École polytechnique fédérale de Lausanne

Disopt

Master semester project

Autumn Semester 2019

A combinatorial approach to the trains routing problem

Student:
Charles Dufour

Supervisors:
Prof. Friedrich Eisenbrand
Jonas Racine

Contents

1 Introduction **1**

2 Modelization of the environment **2**
2.1 Transition network . . . 2
2.2 Time expanded network . . . 3
2.3 Restrictions . . . 4
2.3.1 On the transition network . . . 4
2.3.2 On the time expanded network . . . 5
2.4 Dealing with different trains and stochastic events . . . 5
2.4.1 Different speeds . . . 5
2.4.2 Stochastic events . . . 6

3 Minimum cost multicommodity flow **7**
3.1 Arc flow formulation . . . 7
3.2 Column generation method . . . 7
3.2.1 Reformulation as a path flow problem . . . 8
3.2.2 Column generation method . . . 10
3.2.3 Relaxation and approximation . . . 12

4 Experimental results **13**
4.1 Arc formulation and flow formulation . . . 13
4.1.1 Arc formulation . . . 13
4.1.2 Flow formulation . . . 13
4.1.3 Comparison of the two formulations . . . 13
4.2 Comparison with reinforcement learning approach . . . 14

5 On the mixed RL-CO approach **15**

6 Conclusion **16**

7 Annexes **17**
7.1 More experiences . . . 17
7.2 Technical details about the implementation . . . 17

References **18**

1 Introduction

All the code can be found in the repository (2019).

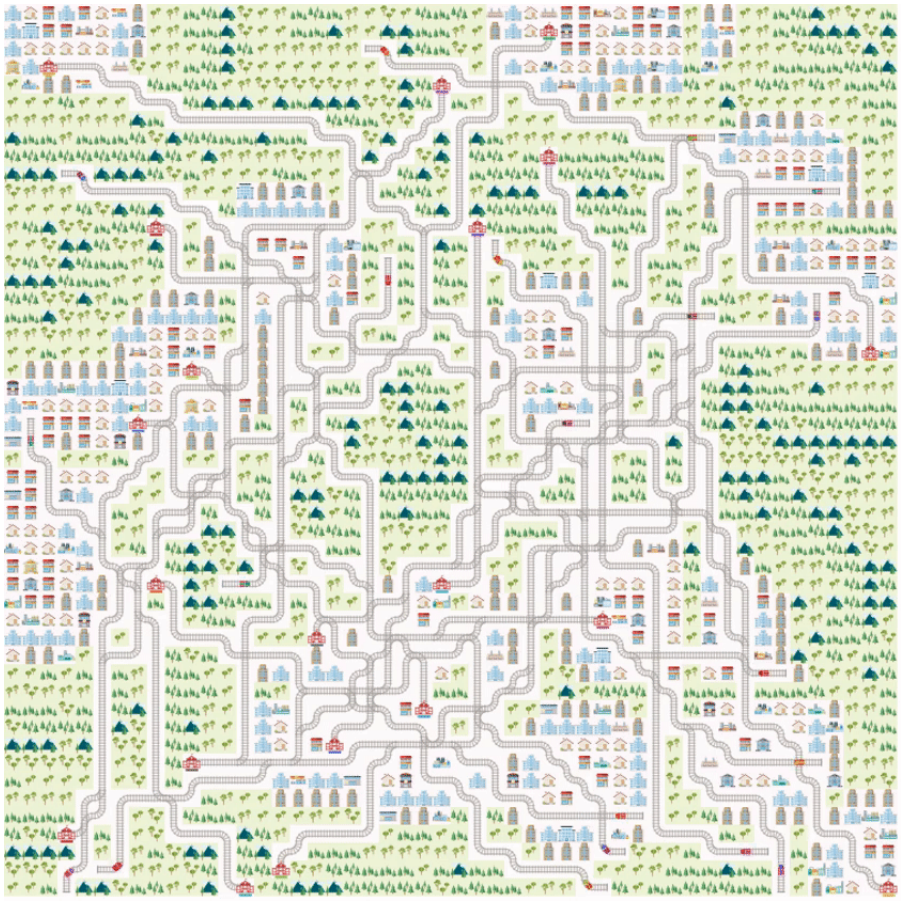


Figure 1: Example of the railway network environment from the Flatland challenge

2 Modelization of the environment

To have a formulation of our problem in term of dynamic multicommodity flows (*multicommodity flows over time*), we need to first define the graph representing the railway network from the 2D grid world. Then we need to embed the time component in our model using a time expanded network, bringing our problem back to a static multicommodity flow problem. We first model the problem with a unique speed for all trains.

2.1 Transition network

We create a graph to model all the possible switches and rails in the rail network from Flatland. In order to do so, we model each cell as a super node containing 8 internal nodes, as can be seen in Figure 2. These internal nodes are responsible for correctly implementing the different transitions the original cell allows.

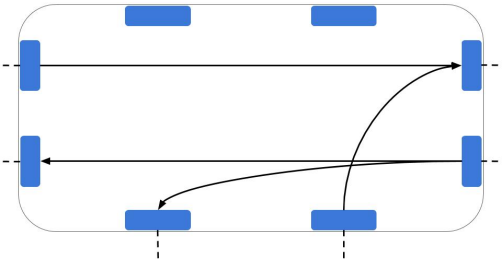
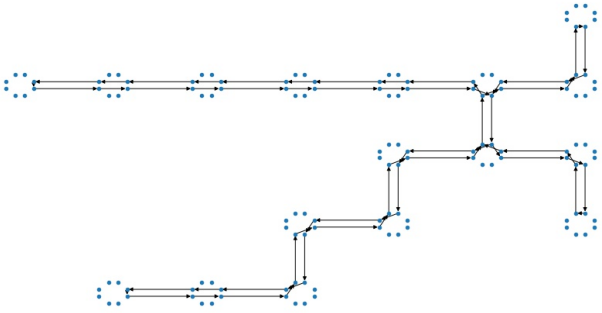


Figure 2: Super node representing the cell represented in Figure 3. The blue rectangles represent the internal nodes of the cell, where the big white rectangle represents the super node.

Figure 3: A switch from the railway network.

The transition network is an oriented graph $G = (V, A)$ that represents the original 2D grid world. Figure 4 shows an example of a randomly generated environment and its corresponding transition graph.

Flatland environment.



Transition graph.

Figure 4: Randomly generated 7×5 grid and its extracted transition graph.

Why did we not use a single node to represent each cell and connect it to its neighbors if there is a connection between them? The problem with the simple node modelization is that it allows transition that are not represented in the original railway network. Take for example the switch in Figure 3: with a simple node representation, a train coming from a cell below could go to the left whereas in the original cell it should not be possible.

There exists alternatives to model railway networks, namely the double vertex graphs (see for example the modelization done in Jaddoe (2005)). Those were not considered here since they are not simple directed graphs. Indeed they have a particular definition of paths, so the usual flows algorithm would not be straightforward to apply.

2.2 Time expanded network

We now have to include the time component in our design. We will use a time expanded network to bring back our dynamic multicommodity flow problem to a static formulation.

Intuitively we define a discrete time step (given by the environment in our case) and a maximum time horizon T. Now we do T copies of the nodes of our graph, and only allow transition from one time step to another. Figure 5 shows the intuition by drawing a path in a time expanded network.

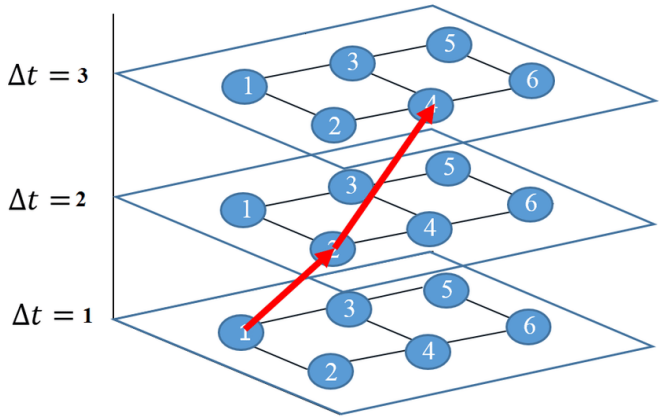


Figure 5: Explanation of the intuition of time expanded networks with only the path $\{1, 2, 4\}$ drawn in time (image from Masoud & Jayakrishnan (2017)).

We now give a formal definition from (Skutella, 2008, p. 19):

Definition 2.1 (Time-expanded network)**.** Let $G = (V, E)$ be a network with capacities u and costs c on the arcs. For a given time horizon $T \in \mathbb{Z}_{>0}$, the corresponding *time-expanded network* $G^T = (V^T, E^T)$ with capacities and costs on the arcs is defined as follows. For each node $v \in V$ we create T copies $v_0, v_1, \ldots, v_{T-1}$, that is,

$$V^T := \{v_\theta | v \in V, \theta = 0, 1, \ldots, T-1\}.$$

For each arc $e = (v, w) \in E$, there are $T-1$ copies $e_0, e_1, \ldots, e_{T-2}$ where arc e_θ connects node v_θ to node $w_{\theta+1}$. Arc e_θ has capacity $u_{e_\theta} := u_e$ and cost $c_{e_\theta} := c_e$. Moreover, E^T contains *waiting* arcs $(v_\theta, v_{\theta+1})$ for $v \in V$ and $\theta = 0, \ldots, T-2$. The capacity of waiting arcs is 1 and they have

cost 1. Summarizing, the set of arcs E^T is given by

$$\begin{aligned} E^T :=& \{e_\theta = (v_\theta, w_{\theta+1}) | e = (u, v) \in E, \theta = 0, 1, \dots, T-2\} \\ & \cup \{(v_\theta, v_{\theta+1}) | v \in V, \theta = 0, 1, \dots, T-2\} \end{aligned}$$

Notice that the size of the time-expanded network G^T is linear in T and therefore only pseudo-polynomial in the input size.

We then proceed to define a time step, which represent what would be added to a time expanded graph, were we to expand the horizon by one.

Definition 2.2 (Time step in time expanded network)**.** A *time step* at time θ is the set of all edges (v_θ, w) for all $v \in V$, $w \in V^T$ such that $(v_\theta, w) \in E^T$. It represents the set of all edges starting at time θ.

Remark. There are no cycles in the time expanded network since $\nexists (v_\theta, w_{\tilde{\theta}}) \in E^T$ with $\tilde{\theta} < \theta$.

2.3 Restrictions

In the environment from Flatland some position are forbidden: two trains cannot be at the same cell at the same time, and to trains can collide, meaning they cannot swap positions. We will first define the restrictions needed on the transition network and then explain how to impose the same restriction on the time expansion of this network.

A restriction will impose that among a certain set of edges, only one of them can be used at the same time.

2.3.1 On the transition network

- Position constraint
  All the edges leading to specific supernode should be *restricted*. Using these restrictions, we ensure that at any time step, at most one train will be in any cell, thus avoiding collisions.
- Swapping constraint All pairs of edges between two supernodes shouls be *restricted*. It is trivial to see that these restrictions will be enough to prevent swapping.

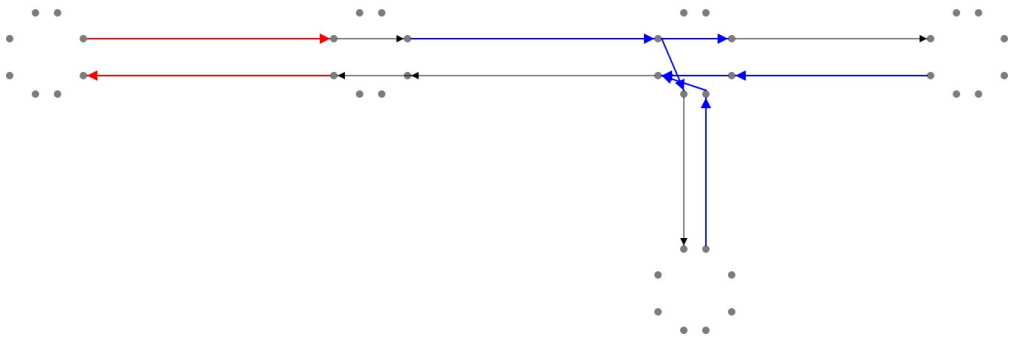


Figure 6: Representation of some restrictions in the transition network. All the edges colored in the same color (blue or red) cannot be used at the same time. The blue edges represent a position restriction and the red edges a swapping restriction.

2.3.2 On the time expanded network

The restrictions on the time expanded network can be easily derived from the restrictions described before. Suppose we have a restriction on $\{(u, v), (i, j)\}$. In the time expanded network, we will have restrictions for all sets $\{(u_t, v_{t+1}), (i_t, j_{t+1})\}$ $\forall t \in \{1, \dots, T-1\}$.

We also have to add to the position constraints the waiting edges. Let's say we consider a cell i with its corresponding supernode, all the waiting edges of this supernode should be added to the position constraint linked to cell i.

Remark. Due to the nature of the restrictions, a path in the time expanded network can only take one edge per restriction. Indeed, otherwise it would mean that at a certain time t, the path takes two edges in the transition network at the same time which is impossible.

2.4 Dealing with different trains and stochastic events

For now, the trains were supposed to all have the same speed and there were supposed to be no changes in the underlying railway network over time. This is not realistic, so in order to have a better model, we discuss now what could be implemented to handle a system with multiple speeds and random breakdown of trains.

2.4.1 Different speeds

In the Flatland challenge, trains can have different speeds. The speeds will be defined as follow. The faster trains will have a speed of 1, meaning that it takes them one time step to go between two nodes in the transition graph. Then all the other train will have a speed $v \in [0, 1]$ with v representing the percentage of the edge that is traveled by said train in one time step. For example, if a train a speed 1/3, it will take 3 time steps to travel through an edge.

Now suppose we have only two different speeds (the discussion easily generalize to k different speeds), namely 1 and v (w.l.o.g. we say that one of the two speeds is 1 otherwise we would just rescale all of them).

We can then build two time expanded networks, one for each speed. The tricky parts are the restrictions: they now span the two networks. Indeed, now one train takes v^-1 time steps to go through an edge, while the other takes only one time step, so it is necessary to have the constraints spanning the 2 networks.

The way to do it is to have the fastest train's network as a point of reference. This one has the usual constraints, to which we had edges from the other graph. Take a constraint, and look at the edges contained in it. You then take the corresponding edges (representing the same railway piece), and add them for the corresponding time steps.

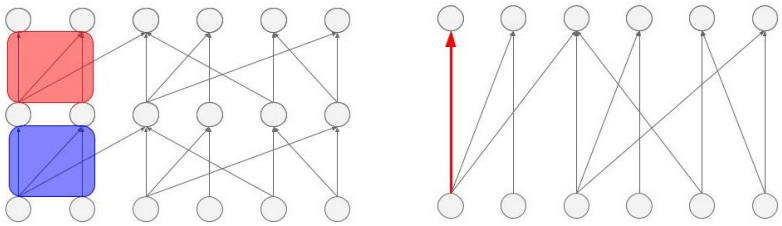


Figure 7: On the left the fast expanded network, on the right the slow ($v = 0.5$). The red edges on the right will belong to both the red and blue restrictions showed on the left (fictional restriction sets).

For example if the second speed was 0.5, we would have edges (u_0, v_1),(u_1, v_2) in the fast time expanded network, and only (u_0, v_2) in the slow network. For the restriction containing (u_0, v_1) we would have to add (u_0, v_2) and the same for the one containing (u_1, v_2). Figure 7 shows this.

A formal way to say this would be that for any edge $(u_{t1}, v_{t2})'$ in the slow network it has to be added to any constraint from the fast network (u_t, v_{t+1}) if $t1 \geq t$ and $t2 \leq t + 1$.

One could argue about building only one network combining the edges of our two networks. The reason we build two different network is for efficiency when finding minimum weighted path during the column generation pricing problem (see subsection 6).

2.4.2 Stochastic events

In the case of failure in the network, there will be unusable tracks (cells) for a certain period of time (known or unknown depending on the setting). In this case one can still use our method. When a failure happens, suppose k trains break down among the n in the network. The cells occupied by the broken trains will be unusable for a certain period of time T. We now propose multiple approaches.

- Restart from scratch all the trains, taking as initial position their position at failure time. When the trains are reactivated (failure is resolved) solve once more to move all the trains to their destination.
- Reroute only the trains affected by the failure in a modified time expanded network

The following sections considers one unique speed for all the trains and no stochastic events. This can easily be adapted following the above discussion.

3 Minimum cost multicommodity flow

We now consider the time expanded network defined in section 2.2 and denote it by $G = (V, A)$. The trains will be the commodities, we suppose that we have K of them.

We now formalize the definition of restriction as described in section 2.3.

Definition 3.1. A restriction R over a time expanded network $G = (V, A)$ is a set of arcs: $R \subset A$ over which we want to impose certains specification (e.g. global capacity).

We define C_R the set of all restrictions over the time expanded network G.

We also introduce a notation:

Definition 3.2. Given α a set,

$$\delta_\alpha(\beta) = \begin{cases} \mathbb{1}\{\beta \cap \alpha \neq \emptyset\} \text{ if } \beta \text{ is a set} \\ \mathbb{1}\{\beta \in \alpha\} \text{ otherwise} \end{cases}$$

Informally this represents the fact that α and β are not disjoint or that β is contained in α.

3.1 Arc flow formulation

The minimum cost multicommodity flow can be formulate as an arc-flow integer program:

$$\begin{aligned} \text{minimize} \quad & \sum_{k=1}^{K} \sum_{(i,j)\in A} x_{ij}^k \\ \text{subject to} \quad & \sum_{k=1}^{K} x_{ij}^k \leq 1, && \forall (i,j) \in A \\ & \sum_{k=1}^{K} \sum_{(i,j)\in R} x_{ij}^k \leq 1, && \forall R \in C_R \\ & Nx^k = b^k, && \forall k \in \{1, \ldots, K\} \\ & x_{ij}^k \in \{0, 1\}, && \forall (i,j) \in A \end{aligned} \tag{3.1.1}$$

Where N^k is the -arc adjacency matrix, and b^k is such that

$$b_i^k = \begin{cases} 1 \text{ if node } i \text{ is a source for commodity } k \\ -1 \text{ if node } i \text{ is a sink for commodity } k \\ 0 \text{ otherwise} \end{cases}$$

This formulation is clear and intuitive: for each commodity we decide whether or not it will use a specific arc at a certain time by setting x_{ij}^k to 1 or to 0. But this formulation (3.1.1) is not scalable due to its high number of restrictions and variables (see section 4.1 for a more detailed explanation). We then proceed to find new ways to solve this problem.

3.2 Column generation method

In this section, we give a column generation solution procedure that works with arbitrary restrictions, as long as the restrictions do not span more than one time step (see section 2.3 for an explanation). We directly consider the relaxation problem.

3.2.1 Reformulation as a path flow problem

For this section we will restate our problem using path flows instead of arc flows as before. In this paradigm, we suppose that we list all the possible paths between the sources and targets, $\mathcal{P}$. Then for each path $P \in \mathcal{P}$ we decide how much we send along this path with the variable $f(P)$.

The problem then becomes:

$$\begin{aligned} \text{minimize} \quad & \sum_{P \in \mathcal{P}} f(P)|P| \\ \text{subject to} \quad & \sum_{P \in \mathcal{P}_R} f(P) \leq 1, \quad \forall R \in C_R \\ & \sum_{P \in P^k} f(P) = 1, \quad \forall k \in \{1, \dots, K\} \\ & f(P) \geq 0, \quad P \in \mathcal{P} \end{aligned} \tag{3.2.1}$$

With:

- $\mathcal{P} = \{\mathcal{P}_1, \dots, \mathcal{P}_K\}$ and $\mathcal{P}_i$ is all the possible distinct paths between s_i and t_i.
- $\mathcal{P}_R = P \in \mathcal{P} : |P \cap R| > 0$. This represents the paths that "goes trough" the restriction R.

The second restriction $\sum_{P \in P^k} f(P) = 1$ contains in our case the fact that we need to have exactly one train going from s_k to t_k for each commodity.

The dual of the primal formulation (3.2.1) is:

$$\begin{aligned} \text{maximize} \quad & \sum_{R \in C_R} y_R + \sum_{i=1}^{K} \sigma_i \\ \text{subject to} \quad & \sum_{R \in C_R} \delta_P(R) \cdot y_R + \sigma_k \leq |P|, \quad \forall P \in \mathcal{P} \\ & y \leq 0, \qquad\qquad y \in \mathbb{R}^{|C_R|} \\ & \qquad\qquad\quad \sigma \in \mathbb{R}^K \end{aligned} \tag{3.2.2}$$

With respect of the dual variables,the reduced cost $c_P^{\sigma,y}$ for each path flow variable $f(P)$ which belongs to commodity k is :

$$c_P^{\sigma,y} = |P| - \sum_{R \in C_R} \delta_P(R) \cdot y_R - \sigma_k \tag{3.2.3}$$

We can then derive complementary slackness conditions.

Theorem 3.1 (Path flow complementary slackness conditions)**.** *The commodity pah flow $f(P)$ are optimal in the path flow formulation* (3.2.1) *of the multicommodity flow problem if and only if for some restriction prices y_R and commodity prices σ_k, the reduced cost and arc flows satisfy the following complementary slackness conditions:*

$$y_R \left[\sum_{k \in [K]} \sum_{P \in P^k} \delta_P(R) \cdot f(P) - 1 \right] = 0 \text{ for all } R \in C_R. \tag{3.2.4}$$

$$c_P^{\sigma,y} \geq 0 \text{ for all } k \in [K] \text{ and all } P \in P^k. \tag{3.2.5}$$

$$c_P^{\sigma,y} \cdot f(P) = 0 \text{ for all } k \in [K] \text{ and all } P \in P^k. \tag{3.2.6}$$

Proof of theorem 3.1.
We show that optimality of the primal (3.2.1) implies the complementarity slackness conditions from theorem 3.1.

Denote $f^*(\mathcal{P}), y_S^*$ and σ_k^* the optimal solution from (3.2.1) and (3.2.2). Since y^* and σ^* are solution of the dual formulation we get condition (3.2.5) directly.

To show the other two conditions, we will write the primal and dual expression in matrix form.

Primal

$$\begin{aligned} \text{minimize} \quad & b^T f(\mathcal{P}) \\ \text{subject to} \quad & \begin{pmatrix} -A_{C_R} \\ A_K \\ -A_K \end{pmatrix} \cdot f(\mathcal{P}) \geq \begin{pmatrix} -1_{|C_R|} \\ 1_K \\ -1_K \end{pmatrix} \\ & f(\mathcal{P}) \geq 0 \end{aligned}$$

With:

- $b \in \mathbb{R}^{|\mathcal{P}|}, \quad b_P = |P|$ is the vector containing the length of the paths.
- $A_{C_R} \in \mathbb{R}^{|C_R| \times |\mathcal{P}|}$ where $(A_{C_R})_{ij}$ is 1 if the i^{th} restriction contains an edge that belongs to the j^{th} path and 0 else.
- $A_K \in \mathbb{R}^{K \times |\mathcal{P}|}$ where $(A_K)_{ij}$ is 1 if the j^{th} path belongs to $\mathcal{P}_i$ (i.e. belongs to the i^{th} commodity) and 0 else.

Dual

$$\begin{aligned} \text{maximize} \quad & \left(-1_{|C_R|}^T, 1_K^T, -1_K^T\right) \cdot \tilde{y} \\ \text{subject to} \quad & \left(-A_{C_R}^T, A_K^T, -A_K^T\right) \cdot \tilde{y} \leq b \\ & \tilde{y} \geq 0 \end{aligned}$$

Where $\tilde{y} = \begin{pmatrix} -y \\ \tilde{\sigma_1} \\ \tilde{\sigma_2} \end{pmatrix}$ with $y \in \mathbb{R}^{|C_R|}$ and $\tilde{\sigma_1} + \tilde{\sigma_2} = \sigma \in \mathbb{R}^K$, with y, σ being as (3.2.2).
Then we rewrite the conditions (3.2.4) and (3.2.6) in matrix form:

$$\left[\left(\left(-1_{|C_R|}^T, 1_K^T, -1_K^T\right) - f(\mathcal{P})^T \left(-A_{C_R}^T, A_K^T, -A_K^T\right)\right) \cdot \tilde{y}\right]_i = 0 \qquad \text{for all } i \in \{0, 1 \ldots, |C_R|\}$$
$$f(\mathcal{P})^T \left(b - \left(-A_{C_R}^T, A_K^T, -A_K^T\right) \tilde{y}\right) = 0$$

By the weak duality theorem we have:

$$\left(-1_{|C_R|}^T, 1_K^T, -1_K^T\right) \cdot \tilde{y} \leq f(\mathcal{P})^T \left(-A_{C_R}^T, A_K^T, -A_K^T\right) \tilde{y} \leq f(\mathcal{P})^T b$$

Using the strong duality theorem, we also have

$$\left(-1_{|C_R|}^T, 1_K^T, -1_K^T\right) \cdot \tilde{y} = f(\mathcal{P})^T b$$

Combining the two results from strong an weak duality we get the following two equalities:

$$\left(-1^T_{|C_R|}, 1^T_K, -1^T_K\right) \cdot \tilde{y} = f(\mathcal{P})^T \left(-A^T_{C_R}, A^T_K, -A^T_K\right) \tilde{y}$$
$$f(\mathcal{P})^T \left(-A^T_{C_R}, A^T_K, -A^T_K\right) \tilde{y} = f(\mathcal{P})^T b$$

Which is exactly what we aimed to obtain.
The other direction is similar.

□

3.2.2 Column generation method

For each train, we will only use one path in the time expanded network, so even if the path formulation (3.2.1) leads to an exponential number of variables, actually only K of them will be useful in the end.

With this remark in mind, the purpose of the column generation method is to gradually improve a feasible solution containing few variables until we have the optimal solution. The idea is that in a few iterations we should be able to find the optimal solution, leading us to consider a small number of variables compared to all the possible paths set.

We define the restricted linear program on a basis (set of variables) as the linear program (3.2.1) setting all variables at 0 except for the variable contained in the basis.

Algorithm 1: Column generation algorithm

```
Result: Optimal solution to (3.2.1)
Input: Basis: initial feasible solution
while solution optimal do
    Solve restricted linear program on the basis;
    Solve pricing problem to find useful variables to add;
    Add these variables to the basis;
end
Return solution of restricted linear program on the basis;
```

Pricing problem Following (Ahuja et al., 1993, p. 669), we consider only (3.2.5). Indeed for any basis use for the restricted linear program, the conditions (3.2.4) and (3.2.6) are automatically respected (can be seen by noticing that these conditions only depend on element in the basis of the LP). The only indication of the basis being not optimal is the second condition.

If we find a path with a negative reduced cost, we can add it to the basis and improve our actual cost. We see now that the pricing problem can be efficiently solved.

Solving the pricing problem efficiently Because of the particular nature of the restrictions in our problem, one can see that no restriction can span more than one time step in the time expanded network, and that a path cannot have two edges in the same time step (see section 2.3.2).

Based on these observations, we define the cost of an arc in the graph as:

$$w_{ij} = -\sum_{R \in C_R} \delta_R((i,j)) \cdot y_R \geq 0$$

One can notice that

$$\sum_{(i,j)\in P}\left(\sum_{R\in C_R}\delta_R((i,j))\cdot y_R\right)=\sum_{R\in C_R}\delta_P(R)\cdot y_R$$

We can then rewrite the reduced cost as:

$$c_P^{\sigma,y}=|P|+\sum_{(i,j)\in P}w_{ij}-\sigma_k$$

So equation (3.2.5) becomes:

$$\begin{aligned}
c_P^{\sigma,y} &\geq 0 \quad \forall P\in P^k\\
|P|+\sum_{(i,j)\in P}w_{ij}-\sigma_k &\geq 0 \quad \forall P\in P^k\\
|P|+\sum_{(i,j)\in P}w_{ij} &\geq \sigma_k \quad \forall P\in P^k\\
\sum_{(i,j)\in P}(w_{ij}+1) &\geq \sigma_k \quad \forall P\in P^k\\
\min_{P\in P^k}\sum_{(i,j)\in P}(w_{ij}+1) &\geq \sigma_k
\end{aligned}$$

This shows that the pricing problem can efficiently be solved by searching for a weighted shortest paths p_k^* between s_k and t_k for all commodities $k\in[K]$. The weight for each arc $(i,j)\in A$ is given by $\tilde{w}_{ij}=w_{ij}+1$. We can use Dijkstra's algorithm to efficiently solve this problem since the weights of the edges are positive.

Initial Solution We produce an initial feasible solution using a greedy algorithm.

Algorithm 2: Finding and initial feasible solution for the column generation method

```
Result: {p_1^0, ..., p_K^0}, where p_k^0 is a path from s_k to t_k
1  pathsFeasible = [ ];
2  Set weight at 1 for all edges;
3  for k ∈ {1, ..., K} do
4  |  find shortest path candidate p̃ from s_k to t_k using Dijkstra's algorithm ;
5  |  if p̃ does not have any conflicts with pathsFeasible then
6  |  |  add p̃ to pathsFeasible;
7  |  end
8  |  else
9  |  |  Increase weight of all edges of p̃ by 1;
10 |  |  Goto 4;
11 |  end
12 end
```

Where we say that a path p^* does not have conflicts with a set of paths P if using formulation (3.2.1) and putting all $f(p)=1 \quad \forall p\in P\cup\{p^*\}$ all the constraints are satisfied.

3.2.3 Relaxation and approximation

After having a set of optimal paths for the fractional formulation of (3.2.1) we have to get an integral solution.

check article from Jonas: Desrosiers & Lubbecke (2011)

4 Experimental results

To solve the linear programs we use the commercial solver Gurobi Optimization (2019). To obtain the integral solution we use their optimized algorithms which might differ from what is described in section 3.2.3. The rest of the implementation is in Python and can be found at repository (2019).

4.1 Arc formulation and flow formulation

4.1.1 Arc formulation

talk about limitations

4.1.2 Flow formulation

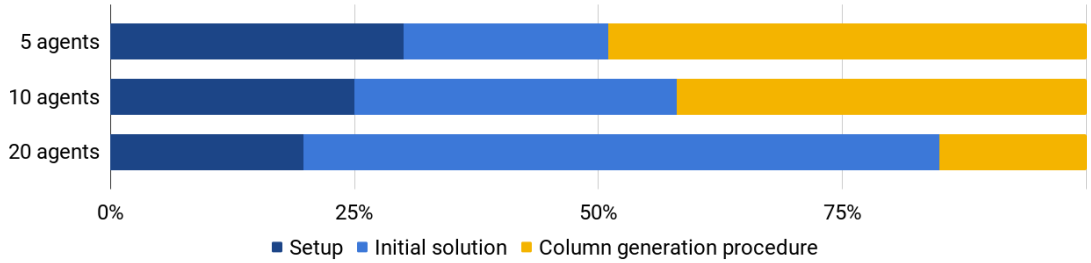


Figure 8: Average time division in the column generation method.

4.1.3 Comparison of the two formulations

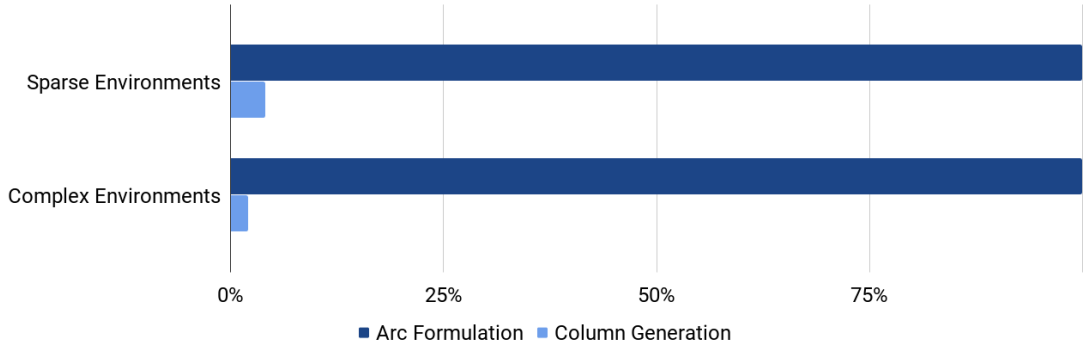


Figure 9: Comparison of the average relative speeds between the two formulation. The values were obtained by doing multiple runs on different random environments and computing the relative difference between the two methods.

Explain the memory issues with the arc formulation

4.2 Comparison with reinforcement learning approach

Mostly about the difference in terms of approach

5 On the mixed RL-CO approach

6 Conclusion

What can be improved

- Initial solution generator (avoid infinite loop by randomizing order of priority)
- merge unnecessary paths
- better data structure
- more robust handling of environment (allow "spawning")

Other approaches such are double vertex graph (Jaddoe (2005)) and multiple agent path finding (MAPF) could be considered, but were not in this project.

7 Annexes

7.1 More experiences

7.2 Technical details about the implementation

we only have to sum over activated constraints.

References

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice hall.

Dai, W., Zhang, J., & Sun, X. (2017). On solving multi-commodity flow problems: An experimental evaluation. *Chinese Journal of Aeronautics*, *30*(4), 1481 – 1492.
URL `http://www.sciencedirect.com/science/article/pii/S100093611730122X`

Desrosiers, J., & Lubbecke, M. (2011). Branch-price-and-cut algorithms.

Gurobi Optimization, L. (2019). Gurobi optimizer reference manual.
URL `http://www.gurobi.com`

Jaddoe, V. (2005). Selecting sets of ddisjoints paths in a railway graph, *Bachelor thesis*.

Masoud, N., & Jayakrishnan, R. (2017). A decomposition algorithm to solve the multi-hop peer-to-peer ride-matching problem. *Transportation Research Part B: Methodological*, *99*, 1–29.

Pfetsch, M. (2006). Lecture notes in multicommodity flows and column generation, *Technische Universität Berlin*.

repository (2019). Comparison of standard combinatorial optimization with reinforcement learning techniques on rescheduling problems, by charles dufour and edouard ghaleb. `https://github.com/dufourc1/SemesterProjectMA3`.

Skutella, M. (2008). An introduction to network flows over time. In *Bonn Workshop of Combinatorial Optimization*.