_{2,1}$, which reflect the four possible transitions over $c_{2,1}$.

Given a grid C, we define a path in $\mathcal{A}(C)$ as a sequence $(c_i)_{i=0}^n$ of vertices with $n > 0$ such that

1. $(c_i, c_{i+1}) \in E$ for all $0 \leq i < n$
2. $d((c_{i-1}, c_i)) \in p((c_i, c_{i+1}))$ for all $0 < i < n$
3. if $c_i = c_j$ then $c_{i+1} \neq c_{j+1}$ for all $0 \leq i < j < n$

We call $(c_i)_{i=0}^n$ a path between c_0 and c_n in $\mathcal{A}(C)$. For example, the transit path $((c_{1,1}, s), (c_{2,1}, w), (c_{2,0}, n))$ between $c_{1,1}$ and $c_{2,0}$ in our grid C in Figure 1 corresponds to the path $(c_{1,1}, c_{2,1}, c_{2,0})$ in $\mathcal{A}(C)$ and satisfies $d((c_{1,1}, c_{2,1})) \in p((c_{2,1}, c_{2,0}))$.

The following proposition shows that our grid representation preserves all paths in the original grid.

Proposition 1. *Let C be a grid.*

1. *If $((c_i, d_i))_{i=0}^n$ is a transit path in grid C, then $(c_i)_{i=0}^n$ is a path in the graph $\mathcal{A}(C)$.*
2. *If $(c_i)_{i=0}^n$ is a path in the graph $\mathcal{A}(C)$, then $((c_i, d_i))_{i=0}^n$ is a transit path in grid C, where $d_i = d((c_i, c_{i+1}))$ for $0 \leq i < n$.*

3.2. Directed graph with sub-nodes

Given a grid C, we define the graph $\mathcal{B}(C) = (V, E)$ as

1. $V = \mathcal{T}(C)$
2. $E = \{((c, d), (c', d')) \in V \times V \mid c \overset{d}{\hookrightarrow} c', (d, d') \in H_{t(c')}\}$

The directed graph $\mathcal{B}(C)$ connects all transit pairs of a grid. Fragmenting cells into transit pairs and relating them in terms of their connectivity allows us to account for the transitions over the various track types.

The result of translating our Flatland grid from Figure 2 is given in Figure 3(b). The cell $c_{2,1}$ is now decomposed into three nodes, $(c_{2,1}, e)$, $(c_{2,1}, w)$, and $(c_{2,1}, n)$, corresponding to its three transit pairs. The dashed circles in Figure 3(b) are only meant to indicate nodes stemming from the same original cell and have no structural meaning. Intuitively, the above edge transition functions are compiled into the refined graph structure. Each node (or transit pair) represents one way of exiting a node. The four possible passages over $c_{2,1}$ are reflected by the four edges entering the three nodes derived from $c_{2,1}$.

Given a grid C, we define a path in $\mathcal{B}(C)$ as a sequence $((c_i, d_i))_{i=0}^n$ of transit pairs with $n > 0$ such that

1. $((c_i, d_i), (c_{i+1}, d_{i+1})) \in E$ for all $0 \leq i < n$

2. if $(c_i, d_i) = (c_j, d_j)$, then $i = j$ for all $0 \leq i < j < n$

We call $((c_i, d_i))_{i=0}^n$ a path between (c_0, d_0) and (c_n, d_n) in $\mathcal{B}(C)$. In fact, the transit path $((c_{1,1}, s), (c_{2,1}, w), (c_{2,0}, n))$ between $c_{1,1}$ and $c_{2,0}$ in our grid C in Figure 1 is also a path in $\mathcal{B}(C)$.

Proposition 2. *Let C be a grid.*

1. *If $((c_i, d_i))_{i=0}^n$ is a transit path in grid C, then $((c_i, d_i))_{i=0}^n$ is a path in the graph $\mathcal{B}(C)$.*
2. *If $((c_i, d_i))_{i=0}^n$ is a path in the graph $\mathcal{B}(C)$, then $((c_i, d_i))_{i=0}^n$ is a transit path in grid C.*

3.3. Directed graph with hyper-edges

Given a grid C, we define the graph $\mathcal{H}(C) = (V, E)$ as

1. $E = \bigcup_{c \in C, d \in D} \{(c, c', c'') \mid c \overset{d}{\hookrightarrow} c', c' \overset{d'}{\hookrightarrow} c'', (d, d') \in H_{t(c')}\}$
2. $V = \{c, c', c'' \mid (c, c', c'') \in E\}$

$\mathcal{H}(C)$ is a graph using directed hyper-edges among the non-trivial cells of the original grid C. Each hyper-edge captures a valid passage over a node by connecting it to its predecessor and successor in this transition.

Consider the graph $\mathcal{H}(C)$ obtained from our Flatland grid in Figure 2. We get four hyper-edges centered upon node $c_{2,1}$, namely,

$$(c_{2,0}, c_{2,1}, c_{2,2}) \qquad (c_{2,0}, c_{2,1}, c_{1,1}) \tag{5}$$

$$(c_{1,1}, c_{2,1}, c_{2,0}) \qquad (c_{2,2}, c_{2,1}, c_{2,0}) \tag{6}$$

These hyper-edges correspond to the four passages over $c_{2,1}$ discussed above.

Given a grid C, we define a path in $\mathcal{H}(C)$ as a sequence $(c_i)_{i=0}^n$ of vertices with $n > 0$ such that

1. $(c_{i-1}, c_i, c_{i+1}) \in E$ for all $0 < i < n$
2. $(c, c_0, c_1) \in E$ and $(c_{n-1}, c_n, c') \in E$ for some $c, c' \in V$
3. if $c_i = c_j$ then $c_{i+1} \neq c_{j+1}$ for all $0 \leq i < j < n$

We call $(c_i)_{i=0}^n$ a path between c_0 and c_n in $\mathcal{H}(C)$. Condition 2 ensures that the constraints imposed by hyper-edges are satisfied at the endpoints of each path, and for paths of length two.

For example, the transit path $((c_{1,1}, s), (c_{2,1}, w), (c_{2,0}, n))$ between $c_{1,1}$ and $c_{2,0}$ in our grid C in Figure 1 corresponds to the path $(c_{1,1}, c_{2,1}, c_{2,0})$ in $\mathcal{H}(C)$. For $\mathcal{H}(C) = (V, E)$, it satisfies the following conditions:

$$(c_{0,1}, c_{1,1}, c_{2,1}) \in E \tag{7}$$

$$(c_{1,1}, c_{2,1}, c_{2,0}) \in E \tag{8}$$

$$(c_{2,1}, c_{2,0}, c_{1,0}) \in E \tag{9}$$

Proposition 3. *Let C be a grid.*

1. *If $((c_i, d_i))_{i=0}^n$ is a transit path in grid C, then $(c_i)_{i=0}^n$ is a path in the graph $\mathcal{H}(C)$.*
2. *If $(c_i)_{i=0}^n$ is a path in the graph $\mathcal{H}(C)$, then $((c_i, d_i))_{i=0}^n$ is a transit path in grid C, where $c_i \overset{d_i}{\hookrightarrow} c_{i+1}$ for $0 \leq i < n$.*

4. *Flaspland*: Flatland to ASP and back

Flaspland is an open-source toolkit that bridges the gap between ASP and the Flatland ecosystem . The toolkit provides a bidirectional translation layer between instance generation and solving. Flatland environments are converted into instance files that adhere to a consistent ASP fact format, while answer sets produced by custom encodings are mapped back to Flatland actions for execution and visualization. Rather than requiring users to construct instances manually, *flaspland* integrates directly with the existing environment generator in Flatland and supports batch generation of environments to streamline experimental workflows at scale. Additionally, the toolkit serves as a repository housing Flatland benchmark instances, providing a unified point of access for reproducible evaluation. These comprise the 150 environments used in the final round of the Flatland 3 competition. The environments range in size, number of agents and cities, and include trains that malfunction and move at various speeds. The *flaspland* toolkit is freely and publicly available at [7].

5. Encodings

We now present the ASP encodings for addressing routing and scheduling in Flatland, starting with the representation of the environment's basic topological and navigational concepts. These encodings are available at [8]. The fact format of the basic concepts regarding directions is given in Listing 1.

```
1      dir(n).          dir(s).          dir(w).          dir(e).
2  inverse(n,s).    inverse(s,n).    inverse(w,e).    inverse(e,w).
3     left(n,w).       left(s,e).       left(w,s).       left(e,n).
4    right(n,e).      right(s,w).      right(w,n).      right(e,s).
5   offset(n,-1,0).  offset(s,1,0).   offset(w,0,-1).  offset(e,0,1).
```

Listing 1: Representation of directions (n,s,w,o)

Predicate `dir/1` gives the cardinal directions, `inverse/2` provides complementary directions, `left/2` and `right/2` capture 90-degree rotations, and `offset/3` gives a coordinate offset when moving in a certain direction.

In Flatland, each track type has a numeric value that is based on its possible transitions;[4] it is represented via predicate `transition/2`, one instance per transition. Listing 2 contains all facts reflecting the four transitions of the simple switch at position $(2, 1)$ in Figure 1.

```
42  transition(3089,(s,w))
43  transition(3089,(w,w))
44  transition(3089,(e,e))
45  transition(3089,(e,n))
```

Listing 2: Selected lines of track type representations

The actual network is represented by atoms of predicate `cell/2`, where the first argument reflects the coordinates of each cell and the second gives its type (cf. Listing 4). The first argument serves as both the identifier and the value of function p for the cell at hand. The second argument gives the cell's track type; its transitions are obtained by combining predicates `cell/2` and `transition/2`. The adjacency function a and the connectivity relation $\hookrightarrow$ are defined in Listing 3, using predicates `adjacent/3` and `connected/3`, respectively.

```
1  adjacent((X,Y),D,(X',Y')) :-  cell((X,Y),_), cell((X',Y'),_),
2      offset(D,DX,DY), X'=X+DX, Y'=Y+DY.

4  connected((X,Y),D,(X',Y')) :- cell((X,Y),T), cell((X',Y'),T'),
```

[4] For example, the switch track at coordinate $(2, 1)$ in Figure 1 has the value 3089, based on its 16-digit binary representation `0000 1100 0001 0001` of the transitions in $H_{(2,1)}$. Each group of four digits represents transitions when an agent is facing north, east, south, or west, respectively. Within each group, the four digits represent transitions out of a cell in the directions north, east, south, or west, respectively. The digit 1 indicates the transition is permissible.

```
5       adjacent((X,Y),D,(X',Y')),
6       transition(T,(D',D)), transition(T',(D,D'')).
```

Listing 3: Representation of grid functions

In addition to representing the grid with the `cell/2` predicate, our instance format uses `start/4` and `end/3` to fix the start and end positions of trains. The first two arguments of these predicates specify the train identifier z and the starting $s(z)$ or ending $g(z)$ position, respectively. The third and fourth arguments of `start/4` indicate the earliest departure time $\delta(z)$ and initial orientation $o(z)$, while the third argument of `end/3` specifies the latest arrival time $\alpha(z)$. An example is given in Line 1 and 2 in Listing 4.

```
1  start(0,(0,2),0,w).
2  end(0,(2,2),12).

4  cell((0,0),    0).  cell((0,1),16386).  cell((0,2),1025).  cell((0,3),4608).
5  cell((1,0),16386).  cell((1,1),38433).  cell((1,2),1025).  cell((1,3),6672).
6  cell((2,0),   72).  cell((2,1), 3089).  cell((2,2),1025).  cell((2,3),2064).
```

Listing 4: The fact representation of our running example (Figure 1)

The fact representation introduced above, generated directly from Flatland scenarios by our *flaspland* framework, constitutes the basic input for our ASP encodings. Building upon this basis, we investigate how the three graph-based representations defined in Section 3 can be utilized to effectively model train dynamics and interactions.

Our encoding design is driven by two main objectives. First, we aim to evaluate the computational trade-offs of the proposed topological representations, specifically regarding how the choice of structural abstraction influences ASP solving performance. Second, our encodings must robustly address the spatio-temporal complexities inherent to train scheduling. Foremost is the need to model waiting actions for collision avoidance, which inherently decouples spatial transitions from strict temporal succession. Further complexities stem from the asynchronous life-cycle of trains, such as their variable appearance, delayed departures, and disappearance upon arrival. Moreover, a train's initial orientation can typically only be verified upon its actual departure.

We address these spatio-temporal complexities by decomposing the synthesis of train drives into two distinct aspects: spatial routing and temporal scheduling. The routing component establishes the backbone of a train's journey by identifying a valid path between its start and destination. Crucially, while these paths may revisit nodes, they prohibit consecutive duplicate nodes. This abstraction of temporal waiting ensures that spatial transitions can be directly verified against the underlying track topology. The scheduling component then refines these spatial paths into fully realized train drives by inserting appropriate dwelling times to enforce collision avoidance (Section 5.5).

In the following, we provide a specialized path encoding for each graph representation and subsequently integrate them into a unified drive encoding in Section 5.4.

5.1. Encodings based on directed graphs with edge transition functions

This section introduces encodings based on a basic graph representation augmented with edge functions, as formalized in Section 3.1. The encoding in Listing 5 translates a Flatland grid into this structure.

```
1  edge(C,C') :- connected(C,_,C').
2  node(C;C') :- edge(C,C').

4  d(C,C',D) :- connected(C,D,C').
5  p(C,C',D) :- connected(C,D',C'), cell(C,T), transition(T,(D,D')).
```

Listing 5: Encoding for directed graphs with edge functions (cf. Section 3.1).

The graph topology is captured by the predicates `node/1` and `edge/2`, while `d/3` and `p/3` encode the functions d and p, respectively.

The encoding in Listing 6 identifies spatial paths of length up to h between each train's starting and goal positions.

```
1  time(0..h).

3  path(Z,C,0; Z,C',1) :- start(Z,C,_,D), adjacent(C,D,C').
4  { path(Z,C',T+1): edge(C,C') } = 1 :- path(Z,C,T), time(T+1), not eop(Z,T).

6  eop(Z,T) :- end(Z,C,_), path(Z,C,T).
7  :- start(Z,_,_,_), not eop(Z,_).

9  :- path(Z,C,T), path(Z,C',T+1), path(Z,C'',T+2), d(C,C',D), not p(C',C'',D).
```

Listing 6: Pathfinding with edge-functions

Line 1 defines the available time steps, while Line 3 initializes each train at its designated starting position and orientation. The path is iteratively extended (Line 4) until the destination is reached. Termination is signaled by the `eop(Z,T)` atom (Line 6),[5] indicating that train `Z` has arrived at its destination at time step `T`. The integrity constraint in Line 7 ensures that every train eventually reaches its goal. Crucially, Line 9 enforces transition validity by applying the edge functions d and p to consecutive nodes in the path, verifying that the incoming and outcoming edge form a valid transition.

5.2. Encodings based on directed graphs with sub-nodes

This section presents encodings leveraging the augmented node structure defined in Section 3.2. The translation from the standard Flatland format into this representation is handled by the encoding in Listing 7.

```
1  node((X,Y,D)) :- cell((X,Y),T), transition(T, (_,D)).

3  edge((X,Y,D), (X',Y',D')) :- node((X,Y,D)), node((X',Y',D')),
4      connected((X,Y),D,(X',Y')), cell((X',Y'), T'), transition(T', (D,D')).
```

Listing 7: Encoding for directed graphs with sub-nodes (cf. Section 3.2).

In this model, each node corresponds to a transit pair (a cell and an outgoing direction), thereby encoding valid transitions directly into the graph's edge construction.

Listing 8 adapts the pathfinding logic to this refined structure.

```
1  time(0..h).

3  path'(Z,(X,Y,D),0) :- start(Z,(X,Y),_,D).
4  { path'(Z,C',T+1) : edge(C,C') } = 1 :- path'(Z,C,T), time(T+1), not eop(Z,T).

6  eop(Z,T) :- end(Z,(X,Y),_), path(Z,(X,Y),T).
7  :- start(Z,_,_,_), not eop(Z,_).

9  path(Z,(X,Y),T) :- path'(Z,(X,Y,D),T).
```

Listing 8: Pathfinding with sub-nodes

While the rules in Lines 1–7 mirror the logic of the edge-function-based encoding, they operate on transit pairs rather than simple cells. To provide a uniform interface for subsequent scheduling, the internal predicate `path'/3` is projected back onto the cell-based `path/3` in Line 9. In this representation, transition validity is guaranteed by construction and requires no explicit verification during path generation.

5.3. Encodings based on directed graphs with hyper-edges

This section introduces encodings utilizing the hypergraph structure defined in Section 3.3. Listing 9 provides the translation into this representation.

[5] eop stands for "end of path".

```
1  hedge(C,C',C'') :- connected(C,D,C'), connected(C',D',C''),
2           cell(C',T), transition(T,(D,D')).

4  node(C;C';C'') :- hedge(C,C',C'').
```

Listing 9: Encoding for directed graphs with hyper-edges (cf. Section 3.3).

While `node/1` continues to designate cell-based nodes, the valid transitions are modeled via the ternary predicate `hedge/3`. This approach naturally captures the dependencies between a node's entry and exit points.

Listing 10 details the corresponding pathfinding encoding.

```
1  time(0..h).

3  path(Z,C,0; Z,C',1) :- start(Z,C,_,D), adjacent(C,D,C').
4  { path(Z,C'',T+1) : hedge(C,C',C'') } = 1 :- path(Z,C,T-1), path(Z,C',T), time(T+1), not eop(Z,T).

6  eop(Z,T) :- end(Z,C,_), path(Z,C,T).
7  :- start(Z,_,_,_), not eop(Z,_).
```

Listing 10: Pathfinding with hyper-edges

The path generation rules (Lines 1–7) leverage the hyper-edges to ensure that every step in the path is valid according to the underlying track topology. This representation is arguably the most direct, as the hyper-edges explicitly encapsulate the ternary nature of cell transitions without requiring auxiliary functions or node fragmentation.

5.4. Encodings of train drives

To transform spatial paths into fully realized train drives, we must account for potential dwelling times at each node, which are essential for collision avoidance. The total dwelling time for a train is constrained by the interval between its earliest departure and its actual arrival at the destination. This constraint is captured by the predicate `delay_budget/2` (Line 1). The cumulative delay at each node along the path is then defined in Lines 3 to 5. Once a specific delay is attributed to a node in Line 3, Line 4 ensures its monotonic accumulation, while Line 5 terminates this upon reaching the goal. Note that we use the index -1 as a convention to represent the initial delay in departing compared to the earliest possible start time. Finally, the resulting `drive` is assembled in Line 7 by mapping the spatial path coordinates to their respective temporal steps based on these accumulated delays.

```
1  delay_budget(Z,h-L-S) :- start(Z,_,S,_), eop(Z,L).

3  { delay(Z,T-1,0..B) } = 1 :- path(Z,_,T), delay_budget(Z,B).
4  :- delay(Z,T,W), delay(Z,T+1,W'), W' < W.
5  delay(Z,T,D) :- eop(Z,T), delay(Z,T-1,D).

7  drive(Z,C,T+S+W..T+S+W') :- path(Z,C,T), start(Z,_,S,_), delay(Z,T-1,W), delay(Z,T,W').

9  #show drive/3.
```

Listing 11: Assembling train drives from paths

5.5. Encodings of collisions, plans, and delay optimality

Beyond individual train dynamics, the feasibility of a solution hinges on the absence of inter-train interference. To this end, the vertex and edge conflict conditions established in Section 2 are implemented in Listing 12. These constraints ensure that no two trains occupy the same cell simultaneously and prevent head-on collisions by forbidding trains from traversing the same edge in opposite directions at the same time step.

```
1  :- drive(Z,C,T), drive(Z',C,T), Z < Z'.
2  :- drive(Z,C,T), drive(Z,C'T+1), drive(Z',C',T), drive(Z',C,T+1), Z < Z'.
```

Listing 12: Vertex and edge conflicts

		MAPF						Edge-functions						Sub-nodes						Hyper-edges					
$\|T\|$	$\|I\|$	Grnd	Global	TO	Rules	Vars	Cons	Grnd	Global	TO	Rules	Vars	Cons	Grnd	Global	TO	Rules	Vars	Cons	Grnd	Global	TO	Rules	Vars	Cons
2	70	3.1	8.9	0	801516	298688	767748	38.9	86.4	0	10424661	4250093	21649502	19.7	20.2	0	4545471	2004087	10059450	36.4	50.1	0	9371445	3932519	19886951
3	70	4.5	81.1	0	1189088	427171	1130507	81.8	201.3	0	21709988	7309423	41164599	45.2	47.4	0	10757686	3413388	20093834	79.6	119.2	0	19907260	6851270	38352250
4	70	6.0	483.9	8	1600794	547378	1484530	139.2	585.6	8	35679753	10507063	62477860	82.5	133.5	1	19682594	4964144	33211557	140.4	333.8	5	32704159	9805305	58390218
5	70	7.5	1069.2	31	2011070	644632	1774885	190.6	468.8	4	25063841	12968607	40274980	129.5	196.2	2	29441568	6306472	45867073	190.3	273.1	3	26650172	12167195	43820106
6	70	9.3	1378.1	56	2462364	755469	2118716	246.4	580.7	10	28215589	15212880	43269829	183.2	306.3	6	33287181	7631051	48155418	243.0	296.2	1	28711553	14083026	44824302
7	10	10.4	1500.0	10	2486713	845793	2223014	260.4	286.1	0	8685563	7852626	11475302	235.0	462.4	1	38357113	8967436	52883832	236.0	252.4	0	7956530	7324500	10824910

Table 2
Average experimental results comparing various drive encoding strategies.

Finally, we address the optimization of the overall schedule. While the pathfinding component identifies valid spatial routes, the scheduling component must minimize the total deviation from the target arrival times. This is achieved by minimizing the sum of delays across all trains, effectively penalizing any dwelling time that extends a train's journey beyond its scheduled window. By combining these constraints, the resulting ASP encoding provides a complete framework for synthesizing delay-optimal solutions within the Flatland environment.

```
1 #minimize{ D : delay(Z,T,D), eop(Z,T) }.
```

Listing 13: Delay minimization

Finally, we note that the concrete sequences of actions, or plans, are implicitly defined by the underlying train drives. Within the *flaspland* framework, these plans are primarily utilized for visualization purposes and are therefore omitted here for conciseness.

6. Experiments

We evaluate the three proposed topological representations against a baseline encoding on 360 benchmark instances to assess their computational performance and scalability. The baseline is an adaptation of a mature, off-the-shelf MAPF encoding [9], augmented with the necessary constraints to handle Flatland's directional dynamics. Our benchmark set is derived from the first ten base environments described in Section 4. Each environment consists of a 30×30 grid populated with seven trains. To investigate how our encodings scale with agent density, we systematically varied the number of active trains while maintaining a fixed track topology. Specifically, for each of the ten base environments and for each subset size $k \in \{2, \ldots, 6\}$, we randomly sampled seven train combinations. For $k = 7$, the unique full instance was included. This sampling strategy yields $10 \times (5 \times 7 + 1) = 360$ benchmark instances in total. For each instance, we set the maximum time horizon to the latest arrival time among the participating trains, ensuring a feasible temporal window for all agents. We ran all instances on a cluster with Intel Xeon E5-2650v4@2.9GHz CPUs with 64GB of memory running Debian Linux 10. We used *clingo* 5.8.0 and imposed a timeout of 1500 seconds and a memory limit of 15GB per run.

Table 2 summarizes our results, comparing the four configurations MAPF, Edge-functions, Sub-nodes, and Hyper-edges, across scenarios with an increasing number of trains ($|\mathsf{T}|$, from two to seven). The off-the-shelf MAPF configuration scales poorly as the number of trains increases. While it produces the smallest problem sizes (featuring fewer rules (Rules), variables (Vars), and constraints (Cons))[6] and boasts the fastest grounding times (Grnd, averaging 6.8s), it struggles to solve complex instances. It reaches a peak of 56 timeouts (TO) for six trains and hits the 1500-second time limit for all ten instances ($|\mathsf{I}|$) with seven trains. In contrast, the three path-driven variants generate an order of magnitude more rules and variables but achieve significantly better overall success rates and lower total solving times (Global) for larger train counts.

Among the proposed graph representations, Sub-nodes and Hyper-edges significantly outperform the Edge-functions variant in both global time and reliability. Edge-functions consistently records the highest global times (averaging 368.16s) and the most timeouts (averaging 3.67), making it the least robust of the three. Interestingly, Sub-nodes generates the largest instantiations (averaging 22.68M rules) compared to Edge-functions (21.63M) and Hyper-edges (20.88M), yet it achieves the fastest grounding times, averaging 115.85s, roughly 25% faster than the other two variants. Regarding overall scalability, Sub-nodes provides the most consistent performance for smaller instances, starting with a minimal global time of 20.17s for two trains. However, it exhibits a steeper increase in global time as the system scales, reaching 462.40s for seven trains. Conversely, Hyper-edges demonstrates better scalability under heavy loads. While starting slightly slower (50.06s for two trains), it maintains a much more stable trajectory, ending at 252.36s for seven trains and yielding the lowest average number of timeouts (1.5).

[6]Specifically, Rules denotes the number of ground rules generated by the grounder, while Vars and Cons represent the number of Boolean variables and constraints, respectively, internally translated and processed by the solver.

7. Conclusion

While Flatland was originally conceived as a benchmark for reinforcement learning [10], the success of symbolic MAPF techniques in the 2020 Flatland competition highlighted the strengths of combinatorial approaches in this domain [11]. Building upon the successful application of ASP to real-world railway scheduling [12], we introduced *flaspland* as a dedicated testbed for tackling these challenges within a declarative framework.

This preliminary report provides a rigorous formalization of the Flatland environment, establishing a solid foundation for verifying the correctness of subsequent ASP encodings. We propose three distinct graph-based representations to capture the environment's topological and navigational constraints, and develop a two-phase ASP scheme that effectively decouples spatial routing from temporal scheduling. Our empirical evaluation demonstrates that this path-driven representation, particularly those employing sub-nodes and hyper-edges, offer significant scalability advantages over off-the-shelf MAPF encodings. By bridging the gap between grid-based simulation and graph-based encodings, *flaspland* provides a scalable benchmark for evaluating the performance of ASP in complex, multi-agent railway scenarios.

8. Declaration on Generative AI

The authors have not employed any generative AI tool in the preparation of this work.

References

[1] V. Lifschitz, Answer Set Programming, Springer-Verlag, 2019. doi:10.1007/978-3-030-24658-7.

[2] E. Erdem, D. Kisa, U. Öztok, P. Schüller, A general formal framework for pathfinding problems with multiple agents, in: M. desJardins, M. Littman (Eds.), Proceedings of the Twenty-seventh National Conference on Artificial Intelligence (AAAI'13), AAAI Press, 2013, pp. 290–296.

[3] D. Brooks, E. Erdem, S. Erdogan, J. Minett, D. Ringe, Inferring phylogenetic trees using answer set programming, Journal of Automated Reasoning 39 (2007) 471–511.

[4] E. Erdem, V. Lifschitz, M. Wong, Wire routing and satisfiability planning, in: J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K. Lau, C. Palamidessi, L. Pereira, Y. Sagiv, P. Stuckey (Eds.), Proceedings of the First International Conference on Computational Logic (CL'00), volume 1861 of *Lecture Notes in Computer Science*, Springer-Verlag, 2000, pp. 822–836.

[5] F. Laurent, M. Schneider, C. Scheller, J. Watson, J. Li, Z. Chen, Y. Zheng, S. Chan, K. Makhnev, O. Svidchenko, V. Egorov, D. Ivanov, A. Shpilman, E. Spirovska, O. Tanevski, A. Nikov, R. Grunder, D. Galevski, J. Mitrovski, G. Sartoretti, Z. Luo, M. Damani, N. Bhattacharya, S. Agarwal, A. Egli, E. Nygren, S. Mohanty, Flatland competition 2020: MAPF and MARL for efficient train coordination on a grid world, in: H. Escalante, K. Hofmann (Eds.), Proceedings of the NeurIPS'20 Competition and Demonstration Track, volume 133 of *Proceedings of Machine Learning Research*, PMLR, 2021, pp. 275–301. URL: http://proceedings.mlr.press/v133/laurent21a.html.

[6] R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. Kumar, R. Barták, E. Boyarski, Multi-agent pathfinding: Definitions, variants, and benchmarks, in: P. Surynek, W. Yeoh (Eds.), Proceedings of the Twelfth International Symposium on Combinatorial Search (SOCS'19), AAAI Press, 2019, pp. 151–159.

[7] R. Murphy, A. Nemes, J. Romero, T. Schaub, *flaspland*, https://github.com/potassco/flaspland-toolkit, 2026.

[8] R. Murphy, A. Nemes, J. Romero, T. Schaub, *flaspland* encodings, https://github.com/potassco/flaspland-encodings, 2026.

[9] J. Behrens, R. Kaminski, T. Schaub, T. Son, J. Švancara, P. Wanko, Routing and scheduling in answer set programming applied to multi-agent path finding: Preliminary report, CoRR abs/2403.12153 (2024). URL: https://arxiv.org/abs/2403.12153.

[10] S. Mohanty, E. Nygren, F. Laurent, M. Schneider, C. Scheller, N. Bhattacharya, J. Watson, A. Egli, C. Eichenberger, C. Baumberger, G. Vienken, I. Sturm, G. Sartoretti, G. Spigler, Flatland-rl : Multi-agent reinforcement learning on trains, CoRR (2020). URL: https://arxiv.org/abs/2012.05893.

[11] J. Li, Z. Chen, Y. Zheng, S. Chan, D. Harabor, P. Stuckey, H. Ma, S. Koenig, Scalable rail planning and replanning: Winning the 2020 flatland challenge, in: S. Biundo, M. Do, R. Goldman, M. Katz, Q. Yang, H. Hankui Zhuo (Eds.), Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS'21), AAAI Press, 2021, pp. 477–485. doi:10.1609/icaps.v31i1.15994.

[12] D. Abels, J. Jordi, M. Ostrowski, T. Schaub, A. Toletti, P. Wanko, Train scheduling with hybrid ASP, Theory and Practice of Logic Programming 21 (2021) 317–347. doi:10.1017/S1471068420000046.