Overview

- Lowest-Cost-First
- Best-First Search
- Iterative Deepening

Lowest-Cost-First Search

When arc costs are equal = Breadth-First search
- A path with a lower cost is preferred over a path with a higher cost
- Frontier is implemented as a priority queue ordered by Cost
- Lowest-Cost-First Search finds the shortest path to a goal node
- Cost of path is the sum of the costs of its arcs
- Sometimes there are costs associated with nodes. The cost of a

A* Search

Heuristic determines
- Search
- Best-First Search
- Lowest-Cost-First
Applying Best-First Search to Top-Down Proofs

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

Heuristic Search

- Previous methods do not take into account goal until it is reached
- Often there is extra knowledge that can be used to guide the search
- $h(n)$ is an estimate of distance from node $n$ to a goal node
- $h(n)$ is an underestimate if it is less than or equal to the actual cost
- $h(n)$ is admissible if it is never greater than the true cost
- $h(n)$ is consistent if $h(n)$ is admissible and $h$ is the minimum of $h$ of similar nodes
- $h(n)$ is optimal if $h(n)$ is consistent and the heuristic is perfect
Overview

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

Example with Top-Down Theorem Proving

\[ g \leftarrow a \]
\[ a \leftarrow b \]
\[ b \leftarrow a \]
\[ d \leftarrow e \]
\[ e \leftarrow f \]
\[ f \leftarrow g \]
\[ q \rightarrow r \]
\[ r \rightarrow q \]

Seems like a good idea but...

- Not guaranteed to find a solution, even if one exits
- It doesn't always find the shortest path
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 $\geq 0$ of the remaining distance.

• Assume A* stops at a goal node with non-optimal path $p$.
  
  So, non-optimal path $p$ was on top of the frontier.

  - Since $p$ is not optimal, $g(p) > f_1$.
  
  - Even if $p$ 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 $\leq f_1$.

  - Hence, it would be higher in the frontier than $p$, and so will have

A* finds optimal solution

• A* always finds an optimal solution if there is a solution.

  - The branching factor is finite (not necessarily a finite number of nodes).
  
  - The arc costs are bounded above zero.
  
  - The heuristic $h(n)$ is an underestimate of the cost from $n$ to a goal.

  - If $h(n)$ is sufficient, all the non-optimal paths are eliminated as early as possible.

  - Hence, the algorithm will find the optimal path.

A* order the frontier by

\[
(f(n)) = g(n) + h(n)
\]

- Where $g(n)$ is the true cost from the start to $n$.
- And $h(n)$ is an underestimate of the cost from $n$ to a goal.

- If $h(n)$ is sufficient, all the non-optimal paths are eliminated as early as possible.
- Hence, the algorithm will find the optimal path.

A* order the frontier by

\[
(f(n)) = g(n) + h(n)
\]

- Where $g(n)$ is the true cost from the start to $n$.
- And $h(n)$ is an underestimate of the cost from $n$ to a goal.

- If $h(n)$ is sufficient, all the non-optimal paths are eliminated as early as possible.
- Hence, the algorithm will find the optimal path.
Overview

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

Summary of Search Strategies

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Frontier Selection</th>
<th>Halts?</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>Last node added</td>
<td>No</td>
<td>Linear</td>
</tr>
<tr>
<td>Breadth-first</td>
<td>First node added</td>
<td>Yes</td>
<td>Exp</td>
</tr>
<tr>
<td>Lowest-cost-first</td>
<td>Global min g(n)</td>
<td>No</td>
<td>Exp</td>
</tr>
<tr>
<td>A*</td>
<td>Global min f(n)</td>
<td>Yes</td>
<td>Exp</td>
</tr>
</tbody>
</table>

Proof that it will find a path

Let an optimal path of length \( n \) exist.
- If we have not stopped earlier,
  - Every node on the optimal path must be in the frontier.
  - A path of length \( n \) in the optimal path is always in the frontier and hence, the frontier contains all nodes on the optimal path.
  - Every node not on the optimal path has a lower \( f \)-score than the next node.
  - Hence, the number of subpaths \( m \) of the optimal path is at most \( n \).
  - Given that the optimal path is unique, we can conclude that the optimal path is the one with the lowest \( f \)-score.
  - Let \( n \) be the number of nodes in the path.

We have an optimal path.
Depth-bounded depth-first search

\[ \text{dbsearch}(N, D, P) \] is true if \( P \) is a path of length \( D \) from \( N \) to \( \text{goal} \).

\[ \text{dbsearch}(\text{Node}, 0, [\text{Node}]) \leftarrow \text{is_goal(} \text{Node} \text{)} . \]

\[ \text{dbsearch}(\text{Node}, D, [\text{Node}|P]) \leftarrow D > 0 \wedge \text{neighbors(} \text{Node} \text{, Neighbors) \wedge member(} \text{NewNode} \text{, Neighbors) \wedge D1 } = \text{D} - 1 \wedge \text{dbsearch(} \text{NewNode} \text{, D1, P)} . \]

? \text{dbsearch(} \text{start} \text{, 5, Path) .} \]

Note it builds the path on the way out.

Iterative Deepening

- So far all search strategies that are guaranteed to halt use exponential space.
- Idea: let's recompute elements of the frontier rather than saving them.
- Look for proofs of depth 0, then 1, then 2, 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

Iterative Deepening

- Look for proofs of depth 0, then 1, then 2, etc.
- If proof cannot be found, expand all successors.
- So far all search strategies that are guaranteed to halt use exponential space.

Iterative Deepening