Implication learning for train dispatching

Bjørnar Luteberget, Giorgio Sartor,
Carlo Mannino

2025-09-02

FP5 TRANS4M-R

Transforming Europe's Rail Freight

Implication learning for train dispatching

2025-09-02

Bjørnar Luteberget

The project FP5-TRANS4M-R is supported by the Europe's Rail Joint Undertaking and its members.

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the Europe's Rail Joint Undertaking. Neither the European Union nor the granting authority can be held responsible for them.

Co-funded by the European Union

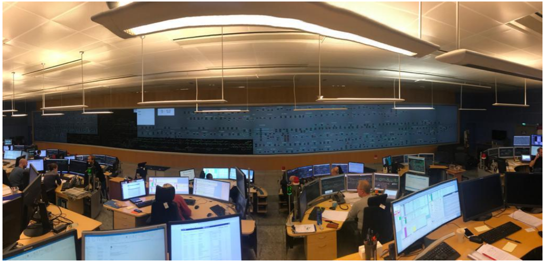
Co-funded by the European Union

The project FP5-TRANS4M-R is supported by the Europe's Rail Joint Undertaking and its members.

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the Europe's Rail Joint Undertaking. Neither the European Union nor the granting authority can be held responsible for them.

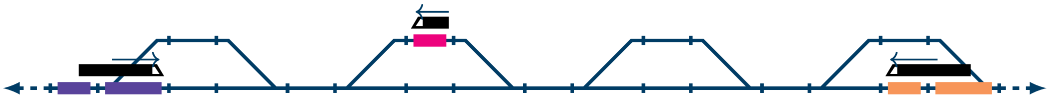
Timetabling and real-time dispatching

A timetable at Oslo central



Dispatchers at Oslo control centre

The train dispatching problem

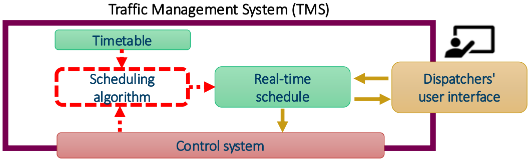


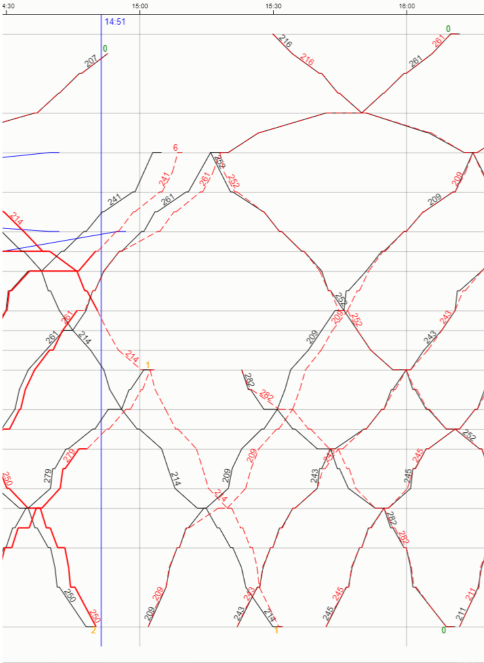
The **train dispatching problem**: given the current position of the trains, decide a **route** and a **schedule** for each train, s.t.:

- … 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





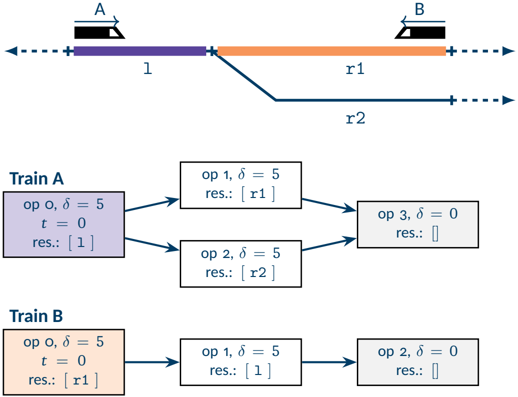
Real-time train dispatching is difficult!

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.



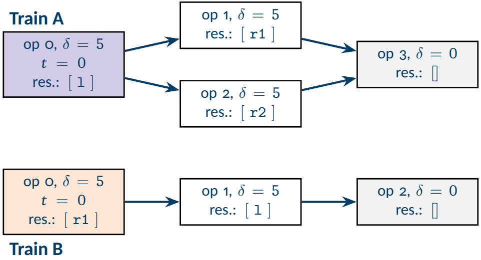
State of the art

Approaches to solving train dispatching problems:

- Alternative graph (AG) models (**“precedence disjunctions”**)
  - 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 (**“discretized time-space packing”**)
  - 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)
- Heuristics
- Spatial and temporal decomposition is also very important (but out of scope here)

Alternative graph models

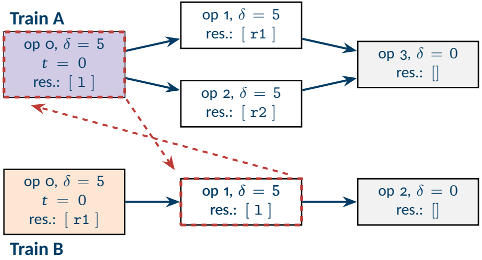
- Let $x_j^i \in \{0, 1\}$ be the decision of whether train i **performs** operation j.
- x must be a path in the graph.
- Let $t_j^i, t^{*i}_j \in \mathbb{R}$ be the **start and end time** of train i's operation j. (End time must be start time of the next operation.)



Alternative graph models

- Let $x^i_j \in \{0, 1\}$ be the decision of whether train i **performs** operation j.
- x must be a path in the graph.
- 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.)
- For pair of conflicting operations x^i_j, x^k_l:

$$(x^i_j = 0) \lor (x^k_l = 0) \lor (t^{*i}_j \le t^k_l) \lor (t^{*k}_l \le t^i_j)$$



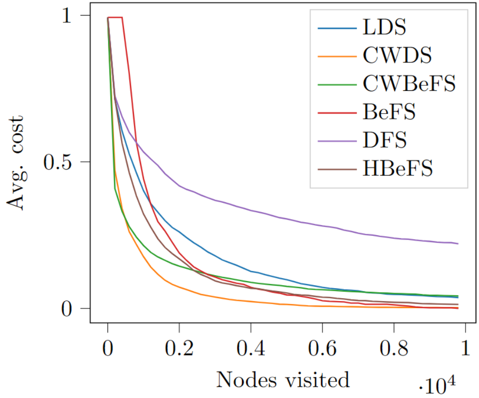
SINTEF

Alternative graph

$$(x^i_j = 0) \lor (x^k_l = 0) \lor (t^{*i}_{\ j} \leq t^k_l) \lor (t^{*k}_{\ l} \leq t^i_j)$$

- 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.

Diving heuristics



- Exact optimality can be found using best-bound-first tree search.
- Diving heuristics seem very effective when routing is “easy”.

SINTEF

Alternative graph

- Introduced without routing in 2002.
  (Mascis & Pacciarelli, 2002)
- Underlies many of the best real-time dispatching algorithms, 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.

$$(x^i_j = 0) \vee (x^k_l = 0) \vee (t^{*i}_j \leq t^k_l) \vee (t^{*k}_l \leq t^i_j)$$

Exact methods for routing+scheduling

- **Without routing**: branch-and-bound (and MILP) is effective.

  (D'Ariano et al. 2006)

- With **"little"** routing, e.g. simple stations: Logic Benders decomposition turns routing into scheduling disjunctions.

  (Lamorgese et al. 2015, Leutwiler et al. 2023)

- **"Wide"** instances, e.g. short infrastructure and/or time horizon: time-indexed formulations, heuristically limited routing, fully enumerated routings.

  (Reynolds et al. 2020, Pellegrini et al. 2014, Samà 2015, Croella et al. 2025)

- **Much routing**, both wide and long: routing local search + exact scheduling, etc.

  (Corman et al. 2010, Samà 2015)

Deadlock detection

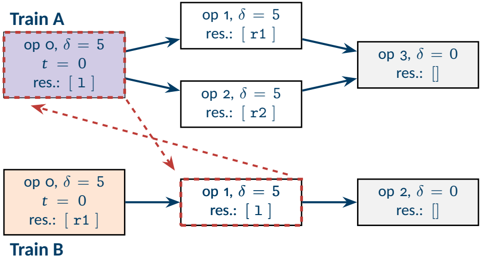
- In passenger traffic: routes and schedules have been planned in advance, so it is often easy to find a feasible solution.
- In American freight traffic: not necessarily any detailed timetable.
- It can be hard to determine whether there is **any solution at all!**
- Specialized deadlock detection models disregard **scheduling objectives** and **(physical) time**.
- Literature: transition system models (Dal Sasso et al., 2021) alternative graph models (Dal Sasso et al. 2025)

SINTEF

Europe's Rail

FP5 TRANS4M-R Transforming Europe's Rail Freight

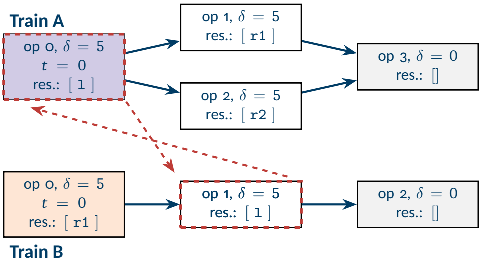
Cycles formulation for deadlocks (Dal Sasso et al. 2023)



Cycles formulation for deadlocks (Dal Sasso et al. 2023)

- Routing variables x and flow conservation

$$\sum_{i \in \delta^+(a)} x_i - \sum_{i \in \delta^-(a)} x_i = b_a$$



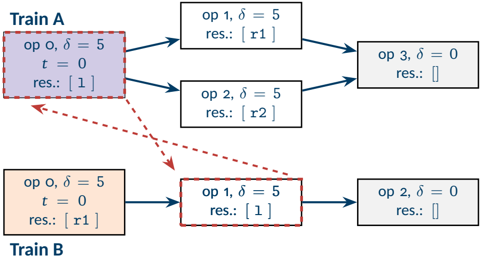
Cycles formulation for deadlocks (Dal Sasso et al. 2023)

- Routing variables x and flow conservation

$$\sum_{i \in \delta^+(a)} x_i - \sum_{i \in \delta^-(a)} x_i = b_a$$

- Precedence variables y for operations with overlapping resources:

$$y_{ab} + y_{ba} = 1$$



Cycles formulation for deadlocks (Dal Sasso et al. 2023)

- Routing variables x and flow conservation

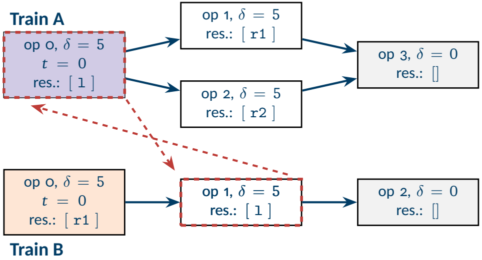
$$\sum_{i \in \delta^+(a)} x_i - \sum_{i \in \delta^-(a)} x_i = b_a$$

- Precedence variables y for operations with overlapping resources:

$$y_{ab} + y_{ba} = 1$$

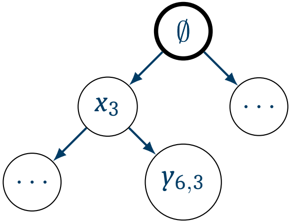
- Cycle elimination constraints

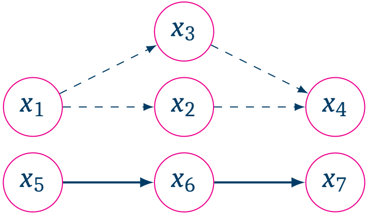
$$\sum_{i \in R(C)} x_i + \sum_{(a,b) \in P(C)} y_{ab} \leq |R(C)| + |P(C)| - 1$$



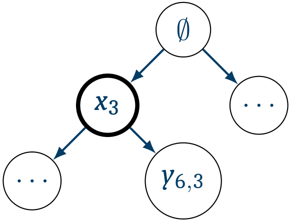
Generating cycle constraints

- The cycles formulation is exponential – efficient row generation is essential.
- How do we actually detect the cycles? Incremental cycle detection algorithm

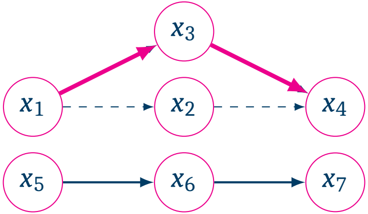




Generating cycle constraints

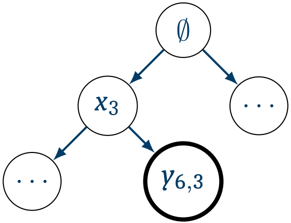


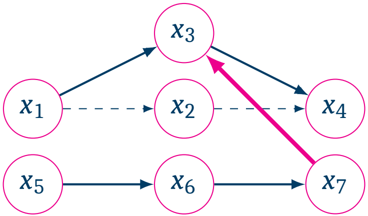
- The cycles formulation is exponential – efficient row generation is essential.
- How do we actually detect the cycles? Incremental cycle detection algorithm



Generating cycle constraints

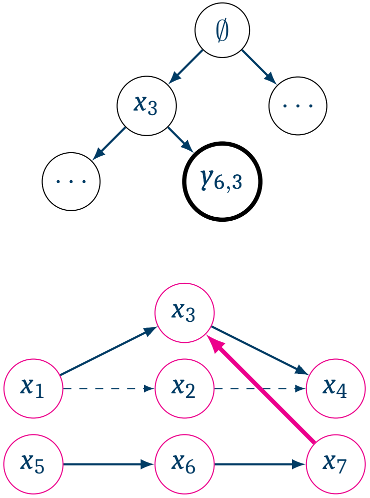
- The cycles formulation is exponential – efficient row generation is essential.
- How do we actually detect the cycles? Incremental cycle detection algorithm





Generating cycle constraints

- The cycles formulation is exponential – efficient row generation is essential.
- How do we actually detect the cycles? Incremental cycle detection algorithm
- With incremental longest path instead, This is actually scheduling.



Cycle elim./disjunctive constraints/clauses/implications

- Disjunctive binary assignment constraints are also called clauses

$$x_1 \vee x_2 \vee \neg x_3$$

$$(x_1 = 1) \vee (x_2 = 1) \vee (x_3 = 0)$$

- They are also linear constraints (over binary variables)

$$x_1 + x_2 + (1 - x_3) \geq 1$$

- We get domain propagation (implied assignment) when all but one alternative is false

$$((x_1 = 0) \wedge (x_3 = 1)) \Rightarrow (x_2 = 1)$$

SINTEF

FP5 TRANS4M-R Transforming Europe's Rail Freight

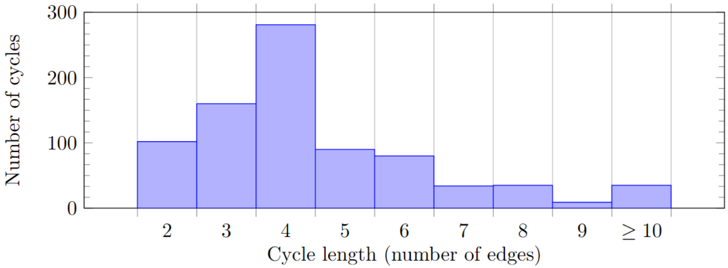
Europe's Rail

SAT deadlock detection

Instance				MIP ticks (Dal Sasso et al., 2021)		MIP cycles (Dal Sasso et al., 2023)		SAT transition system (Luteberget, 2021)		SMT cycles (unpublished)	
#	Result	n_r	n_t	Steps	Time (s)	Iter.	Time (s)	Steps	Time (s)	Iter.	Time (s)
T11	Dead	62	2	27	17.60	N/A		3	0.00	650	0.01
T12	Dead	62	4	39	>60.00	N/A		10	0.10	2217	0.07
T13	Dead	62	4	39	>60.00	N/A		10	0.01	1262	0.04
T14	Live	62	4	39	3.27	N/A		6	0.00	103	0.04
T15	Dead	46	4	42	>60.00	N/A		6	0.01	48	0.00
T16	Live	62	5	50	5.33	N/A		5	0.01	50	0.00
T17	Live	62	4	50	43.11	N/A		6	0.01	170	0.00
T18	Dead	62	4	50	>60.00	N/A		6	0.02	234	0.00
T19	Dead	62	5	51	>60.00	N/A		6	0.01	132	0.00
T20	Dead	70	5	57	>60.00	N/A		8	0.05	7	0.00
C11	Dead	274	5	114	>60.00	3	0.69	7	0.09	652	0.02
C12	Live	35	5	21	0.08	0	0.02	4	0.00	3	0.00
C13	Dead	257	6	68	0.48	0	0.13	8	0.77	1	0.00
C14	Live	317	6	77	1.79	3	1.19	4	0.01	26	0.01
C15	Dead	102	6	32	0.20	0	0.05	7	0.02	18	0.00
C16	Live	142	6	34	0.54	0	0.16	4	0.00	14	0.00
C17	Live	137	6	70	1.86	3	0.24	6	0.01	71	0.00
C18	Live	325	7	54	3.46	0	6.35	5	0.08	38	0.06
C19	Live	219	9	108	4.50	29	10.90	7	0.02	117	0.01
C20	Live	837	9	194	14.73	1	3.93	4	0.07	378	0.09

Implication learning from scheduling cycles

- In the pure-scheduling instances, cycles are only very short, most of them 2 edges (which can be avoided using look-ahead).
- In deadlock detection instances, more varied cycle lengths.



Resolution

$$\frac{A \lor x \qquad B \lor \neg x}{A \lor B}$$

Implication learning in SAT solvers

- SAT solvers quickly assign variables but ensures that each assignment is **consistent with unit propagation**.
- Then, analyze which decisions caused the conflict by performing **resolution**.
- This makes sure that you can **never find the same** partial assignment (unit propagation will happen before).

(Marques-Silva et al., 2002)

Implication learning in SAT solvers

- SAT solvers quickly assign variables but ensures that each assignment is **consistent with unit propagation**.
- Then, analyze which decisions caused the conflict by performing **resolution**.
- This makes sure that you can **never find the same** partial assignment (unit propagation will happen before).

(Marques-Silva et al., 2002)

Start with clauses:

- $c_1 : x_1 \vee x_2 \vee x_3$
- $c_2 : x_1 \vee \neg x_2$
- $c_3 : x_1 \vee \neg x_3$

Implication learning in SAT solvers

- SAT solvers quickly assign variables but ensures that each assignment is **consistent with unit propagation**.
- Then, analyze which decisions caused the conflict by performing **resolution**.
- This makes sure that you can **never find the same** partial assignment (unit propagation will happen before).

(Marques-Silva et al., 2002)

Start with clauses:

- $c_1 : x_1 \vee x_2 \vee x_3$
- $c_2 : x_1 \vee \neg x_2$
- $c_3 : x_1 \vee \neg x_3$

Assign $\neg x_1$.

Implication learning in SAT solvers

- SAT solvers quickly assign variables but ensures that each assignment is **consistent with unit propagation**.
- Then, analyze which decisions caused the conflict by performing **resolution**.
- This makes sure that you can **never find the same** partial assignment (unit propagation will happen before).

(Marques-Silva et al., 2002)

Start with clauses:

- $c_1 : x_1 \vee x_2 \vee x_3$
- $c_2 : x_1 \vee \neg x_2$
- $c_3 : x_1 \vee \neg x_3$

Assign $\neg x_1$. **Propagate** $\neg x_2, \neg x_3$.

Implication learning in SAT solvers

- SAT solvers quickly assign variables but ensures that each assignment is **consistent with unit propagation**.
- Then, analyze which decisions caused the conflict by performing **resolution**.
- This makes sure that you can **never find the same** partial assignment (unit propagation will happen before).

(Marques-Silva et al., 2002)

Start with clauses:

- $c_1 : x_1 \vee x_2 \vee x_3$
- $c_2 : x_1 \vee \neg x_2$
- $c_3 : x_1 \vee \neg x_3$

Assign $\neg x_1$. **Propagate** $\neg x_2, \neg x_3$. **Conflicts** with c_1.

Implication learning in SAT solvers

- SAT solvers quickly assign variables but ensures that each assignment is **consistent with unit propagation**.
- Then, analyze which decisions caused the conflict by performing **resolution**.
- This makes sure that you can **never find the same** partial assignment (unit propagation will happen before).

(Marques-Silva et al., 2002)

Start with clauses:

- $c_1 : x_1 \vee x_2 \vee x_3$
- $c_2 : x_1 \vee \neg x_2$
- $c_3 : x_1 \vee \neg x_3$

Assign $\neg x_1$. **Propagate** $\neg x_2, \neg x_3$.
Conflicts with c_1.
Resolve c_1 and c_2:

$$x_1 \vee x_3$$

Implication learning in SAT solvers

- SAT solvers quickly assign variables but ensures that each assignment is **consistent with unit propagation**.
- Then, analyze which decisions caused the conflict by performing **resolution**.
- This makes sure that you can **never find the same** partial assignment (unit propagation will happen before).

(Marques-Silva et al., 2002)

Start with clauses:

- $c_1 : x_1 \vee x_2 \vee x_3$
- $c_2 : x_1 \vee \neg x_2$
- $c_3 : x_1 \vee \neg x_3$

Assign $\neg x_1$. **Propagate** $\neg x_2, \neg x_3$.

Conflicts with c_1.

Resolve c_1 and c_2:

$$x_1 \vee x_3$$

Resolve with c_3:

$c_4 : x_1$

Exact routing+scheduling

Two **observations**:

1. **SAT solvers** can determine deadlocks (and find feasible schedules) much faster than MILP-based solvers.
   … but with **no regard to the objective value**.
2. **Diving heuristics** can be very effective when feasibility is "easy", i.e., no choices are logically wrong.

… can we combine the two?

Idea

Working hypothesis

implication learning (i.e., unit propagation and resolution)
makes diving effective in train routing+scheduling.

Implication graph search

Clauses:

$\neg y_{6,9} \vee \neg y_{9,6}$

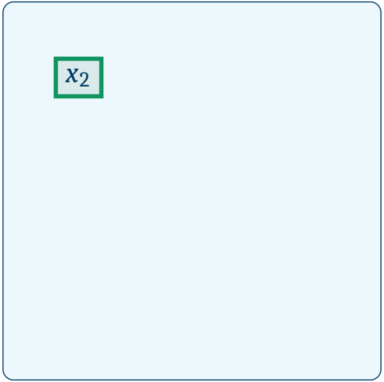
$\neg x_2 \vee \neg x_{12} \vee y_{2,12}$

$\neg x_{12} \vee \neg x_8 \vee \neg x_6 \vee \neg y_{8,6}$

$y_{6,9} \vee y_{8,6} \vee \neg x_8$

Loop:

Implication graph search



Clauses:

$\neg y_{6,9} \lor \neg y_{9,6}$

$\neg x_2 \lor \neg x_{12} \lor y_{2,12}$

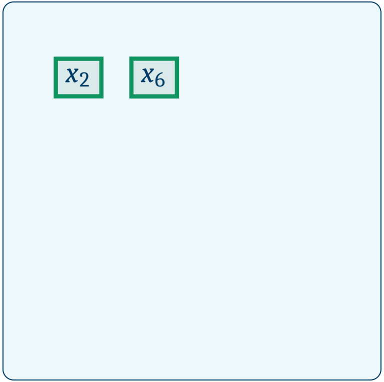
$\neg x_{12} \lor \neg x_8 \lor \neg x_6 \lor \neg y_{8,6}$

$y_{6,9} \lor y_{8,6} \lor \neg x_8$

Loop:

1. Assign a value to variable.

Implication graph search



Clauses:

$\neg y_{6,9} \vee \neg y_{9,6}$

$\neg x_2 \vee \neg x_{12} \vee y_{2,12}$

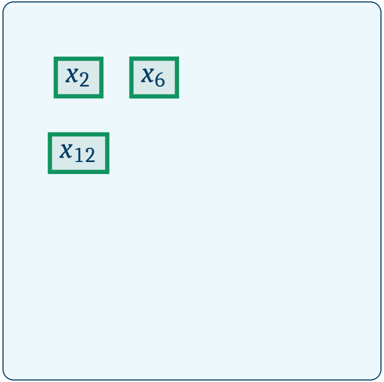
$\neg x_{12} \vee \neg x_8 \vee \neg x_6 \vee \neg y_{8,6}$

$y_{6,9} \vee y_{8,6} \vee \neg x_8$

Loop:

1. Assign a value to variable.

Implication graph search



Clauses:

$\neg y_{6,9} \vee \neg y_{9,6}$

$\neg x_2 \vee \neg x_{12} \vee y_{2,12}$

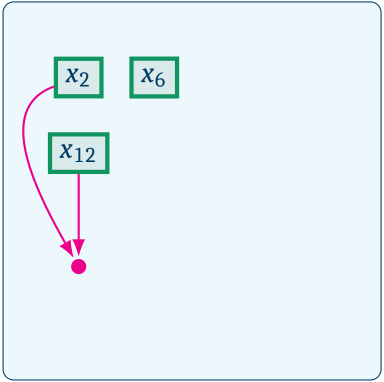
$\neg x_{12} \vee \neg x_8 \vee \neg x_6 \vee \neg y_{8,6}$

$y_{6,9} \vee y_{8,6} \vee \neg x_8$

Loop:

1. Assign a value to variable.

Implication graph search



Clauses:

$\neg y_{6,9} \vee \neg y_{9,6}$

$\neg x_2 \vee \neg x_{12} \vee y_{2,12}$

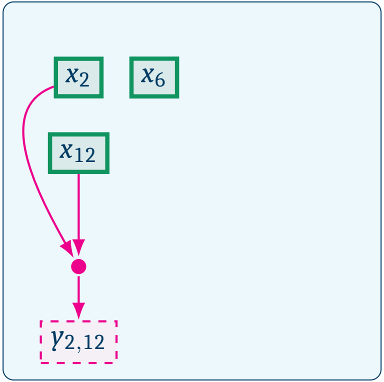
$\neg x_{12} \vee \neg x_8 \vee \neg x_6 \vee \neg y_{8,6}$

$y_{6,9} \vee y_{8,6} \vee \neg x_8$

Loop:

1. Assign a value to variable.
2. Compute unit propagations.

Implication graph search



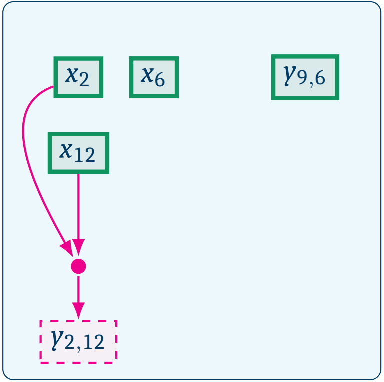
Clauses:

$\neg y_{6,9} \vee \neg y_{9,6}$
$\neg x_2 \vee \neg x_{12} \vee y_{2,12}$
$\neg x_{12} \vee \neg x_8 \vee \neg x_6 \vee \neg y_{8,6}$
$y_{6,9} \vee y_{8,6} \vee \neg x_8$

Loop:

1. Assign a value to variable.
2. Compute unit propagations.

Implication graph search



Clauses:

$\neg y_{6,9} \vee \neg y_{9,6}$

$\neg x_2 \vee \neg x_{12} \vee y_{2,12}$

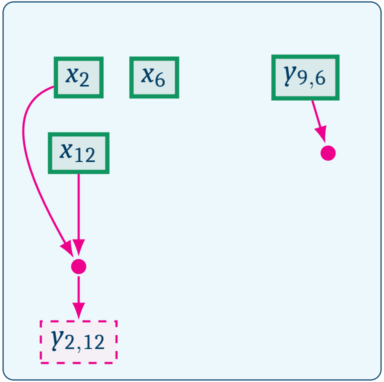
$\neg x_{12} \vee \neg x_8 \vee \neg x_6 \vee \neg y_{8,6}$

$y_{6,9} \vee y_{8,6} \vee \neg x_8$

Loop:

1. Assign a value to variable.
2. Compute unit propagations.

Implication graph search



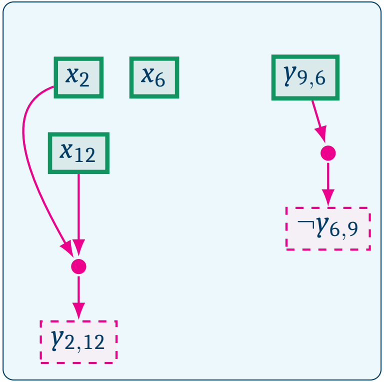
Clauses:

$\neg y_{6,9} \vee \neg y_{9,6}$
$\neg x_2 \vee \neg x_{12} \vee y_{2,12}$
$\neg x_{12} \vee \neg x_8 \vee \neg x_6 \vee \neg y_{8,6}$
$y_{6,9} \vee y_{8,6} \vee \neg x_8$

Loop:

1. Assign a value to variable.
2. Compute unit propagations.

Implication graph search



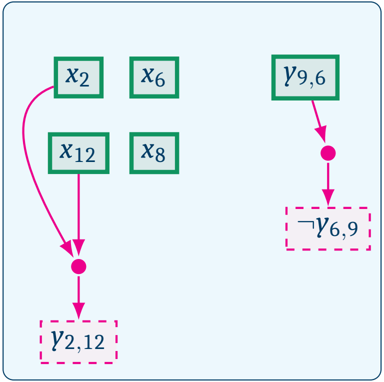
Clauses:

$\neg y_{6,9} \vee \neg y_{9,6}$
$\neg x_2 \vee \neg x_{12} \vee y_{2,12}$
$\neg x_{12} \vee \neg x_8 \vee \neg x_6 \vee \neg y_{8,6}$
$y_{6,9} \vee y_{8,6} \vee \neg x_8$

Loop:

1. Assign a value to variable.
2. Compute unit propagations.

Implication graph search



Clauses:

$\neg y_{6,9} \vee \neg y_{9,6}$

$\neg x_2 \vee \neg x_{12} \vee y_{2,12}$

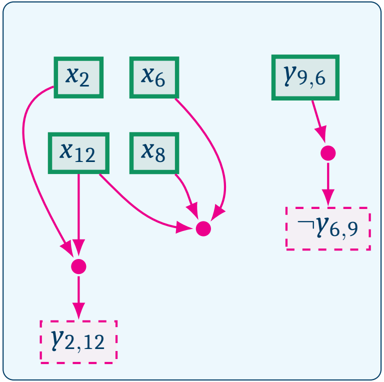
$\neg x_{12} \vee \neg x_8 \vee \neg x_6 \vee \neg y_{8,6}$

$y_{6,9} \vee y_{8,6} \vee \neg x_8$

Loop:

1. Assign a value to variable.
2. Compute unit propagations.

Implication graph search



Clauses:

$\neg y_{6,9} \lor \neg y_{9,6}$

$\neg x_2 \lor \neg x_{12} \lor y_{2,12}$

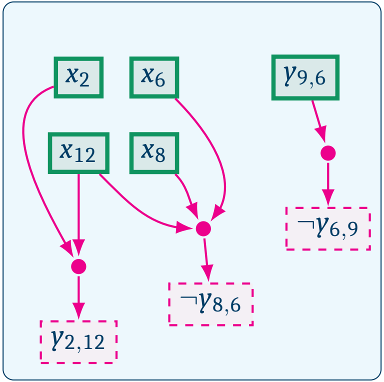
$\neg x_{12} \lor \neg x_8 \lor \neg x_6 \lor \neg y_{8,6}$

$y_{6,9} \lor y_{8,6} \lor \neg x_8$

Loop:

1. Assign a value to variable.
2. Compute unit propagations.

Implication graph search



Clauses:

$\neg y_{6,9} \lor \neg y_{9,6}$

$\neg x_2 \lor \neg x_{12} \lor y_{2,12}$

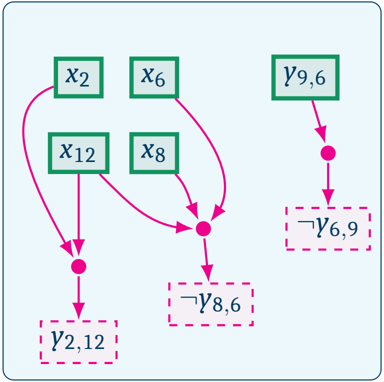
$\neg x_{12} \lor \neg x_8 \lor \neg x_6 \lor \neg y_{8,6}$

$y_{6,9} \lor y_{8,6} \lor \neg x_8$

Loop:

1. Assign a value to variable.
2. Compute unit propagations.

Implication graph search



Clauses:

$\neg y_{6,9} \vee \neg y_{9,6}$

$\neg x_2 \vee \neg x_{12} \vee y_{2,12}$

$\neg x_{12} \vee \neg x_8 \vee \neg x_6 \vee \neg y_{8,6}$

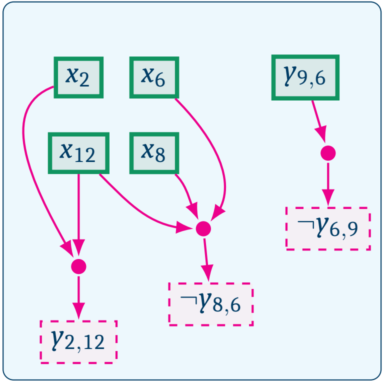
$y_{6,9} \vee y_{8,6} \vee \neg x_8$

Loop:

1. Assign a value to variable.
2. Compute unit propagations.
3. Find violated clause

   $y_{6,9} \vee y_{8,6} \vee \neg x_8$

Implication graph search



Clauses:

$\neg y_{6,9} \vee \neg y_{9,6}$

$\neg x_2 \vee \neg x_{12} \vee y_{2,12}$

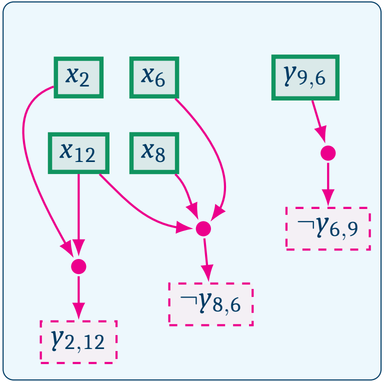
$\neg x_{12} \vee \neg x_8 \vee \neg x_6 \vee \neg y_{8,6}$

$y_{6,9} \vee y_{8,6} \vee \neg x_8$

Loop:

1. Assign a value to variable.
2. Compute unit propagations.
3. Find violated clause
   $y_{6,9} \vee y_{8,6} \vee \neg x_8$
4. Resolve to decisions
   $\neg y_{9,6} \vee \neg x_8 \vee \neg x_{12} \vee \neg x_6$

Implication graph search



Clauses:

$\neg y_{6,9} \vee \neg y_{9,6}$

$\neg x_2 \vee \neg x_{12} \vee y_{2,12}$

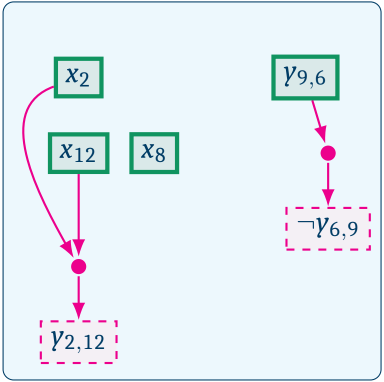
$\neg x_{12} \vee \neg x_8 \vee \neg x_6 \vee \neg y_{8,6}$

$y_{6,9} \vee y_{8,6} \vee \neg x_8$

Loop:

1. Assign a value to variable.
2. Compute unit propagations.
3. Find violated clause
   $y_{6,9} \vee y_{8,6} \vee \neg x_8$
4. Resolve to decisions
   $\neg y_{9,6} \vee \neg x_8 \vee \neg x_{12} \vee \neg x_6$
5. Retract any of the decisions.

Implication graph search



Clauses:

$\neg y_{6,9} \vee \neg y_{9,6}$

$\neg x_2 \vee \neg x_{12} \vee y_{2,12}$

$\neg x_{12} \vee \neg x_8 \vee \neg x_6 \vee \neg y_{8,6}$

$y_{6,9} \vee y_{8,6} \vee \neg x_8$

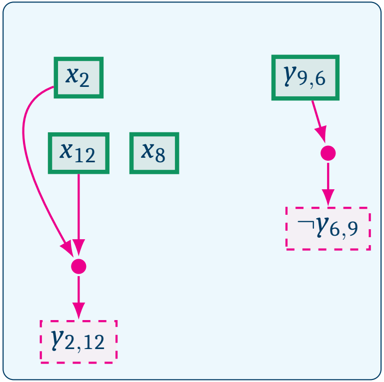
Loop:

1. Assign a value to variable.
2. Compute unit propagations.
3. Find violated clause
   $y_{6,9} \vee y_{8,6} \vee \neg x_8$
4. Resolve to decisions
   $\neg y_{9,6} \vee \neg x_8 \vee \neg x_{12} \vee \neg x_6$
5. Retract any of the decisions.

SINTEF

Europe's Rail FP5 TRANS4M-R Transforming Europe's Rail Freight

Implication graph search



Clauses:

$\neg y_{6,9} \vee \neg y_{9,6}$

$\neg x_2 \vee \neg x_{12} \vee y_{2,12}$

$\neg x_{12} \vee \neg x_8 \vee \neg x_6 \vee \neg y_{8,6}$

$y_{6,9} \vee y_{8,6} \vee \neg x_8$

Loop:

1. Assign a value to variable.
2. Compute unit propagations.
3. Find violated clause
   $y_{6,9} \vee y_{8,6} \vee \neg x_8$
4. Resolve to decisions
   $\neg y_{9,6} \vee \neg x_8 \vee \neg x_{12} \vee \neg x_6$
5. Retract **any** of the decisions.

Exact algorithm! Related to ***resolution search***.
(Chvátal 1997, Hanafi and Glover 2002)

The way ahead

- ✓ Naïve implementation
- ✓ Finds solutions to American freight instances from DISPLIB (`wab_*`).
- … but with no objective or heuristics, objective values are **poor**

- ⏱ Efficient **unit propagation**
- ⏱ Efficient, incremental **scheduling graph** updates
- ⏱ **Heuristic** decisions and retractions based on objective values
- …

Technology for a better society