Search Strategies

Informatics 2D: Reasoning and Agents

Lecture 3

Adapted from slides provided by Dr Petros Papapanagiotou
Search strategies

A search strategy is defined by picking the order of node expansion.

- Nodes are taken from the **frontier**.
Evaluating search strategies

- **Completeness**: Does it always find a solution if one exists?
- **Time Complexity**: Number of nodes generated / expanded
- **Space Complexity**: Maximum number of nodes in memory
- **Optimality**: Does it always find a least-cost solution?

Time and space complexity are measured in terms of:

- $b$: Maximum branching factor of the search tree
- $d$: Depth of the least-cost solution
- $m$: Maximum depth of the state space (may be $\infty$)
Recall: Tree Search

**function** TREE-SEARCH(problem) returns a solution, or failure
initialize the frontier using the initial state of problem

**loop**
- **if** the frontier is empty then return failure
- choose a leaf node and remove it from the frontier
- **if** the node contains a goal state then return the corresponding solution
- expand the chosen node, adding the resulting nodes to the frontier

```
initialize the frontier using the initial state of problem
loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier
```
Repeated states

Failure to detect repeated states can turn a linear problem into an exponential one!
Graph search

Augment TREE-SEARCH with a new data-structure:

- the explored set (closed list), which remembers every expanded node
- newly expanded nodes already in explored set are discarded

```
function GRAPH-SEARCH(problem) returns a solution, or failure
initialize the frontier using the initial state of problem
initialize the explored set to be empty
loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
    only if not in the frontier or explored set
```
Breadth-first search
Breadth-first search

Expand **shallowest** unexpanded node

**Implementation:**
- *frontier* is a **FIFO** queue, i.e., new successors go at end
Breadth-first search

Expand **shallowest** unexpanded node

**Implementation:**

- **frontier** is a **FIFO** queue, i.e., new successors go at end
Breadth-first search

Expand **shallowest** unexpanded node

Implementation:
- *frontier* is a **FIFO** queue, i.e., new successors go at end
Breadth-first search

Expand **shallowest** unexpanded node

**Implementation:**
- `frontier` is a **FIFO** queue, i.e., new successors go at end
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure

node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0

if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)

frontier ← a FIFO queue with node as the only element

explored ← an empty set

loop do

if EMPTY?(frontier) then return failure

node ← POP(frontier) /* chooses the shallowest node in frontier */

add node.STATE to explored

for each action in problem.ACTIONS(node.STATE) do

child ← CHILD-NODE(problem, node, action)

if child.STATE is not in explored or frontier then

if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)

frontier ← INSERT(child, frontier)
Properties of breadth-first search

- Complete?
- Time complexity?
- Space complexity?
- Optimal?
Properties of breadth-first search

**Complete?**
Yes (if $b$ is finite)

**Time complexity?**

**Space complexity?**

**Optimal?**
## Properties of breadth-first search

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Complete?</strong></td>
<td>Yes (if $b$ is finite)</td>
</tr>
<tr>
<td><strong>Time complexity?</strong></td>
<td>$b + b^2 + b^3 + \ldots + b^d = O(b^d)$ (worst-case)</td>
</tr>
<tr>
<td><strong>Space complexity?</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Optimal?</strong></td>
<td></td>
</tr>
</tbody>
</table>
Properties of breadth-first search

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes (if ( b ) is finite)</td>
</tr>
<tr>
<td>Time complexity?</td>
<td>( b + b^2 + b^3 + \ldots + b^d = O(b^d) ) (worst-case)</td>
</tr>
<tr>
<td>Space complexity?</td>
<td>( O(b^d) ) (keeps every node in memory)</td>
</tr>
<tr>
<td>Optimal?</td>
<td>INF2D: REASONING AND AGENTS</td>
</tr>
</tbody>
</table>
Properties of breadth-first search

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes (if $b$ is finite)</td>
</tr>
<tr>
<td>Time complexity?</td>
<td>$b + b^2 + b^3 + \ldots + b^d = O(b^d)$ (worst-case)</td>
</tr>
<tr>
<td>Space complexity?</td>
<td>$O(b^d)$ (keeps every node in memory)</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes (if cost = 1 per step)</td>
</tr>
</tbody>
</table>
Properties of breadth-first search

<table>
<thead>
<tr>
<th><strong>Complete?</strong></th>
<th>Yes (if ( b ) is finite)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time complexity?</strong></td>
<td>( b + b^2 + b^3 + \ldots + b^d = O(b^d) ) (worst-case)</td>
</tr>
<tr>
<td><strong>Space complexity?</strong></td>
<td>( O(b^d) ) (keeps every node in memory)</td>
</tr>
<tr>
<td><strong>Optimal?</strong></td>
<td>Yes (if cost = 1 per step), then optimal solution is closest to start!</td>
</tr>
</tbody>
</table>
Properties of breadth-first search

| Complete? | Yes (if $b$ is finite) |
| Time complexity? | $b + b^2 + b^3 + \ldots + b^d = O(b^d)$ (worst-case) |
| Space complexity? | $O(b^d)$ (keeps every node in memory) |
| Optimal? | Yes (if cost = 1 per step) |

**Space** is the bigger problem (more than time)
<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 ms</td>
<td>107 kb</td>
</tr>
<tr>
<td>4</td>
<td>11,110</td>
<td>11 ms</td>
<td>10.6 MB</td>
</tr>
<tr>
<td>6</td>
<td>$10^6$</td>
<td>1.1 s</td>
<td>1 GB</td>
</tr>
<tr>
<td>8</td>
<td>$10^8$</td>
<td>2 min</td>
<td>103 GB</td>
</tr>
<tr>
<td>10</td>
<td>$10^{10}$</td>
<td>3 h</td>
<td>10 TB</td>
</tr>
<tr>
<td>12</td>
<td>$10^{12}$</td>
<td>13 d</td>
<td>1 PB</td>
</tr>
<tr>
<td>14</td>
<td>$10^{14}$</td>
<td>3.5 y</td>
<td>99 PB</td>
</tr>
<tr>
<td>16</td>
<td>$10^{16}$</td>
<td>350 y</td>
<td>10 EB</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.
Depth-first search
Depth-first search

Expand **deepest** unexpanded node

**Implementation:**

- **frontier** is a **LIFO** queue, i.e., new successors go at front
Depth-first search

Expand **deepest** unexpanded node

**Implementation:**
- *frontier* is a **LIFO** queue, i.e., new successors go at front
Depth-first search

Expand **deepest** unexpanded node

**Implementation:**
- *frontier* is a **LIFO** queue, i.e., new successors go at front
Depth-first search

Expand **deepest** unexpanded node

**Implementation:**
- *frontier* is a **LIFO** queue, i.e., new successors go at front
Depth-first search

Expand **deepest** unexpanded node

**Implementation:**
- `frontier` is a **LIFO** queue, i.e., new successors go at front
Depth-first search

Expand **deepest unexpanded node**

**Implementation:**
- *frontier* is a **LIFO** queue, i.e., new successors go at front
Depth-first search

Expand **deepest** unexpanded node

Implementation:
- *frontier* is a **LIFO** queue, i.e., new successors go at front
Depth-first search

Expand **deepest** unexpanded node

**Implementation:**
- *frontier* is a **LIFO** queue, i.e., new successors go at front
Depth-first search

Expand **deepest** unexpanded node

**Implementation:**
- *frontier* is a **LIFO** queue, i.e., new successors go at *front*
**Depth-first search**

Expand **deepest** unexpanded node

**Implementation:**
- *frontier* is a **LIFO** queue, i.e., new successors go at front
Properties of depth-first search

Complete?

Time complexity?

Space complexity?

Optimal?
Properties of depth-first search

- **Complete?**
  No: fails in infinite-depth spaces, spaces with loops

- **Time complexity?**

- **Space complexity?**

- **Optimal?**
## Properties of depth-first search

<table>
<thead>
<tr>
<th>Complete?</th>
</tr>
</thead>
<tbody>
<tr>
<td>No: fails in infinite-depth spaces, spaces with loops</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time complexity?</th>
</tr>
</thead>
<tbody>
<tr>
<td>avoid repeated states along path; complete in finite spaces</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Space complexity?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimal?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>
Properties of depth-first search

Complete?
No: fails in infinite-depth spaces, spaces with loops

Time complexity?
$O(b^m)$: terrible if $m$ is much larger than $d$

Space complexity?

Optimal?
## Properties of depth-first search

<table>
<thead>
<tr>
<th><strong>Complete?</strong></th>
<th>No: fails in infinite-depth spaces, spaces with loops</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time complexity?</strong></td>
<td>$O(b^m)$: terrible if $m$ is much larger than $d$</td>
</tr>
<tr>
<td><strong>Space complexity?</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Optimal?</strong></td>
<td></td>
</tr>
</tbody>
</table>

If solutions are dense, depth-first may be much faster than breadth-first!
## Properties of depth-first search

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Complete?</strong></td>
<td>No: fails in infinite-depth spaces, spaces with loops</td>
</tr>
<tr>
<td><strong>Time complexity?</strong></td>
<td>$O(b^m)$: terrible if $m$ is much larger than $d$</td>
</tr>
<tr>
<td><strong>Space complexity?</strong></td>
<td>$O(bm)$, i.e., linear space!</td>
</tr>
<tr>
<td><strong>Optimal?</strong></td>
<td></td>
</tr>
</tbody>
</table>
Properties of depth-first search

**Complete?**
No: fails in infinite-depth spaces, spaces with loops

**Time complexity?**
$O(b^m)$: terrible if $m$ is much larger than $d$

**Space complexity?**
$O(bm)$, i.e., linear space!

**Optimal?**
No
Mid-Lecture Exercise

BREADTH-FIRST

DEPTH-FIRST

INF2D: REASONING AND AGENTS
Mid-Lecture Exercise

**BREADTH-FIRST**
- When completeness is important.
- When optimal solutions are important.

**DEPTH-FIRST**
- When solutions are dense and low-cost is important, especially space costs.
**Depth-limited search**

This is depth-first search with depth limit \( l \), i.e., nodes at depth \( l \) have no successors.
### Properties of depth-limited tree search

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Complete?</strong></td>
<td>No</td>
</tr>
<tr>
<td><strong>Time complexity?</strong></td>
<td>$O(b^l)$</td>
</tr>
<tr>
<td><strong>Space complexity?</strong></td>
<td>$O(bl)$, i.e., linear space!</td>
</tr>
<tr>
<td><strong>Optimal?</strong></td>
<td>No</td>
</tr>
</tbody>
</table>
Iterative deepening search

... or how to improve depth-first search
Iterative deepening search

function ITERATIVE-DEEPENING-SEARCH(\textit{problem}) \textbf{returns} a solution, or failure
\begin{verbatim}
for depth = 0 to \infty do
    result \leftarrow\textsc{Depth-Limited-Search}(\textit{problem}, depth)
    if result \neq \textit{cutoff} then return result
\end{verbatim}
Iterative deepening search \( l = 0 \)
Iterative deepening search $l = 1$
Iterative deepening search $l = 2$
Iterative deepening search $l = 3$
Iterative deepening search

Number of nodes generated in an iterative deepening search to depth $d$ with branching factor $b$:

$$N_{IDS} = (d)b + (d-1)b^2 + \ldots + (2)b^{d-1} + (1)b^d$$

Some cost associated with generating upper levels multiple times

Example: For $b = 10$, $d = 5$,

- $N_{BFS} = 10 + 100 + 1,000 + 10,000 + 100,000 = 111,110$
- $N_{IDS} = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$

Overhead = $(123,450 - 111,110)/111,110 = 11\%$
Properties of iterative deepening search

- Complete?
- Time complexity?
- Space complexity?
- Optimal?
Properties of iterative deepening search

<table>
<thead>
<tr>
<th>Property</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes</td>
</tr>
<tr>
<td>Time complexity?</td>
<td></td>
</tr>
<tr>
<td>Space complexity?</td>
<td></td>
</tr>
<tr>
<td>Optimal?</td>
<td></td>
</tr>
</tbody>
</table>
### Properties of iterative deepening search

<table>
<thead>
<tr>
<th><strong>Complete?</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
</tr>
</tbody>
</table>

**Time complexity?**

\[(d)b + (d-1)b^2 + \ldots + (1)b^d = O(b^d)\]

**Space complexity?**

\n
**Optimal?**

\n
Properties of iterative deepening search

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes</td>
</tr>
<tr>
<td>Time complexity?</td>
<td>((d)b + (d-1)b^2 + \ldots + (1)b^d = O(b^d))</td>
</tr>
<tr>
<td>Space complexity?</td>
<td>(O(bd))</td>
</tr>
<tr>
<td>Optimal?</td>
<td></td>
</tr>
</tbody>
</table>
Properties of iterative deepening search

- **Complete?**
  Yes

- **Time complexity?**
  
  $$(d)b + (d-1)b^2 + \ldots + (1)b^d = O(b^d)$$

- **Space complexity?**
  
  $O(bd)$

- **Optimal?**
  Yes, if step cost = 1
## Summary of algorithms

<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? Time</td>
<td>Yes(^a)</td>
<td>Yes(^a, b)</td>
<td>No</td>
<td>No</td>
<td>Yes(^a)</td>
<td>Yes(^a, d) O(b^{d/2})</td>
</tr>
<tr>
<td>Space</td>
<td>O(b^d)</td>
<td>O(b^1+[C^*/\varepsilon])</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>Optimal?</td>
<td>Yes(^c)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes(^c)</td>
<td>Yes(^c, d)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Summary

Variety of *uninformed search* strategies:
- breadth-first, depth-first, depth-limited, iterative deepening

Iterative deepening search uses only *linear space* and *not much more time* than other uninformed algorithms
Why?

- Very common algorithms.
- Used whenever we are looking for a path in a tree or graph.
  - Anywhere from games to programming languages.
- Properties matter!
  - time and/or space complexity.
- Understanding which algorithm to use in what circumstances.