Uninformed Search

Chapter 3.1 – 3.4

Many AI Tasks can be Formulated as Search Problems

- Puzzles
- Games
- Navigation
- Assignment
- Layout
- Scheduling
- Routing

Search Example: Route Finding

Search Example: River Crossing Problem

Rules:
1) Farmer must row the boat
2) Only room for one other
3) Without the farmer present:
   - Dog bites sheep
   - Sheep eats cabbage
Search Example: 8-Puzzle

Start State

| 7 | 2 | 4 |
| 5 | 6 |   |
| 8 | 3 | 1 |

Goal State

| 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

Search Example: Water Jugs Problem

Given 4-liter and 3-liter pitchers, how do you get exactly 2 liters into the 4-liter pitcher?

Remove 5 Sticks Problem

Remove exactly 5 of the 17 sticks so the resulting figure forms exactly 3 squares.

Search Example: 8-Queens

Remove 5 Sticks Problem

Remove exactly 5 of the 17 sticks so the resulting figure forms exactly 3 squares.
Basic Search Task Assumptions (usually, though not games)

- Fully observable
- Deterministic
- Static
- Discrete
- Single agent

What Knowledge does the Agent Need?

- The information needs to be
  - sufficient to describe all relevant aspects for reaching the goal
  - adequate to describe the world state/situation
- Fully observable assumption, also known as the closed world assumption, means
  - All necessary information about a problem domain is accessible so that each state is a complete description of the world; there is no missing information at any point in time

How should the Environment be Represented?

- Knowledge representation problem:
  - What information from the sensors is relevant?
  - How to represent domain knowledge?
- Determining what to represent is difficult and is usually left to the system designer to specify
- Problem State = representation of all necessary information about the environment
- State Space (aka Problem Space) = all possible valid configurations of the environment

What Goal does the Agent want to Achieve?

- How do you describe the goal?
  - as a task to be accomplished
  - as a state to be reached
  - as a set of properties to be satisfied
- How do you know when the goal is reached?
  - with a goal test that defines what it means to have achieved/satisfied the goal
  - or, with a set of goal states
- Determining the goal is usually left to the system designer or user to specify
What Actions does the Agent Need?

- Discrete and Deterministic task assumptions imply

- Given:
  - an action (aka operator or move)
  - a description of the current state of the world

- Action completely specifies:
  - if that action can be applied (i.e., legal)
  - what the exact state of the world will be after the action is performed in the current state (no "history" information needed to compute the successor state)

Search Example: 8-Puzzle

- States = configurations
- Actions = up to 4 kinds of moves: up, down, left, right

Water Jugs Problem

Given 4-liter and 3-liter pitchers, how do you get exactly 2 liters into the 4-liter pitcher?

State: \((x, y)\) for # liters in 4-liter and 3-liter pitchers, respectively
Actions: empty, fill, pour water between pitchers
Initial state: \((0, 0)\)
Goal state: \((2, \ast)\)

Actions / Successor Functions

1. \((x, y | x < 4) \rightarrow (4, y)\)  \(\text{Fill 4}\)
2. \((x, y | y < 3) \rightarrow (x, 3)\)  \(\text{Fill 3}\)
3. \((x, y | x > 0) \rightarrow (0, y)\)  \(\text{Empty 4}\)
4. \((x, y | y > 0) \rightarrow (x, 0)\)  \(\text{Empty 3}\)
5. \((x, y | x+y \geq 4 \text{ and } y > 0) \rightarrow (4, y - (4 - x))\)  \(\text{Pour from 3 to 4 until 4 is full}\)
6. \((x, y | x+y \geq 3 \text{ and } x > 0) \rightarrow (x - (3 - y), 3)\)  \(\text{Pour from 4 to 3 until 3 is full}\)
7. \((x, y | x+y \leq 4 \text{ and } y > 0) \rightarrow (x+y, 0)\)  \(\text{Pour all water from 3 to 4}\)
Formalizing Search in a State Space

- A state space is a graph: \((V, E)\)
  - \(V\) is a set of nodes (vertices)
  - \(E\) is a set of arcs (edges)
  - each arc is directed from one node to another node

- Each node is a data structure that contains:
  - a state description
  - other information such as:
    - link to parent node
    - name of action that generated this node (from its parent)
    - other bookkeeping data

- Each arc corresponds to one of the finite number of actions:
  - when the action is applied to the state associated with the arc's source node
  - then the resulting state is the state associated with the arc's destination node

- Each arc has a fixed, positive cost:
  - corresponds to the cost of the action

Formalizing Search in a State Space

- Each node has a finite set of successor nodes:
  - corresponds to all of the legal actions that can be applied at the source node's state

- Expanding a node means:
  - generate all of the successor nodes
  - add them and their associated arcs to the state-space search tree

- One or more nodes are designated as start nodes

- A goal test is applied to a node's state to determine if it is a goal node

- A solution is a sequence of actions associated with a path in the state space from a start to a goal node:
  - just the goal state (e.g., cryptarithmetic)
  - a path from start to goal state (e.g., 8-puzzle)

- The cost of a solution is the sum of the arc costs on the solution path
Formalizing Search

A search problem has five components: 

1. **State space** $S$: all valid configurations 
2. **Initial states** $I \subseteq S$: a set of start states $I = \{(FCDS,)} \subseteq S$
3. **Goal states** $G \subseteq S$: a set of goal states $G = \{(FCDS,)} \subseteq S$
4. **An action function** $\text{successors}(s) \subseteq S$: states reachable in one step (one arc) from $s$
   
   $\text{successors}((FCDS,)) = \{(CD,FS), (D,FCS), (C,FSD)\}$
5. **A cost function** $\text{cost}(s, s')$: The cost of moving from $s$ to $s'$
   
   - The goal of search is to find a solution path from a state in $I$ to a state in $G$

Sizes of State Spaces

<table>
<thead>
<tr>
<th>Problem</th>
<th>Nodes</th>
<th>Brute-Force Search Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tic-Tac-Toe</td>
<td>$3^9$</td>
<td>.01 seconds</td>
</tr>
<tr>
<td>8 Puzzle</td>
<td>$10^5$</td>
<td>.2 seconds</td>
</tr>
<tr>
<td>2³ Rubik’s Cube</td>
<td>$10^6$</td>
<td>6 days</td>
</tr>
<tr>
<td>15 Puzzle</td>
<td>$10^{13}$</td>
<td>68,000 years</td>
</tr>
<tr>
<td>3³ Rubik’s Cube</td>
<td>$10^{19}$</td>
<td>12 billion years</td>
</tr>
<tr>
<td>24 Puzzle</td>
<td>$10^{25}$</td>
<td>12 billion years</td>
</tr>
<tr>
<td>Checkers</td>
<td>$10^{40}$</td>
<td></td>
</tr>
<tr>
<td>Chess</td>
<td>$10^{120}$</td>
<td></td>
</tr>
</tbody>
</table>

8-Puzzle State Space

(Ignoring “backwards” moves)

State Space = A Directed Graph

- In general there will be many generated, but unexpanded, states at any given time
- One has to choose which one to “expand” next
Different Search Strategies

- The generated, but not yet expanded, states define the **Frontier** (aka **Open** or **Fringe**) list.
- The essential difference is, **which one to expand first?**
- Deep or shallow?

Formalizing Search in a State Space

**State-space search** is the process of searching through a state space for a solution by making explicit a sufficient portion of an implicit state-space graph to include a goal node:

**TREE SEARCH Alg.**

```
Frontier = \{S\}, where S is the start node
Loop do
  if Frontier is empty then return failure
  pick a node, n, from Frontier
  if n is a goal node then return solution
  Generate all n’s successor nodes and add them all to Frontier
  Remove n from Frontier
```

Uninformed Search on Trees

- **Uninformed** means we *only* know:
  - The goal test
  - The **successors**() function
- But *not* which non-goal states are better
- For now, also assume state space graph is a **tree**
  - That is, we won’t encounter (or at least worry about) repeated states
  - We will relax this later
- **Search strategies differ by what un-expanded node is expanded next**
Uninformed Search Strategies

Uninformed Search: strategies that order nodes \textit{without} using any domain specific information, i.e., doesn't use any information stored in a state

- **BFS**: breadth-first search
  - Queue (FIFO) used for the Frontier list
  - remove from front, add to back

- **DFS**: depth-first search
  - Stack (LIFO) used for the Frontier list
  - remove from front, add to front

Breadth-First Search (BFS)

Expand the shallowest node first:
1. Examine states \textit{one} step away from the initial states
2. Examine states \textit{two} steps away from the initial states
3. and so on

Breadth-First Search (BFS)

\text{generalSearch}(\text{problem}, \text{queue})

- # of nodes tested: 0, expanded: 0
- \text{expnd. node} \quad \text{Frontier list}
  - S

\begin{array}{|c|c|}
  \hline
  \text{expnd. node} & \text{Frontier list} \\
  \hline
  S & \{S\} \\
  \hline
\end{array}

Breadth-First Search (BFS)

\text{generalSearch}(\text{problem}, \text{queue})

- # of nodes tested: 1, expanded: 1
- \text{expnd. node} \quad \text{Frontier list}
  - S
  - S not goal \quad \{A,B,C\}

\begin{array}{|c|c|}
  \hline
  \text{expnd. node} & \text{Frontier list} \\
  \hline
  S & \{S\} \\
  \hline
  S & \{A,B,C\} \\
  \hline
\end{array}
Breadth-First Search (BFS)

**generalSearch**(problem, queue)

# of nodes tested: 2, expanded: 2

<table>
<thead>
<tr>
<th>expnd. node</th>
<th>Frontier list</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>(A, B, C)</td>
</tr>
<tr>
<td>A not goal</td>
<td>(B, C, D, E)</td>
</tr>
</tbody>
</table>

Breadth-First Search (BFS)

**generalSearch**(problem, queue)

# of nodes tested: 3, expanded: 3

<table>
<thead>
<tr>
<th>expnd. node</th>
<th>Frontier list</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>(A, B, C)</td>
</tr>
<tr>
<td>A</td>
<td>(B, C, D, E)</td>
</tr>
<tr>
<td>B not goal</td>
<td>(C, D, E, F)</td>
</tr>
</tbody>
</table>

Breadth-First Search (BFS)

**generalSearch**(problem, queue)

# of nodes tested: 4, expanded: 4

<table>
<thead>
<tr>
<th>expnd. node</th>
<th>Frontier list</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>(A, B, C)</td>
</tr>
<tr>
<td>A</td>
<td>(B, C, D, E)</td>
</tr>
<tr>
<td>B</td>
<td>(C, D, E, G)</td>
</tr>
<tr>
<td>C not goal</td>
<td>(D, E, F, G)</td>
</tr>
</tbody>
</table>

Breadth-First Search (BFS)

**generalSearch**(problem, queue)

# of nodes tested: 5, expanded: 5

<table>
<thead>
<tr>
<th>expnd. node</th>
<th>Frontier list</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>(A, B, C)</td>
</tr>
<tr>
<td>A</td>
<td>(B, C, D, E)</td>
</tr>
<tr>
<td>B</td>
<td>(C, D, E, G)</td>
</tr>
<tr>
<td>C</td>
<td>(D, E, G, F)</td>
</tr>
<tr>
<td>D not goal</td>
<td>(E, G, F, H)</td>
</tr>
</tbody>
</table>
Breadth-First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 7, expanded: 6

<table>
<thead>
<tr>
<th>expnd. node</th>
<th>Frontier list</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>{A, B, C}</td>
</tr>
<tr>
<td>A</td>
<td>{B, C, D, E}</td>
</tr>
<tr>
<td>B</td>
<td>{C, D, E, G}</td>
</tr>
<tr>
<td>C</td>
<td>{D, E, G, F}</td>
</tr>
<tr>
<td>D</td>
<td>{E, G, F, H}</td>
</tr>
<tr>
<td>E (not goal)</td>
<td>{G, F, H, G}</td>
</tr>
</tbody>
</table>

Breadth-First Search (BFS)

`generalSearch(problem, queue)`

# of nodes tested: 7, expanded: 6

<table>
<thead>
<tr>
<th>expnd. node</th>
<th>Frontier list</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>{A, B, C}</td>
</tr>
<tr>
<td>A</td>
<td>{B, C, D, E}</td>
</tr>
<tr>
<td>B</td>
<td>{C, D, E, G}</td>
</tr>
<tr>
<td>C</td>
<td>{D, E, G, F}</td>
</tr>
<tr>
<td>D</td>
<td>{E, G, F, H}</td>
</tr>
<tr>
<td>E (not goal)</td>
<td>{G, F, H, G}</td>
</tr>
<tr>
<td>G (goal)</td>
<td>{F, H, G}</td>
</tr>
</tbody>
</table>

Evaluating Search Strategies

- **Completeness**
  If a solution exists, will it be found?
  - a complete algorithm will find a solution (not all)

- **Optimality / Admissibility**
  If a solution is found, is it guaranteed to be optimal?
  - an admissible algorithm will find a solution with minimum cost
Evaluating Search Strategies

- **Time Complexity**
  - How long does it take to find a solution?
  - Usually measured for worst case
  - Measured by counting number of nodes expanded

- **Space Complexity**
  - How much space is used by the algorithm?
  - Measured in terms of the maximum size of the Frontier list during the search

What’s in the Frontier (Queue) for BFS?

- If goal is at depth $d$, how big is the frontier (worst case)?

Breadth-First Search (BFS)

- **Complete**

- **Optimal / Admissible**
  - Yes, if all operators (i.e., arcs) have the same constant cost, or costs are positive, non-decreasing with depth
  - Otherwise, not optimal but does guarantee finding solution of shortest length (i.e., fewest arcs)

- **Time and space complexity: $O(b^d)$ (i.e., exponential)**
  - $d$ is the depth of the solution
  - $b$ is the branching factor at each non-leaf node

- Very slow to find solutions with a large number of steps because must look at all shorter length possibilities first
Breadth-First Search (BFS)

- A complete search tree has a total # of nodes = \(1 + b + b^2 + \ldots + b^d = (b^{d+1} - 1)/(b-1)\)
  - \(d\): the tree’s depth
  - \(b\): the branching factor at each non-leaf node
- For example: \(d = 12, b = 10\)
  
  \[1 + 10 + 100 + \ldots + 10^{12} = (10^{13} - 1)/9 = O(10^{12})\]
  - If BFS expands 1,000 nodes/sec and each node uses 100 bytes of storage, then BFS will take 35 years to run in the worst case, and it will use 111 terabytes of memory!

Depth-First Search (DFS)

\[
\text{generalSearch}(\text{problem, stack})
\]

# of nodes tested: 0, expanded: 0

<table>
<thead>
<tr>
<th>expnd. node</th>
<th>Frontier list</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{S}</td>
</tr>
</tbody>
</table>

Depth-First Search (DFS)

\[
\text{generalSearch}(\text{problem, stack})
\]

# of nodes tested: 1, expanded: 1

<table>
<thead>
<tr>
<th>expnd. node</th>
<th>Frontier list</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>{A,B,C}</td>
</tr>
<tr>
<td>S not goal</td>
<td>{A,B,C}</td>
</tr>
</tbody>
</table>
Depth-First Search (DFS)

<table>
<thead>
<tr>
<th>expnd. node</th>
<th>Frontier list</th>
</tr>
</thead>
<tbody>
<tr>
<td>[S]</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>(A,B,C)</td>
</tr>
<tr>
<td>A not goal</td>
<td>(E,B,C)</td>
</tr>
</tbody>
</table>

Depth-First Search (DFS)

<table>
<thead>
<tr>
<th>expnd. node</th>
<th>Frontier list</th>
</tr>
</thead>
<tbody>
<tr>
<td>[S]</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>(A,B,C)</td>
</tr>
<tr>
<td>A</td>
<td>(D,E,B,C)</td>
</tr>
<tr>
<td>D not goal</td>
<td>(E,B,C)</td>
</tr>
</tbody>
</table>

Depth-First Search (DFS)

<table>
<thead>
<tr>
<th>expnd. node</th>
<th>Frontier list</th>
</tr>
</thead>
<tbody>
<tr>
<td>[S]</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>(A,B,C)</td>
</tr>
<tr>
<td>A</td>
<td>(D,E,B,C)</td>
</tr>
<tr>
<td>H not goal</td>
<td>(E,B,C)</td>
</tr>
</tbody>
</table>

Depth-First Search (DFS)

<table>
<thead>
<tr>
<th>expnd. node</th>
<th>Frontier list</th>
</tr>
</thead>
<tbody>
<tr>
<td>[S]</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>(A,B,C)</td>
</tr>
<tr>
<td>A</td>
<td>(D,E,B,C)</td>
</tr>
<tr>
<td>H</td>
<td>(E,B,C)</td>
</tr>
<tr>
<td>E not goal</td>
<td>(E,B,C)</td>
</tr>
</tbody>
</table>

Depth-First Search (DFS)

<table>
<thead>
<tr>
<th>expnd. node</th>
<th>Frontier list</th>
</tr>
</thead>
<tbody>
<tr>
<td>[S]</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>(A,B,C)</td>
</tr>
<tr>
<td>A</td>
<td>(D,E,B,C)</td>
</tr>
<tr>
<td>H</td>
<td>(E,B,C)</td>
</tr>
<tr>
<td>E</td>
<td>(E,B,C)</td>
</tr>
</tbody>
</table>
Depth-First Search (DFS)

generalSearch(problem, stack)

# of nodes tested: 6, expanded: 5

<table>
<thead>
<tr>
<th>expnd. node</th>
<th>Frontier list</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>{A,B,C}</td>
</tr>
<tr>
<td>A</td>
<td>{D,E,B,C}</td>
</tr>
<tr>
<td>D</td>
<td>{H,E,B,C}</td>
</tr>
<tr>
<td>H</td>
<td>{E,B,C}</td>
</tr>
<tr>
<td>E</td>
<td>{G,B,C}</td>
</tr>
<tr>
<td>G (goal)</td>
<td>{B,C} no expand</td>
</tr>
</tbody>
</table>

Depth-First Search (DFS)

May not terminate without a depth bound
i.e., cutting off search below a fixed depth, $D$

Not complete
- with or without cycle detection
- and, with or without a depth cutoff

Not optimal / admissible

Can find long solutions quickly if lucky

Depth-First Search (DFS)

Time complexity: $O(b^d)$ exponential
Space complexity: $O(b^d)$ linear
- $d$ is the depth of the solution
- $b$ is the branching factor at each non-leaf node

Performs "chronological backtracking"
- i.e., when search hits a dead end, backs up one level at a time
- problematic if the mistake occurs because of a bad action choice near the top of search tree

Depth-First Search (DFS)

path: S,A,E,G cost: 15
Uniform-Cost Search (UCS)

- Use a “Priority Queue” to order nodes on the Frontier list, sorted by path cost.
- Let $g(n) = \text{cost of path from start node } s \text{ to current node } n$.
- Sort nodes by increasing value of $g$.

**Diagrams:**
1. General Search (problem, priorityQueue)
   - # of nodes tested: 0, expanded: 0
   - Frontier list:
     - [S]
2. Uniform-Cost Search (UCS)
   - expnd. node | Frontier list
     - [S:0]
     - S not goal: [B:2,C:4,A:5]
3. General Search (problem, priorityQueue)
   - # of nodes tested: 1, expanded: 1
   - Frontier list:
     - [S:0]
     - S not goal: [B:2,C:4,A:5]
4. Uniform-Cost Search (UCS)
   - expnd. node | Frontier list
     - [S:0]
     - S not goal: [B:2,C:4,A:5]
8. General Search (problem, priorityQueue)
   - # of nodes tested: 2, expanded: 2
   - Frontier list:
     - [S:0]
     - S: [B:2,C:4,A:5]
     - B: [C:4,A:5,G:2+6]
Uniform-Cost Search (UCS)

generalSearch(problem, priorityQueue)
# of nodes tested: 3, expanded: 3

expnd. node  Frontier list
[S]            
S  (B:2,C:4,A:5)
B  (C:4,A:5,G:8)
C not goal (A:5,F:4+2,G:8)

Uniform-Cost Search (UCS)

generalSearch(problem, priorityQueue)
# of nodes tested: 4, expanded: 4

expnd. node  Frontier list
[S]            
S  (B:2,C:4,A:5)
B  (C:4,A:5,G:8)
C  (A:5,F:6,G:8)
A not goal (F:6,G:8,E:5+4, D:5+9)

Uniform-Cost Search (UCS)

generalSearch(problem, priorityQueue)
# of nodes tested: 5, expanded: 5

expnd. node  Frontier list
[S]            
S  (B:2,C:4,A:5)
B  (C:4,A:5,G:8)
C  (A:5,F:6,G:8)
A  (F:6,G:8,E:9,D:14)
F not goal (G:4+2+1,G:8,E:9, D:14)

Uniform-Cost Search (UCS)

generalSearch(problem, priorityQueue)
# of nodes tested: 6, expanded: 5

expnd. node  Frontier list
[S]            
S  (B:2,C:4,A:5)
B  (C:4,A:5,G:8)
C  (A:5,F:6,G:8)
A  (F:6,G:8,E:9,D:14)
F  (G:7,G:8,E:9,D:14)
G goal (G:8,E:9,D:14) no expand

# of nodes tested: 5, expanded: 5

expnd. node  Frontier list
[S]            
S  (B:2,C:4,A:5)
B  (C:4,A:5,G:8)
C  (A:5,F:6,G:8)
A  (F:6,G:8,E:9,D:14)
F  (G:7,G:8,E:9,D:14)
G goal (G:8,E:9,D:14) no expand
Uniform-Cost Search (UCS)

\[ \text{generalSearch(problem, priorityQueue)} \]

\# of nodes tested: 6, expanded: 5

<table>
<thead>
<tr>
<th>expnd. node</th>
<th>Frontier list</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>(B:2,C:4,A:5)</td>
</tr>
<tr>
<td>B</td>
<td>(C:4,A:5,G:8)</td>
</tr>
<tr>
<td>C</td>
<td>(A:5,F:6,G:8)</td>
</tr>
<tr>
<td>A</td>
<td>(F:6,G:8,E:9,D:14)</td>
</tr>
<tr>
<td>F</td>
<td>(G:7,G:8,E:9,D:14)</td>
</tr>
<tr>
<td>G</td>
<td>(G:8,E:9,D:14)</td>
</tr>
</tbody>
</table>

Uniform-Cost Search (UCS)

- Called Dijkstra’s Algorithm in the algorithms literature
- Similar to Branch and Bound Algorithm in Operations Research literature
- Complete
- Optimal / Admissible
  - requires that the goal test is done when a node is removed from the Frontier list rather than when the node is generated by its parent node

Uniform-Cost Search (UCS)

- Time and space complexity: \( O(b^d) \) (i.e., exponential)
  - \( d \) is the depth of the solution
  - \( b \) is the branching factor at each non-leaf node
- More precisely, time and space complexity is \( O(b^{C^*}) \) where all edge costs \( \geq \epsilon > 0 \), and \( C^* \) is the best goal path cost

Iterative-Deepening Search (IDS)

- requires modification to DFS search algorithm:
  - do DFS to depth 1 and treat all children of the start node as leaves
  - if no solution found, do DFS to depth 2
  - repeat by increasing “depth bound” until a solution found
- Start node is at depth 0
Iterative-Deepening Search (IDS)

depth = 1, # of nodes expanded: 0, tested: 0

expn. node Frontier list
(S)

Iterative-Deepening Search (IDS)

depth = 1, # of nodes tested: 1, expanded: 1

expn. node Frontier list
(S)
S not goal (A,B,C)

Iterative-Deepening Search (IDS)

depth = 1, # of nodes tested: 2, expanded: 1

expn. node Frontier list
(S)
S {A,B,C}
A not goal (B,C) no expand

Iterative-Deepening Search (IDS)

depth = 1, # of nodes tested: 3, expanded: 1

expn. node Frontier list
(S)
S {A,B,C}
A {B,C} no expand

S not goal (C) no expand
Iterative-Deepening Search (IDS)

depth: 1, # of nodes tested: 4, expanded: 1
expnd. node | Frontier list
---|---
S (A,B,C) | S
A (B,C) | A
B (C) | B
C not goal | C

Expanding node A
Frontier list: S, A, B, C

Expanding node B
Frontier list: S, A, B, C

Expanding node C
Frontier list: S, A, B, C

Iterative-Deepening Search (IDS)

depth: 2, # of nodes tested: 4(1), expanded: 2
expnd. node | Frontier list
---|---
S (A,B,C) | S
A (B,C) | A
B (C) | B
C | C

Expanding node D
Frontier list: S, A, B, C, D

Expanding node E
Frontier list: S, A, B, C, D, E

Expanding node F
Frontier list: S, A, B, C, D, E, F

Iterative-Deepening Search (IDS)

depth: 2, # of nodes tested: 4(2), expanded: 3
expnd. node | Frontier list
---|---
S (A,B,C) | S
A (B,C) | A
B (C) | B
C | C

Expanding node G
Frontier list: S, A, B, C, D, E, F, G

Expanding node H
Frontier list: S, A, B, C, D, E, F, G, H

Iterative-Deepening Search (IDS)

depth: 2, # of nodes tested: 5(2), expanded: 3
expnd. node | Frontier list
---|---
S (A,B,C) | S
A (B,C) | A
B (C) | B
C | C

Expanding node G
Frontier list: S, A, B, C, D, E, F, G, H

Expanding node D
Frontier list: S, A, B, C, D, E, F, G, H, D

Expanding node E
Frontier list: S, A, B, C, D, E, F, G, H, D, E

Expanding node F
Frontier list: S, A, B, C, D, E, F, G, H, D, E, F

Expanding node G
Frontier list: S, A, B, C, D, E, F, G, H, D, E, F, G

Expanding node H
Frontier list: S, A, B, C, D, E, F, G, H, D, E, F, G, H

Expanding node I
Frontier list: S, A, B, C, D, E, F, G, H, D, E, F, G, H, I

Expanding node J
Frontier list: S, A, B, C, D, E, F, G, H, D, E, F, G, H, I, J
Iterative-Deepening Search (IDS)

deepeningSearch(problem)
depth: 2, # of nodes tested: 6(2), expanded: 3
expnd. node Frontier list
(S) S
A (A,B,C) A
B (C) B
C () C
S (A,B,C) S
A (D,E,B,C) A
D (E,B,C) D
E not goal (B,C) no expand E

Iterative-Deepening Search (IDS)

deepeningSearch(problem)
depth: 2, # of nodes tested: 6(3), expanded: 4
expnd. node Frontier list
(S) S
A (A,B,C) A
B (C) B
C () C
S (A,B,C) S
A (D,E,B,C) A
D (E,B,C) D
E (B,C) E
B no test (G,C) B

Iterative-Deepening Search (IDS)

deepeningSearch(problem)
depth: 2, # of nodes tested: 7(3), expanded: 4
expnd. node Frontier list
(S) S
A (A,B,C) A
B (C) B
C () C
S (A,B,C) S
A (D,E,B,C) A
D (E,B,C) D
E (B,C) E
G goal (C) G

Iterative-Deepening Search (IDS)

deepeningSearch(problem)
depth: 2, # of nodes tested: 7(3), expanded: 4
expnd. node Frontier list
(S) S
A (A,B,C) A
B (C) B
C () C
S (A,B,C) S
A (D,E,B,C) A
D (E,B,C) D
E (B,C) E
B no test (G,C) B

path: S,B,G cost: 8
Iterative-Deepening Search (IDS)

- Has advantages of BFS
  - completeness
  - optimality as stated for BFS

- Has advantages of DFS
  - limited space
  - in practice, even with redundant effort it still finds longer paths more quickly than BFS

Iterative-Deepening Search (IDS)

- Space complexity: \( O(bd) \) (i.e., linear like DFS)

- Time complexity is a little worse than BFS or DFS
  - because nodes near the top of the search tree are generated multiple times (redundant effort)

- Worst case time complexity: \( O(b^d) \) exponential
  - because most nodes are near the bottom of tree

Iterative-Deepening Search (IDS)

How much redundant effort is done?

- The number of times the nodes are generated:
  \[ 1b^d + 2b^{(d-1)} + \ldots + db \leq b^d / (1 - 1/b)^2 = O(b^d) \]
  - \( d \): the solution's depth
  - \( b \): the branching factor at each non-leaf node

- For example: \( b = 4 \)
  \[ 4^d / (1 - \frac{1}{4})^2 = 4^d / (.75)^2 = 1.78 \times 4^d \]
  - in the worst case, 78% more nodes are searched (redundant effort) than exist at depth \( d \)
  - as \( b \) increases, this % decreases

Bidirectional Search

- Breadth-first search from both start and goal
- Frontiers meet
- Generates \( O(b^{d/2}) \) instead of \( O(b^d) \) nodes
Which Direction Should We Search?

Our choices: Forward, backwards, or bidirectional

The issues:
- How many start and goal states are there?
- Branching factors in each direction
- How much work is it to compare states?

Performance of Search Algorithms on Trees

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth-first search</td>
<td>Y</td>
<td>Y, if 1</td>
<td>O(b^d)</td>
<td>O(b^d)</td>
</tr>
<tr>
<td>Uniform-cost search</td>
<td>Y</td>
<td>Y</td>
<td>O(b^c/ε)</td>
<td>O(b^c/ε)</td>
</tr>
<tr>
<td>Depth-first search</td>
<td>N</td>
<td>N</td>
<td>O(b^m)</td>
<td>O(bm)</td>
</tr>
<tr>
<td>Iterative deepening search</td>
<td>Y</td>
<td>Y, if 1</td>
<td>O(b^d)</td>
<td>O(bd)</td>
</tr>
<tr>
<td>Bidirectional search</td>
<td>Y</td>
<td>Y, if 1</td>
<td>O(b^d/ε)</td>
<td>O(b^d/ε)</td>
</tr>
</tbody>
</table>

1. edge cost constant, or positive non-decreasing in depth
2. edge costs ≥ ε > 0. C* is the best goal path cost
3. both directions BFS; not always feasible

If State Space is Not a Tree

- The problem: repeated states

<table>
<thead>
<tr>
<th>Edge</th>
<th>State 1</th>
<th>State 2</th>
<th>State 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>DSF</td>
<td>CFS</td>
<td>SF</td>
</tr>
<tr>
<td>D</td>
<td>CSF</td>
<td>D</td>
<td>SF</td>
</tr>
<tr>
<td>S</td>
<td>CSF</td>
<td>D</td>
<td>CFS</td>
</tr>
<tr>
<td>F</td>
<td>CSF</td>
<td>CFS</td>
<td>SF</td>
</tr>
</tbody>
</table>

- Ignoring repeated states: wasteful (BFS) or impossible (DFS). Why?
- How to prevent these problems?

If State Space is Not a Tree

- We have to remember already-expanded states (called Exploded (aka Closed) list too)

- Why?

- When we pick a state from Frontier
  - Remove it from Frontier
  - Add it to Exploded
  - Expand node, generating all successors
  - For each successor, child,
    - If child is in Exploded, throw child away
    - Otherwise, check whether child is in Frontier
      - If no, add it to Frontier
      - If yes and path-cost(child) < path-cost of node already in Frontier, then replace that Frontier node with child
How are nodes expanded by:

- Depth First Search: S A D E G
  Solution found: S A G

- Breadth First Search: S A B C D E G
  Solution found: S A G

- Uniform Cost Search: S A D B C E G
  Solution found: S B G

- Iterative Deepening Search: S A B C S A D E G
  Solution found: S A G

Are the solutions the same?