Limited Discrepancy Beam Search

Paper by: David Furcy & Sven Koenig

Presentation by: Michael Niggli

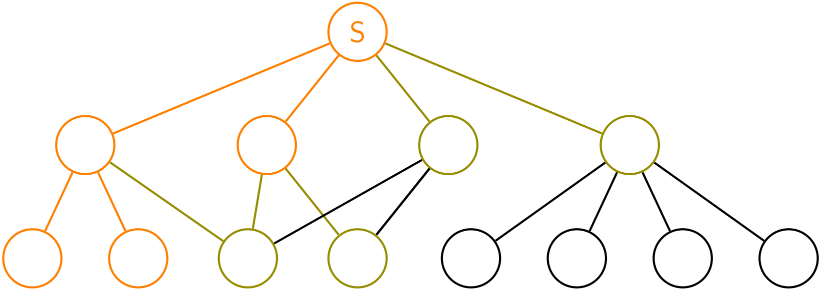
November 1, 2012

Presention made as part of the proceedings of the 2012 fall semester's *Search and Optimization* seminar at the University of Basel

Beam Search

- Optimization of Best-First search to reduce memory usage, sacrificing completeness
- Build search tree using breadth-first
- On each level:
  - Expand all successors for the current level's states
  - Order by heuristic
  - Drop all but the best b successor states (yielding a beam width of b)
- Terminate upon reaching a goal state or exhausting memory (or the searched space)

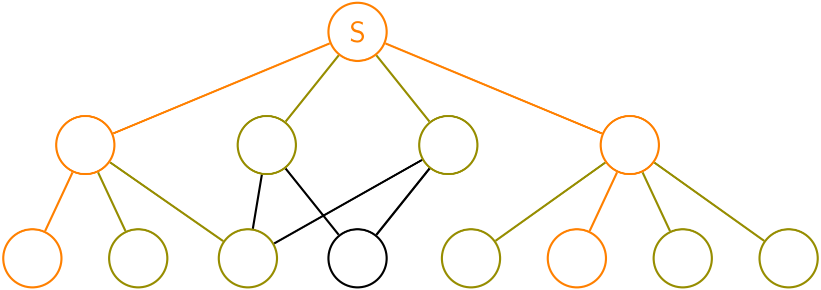
Beam Search



Beam width = 2

- Nodes contained in the Beam
- Nodes that were expanded, but pruned
- Nodes that were not expanded at all

Beam Search (unsorted)



Beam width $= 2$

Nodes contained in the Beam
Nodes that were expanded, but pruned
Nodes that were not expanded at all

Beam Search vs. 48-Puzzle

b	Path length	States generated	States stored	Runtime (s)	Problems solved
1	N/A	N/A	N/A	N/A	0 %
5	11737.12	147239	58680	0.09	100 %
10	36281.64	904632	362799	0.601	100 %
50	25341.44	3211244	1266902	2.495	86 %
100	12129.88	3079594	1212579	2.296	86 %
500	2302.86	2899765	1148559	2.205	74 %
1000	1337.95	3346004	1331451	2.822	84 %
5000	481.30	5814061	2365603	5.500	86 %
10000	440.07	10569816	4312007	11.307	80 %
50000	N/A	N/A	N/A	N/A	0%

- Larger Beams
  - $\rightarrow$ *better* solutions, higher memory consumption
  - $\rightarrow$ not necessarily *more* solutions

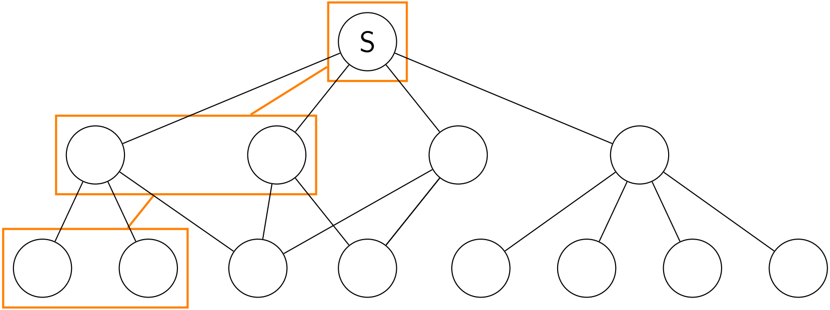
Improving Beam search

- Goal: 100% of the puzzles solved, with shorter solutions paths
- Varying beam width won't work - larger beams find less solutions, smaller ones find longer paths
- Misleading heuristic values prevent finding of a solution

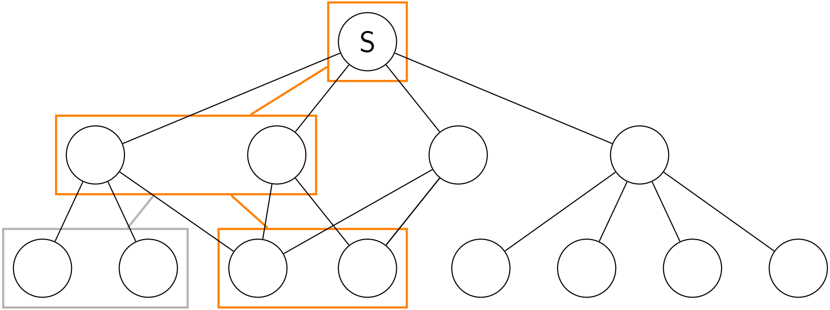
Improving Beam search

- Goal: 100% of the puzzles solved, with shorter solutions paths
- Varying beam width won't work - larger beams find less solutions, smaller ones find longer paths
- Misleading heuristic values prevent finding of a solution
- Backtracking to beam search circumvents this
- Let's call this Depth-first Beam search (DB)

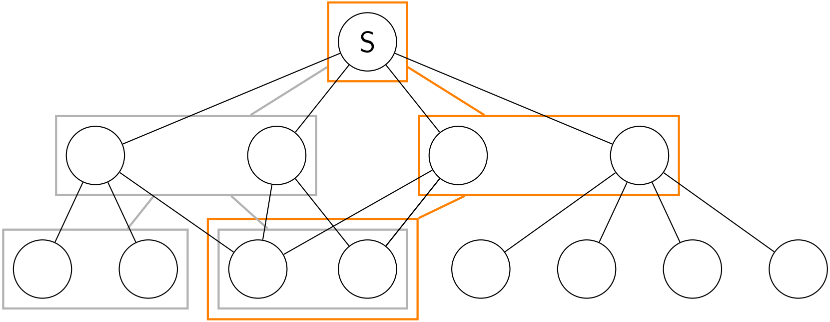
Depth-first Beam search (DB)



Depth-first Beam search (DB)

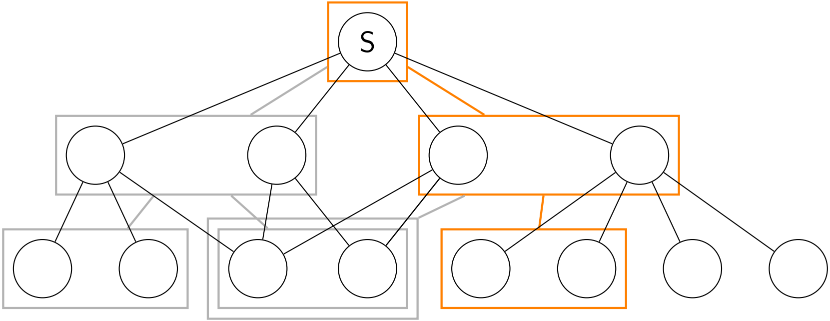


Depth-first Beam search (DB)

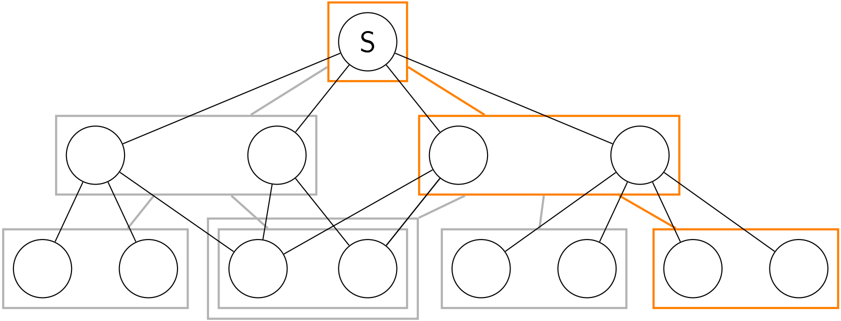


Note: Nodes are expanded a second time!

Depth-first Beam search (DB)



Depth-first Beam search (DB)



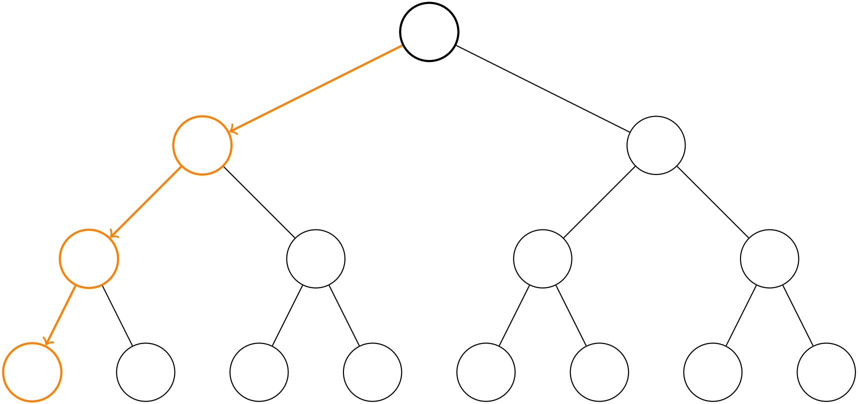
Depth-first Beam search (DB)

- DB is *very* slow
- Presumed reason: heuristics mislead early on rather than close to the goal
- Idea: Revisit states closer to the start early; heuristics fail there more often than further down the tree.

Limited Discrepancy Search

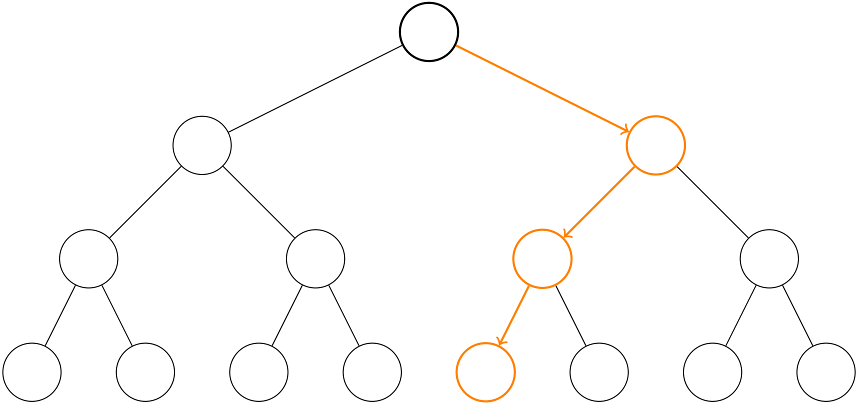
- Designed for finite binary trees
- Successors are sorted by heuristic, the better option is always *left*
- A discrepancy is choosing right over left against the heuristic value
- Try finding a solution first without, then with increasing number of allowed discrepancies (until a solution is found)

Limited Discrepancy Search



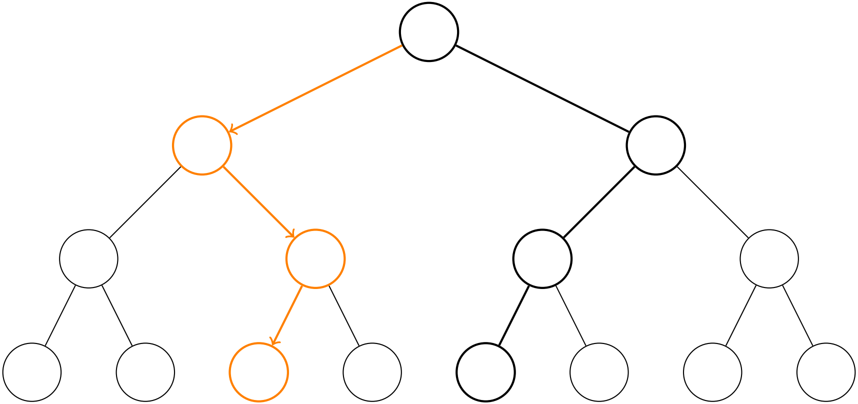
LDS without any allowed discrepancies

Limited Discrepancy Search



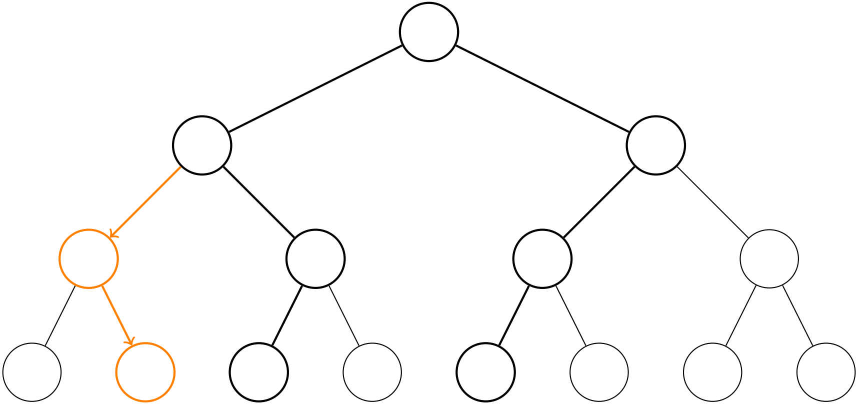
LDS with 1 allowed discrepancy

Limited Discrepancy Search



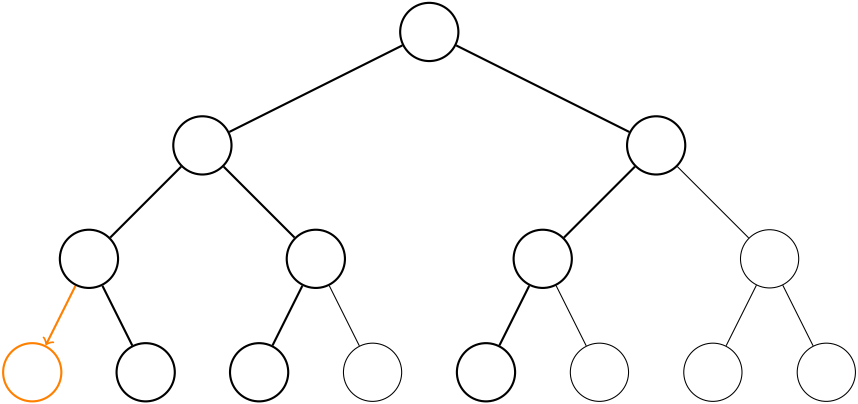
LDS with 1 allowed discrepancy

Limited Discrepancy Search



LDS with 1 allowed discrepancy

Limited Discrepancy Search



LDS with 1 allowed discrepancy

Generalized LDS

- LDS was for binary trees only
- Use hash table for cycle detection in GLDS
- Count going to a non-best successor as one discrepancy
- A discrepancy of one thus allows us to search the sub-trees under all non-best successors with a discrepancy of zero

BULB – Beam Search Using Limited Discrepancy Backtracking

- GLDS combined with Depth-first Beam search
- As in DB, we work with slices of states

BULB – Beam Search Using Limited Discrepancy Backtracking

- GLDS combined with Depth-first Beam search
- As in DB, we work with slices of states

$$\begin{tikzcd}
 & \text{DFS} \arrow[r] \arrow[dr] & \text{LDS} \arrow[r] & \text{GLDS} \arrow[d] \\
\text{BFS} \arrow[r] & \text{Beam Search} \arrow[r] & \text{DB} \arrow[r] & \text{BULB}
\end{tikzcd}$$

Influences between algorithms

Properties of BULB

- Memory consumption $O(Bd)$, with beam width B and max search tree depth d
- Complete (we find a solution if one exists and we have enough memory to find it)
- Being complete makes BULB better than Beam search
- Maximum tree depth $\sim M/B$, where M is the available memory (better than Breadth-first search, which has max depth of only $\log_B M$)
- Pretty fast (in experiments)

Experiments - N-Puzzle

- 48-Puzzle
  - BULB solves all instances with beam width 10000, avg. path length of 440
  - Regular beam search had avg. length of 11737 when solving all instances! (B=5)
- 80-Puzzle, memory for 3'000'000 states
  - Not all 50 random instances solvable with Beam search
  - BULB does them all
  - Fastest run: 12 seconds, avg. path length ~181000
  - Spending 120 seconds brings avg. path length of ~1130, just 5 times the shortest path

Experiments - 4-Peg Towers of Hanoi

- 50 random instances with 22 disks each
- Memory capacity for 1'000'000 states
- Pattern DB as heuristic function
- Not all solved by beam search
- Fastest average run time with BULB: 1.5s (b=40)
- b = 1000 takes 7s, but brings path length down to 870 (from 10'000)

Questions?

?