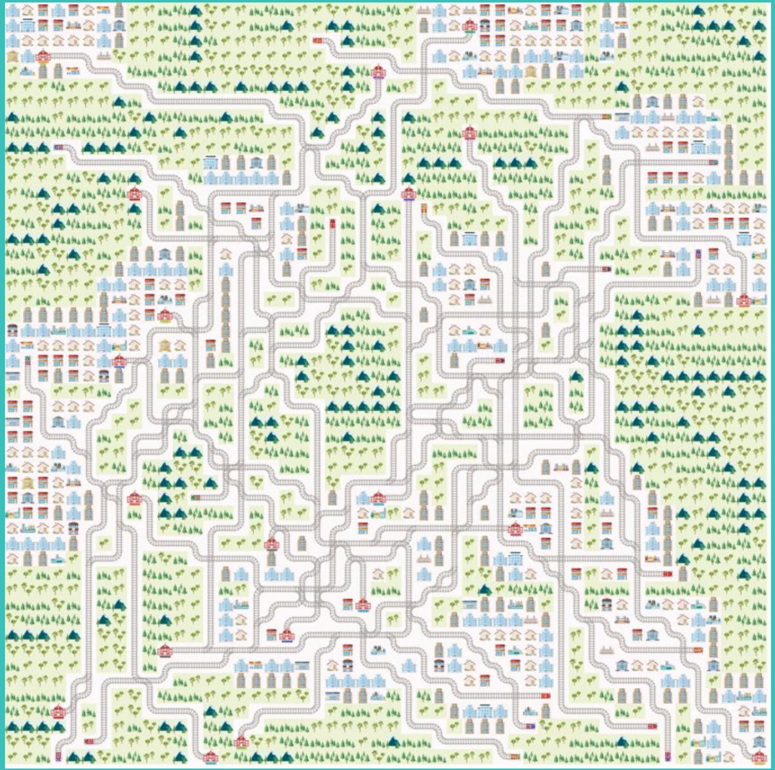
Railway routing

One challenge, two different ways

Overview of the presentation

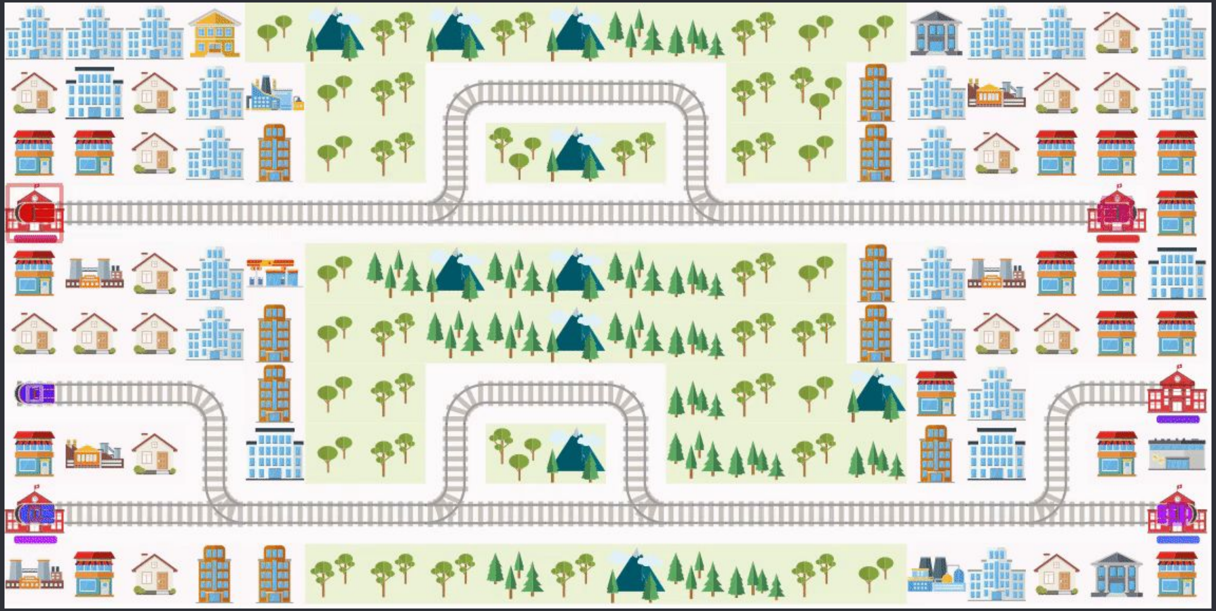
- Presentation of the Flatland challenge
- Reinforcement learning approach
- Combinatorial optimization approach

Flatland

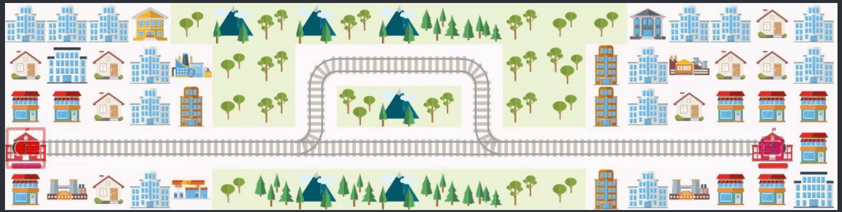


Online competition by SBB

“How can trains learn to automatically coordinate among themselves, so that there are minimal delays in large train networks ?”



Details of the challenge



Grid world constraints

No collisions (two trains in the same cell)
No swapping (trains cannot switch positions)

Cost function

Sum of travel time over all trains

Increasing difficulty

Train-specific speeds and train sometimes breaks

Reinforcement learning approach

What is Reinforcement Learning ?

Active Learning

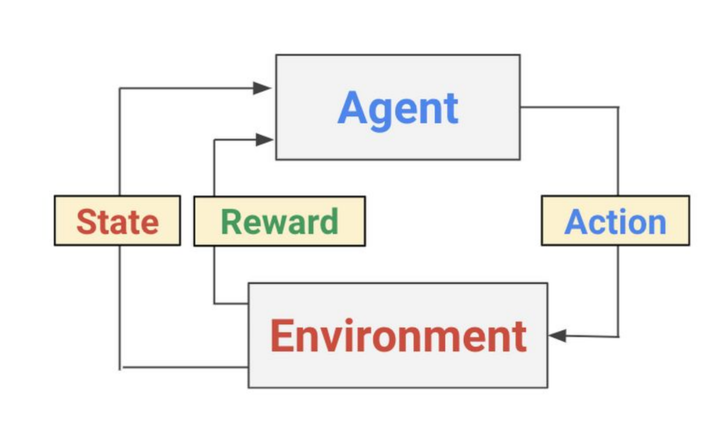
We learn by interacting with our environment.

Sequential Interactions

Future interactions can depend on earlier ones.

Independent Learning

We can learn without examples of optimal behaviour.



What is Reinforcement Learning ?

Active Learning

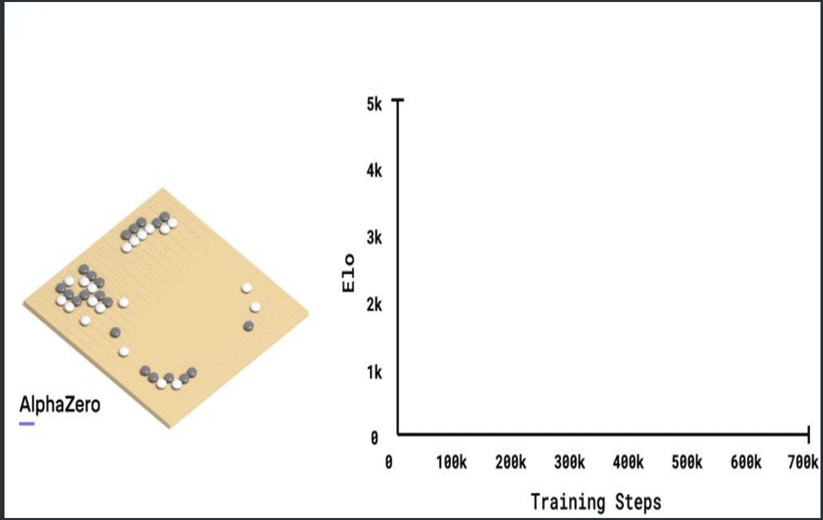
We learn by interacting with our environment.

Sequential Interactions

Future interactions can depend on earlier ones.

Independent Learning

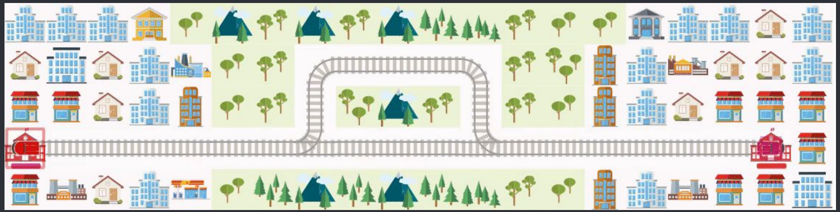
We can learn without examples of optimal behaviour.



Learning to make decisions from interactions

We have to think about:

- Time
- Long-term consequences of actions
- Actively gathering experience
- Dealing with uncertainty



The SBB problem captures many of these ideas !

Markov Decision processes (MDP)

$(x, y,\ direction)$

$\{stop,\ go\ forward,\ turn\}$

- A set of agent states : $\mathcal{S}$.
- A set of actions $\mathcal{A}$ of the agent.
- Probability transitions from state s to s' under action a:

$$P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$$

- The immediate reward after transition from state s to s' under action a :

$$R_a(s, s')$$

- The MDP and agent together give rise to a trajectory :

$$S_0, A_0, R_1, S_1, A_1, R_2 \ldots$$

Goals and Rewards

- A reward R_t is a scalar feedback which indicates how well the agent is doing at time t - this defines our goal.
- The agent's job is to maximise its cumulative reward or *return*:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots,$$
$$\gamma \in [0, 1]$$

- Reinforcement learning is based on the *reward hypothesis* :

"Any goal can be formalized as the outcome of maximizing a cumulative reward."

Policy, State and Action value function

- The policy defines the agent's behaviour, it can be
  - deterministic : $A = \pi(S)$
  - stochastic : $\pi(A|S) = p(A|S)$
- State-Value function:

$$V_\pi(s) = E_\pi[G_t|S_t = s]$$

- Action-Value function:

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a]$$

Our approach so far

Q-Learning Algorithm : Learning the Q-table, using an *ε-greedy policy.*

- Agent is in a state s
  - If we knew the real Q-table, we would pick $\arg\max_a Q(s,a)$ every time.
  - But we need to learn it !

- Initialize the Q-Table with 0.
  - Use an *ε-greedy policy* :

    With probability ε, pick an action at random, otherwise take $\arg\max_a Q(s,a)$.

⟶ *Exploration vs Exploitation trade-off*

	actions			
states	a_0	a_1	a_2	...
s_0	$Q(s_0,a_0)$	$Q(s_0,a_1)$	$Q(s_0,a_2)$	...
s_1	$Q(s_1,a_0)$	$Q(s_1,a_1)$	$Q(s_1,a_2)$	...
s_2	$Q(s_2,a_0)$	$Q(s_2,a_1)$	$Q(s_2,a_2)$	...
⋮	⋮	⋮	⋮	⋮

Our approach so far

Q-Learning Algorithm : Learning the Q-table, using an ε-greedy policy.

1. Initialize $Q(s, a)$ to zero for all state-action pairs
2. For each episode:
   - 2.1. For each step:
     - 2.1.1. With probability ε, pick an action at random otherwise take $\arg\max_a Q(s, a)$.
     - 2.1.2. Observe the reward and the new state
     - 2.1.3. Update the Q-table with the following formula:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

   - 2.2. Until the state is terminal

What's next

Deep Reinforcement Learning

- Allows reinforcement learning to be applied to larger problems.
- Typically use function approximation to appreciate a state-action value instead of tabular methods.
- Allows a more complex description of the state of the environment $\longrightarrow$ adding features !

Combinatorial optimization approach

1 Minimum cost multicommodity flows

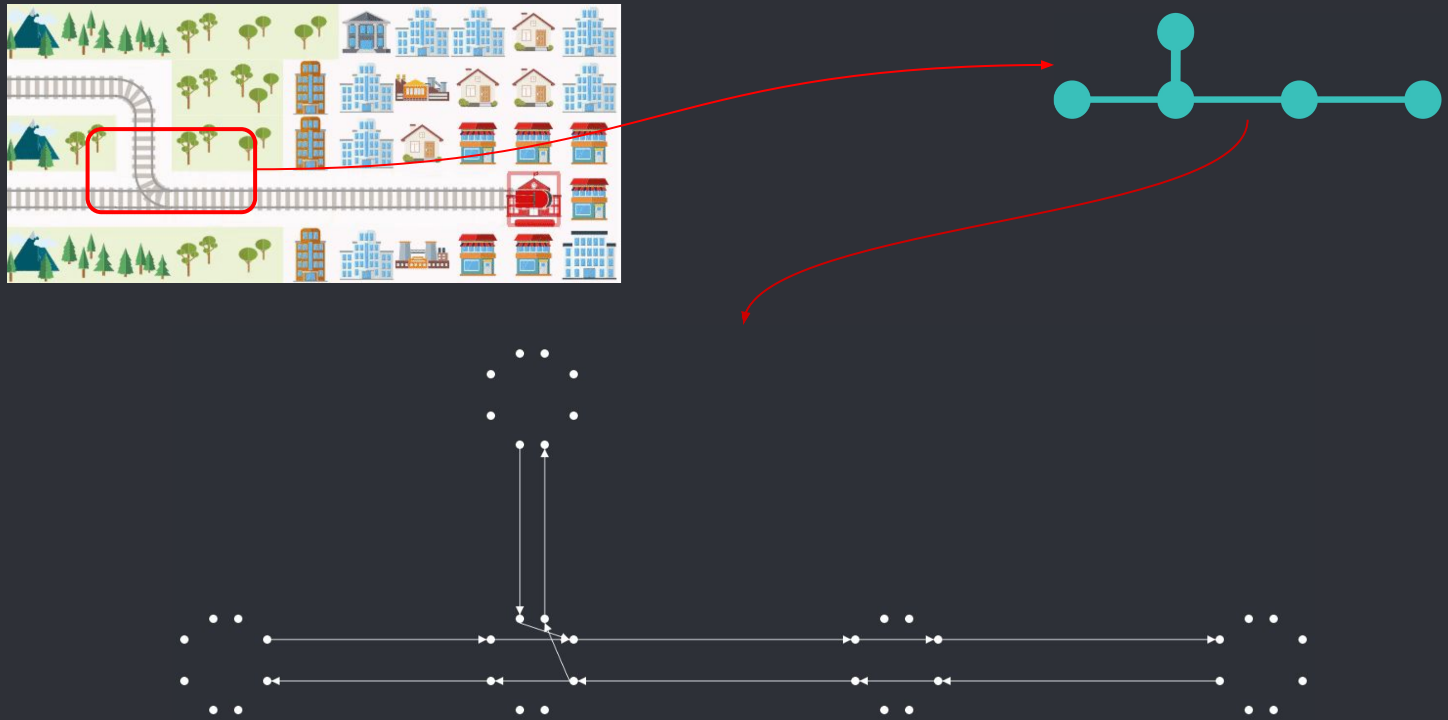
The mathematical abstraction

Implementation of minimum cost multicommodity flows

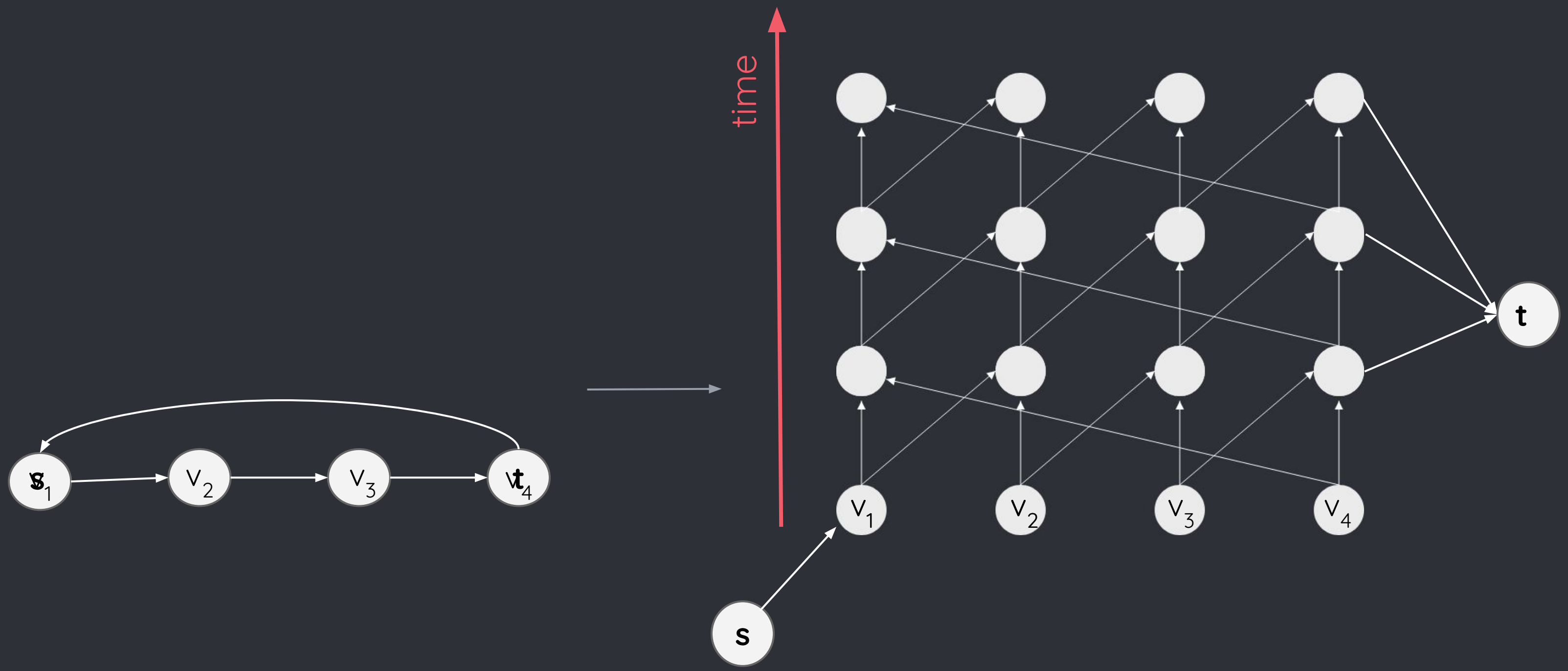
We need to :

- Extract a graph
- Respect the grid world constraints
- Minimize the same cost function

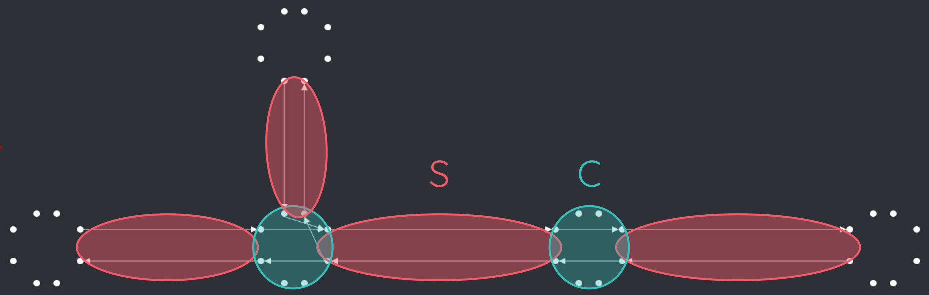
Extracting the graph from the environment, Transition graph

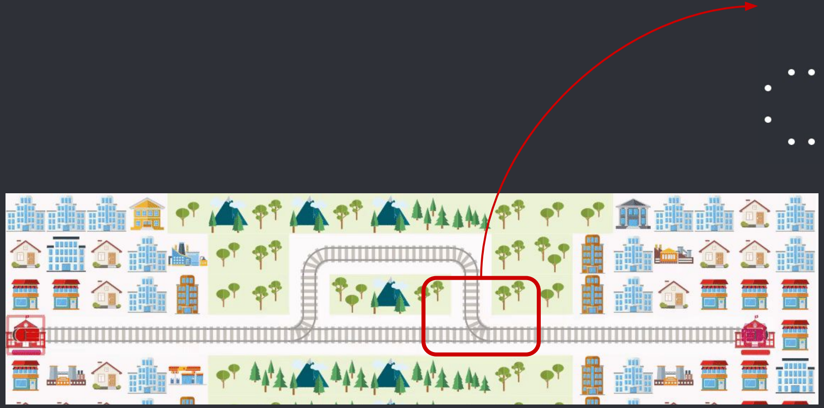


Extracting the graph from the environment, Time expanded graph



Respecting the environment constraints: Position constraints





Localization → $$\sum_k \sum_{(i,j)\in C} x_{ij}^k \leq 1 \quad \forall C$$

Swapping → $$\sum_k \sum_{(i,j)\in S} x_{ij}^k \leq 1 \quad \forall S$$

Formal presentation

$G = (V, A)$ directed graph

$$\text{minimize} \sum_{k=1}^{K} \sum_{(i,j)\in A} x_{ij}^k$$

Sum of time steps for all agents

$$\begin{aligned}
\text{subject to} \quad & \sum_{k=1}^{K} x_{ij}^k \leq 1, && \forall (i,j) \in A \\
& \sum_{k=1}^{K} \sum_{(i,j)\in S} x_{ij}^k \leq 1, && \forall S \\
& \sum_{k=1}^{K} \sum_{(i,j)\in C} x_{ij}^k \leq 1, && \forall C \\
& \mathcal{N} x^k = b^k && k \in [K] \\
& x_{ij}^k \in \{0,1\}, && \forall (i,j) \in A, k \in [K]
\end{aligned}$$

Capacity constraints

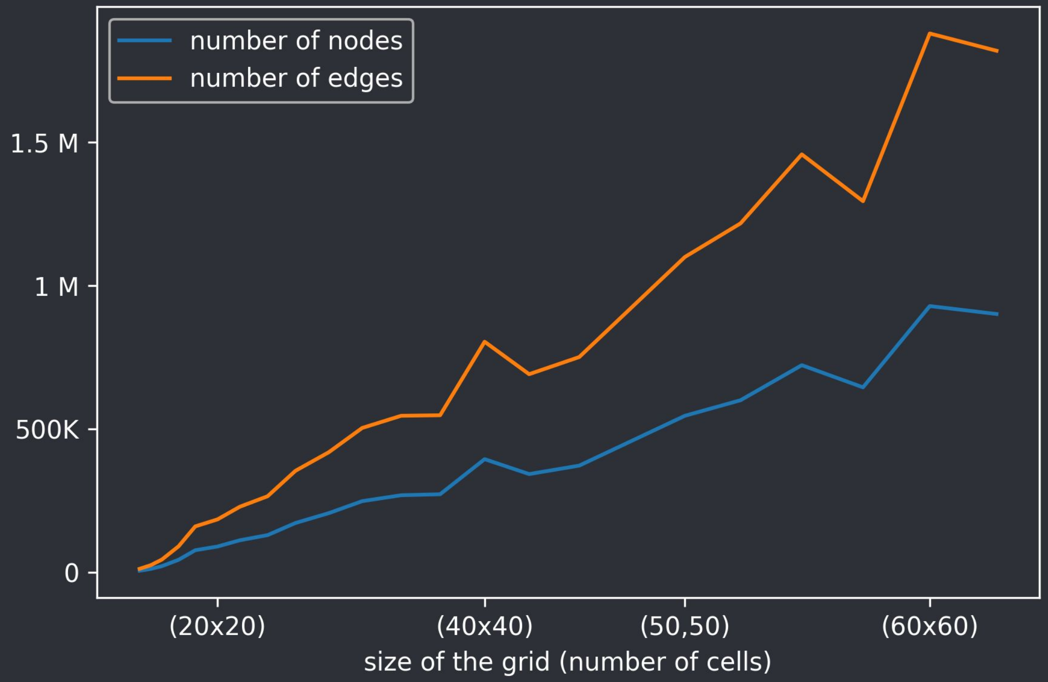
“Position” constraints

Flow conservation &

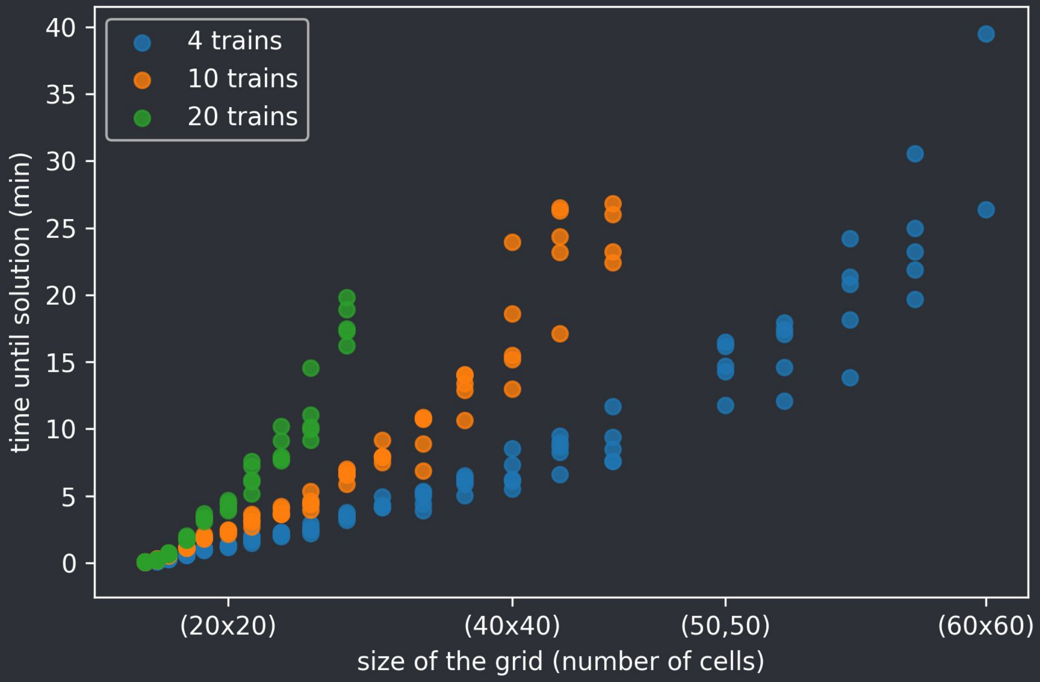
Supply and demand

2 Results

Size of the time expanded graph



Experiments with the multi-commodity flow formulation until memory error (model > 40 Gbs)



Clear limitations of the actual modelization

What's next

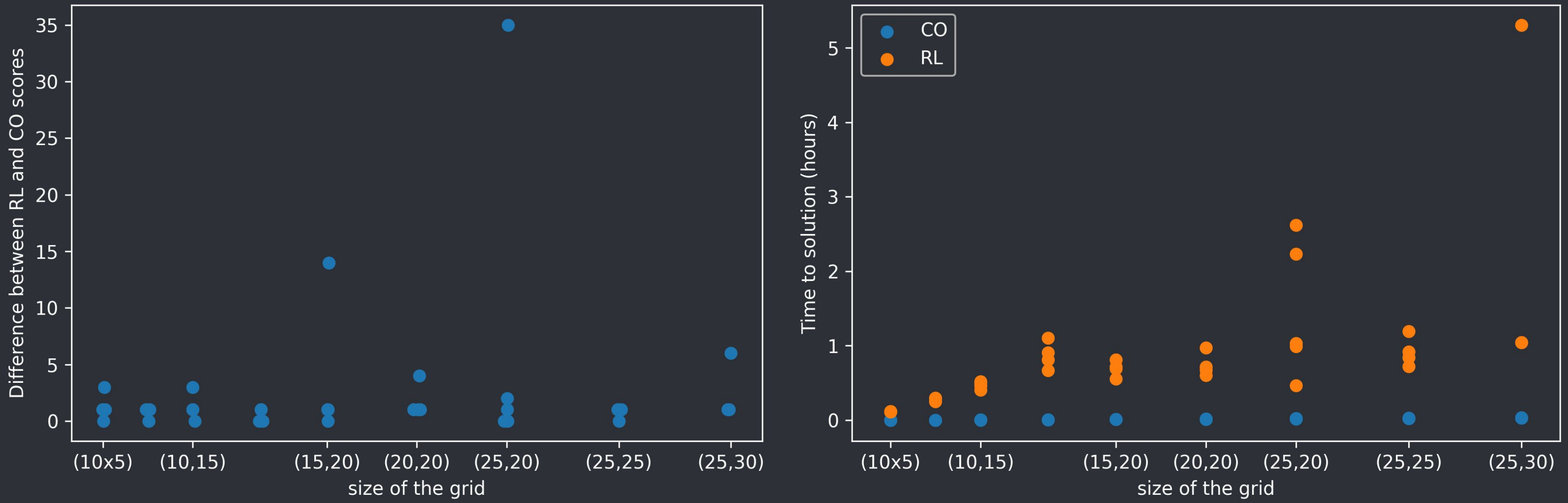
Multicommodity flow

- Merge unnecessary paths
- Smarter constraints
- Columns generation methods

CSP formulation

- Constraint satisfaction problem formulation
- New graph definition to reduce size
- Linear program to solve conflicts

Experimental comparison



Comparison of the two approaches in term of score and time with 2 trains
Combinatorial Optimization (CO), Reinforcement Learning (RL)

Thanks!

ANY QUESTIONS?

Technical annexe

Annexe RL

Return:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} \dots$$

Discounted Return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots$$
$$\gamma \in [0, 1]$$

State Value function:

$$V_\pi(s) = E_\pi[G_t | S_t = s] = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots | S_t = s],$$
$$\gamma \in [0, 1], \forall s \in \mathcal{S}$$

Action-State Value function

$$q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a]$$

Annexe RL (2)

Optimal policies and value functions

For two policies, we have the partial ordering: $\pi' \geq \pi \iff v_{\pi'}(s) \geq v_{\pi}(s), \forall s \in \mathcal{S}$

For finite MDPS, with discrete actions space, discrete state space and bounded rewards, there is always at least one policy *better than or equal* to any other policy. We call these policies *optimal policies*. We denote all the optimal policies by π_*.

These share the same state-value functions as well as action-state value functions, denoted and defined by:

$$v_*(s) = \max_{\pi} v_{\pi}(s), \quad \forall s \in \mathcal{S}$$
$$q_*(s,a) = \max_{\pi} q_{\pi}(s,a), \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$$

With the relationship:

$$q_*(s,a) = E[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$

Annexe CO