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

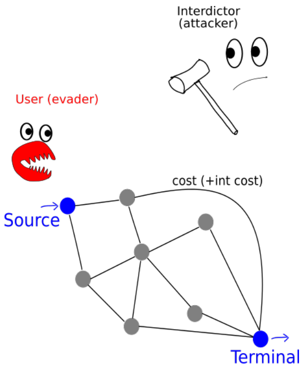
Dissertation defense

Alexey Bochkarev

2021-11-02, 08-30, Zoom / Freeman Hall 123

Outline

A game of “interdiction”: intro



- Network: a directed graph with two special nodes (source ⓢ and terminal ⓣ), and a pair of "costs" associated to each edge.
- User: seeks to run through the graph, ⓢ to ⓣ, 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 and Smith(2016)]. (NP-hard)

A game of "interdiction": formulation

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

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

where:

- S: interdiction set,
- i: current Evader's node, $\mathrm{FS}(i)$ – forward star of node i,
- $\widetilde{c}_{ij}$: 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:

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

where:

- S: interdiction set,
- i: current Evader's node, $\mathrm{FS}(i)$ – forward star of node i,
- $\widetilde{c}_{ij}$: 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
  - $\mathrm{p}(j) \in \mathcal{N}$: 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
  - $\mathrm{p}(j) \in \mathcal{N}$: where is the Evader,
  - $S_j \subseteq \mathcal{A}$: what is interdicted,
  - $\tau(j)$: who’s turn is it (Interdictor/Evader)
- Possible further development
  - $\texttt{children}(j)$: child game tree nodes,
  - $\texttt{actions}(j)$: available actions.

The "game tree"

Create a "game tree", where nodes contain the following information.

- Current status
  - $\mathrm{p}(j) \in \mathcal{N}$: where is the Evader,
  - $S_j \subseteq \mathcal{A}$: what is interdicted,
  - $\tau(j)$: who's turn is it (Interdictor/Evader)
- Possible further development
  - $\texttt{children}(j)$: child game tree nodes,
  - $\texttt{actions}(j)$: available actions.
- Costs info, to drive the search and prune the tree
  - $\widehat{Q}_j$: 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

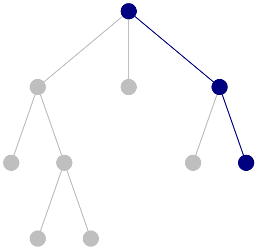
The “game tree”

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

- Current status
  - $\mathrm{p}(j) \in \mathcal{N}$: where is the Evader,
  - $S_j \subseteq \mathcal{A}$: what is interdicted,
  - $\tau(j)$: who’s turn is it (Interdictor/Evader)
- Possible further development
  - $\texttt{children}(j)$: child game tree nodes,
  - $\texttt{actions}(j)$: available actions.
- Costs info, to drive the search and prune the tree
  - $\widehat{Q}_j$: 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



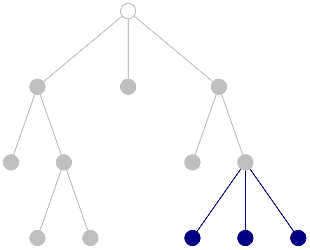
What's happening:

- 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:

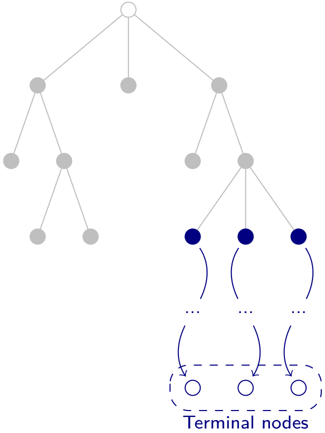
- Create child nodes for possible actions.

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:

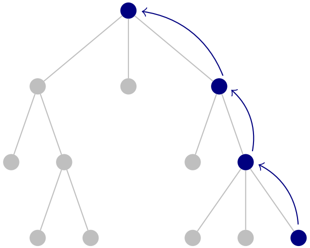
- Run a random simulated game from each node,
- Calculate cost-to-go estimate $\widehat{Q}_j$ as the simulated game cost.

What's updated

- Cost-to-go for each new node.

Note: We do not record the intermediate game states occured 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 $k = 1, \ldots, K$ **DO**

- Selection
- Expansion
- Roll-outs
- Backpropagation

END.

Step 2. ACT.

$\ldots$ then pick an action corresponding to the "most attractive" child node of the root.

There are several secret ingredients

SI-1. How to select?

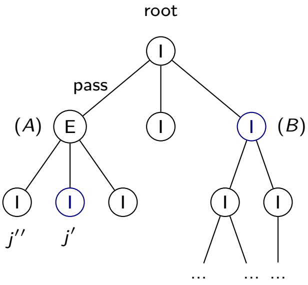
- **with probability** ε: choose at random;
- **otherwise**, a child node with the best score:

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

“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):

- α: the worst (minimum) alternative cost achievable by the Interdictor,
- β: the worst (maximum) alternative cost achievable by the Evader.

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

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

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

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

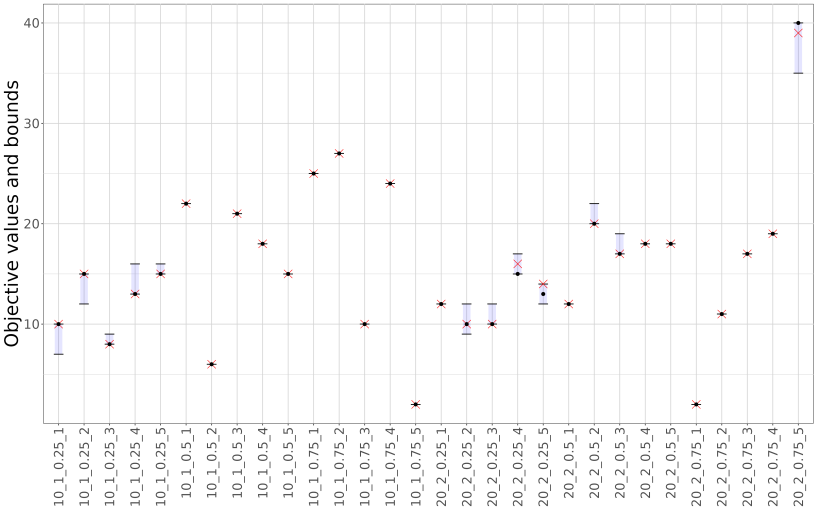
- Update the cost-to-go estimate:

$$\widehat{Q}_i \leftarrow \min_{j \in \texttt{children}(i)} \left\{ \widetilde{C}_{ij}(S_i) + \widehat{Q}_j \right\}.$$

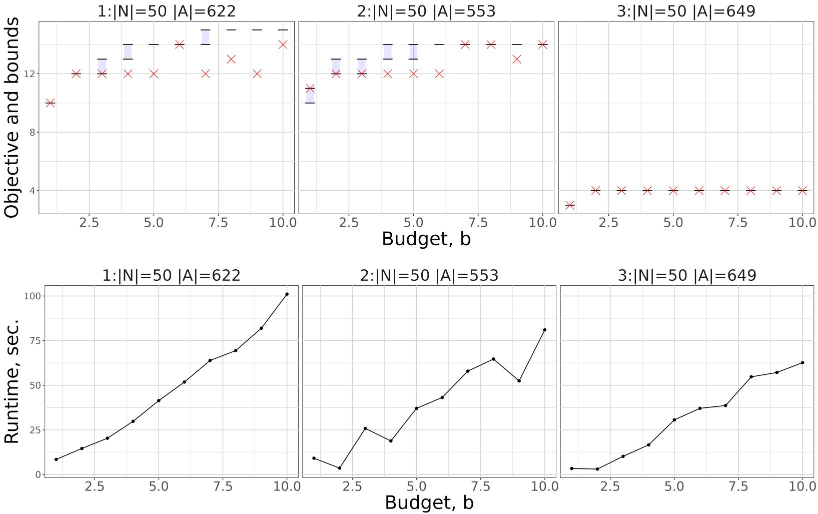
Numerical experiments: strategy

- 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?

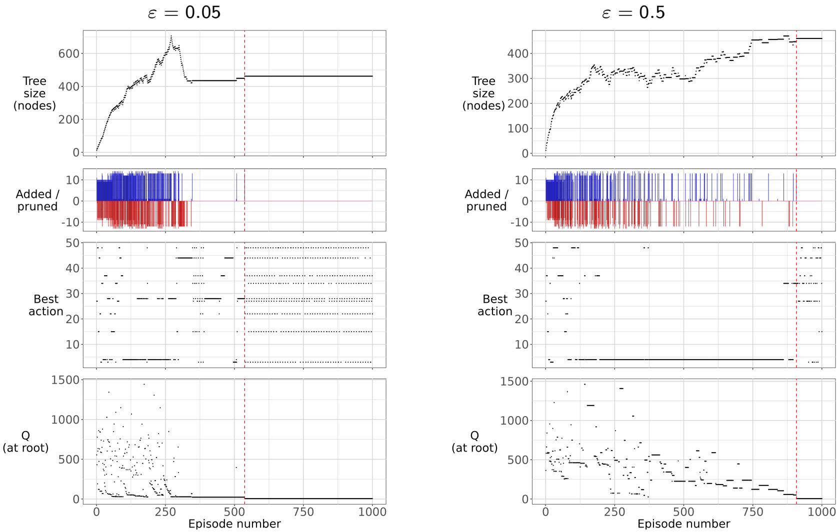
Pre-defined instances



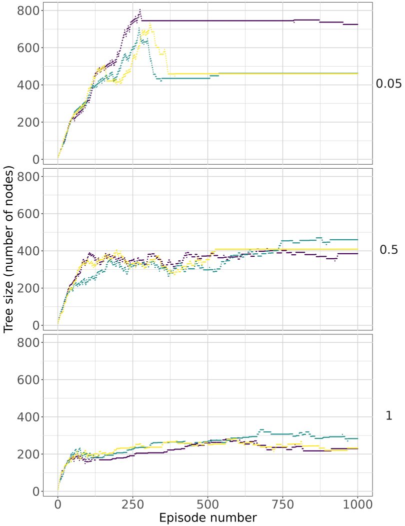
Random instances (snapshot)



Convergence profiles

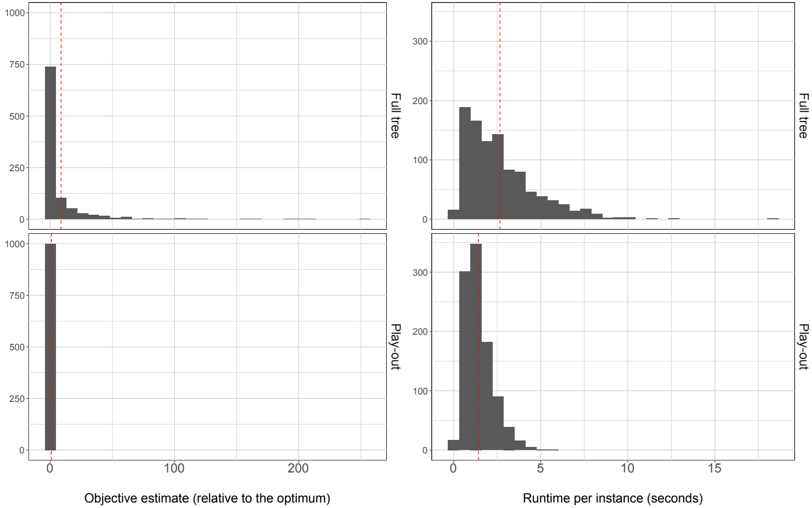


A remark: that's not just different runs



For each value of ε (0.05, 0.5, and 1) we performed three runs, to confirm these are indeed different "modes" of the algorithm.

Play-out vs. first-move strategy



Why does it work? (A sketch on "correctness")

Theorem (Proposition)

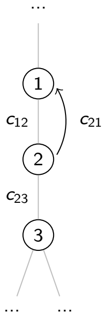
$$\lim_{k \to \infty} \mathbb{P}\{Q^k_{root} = \textit{true optimum}\} = 1$$

A sketch of the proof:

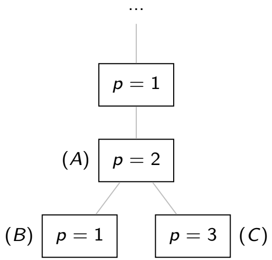
- The game tree has finite number of nodes (there is a bound independent from K).
- We never cut off all the optima $\Rightarrow$ the tree contains at least one.
- What is left is a finite-size minimax tree, containing an optimum.
- As $K \to \infty$, probability to select every node for expansion converges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

Why in the world the tree is finite?

Network:

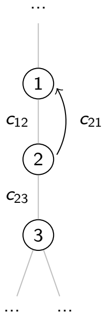


After the first expansion:

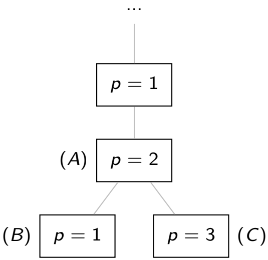


Why in the world the tree is finite?

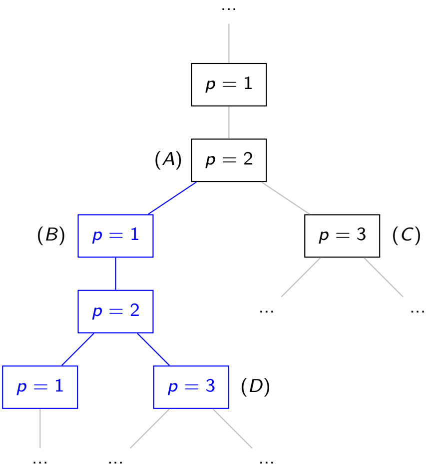
Network:



After the first expansion:



After the second expansion:



Mentioned sources

Jorge A. Sefair and J. Cole Smith.
Dynamic shortest-path interdiction.
Networks, 68(4):315–330, 2016.
ISSN 1097-0037.
doi: 10.1002/net.21712.