Learning Objectives

At the end of the class you should be able to:

- explain how cycle checking and multiple-path pruning can improve efficiency of search algorithms
- explain the complexity of cycle checking and multiple-path pruning for different search algorithms
- justify why the monotone restriction is useful for $A^*$ search
- predict whether forward, backward, bidirectional or island-driven search is better for a particular problem
- demonstrate how dynamic programming works for a particular problem
### Summary of Search Strategies

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Frontier Selection</th>
<th>Complete</th>
<th>Halts</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>Last node added</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Breadth-first</td>
<td>First node added</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heuristic depth-first</td>
<td>Local min $h(p)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Best-first</td>
<td>Global min $h(p)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lowest-cost-first</td>
<td>Minimal $cost(p)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A^*$</td>
<td>Minimal $f(p)$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Complete** — if there a path to a goal, it can find one, even on infinite graphs.

**Halts** — on finite graph (perhaps with cycles).

**Space** — as a function of the length of current path.
<table>
<thead>
<tr>
<th>Strategy</th>
<th>Frontier Selection</th>
<th>Complete</th>
<th>Halts</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth-first</td>
<td>Last node added</td>
<td>No</td>
<td>No</td>
<td>Linear</td>
</tr>
<tr>
<td>Breadth-first</td>
<td>First node added</td>
<td>Yes</td>
<td>No</td>
<td>Exp</td>
</tr>
<tr>
<td>Heuristic depth-first</td>
<td>Local min $h(p)$</td>
<td>No</td>
<td>No</td>
<td>Linear</td>
</tr>
<tr>
<td>Best-first</td>
<td>Global min $h(p)$</td>
<td>No</td>
<td>No</td>
<td>Exp</td>
</tr>
<tr>
<td>Lowest-cost-first</td>
<td>Minimal $\text{cost}(p)$</td>
<td>Yes</td>
<td>No</td>
<td>Exp</td>
</tr>
<tr>
<td>$A^*$</td>
<td>Minimal $f(p)$</td>
<td>Yes</td>
<td>No</td>
<td>Exp</td>
</tr>
</tbody>
</table>

**Complete** — if there a path to a goal, it can find one, even on infinite graphs.

**Halts** — on finite graph (perhaps with cycles).

**Space** — as a function of the length of current path
A searcher can prune a path that ends in a node already on the path, without removing an optimal solution.

In depth-first methods, checking for cycles can be done in ________ time in path length.

For other methods, checking for cycles can be done in ________ time in path length.

Does cycle checking mean the algorithms halt on finite graphs?
Multiple-Path Pruning

- Multiple path pruning: prune a path to node $n$ that the searcher has already found a path to.
- What needs to be stored?
- How does multiple-path pruning compare to cycle checking?
- Do search algorithms with multiple-path pruning always halt on finite graphs?
- What is the space & time overhead of multiple-path pruning?
- Can multiple-path pruning prevent an optimal solution being found?
Problem: what if a subsequent path to $n$ is shorter than the first path to $n$?
Problem: what if a subsequent path to $n$ is shorter than the first path to $n$?

- remove all paths from the frontier that use the longer path.
- change the initial segment of the paths on the frontier to use the shorter path.
- ensure this doesn’t happen. Make sure that the shortest path to a node is found first.
Suppose path $p$ to $n$ was selected, but there is a shorter path to $n$. Suppose this shorter path is via path $p'$ on the frontier.

Suppose path $p'$ ends at node $n'$.

$p$ was selected before $p'$, so:
Multiple-Path Pruning & \( A^* \)

- Suppose path \( p \) to \( n \) was selected, but there is a shorter path to \( n \). Suppose this shorter path is via path \( p' \) on the frontier.
- Suppose path \( p' \) ends at node \( n' \).
- \( p \) was selected before \( p' \), so:
  \[
  \text{cost}(p) + h(n) \leq \text{cost}(p') + h(n').
  \]
- Suppose \( \text{cost}(n', n) \) is the actual cost of a path from \( n' \) to \( n \).
  The path to \( n \) via \( p' \) is shorter than \( p \) so:
Multiple-Path Pruning & $A^*$

- Suppose path $p$ to $n$ was selected, but there is a shorter path to $n$. Suppose this shorter path is via path $p'$ on the frontier.
- Suppose path $p'$ ends at node $n'$.
- $p$ was selected before $p'$, so:
  \[ \text{cost}(p) + h(n) \leq \text{cost}(p') + h(n'). \]
- Suppose $\text{cost}(n', n)$ is the actual cost of a path from $n'$ to $n$. The path to $n$ via $p'$ is shorter than $p$ so:
  \[ \text{cost}(p') + \text{cost}(n', n) < \text{cost}(p). \]
  \[ \text{cost}(n', n) < \]
Suppose path $p$ to $n$ was selected, but there is a shorter path to $n$. Suppose this shorter path is via path $p'$ on the frontier.

Suppose path $p'$ ends at node $n'$.

$p$ was selected before $p'$, so:

$$\text{cost}(p) + h(n) \leq \text{cost}(p') + h(n').$$

Suppose $\text{cost}(n', n)$ is the actual cost of a path from $n'$ to $n$. The path to $n$ via $p'$ is shorter than $p$ so:

$$\text{cost}(p') + \text{cost}(n', n) < \text{cost}(p).$$

$$\text{cost}(n', n) < \text{cost}(p) - \text{cost}(p') \leq$$
Suppose path \( p \) to \( n \) was selected, but there is a shorter path to \( n \). Suppose this shorter path is via path \( p' \) on the frontier.

Suppose path \( p' \) ends at node \( n' \).

\( p \) was selected before \( p' \), so:
\[
\text{cost}(p) + h(n) \leq \text{cost}(p') + h(n').
\]

Suppose \( \text{cost}(n', n) \) is the actual cost of a path from \( n' \) to \( n \). The path to \( n \) via \( p' \) is shorter than \( p \) so:
\[
\text{cost}(p') + \text{cost}(n', n) < \text{cost}(p).
\]

\[
\text{cost}(n', n) < \text{cost}(p) - \text{cost}(p') \leq h(n') - h(n).
\]

We can ensure this doesn’t occur if
\[
|h(n') - h(n)| \leq \text{cost}(n', n).
\]
Monotone Restriction

- Heuristic function $h$ satisfies the monotone restriction if $|h(m) - h(n)| \leq cost(m, n)$ for every arc $\langle m, n \rangle$.
- If $h$ satisfies the monotone restriction, $A^*$ with multiple path pruning always finds the shortest path to a goal.
- This is a strengthening of the admissibility criterion.
The definition of searching is symmetric: find path from start nodes to goal node or from goal node to start nodes.

- **Forward branching factor:** number of arcs out of a node.
- **Backward branching factor:** number of arcs into a node.

Search complexity is $b^n$. Should use forward search if forward branching factor is less than backward branching factor, and vice versa.

- Note: when graph is dynamically constructed, the backwards graph may not be available.
Bidirectional Search

- Idea: search backward from the goal and forward from the start simultaneously.
- This wins as $2b^{k/2} \ll b^k$. This can result in an exponential saving in time and space.
- The main problem is making sure the frontiers meet.
- This is often used with one breadth-first method that builds a set of locations that can lead to the goal. In the other direction another method can be used to find a path to these interesting locations.
Island Driven Search

- **Idea:** find a set of islands between $s$ and $g$.

\[ s \rightarrow i_1 \rightarrow i_2 \rightarrow \ldots \rightarrow i_{m-1} \rightarrow g \]

There are $m$ smaller problems rather than 1 big problem.

- This can win as $mb^k/m \ll b^k$.
- The problem is to identify the islands that the path must pass through. It is difficult to guarantee optimality.
- The subproblems can be solved using islands $\Rightarrow$ hierarchy of abstractions.
**Idea:** for statically stored graphs, build a table of $dist(n)$ the actual distance of the shortest path from node $n$ to a goal. This can be built backwards from the goal:

$$
    dist(n) = \begin{cases} 
    0 & \text{if is\_goal}(n), \\
    \min_{\langle n, m \rangle \in A} (|\langle n, m \rangle| + dist(m)) & \text{otherwise}.
    \end{cases}
$$

This can be used locally to determine what to do.

There are two main problems:

- It requires enough space to store the graph.
- The $dist$ function needs to be recomputed for each goal.