
Overview

⇒ Lowest-Cost-First

- Best-First Search
- A* Search
- Iterative Deepening

Lowest-cost-first Search

- Sometimes there are costs associated with arcs. The cost of a path g is the sum of the costs of its arcs
- Lowest-cost-first search finds the shortest path to a goal node
- Frontier is implemented as a priority queue ordered by g
 - At each stage, it selects the shortest path on the frontier
- When arc costs are equal ⇒ breadth-first search

Overview

- **Lowest-Cost-First**
⇒ **Best-First Search**
- **A* Search**
- **Iterative Deepening**

Heuristic Search

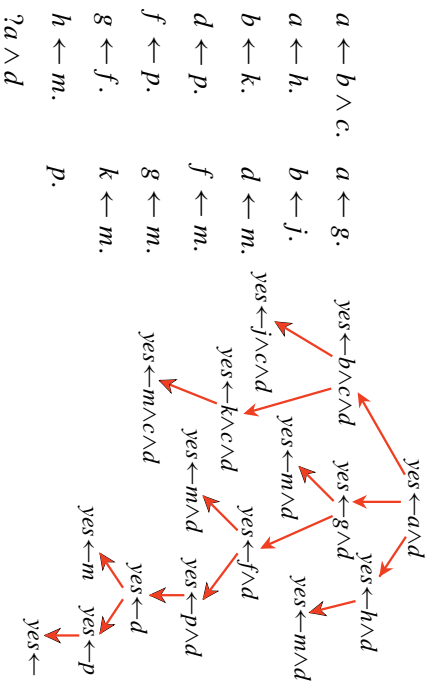
- Previous methods do not take into account goal until at goal node
- Often there is extra knowledge that can be used to guide the search: heuristics
- Use $h(n)$ as estimate of distance from node n to a goal node
- $h(n)$ is underestimate if it is less than or equal to the actual cost of the shortest path from node n to a goal
- $h(n)$ uses only readily obtainable information about a node

Best-first Search

- Idea: always select node on the frontier with smallest h -value
- Treat the frontier as a priority queue ordered by h
- Uses space exponential in path length

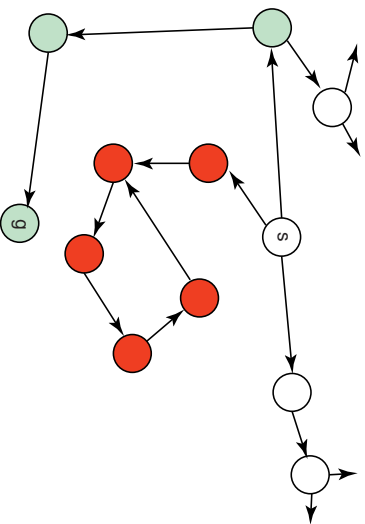
Applying Best-First Search to Top-Down Proofs

- How could we use this in searching through resolutions?



Seems Like A Good Idea But ...

- Not guaranteed to find a solution, even if one exists



- It doesnt always find the shortest path

Example with Top-Down Theorem Proving

- Not guaranteed to find a solution, even if one exists
 - $g \leftarrow a$
 - $g \leftarrow d \wedge e$
 - $a \leftarrow b$
 - $b \leftarrow a$
 - d
 - e
 - $? g$

Overview

- Lowest-Cost-First
- Best-First Search
 - ⇒ A* Search
- Iterative Deepening

A* Search

- A* search takes path to a node and heuristic value into account
- Let $g(n)$ be the cost of the path found to node n
 - From lowest-cost first search
- Let $h(n)$ be the estimate of the cost from n to a goal
 - From best-first search
- Let $f(n) = g(n) + h(n)$.
 $f(n)$ is an estimate of a path from the start to a goal via n

$$\underbrace{\underbrace{\text{start} \xrightarrow{\text{actual}} n}_{g(n)}}_{f(n)} \xrightarrow{\text{estimate}} \underbrace{\text{goal}}_{h(n)}$$

- A* orders the frontier by $f(n)$

A* Finds Optimal Solution

- If there is a solution, A* always finds an optimal solution
 - the first path to goal that it finds is optimal
- If ...
 - the branching factor is finite (not necessarily a finite number of nodes)
 - arc costs are bounded above zero (there is some $\epsilon > 0$ such that all of the arc costs are greater than ϵ)
 - $h(n)$ is an underestimate of the cost of the best path from n to a goal node and ≥ 0

Proof that if it finds a path, the path is optimal

- Let an optimal path have weight f_1
- Cells in the frontier are ordered by $g(n) + h(n)$
 - Where $g(n)$ is strictly increasing as you go down the path
 - And $h(n)$ is a lower-estimate ≥ 0 of the remaining distance
- Assume A* stops at a goal node with non-optimal path
 - So, non-optimal path p was on top of the frontier
 - Since p is not optimal, $g(p) > f_1$
 - Since p ends at the goal $g(p) = f(p)$, and so $f(p) > f_1$
 - But, part of the optimal path will be in the frontier, and it will have an f -value $\leq f_1$ (since f -values never over estimate)
- Hence, it would have been higher in the frontier than p , and so p would not have been chosen

Proof that it will find a path

- Let an optimal path have weight f_1
- Only a finite number of subpaths m have $g\text{-score} \leq f_1$
 - Because each arc has weight at least ϵ and finite branching
 - Note: subpath might not end at a goal node and $g\text{-score}$ measures the full cost of the subpath
- Hence, finite number of subpaths $n \leq m$ have $f\text{-score}$ at most f_1
 - Because $f\text{-score}$ of subpath is greater than its $g\text{-score}$
- A subpath of the optimal path is always in frontier and its $f\text{-score}$ always at most f_1
- After at most n steps, optimal path must be on top of frontier (if we haven't stopped earlier)

Summary of Search Strategies

Strategy	Frontier Selection	Halts?	Space
Depth-first	Last node added	No	Linear
Breadth-first	First node added	Yes	Exp
Best-first	Global min $h(n)$	No	Exp
Lowest-cost-first	Global min $g(n)$	Yes	Exp
A*	Global min $f(n)$	Yes	Exp

Overview

- Lowest-Cost-First
 - Best-First Search
 - A* Search
- ⇒ Iterative Deepening

Iterative Deepening

- So far all search strategies that are guaranteed to halt use exponential space
- Idea: lets recompute elements of the frontier rather than saving them
- Look for proofs of depth 0, then 1, then 2, then 3, etc
- You need a depth-bounded depth-first searcher
- If proof cannot be found at depth B , look for proof at depth $B + 1$

Depth-bounded depth-first search

- *dbsearch*(*N*, *D*, *P*) is true if *P* is path of length *D* from *N* to goal

```

dbsearch(Node, 0, [Node]) <-
    is_goal(Node).
dbsearch(Node, D, [Node|P] <-
    D > 0 ^
    neighbors(Node, Neighbors) ^
    member(NewNode, Neighbors) ^
    D1 is D - 1 ^
    dbsearch(NewNode, D1, P).

```

? dbsearch(start, 5, Path)

- Note it builds the path on the way out

- Uses Prologs depth-first search to iterate through all members