AI Search

Chao Lan
Introduction

Uninformed Search vs. Informed Search
Two Types of Search

Uninformed Search: all nodes are equally promising to expand.

Informed Search: some nodes are more promising to expand.
Introduction

**Uninformed Search** vs. **Informed Search**
Overview of Uninformed Search

Uninformed search algorithms can only generate successors and distinguish goal and non-goal state.

Different strategies have different orders of node expansion.

- breadth-first
- uniform-cost
- depth-first
- depth-limited
- iterative-deepening
- bidirectional
Overview of Uninformed Search

Uninformed search algorithms can only generate successors and distinguish goal and non-goal state.

Different strategies have different orders of node expansion.

- breadth-first
- uniform-cost
- depth-first
- depth-limited
- iterative-deepening
- bidirectional
Breadth-First Search

Always expand node at the shallowest depth.
Breadth-First Search

Always expand node at the shallowest depth.

Figure 3.12  Breadth-first search on a simple binary tree. At each stage, the node to be expanded next is indicated by a marker.
Properties of Breadth-First Search

BFS is complete.
Properties of Breadth-First Search

BFS is complete.

BFS is not necessarily optimal (although it finds the shallowest path).
Properties of Breadth-First Search

BFS is complete.

BFS is not necessarily optimal (although it finds the shallowest path).

Q: when is BFS optimal?
Properties of Breadth-First Search

BFS is complete.

BFS is not necessarily optimal (although it finds the shallowest path).

Q: when is BFS optimal? when all steps have the same cost.
Properties of Breadth-First Search

Time and space complexities of BFS are high.

Q: if each node has b children & optimum is at depth d, what are the time and space complexities of BFS? (tip: time=#GeneratedNode, space=#StoredNode)
Properties of Breadth-First Search

Time and space complexities of BFS are high.

Q: if each node has b children & optimum is at depth d, what are the time and space complexities of BFS? (tip: time=#GeneratedNode, space=#StoredNode)

A: time = $O(b^d)$, space = $O(b^d)$ (why?)
## Properties of Breadth-First Search

How big is $O(b^d)$?

<table>
<thead>
<tr>
<th>Depth</th>
<th>Nodes</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>110</td>
<td>.11 milliseconds</td>
<td>107 kilobytes</td>
</tr>
<tr>
<td>4</td>
<td>11,110</td>
<td>11 milliseconds</td>
<td>10.6 megabytes</td>
</tr>
<tr>
<td>6</td>
<td>$10^6$</td>
<td>1.1 seconds</td>
<td>1 gigabyte</td>
</tr>
<tr>
<td>8</td>
<td>$10^8$</td>
<td>2 minutes</td>
<td>103 gigabytes</td>
</tr>
<tr>
<td>10</td>
<td>$10^{10}$</td>
<td>3 hours</td>
<td>10 terabytes</td>
</tr>
<tr>
<td>12</td>
<td>$10^{12}$</td>
<td>13 days</td>
<td>1 petabyte</td>
</tr>
<tr>
<td>14</td>
<td>$10^{14}$</td>
<td>3.5 years</td>
<td>99 petabytes</td>
</tr>
<tr>
<td>16</td>
<td>$10^{16}$</td>
<td>350 years</td>
<td>10 exabytes</td>
</tr>
</tbody>
</table>

**Figure 3.13** Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.
Overview of Uninformed Search

Uninformed search algorithms can only generate successors and distinguish goal and non-goal state.

Different strategies have different orders of node expansion.

- breadth-first
- uniform-cost
- depth-first
- depth-limited
- iterative-deepening
- bidirectional
Uniform-Cost Search

Always expand node $n$, which has the lowest path-cost $g(n)$.

- $g(n)$ is the path cost from initial state to $n$
Example of Uniform-Cost Search

Path from Sibiu to Bucharest?
Example of Uniform-Cost Search

Path from Sibiu to Bucharest?
- expand Sibiu
  - obtain Rimnicu (80) & Fagars (99)
- expand Rimnicu
  - obtain Pitesti (80+97=177)
- expand Fagaras
  - obtain Bucharest (99+211=310)
- expand Pitesti
  - obtain Bucharest (80+97+101=278)
- choose Sibiu-Rimnicu-Pitesti-Bucharest (278 < 310)

UCS will stop after goal node is expanded.
Properties of Uniform-Cost Search

UCS is complete in general.
Properties of Uniform-Cost Search

UCS is complete in general.

Q: can you give a counter-example?
Properties of Uniform-Cost Search

UCS is complete in general.

Q: can you give a counter-example?
A: an infinite path with zero cost
Properties of Uniform-Cost Search

UCS is complete in general.

Q: can you give a counter-example?
A: an infinite path with zero cost

Q: when is UCS guaranteed to find a solution?
Properties of Uniform-Cost Search

UCS is complete in general.

Q: can you give a counter-example?
A: an infinite path with zero cost

Q: when is UCS guaranteed to find a solution?
A: if cost of every path > small positive constant

Or, if cost of a finite number of paths > small positive constant. (why?)
Properties of Uniform-Cost Search

UCS is complete in general.

UCS is optimal in general.
Properties of Uniform-Cost Search

UCS is complete in general.

UCS is optimal in general.

Q: proof?
Properties of Uniform-Cost Search

UCS is complete in general.

UCS is optimal in general.

Q: proof?
A: basically, a node will not be expanded until its optimal path is established.
A quick review of big O notation and time complexity...
Big O Notation

Big O Notation: let $f(n)$, $g(n)$ be two functions. We say $f(n) \in O(g(n))$ if there is a positive $C$ and an $n_0$ such that for all $n > n_0$, there is $f(n) < C \cdot g(n)$. 
Exercise

Big O Notation: let $f(n)$, $g(n)$ be two functions. We say $f(n) \in O(g(n))$ if there is a positive $C$ and an $n_0$ such that for all $n > n_0$, there is $f(n) < C\cdot g(n)$.

True or False ?

(1) $5n \in O(n^2)$
Exercise

Big O Notation: let $f(n)$, $g(n)$ be two functions. We say $f(n) \in O(g(n))$ if there is a positive $C$ and an $n_0$ such that for all $n > n_0$, there is $f(n) < C \cdot g(n)$.

True or False?

(1) $5n \in O(n^2)$

(2) $5n^2 \in O(n^2)$
Exercise

Big O Notation: let $f(n)$, $g(n)$ be two functions. We say $f(n) \in O(g(n))$ if there is a positive $C$ and an $n_0$ such that for all $n > n_0$, there is $f(n) < C \cdot g(n)$.

True or False?

(1) $5n \in O(n^2)$
(2) $5n^2 \in O(n^2)$
(3) $5n^3 \in O(n^2)$
Exercise

Big O Notation: let $f(n)$, $g(n)$ be two functions. We say $f(n) \in O(g(n))$ if there is a positive $C$ and an $n_0$ such that for all $n > n_0$, there is $f(n) < C \cdot g(n)$.

True or False?

(1) $5n \in O(n^2)$

(2) $5n^2 \in O(n^2)$

(3) $5n^3 \in O(n^2)$

(4) $100n + 0.01n^2 \in O(n)$
Exercise

Big O Notation: let \( f(n) \), \( g(n) \) be two functions. We say \( f(n) \in O(g(n)) \) if there is a positive \( C \) and an \( n_0 \) such that for all \( n > n_0 \), there is \( f(n) < C \cdot g(n) \).

True or False?

(1) \( 5n \in O(n^2) \)
(2) \( 5n^2 \in O(n^2) \)
(3) \( 5n^3 \in O(n^2) \)
(4) \( 100n + 0.01n^2 \in O(n) \)
(5) \( 0.1n + 10n^2 + 37n^3 + 0.001n^4 \in O(?) \)
Time Complexity and Big O Notation

Big O Notation: let $f(n)$, $g(n)$ be two functions. We say $f(n) \in O(g(n))$ if there is a positive $C$ and an $n_0$ such that for all $n > n_0$, there is $f(n) < C \cdot g(n)$.

In the context of time complexity or space complexity, big O notation takes problem sizes as input...
Example (Breadth-First-Search)

Time and space complexities of BFS are high.

Q: if each node has \( b \) children & optimum is at depth \( d \), what are the time and space complexities of BFS? (tip: time=\#GeneratedNode, space=\#StoredNode)

A: time = \( O(b^d) \), space = \( O(b^d) \)

Recall \( b \) is branching factor, and \( d \) is depth of goal node.
End of review...
Properties of Uniform-Cost Search

UCS is complete in general.

UCS is optimal in general.

Complexity is not easy to characterize.

Q: if branching factor is $b$, optimal cost is $C$, each step cost is at least $e$, what’s the worst-case complexity?
Properties of Uniform-Cost Search

UCS is complete in general.

UCS is optimal in general.

Complexity is not easy to characterize.

Q: if branching factor is $b$, optimal cost is $C$, each step cost is at least $e$, what’s the worst-case complexity?  
A: $O(b^{1+C/e})$
Properties of Uniform-Cost Search

UCS is complete in general.

UCS is optimal in general.

Complexity is not easy to characterize.

Q: when is BFS = UCS in complexity?
Properties of Uniform-Cost Search

UCS is complete in general.

UCS is optimal in general.

Complexity is not easy to characterize.

Q: when is BFS = UCS in complexity?
A: when all paths have the same cost.
Overview of Uninformed Search

Uninformed search algorithms can only generate successors and distinguish goal and non-goal state.

Different strategies have different orders of node expansion.

- breadth-first
- uniform-cost
- depth-first
- depth-limited
- iterative-deepening
- bidirectional
Depth-First Search

Always expand the deepest node.
Depth-First Search

Always expand the deepest node.

DFS is implemented using a LIFO queue in frontier. (expand most recently generated node)
Properties of Depth-First Search

DFS is complete by graph-search, but not necessarily by tree-search.

Q: can you give an example?

Tree search can revisit a state, but graph search will not.
Properties of Depth-First Search

DFS is complete by graph-search, but not necessarily by tree-search.
Properties of Depth-First Search

DFS is complete by graph-search, but not necessarily by tree-search.

DFS is not optimal.
Properties of Depth-First Search

DFS is complete by graph-search, but not necessarily by tree-search.

DFS is not optimal.

DFS can have high time complexity, worst-case $O(b^m)$.

$b$ is branching factor, $m$ is maximum depth of any node.
Properties of Depth-First Search

DFS is complete by graph-search, but not necessarily by tree-search.

DFS is not optimal.

DFS can have high time complexity, worst-case $O(b^m)$.

DFS has low space complexity because nodes can be removed.
Properties of Depth-First Search

DFS is complete by graph-search, but not necessarily by tree-search.

DFS is not optimal.

DFS can have high time complexity, worst-case $O(b^m)$.

DFS has low space complexity, worst-case $O(bm)$.

$b$ is branching factor, $m$ is maximum depth of any node.
Overview of Uninformed Search

Uninformed search algorithms can only generate successors and distinguish goal and non-goal state.

Different strategies have different orders of node expansion.
- breadth-first
- uniform-cost
- depth-first
- depth-limited
- iterative-deepening
- bidirectional
Depth-Limit Search

DLS is depth-first search with a preset depth limit $L$.

It addresses the infinite-path problem of depth-first search.
Properties of Depth-Limit Search

DLS is not complete if \( L < d \).

DLS is not optimal if \( L > d \).

DLS has time complexity \( O(b^L) \), space complexity \( O(bL) \).
Choosing Depth Limit

One often choose depth limit by domain knowledge, e.g.,

1. there are only 20 cities in Romania, so $L = 20$
2. any two cities can be connected through 9 cities, so $L = 9$ (diameter)
Overview of Uninformed Search

Uninformed search algorithms can only generate successors and distinguish goal and non-goal state.

Different strategies have different orders of node expansion.

- breadth-first
- uniform-cost
- depth-first
- depth-limited
- iterative-deepening
- bidirectional
Iterative-Deepening Search

It applies depth-limit search with increasing limit, until a goal is found.
Properties of Iterative Deepening Search

IDS combines benefits of breadth-first search and depth-first search.

It is complete if branching factor is finite. [like breadth-first search]

It is not optimal.

Its time complexity is $O(b^d)$. [like breadth-first search]

Its space complexity is $O(bd)$. [like depth-first search]
Properties of Iterative Deepening Search

“In general, iterative-deepening is the preferred uninformed search method when the search space is large and the depth of the solution is unknown.”
Overview of Uninformed Search

Uninformed search algorithms can only generate successors and distinguish goal and non-goal state.

Different strategies have different orders of node expansion.

- breadth-first
- uniform-cost
- depth-first
- depth-limited
- iterative-deepening
- bidirectional
Bidirectional Search

Search from both initial and goal state. Hope they meet in the middle.
Properties of Bidirectional Search

It can be implemented by applying search algorithms from both sides, and stop when their frontiers overlap.
Properties of Bidirectional Search

It can be implemented by applying search algorithms from both sides, and stop when their frontiers overlap.

BS has much less time complexity in general, e.g., \(O(b^{d/2}) + O(b^{d/2}) \ll O(b^d)\)

BS is easier to implement if all actions in the state space are reversible.
We’re done with uninformed search!

Uninformed search algorithms can only generate successors and distinguish goal and non-goal state.

Different strategies have different orders of node expansion.

- breadth-first
- uniform-cost
- depth-first
- depth-limited
- iterative-deepening
- bidirectional
### Compare Uninformed Search Strategies

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening</th>
<th>Bidirectional (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Yes&lt;sup&gt;a,b&lt;/sup&gt;</td>
<td>No</td>
<td>No</td>
<td>Yes&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Yes&lt;sup&gt;a,d&lt;/sup&gt;</td>
</tr>
<tr>
<td>Time</td>
<td>$O(b^d)$</td>
<td>$O(b^{1+[C^*/\epsilon]})$</td>
<td>$O(b^m)$</td>
<td>$O(b^l)$</td>
<td>$O(b^d)$</td>
<td>$O(b^{d/2})$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^d)$</td>
<td>$O(b^{1+[C^*/\epsilon]})$</td>
<td>$O(bm)$</td>
<td>$O(bl)$</td>
<td>$O(bd)$</td>
<td>$O(b^{d/2})$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes&lt;sup&gt;c&lt;/sup&gt;</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes&lt;sup&gt;c&lt;/sup&gt;</td>
<td>Yes&lt;sup&gt;c,d&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

**Figure 3.21**  Evaluation of tree-search strategies. $b$ is the branching factor; $d$ is the depth of the shallowest solution; $m$ is the maximum depth of the search tree; $l$ is the depth limit. Superscript caveats are as follows: <sup>a</sup> complete if $b$ is finite; <sup>b</sup> complete if step costs $\geq \epsilon$ for positive $\epsilon$; <sup>c</sup> optimal if step costs are all identical; <sup>d</sup> if both directions use breadth-first search.