Computing Abstract Plans for Counterexample-Guided Cartesian Abstraction Refinement

Samuel von Allmen

2019-05-21

The Big Picture

What are we trying to accomplish?

- Solve planning problems with informed search algorithms.
- Informed search needs a heuristic (h).
- How to get a good heuristic?
- Counterexample-Guided Cartesian Abstraction Refinement

The CEGAR Algorithm

We iteratively refine an abstract representation of the problem:

1. Find abstract solution (if not $\rightarrow$ unsolvable)
2. Apply abstract solution to concrete problem
3. Find flaw in abstract solution (if not $\rightarrow$ solved)
4. Refine abstraction to remove flaw

If run long enough, this approach will eventually solve the problem or prove it to be unsolvable. If stopped before, the goal distance of each abstract state can serve as a heuristic for the concrete states represented by it.

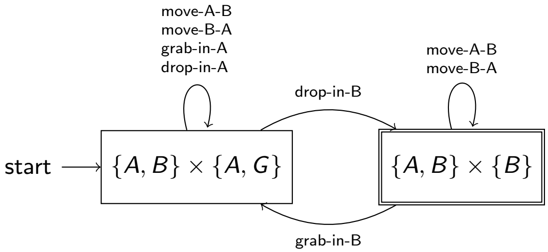
Example: Gripper

Example for a planning task: A robot needs to pick up a ball in room A and drop it in room B. Possible locations of the robot: $\{A, B\}$ (either room). Possible locations of the ball: $\{A, B, G\}$ (either room or in gripper).

Example: Gripper

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\text{start} \arrow[r] & \boxed{\langle A, A\rangle} \arrow[r, bend left, "\text{grab-in-A}"] \arrow[d, bend left, "\text{move-A-B}"] & \boxed{\langle A, G\rangle} \arrow[l, bend left, "\text{drop-in-A}"] \arrow[d, bend left, "\text{move-A-B}"] & \boxed{\boxed{\langle A, B\rangle}} \arrow[d, bend left, "\text{move-A-B}"] \\
& \boxed{\langle B, A\rangle} \arrow[u, bend left, "\text{move-B-A}"] & \boxed{\langle B, G\rangle} \arrow[u, bend left, "\text{move-B-A}"] \arrow[r, bend left, "\text{drop-in-B}"] & \boxed{\boxed{\langle B, B\rangle}} \arrow[u, bend left, "\text{move-B-A}"] \arrow[l, bend left, "\text{grab-in-B}"]
\end{tikzcd}$$

A simple abstraction



The Flaw

In the above abstraction, $\langle$drop-in-B$\rangle$ is a valid path. The action "drop-in-B" has the prerequisites "robot in B" and "ball in G", and the effect "ball in B". Applying it to the concrete problem, we see that drop-in-B cannot be applied to the initial state as both its prerequisites are not fulfilled. So we refine the abstract state with regard to one of these prerequisites. Let's pick "robot in B". We therefore separate the abstract state $\{A, B\} \times \{A, G\}$ into $\{A\} \times \{A, G\}$ and $\{B\} \times \{A, G\}$.

The refined abstraction

$$\begin{tikzcd}
\text{start} \arrow[r] & \boxed{\{A\} \times \{A, G\}} \arrow[loop above, "\substack{\text{grab-in-A}\\ \text{drop-in-A}}"] \arrow[d, bend left, "\text{move-A-B}"] & \\
& \boxed{\{B\} \times \{A, G\}} \arrow[u, bend left, "\text{move-B-A}"] \arrow[r, bend left, "\text{drop-in-B}"] & \boxed{\boxed{\{A, B\} \times \{B\}}} \arrow[l, bend left, "\text{grab-in-B}"] \arrow[loop above, "\substack{\text{move-A-B}\\ \text{move-B-A}}"]
\end{tikzcd}$$

Finding Abstract Solutions

- Step 1 of the CEGAR algorithm requires a search $\rightarrow$ Originally implemented as A*
- A* needs a heuristic.
- Helpful property: over subsequent iterations, h can only increase!

Updating h based on g

We can use information gained during searches on previous abstractions to construct a heuristic for the current abstraction. If $C^\star$ is the cost of an optimal abstract path and $g(s)$ is the distance of state s from the initial state, then

$$h(s) = C^\star - g(s)$$

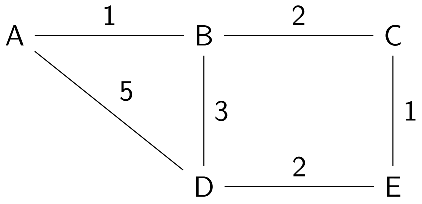
is an admissible heuristic:

$$h(s) \leq h^\star(s)$$

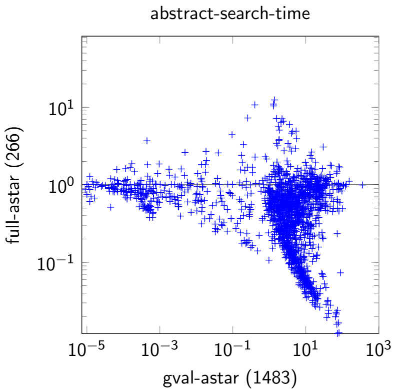
What if we could have perfect information about the abstraction at all times?

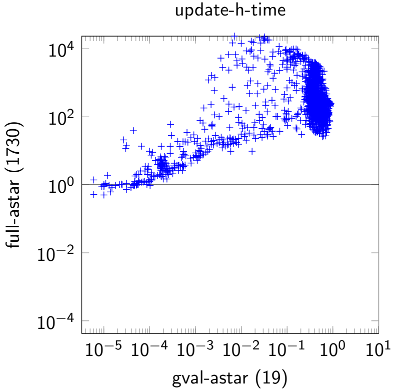
Dijkstra's Algorithm

Dijkstra's algorithm computes the shortest path from each node to one specific node:



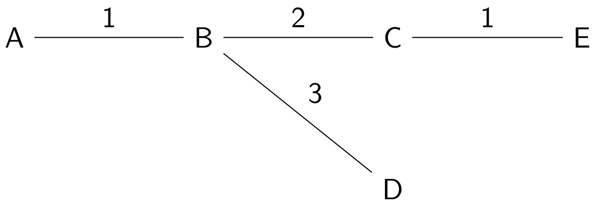
Dijkstra's Algorithm - Results





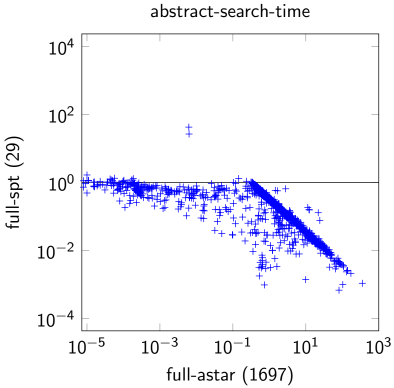
The Shortest-Path Tree

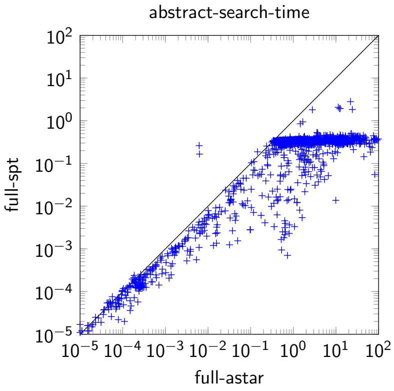
Dijkstra's algorithm computes an optimal path for every state (including the initial state), so performing a search is no longer necessary. Instead, the optimal paths form a *shortest-path tree*:



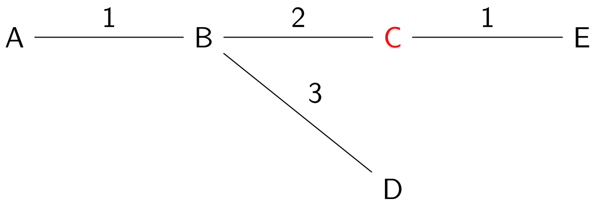
Finding an optimal solution is now the trivial task of traversing the tree.

Shortest-Path Tree: Results



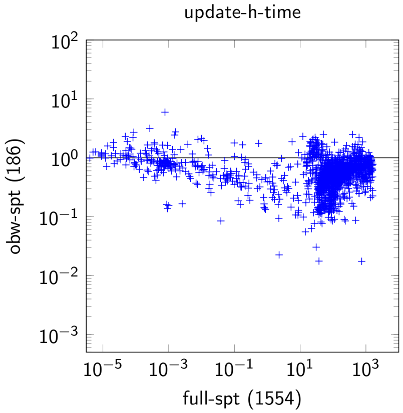


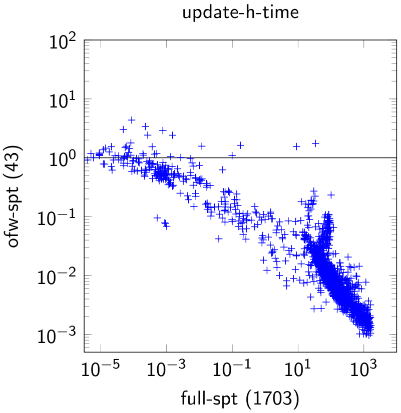
Identifying Orphaned States



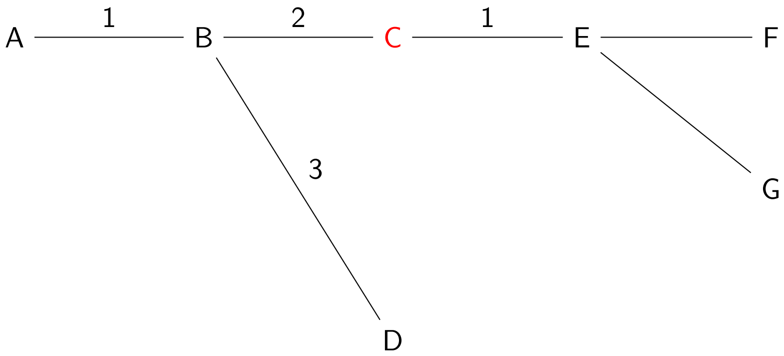
If state C is split into two new states, the optimal solutions for B and D cannot possibly change. The two new states as well as state E, however, are now "orphaned" and need to be recomputed.

Orphaned States: Results



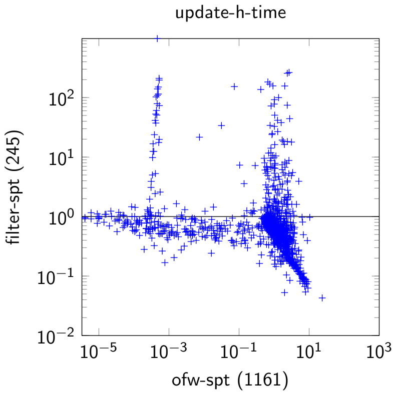


Filtered Orphaned List

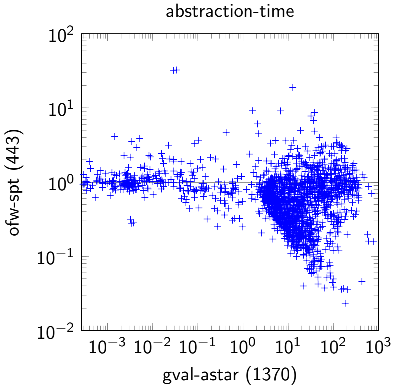


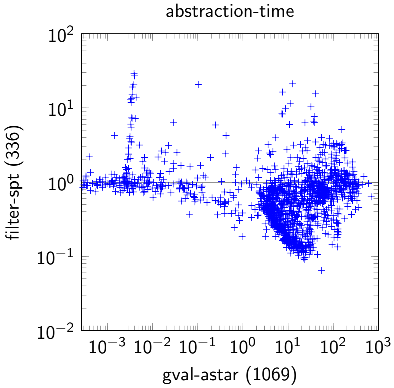
If there is still an optimal solution for E of cost 4 in the new abstraction, we don't need to recompute F and G. Filtering such cases out (and reconnecting them) reduces the number of orphaned states.

Filtered Orphans: Results



Final Comparison





Summary

- It is possible to have perfect information about goal distance and the shortest path for every abstract state, and updating this information incrementally every time the abstraction changes.
- While doing so is costly, the most efficient algorithms can compete with the speed of $A^\star$ and an incrementally updated admissible heuristic.