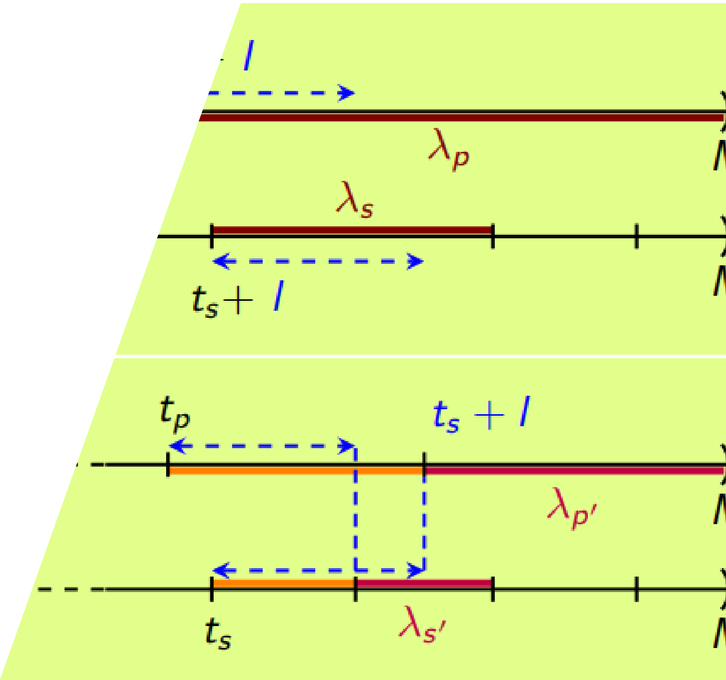
SINTEF



Dynamic time discretization for train scheduling

Bjørnar Luteberget

24 January 2024

SINTEF

Summary: DDD for train scheduling

- Schedule optimization often uses **time discretization**, but this has **severe drawbacks for train scheduling**.
- We have developed a **Dynamic discretization discovery (DDD)** that can overcome some of these drawbacks.
- We have tested the DDD on a simplified train dispatching problem.
- For competitive performance, the mathematical solver also needs to work dynamically. A **MaxSAT** algorithm outperforms MILP solvers on some objectives.

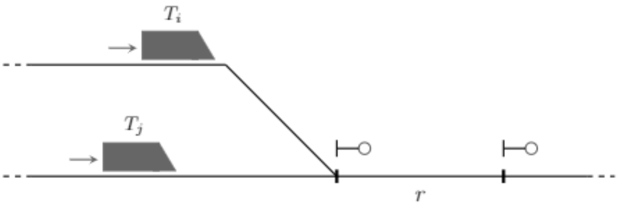
Joint work with *Anna Livia Croella*, *Carlo Mannino*, and *Paolo Ventura*.
Nominated for INFORMS RAS Student Paper Award 2022.
Journal paper currently under review.

Train Re-Scheduling

Unexpected (endogenous or exogenous) events can determine **delays and deviations** from the planned activity.

- **Timetables** may become **infeasible** and parts of the network may become unavailable for inbound trains
- The original schedule must be adjusted in real-time to mitigate the **impact** on the overall traffic.
- One aim to restore a feasible situation while **minimizing** some measure of the **deviation** of the actual schedule from the official timetable.

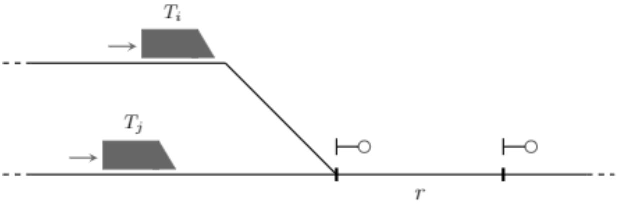
A simplified train re-scheduling problem



- Trains travel on fixed routes through tracks and stations.
- Trains spend a fixed amount of time to traverse a track.
- Station capacities and routing are ignored.

- Extension to **variable** travel times is straight-forward, **station capacity** is easy, general **routing** is possible.

A simplified train re-scheduling problem



- $t_i^{s2} - t_i^{s1} \geq l$
- $t_j^{s2} - t_j^{s1} \geq l$
- $t_j^{s2} - t_i^{s1} \geq 0 \bigvee t_j^{s1} - t_i^{s2} \geq 0$

MILP formulations

Two classes of MILP models are adopted in the literature:

- **big-M formulations** $\Rightarrow$ continuous time variables t_{ir}

$$t_{ir} - t_{jr} + M(1 - y_r^{ij}) \geq l_r^{ij}$$

$$t_{jr} - t_{ir} + My_r^{ij} \geq l_r^{ji}$$

MILP formulations

Two classes of MILP models are adopted in the literature:

- **big-M formulations** $\Rightarrow$ continuous time variables t_{ir}

$$t_{ir} - t_{jr} + M(1 - y_r^{ij}) \geq l_r^{ij}$$

$$t_{jr} - t_{ir} + My_r^{ij} \geq l_r^{ji}$$

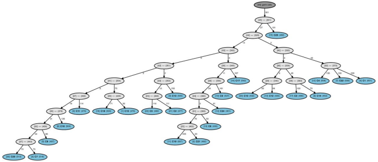
- **Time-Indexed formulations** $\Rightarrow$ discrete time variables x_{ir}^p

$$x_{ir}^p + x_{jr}^q \leq 1$$

Classical MILP formulations drawbacks

big-M formulations

- Poor bounds
- Large branching trees



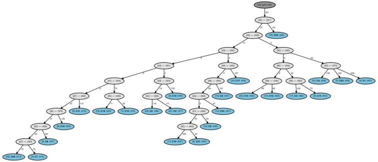
Classical MILP formulations drawbacks

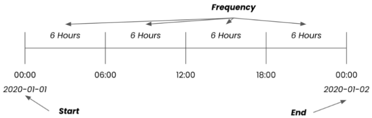
big-M formulations

- Poor bounds
- Large branching trees

TI formulations

- Oversize
- Bad approximation





We introduce a new TI based formulation:
the Interval Assignment Problem (IAP)

A novel paradigm

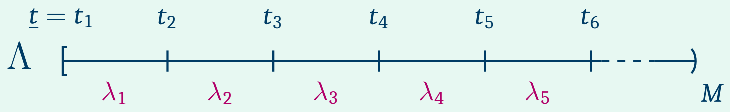
- E. He, N. Boland, G. Nemhauser, and M. Savelsbergh. **A dynamic discretization discovery algorithm for the minimum duration time-dependent shortest path problem**,2018
- Y. He, F. Lehuédé, and O. Péton. **A dynamic discretization approach to the integrated service network design and vehicle routing problem** In VeRoLog 2019 : seventh annual workshop of the EURO Working Group on Vehicle Routing and Logistics Optimization, Sevilla, Spain, June 2019.
- D. M. Vu, M. Hewitt, N. Boland, and M. Savelsbergh. **Dynamic discretization discovery for solving the time-dependent traveling salesman problem with time windows** Transportation Science, 2020
- Y. O. Scherr, M. Hewitt, B. A. N. Saavedra, and D. C. Mattfeld. **Dynamic discretization discovery for the service network design problem with mixed autonomous fleets.** Transportation Research Part B: Methodological, 2020
- L. Marshall, N. Boland, M. Savelsbergh, and M. Hewitt. **Interval-based dynamic discretization discovery for solving the continuous-time service network design problem** Transportation Science, 2021.

The DDD consists in solving a sequence of models with both a
fine discretization & **limited size**.

Classical TI formulation

For each train and each track segment

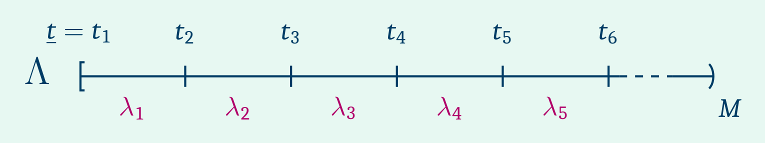
$\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ be a partition of the time horizon $[\underline{t}, M)$
such that $\lambda_p = [t_p, t_{p+1})$



Classical TI formulation

For each train and each track segment

$\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ be a partition of the time horizon $[\underline{t}, M)$
such that $\lambda_p = [t_p, t_{p+1})$

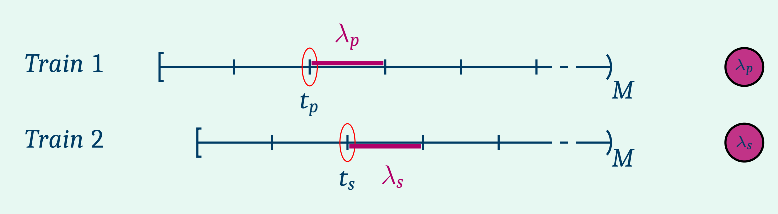


$$x_p = \begin{cases} 1 & \text{if train enters the track segment} \\ & \text{at the beginning } t_p \text{ of the interval } \lambda_p \\ 0 & \text{otherwise} \end{cases}$$

TI-Incompatibility

Given two trains traversing the same track segment, two intervals $\lambda_p = [t_p, t_{p+1})$ and $\lambda_s = [t_s, t_{s+1})$ are said TI-incompatible if

$$[t_p, t_p + l) \cap [t_s, t_s + l) \neq \emptyset$$

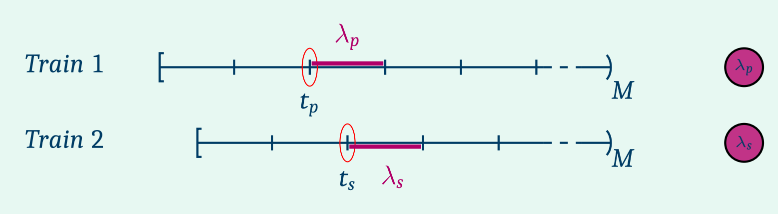


$$\boxed{t_s < t_p + l}$$

TI-Incompatibility

Given two trains traversing the same track segment, two intervals $\lambda_p = [t_p, t_{p+1})$ and $\lambda_s = [t_s, t_{s+1})$ are said TI-incompatible if

$$t_p < t_s + l \quad \text{and} \quad t_s < t_p + l$$

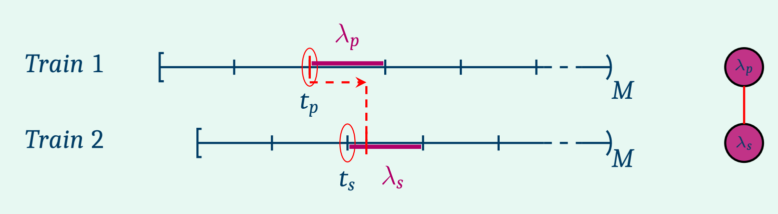


$$\boxed{t_s < t_p + l}$$

TI-Incompatibility

Given two trains traversing the same track segment, two intervals $\lambda_p = [t_p, t_{p+1})$ and $\lambda_s = [t_s, t_{s+1})$ are said TI-incompatible if

$$t_p < t_s + l \quad \text{and} \quad t_s < t_p + l$$



$$\boxed{x_p + x_s \leq 1 \;\; \forall \; \lambda_p \text{ TI-incompatible with } \lambda_s}$$

TI formulation

Given a set of partitions $\Lambda = \{\Lambda^{ir} : i \in I, r \in R_i\}$,
we want to find x^*, the incidence vector of a set of non-TI-incompatible intervals of minimum cost $\bar{c}(x^*)$.

$$
\begin{array}{lll}
\min & \sum_{i \in I} \sum_{r \in R_i} \sum_{\lambda_p \in \Lambda^{ir}} \bar{c}_p \cdot (x_p) & \\
s.t. & & \\
(1) & \sum_{\lambda_p \in \Lambda^{ir}} x_p = 1, & i \in I,\ r \in R_i \\
(2) & x_p + x_s \leq 1, & \lambda_p \text{ TI-incompatible with } \lambda_s \\
 & x_p \in \{0, 1\} & i \in I,\ r \in R_i,\ \lambda_p \in \Lambda^{ir}
\end{array}
$$

TI formulation

Given a set of partitions $\Lambda = \{\Lambda^{ir} : i \in I, r \in R_i\}$,
we want to find x^*, the incidence vector of a set of non-TI-incompatible intervals of minimum cost $\bar{c}(x^*)$.

$$
\begin{array}{lll}
\min & \sum_{i\in I} \sum_{r\in R_i} \sum_{\lambda_p\in\Lambda^{ir}} \bar{c}_p \cdot (x_p) & \\
s.t. & & \\
(1) & \sum_{\lambda_p\in\Lambda^{ir}} x_p = 1, & i \in I,\ r \in R_i \\
(2) & x_p + x_s \leq 1, & \lambda_p \text{ TI-incompatible with } \lambda_s \\
 & x_p \in \{0,1\} & i \in I,\ r \in R_i,\ \lambda_p \in \Lambda^{ir}
\end{array}
$$

We can obtain a schedule $t^* = \Phi(x^*)$,
if x^* is optimal then t^* is optimal for the TRP

IAP formulation

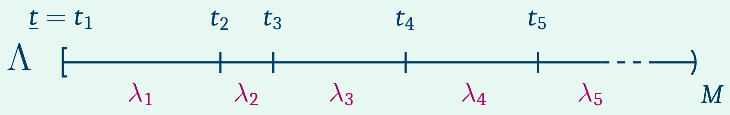
For each train and each track segment

$\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ be a partition of the time horizon $[\underline{t}, M)$ such that $\lambda_p = [t_p, t_{p+1})$

IAP formulation

For each train and each track segment

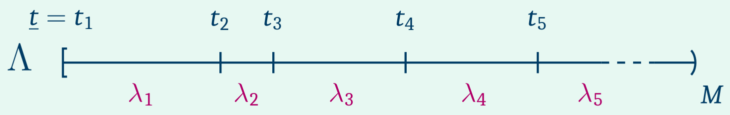
$\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ be a partition of the time horizon $[\underline{t}, M)$ such that $\lambda_p = [t_p, t_{p+1})$



IAP formulation

For each train and each track segment

$\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ be a partition of the time horizon $[\underline{t}, M)$ such that $\lambda_p = [t_p, t_{p+1})$

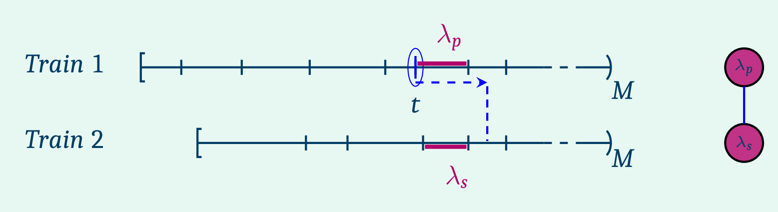


$$x_p = \begin{cases} 1 & \text{if train enters the track segment} \\ & \text{at some time in the interval } \lambda_p \\ 0 & \text{otherwise} \end{cases}$$

DDD-Incompatibility

Given two trains traversing the same track segment, two distinct intervals λ_p and λ_s are said DDD-incompatible if for any $t \in \lambda_p$ and any $t' \in \lambda_s$ we have:

$$t < t' + l \quad \text{and} \quad t' < t + l$$

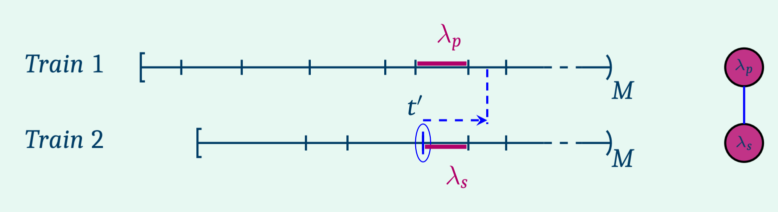


$$\boxed{x_p + x_s \leq 1 \;\; \forall \; \lambda_p \text{ DDD-incompatible with } \lambda_s}$$

DDD-Incompatibility

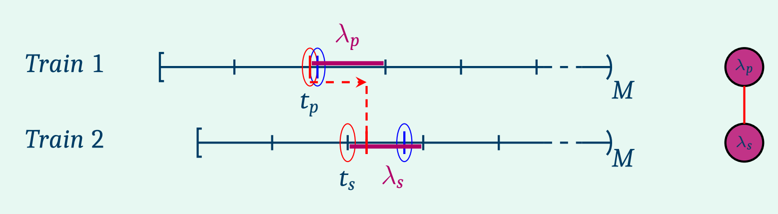
Given two trains traversing the same track segment, two distinct intervals λ_p and λ_s are said DDD-incompatible if for any $t \in \lambda_p$ and any $t' \in \lambda_s$ we have:

$$t < t' + l \quad \text{and} \quad t' < t + l$$



$$\boxed{x_p + x_s \leq 1 \;\; \forall\; \lambda_p \text{ DDD-incompatible with } \lambda_s}$$

TI-Incompatibility vs DDD-incompatibility



Note:

Two intervals λ_p and λ_s can be

TI-incompatible and **not DDD-incompatible**

Interval Assignment Problem (IAP)

Given a set of partitions $\Lambda = \{\Lambda^{ir} : i \in I, r \in R_i\}$,
we want to find x^*, the incidence vector of a set of non-DDD-incompatible intervals of minimum cost $\bar{c}(x^*)$.

$$
\begin{array}{lll}
\min & \sum_{i\in I} \sum_{r\in R_i} \sum_{\lambda_p\in\Lambda^{ir}} \bar{c}_p \cdot (x_p) & \\
s.t. & & \\
(1) & \sum_{\lambda_p\in\Lambda^{ir}} x_p = 1, & i \in I,\ r \in R_i \\
(2) & x_p + x_s \leq 1, & \lambda_p \text{ TI-incompatible with } \lambda_s \\
 & x_p \in \{0, 1\} & i \in I,\ r \in R_i,\ \lambda_p \in \Lambda^{ir}
\end{array}
$$

We can obtain a schedule $t^* = \Phi(x^*)$,
if x^* is optimal then t^* is optimal for the TRP

Interval Assignment Problem (IAP)

Given a set of partitions $\Lambda = \{\Lambda^{ir} : i \in I, r \in R_i\}$,
we want to find x^*, the incidence vector of a set of non-DDD-incompatible intervals of minimum cost $\bar{c}(x^*)$.

$$
\begin{array}{lll}
\min & \sum_{i \in I} \sum_{r \in R_i} \sum_{\lambda_p \in \Lambda^{ir}} \bar{c}_p \cdot (x_p) & \\
s.t. & & \\
(1) & \sum_{\lambda_p \in \Lambda^{ir}} x_p = 1, & i \in I,\ r \in R_i \\
(2) & x_p + x_s \leq 1, & \lambda_p \text{ DDD-incompatible with } \lambda_s \\
 & x_p \in \{0, 1\} & i \in I,\ r \in R_i,\ \lambda_p \in \Lambda^{ir}
\end{array}
$$

We can obtain a schedule $t^* = \Phi(x^*)$,
t^* is a lower bound for the TRP

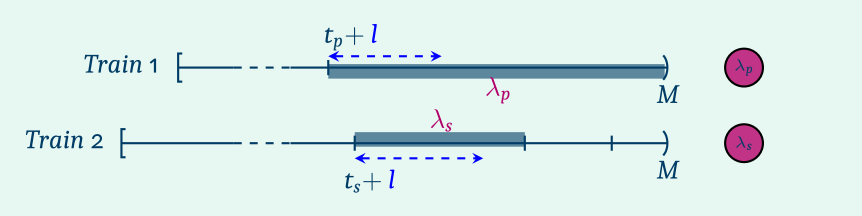
Initialize the IAP

The initial problem D_0 considers for each track segment traversed by each train, a single interval of the type:

$$\Lambda_0 = \{[\underline{t}, M),\ \text{for each } i \in I \text{ and each } r \in R_i\}$$

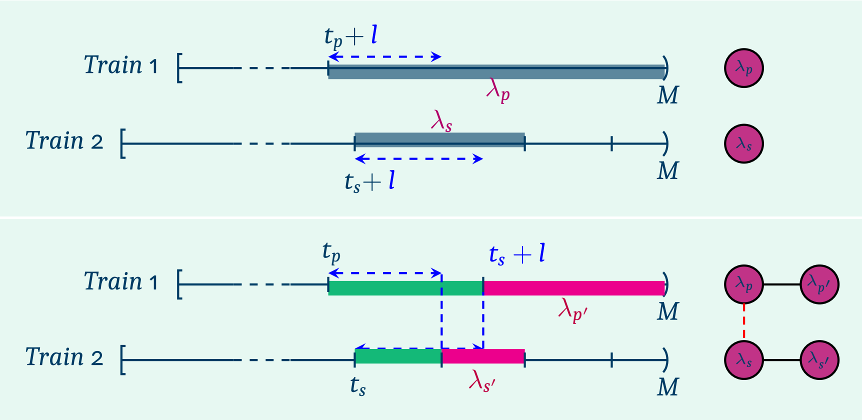
Refine the IAP problem

Example of refinement for two DDD-incompatible intervals



Refine the IAP problem

Example of refinement for two DDD-incompatible intervals



Computational experiments

- In our experience, running the **DDD algorithm** using a **MILP solver (Gurobi)** is not competitive with the big-M formulation.
- Row and column (variables and constraints) generation: MILP solvers typically **restart** completely.

Computational experiments

- In our experience, running the **DDD algorithm** using a **MILP solver (Gurobi)** is not competitive with the big-M formulation.
- Row and column (variables and constraints) generation: MILP solvers typically **restart** completely.
- Two ways forward:
  1. Use a more incremental solver: **core-based MaxSAT**
  2. Use a custom branch-and-bound algorithm

SAT solvers

- The Boolean satisfiability (SAT) problem asks whether there is an assignment to binary variables that satisfies a set of **clause** constraints:

$$x_1 + \ldots + x_k + (1 - x_{k+1}) + \ldots + (1 - x_n) \geq 1$$

- Very good open source solvers such as MiniSat and CaDiCaL.

SAT solvers

- The Boolean satisfiability (SAT) problem asks whether there is an assignment to binary variables that satisfies a set of **clause** constraints:

$$x_1 + \ldots + x_k + (1 - x_{k+1}) + \ldots + (1 - x_n) \geq 1$$

- Very good open source solvers such as MiniSat and CaDiCaL.
- Many applications in computer science, many based on **incremental** use – similar to **row and column generation**.
- Last 5 years has seen good progress also in MaxSAT, the optimization version of SAT – no “native” numbers, so typically small integer objectives.

SAT solvers

Train scheduling DDD translates nicely to clause constraints

$\Rightarrow$ can be solved as a **MaxSAT problem**.

(we solved it using the RC2 algorithm)

SINTEF

Instances

The test set consists of **24 real-life instances** derived from two single-track railway networks of the Norwegian railroad.

	Line A	Line B
Number of instances	12	12
Number of routes	33	25
Avg Train	20	11
Avg Track per Train	19	15

We created an **additional 48 test instance**s by letting some trains take longer to travel tracks or wait longer in stations.

Objective functions

We minimize the train **delays at their final destination** stations f considering a delay function of the type:

$$c(t^{if}) = \max(0, t^{if} - \underline{t}^{if})$$

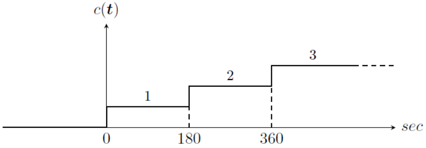
Objective functions

We minimize the train **delays at their final destination** stations f considering a delay function of the type:

$$c(t^{if}) = \max(0, t^{if} - \underline{t}^{if})$$

We tested stepwise functions with different number of steps:

1. **Linear rounded function:** $\Sigma_{t^{if} \in F} \lfloor c(t^{if})/Q \rfloor$
   where Q=180 is the time between stepwise increases
2. **3 steps function (0-3-6min):**



3. **1 step function (0-5min).**

Computational results

DDD-ALG and Big-M computation times (in ms) for different objective functions on the **10 hardest instance**s of our set.

Instance	Linear rounded		3 steps		1 step	
	Big-M	DDD-ALG	Big-M	DDD-ALG	Big-M	DDD-ALG
$\mathcal{I}_{11}^{AT}$	T/O	T/O	2541	453	689	221
$\mathcal{I}_{12}^{AT}$	T/O	T/O	2405	362	442	231
$\mathcal{I}_{12}^{AS}$	T/O	T/O	2080	380	565	172
$\mathcal{I}_{11}^{AS}$	T/O	T/O	917	335	566	198
$\mathcal{I}_{8}^{AS}$	115622	40404	1811	241	1188	126
$\mathcal{I}_{8}^{AT}$	37574	13416	875	247	254	191
$\mathcal{I}_{1}^{AS}$	1694	512	1161	178	393	144
$\mathcal{I}_{11}^{BO}$	1187	78	123	23	71	17
$\mathcal{I}_{2}^{AS}$	555	306	372	83	127	46
$\mathcal{I}_{8}^{AO}$	587	198	200	57	92	77

Computational results

It turned out that:

- when using a **linear rounded objective** function the DDD-ALG is typically between **2x and 10x faster** than Big-M (the rounding makes a large difference to the DDD-ALG)

Computational results

It turned out that:

- when using a **linear rounded objective** function the DDD-ALG is typically between **2x and 10x faster** than Big-M (the rounding makes a large difference to the DDD-ALG)
- with a **3 steps objective** function the computation **times are always much lower** (the DDD-ALG is faster than Big-M by 3x-10x on all instances)

Computational results

It turned out that:

- when using a **linear rounded objective** function the DDD-ALG is typically between **2x and 10x faster** than Big-M (the rounding makes a large difference to the DDD-ALG)
- with a **3 steps objective** function the computation **times are always much lower** (the DDD-ALG is faster than Big-M by 3x-10x on all instances)
- with a **1 step objective** function the DDD-ALG has computational times lower than 300 ms

Computational results

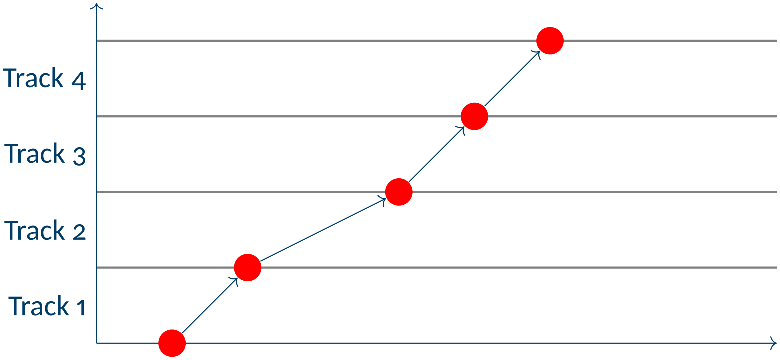
It turned out that:

- when using a **linear rounded objective** function the DDD-ALG is typically between **2x and 10x faster** than Big-M (the rounding makes a large difference to the DDD-ALG)
- with a **3 steps objective** function the computation **times are always much lower** (the DDD-ALG is faster than Big-M by 3x-10x on all instances)
- with a **1 step objective** function the DDD-ALG has computational times lower than 300 ms
- when using a **linear objective**, the DDD-ALG is **not** a **successful** approach (*general weakness with exact core-based MaxSAT solvers*)

Sketch of a branch-and-bound algorithm

Idea:

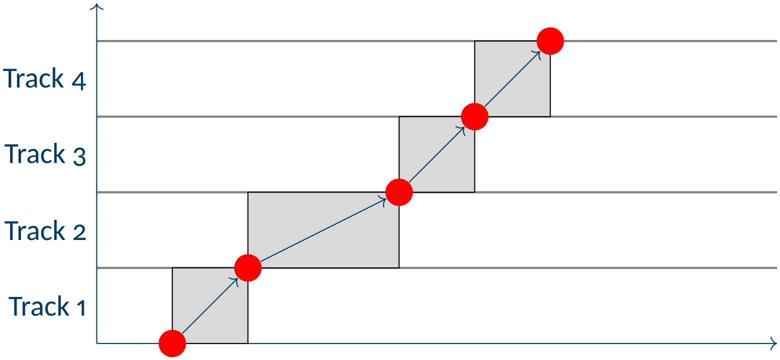
- Solve each train's schedule separately as a **shortest path problem** in a **time-expanded network**.



Sketch of a branch-and-bound algorithm

Idea:

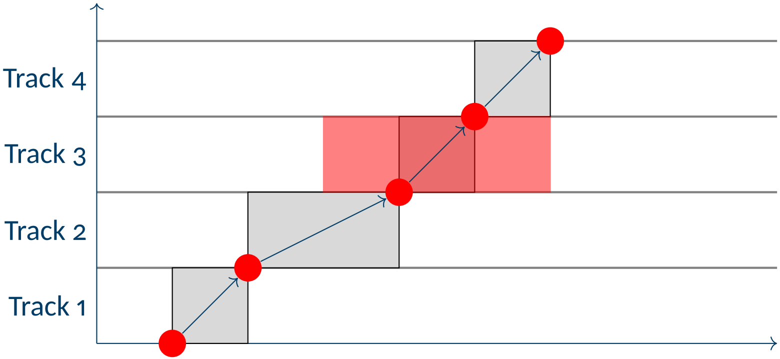
- The shortest path corresponds to a set of intervals where resources are occupied by the train.



Sketch of a branch-and-bound algorithm

Idea:

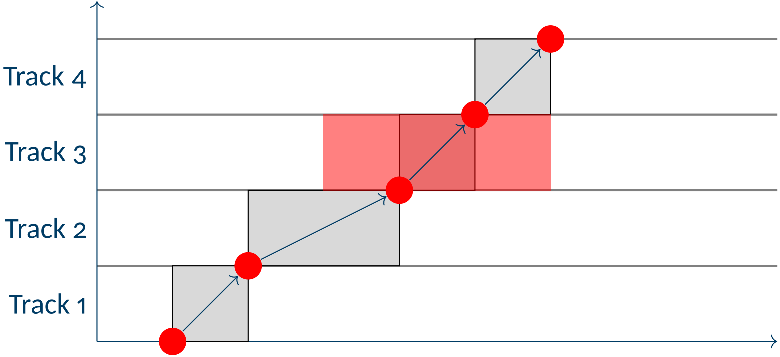
- When combining schedules into a complete solution, there will be **resource conflicts** between **pairs of trains**.

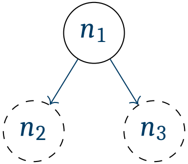


Sketch of a branch-and-bound algorithm

Idea:

- We can branch on **which** of the two trains is granted the resource in the **time interval**.

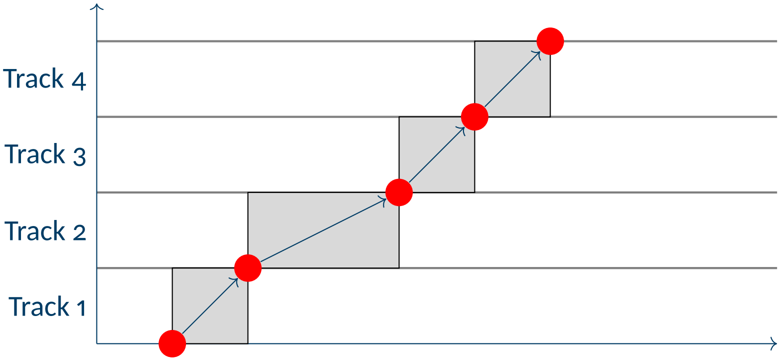


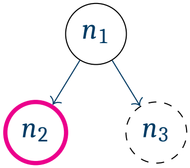


Sketch of a branch-and-bound algorithm

Idea:

- We can branch on **which** of the two trains is granted the resource in the **time interval**.

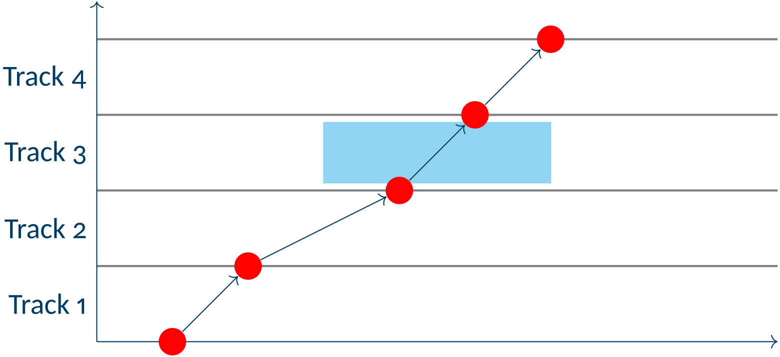


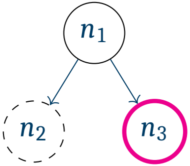


Sketch of a branch-and-bound algorithm

Idea:

- We can branch on **which** of the two trains is granted the resource in the **time interval**.

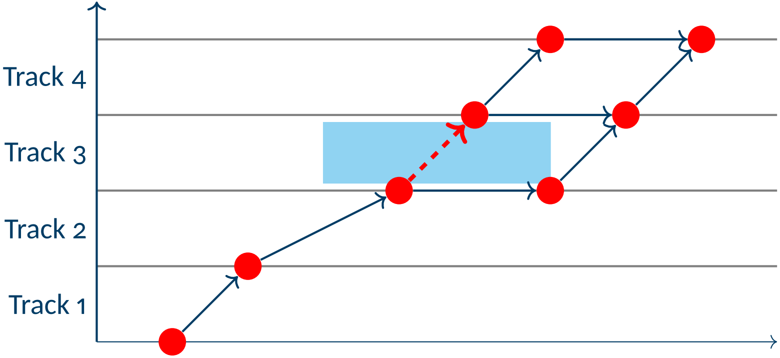


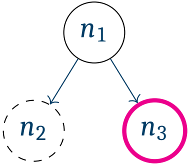


Sketch of a branch-and-bound algorithm

Idea:

- The shortest path problem is solved again for **one train** per new node.





Future work

- The custom branch and bound is a continuous-time (DDD) version of **Multi-agent path finding**.
- Routing is solved per-train as part of the shortest path problem (theoretically easy!)
- Branching space is still very large and easy to get stuck in deep branches.
- Pending computational experiments…

Technology for a better society