Branch-and-bound for real-time train rescheduling

Bjørnar Luteberget

SINTEF Digital, Mathematics and Cybernetics

2 November 2026, UiO IN3130



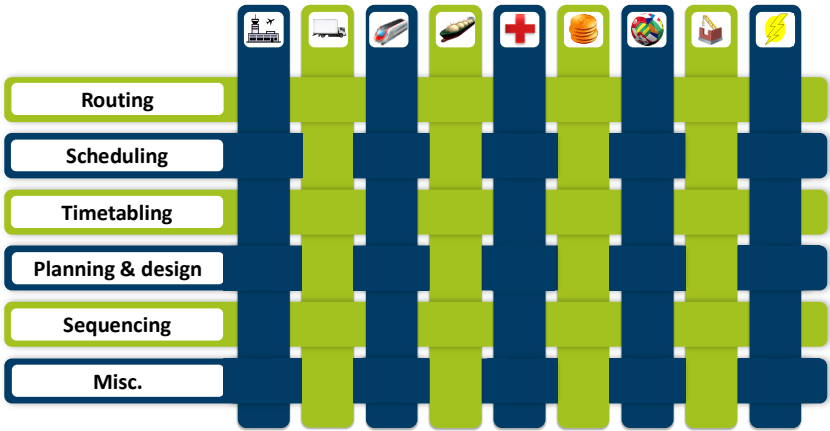
SINTEF



Et av de største uavhengige forskningsinstituttene i Europa

OMSETNING 4,2 mrd. NOK	ANSATTE 2200	PROSJEKTER 6400	KUNDER 3300
OMSETNING INTERNASJONALT 808 mill. NOK	NASJONALITETER 80	PUBLIKASJONER (INKL. FORMIDLING) 6200	KUNDETILFREDSHET 4,6 av 5

SINTEF Optimization – Application areas



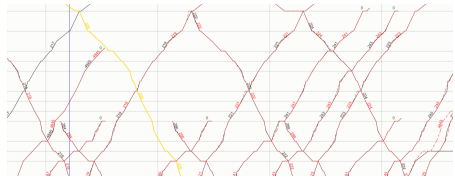
SINTEF

Today: two railway problems that need efficient algorithms

A. Getting trains back on time

Real-time rescheduling on a railway line.

- A longest-path algorithm for **easy** schedule optimization
- A branch-and-bound algorithm for **hard** deconflicting



B. Parking trains for the night

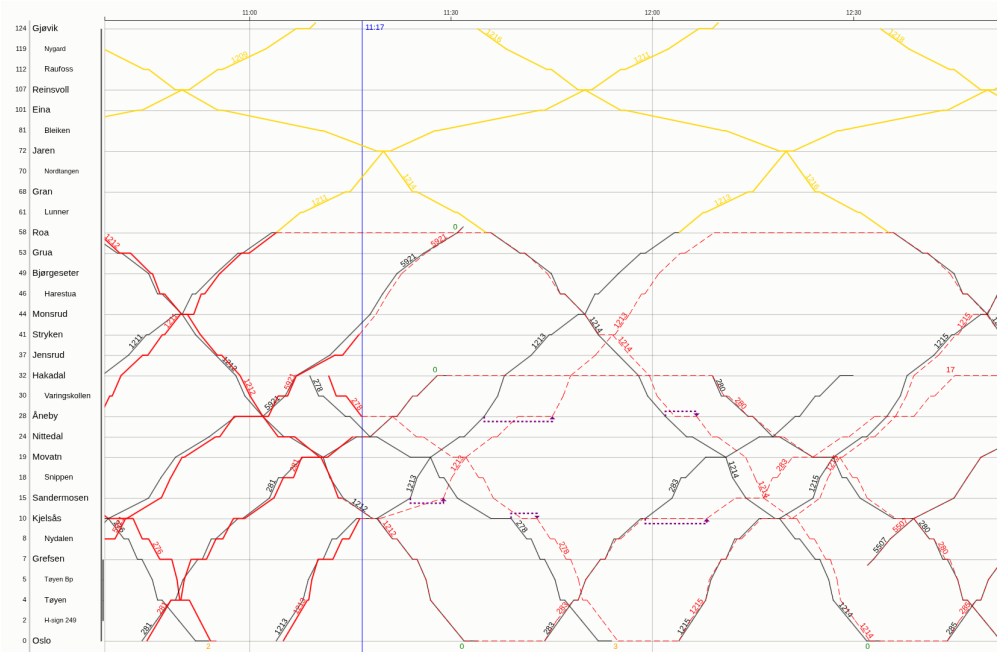
Parking passenger trains in a depot overnight.

- Translate the problem into a **SAT formula** and use an off-the-shelf solver.

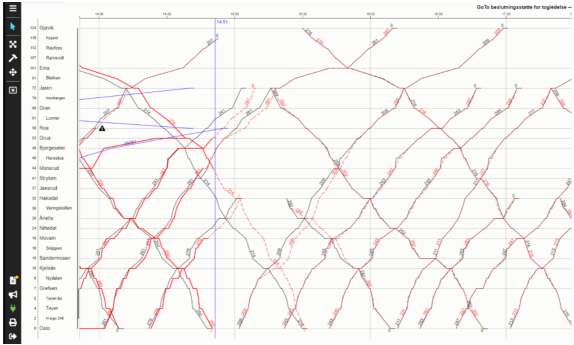


PART 1

The train dispatching problem


11:00
11:30
12:00
12:30
11.17
124 Gjøvik
119 Nygard
112 Raufoss
107 Reinsvoll
101 Eina
81 Bleiken
72 Jaren
70 Nordtangen
68 Gran
61 Lunner
58 Roa
53 Grua
49 Bjørgeseter
46 Harestua
44 Monsrud
41 Stryken
37 Jensrud
32 Hakadal
30 Varingskollen
28 Åneby
24 Nittedal
19 Movatn
18 Snippen
15 Sandermosen
10 Kjelsås
8 Nydalen
7 Grefsen
5 Tøyen Bp
4 Tøyen
2 H-sign 249
0 Oslo

Reading the train graph

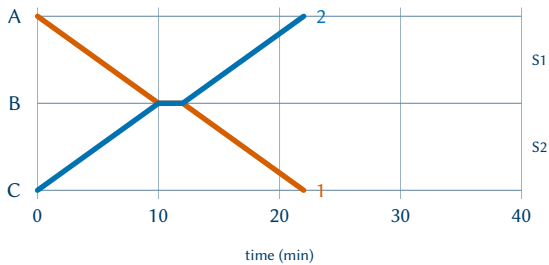


Dispatching support tool, Bane NOR / SINTEF.

- A **single-track** line: trains can only meet or overtake **at stations**.
- Two lines that **cross on a station row** are a meeting. Two lines that would **cross between stations** are a collision, so one train must wait.
- A **primary delay** (a late train) forces waits on other trains: **secondary delays**.
- The **dispatcher** decides where trains meet, right now, with trains moving. A decision support tool must answer in **seconds**.

A toy instance

Three stations, two single-track sections of 10 minutes each, station B has two tracks and a minimum dwell time of 2 minutes.



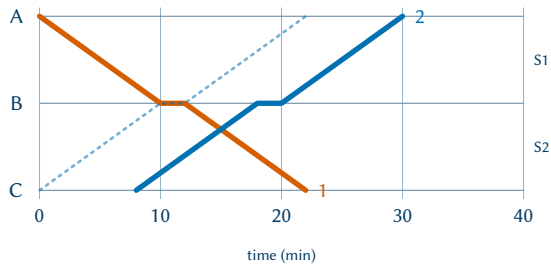
train 1 leaves A at 0, towards C.

train 2 leaves C at 0, towards A.

They meet at B, which has two tracks.

A toy instance

Three stations, two single-track sections of 10 minutes each, station B has two tracks and a minimum dwell time of 2 minutes.



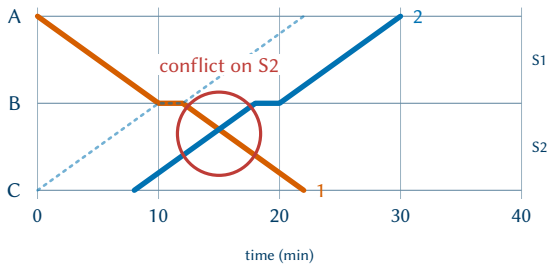
train 1 leaves A at 0, towards C.

train 2 leaves C at 0, towards A.

train 2 is **delayed** and ready at 8.

A toy instance

Three stations, two single-track sections of 10 minutes each, station B has two tracks and a minimum dwell time of 2 minutes.



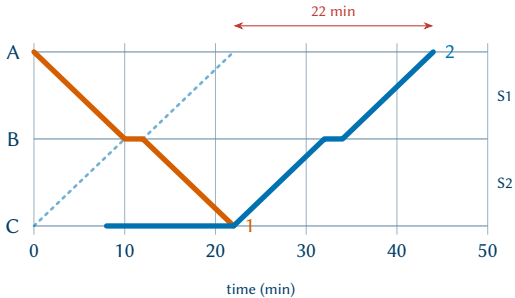
train 1 leaves A at 0, towards C.

train 2 leaves C at 0, towards A.

train 2 is **delayed** and ready at 8.

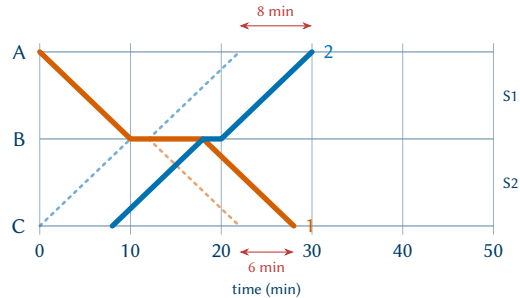
They both want S2 at the same time.
One of them has to wait.

Option 1: train 1 first on S2



train 2 waits at C and arrives 22 min late. Total delay 22.

Option 2: train 2 first on S2



train 1 waits at B and arrives 6 min late, **train 2** 8 min. Total delay 14.

Every conflict is a **choice of order**, and the choice changes which other conflicts appear later.

Real instances

Line	Trains	Stations	Conflicts
Gjøvikbanen	82	35	59–86
Jærbanen	160	28	19–31
Kongsvingerbanen	81	26	70–95

Per instance (a dispatching region, 10–22 h horizon).

- Up to about 100 conflicts per instance. Even with only two options each, that is 2^{100} combinations.
- The problem is **NP-hard**: it contains job-shop scheduling with blocking.
- Answer needed within **20 seconds**.



Dispatchers at the central signal box of Bohumín station, Czech Republic. Photo: Petr Štefek, CC BY 3.0, via Wikimedia Commons.

Partners and collaborators

Train dispatching research at SINTEF since 2012.

Project	Years	
Prioritization of trains	2012–2014	one line
CO2ReOpt	2016–2018	Norway–Sweden
OPSTRA	2017–2021	algorithms, also flights
GoTo	2019–2022	lines out of Oslo
MOTIONAL (EU)	2023–2026	up to timetabling
TRANS4M-R (EU)	2023–2026	freight and yards
REPLACE	2025–2028	rail replacement

More than **20 peer-reviewed articles**, on decomposition, reformulation, learning and discretization.

PART 2

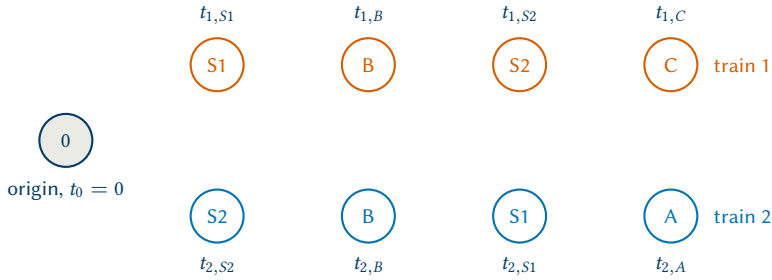
Schedules as a graph

What we want to find

- A **timetable**: a time for every departure and every arrival of every train.
- Every train needs at least its **running time** on each section and its **dwell time** at each station.
- Two trains are never on the same single-track section **at the same time**.
- Among all such timetables, we want the one with the **least delay**, summed over all trains at their last station.

Step 1: events

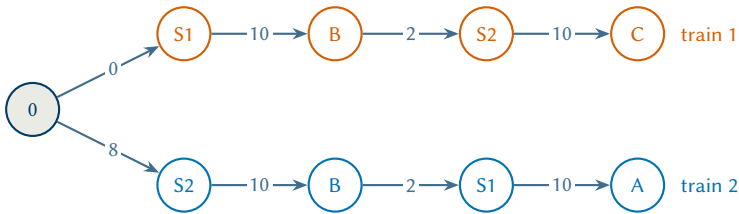
One event per departure and per arrival in the train graph, each with an unknown time.



- Each event is named after **where the train goes next**: $t_{1,S2}$ is when **train 1** departs B onto section S2, $t_{1,C}$ is when it arrives at C.
- A **schedule** is an assignment of times to all events. Each schedule can be drawn as a train graph.
- An extra **origin** event fixes the zero of time.

Step 2: fixed precedences

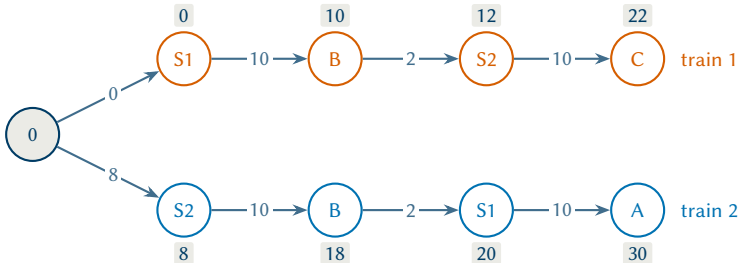
A constraint $t_v - t_u \geq \ell_{uv}$ is an arc $u \to v$ of length ℓ_{uv}.



- **Running time**: $t_{1,B} - t_{1,S1} \geq 10$. **Dwell**: $t_{1,S2} - t_{1,B} \geq 2$.
- **Earliest departure**: $t_{2,S2} - t_0 \geq 8$ (train 2 is ready at 8).

Step 2: fixed precedences

A constraint $t_v - t_u \geq \ell_{uv}$ is an arc $u \to v$ of length ℓ_{uv}.



- **Running time**: $t_{1,B} - t_{1,S1} \geq 10$. **Dwell**: $t_{1,S2} - t_{1,B} \geq 2$.
- **Earliest departure**: $t_{2,S2} - t_0 \geq 8$ (train 2 is ready at 8).
- Values in the boxes: the **earliest** time each event can happen given these arcs.

Separable optimization over difference constraints

$$\begin{aligned} \min\ & \textstyle\sum_v f_v(t_v) \\ \text{s.t. } & t_v - t_u \geq \ell_{uv} \quad \forall \text{ arcs } (u, v) \\ & t_0 = 0 \end{aligned}$$

- f_v **nondecreasing**: later is never better. Example: delay at the destination, $f_v(t) = \max(0, t - \text{planned}_v)$.
- The objective is **separable**: a sum of one term per event.

Theorem. Let L_v be the **longest path** from the origin to v. Then $t = L$ is an optimal schedule.

Proof. Take any feasible t and the longest path $0 = u_0 \to \cdots \to u_k = v$. Chain the arc constraints along it: $t_v \geq t_{u_{k-1}} + \ell \geq \cdots \geq \sum \ell = L_v$. So every feasible schedule is **pointwise** $\geq L$, and L is itself feasible. With f_v nondecreasing, $f_v(L_v) \leq f_v(t_v)$ for every v. □

Infeasible exactly when there is a **positive cycle**: around it, $0 \geq \sum \ell > 0$.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms*, 3rd ed., MIT Press, 2009, Section 24.4.
J. E. Kelley Jr. Critical-path planning and scheduling: mathematical basis. *Operations Research* 9(3), 1961.

Computing longest paths

No cycles (a DAG): dynamic programming over a topological order.

$L_0 \leftarrow 0$, $L_v \leftarrow -\infty$ for all other v
for v in topological order:
  for each arc (u, v):
    $L_v \leftarrow \max(L_v,\ L_u + \ell_{uv})$
return L

Time $O(n + m)$.

With cycles: Bellman–Ford, with $\max$ instead of $\min$.

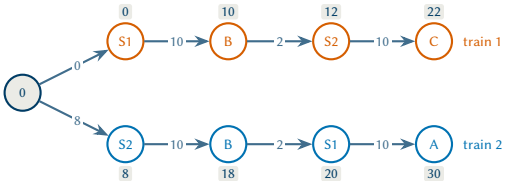
$L_0 \leftarrow 0$, $L_v \leftarrow -\infty$ for all other v
repeat $n - 1$ times:
  for each arc (u, v):
    $L_v \leftarrow \max(L_v,\ L_u + \ell_{uv})$
for each arc (u, v):
  if $L_u + \ell_{uv} > L_v$: **fail**, **positive cycle**
return L

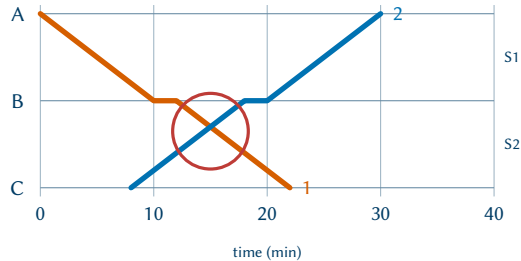
Time $O(nm)$.

A longest path without repeated events has at most $n - 1$ arcs, so $n - 1$ rounds are enough. If an arc can still improve L after that, the graph has a positive cycle and no schedule exists.

What the relaxed schedule looks like

Longest paths with only the fixed arcs: every train runs as if alone.





- Cost 8: **train 2** arrives at 30 instead of 22. But **train 1** is on S2 during $[12, 22)$ and **train 2** during $[8, 18)$.
- The fixed arcs are a **relaxation**: they ignore that S2 holds one train at a time.
- The cost of a relaxation is **never higher** than the true optimum, so it is a lower bound.

PART 3

Deconflicting

No two trains on a section at the same time

Train i occupies a single-track section from when it enters, s_i, until it leaves, e_i. For two trains i and j on the same section, we need

- *either* i leaves before j enters: $\quad e_i \leq s_j$,
- *or* j leaves before i enters: $\quad e_j \leq s_i$.

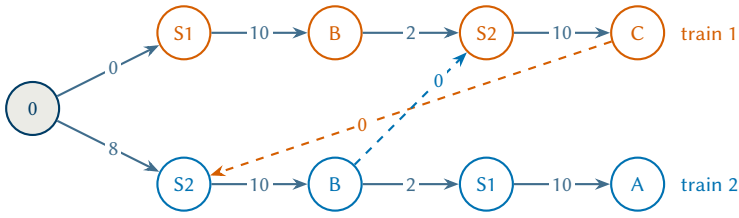
As difference constraints, the same two options read

$$s_j - e_i \geq 0 \qquad \vee \qquad s_i - e_j \geq 0,$$

that is, either the arc $e_i \to s_j$ or the arc $e_j \to s_i$, each with length 0.

A conflict is a disjunction

Two trains on one single-track section: one of them leaves before the other enters.

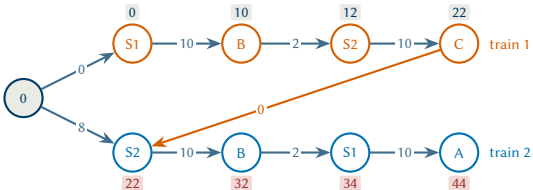


- Each term of the disjunction is an ordinary arc. We call them **disjunctive arcs** (Balas 1969, job-shop scheduling). We write C_1 for the event $t_{1,C}$.
- **Pick one arc from every disjunction** and the graph is plain again: longest paths give the schedule.

E. Balas. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operations Research* 17(6), 1969.
A. Mascis, D. Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research* 143(3), 2002.

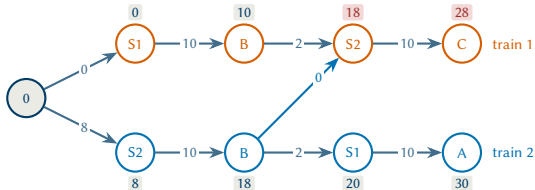
Choosing arcs gives a schedule

Arc $C_1 \to S2_2$ (train 1 first)



$t_{2,A} = 44$, planned 22: cost **22**.

Arc $B_2 \to S2_1$ (train 2 first)

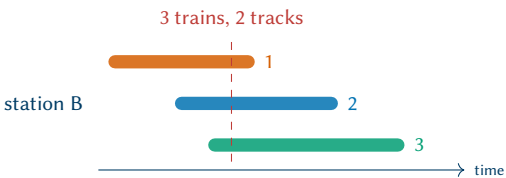


$t_{1,C} = 28$, $t_{2,A} = 30$, planned 22: cost $6 + 8 = \mathbf{14}$.

The train dispatching problem: choose one arc per disjunction so that the graph has **no positive cycle** and the longest-path cost is **minimal**.

Stations with c tracks

Where do the disjunctions come from? A checker reads the relaxed schedule and looks for overfull resources.

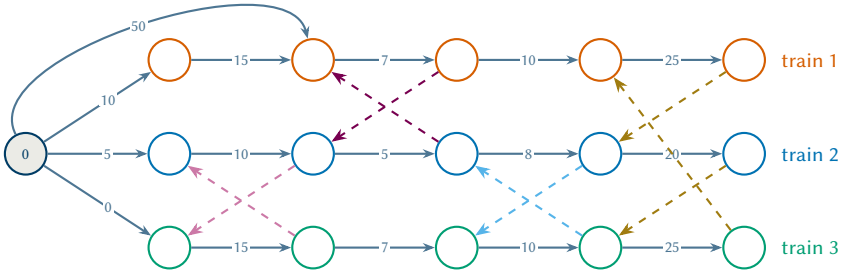


Sections and stations are **resources**: a section holds 1 train, a station c. Train i **occupies** B from arriving until departing, the interval $[t_{i,B},\ t_{i,B+1})$. With $c = 2$, three overlapping intervals are too many: some pair must be separated, in one order or the other.

A set Q of $c + 1$ trains overlapping at one time gives one disjunction with $(c + 1)$ c terms:

$$\bigvee_{i \neq j \in Q} t_{i,B} - t_{j,B+1} \geq 0.$$

Finding a violation means finding a $(c + 1)$-**clique** in the **interval graph** of the occupation intervals.

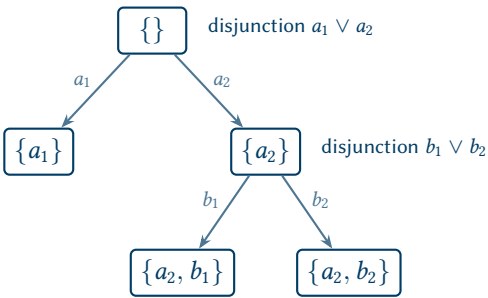

50
10
5
0
0
15
7
10
25
train 1
10
5
8
20
train 2
15
7
10
25
train 3

PART 4

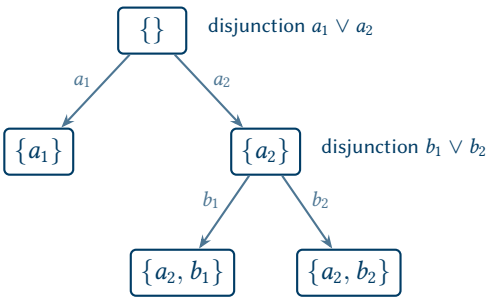
Branch-and-bound

The search tree

- A **node** is a set of chosen disjunctive arcs.

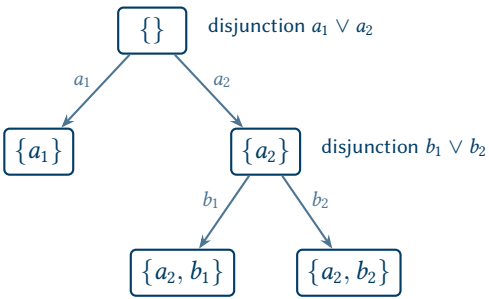


The search tree



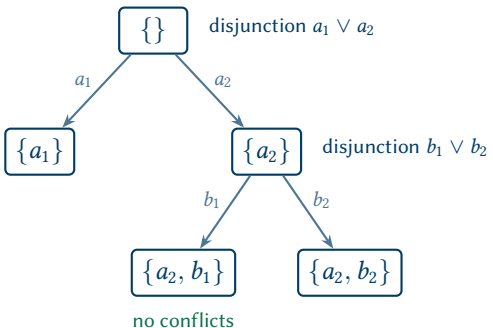
- A **node** is a set of chosen disjunctive arcs.
- The root node has no chosen arcs.

The search tree



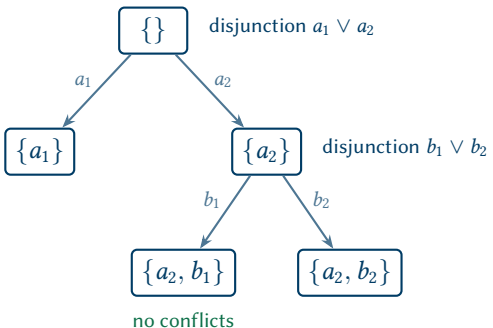
- A **node** is a set of chosen disjunctive arcs.
- The root node has no chosen arcs.
- Its **children**: take a disjunction and make one child per arc in it. Each branch of the tree **adds one disjunctive arc**.

The search tree



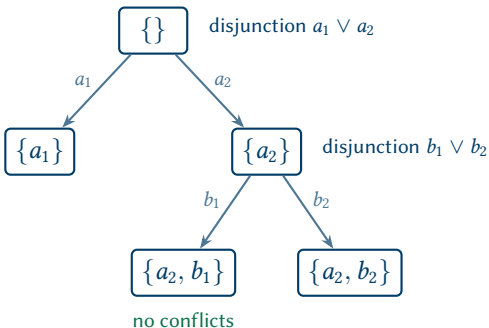
- A **node** is a set of chosen disjunctive arcs.
- The root node has no chosen arcs.
- Its **children**: take a disjunction and make one child per arc in it. Each branch of the tree **adds one disjunctive arc**.
- Its **bound**: the longest-path cost with its arcs. Adding arcs only makes times later, so bounds grow down the tree.

The search tree



- A **node** is a set of chosen disjunctive arcs.
- The root node has no chosen arcs.
- Its **children**: take a disjunction and make one child per arc in it. Each branch of the tree **adds one disjunctive arc**.
- Its **bound**: the longest-path cost with its arcs. Adding arcs only makes times later, so bounds grow down the tree.
- A node with **no conflicts** is a schedule. The best one found so far is the **incumbent**.

The search tree



- A **node** is a set of chosen disjunctive arcs.
- The root node has no chosen arcs.
- Its **children**: take a disjunction and make one child per arc in it. Each branch of the tree **adds one disjunctive arc**.
- Its **bound**: the longest-path cost with its arcs. Adding arcs only makes times later, so bounds grow down the tree.
- A node with **no conflicts** is a schedule. The best one found so far is the **incumbent**.
- **Prune** a node whose bound is not better than the incumbent, or whose arcs make a positive cycle.

The algorithm

Each node is stored with its longest-path schedule t. Its bound is $\text{cost}(t)$.

incumbent $\leftarrow \infty$
queue $\leftarrow \{(\{\},\ t_0)\}$ (t_0: longest paths, no disjunctive arcs)
while queue not empty:
  pop (N, t) with the **lowest** $\text{cost}(t)$ (best-first)
  if $\text{cost}(t) \geq$ incumbent: **prune**, continue
  $\omega \leftarrow$ a violated disjunction in t
  if none: **incumbent** $\leftarrow$ **cost**(t) (a feasible schedule)
  else for each arc $a \in \omega$:
    $t' \leftarrow$ longest paths with $N \cup \{a\}$ (**prune** if positive cycle)
    push $(N \cup \{a\},\ t')$

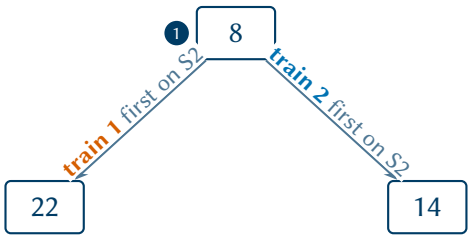
A. D'Ariano, D. Pacciarelli, M. Pranzo. A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research* 183(2), 2007.

Running the algorithm



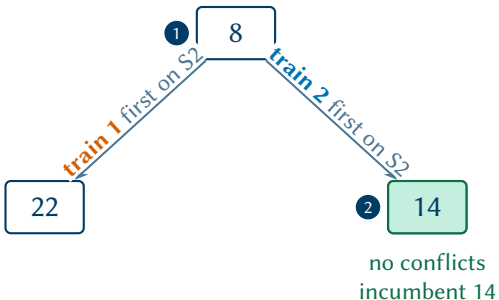
1. **Root.** No disjunctive arcs. Longest paths: cost 8. Checker: **train 1** and **train 2** overlap on S2. Branch.

Running the algorithm



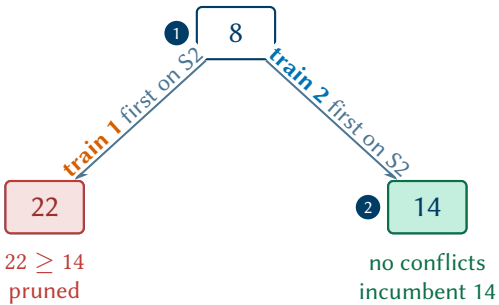
1. **Root.** No disjunctive arcs. Longest paths: cost 8. Checker: **train 1** and **train 2** overlap on S2. Branch.
2. **Children.** One disjunctive arc each. Bounds 22 and 14, computed as on the previous slides.

Running the algorithm



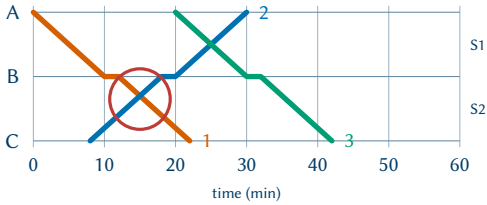
1. **Root.** No disjunctive arcs. Longest paths: cost 8. Checker: **train 1** and **train 2** overlap on S2. Branch.
2. **Children.** One disjunctive arc each. Bounds 22 and 14, computed as on the previous slides.
3. **Pop 14.** Checker finds no conflict. Feasible: incumbent $= 14$.

Running the algorithm

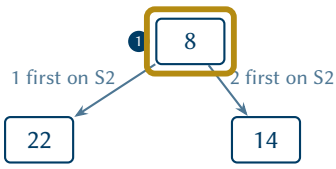


1. **Root.** No disjunctive arcs. Longest paths: cost 8. Checker: **train 1** and **train 2** overlap on S2. Branch.
2. **Children.** One disjunctive arc each. Bounds 22 and 14, computed as on the previous slides.
3. **Pop 14.** Checker finds no conflict. Feasible: incumbent $= 14$.
4. **Pop 22.** $22 \geq 14$: pruned without expanding it. **Done, optimal.**

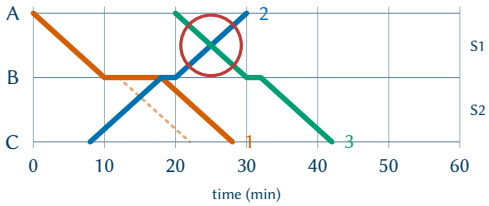
Another example

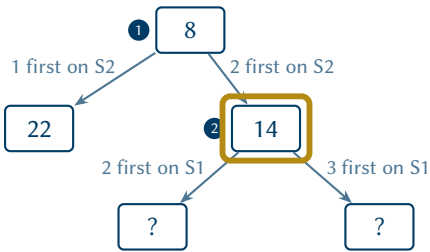


1. **Root**: same first conflict, children 22 and 14.



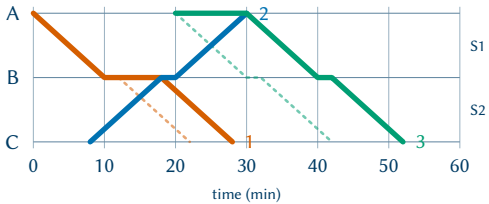
Another example



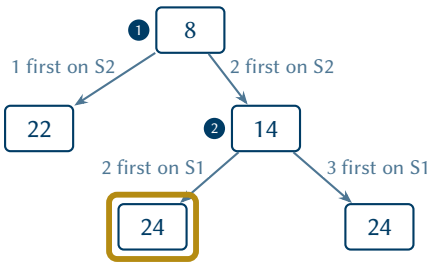


1. **Root**: same first conflict, children 22 and 14.
2. **Pop 14**. **train 2** now holds S1 during $[20, 30)$, and **train 3** wants it at 20. Two children.

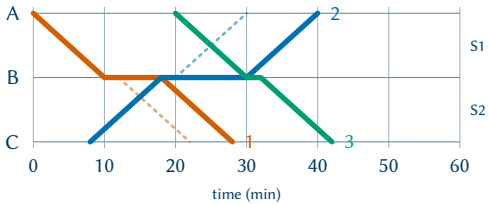
Another example



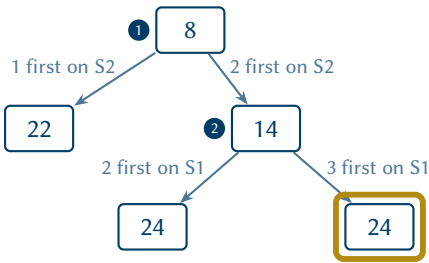
1. **Root**: same first conflict, children 22 and 14.
2. **Pop 14**. **train 2** now holds S1 during $[20, 30)$, and **train 3** wants it at 20. Two children.
3. 24 (**train 3** waits at A until 30) and 24 (**train 2** waits at B until 30).



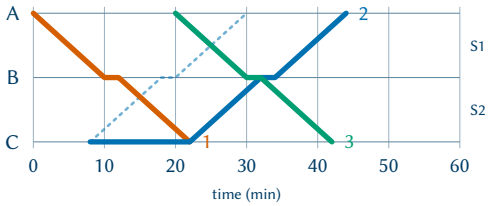
Another example

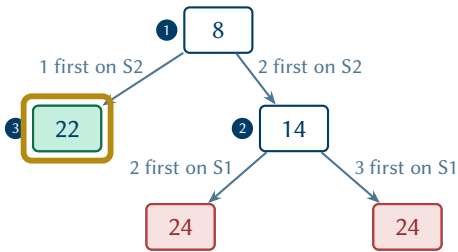


1. **Root**: same first conflict, children 22 and 14.
2. **Pop 14**. **train 2** now holds S1 during $[20, 30)$, and **train 3** wants it at 20. Two children.
3. 24 (**train 3** waits at A until 30) and 24 (**train 2** waits at B until 30).



Another example





1. **Root**: same first conflict, children 22 and 14.
2. **Pop 14**. **train 2** now holds S1 during $[20, 30)$, and **train 3** wants it at 20. Two children.
3. 24 (**train 3** waits at A until 30) and 24 (**train 2** waits at B until 30).
4. **Pop 22**. **train 3** runs ahead of **train 2** on S1 and behind it on S2. No conflict: **incumbent 22**. Both nodes with bound 24 are pruned.

Which disjunction to branch on?

Chronological. Branch on the earliest conflict in time. Cheap and often sensible.

Strong branching. For every violated disjunction ω and every arc $a \in \omega$, *try it*: add a, read the new bound, remove a. Let $\mu_\omega^{\max}$, $\mu_\omega^{\min}$ be the largest and smallest bound increase over ω, and score

$$s_\omega = \tfrac{1}{6}\,\mu_\omega^{\max} + \tfrac{5}{6}\,\mu_\omega^{\min}.$$

Branch on the disjunction with the highest score (Achterberg, Koch, Martin 2005).

Why that score? A disjunction where *every* choice hurts a lot is a decision that **must** be made and that **splits the problem well**. The $\mu^{\min}$ term dominates, because we want the bound to rise in every child, not only in one.

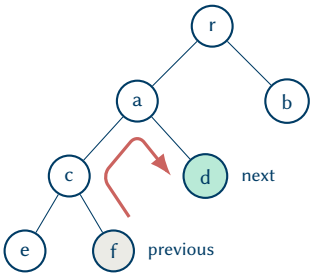
The catch: with 50 violated disjunctions and 2 to 6 arcs each, that is hundreds of bound evaluations *per tree node*. Only affordable if evaluating a child is **very cheap**.

T. Achterberg, T. Koch, A. Martin. Branching rules revisited. *Operations Research Letters* 33(1), 2005.

PART 5

Making it fast

Moving between tree nodes



- A child is its parent **plus one disjunctive arc**.
- To move to the next tree node: walk up to the common ancestor **popping** one disjunctive arc per step, then down **pushing** one per step.
- A strong-branching trial is a push, a bound read, and a pop.

The whole search is a sequence of **push arc** and **pop arc** operations on one graph, so these two operations set the speed of the search.

Incremental longest paths

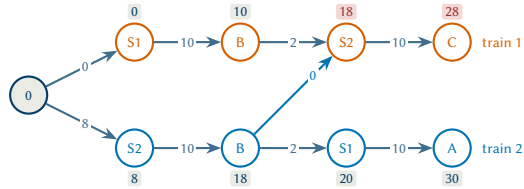
Update only the events whose time changes.

push$(u \to v, \ell)$:

if $L_u + \ell \le L_v$: nothing changes, return
queue $\leftarrow \{(u, v)\}$
while queue not empty:
  take (p, q) with the largest increase $L_p + \ell_{pq} - L_q$
  if $L_p + \ell_{pq} > L_q$:
    if $q = u$: **positive cycle**, undo and fail
    log(q, L_q); $L_q \leftarrow L_p + \ell_{pq}$
    push all arcs out of q onto the queue

pop: walk the log backwards and restore each L_q.

Largest increase first is Dijkstra on the increases, so each event changes at most once.



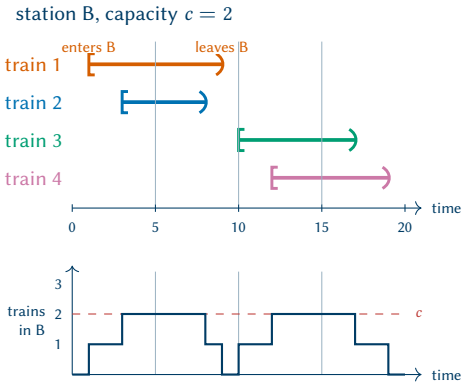
Pushing the arc $B_2 \to S2_1$ changes two of eight values. The undo log is $[(S2_1, 12), (C_1, 22)]$.

Cost grows with the **number of events whose time changes**, not with the graph (thousands of events).

G. Ramalingam, J. Song, L. Joskowicz, R. E. Miller. Solving systems of difference constraints incrementally. *Algorithmica* 23, 1999.

Incremental conflict detection

The checker should also not start from scratch at every tree node.

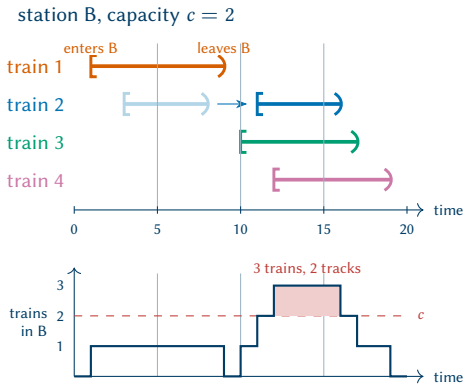


Before: at most 2 trains at a time.

- Per resource: all entry and exit times in one **sorted sequence**, with the **occupation count** on each piece between them.

Incremental conflict detection

The checker should also not start from scratch at every tree node.

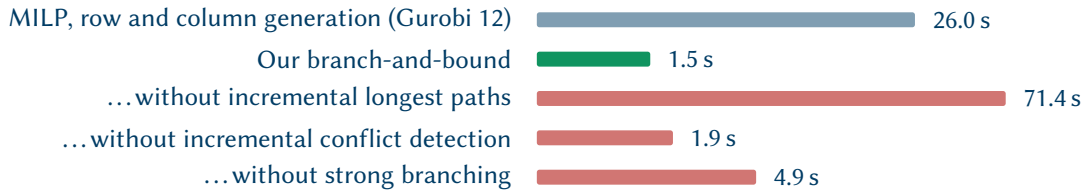


A push delays **train 2**: both its endpoints move right.

- Per resource: all entry and exit times in one **sorted sequence**, with the **occupation count** on each piece between them.
- A moved endpoint travels to its new place by **adjacent swaps**.
- Each swap changes the count on **one piece only**, between the two swapped endpoints. Above c: conflict.

Does it pay off?

Total solve time over 12 real instances from the three lines, single thread. Log scale.



- 18.5× faster than the MILP state of the art.
- Incremental longest paths: about 50× in total.
- Incremental conflict detection: a modest 28 %.
- Strong branching: 3× in total, but **slower** on one of the three lines (Kongsvinger), where conflicts are local and the chronological choice is already good.
- Which heuristic works best depends on the instances, so **measure** it.

L. Lamorgese, C. Mannino. An exact decomposition approach for the real-time train dispatching problem. *Operations Research* 63(1), 2015.

Takeaways

1. **Model it as a graph.** A schedule is a time for each event, and every rule says "at least ℓ minutes after". Then **longest paths** give the best schedule, and we know how to compute them fast.
2. **Relax and repair.** Drop the hard choices and solve the rest. The answer does two jobs: its cost is a **bound** for pruning, and its schedule shows **which conflict** to fix next.
3. **Incremental updates.** Neighboring tree nodes differ by one arc, so little changes between them. Incremental algorithms save most of the work.

The problem is NP-hard, but the real instances have structure, and an algorithm that exploits it answers in a fraction of a second.

SINTEF



Technology for a better society