Overview

• Lowest-Cost-First

• Best-First Search

• A* Search

= Lowest-Cost-First

Lowest-cost-first Search

• Sometimes there are costs associated with arcs. The cost of a path is the sum of the costs of the arcs.

• At each stage, it selects the shortest path on the frontier.

• Frontier is implemented as a priority queue ordered by \( g \) value.

• Lowest-cost-first search finds the shortest path to a goal node.

• When arc costs are equal \( \Leftrightarrow \) breadth-first search.
How could we use this in searching through resolutions?

### Applying Best-First Search to Top-Down Proofs

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

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

$c\leftarrow a\land d$
yes $\leftarrow a\land d$
yes $\leftarrow j\land c\land d$
yes $\leftarrow k\land c\land d$
yes $\leftarrow m\land c\land d$
yes $\leftarrow g\land d$
yes $\leftarrow b\land c\land d$
yes $\leftarrow m\land d$
yes $\leftarrow f\land d$
yes $\leftarrow p\land d$
yes $\leftarrow h\land p$
yes $\leftarrow m$
yes $\leftarrow p$
yes $\leftarrow g$
yes $\leftarrow a\lor c\lor p$
yes $\leftarrow a$
yes $\leftarrow f$
yes $\leftarrow m$
yes $\leftarrow p$
yes $\leftarrow h$
yes $\leftarrow a$
yes $\leftarrow m$
yes $\leftarrow p$
yes $\leftarrow g$
yes $\leftarrow a$
Overview

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

Example with Top-Down Theorem Proving

\[ \begin{align*}
    g & \leftarrow a \\
    g & \leftarrow d \land e \\
    a & \leftarrow b \\
    b & \leftarrow a \\
    d & \land e & \rightarrow f \\
    a & \rightarrow g
\end{align*} \]

Seems Like A Good Idea But...

- Not guaranteed to find a solution, even if one exists
- Doesn't always find the shortest path

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

- Let an optimal path have weight $f^*$
- 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-bound of the remaining distance
- Choose the lowest-weight cell first
- Let an optimal path have weight $f^*$

A* Finds Optimal Solution

- If there is a solution, A* always finds an optimal solution
- If there is a solution, A* always finds an optimal solution
- A* starts at the root
- A* expands nodes by $f(n)$
- A* orders the frontier by $f(n)$
- A* stops when the root is expanded
- A* searches path to a node and heuristic value into account
- A* starts at the root
- A* expands nodes by $f(n)$
- A* orders the frontier by $f(n)$
- A* stops when the root is expanded
- A* searches path to a node and heuristic value into account
Overview

- Lowest-Cost-First
- Best-First 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>Yes</td>
<td>Exp</td>
</tr>
<tr>
<td>Breadth-first</td>
<td>First node added</td>
<td>No</td>
<td>Exp</td>
</tr>
<tr>
<td>Best-first</td>
<td>Global min h(n)</td>
<td>No</td>
<td>Exp</td>
</tr>
<tr>
<td>Lowest-cost-first</td>
<td>Global min g(n)</td>
<td>Yes</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

After at most n steps, optimal path must be on top of frontier (f-score always at most f)

- f-score of the optimal path is always in frontier and is a subpath of the optimal path
- Because f-score of subpath is greater than its g-score
- Hence, the number of subpaths with f-score at most f is finite

- A subpath of the optimal path is always in frontier and its g-score measures the full cost of the subpath
- Because each arc has a weight at least ε and initial branching
- Only a finite number of subpaths m have g-score at most f
- Let an optimal path have weight f
Depth-bounded depth-first search

\[ \text{dbsearch}(N, D, P) \]

is true if \( P \) is a path of length \( D \) from \( N \) to goal.

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

← is goal(Node).

\[ \text{dbsearch}(Node, D, \text{NewP}) \]

← \( D > 0 \) \( \land \) neighbors(Node, Neighbors) \( \land \) member(NewNode, Neighbors) \( \land \) non deterministic.

\[ D_1 \text{ is } D - 1 \]

\[ \land \]

\[ \text{dbsearch}(\text{NewNode}, D_1, \text{P}) \]

\[ \land \]

\[ \text{NewP} = \{\text{Node} \mid P\}. \]

A bit different from previous versions

- Gathers up all neighbors, and then non-deterministically chooses one
- Builds the path on the way out

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 it.

Depth-Bounded depth-first search

If proof cannot be found at depth \( D \), look for proof at depth \( D + 1 \).

You need a depth-bounded depth-first searcher.

Look for proofs of depth 0, then 1, then 2, then 3, etc.

So far all search strategies that are guaranteed to halt use exponential space.

Ierative Deepening