

# JuLS - A Julia Local Search Solver

Axel Navarro

Amazon Science

Tuesday, November 18, 2025

# Plan

1 Amazon usecase

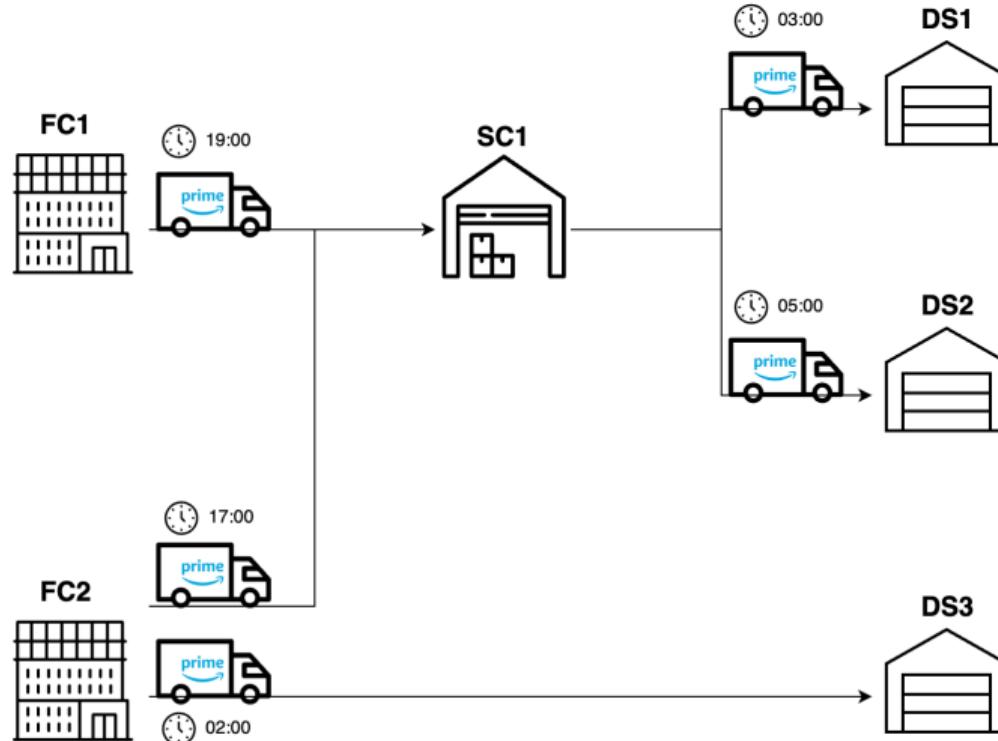
2 Model

3 Implementation

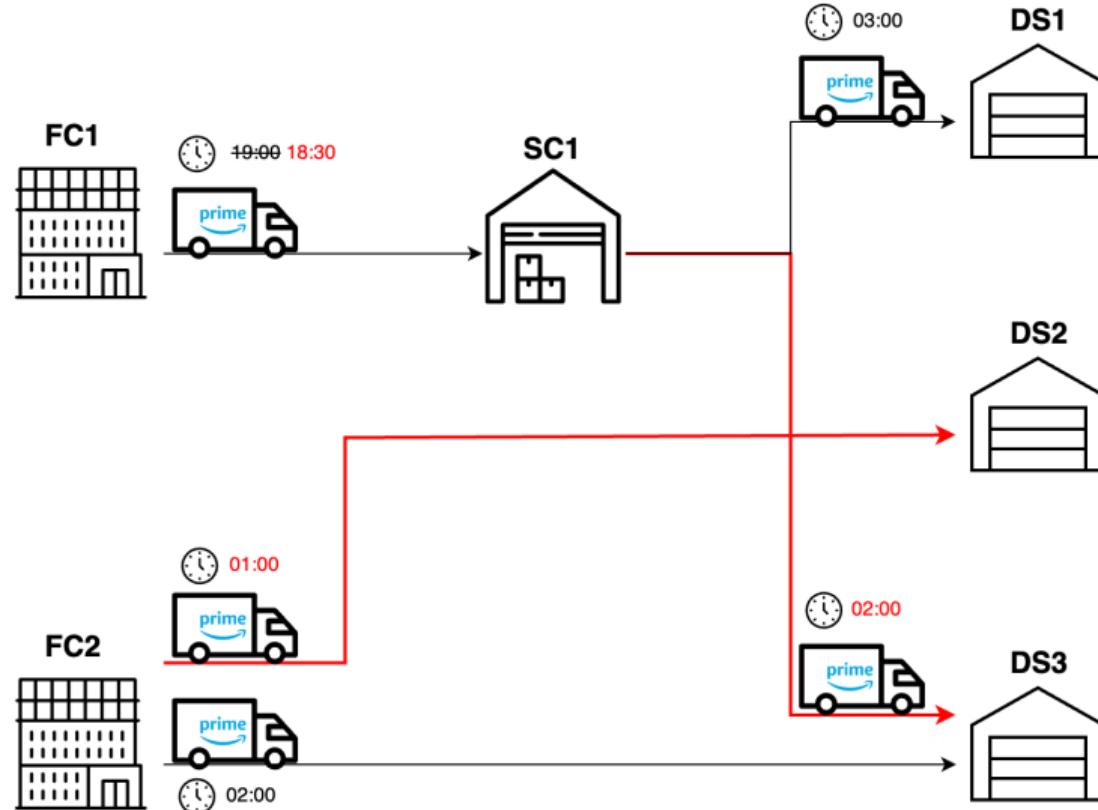
4 Conclusion

# Amazon usecase

# Middle Mile Network Design



# Middle Mile Network Design



# Simplified modelization

## Decision variables

- Trucks departure time (can be null if the route is not configured)

# Simplified modelization

## Decision variables

- Trucks departure time (can be null if the route is not configured)

## Constraints

- Warehouse resource : each time slot has a certain outbound and inbound capacity

# Simplified modelization

## Decision variables

- Trucks departure time (can be null if the route is not configured)

## Constraints

- Warehouse resource : each time slot has a certain outbound and inbound capacity

## Objectives

- Maximize the delivery speed with forecasted demand (**black box**)
- Minimize transportation cost

# Objective

Build a generic **local search solver** with the following features :

## ① Performance

- Fast iteration process
- Black box optimization

## ② Modularity

- Easy integration of external evaluation tools
- Multiple search strategy support

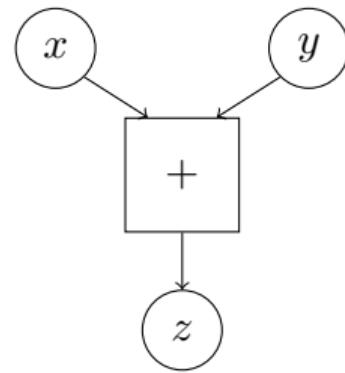
# Model

# Invariant

$$z = x + y$$

# Invariant

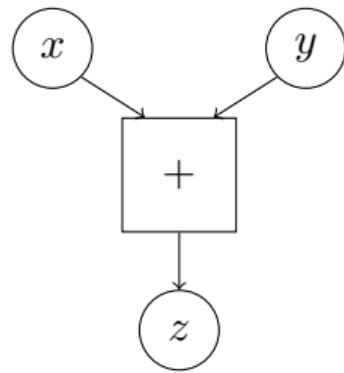
$$z = x + y$$



# Invariant

$$z = x + y$$

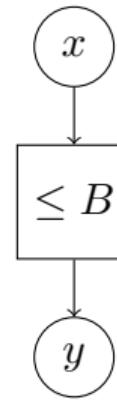
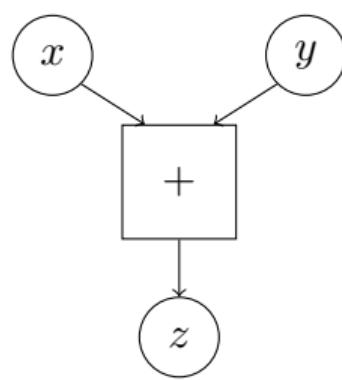
$$x \leq B$$



# Invariant

$$z = x + y$$

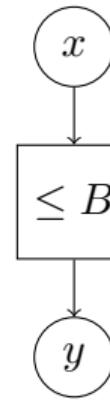
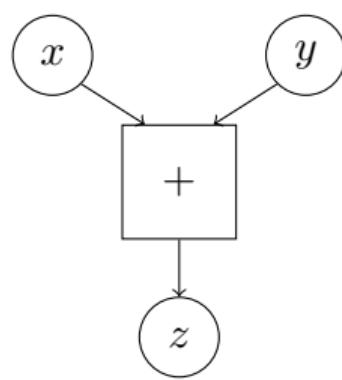
$$x \leq B$$



# Invariant

$$z = x + y$$

$$x \leq B$$



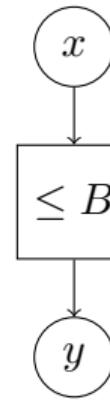
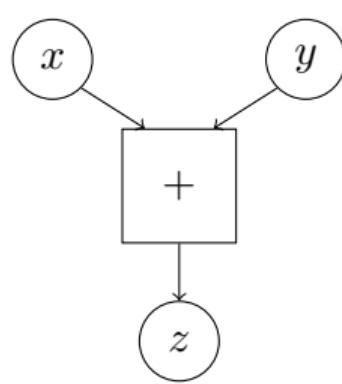
$$y = \max(0, x - B)$$

# Invariant

$$z = x + y$$

$$x \leq B$$

$$y = f(x_1, x_2, x_3)$$



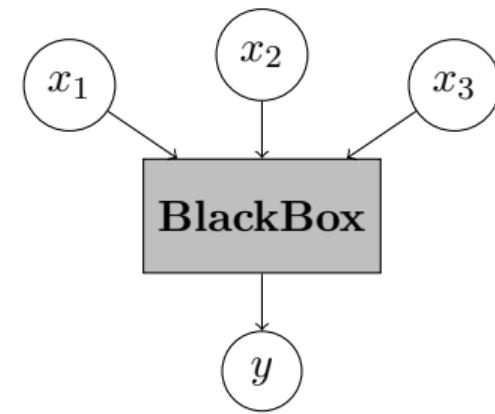
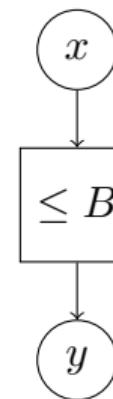
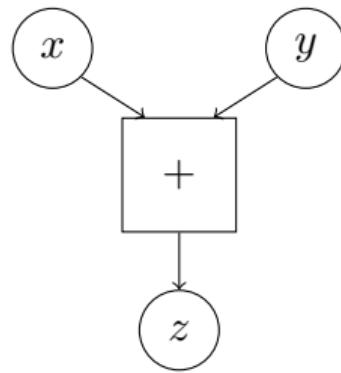
$$y = \max(0, x - B)$$

# Invariant

$$z = x + y$$

$$x \leq B$$

$$y = f(x_1, x_2, x_3)$$



$$y = \max(0, x - B)$$

# DAG structure

$x_1$

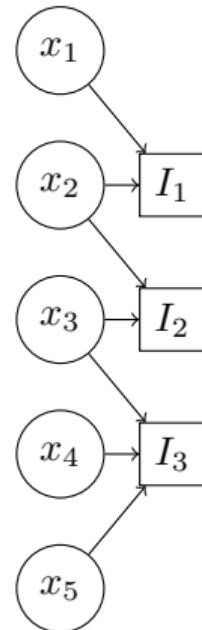
$x_2$

$x_3$

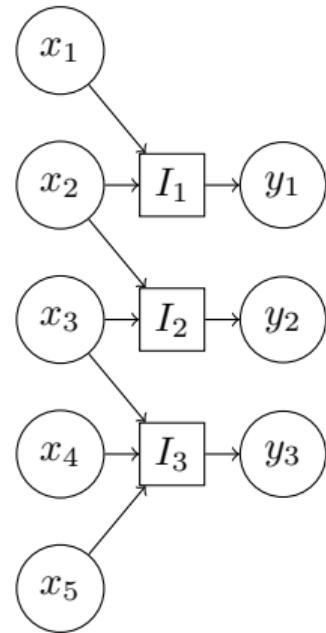
$x_4$

$x_5$

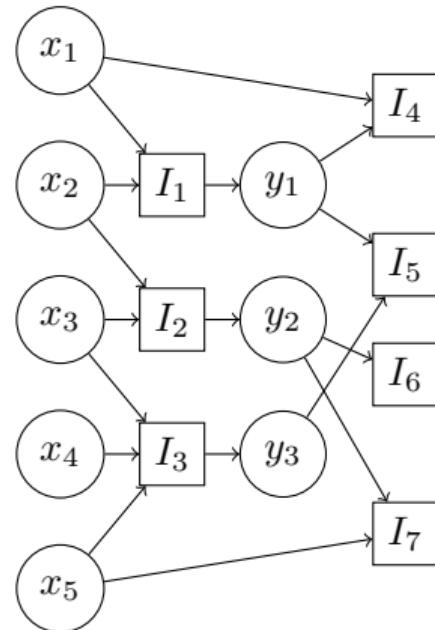
# DAG structure



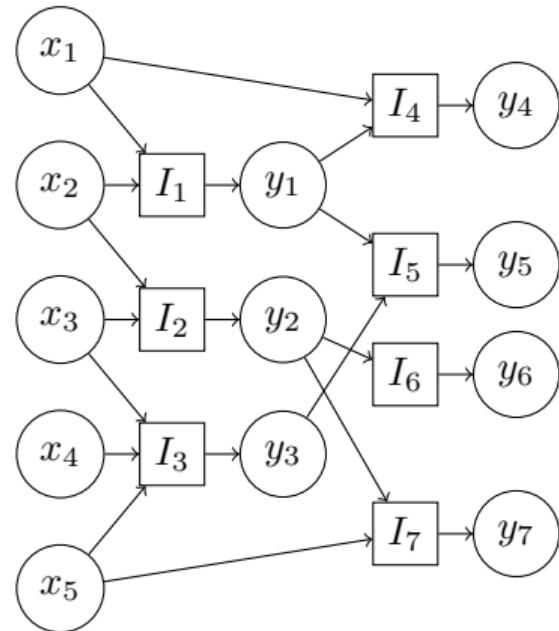
# DAG structure



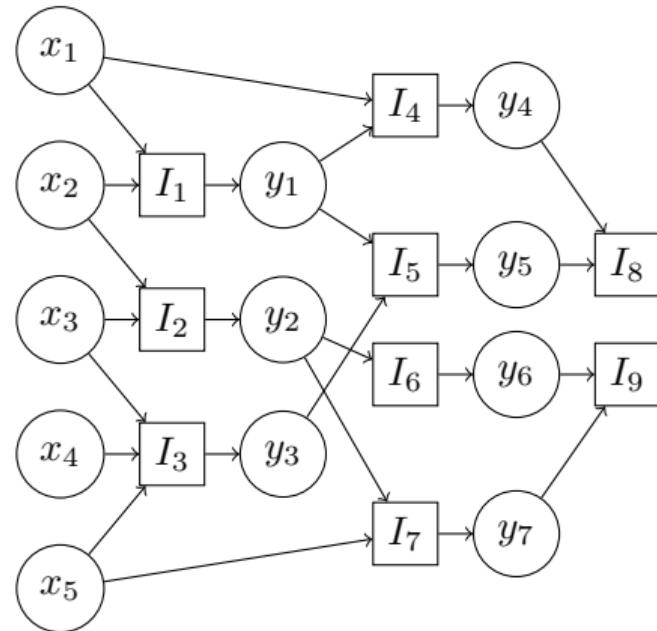
# DAG structure



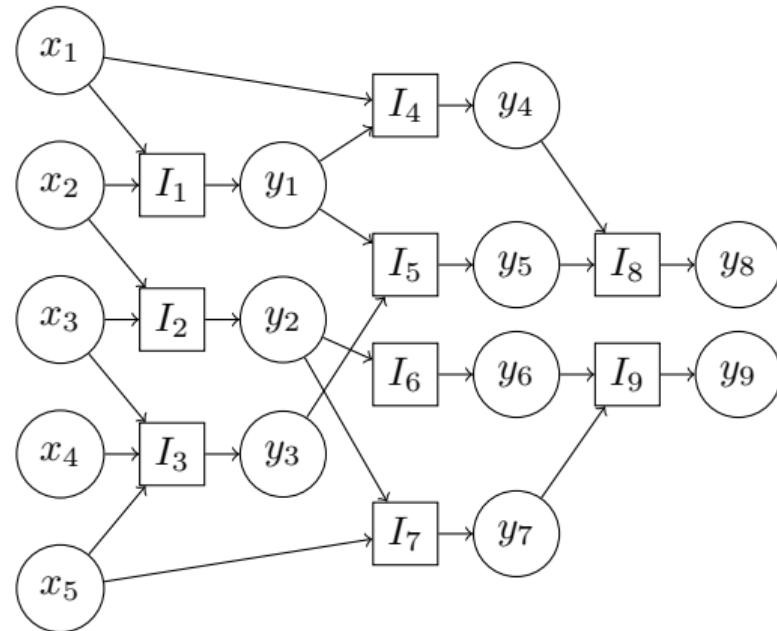
# DAG structure



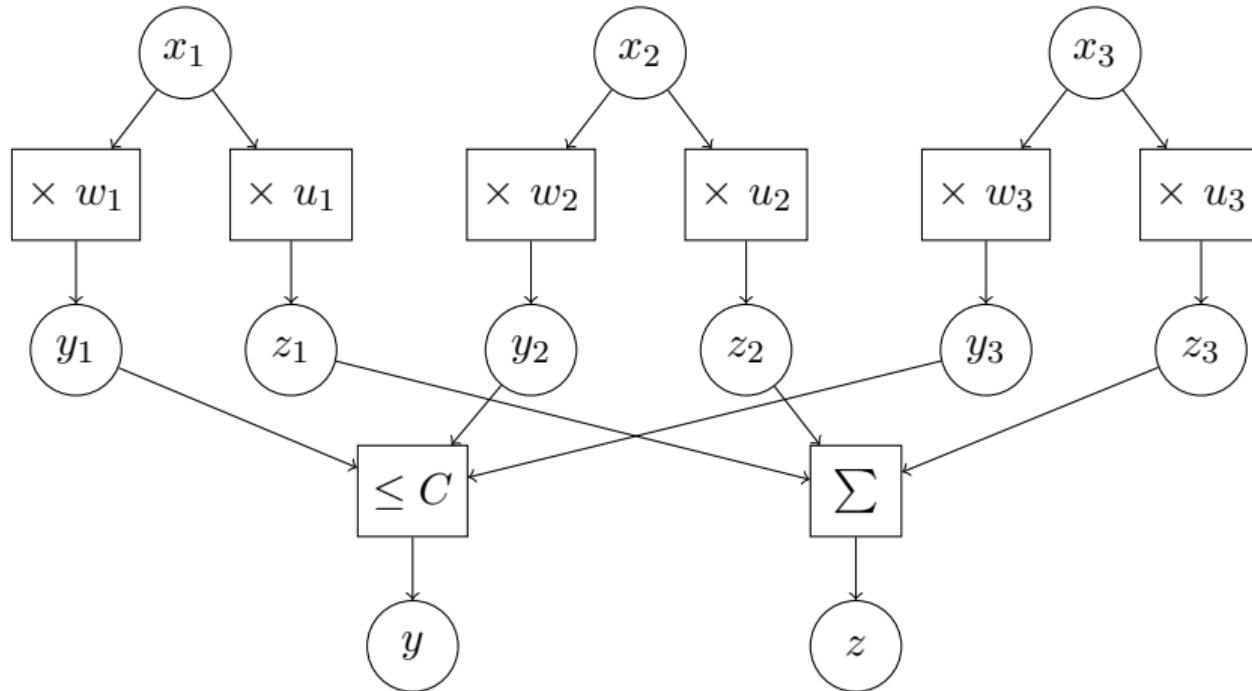
# DAG structure



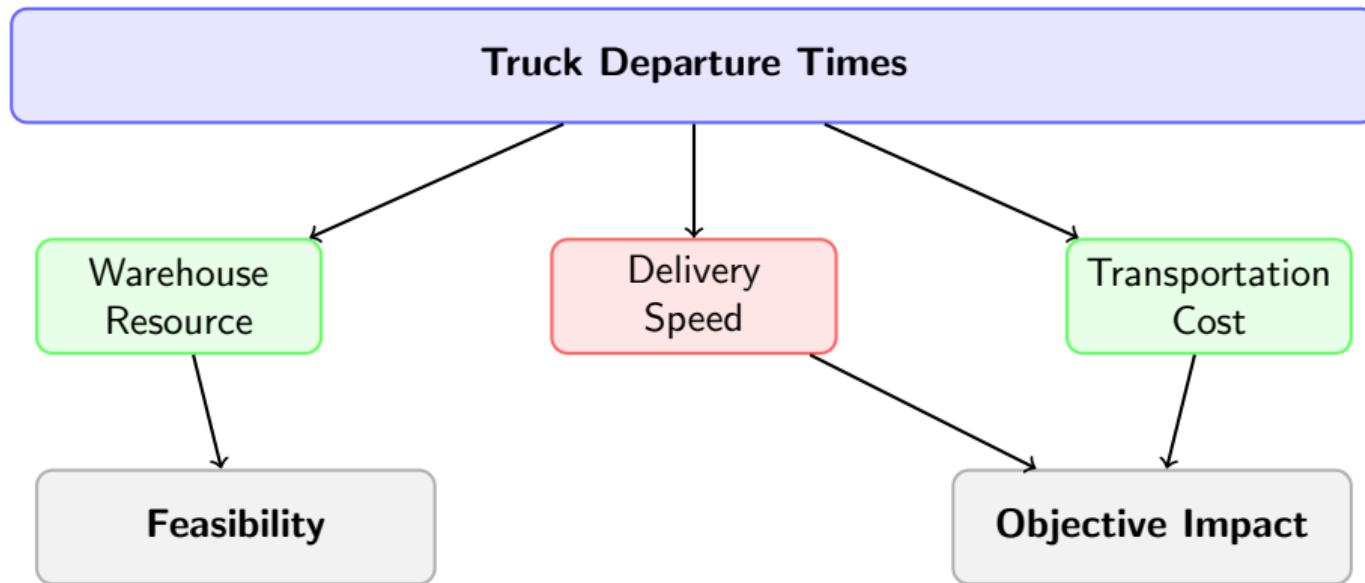
# DAG structure



# DAG structure

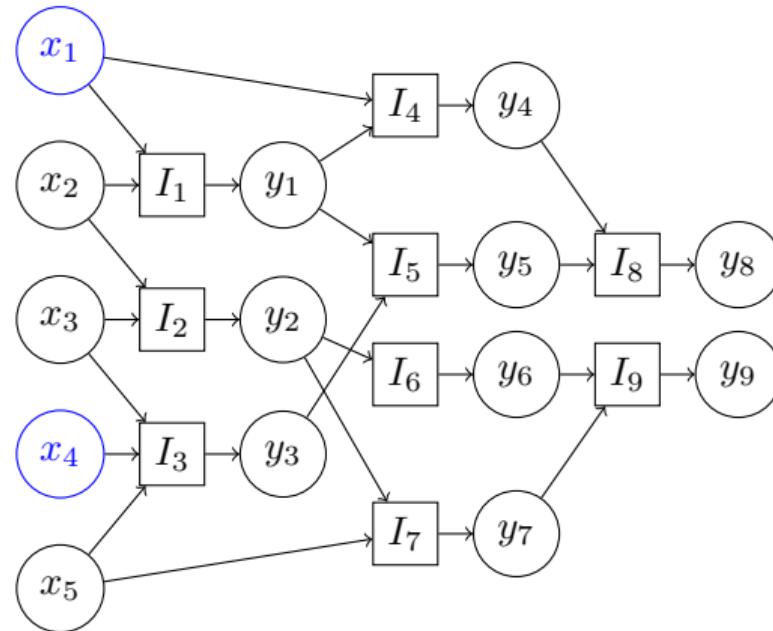


# DAG structure

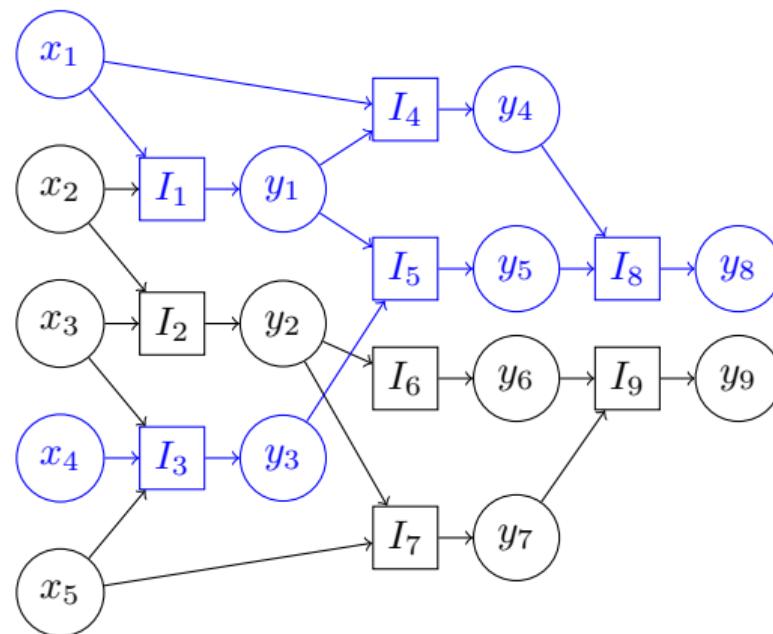


- 10k decision variables and 500k invariants on average

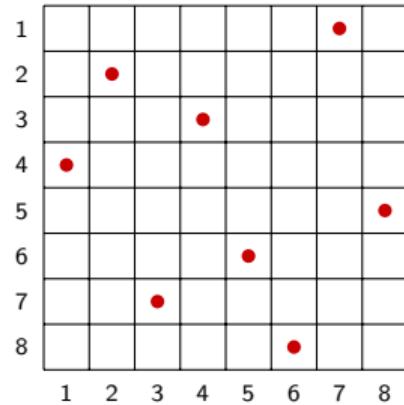
# Move evaluation



# Move evaluation

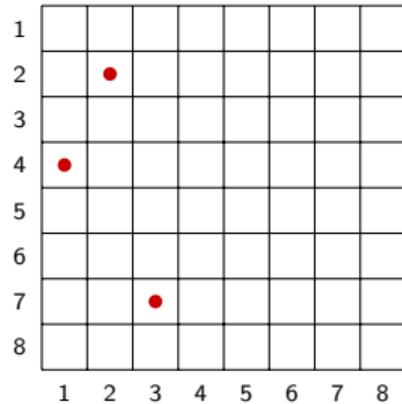


# Move evaluation



Solution  $\sigma$

# Move evaluation

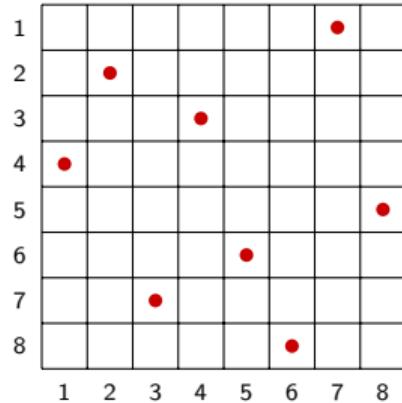


Solution  $\sigma$

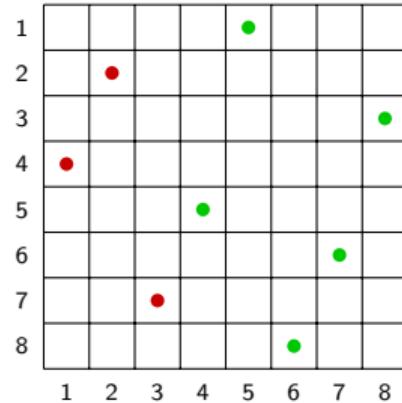
Relaxation of column 4, 5, 6, 7, 8

$\rightarrow 8^5 = 32768$  moves to evaluate

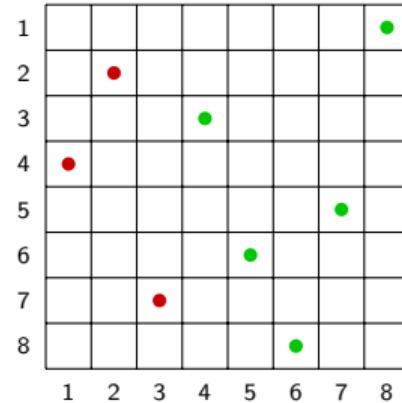
# Move evaluation



Solution  $\sigma$

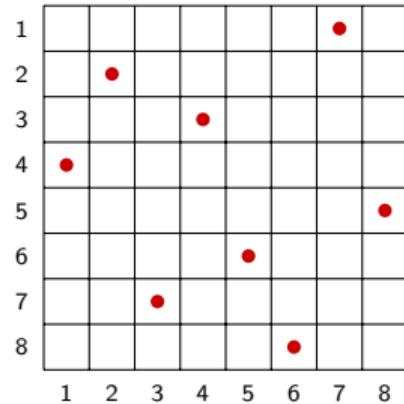


Move  $m_1$

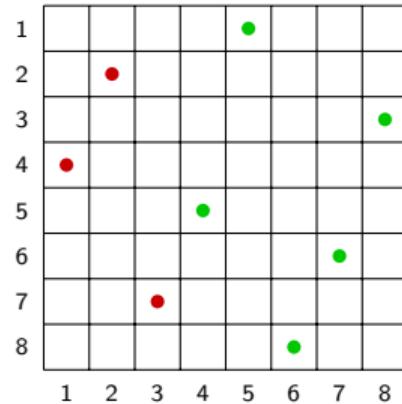


Move  $m_2$

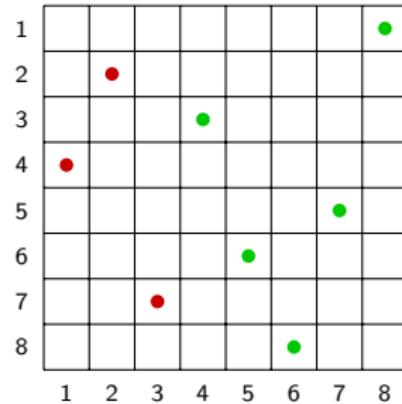
# Move evaluation



Solution  $\sigma$



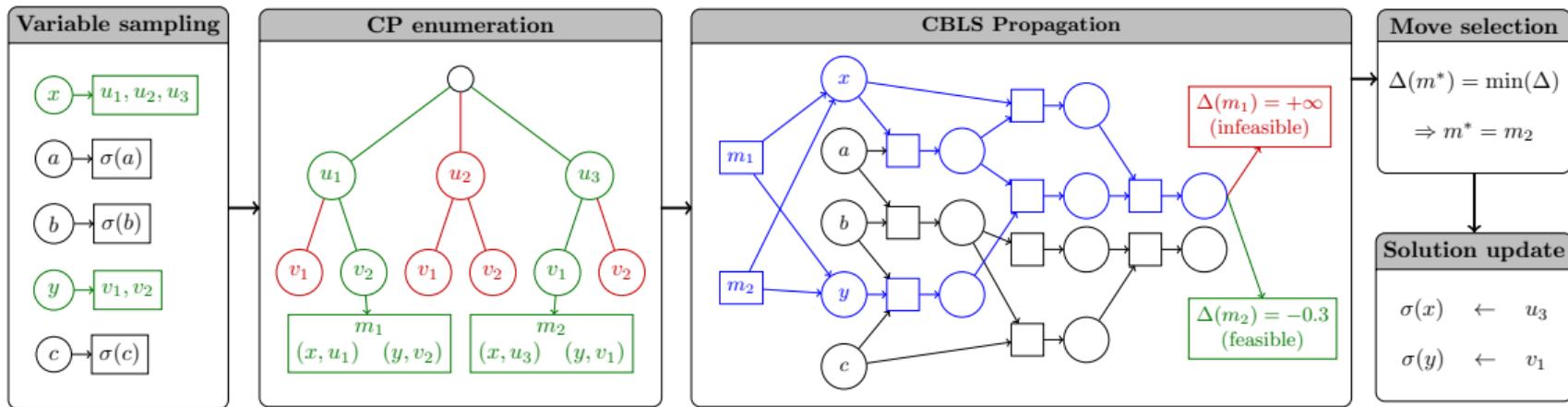
Move  $m_1$



Move  $m_2$

Solution  $\rightarrow$  **Constraint Programming** enumeration

# Iteration process



# Implementation

# Main components of the model

## Neighborhood heuristic (neigh)

Generates potential moves from a given solution.

# Main components of the model

## Neighborhood heuristic (neigh)

Generates potential moves from a given solution.

## Move filter (move\_filter)

Filters out a subset of generated moves (Constraint Programming, random sampling...).

# Main components of the model

## Neighborhood heuristic (neigh)

Generates potential moves from a given solution.

## Move filter (move\_filter)

Filters out a subset of generated moves (Constraint Programming, random sampling...).

## Move evaluator or DAG (dag)

Evaluates objective impact and feasibility of a move using CBLS propagation.

# Main components of the model

## Neighborhood heuristic (neigh)

Generates potential moves from a given solution.

## Move filter (move\_filter)

Filters out a subset of generated moves (Constraint Programming, random sampling...).

## Move evaluator or DAG (dag)

Evaluates objective impact and feasibility of a move using CBLS propagation.

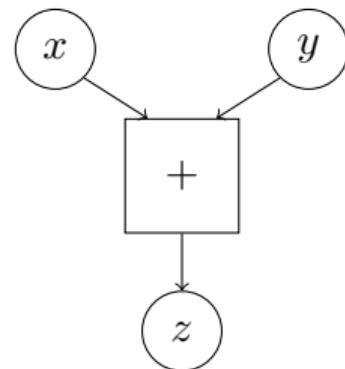
## Move selection heuristic (pick)

Selects the evaluated move that will be applied

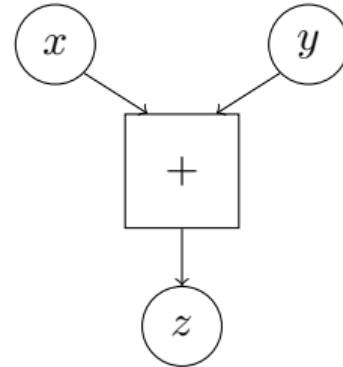
# Iteration pseudo code

```
1 moves = generate_moves(model.neigh)
2
3 filtered_moves = filter_moves(model.move_filter, moves)
4
5 Threads.@threads for (i, move) in filtered_moves
6     evaluated_moves[i] = eval(model.dag, move)
7 end
8
9 picked_move = pick_a_move(model.pick, evaluated_moves)
10
11 apply_move!(model, picked_move)
```

# DAG construction

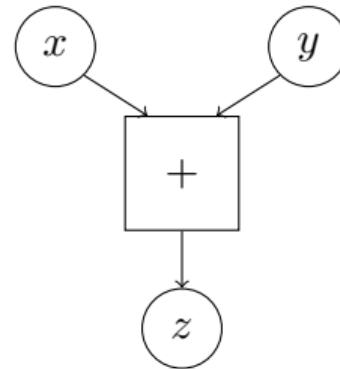


# DAG construction



```
1 @constraint(model, x + y == z) # JuMP
```

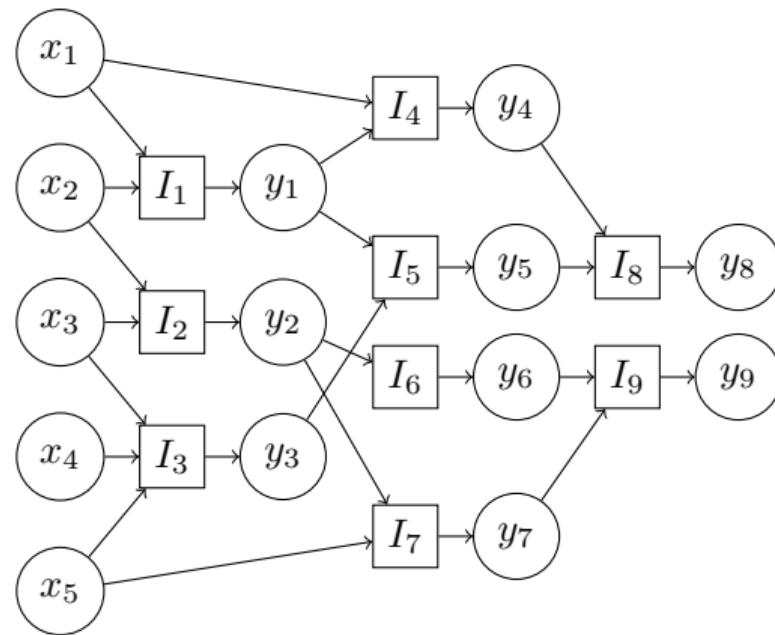
# DAG construction



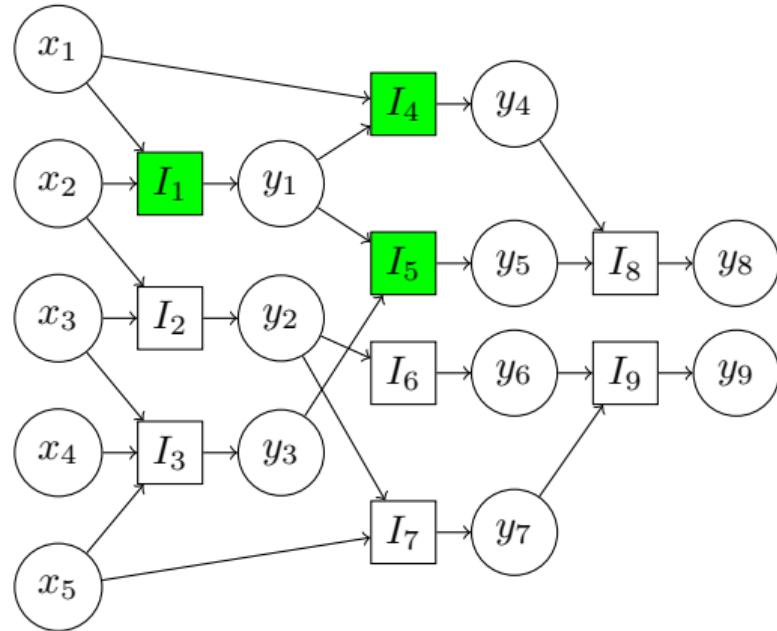
```
1 @constraint(model, x + y == z) # JuMP
```

```
1 z = add_invariant!(dag, SumInvariant(); parent_variables = [x, y]) # JuLS
```

# DAG construction

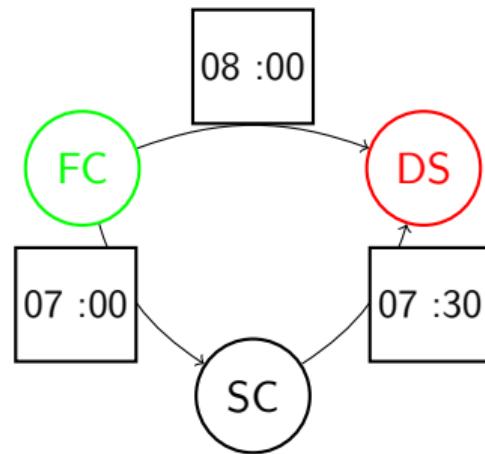


# DAG construction



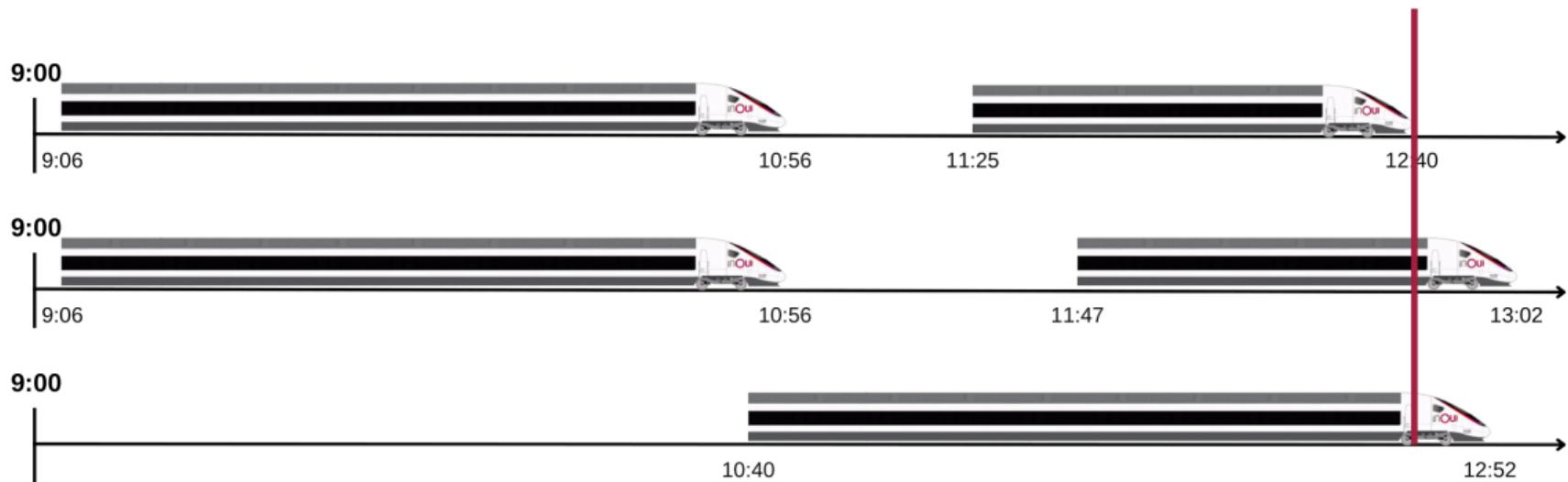
```
1 add_invariant(dag, I; parent_variables = X, using_cp = true)
```

# Routing invariant



# The Earliest Arrival Time Problem

**Earliest arrival = 12:40**



Leaving after 9 :00, what is the journey that will make me arrive at destination at the earliest time ?

# The Connection Scan Algorithm ([1])

- [1] Julian DIBBELT et al. “Intriguingly Simple and Fast Transit Routing”. In : *Experimental Algorithms*. T. 7933. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013.

# The Connection Scan Algorithm ([1])

- [1] Julian DIBBELT et al. “Intriguingly Simple and Fast Transit Routing”. In : *Experimental Algorithms*. T. 7933. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013.

Given a sorted array of trains that go along a set of nodes at a given time, and an earliest departure time, this algorithm finds the journey between two nodes that will arrive the earliest.

# The Connection Scan Algorithm ([1])

- [1] Julian DIBBELT et al. “Intriguingly Simple and Fast Transit Routing”. In : *Experimental Algorithms*. T. 7933. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013.

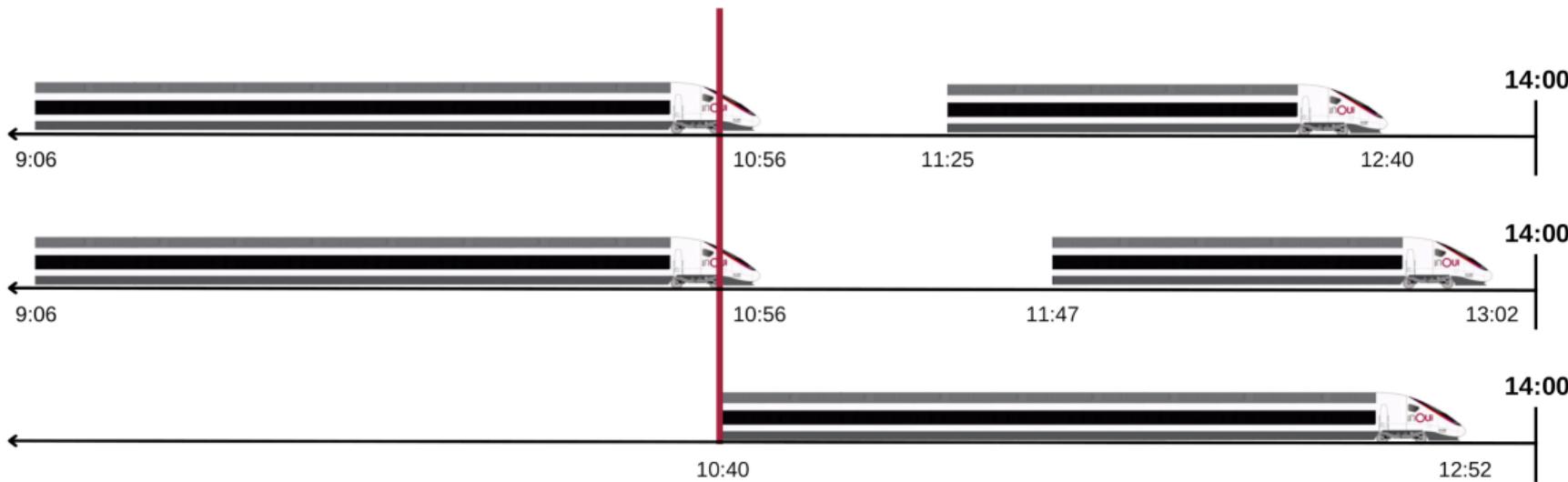
Given a sorted array of trains that go along a set of nodes at a given time, and an earliest departure time, this algorithm finds the journey between two nodes that will arrive the earliest.

- In London network (20k nodes), the best journey is found in **1.3ms** !

# The reversed CSA, for Latest Departure Time Problem

## The Latest Departure Time Problem

Latest departure = 10:40



Arriving before 14:00, what is the journey that will make me leave the origin at the latest time?

# Conclusion

# Conclusion

## Advantages

- Highly modular solver with customizable heuristics
- Efficient parallel evaluation of multiple moves per iteration
- Leveraging both classical constraint programming and black-box constraint evaluation

## Limitations

- Parallel implementation can lead to memory management challenges
- Converting optimization problems into DAG structure is not user friendly