Memory Bounded Bidirectional Search

Eshed Shaham, Ariel Felner, Jeffrey S. Rosenschein

The Problem

- Unidirectional search goal test:

$$node == goal$$

Memory required: 1

- Naïve Bidirectional search goal test:

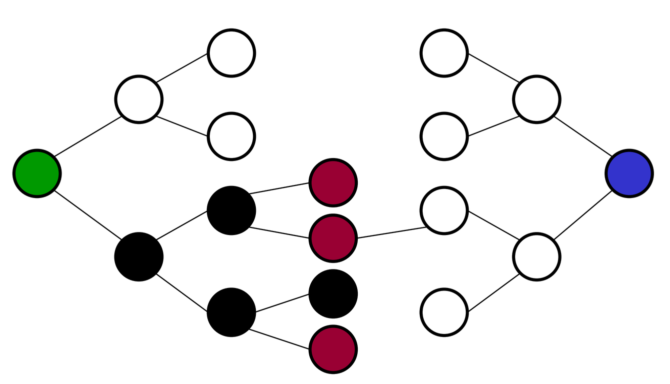
$$open_F \cup open_B == \emptyset$$

Memory required: $\min(|open_F|, |open_B|)$

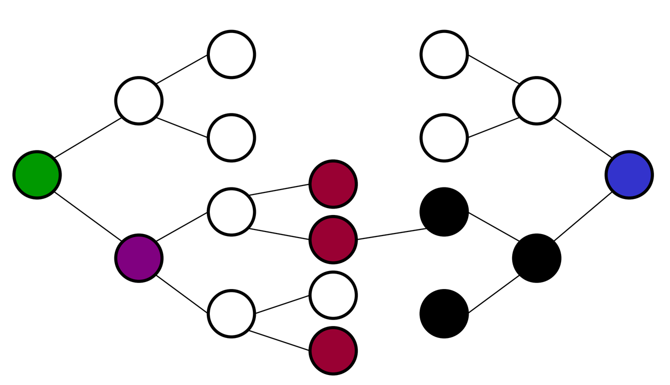
Hard to bound!

Common Ancestor Search

Step 1: run IDA* from **start** to **goal** and store **potential meeting points** in memory until it is full



Step 2: run A* from to **goal** to the **common ancestor of nodes in memory**



Iterative Deepening Bidirectional Search

- A general framework for running Memory bounded bidirectional search.
- Uses two thresholds – th_F for the forward search and th_B for the backward search
- Every iteration uses an external search procedure to look for a node distanced th_F from the start and th_B from the goal.
- If the search found no solution, either th_F or th_B is increased according to a predetermined policy.

Examples:

Search	Policy	Result
DFS	Always th_F	IDA* [1]
BFS	Always th_F	BFIDA* [2]
BFS	Alternate	Iterative version of MM [3]

Type Systems

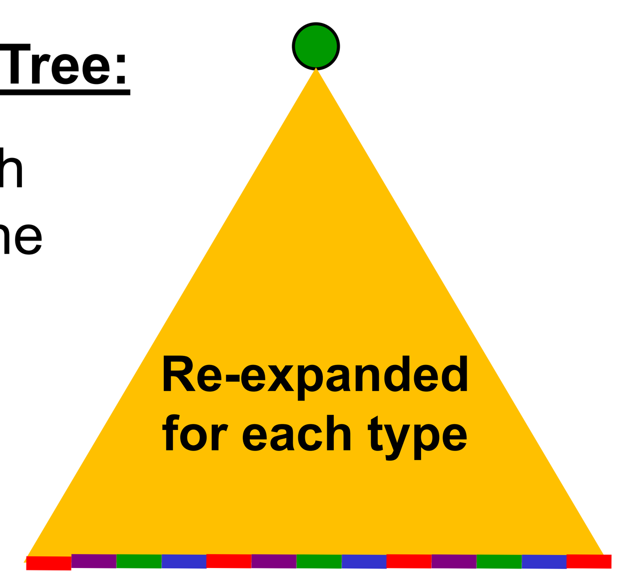
- Partition $open_F$ into k disjoint types $T_1, T_2, \ldots, T_k$, such that every type fits in memory.
- Run the search algorithm k times, each time only looking for meeting points in a single type.

Blind

- **Example**: Partition according to an arbitrary hash function on the state
- Performs the entire search every iteration
- Domain agnostic, requires no special properties
- Saves a factor of k in space for a cost of k in running time

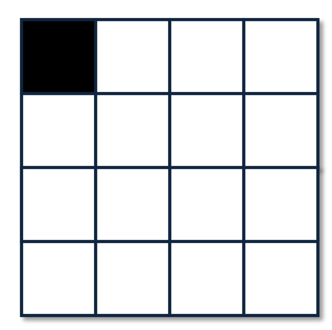
Forward Search Tree:

Types marked with colored lines on the search front

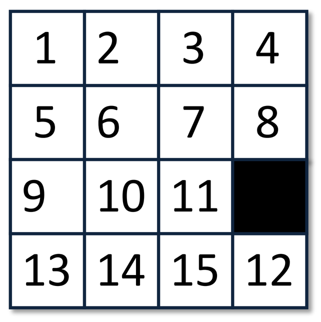


Informed

- **Example**: In the sliding tile puzzle, partition according to the location of the blank.
- Performs only part of the search every iteration
- Prunes states with no descendants in the current type.
- Saves a factor of k in space for a significantly smaller cost in running time



Current type: Blank in top left corner



This node can be pruned if $g > th_F - 5$

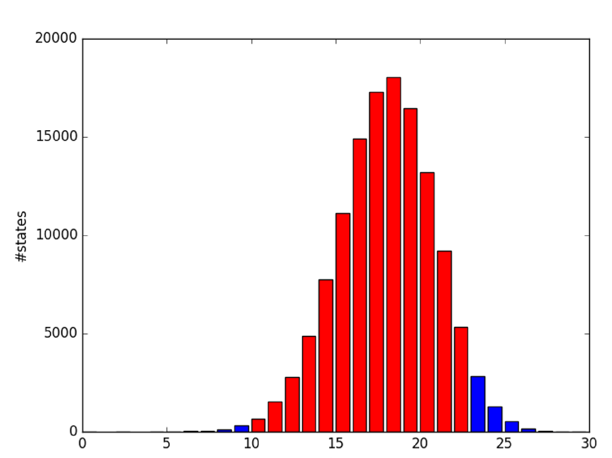
Reference Point Search

- An informed type system where states are partitioned into types according to the heuristic value from some state R
- Assuming a consistent heuristic, we can prune node n in iteration i if

$$|h(r,n) - i| > th - g(n)$$

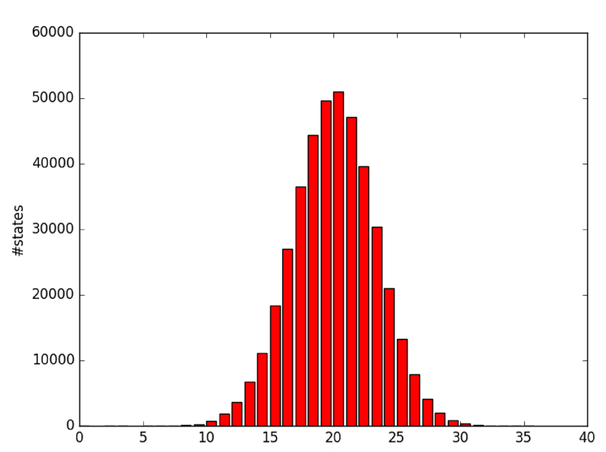
- Runtime cost for finding a solution is less than half the full traversal of the search tree and memory requirements are orders of magnitude smaller

Domain	Time ratio	Memory ratio
Pancake 35	0.44	1/98.7
Pancake 40	0.47	1/124.2
Pancake 45	0.41	1/170.5

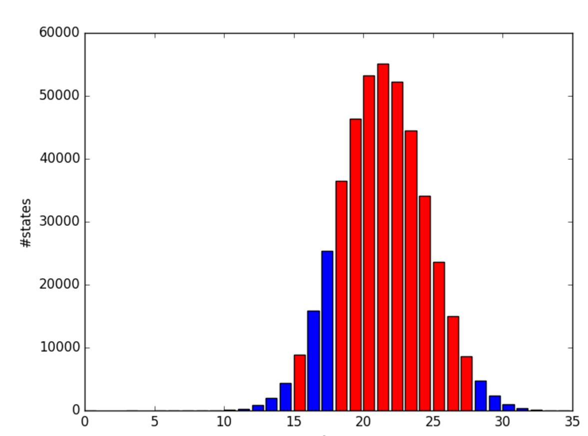


Pancake 35

Distribution of states in each type (h-value) For sample instances
Red: types containing a solution
Blue: types without a solution



Pancake 40



Pancake 45

Bloom Filters and Results

- An orthogonal approach of storing the open list in Bloom Filters was examined.
- Experiments applying Iterative Deepening Bidirectional Search with Reference Point Search and Bloom Filters managed to optimally solve the 86-pancake puzzle in a few hours on a workstation.
- A memory bound of approx. 8×10^5 states was used and in total, more than 4×10^{10} nodes were generated.
- This is the largest Pancake puzzle ever solved.

[1] Korf R. E. Artificial intelligence 27(1):97–109 (1985).
[2] Rong Zhou and Eric A. Hansen, ICAPS 2004.
[3] R. Holte et al., AAAI 2016.