

# Selected Topics in Network Optimization: Aligning BDDs for a Facility Location Problem and a Search Tree Method for DSPI.

Dissertation defense

Alexey Bochkarev

2021-11-02 Tue, 08-30, Zoom / Freeman Hall 129



**Thanks for joining!**

# Presentation outline

## 1 Aligning BDDs

- On BDD representations
- BDD alignment (“original”) problem
- The simplified problem
- Numerical experiments

## 2 A BDD-based approach to a Facility Location Problem

- Problem description
- BDD representation
- Solving the CPP
- Numerical experiments

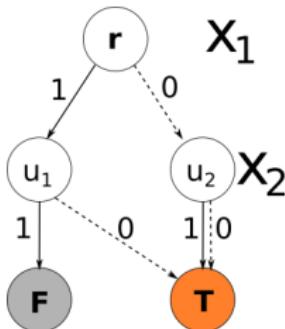
## 3 Monte Carlo Tree Search for DSPI

- Problem formulation
- MCTS framework
- Numerical experiments
- On correctness

# What is an (0)BDD?

## In a nutshell:

- A (maybe weighted) layered (D)AG
- Two outgoing arcs from each node
- One root, two terminals



## Applications:

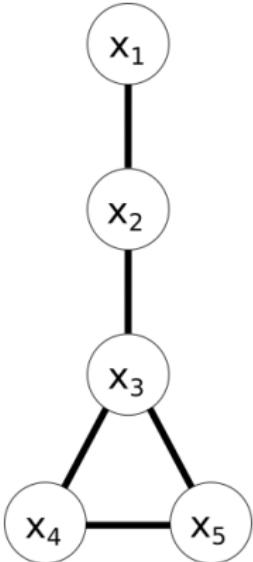
- Can encode a Boolean function
- ...
- ... or a combinatorial opt problem.

## How it works:

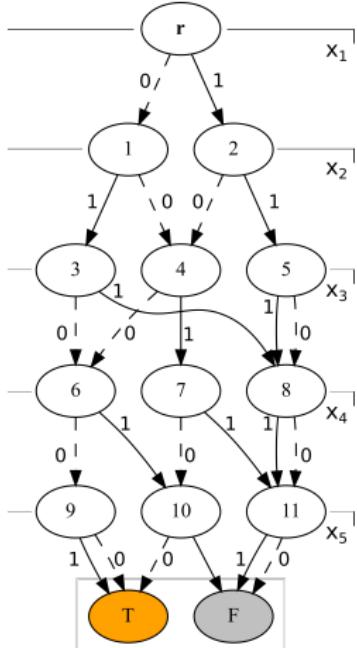
- A layer corresponds to a “decision” (binary variable),
- **one-arc** (or yes-arc) –  $x_i = 1$ ,
- **zero-arc** (or no-arc) –  $x_i = 0$ .
- A **path** from root to terminal corresponds to an assignment of all variables.

## Order of variables matters: a MIS example

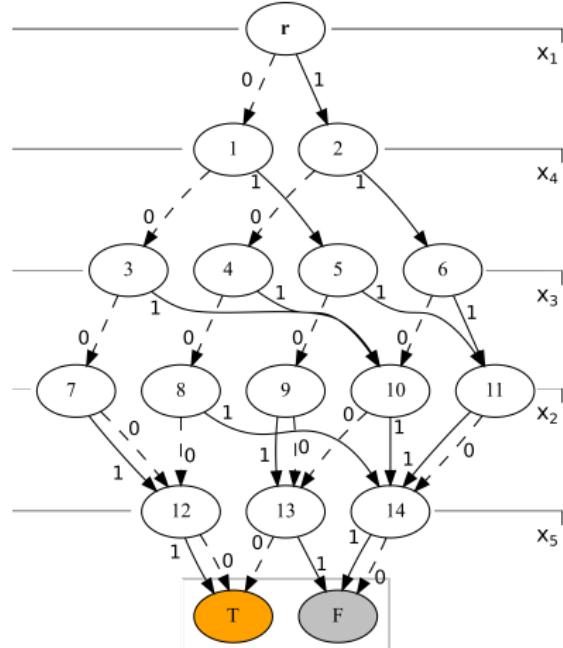
## Original graph:



14 nodes;



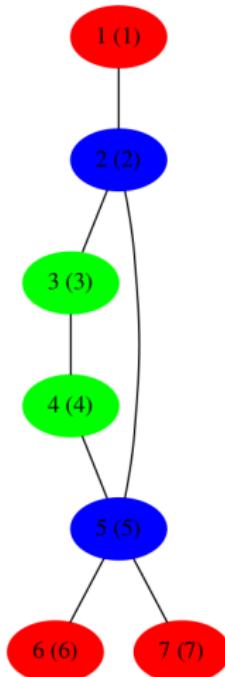
17 nodes;



The **same** order of variables also matters

- Many problems can be reformulated as a set of (connected) instances over a collection of BDDs.
  - If we have the same order of variables: Lozano et al. proposed an algorithm how to tackle such instances.
  - So, finding a good shared order of variables might be desirable.

## A motivating example: t-UFLP



**Assume:**

- A graph; can **locate** a facility at any point,
  - A located facility **covers** all neighboring points.

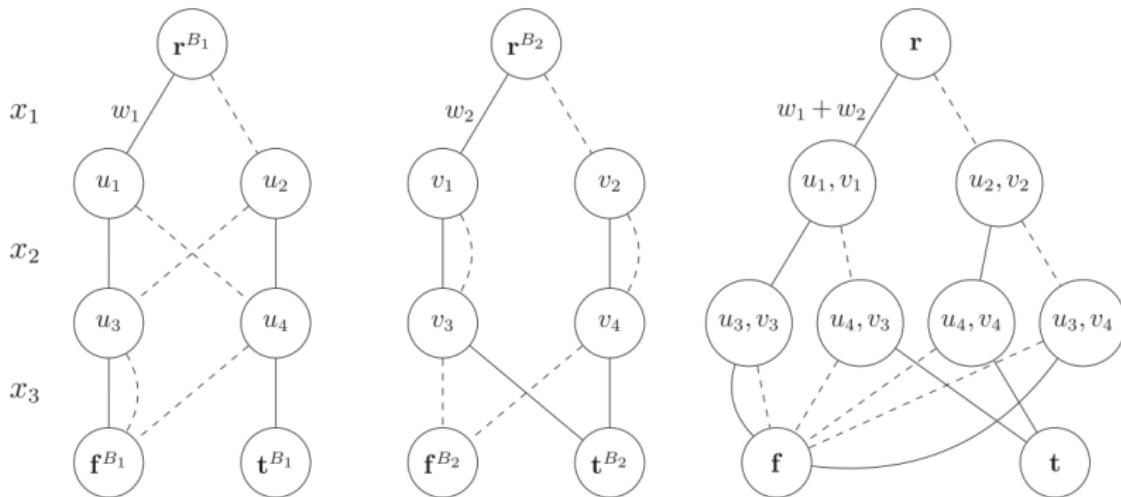
**Objective:** cover all points at min cost.

## **Constraints:**

- Each point implies a facility “type”,
  - There is a limit on number of facilities by type.

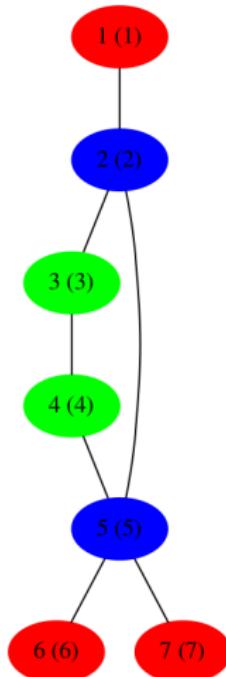
# How to merge two diagrams?

**Figure 4.** Building an Intersection BDD



{Source: Lozano et al. 2020}

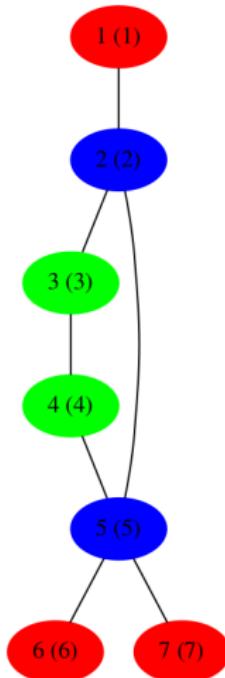
# A motivating example: t-UFLP



## Plan of attack:

- Formulate the problem with two BDDs,
- Enforce the shared order of variables,
- Merge (“intersect”) the two diagrams into one\*
- Solve a shortest-path in the intersection DD.

# A motivating example: t-UFLP



## Plan of attack:

- Formulate the problem with two BDDs,
- Enforce the shared order of variables, ← this part
- Merge (“intersect”) the two diagrams into one\*
- Solve a shortest-path in the intersection DD.

Aligning BDDs

oooooooo●oooooooooooooooooooo

On BDD representations

BDDs for t-UFLP

oooooooooooo

Monte Carlo Tree Search for DSPI

oooooooooooooooooooooooooooo

Conclusion

ooo

## In this section:

- BDD alignment problem,
- A simplified model / heuristic,
- Numerical experiments on aligning BDDs.

# BDD alignment problem

Let us denote  $T^*[D, \mathbf{v}]$  – a “transformed” version of BDD  $D$  to variable order  $\mathbf{v} = (v_1, \dots, v_N)$ . Then:

## BDD Alignment Problem / “original” problem

$$s^* = \min_{\mathbf{v}} \left\{ |T^*[A, \mathbf{v}]| + |T^*[B, \mathbf{v}]| \right\},$$

where  $|D|$  is the number of nodes in diagram  $D$ . The problem is NP-hard (because the min single BDD is NP-hard).

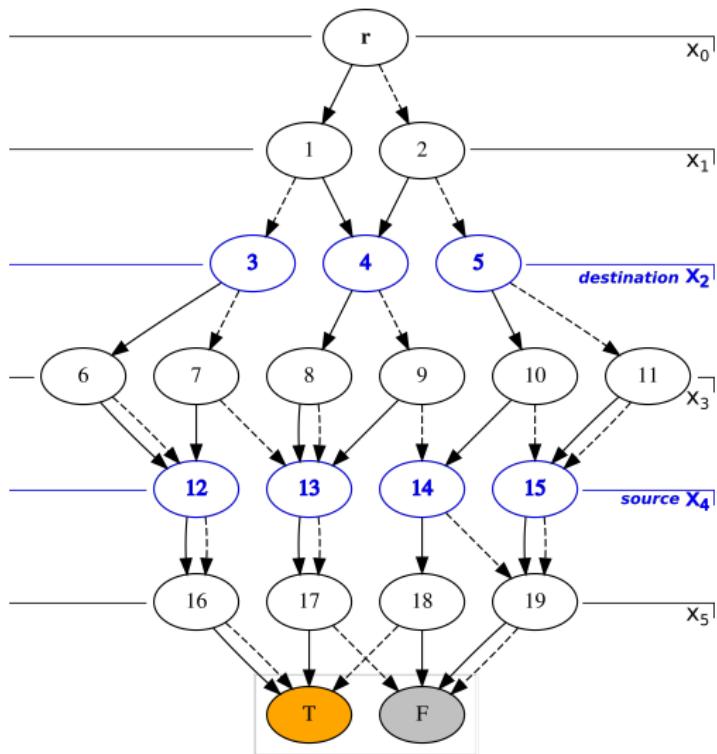
# Research context

- Vast literature on minimizing single BDD.
- One of the central ideas: “Dynamic variable reordering” / **Sifting** algorithm (we will use it as a baseline).
- Limited consideration of the multi-BDD version, and all working with BDDs (obviously).

## The purpose of this work

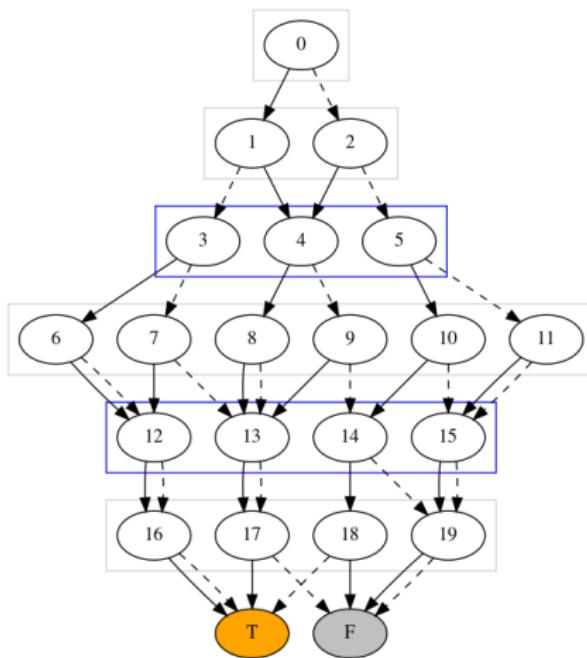
Try to avoid manipulations with BDD as much as possible, by introducing a “simpler” auxiliary problem.

# How the BDDs are transformed?



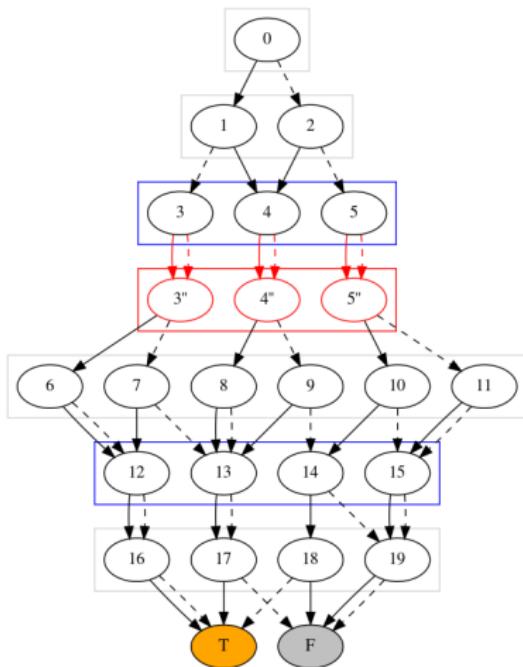
# How the BDDs are transformed?

Step 1.a: duplicate B



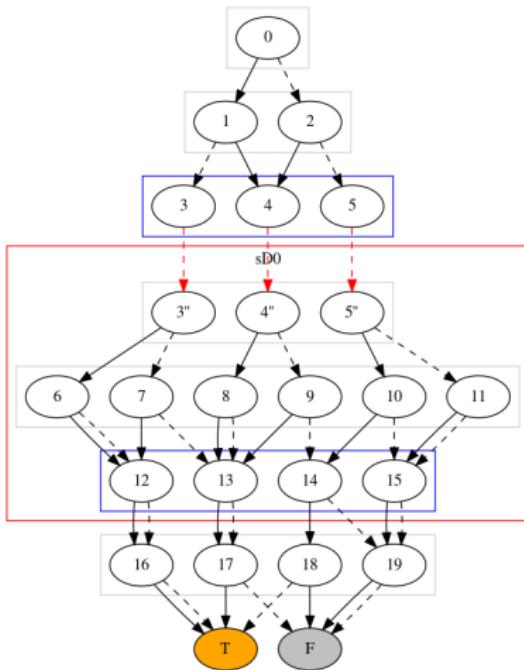
# How the BDDs are transformed?

Step 1.b: duplicate L\_d



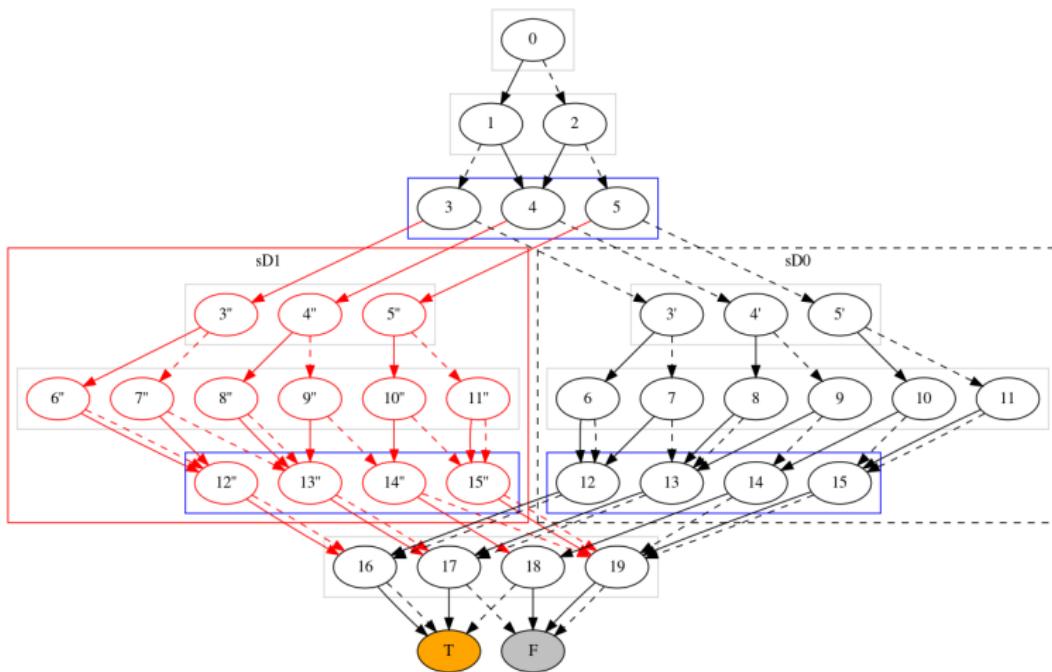
# How the BDDs are transformed?

Step 2: create sD0...



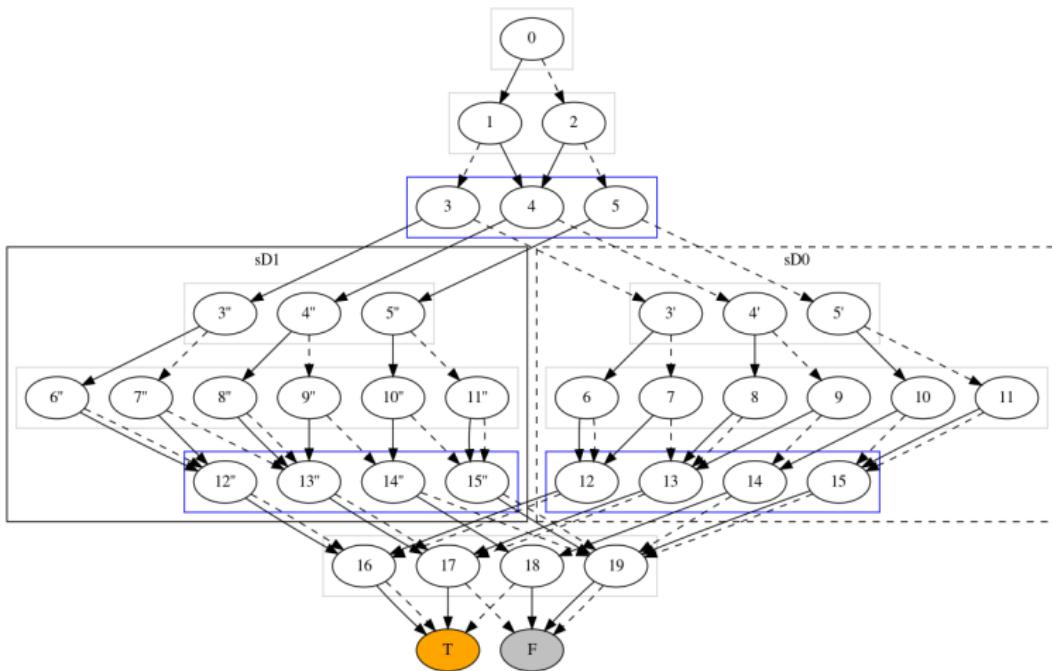
# How the BDDs are transformed?

Step 2: ... and create sD1



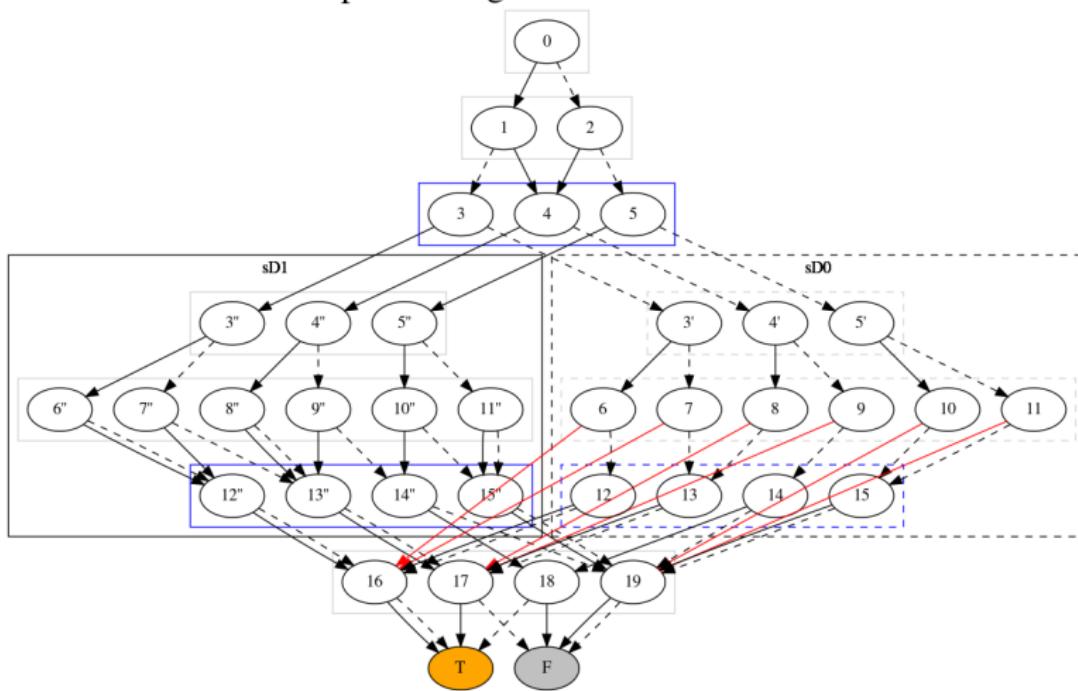
# How the BDDs are transformed?

Step 3: reassign the L\_d label



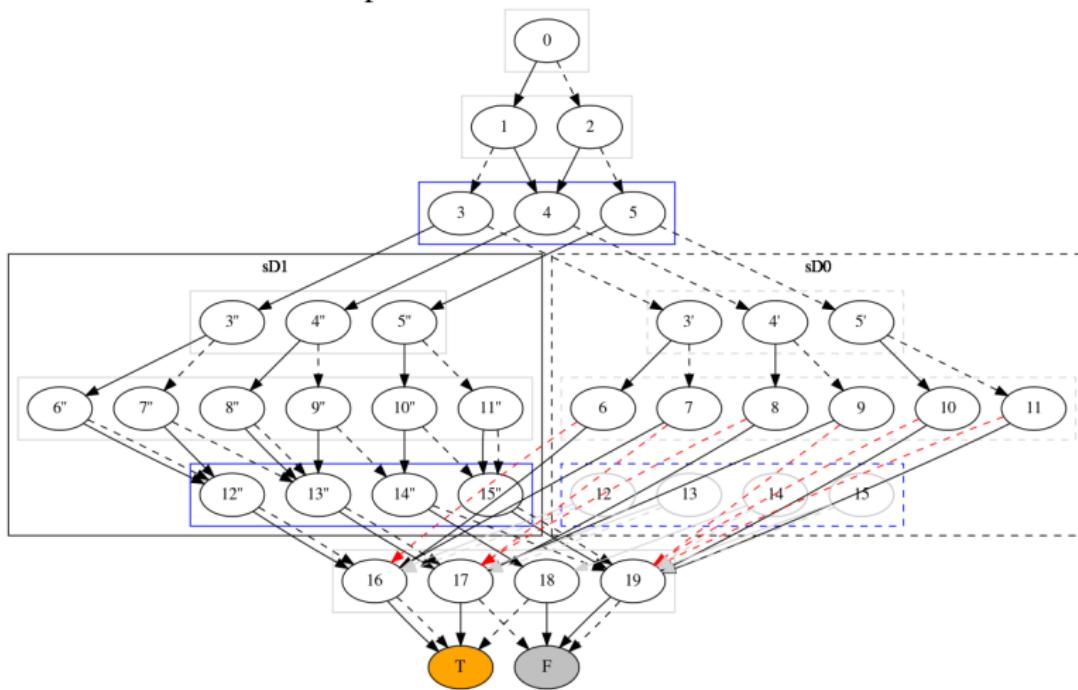
# How the BDDs are transformed?

Step 4: reassign one-arcs for sD0...



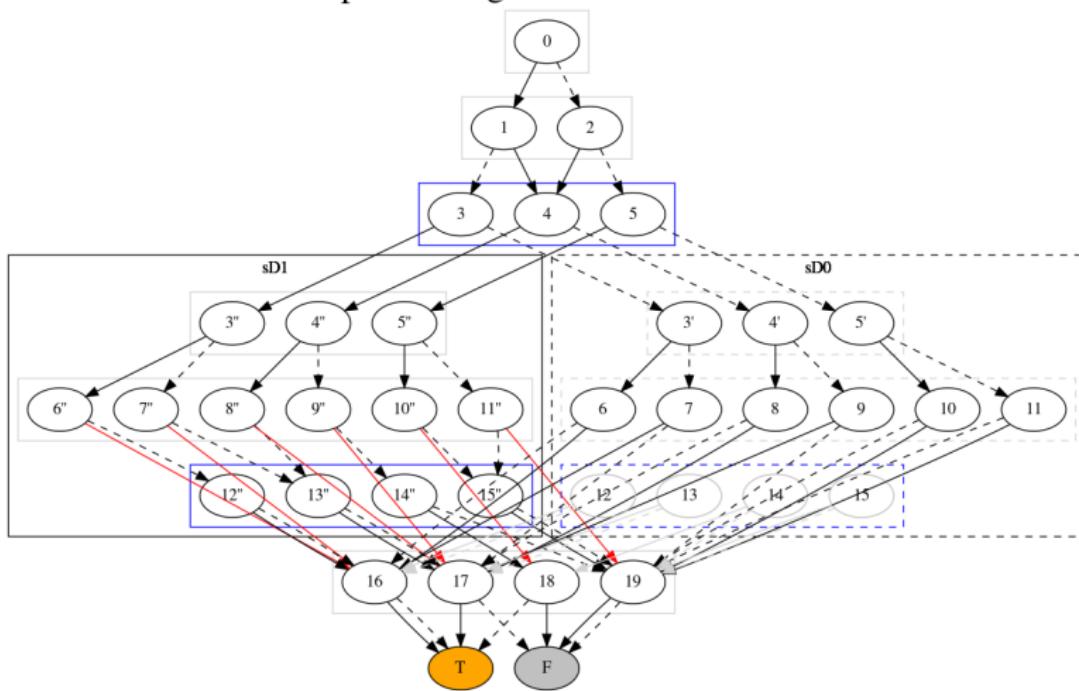
# How the BDDs are transformed?

Step 4: ... and zero-arcs for sD0



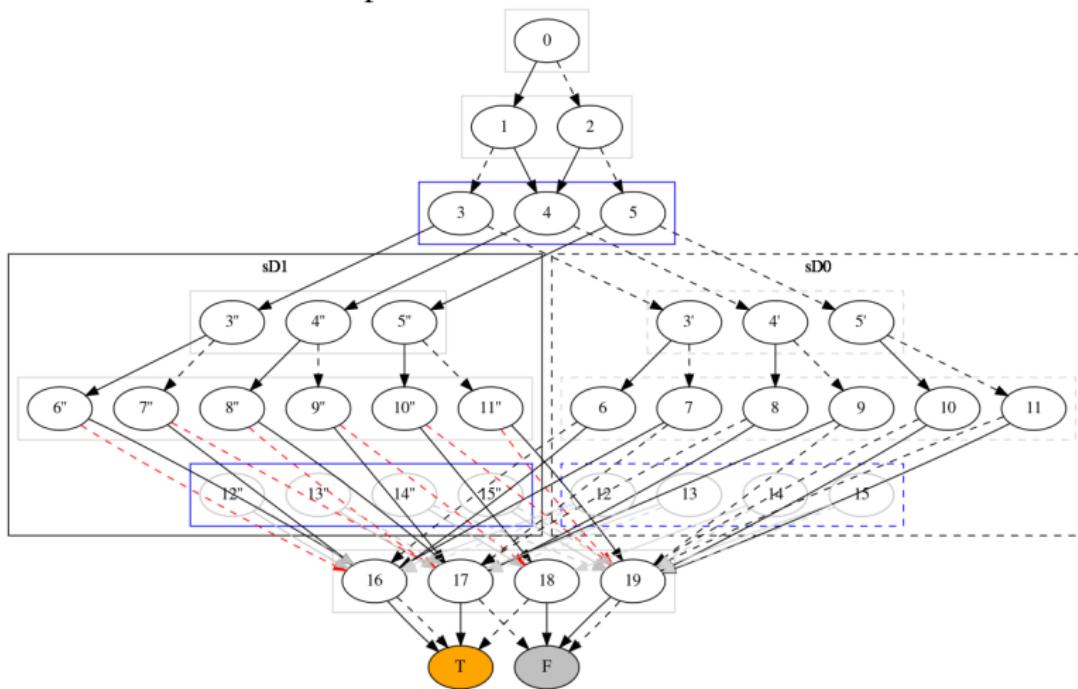
# How the BDDs are transformed?

Step 5: reassign one-arcs for sD1...



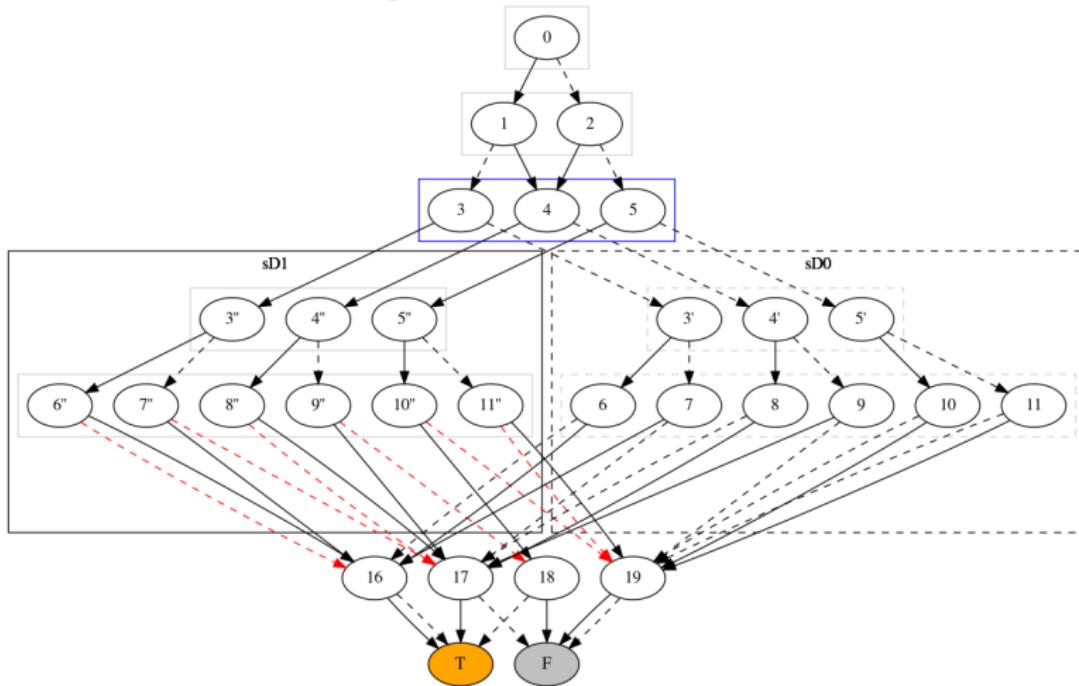
# How the BDDs are transformed?

Step 5: ... and zero-arcs for sD1



# How the BDDs are transformed?

Step 6: remove the  $L_{\{d+1\}}$



Aligning BDDs

oooooooooooo●oooooooooooo

## The simplified problem

BDDs for t-UFLP

○○○○○○○○○○

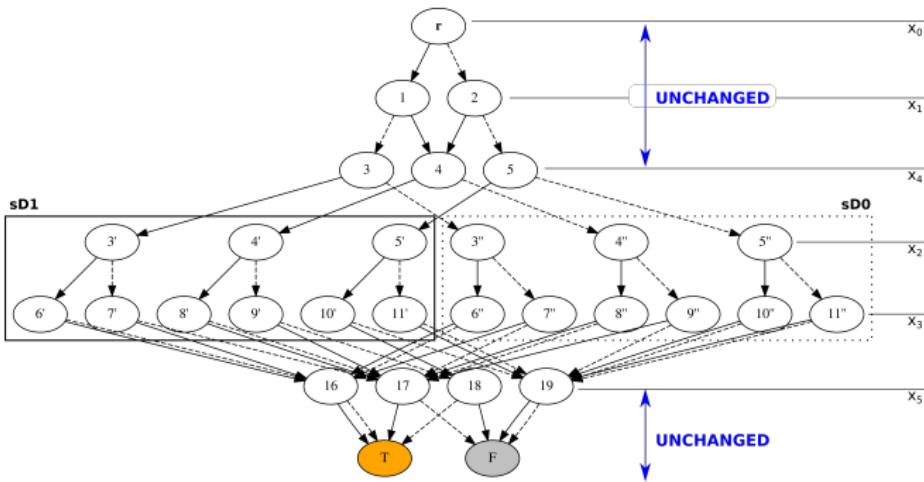
Monte Carlo Tree Search for DSPI

oooooooooooooooooooo

## Conclusion

000

## How the BDDs are transformed?



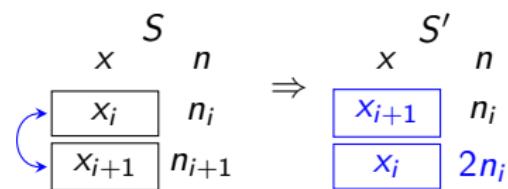
# The idea: let's just consider bounds!

Ordered lists of **labels** and the associated **weights**...

 $S$ 

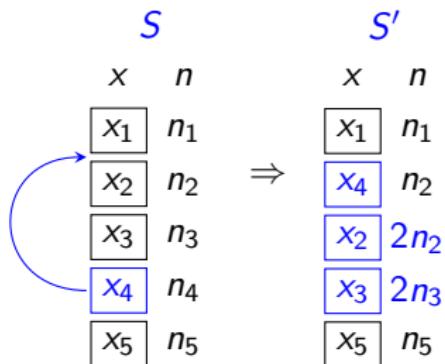
$x$	$n$
$x_1$	$n_1$
$x_2$	$n_2$
$x_3$	$n_3$
$x_4$	$n_4$
$x_5$	$n_5$

... and a **swap** operation.

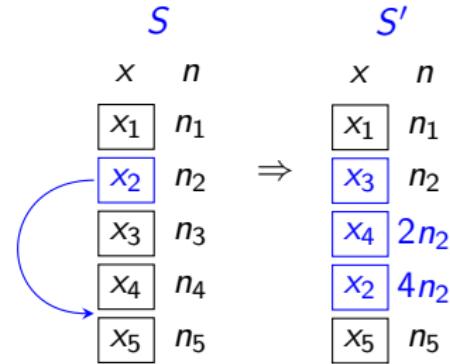


# Bounds for sift operations

A sift up:



A sift down:



# Plan of attack

**① Initial problem:**

$$\min_{\mathbf{v}} \{ |T^*[A, \mathbf{v}]| + |T^*[B; \mathbf{v}]| \}$$

**② Simplified problem:**  $A \rightarrow S_A$ ,  $B \rightarrow S_B$ , solve:

$$\min_{\mathbf{v}} \{ |T^*[S_A, \mathbf{v}]| + |T^*[S_B, \mathbf{v}]| \}$$

(We'll do branch-and-bound.)

**③ Heuristic solution:** revise  $A$  and  $B$  to the obtained  $\mathbf{v}^*$ .

# We use several properties to design a solution

Aligned pairs of elements remain aligned in an optimum.

$S_A$	$S_B$	$v^*$
...	...	...
$a$	...	$\Rightarrow$
...	$a$	
$b$	...	$b$
...	$b$	...

Corollary: it limits choices for the **last element**.

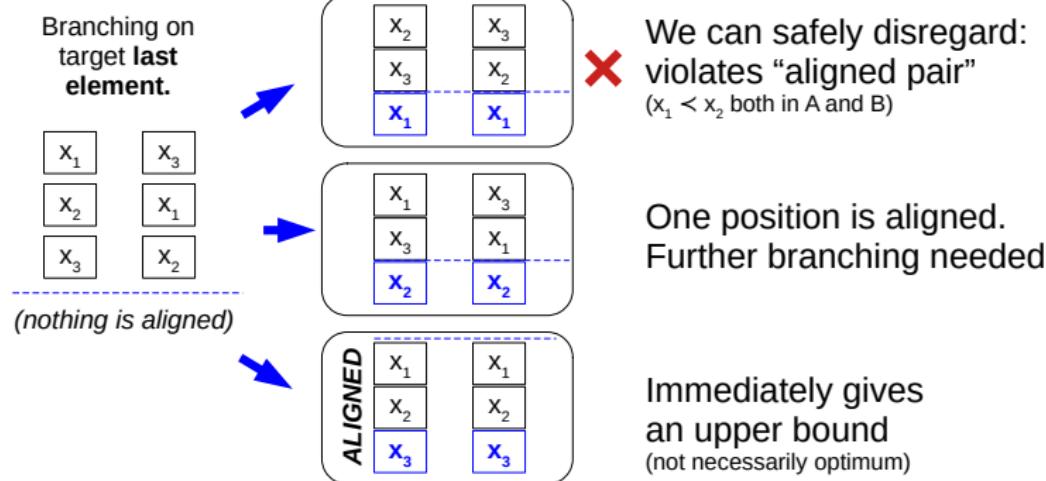
$S_A$	$S_B$
...	...
$w$	
...	
	$w$
...	...

$\leftarrow$  Disjoint  $\rightarrow$

No cost to revise an **exponentially weighted subsequence**.

$x$	$n$
$x_1$	$n_1$
$x_2$	$2n_1$
...	
$x_k$	$2^{k-1} n_1$
...	

# Branch...



... and bound.

**Upper bound:**

Any shared order would work:

- random order,
- $S_A$  or  $S_B$ ,
- “cheapest” sifts, etc.

**Lower bound:** based on the following lemma.

$S$	Target $\mathbf{v}$
$x$	$x$
$x_1$	$n_1$
$x_2$	$\dots$
$\dots$	$x_2$
$x_k$	$n_k$
$\dots$	$\Rightarrow  T^*[S, \mathbf{v}]  -  S  \geq 2n_1 - n_2.$

# A note on the lower bound

$S_A$		$S_B$		In this situation, the new size $ T^*[S_A, \mathbf{v}^*]  +  T^*[S_B, \mathbf{v}^*] $ will be at least: $ S_A  +  S_B  + \min\{2n_i^A - n_{i+1}^A, 2n_j^B - n_{j+1}^B\}.$
$x$	$n$	$x$	$n$	
...		...		
$a$	$n_i^A$	$b$	$n_j^B$	
...		...		
$b$		$a$		(done.)
...		...		

## Numerical experiments: strategy

- First: consider the simplified problem in detail.
- Then: solve the original problem, benchmark vs the baseline and consider scaling.
- Analyze the solutions: original vs. simplified, problem structure.

## Aligning BDDs

BDDs for t-UFLP

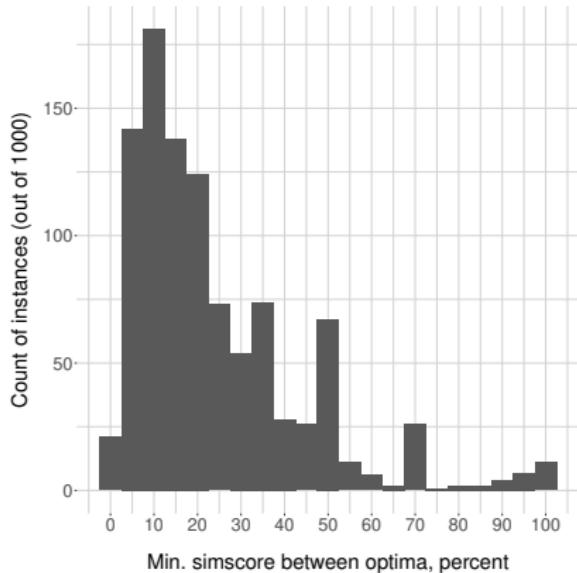
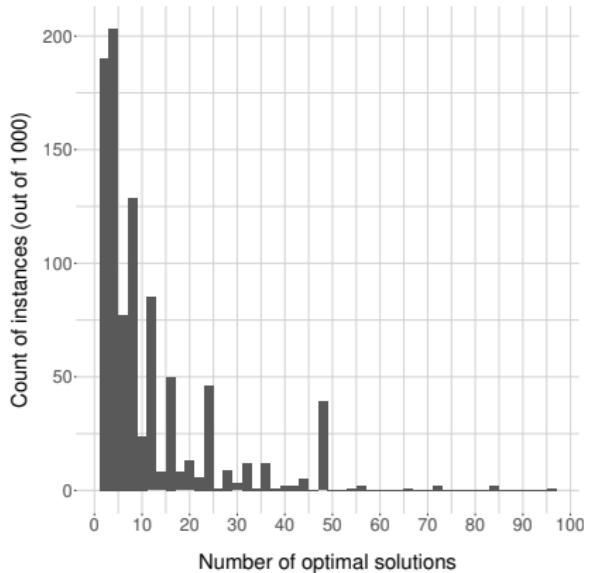
Monte Carlo Tree Search for DSPI

## Conclusion

○○○○○○○○○○

## Numerical experiments

## Simplified problem: optima



Aligning BDDs

oooooooooooooooooooo●oooo

Numerical experiments

BDDs for t-UFLP

oooooooooooo

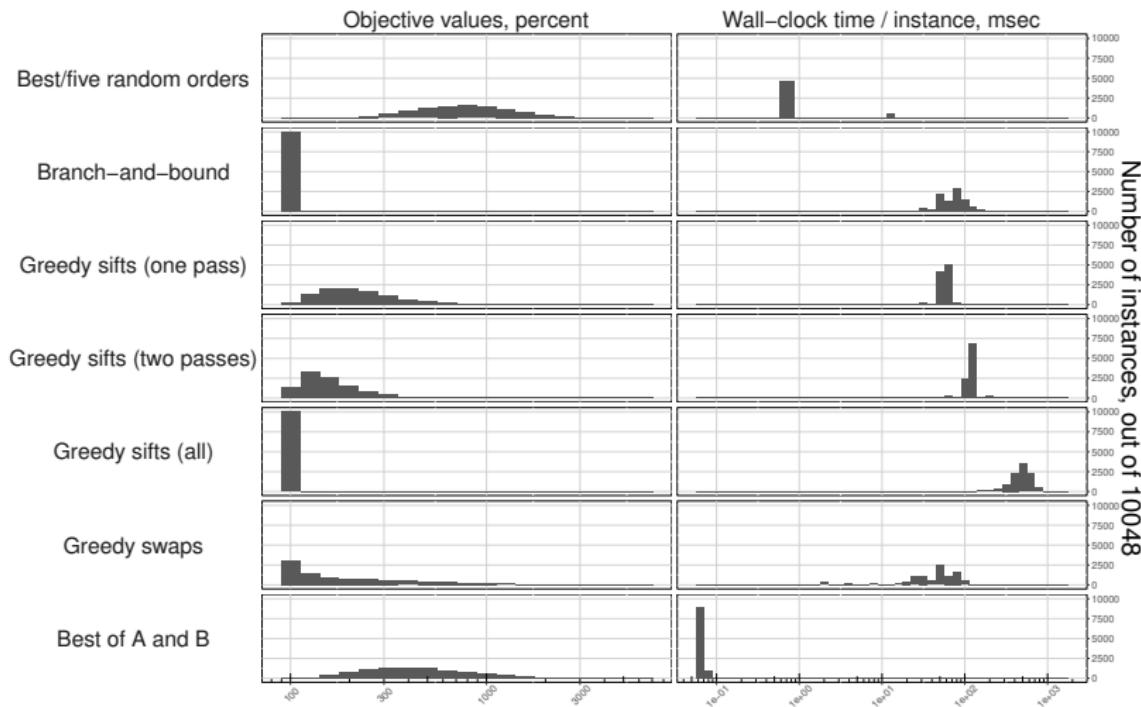
Monte Carlo Tree Search for DSPI

oooooooooooooooooooooooo

Conclusion

OOO

# Simplified problem: heuristics



Aligning BDDs

oooooooooooooooooooo●ooo

Numerical experiments

BDDs for t-UFLP

oooooooooooo

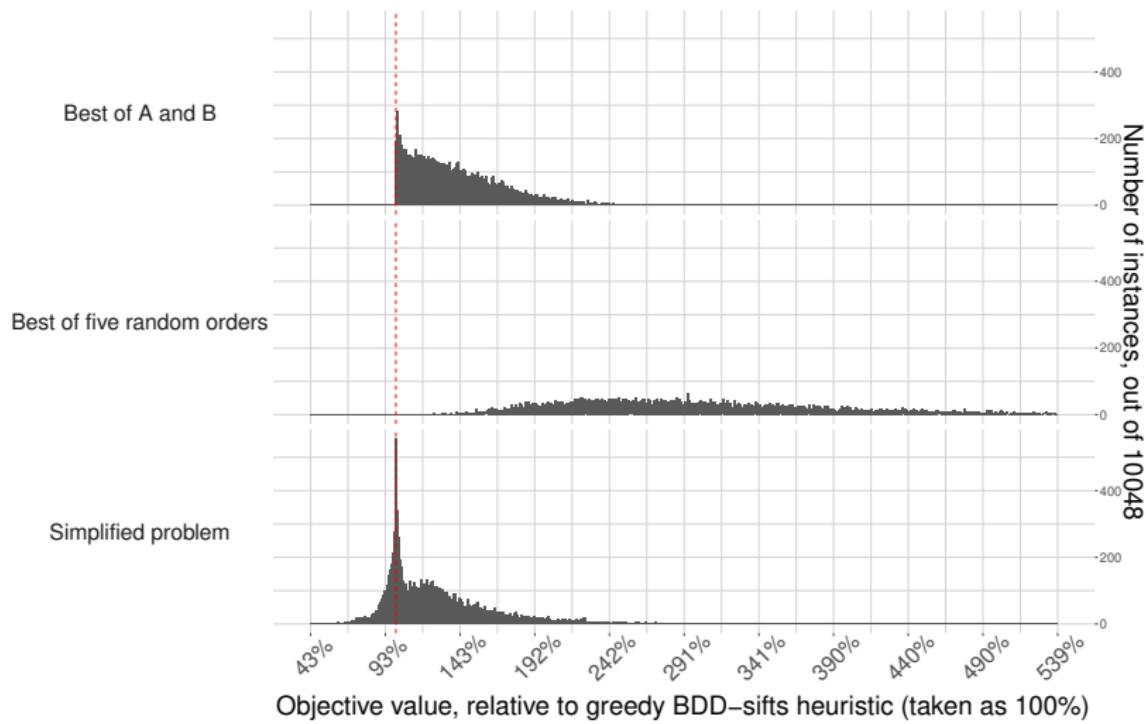
Monte Carlo Tree Search for DSPI

oooooooooooooooooooooooo

Conclusion

○○○

# Objectives relative to the “greedy sifts”



## Aligning BDDs

BDDs for t-UFLP

Monte Carlo Tree Search for DSPI

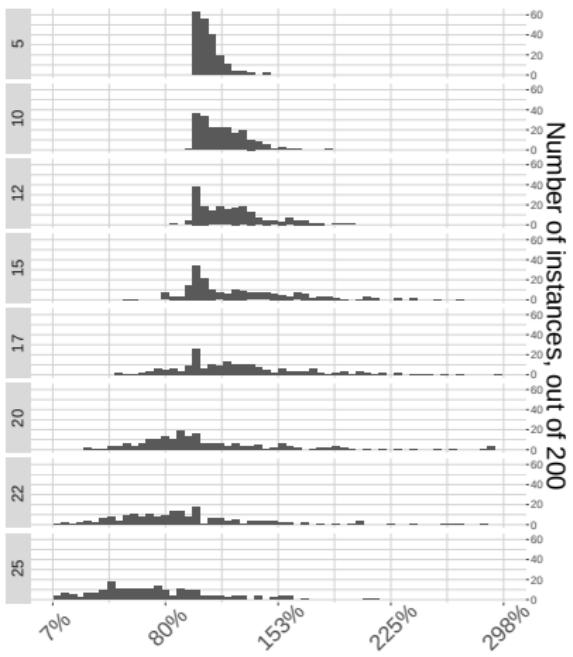
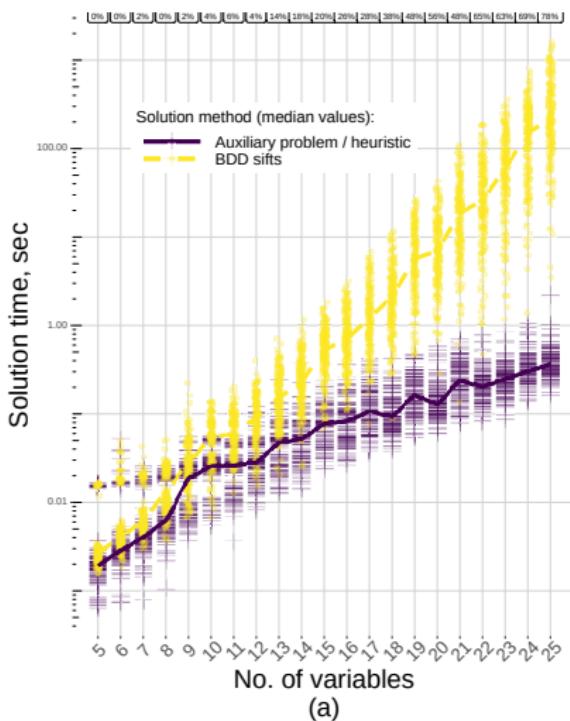
### Conclusion

○○○○○○○○○○

ooo

## Numerical experiments

## Scaling with the problem size



Aligning BDDs

oooooooooooooooooooo●○

Numerical experiments

BDDs for t-UFLP

oooooooooooo

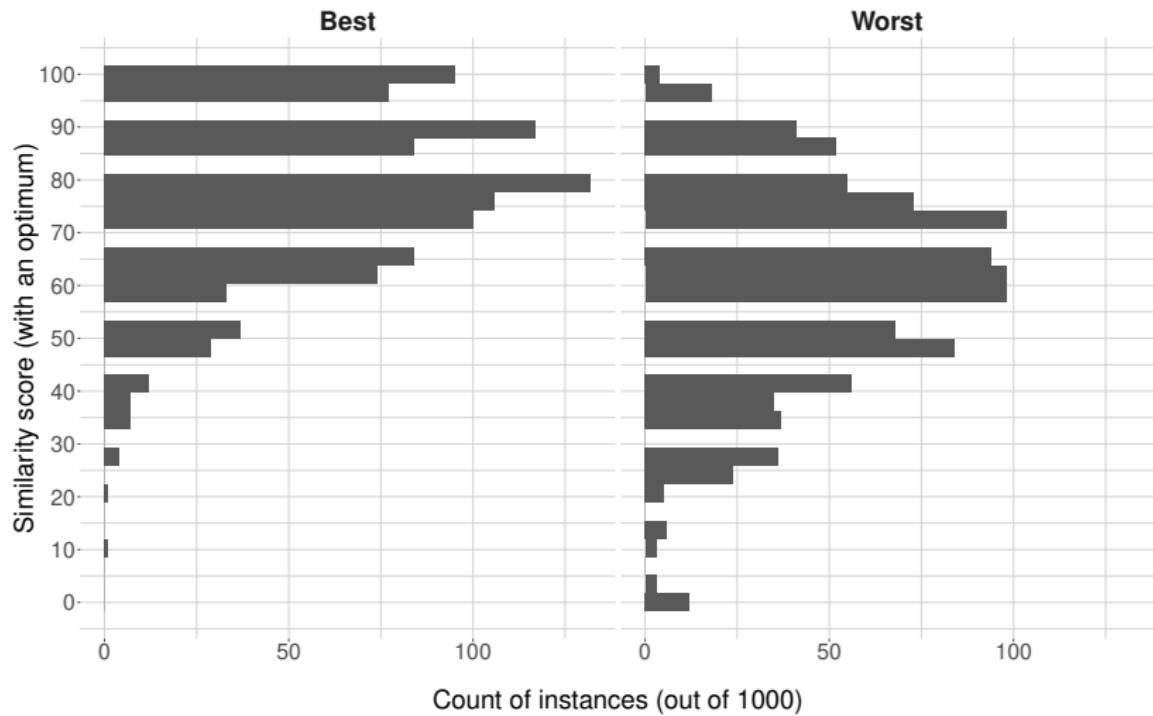
Monte Carlo Tree Search for DSPI

ooooooooooooooooooooooo

Conclusion

○○○

# Simplified vs. Original problem solutions



Aligning BDDs

oooooooooooooooooooooo●

Numerical experiments

BDDs for t-UFLP

oooooooooooo

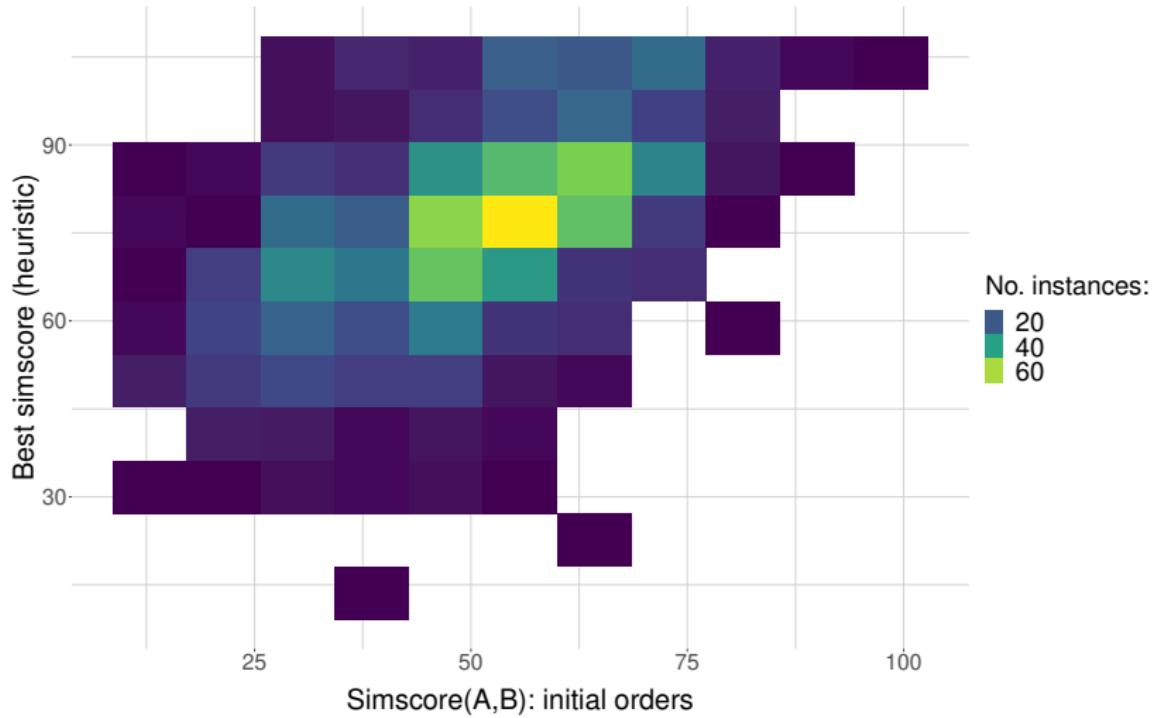
Monte Carlo Tree Search for DSPI

ooooooooooooooooooooooo

Conclusion

ooo

# Effects of the problem structure



## 1 Aligning BDDs

- On BDD representations
- BDD alignment (“original”) problem
- The simplified problem
- Numerical experiments

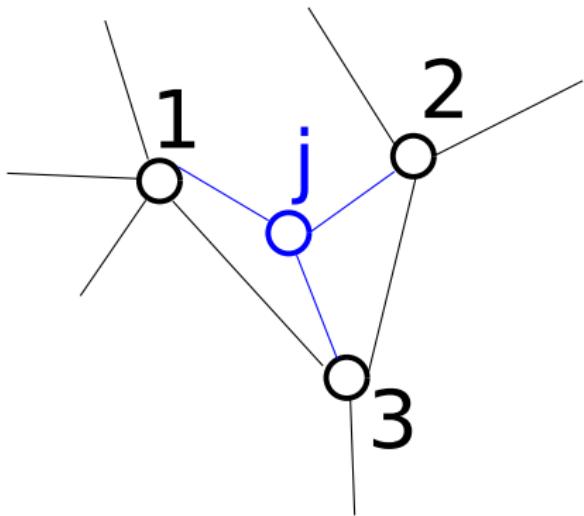
## 2 A BDD-based approach to a Facility Location Problem

- Problem description
- BDD representation
- Solving the CPP
- Numerical experiments

## 3 Monte Carlo Tree Search for DSPI

- Problem formulation
- MCTS framework
- Numerical experiments
- On correctness

# A variant of the Facility Location Problem



- Locate facilities in a cheapest way, ...
- ... to cover all points, ...
- ... respecting the budget constraints, by facility "type".

# Problem formulation

$$\begin{aligned}
 & \min \sum_{i=1}^M f_i x_i \\
 \text{s.t. } & \sum_{j \in S_i} x_j \geq 1 \quad i = 1, \dots, M, \\
 & \sum_{j \in T_t} x_j \leq k_t \quad t = 1, \dots, K, \\
 & x_i \in \{0, 1\} \quad i = 1, \dots, M.
 \end{aligned}$$

Here:

- $S_j$  – adjacency lists,
- $k_t$  – budgets,
- $f_i$  – location costs,
- $x_i$  – location decisions.

# Problem formulation

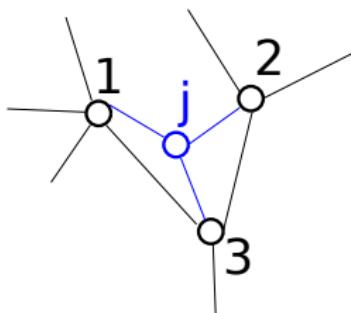
$$\begin{aligned}
 & \min \sum_{i=1}^M f_i x_i \\
 \text{s.t. } & \sum_{j \in S_i} x_j \geq 1 \quad i = 1, \dots, M, \rightarrow \text{“Cover” BDD} \\
 & \sum_{j \in T_t} x_j \leq k_t \quad t = 1, \dots, K, \rightarrow \text{“Type” BDD} \\
 & x_i \in \{0, 1\} \quad i = 1, \dots, M.
 \end{aligned}$$

Here:

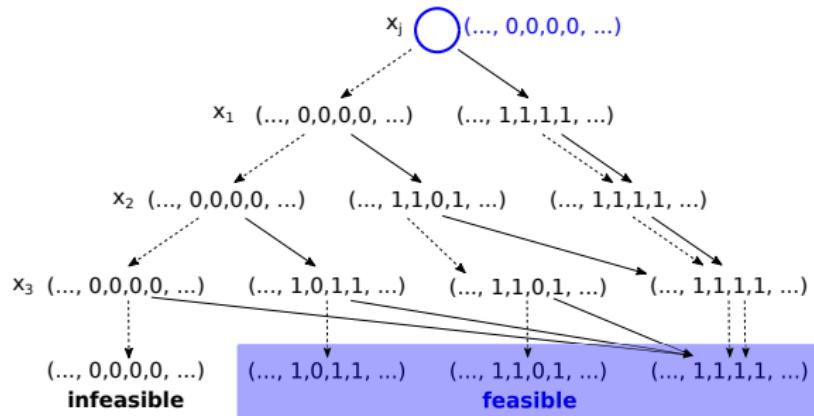
- $S_j$  – adjacency lists,
- $k_t$  – budgets,
- $f_i$  – location costs,
- $x_i$  – location decisions.

# How to build the cover BDD

Original graph:



Cover BDD:

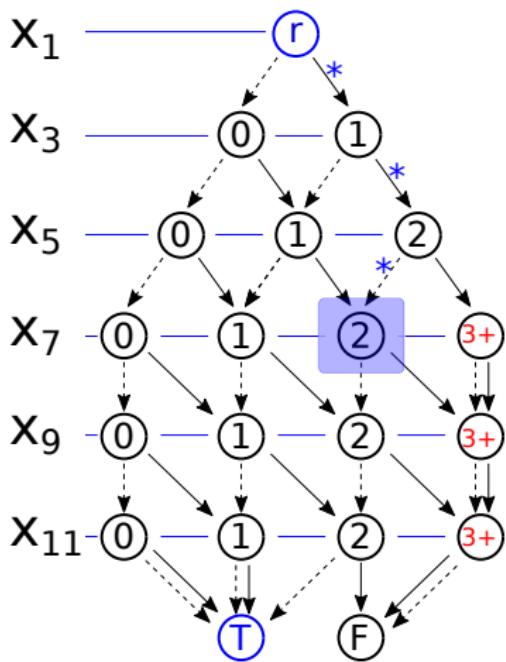


**Building the diagram:**

- Pick next point with the least “uncertainty”: # of neighbors to be added to the BDD,
- Process its adjacency list,
- Repeat.

## How to build the type BDD

## Part of Type BDD:



- Consider:  $\sum_{j \in T} x_j \leq k$  for  $T = \{1, 3, 5, 7, 9, 11\}$  and  $k = 2$ .
  - Stack such blocks vertically.
  - Minimize the number of inversions with the Cover BDD by choosing the order of variables within each block.
  - Randomize the order of blocks.

# Consistent Path Problem for t-UFLP

$$\begin{aligned}
 & \min \sum f_i v_{i, \text{HI}(i)}^C, \\
 \text{s.t.} \quad & \sum_{i: \text{HI}(i)=u \text{ or } \text{LO}(i)=u} v_{iu}^C = \sum_{j: \text{HI}(u)=j \text{ or } \text{LO}(u)=j} v_{uj}^C \text{ for all } u \in L_2^C \cup \dots \cup L_{(M-1)}^C, \\
 & \sum_{j: \text{HI}(r)=j \text{ or } \text{LO}(r)=j} v_{rj}^C = 1, \\
 & \sum_{j: \text{HI}(j)=u \text{ or } \text{LO}(j)=u} v_{ju}^C = 1 \text{ for } u \in \{\mathbf{T}^C, \mathbf{F}^C\}, \\
 & \sum_{i: \text{HI}(i)=u \text{ or } \text{LO}(i)=u} v_{iu}^T = \sum_{j: \text{HI}(u)=j \text{ or } \text{LO}(u)=j} v_{uj}^T \text{ for all } u \in L_2^T \cup \dots \cup L_{(M-1)}^T, \\
 & \sum_{j: \text{HI}(r)=j \text{ or } \text{LO}(r)=j} v_{rj}^T = 1, \\
 & \sum_{j: \text{HI}(j)=u \text{ or } \text{LO}(j)=u} v_{ju}^T = 1 \text{ for } u \in \{\mathbf{T}^T, \mathbf{F}^T\}, \\
 & \sum_{j \in L_q^C} v_{j \text{HI}(j)}^C = x_q \text{ for all } q = 1, \dots, M, \\
 & \sum_{j \in L_q^T} v_{j \text{HI}(j)}^T = x_q \text{ for all } q = 1, \dots, M, \\
 & v_{pq} \geq 0 \text{ for all valid } p, q; \quad x_i \in \{0, 1\}.
 \end{aligned}$$

# Consistent Path Problem for t-UFLP

$$\begin{aligned}
 & \min \sum f_i v_{i,\text{HI}(i)}^c, \\
 \text{s.t. } & \sum_{i:\text{HI}(i)=u \text{ or LO}(i)=u} v_{iu}^c = \sum_{j:\text{HI}(u)=j \text{ or LO}(u)=j} v_{uj}^c \text{ for all } u \in L_2^c \cup \dots \cup L_{(M-1)}^c, \\
 & \sum_{j:\text{HI}(r)=j \text{ or LO}(r)=j} v_{rj}^c = 1, \\
 & \sum_{j:\text{HI}(j)=u \text{ or LO}(j)=u} v_{ju}^c = 1 \text{ for } u \in \{\mathbf{T}^c, \mathbf{F}^c\}, \\
 & \sum_{i:\text{HI}(i)=u \text{ or LO}(i)=u} v_{iu}^t = \sum_{j:\text{HI}(u)=j \text{ or LO}(u)=j} v_{uj}^t \text{ for all } u \in L_2^t \cup \dots \cup L_{(M-1)}^t, \\
 & \sum_{j:\text{HI}(r)=j \text{ or LO}(r)=j} v_{rj}^t = 1, \\
 & \sum_{j:\text{HI}(j)=u \text{ or LO}(j)=u} v_{ju}^t = 1 \text{ for } u \in \{\mathbf{T}^t, \mathbf{F}^t\}, \\
 & \sum_{j \in L_q^c} v_{j\text{HI}(j)}^c = x_q \text{ for all } q = 1, \dots, M, \\
 & \sum_{j \in L_q^t} v_{j\text{HI}(j)}^t = x_q \text{ for all } q = 1, \dots, M, \\
 & v_{pq} \geq 0 \text{ for all valid } p, q; \quad x_i \in \{0, 1\}.
 \end{aligned}$$

# Consistent Path Problem for t-UFLP

$$\min \sum f_i v_{i,\text{HI}(i)}^c,$$

$$\text{s.t. } \sum_{i:\text{HI}(i)=u \text{ or } \text{LO}(i)=u} v_{iu}^c = \sum_{j:\text{HI}(u)=j \text{ or } \text{LO}(u)=j} v_{uj}^c \text{ for all } u \in L_2^c \cup \dots \cup L_{(M-1)}^c,$$

$$\sum_{j:\text{HI}(r)=j \text{ or } \text{LO}(r)=j} v_{rj}^c = 1,$$

$$\sum_{j:\text{HI}(j)=u \text{ or } \text{LO}(j)=u} v_{ju}^c = 1 \text{ for } u \in \{\mathbf{T}^c, \mathbf{F}^c\},$$

$$\sum_{i:\text{HI}(i)=u \text{ or } \text{LO}(i)=u} v_{iu}^t = \sum_{j:\text{HI}(u)=j \text{ or } \text{LO}(u)=j} v_{uj}^t \text{ for all } u \in L_2^t \cup \dots \cup L_{(M-1)}^t,$$

$$\sum_{j:\text{HI}(r)=j \text{ or } \text{LO}(r)=j} v_{rj}^t = 1,$$

$$\sum_{j:\text{HI}(j)=u \text{ or } \text{LO}(j)=u} v_{ju}^t = 1 \text{ for } u \in \{\mathbf{T}^t, \mathbf{F}^t\},$$

$$\sum_{j \in L_q^c} v_{j\text{HI}(j)}^c = x_q \text{ for all } q = 1, \dots, M,$$

$$\sum_{j \in L_q^t} v_{j\text{HI}(j)}^t = x_q \text{ for all } q = 1, \dots, M,$$

$$v_{pq} \geq 0 \text{ for all valid } p, q; \quad x_i \in \{0, 1\}.$$

# Consistent Path Problem for t-UFLP

$$\begin{aligned}
 & \min \sum f_i v_{i, \text{HI}(i)}^C, \\
 \text{s.t. } & \sum_{i: \text{HI}(i)=u \text{ or LO}(i)=u} v_{iu}^C = \sum_{j: \text{HI}(u)=j \text{ or LO}(u)=j} v_{uj}^C \text{ for all } u \in L_2^C \cup \dots \cup L_{(M-1)}^C, \\
 & \sum_{j: \text{HI}(r)=j \text{ or LO}(r)=j} v_{rj}^C = 1, \\
 & \sum_{j: \text{HI}(j)=u \text{ or LO}(j)=u} v_{ju}^C = 1 \text{ for } u \in \{\mathbf{T}^C, \mathbf{F}^C\}, \\
 & \sum_{i: \text{HI}(i)=u \text{ or LO}(i)=u} v_{iu}^T = \sum_{j: \text{HI}(u)=j \text{ or LO}(u)=j} v_{uj}^T \text{ for all } u \in L_2^T \cup \dots \cup L_{(M-1)}^T, \\
 & \sum_{j: \text{HI}(r)=j \text{ or LO}(r)=j} v_{rj}^T = 1, \\
 & \sum_{j: \text{HI}(j)=u \text{ or LO}(j)=u} v_{ju}^T = 1 \text{ for } u \in \{\mathbf{T}^T, \mathbf{F}^T\}, \\
 & \sum_{j \in L_q^C} v_{j \text{HI}(j)}^C = x_q \text{ for all } q = 1, \dots, M, \\
 & \sum_{j \in L_q^T} v_{j \text{HI}(j)}^T = x_q \text{ for all } q = 1, \dots, M, \\
 & v_{pq} \geq 0 \text{ for all valid } p, q; \quad x_i \in \{0, 1\}.
 \end{aligned}$$

# Consistent Path Problem for t-UFLP

$$\begin{aligned}
 & \min \sum f_i v_{i, \text{HI}(i)}^C, \\
 \text{s.t.} \quad & \sum_{i: \text{HI}(i)=u \text{ or } \text{LO}(i)=u} v_{iu}^C = \sum_{j: \text{HI}(u)=j \text{ or } \text{LO}(u)=j} v_{uj}^C \text{ for all } u \in L_2^C \cup \dots \cup L_{(M-1)}^C, \\
 & \sum_{j: \text{HI}(r)=j \text{ or } \text{LO}(r)=j} v_{rj}^C = 1, \\
 & \sum_{j: \text{HI}(j)=u \text{ or } \text{LO}(j)=u} v_{ju}^C = 1 \text{ for } u \in \{\mathbf{T}^C, \mathbf{F}^C\}, \\
 & \sum_{i: \text{HI}(i)=u \text{ or } \text{LO}(i)=u} v_{iu}^T = \sum_{j: \text{HI}(u)=j \text{ or } \text{LO}(u)=j} v_{uj}^T \text{ for all } u \in L_2^T \cup \dots \cup L_{(M-1)}^T, \\
 & \sum_{j: \text{HI}(r)=j \text{ or } \text{LO}(r)=j} v_{rj}^T = 1, \\
 & \sum_{j: \text{HI}(j)=u \text{ or } \text{LO}(j)=u} v_{ju}^T = 1 \text{ for } u \in \{\mathbf{T}^T, \mathbf{F}^T\}, \\
 & \boxed{\sum_{j \in L_q^C} v_{j, \text{HI}(j)}^C = x_q \text{ for all } q = 1, \dots, M,} \\
 & \boxed{\sum_{j \in L_q^T} v_{j, \text{HI}(j)}^T = x_q \text{ for all } q = 1, \dots, M,} \\
 & v_{pq} \geq 0 \text{ for all valid } p, q; \quad x_i \in \{0, 1\}.
 \end{aligned}$$

# Approaches to solve the problem

- Simple MIP (with  $x_i \in \{0, 1\}$  as decision variables),
- CPP + MIP (with continuous BDD flows +  $x_i$  as variables),
- CPP → Align-BDD → shortest-path (no binary variables!)

## Aligning BDDs

BDDs for t-UFLP

Monte Carlo Tree Search for DSPI

### Conclusion

00000000000000000000000000000000

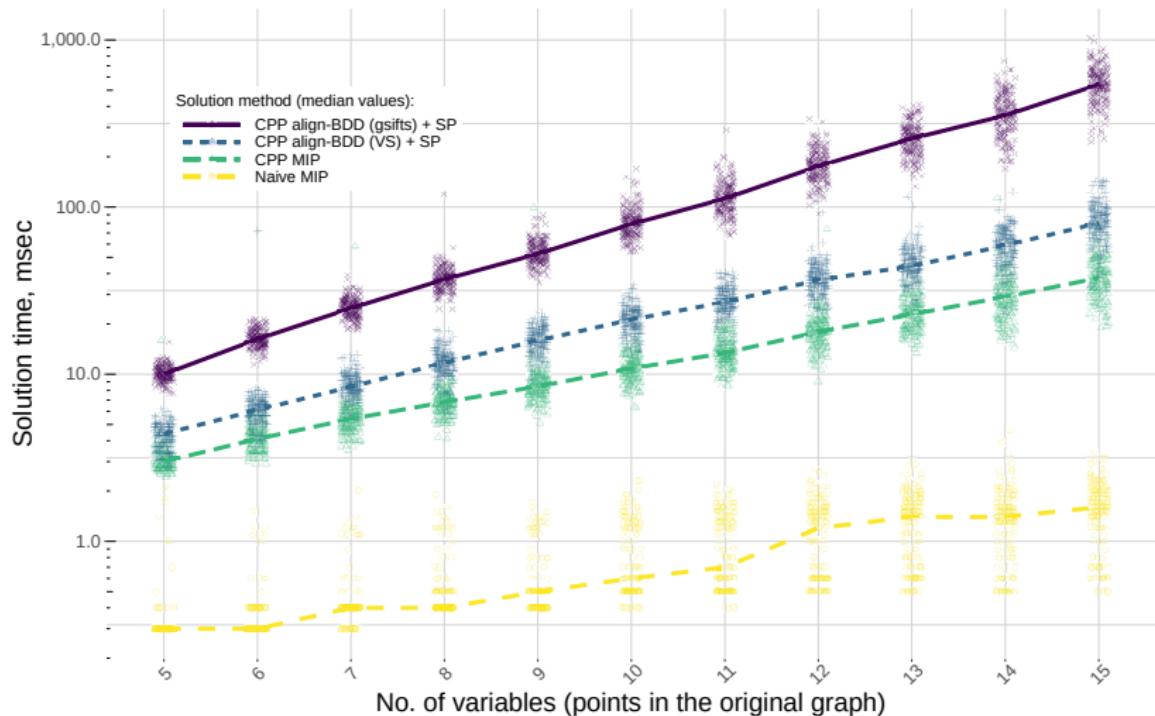
000000●000

00000000000000000000000000000000

000

## Numerical experiments

## Comparing runtimes (random instances)



Aligning BDDs

ooooooooooooooooooooooo

Numerical experiments

BDDs for t-UFLP

oooooooooooo●○○

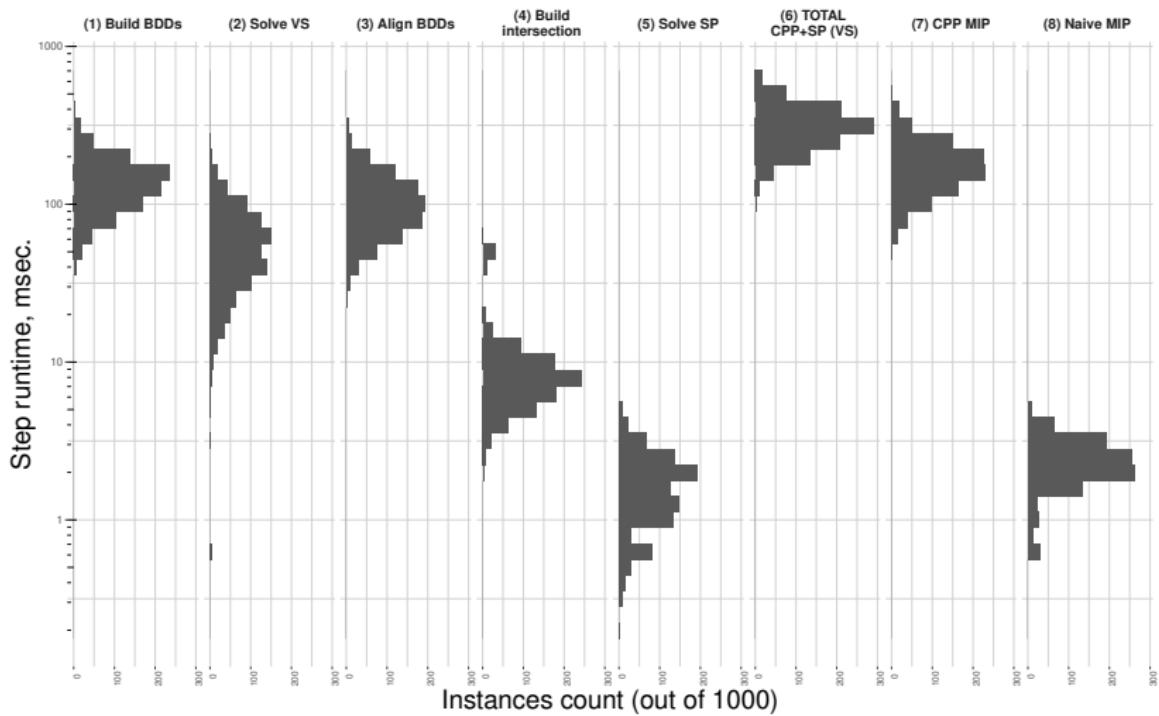
Monte Carlo Tree Search for DSPI

ooooooooooooooooooooooo

Conclusion

○○○

# Runtimes breakdown



## Aligning BDDs

BDDs for t-UFLP

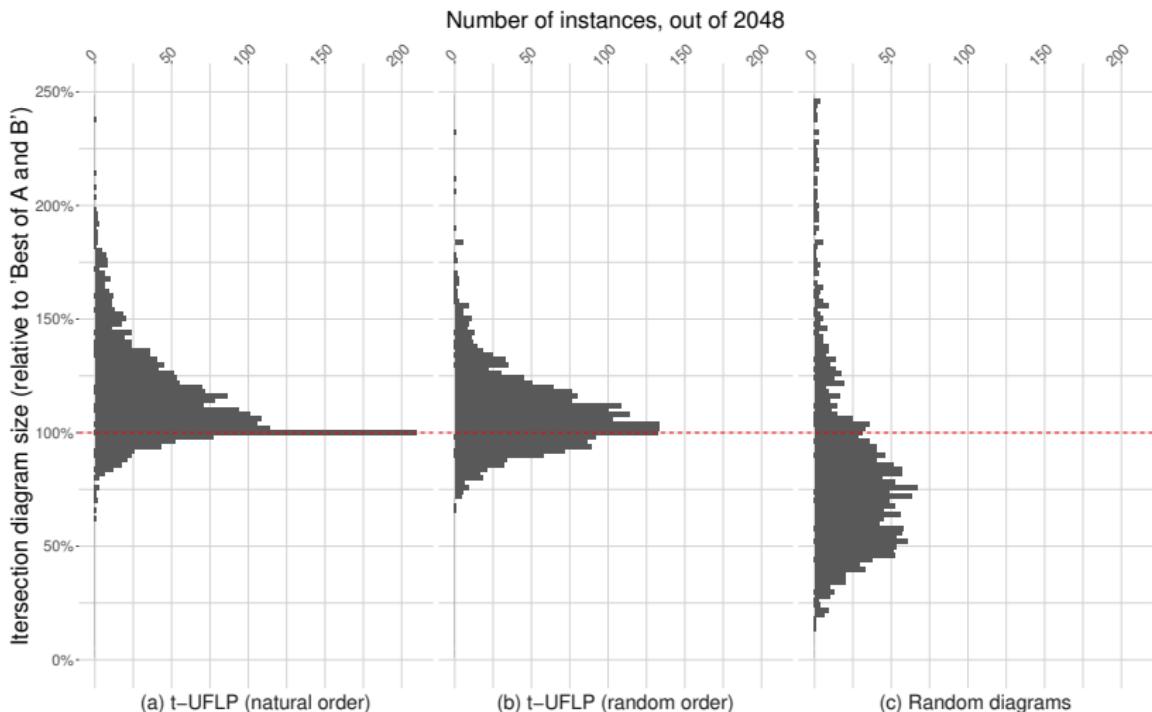
Monte Carlo Tree Search for DSPI

## Conclusion

○○○○○○○●○

## Numerical experiments

## Alignment heuristic performance vs. problem structure



# Summary on t-UFLP

- present an illustration for the Align-BDD heuristic (simplified problem),
- introduce a problem, propose a CPP reformulation,
- apply the heuristic to align the diagrams and obtain an LP,
- demonstrate the runtimes in numerical experiments,
- highlight the limits / effects of the problem structure.

## 1 Aligning BDDs

- On BDD representations
- BDD alignment (“original”) problem
- The simplified problem
- Numerical experiments

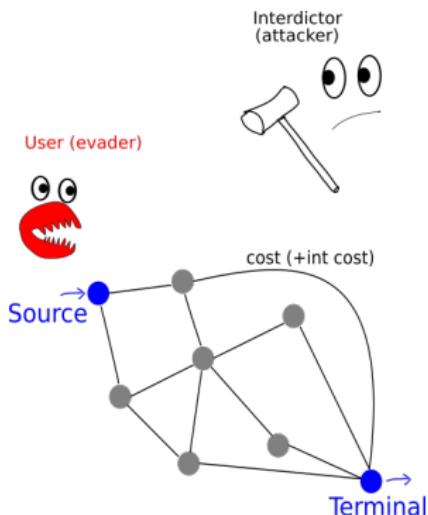
## 2 A BDD-based approach to a Facility Location Problem

- Problem description
- BDD representation
- Solving the CPP
- Numerical experiments

## 3 Monte Carlo Tree Search for DSPI

- Problem formulation
- MCTS framework
- Numerical experiments
- On correctness

# A game of “interdiction”: intro



- **Network:** a directed graph with two special nodes (source  $\circled{S}$  and terminal  $\circled{T}$ ), and a pair of "costs" associated to each edge.
- **User:** seeks to run through the graph,  $\circled{S}$  to  $\circled{T}$ , at min cost.
- **Attacker:** maximizes the User's cost by "attacking" the arcs, having a limited "budget".

We consider a **dynamic** version of the game, following Sefair & Smith 2016. (NP-hard)

# A game of “interdiction”: formulation

The Interdictor's optimal objective  $z^*$  can be expressed as:

$$z^*(S, i) = \max_{S' \subseteq \text{FS}(i) \setminus S : |S \cup S'| \leq b} \left\{ \min_{j \in \text{FS}(i)} \{z^*(S \cup S', j) + \tilde{c}_{ij}(S \cup S')\} \right\},$$

where:

- $S$ : interdiction set,
- $i$ : current Evader's node,  $\text{FS}(i)$  – forward star of node  $i$ ,
- $\tilde{c}_{ij}$ : arc traversal costs (given the interdiction),
- $b$ : Interdictor's budget.

# A game of “interdiction”: formulation

The Interdictor's optimal objective  $z^*$  can be expressed as:

$$z^*(S, i) = \max_{S' \subseteq \text{FS}(i) \setminus S : |S \cup S'| \leq b} \left\{ \min_{j \in \text{FS}(i)} \{z^*(S \cup S', j) + \tilde{c}_{ij}(S \cup S')\} \right\},$$

where:

- $S$ : interdiction set,
- $i$ : current Evader's node,  $\text{FS}(i)$  – forward star of node  $i$ ,
- $\tilde{c}_{ij}$ : arc traversal costs (given the interdiction),
- $b$ : Interdictor's budget.

## Existing algorithms (by Sefair & Smith)

- polynomial DP algorithm for a DAG
- exact DP algorithm, exp time for general case.

# Monte Carlo Tree Search

- Maintain the game tree,
- Try not to create all the nodes,
- Prune the definitely suboptimal ones,
- Drive the tree growth by a computationally cheap objective estimate (e.g., based on simulated games).

# The “game tree”

Create a “game tree”, where **nodes** contain the following information.

- Current **status**

- $p(j) \in \mathcal{N}$ : where is the Evader,
- $S_j \subseteq \mathcal{A}$ : what is interdicted,
- $\tau(j)$ : who's turn is it (Interdictor/Evader)

# The “game tree”

Create a “game tree”, where **nodes** contain the following information.

- Current **status**
  - $p(j) \in \mathcal{N}$ : where is the Evader,
  - $S_j \subseteq \mathcal{A}$ : what is interdicted,
  - $\tau(j)$ : who's turn is it (Interdictor/Evader)
- Possible further **development**
  - $\text{children}(j)$ : child game tree nodes,
  - $\text{actions}(j)$ : available actions.

# The “game tree”

Create a “game tree”, where **nodes** contain the following information.

- Current **status**
  - $p(j) \in \mathcal{N}$ : where is the Evader,
  - $S_j \subseteq \mathcal{A}$ : what is interdicted,
  - $\tau(j)$ : who's turn is it (Interdictor/Evader)
- Possible further **development**
  - $\text{children}(j)$ : child game tree nodes,
  - $\text{actions}(j)$ : available actions.
- **Costs** info, to drive the search and prune the tree
  - $\hat{Q}_j$ : cost-to-go (starting from this node),
  - $LB(j)$ ,  $UB(j)$ : bounds on the true cost-to-go,
  - $N_j \in \mathbb{N}$ : how many times the node was visited

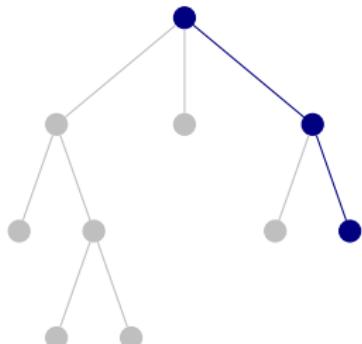
# The “game tree”

Create a “game tree”, where **nodes** contain the following information.

- Current **status**
  - $p(j) \in \mathcal{N}$ : where is the Evader,
  - $S_j \subseteq \mathcal{A}$ : what is interdicted,
  - $\tau(j)$ : who's turn is it (Interdictor/Evader)
- Possible further **development**
  - $\text{children}(j)$ : child game tree nodes,
  - $\text{actions}(j)$ : available actions.
- **Costs** info, to drive the search and prune the tree
  - $\hat{Q}_j$ : cost-to-go (starting from this node),
  - $LB(j)$ ,  $UB(j)$ : bounds on the true cost-to-go,
  - $N_j \in \mathbb{N}$ : how many times the node was visited

So, we iterate through **episodes**, each one implying a “full cycle” of the game tree update in four **phases**.

# Phase 1. Selection



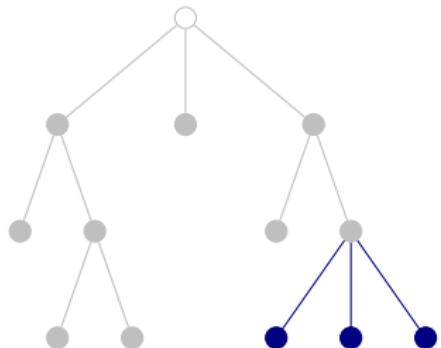
## What's happening:

- Start at the root node,
- Use *tree policy* to choose the next node recursively...
- ... pruning nodes as we go, when possible ...
- ... until we reach a leaf.

## What's updated:

- Nothing in the tree.
- Along the way: bounds for pruning (more momentarily!) + path costs.

## Phase 2. Expansion



### What's happening:

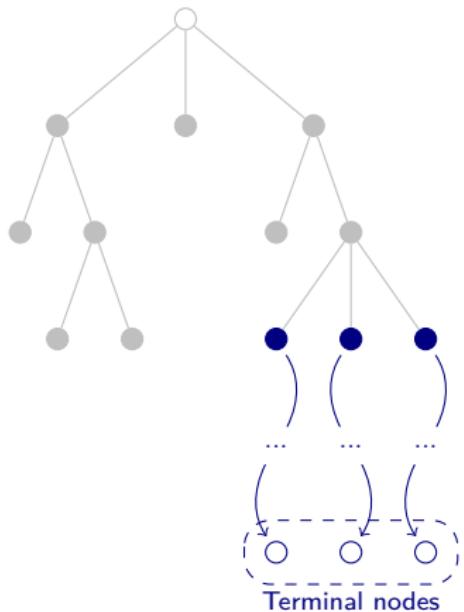
- Create child nodes for possible actions.

### What's updated:

- New nodes are created,
- UBS and LBs are calculated

**Note:** Some inconsistencies can be introduced here, between child and parent nodes.

## Phase 3. Roll-outs



### What's happening:

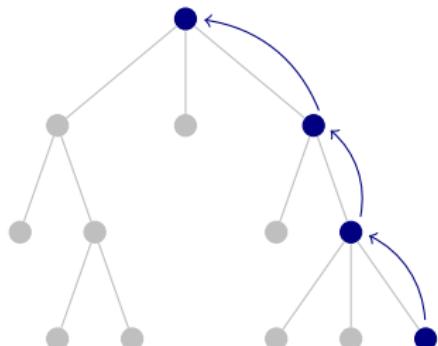
- Run a random simulated game from each node,
- Calculate cost-to-go estimate  $\hat{Q}_j$  as the simulated game cost.

### What's updated

- Cost-to-go for each new node.

**Note:** We do not record the intermediate game states occurred during roll-outs!

## Phase 4. Backpropagation



### What's happening:

- Start at the selected node,
- Recursively update (“propagate”) node information for parents ...
- ... until we reach the root.

**What's updated:** Information in each parent node, using the child nodes:

- UBS and LBS
- Cost-to-go estimate: the best value (given the turn).

# The Algorithm

The algorithm can perform actions for both players. Each turn involves two steps:

## **Step 1. THINK.**

We iteratively improve the tree (while we have budget):

**FOR**  $k = 1, \dots, K$  **DO**

- Selection
- Expansion
- Roll-outs
- Backpropagation

**END.**

## **Step 2. ACT.**

... then pick an action corresponding to the “most attractive” child node of the root.

Aligning BDDs

ooooooooooooooooooooooo

MCTS framework

BDDs for t-UFLP

oooooooooooo

Monte Carlo Tree Search for DSPI

oooooooooooo●oooooooooooo

Conclusion

ooo

# There are several secret ingredients



# SI-1. How to select?

- **with probability**  $\varepsilon$ : choose at random;
- **otherwise**, a child node with the **best score**:

$$R_j = \underbrace{\sigma_i(\tilde{C}_{ij} + \hat{Q}_j)}_{\text{best cost-to-go}} + \underbrace{C_p \sqrt{\log(N_i)/N_j}}_{\text{encourage exploration}}, \quad \text{for all } j \in \text{children}(i)$$

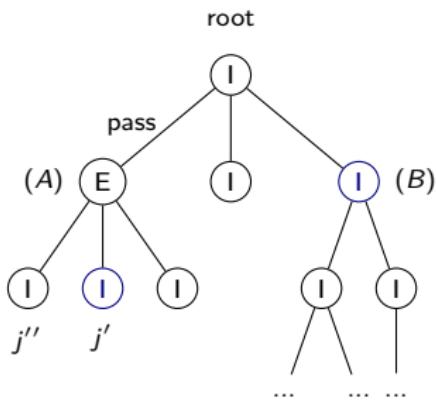
“Best” here depends on the turn (the Evader will choose the smallest cost estimate, the Interdictor — the largest).

## SI-2. How to prune?

We leverage the classic idea of **alpha–beta pruning**:

Maintain two running numbers (bounds):

- $\alpha$ : the worst (minimum) alternative cost achievable by the Interdictor,
- $\beta$ : the worst (maximum) alternative cost achievable by the Evader.



**Pruning condition:**  $\beta \leq \hat{\alpha}_j$  or  $\hat{\beta}_j \leq \alpha$ , where

- $\hat{\alpha}_j = \pi_n + LB(j)$  (Interdictor's turns), and
- $\hat{\beta}_j = \pi_n + \widetilde{C}_{nj} + UB(j)$  (Evader's turns)

## SI-3. How to back-propagate?

Assuming the Evader's turn, and  $i$  being the current game tree node:

- Update the bounds:

$$UB(i) \leftarrow \min_{j \in \text{children}(i)} \left\{ \tilde{C}_{ij}(S_i) + UB(j) \right\},$$

$$LB(i) \leftarrow \min_{j \in \text{children}(i)} \left\{ \tilde{C}_{ij}(S_i) + LB(j) \right\}.$$

- Update the cost-to-go estimate:

$$\hat{Q}_i \leftarrow \min_{j \in \text{children}(i)} \left\{ \tilde{C}_{ij}(S_i) + \hat{Q}_j \right\}.$$

## SI-4. Bounds

Are inspired by Sefair & Smith 2016:

- **Lower bound:** restrict the **Attacker**. Make attacks “expire” after a single turn).
- **Upper bound:** restrict the **Interdictor**. Remove some arcs so that  $G$  is a DAG ( $\Rightarrow$  DSPI is simple to solve.) Repeat multiple times and choose the best bound.

# Numerical experiments: strategy

- How does it perform on pre-defined instances? (relative to the known optimum, and to the bounds)
- How does it perform on randomly generated instances with different budgets?
- What's the dynamics of the tree construction? How does the algorithm work?
- What's the point of “playing out”, i.e., changing root nodes?

Aligning BDDs

ooooooooooooooooooooooo

Numerical experiments

BDDs for t-UFLP

oooooooooooo

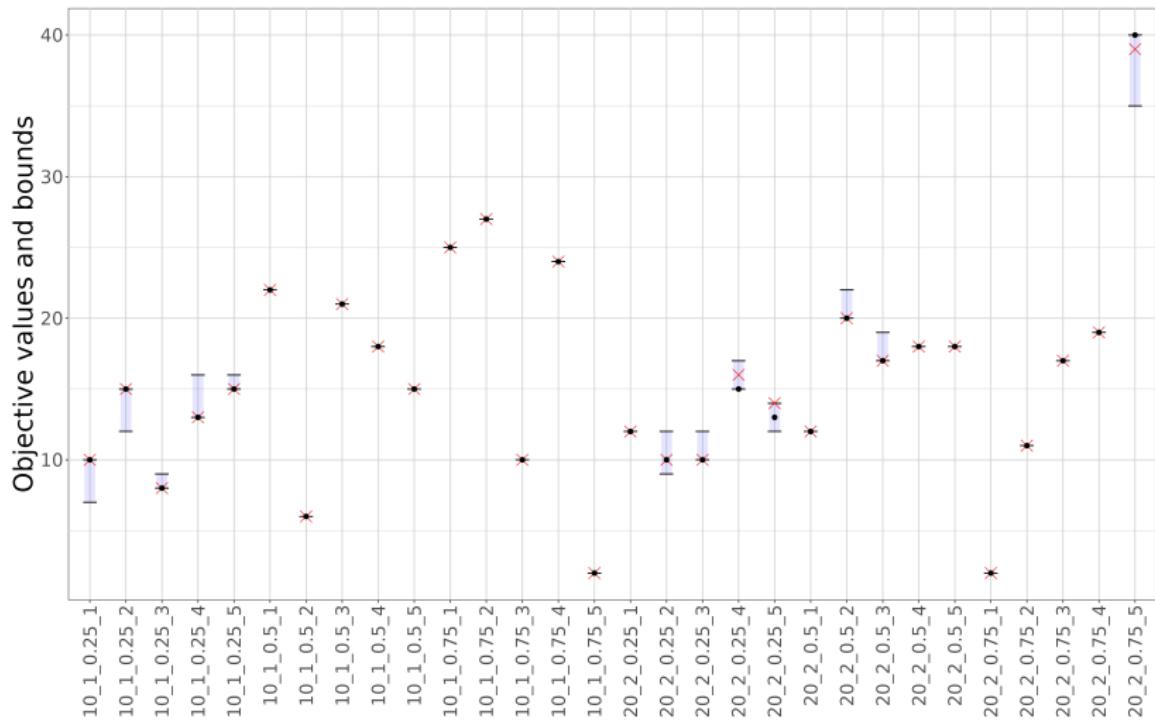
Monte Carlo Tree Search for DSPI

oooooooooooooooooooo

Conclusion

ooo

# Pre-defined instances



Aligning BDDs

ooooooooooooooooooooooo

Numerical experiments

BDDs for t-UFLP

oooooooooooo

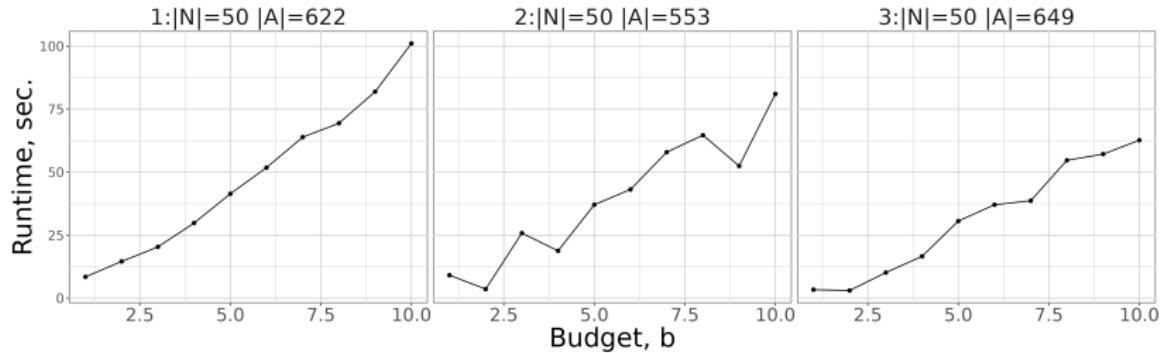
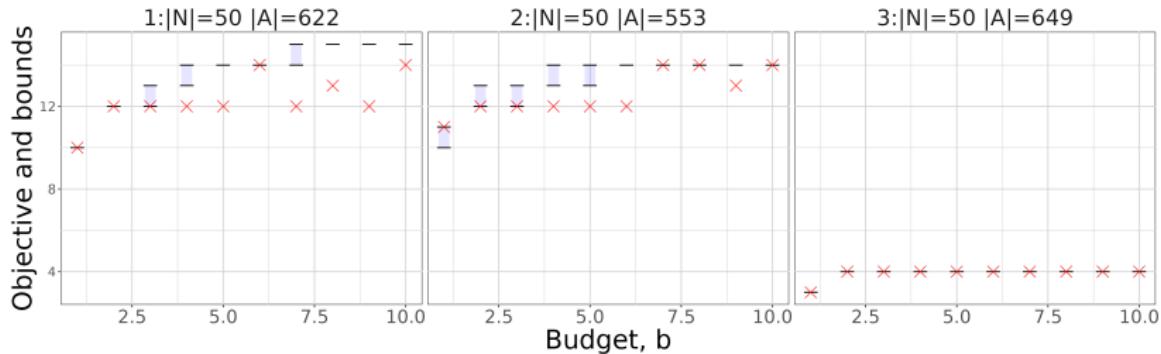
Monte Carlo Tree Search for DSPI

oooooooooooooooooooo●oooo

Conclusion

ooo

# Random instances (snapshot)



Aligning BDDs

ooooooooooooooooooooooo

Numerical experiments

BDDs for t-UFLP

oooooooooooo

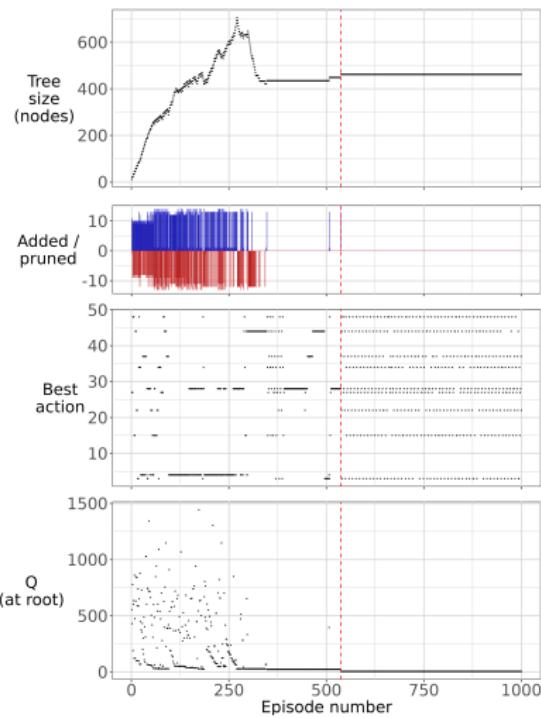
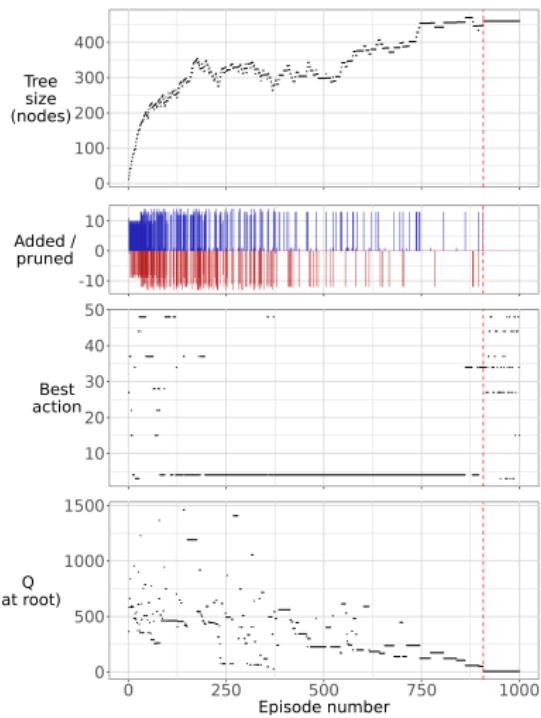
Monte Carlo Tree Search for DSPI

oooooooooooooooooooo●○○○○

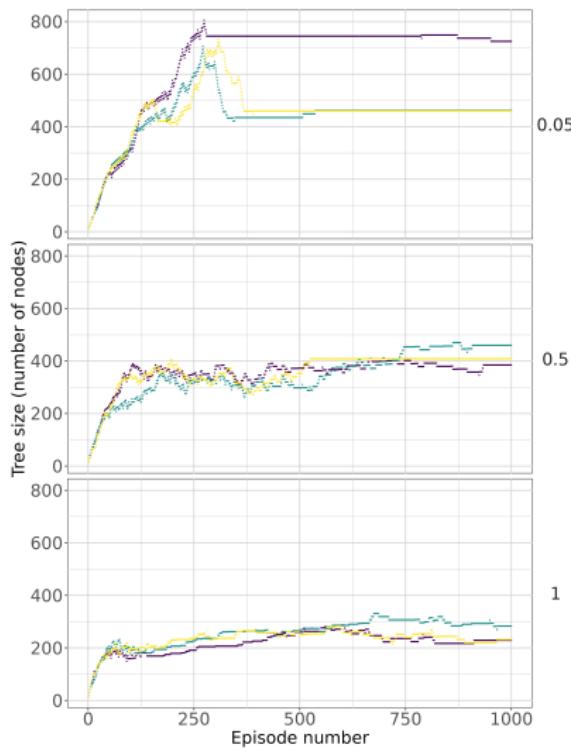
Conclusion

○○○

# Convergence profiles

 $\varepsilon = 0.05$  $\varepsilon = 0.5$ 

# A remark: that's not just different runs



For each value of  $\varepsilon$  (0.05, 0.5, and 1) we performed three runs, to confirm these are indeed different “modes” of the algorithm.

Aligning BDDs

oooooooooooooooooooo

Numerical experiments

BDDs for t-UFLP

oooooooooooo

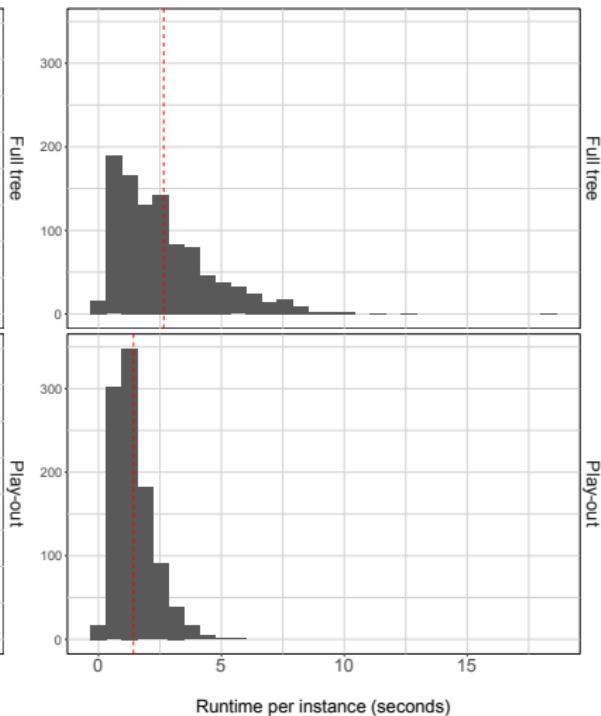
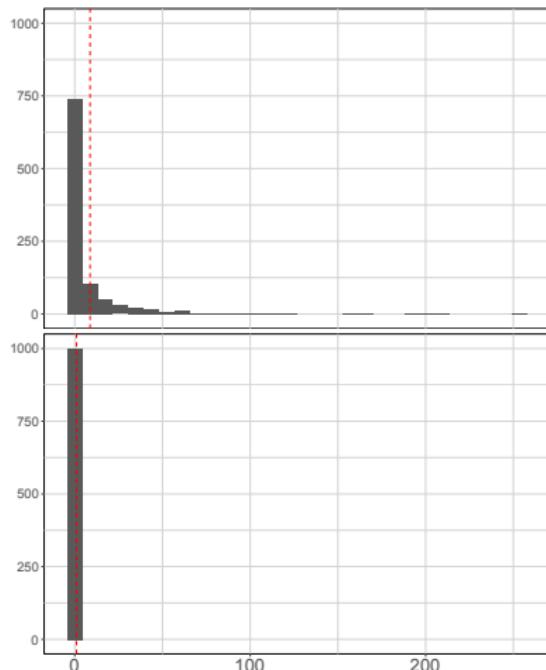
Monte Carlo Tree Search for DSPI

oooooooooooooooooooo●○○

Conclusion

○○○

# Play-out vs. first-move strategy



# Why does it work? (A sketch on “correctness”)

## Theorem (Proposition)

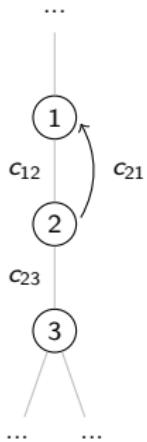
$$\lim_{k \rightarrow \infty} \mathbb{P}\{Q_{root}^k = \text{true optimum}\} = 1$$

A sketch of the proof:

- The game tree has finite number of nodes (there is a bound independent from  $K$ ).
- We never cut off all the optima  $\Rightarrow$  the tree contains at least one.
- What is left is a finite-size minimax tree, containing an optimum.
- As  $K \rightarrow \infty$ , probability to select every node for expansion converges to 1.

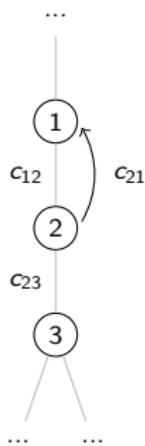
# Why in the world the tree is finite?

Network:

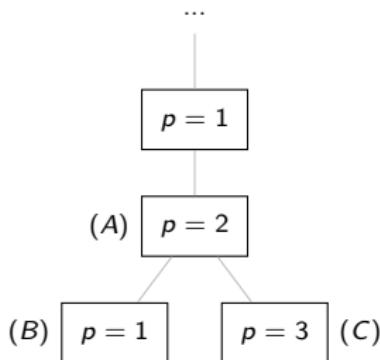


# Why in the world the tree is finite?

Network:



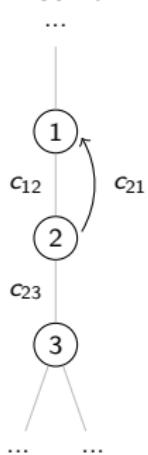
After the **first**  
expansion:



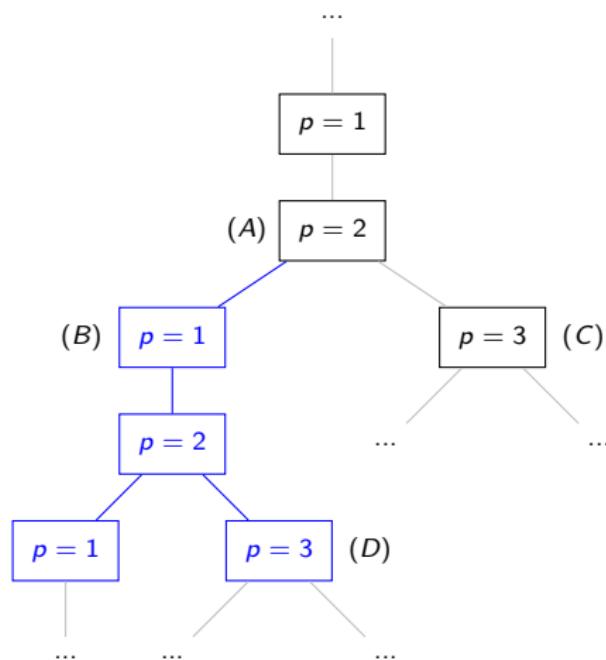
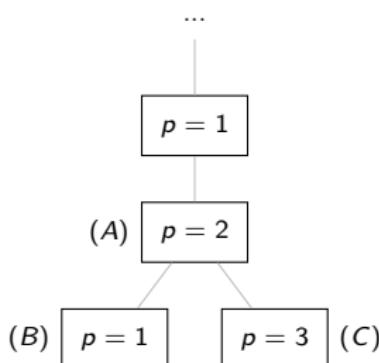
# Why in the world the tree is finite?

After the **second** expansion:

Network:



After the **first** expansion:



# Summary

## 1 Aligning BDDs

- On BDD representations
- BDD alignment (“original”) problem
- The simplified problem
- Numerical experiments

## 2 A BDD-based approach to a Facility Location Problem

- Problem description
- BDD representation
- Solving the CPP
- Numerical experiments

## 3 Monte Carlo Tree Search for DSPI

- Problem formulation
- MCTS framework
- Numerical experiments
- On correctness

# Future research

- BDDs and related topics.
  - Applications, **relaxed** and **restricted** BDDs.
  - Other types of diagrams
  - Perhaps, improving the simplified problem.
  - Separately: perhaps, parallel algorithms?
- MCTS and RL in general.
  - Transposition tables,
  - DNN / GNN for tree and roll-out policies.
  - Especially – separately – learning algorithms?
  - Other possible applications for MCTS (other variants of interdiction? SP?)

# Your comments / questions are very welcome!



Also: I'm always available via email

- a@bochkarev.io
- abochka@g.clemson.edu

[www.bochkarev.io/contact](http://www.bochkarev.io/contact)