Conflict-based search for real-time railway dispatching

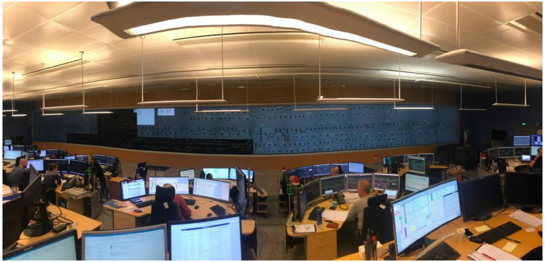
Bjørnar Luteberget, Paolo Ventura, and

Carlo Mannino

2024-09-12

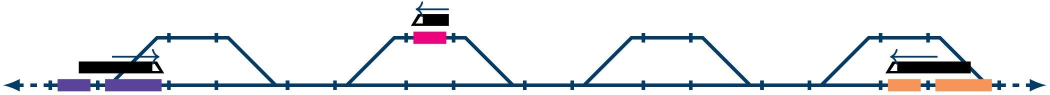
Timetabling and real-time dispatching

A timetable at Oslo central



Dispatchers at Oslo control centre

The train dispatching problem

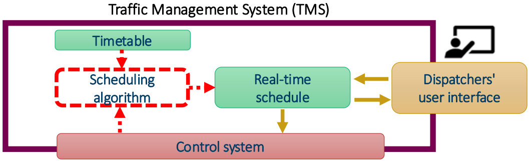


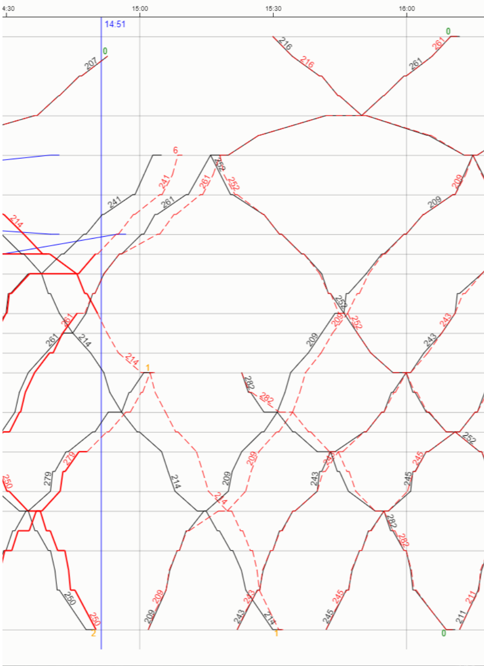
The **train dispatching problem**: given the current position of the trains, decide a **route** and a **schedule** for each train, s.t.:

- … trains do not use the same track segments at the same time, and
- … delays (compared to the timetable) are minimized.

Real-time application

- **Optimized Train Scheduling** can increase throughput and reduce delays.
- Yet, it is still the missing piece of many TMS systems
- That is because it requires state-of-the-art optimization methods, advanced software engineering skills, and a tremendous amount of experience





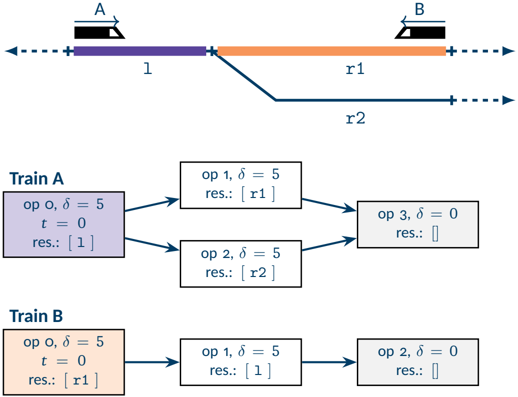
Real-time train dispatching is difficult!

It is very hard to solve train dispatching problems in practice:

1. The core is a **job-shop scheduling problem** (NP-hard) (Mascis & Pacciarelli 2002).
2. In practice, additional rules and constraints.
3. Very large instances (of practical interest)
4. Short computational time (< 2 minutes)

Formalizing the train dispatching problem

- A train is a directed acyclic graph of **operations**.
- A **route** for a train is a path through the graph from an initial to a final node.
- A **schedule** is an assignment of start times to each operation in the route.
- Each operation requires exclusive access to some **resources**.
- **Operations** have start time and duration bounds.



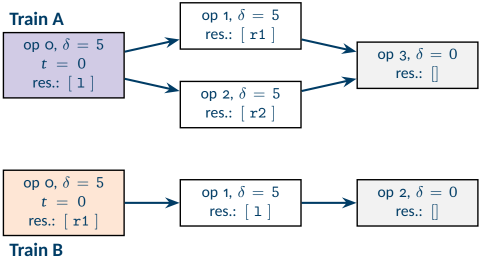
State of the art

Approaches to solving train dispatching problems:

- Heuristics (out of scope here)
- Alternative graph (AG) models (**“resource-oriented”**)
  - Create graph with all potential precedence constraints between conflicting operations and select a subset of precedence constraints.
  - MILP with many big-M constraints (custom branch-and-bound beneficial)
- Time-indexed models (**“time-oriented”**)
  - Decide on a set of discrete time ”slots” and allow only one train in each location in each time slot.
  - MILP with packing constraints (column generation typically beneficial)
- Spatial decomposition is also very important (but out of scope here)

Alternative graph models

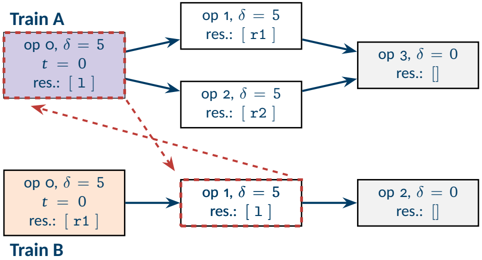
- Let $x^i_j \in \{0, 1\}$ be the decision of whether train i **performs** operation j.
- Flow conservation constraints select a path for each train.
- Let $t^i_j, t^{*i}_j \in \mathbb{R}$ be the **start and end time** of train i's operation j. (End time must be start time of the next operation.)



Alternative graph models

- Let $x_j^i \in \{0, 1\}$ be the decision of whether train i **performs** operation j.
- Flow conservation constraints select a path for each train.
- Let $t_j^i, t^{*i}_j \in \mathbb{R}$ be the **start and end time** of train i's operation j. (End time must be start time of the next operation.)
- For pair of conflicting operations x_j^i, x_l^k:

$$(x_j^i = 0) \vee (x_l^k = 0) \vee (t^{*i}_j \leq t_l^k) \vee (t^{*k}_l \leq t_j^i)$$



Alternative graph

$$(x_j^i = 0) \lor (x_l^k = 0) \lor ({t^*}_j^i \leq t_l^k) \lor ({t^*}_l^k \leq t_j^i)$$

- Introduce variables y for selecting the last two disjuncts.
- The disjunctive constraint can be linearized with big-M.
- Branch-and-bound over decisions x and y.
- Node relaxations optimize timing t as a longest path problem (easy).
- Called **"alternative graph"** because the relaxation is a project scheduling graph and the branch-and-bound selects between alternative choices of precedences.

Alternative graph

- Introduced without routing in 2002.

  (Mascis & Pacciarelli, 2002)

- Used in practice with extensions to routing and spatial decomposition.

  (D'Ariano et al., 2007), (Pellegrini et al., 2015), (Lamorgese & Mannino, 2015), (Leutwiler & Corman 2022)

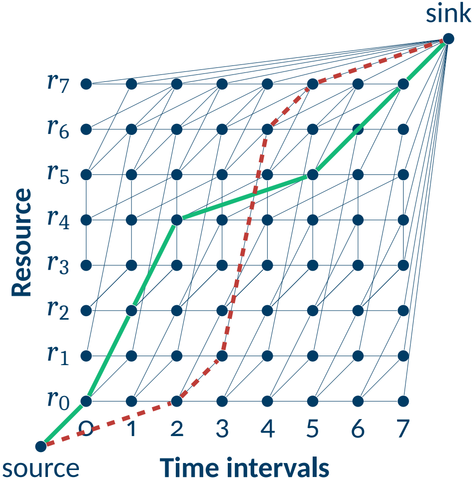
- Reasonably scalable **when routing is very limited** (or decomposable).

- With comprehensive routing choices: multiple independent decisions to make before the node relaxation's bound increases.

$$(x_j^i = 0) \lor (x_l^k = 0) \lor ({t^*}_j^i \leq t_l^k) \lor ({t^*}_l^k \leq t_j^i)$$

Time-oriented models

- Alternative: **time-oriented models**: decide on a set of relevant time points (discretization).
- As a MILP: packing constraints give better relaxation bounds.
- Granulatity trade-off bites: fine discretizations give a huge number of decisions



Time-oriented models

- Can be used for dispatching, especially in station areas.
  (Zwaneveld et al., 2001), (Harrod, 2011)
- Somewhat scalable in number of trains and resources (w/column generation).
  (Lusby et al., 2013), (Reynolds et al. 2020) .
- … but harder to scale time horizon and to longer lines.
- Dynamic discretization helps. (Croella et al. 2023)

Time-oriented branch-and-bound

What has been done in similar applications?

- **Robotics** path planning literature has many similarities!

  Already noted for shunting (Mulderij et al., 2020)

  (Hanou et al., 2024)
- The **multi-agent path finding** problem

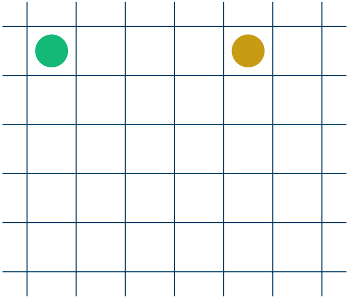
Time-oriented branch-and-bound

What has been done in similar applications?

- **Robotics** path planning literature has many similarities!

  Already noted for shunting (Mulderij et al., 2020)

  (Hanou et al., 2024)
- The **multi-agent path finding** problem:
  - agents placed on a grid (or graph)



Time-oriented branch-and-bound

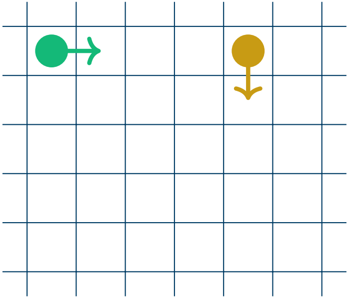
What has been done in similar applications?

- **Robotics** path planning literature has many similarities!

  Already noted for shunting (Mulderij et al., 2020)

  (Hanou et al., 2024)

- The **multi-agent path finding** problem:
  - agents placed on a grid (or graph)
  - traveling one cell per time unit



Time-oriented branch-and-bound

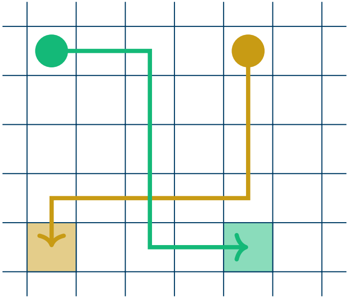
What has been done in similar applications?

- **Robotics** path planning literature has many similarities!

  Already noted for shunting (Mulderij et al., 2020)

  (Hanou et al., 2024)

- The **multi-agent path finding** problem:
  - agents placed on a grid (or graph)
  - traveling one cell per time unit
  - each agent ends up in a specified cell



Time-oriented branch-and-bound

What has been done in similar applications?

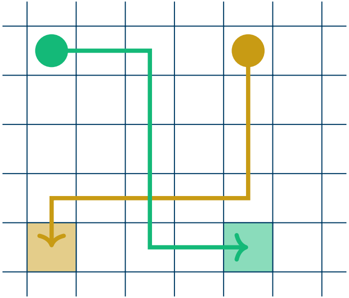
- **Robotics** path planning literature has many similarities!

  Already noted for shunting (Mulderij et al., 2020)

  (Hanou et al., 2024)

- The **multi-agent path finding** problem:
  - agents placed on a grid (or graph)
  - traveling one cell per time unit
  - each agent ends up in a specified cell

  … minimizing delays



Conflict based search

Much progress in multi-agent path finding, especially a **branch-and-bound algorithm** called **conflict-based search** (Sharon et al., 2015).

Idea:

- Plan each agent as a **shortest path problem** in a time-expanded graph.
- Combine the individual plans and look for conflicts (two agents in the same cell at the same time).
- For a conflict in cell c at time t, branch on the decision of whether
  - agent A is not in cell c at time t
  - agent B is not in cell c at time t

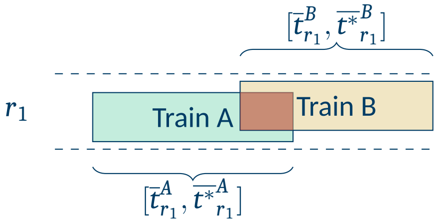
$$(z_t^{A,c} = 0) \vee (z_t^{B,c} = 0)$$

Adaptation to train dispatching

Adapting CBS to deal in continuous time similar to (Andreychuk et al., 2022) and (Walker et al., 2018):

- For a conflict in resource r_1
  - used by train A in the interval $[\bar{t}^A_{r_1}, \overline{t^*}^A_{r_1}]$
  - used by train B in the interval $[\bar{t}^B_{r_1}, \overline{t^*}^B_{r_1}]$

What **constraint** can we use to **eliminate** this solution?



From precedences to time windows

Starting from the familiar precedence constraint,

$$(x^A = 0) \vee (x^B = 0) \vee (t^{*B} \leq t^A) \vee (t^{*A} \leq t^B)$$

From precedences to time windows

Starting from the familiar precedence constraint,

$$\ldots \vee \left(t^{*B} \leq t^{A}\right) \vee \ldots$$

From precedences to time windows

Starting from the familiar precedence constraint,

$$\ldots \vee (t^{*B} \leq t^{A}) \vee \ldots$$

... choose any constant $\lambda \in \mathbb{R}$, then we have:

$$\ldots \vee (t^{*B} < \lambda) \vee (\lambda \leq t^{A}) \vee \ldots$$

From precedences to time windows

Starting from the familiar precedence constraint,

$$(x^A = 0) \vee (x^B = 0) \vee (t^{*B} \leq t^A) \vee (t^{*A} \leq t^B)$$

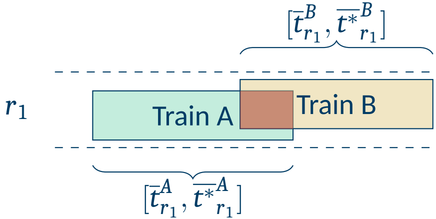
... choose constants $\lambda_1, \lambda_2 \in \mathbb{R}$, then we have:

$$(x^A = 0) \vee (x^B = 0) \vee (t^{*B} < \lambda_1) \vee (\lambda_1 \leq t^A) \vee (t^{*A} < \lambda_2) \vee (\lambda_2 \leq t^B)$$

Adaptation to train dispatching

The constants $\lambda_1 = \overline{t^*}^B_{r_1}$ and $\lambda_2 = \overline{t^*}^A_{r_1}$ makes most sense, since the end time is **likely to already be** as early as possible.

$$(x^A_{r_1} = 0) \vee (x^B_{r_1} = 0) \vee$$
$$(t^A_{r_1} \geq \overline{t^*}^B_{r_1}) \vee (t^{*B}_{r_1} < \overline{t^*}^B_{r_1}) \vee$$
$$(t^B_{r_1} \geq \overline{t^*}^A_{r_1}) \vee (t^{*A}_{r_1} < \overline{t^*}^A_{r_1})$$



Adaptation to train dispatching

How to deal with the 6-way **disjunction**?

Rearrange:

$$(x^A_{r_1} = 0) \vee (x^B_{r_1} = 0) \vee (t^A_{r_1} \geq \overline{t^*}^B_{r_1}) \vee ({t^*}^B_{r_1} < \overline{t^*}^B_{r_1}) \vee (t^B_{r_1} \geq \overline{t^*}^A_{r_1}) \vee ({t^*}^A_{r_1} < \overline{t^*}^A_{r_1})$$

Adaptation to train dispatching

How to deal with the 6-way **disjunction**?

Rearrange:

$$(x^A_{r_1} = 0) \vee (t^A_{r_1} \geq \overline{t^*}^B_{r_1}) \vee ({t^*}^A_{r_1} < \overline{t^*}^A_{r_1}) \vee (x^B_{r_1} = 0) \vee ({t^*}^B_{r_1} < \overline{t^*}^B_{r_1}) \vee (t^B_{r_1} \geq \overline{t^*}^A_{r_1})$$

Adaptation to train dispatching

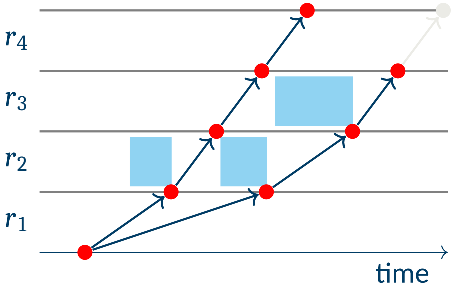
How to deal with the 6-way **disjunction**?
Rearrange:

$$\underbrace{(x^A_{r_1} = 0) \vee (t^A_{r_1} \geq \overline{t^*}^B_{r_1}) \vee (t^{*A}_{r_1} < \overline{t^*}^A_{r_1})}_{\text{Left branch: timing/path constraint on train A}} \vee \underbrace{(x^B_{r_1} = 0) \vee (t^{*B}_{r_1} < \overline{t^*}^B_{r_1}) \vee (t^B_{r_1} \geq \overline{t^*}^A_{r_1})}_{\text{Right branch: timing/path constraint on train B}}$$

Shortest path subproblem

In each branch-and-bound node, find **shortest path in time-expanded network**.

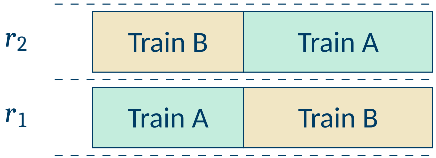
- The relevant times are only the operation lower bounds and the times mentioned in the constraints.
- We generalize slightly from **Safe Interval Path Planning**. (Phillips & Likhachev, 2011)
- Implementation is straight-forward with "**labelled Dijkstra's**".
- Strong domination when constraints are sparse. E.g., r_4 is unconstrained, so using the earliest start time suffices.



Beware of swapping!

If trains move immediately from one resource to the next, we can get **swapping**.

- Node relaxation has a pair of train **marginally overlapping** in adjacent resources r_1, r_2.
- No time-window constraint on either resource will suffice.



Solution: condition the constraint on **all four operations**.

$$(x^A_{r_1} = 0) \vee (x^A_{r_2} = 0) \vee (x^B_{r_1} = 0) \vee (x^B_{r_2} = 0) \vee$$

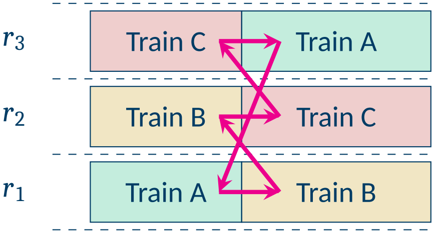
$$(t^A_{r_1} \geq \overline{t^*}^B_{r_2}) \vee (t^{*B}_{r_2} < \overline{t^*}^B_{r_2}) \vee (t^B_{r_1} \geq \overline{t^*}^A_{r_2}) \vee (t^{*A}_{r_2} < \overline{t^*}^A_{r_2})$$

Beware of swapping in cycles!

We can also see three or more trains swapping in a cycle.

- Node relaxation has a pair of train **marginally overlapping** in a **cycle** between resources r_1, r_2, r_3.
- No two-way disjunction on a pair of trains will suffice.

Solution: **three-way disjunction**.
(fortunately, this happens very rarely)



Conflict based search for train dispatching

We implemented:

- Shortest path search with **path/timing constraints**.
- Incremental **conflict detector** using interval trees.
- **Constraint generator** with cycle conflict detection.
- Branch-and-bound with strong branching and probing.

- Using the Rust programming langauge
- Running on AMD Ryzen 9 7900X 12-core desktop computer
- Gurobi v10.0 for MILP comparison

Experiments

We tested on problem instances provided by Siemens Mobility.
Freight-dominated, **5-12 trains**, **avg. ∼1400 operations** per instance.
MILP with big-M for comparison. Note that MILP has heuristics, while CBS is pure best-first.

Instance		Alternative graph MILP			Conflict-based search		
Trains	Ops.	Gap	Nodes	Time (s)	Gap	Nodes	Time (s)
6	443	0%	7594	12.1	0%	22	0.0
6	1754	82%	1	60.1	-	1501	60.0
6	1954	100%	1	60.0	0%	176	1.8
5	1854	100%	1533	60.0	0%	451	4.6
13	1437	-	1	60.0	0%	98	0.4
5	113	0%	0	0.0	0%	8	0.0
9	1718	83%	1406	60.0	0%	725	2.9
7	948	76%	1089	60.0	0%	131	0.4
8	1099	100%	4905	60.0	-	9814	60.0
8	2245	100%	7	60.0	0%	692	11.3
6	1416	93%	11018	60.0	0%	68	0.3
9	1975	90%	5679	60.0	0%	260	2.3

(comparison with time-indexed approach to come...)

Cross-fertilization:
train dispatching ⇄ multi-agent path finding

Many ideas from multi-agent path finding seem promising for train dispatching:

- Conflict prioritization (=**strong branching**)
- **Disjoint splitting** (Andreychuk et al., 2021)
- **Bypassing** / deep bypassing
- Better lower bound using **conflict graphs** (Li et al., 2019)
- **Heuristics**: priority inheritance, Monte Carlo methods, large neighborhood search (Okumura, 2023)

SINTEF

Technology for a better society