Blind (Uninformed) Search
(Where we systematically explore alternatives)

R&N: Chap. 3, Sect. 3.3-5

Acknowledgment:
These materials here are adopted from the ppts by © Dr. Jean-Claude Latombe at Stanford University, used with permission. ai.stanford.edu/~latombe
Simple Problem-Solving-Agent
Agent Algorithm

1. $s_0 \leftarrow \text{sense/read initial state}$
2. $\text{GOAL?} \leftarrow \text{select/read goal test}$
3. $\text{Succ} \leftarrow \text{read successor function}$
4. $\text{solution} \leftarrow \text{search}(s_0, \text{GOAL?}, \text{Succ})$
5. $\text{perform(solution)}$
Search Tree

Note that some states may be visited multiple times
Search Nodes and States

8 2
3 4 7
5 1 6

8 2 7
3 4
5 1 6

8 2 1
3 4 7
5 1 6

8 4 2
3 7
5 1 6

8 2
3 4 7
5 1 6
If states are allowed to be revisited, the search tree may be infinite even when the state space is finite.
Data Structure of a Node

Depth of a node $N$ = length of path from root to $N$
(depth of the root = 0)
Node expansion

The expansion of a node \( N \) of the search tree consists of:

1) Evaluating the successor function on \( \text{STATE}(N) \)

2) Generating a child of \( N \) for each state returned by the function

\textit{node generation} \( \neq \) \textit{node expansion}
The **fringe** is the set of all search nodes that haven’t been expanded yet.
Is it identical to the set of leaves?
Search Strategy

- The fringe is the set of all search nodes that haven’t been expanded yet.
- The fringe is implemented as a priority queue FRINGE:
  - INSERT(node,FRINGE)
  - REMOVE(FRINGE)
- The ordering of the nodes in FRINGE defines the search strategy.
Search Algorithm #1

**SEARCH#1**

1. If \(\text{GOAL?}(\text{initial-state})\) then return initial-state
2. \(\text{INSERT}(\text{initial-node}, \text{FRINGE})\)
3. Repeat:
   a. If empty(FRINGE) then return failure
   b. \(N \leftarrow \text{REMOVE}(\text{FRINGE})\)  
      Expansion of \(N\)
   c. \(s \leftarrow \text{STATE}(N)\)
   d. For every state \(s'\) in \(\text{SUCCESSORS}(s)\)
      i. Create a new node \(N'\) as a child of \(N\)
      ii. If \(\text{GOAL?}(s')\) then return path or goal state
      iii. \(\text{INSERT}(N', \text{FRINGE})\)
Performance Measures

- **Completeness**
  A search algorithm is complete if it finds a solution whenever one exists
  [What about the case when no solution exists?]

- **Optimality**
  A search algorithm is optimal if it returns a minimum-cost path whenever a solution exists

- **Complexity**
  It measures the time and amount of memory required by the algorithm
Blind vs. Heuristic Strategies

- **Blind (or un-informed)** strategies do not exploit state descriptions to order FRINGE. They only exploit the positions of the nodes in the search tree.

- **Heuristic (or informed)** strategies exploit state descriptions to order FRINGE (the most “promising” nodes are placed at the beginning of FRINGE).
Example

For a blind strategy, \( N_1 \) and \( N_2 \) are just two nodes (at some position in the search tree).

\[
\begin{array}{ccc}
8 & 2 & \text{STATE} \\
3 & 4 & 7 \\
5 & 1 & 6 \\
\end{array}
\]

\[ N_1 \]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & \text{STATE} \\
7 & 8 & 6 \\
\end{array}
\]

\[ N_2 \]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & \text{Goal state} \\
\end{array}
\]
Example

For a heuristic strategy counting the number of misplaced tiles, $N_2$ is more promising than $N_1$.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Goal state
Remark

- Some search problems, such as the \((n^2-1)\)-puzzle, are NP-hard
- One can’t expect to solve all instances of such problems in less than exponential time (in \(n\))
- One may still strive to solve each instance as efficiently as possible

\(\Rightarrow\) This is the purpose of the search strategy
Blind Strategies

- **Breadth-first**
  - Bidirectional

- **Depth-first**
  - Depth-limited
  - Iterative deepening

- **Uniform-Cost**
  - (variant of breadth-first)

\[ \text{Arc cost} = 1 \]

\[ \text{Arc cost} = c(\text{action}) \geq \varepsilon > 0 \]
Breadth-First Strategy

New nodes are inserted at the end of FRINGE

FRINGE = (1)
Breadth-First Strategy

New nodes are inserted at the end of FRINGE

FRINGE = (2, 3)
Breadth-First Strategy

New nodes are inserted at the end of FRINGE

FRINGE = (3, 4, 5)
Breadth-First Strategy

New nodes are inserted at the end of FRINGE

FRINGE = (4, 5, 6, 7)
Important Parameters

1) Maximum number of successors of any state
   \rightarrow \text{branching factor } b \text{ of the search tree}

2) Minimal length (≠ cost) of a path between the initial and a goal state
   \rightarrow \text{depth } d \text{ of the shallowest goal node in the search tree}
Evaluation

- \( b \): branching factor
- \( d \): depth of shallowest goal node
- Breadth-first search is:
  - Complete? Not complete?
  - Optimal? Not optimal?
Evaluation

- $b$: branching factor
- $d$: depth of shallowest goal node
- Breadth-first search is:
  - Complete
  - Optimal if step cost is 1
- Number of nodes generated: ???
Evaluation

- **b**: branching factor
- **d**: depth of shallowest goal node
- **Breadth-first search is:**
  - *Complete*
  - *Optimal* if step cost is 1
- **Number of nodes generated:**
  \[1 + b + b^2 + ... + b^d = ???\]
Evaluation

- **b**: branching factor
- **d**: depth of shallowest goal node
- Breadth-first search is:
  - *Complete*
  - *Optimal* if step cost is 1
- Number of nodes generated:
  \[1 + b + b^2 + \ldots + b^d = \frac{b^{d+1} - 1}{b - 1} = O(b^d)\]
- → Time and space complexity is \(O(b^d)\)
Big O Notation

\[ g(n) = O(f(n)) \text{ if there exist two positive constants } a \text{ and } N \text{ such that:} \]

\[ \text{for all } n > N: \quad g(n) \leq a \times f(n) \]
### Time and Memory Requirements

<table>
<thead>
<tr>
<th>d</th>
<th># Nodes</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>111</td>
<td>.01 m sec</td>
<td>11 K bytes</td>
</tr>
<tr>
<td>4</td>
<td>11,111</td>
<td>1 m sec</td>
<td>1 M byte</td>
</tr>
<tr>
<td>6</td>
<td>$\sim10^6$</td>
<td>1 sec</td>
<td>100 M Mb</td>
</tr>
<tr>
<td>8</td>
<td>$\sim10^8$</td>
<td>100 sec</td>
<td>10 G bytes</td>
</tr>
<tr>
<td>10</td>
<td>$\sim10^{10}$</td>
<td>2.8 hours</td>
<td>1 T byte</td>
</tr>
<tr>
<td>12</td>
<td>$\sim10^{12}$</td>
<td>11.6 days</td>
<td>100 T bytes</td>
</tr>
<tr>
<td>14</td>
<td>$\sim10^{14}$</td>
<td>3.2 years</td>
<td>10,000 T bytes</td>
</tr>
</tbody>
</table>

**Assumptions:** $b = 10$; 1,000,000 nodes/sec; 100 bytes/node
# Time and Memory Requirements

<table>
<thead>
<tr>
<th>$d$</th>
<th># Nodes</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>111</td>
<td>.01 msec</td>
<td>11 Kbytes</td>
</tr>
<tr>
<td>4</td>
<td>11,111</td>
<td>1 msec</td>
<td>1 Mbyte</td>
</tr>
<tr>
<td>6</td>
<td>$\sim 10^6$</td>
<td>1 sec</td>
<td>100 Mb</td>
</tr>
<tr>
<td>8</td>
<td>$\sim 10^8$</td>
<td>100 sec</td>
<td>10 Gbytes</td>
</tr>
<tr>
<td>10</td>
<td>$\sim 10^{10}$</td>
<td>2.8 hours</td>
<td>1 Tbyte</td>
</tr>
<tr>
<td>12</td>
<td>$\sim 10^{12}$</td>
<td>11.6 days</td>
<td>100 Tbytes</td>
</tr>
<tr>
<td>14</td>
<td>$\sim 10^{14}$</td>
<td>3.2 years</td>
<td>10,000 Tbytes</td>
</tr>
</tbody>
</table>

Assumptions: $b = 10$; 1,000,000 nodes/sec; 100 bytes/node
Remark

If a problem has no solution, breadth-first may run for ever (if the state space is infinite or states can be revisited arbitrary many times)
Bidirectional Strategy

2 fringe queues: FRINGE1 and FRINGE2

Time and space complexity is $O(b^{d/2}) << O(b^d)$ if both trees have the same branching factor $b$

Question: What happens if the branching factor is different in each direction?
Depth-First Strategy

New nodes are inserted at the front of FRINGE

FRINGE = (1)
Depth-First Strategy

New nodes are inserted at the front of FRINGE

FRINGE = (2, 3)
Depth-First Strategy

New nodes are inserted at the front of FRINGE

FRINGE = (4, 5, 3)
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Depth-First Strategy

New nodes are inserted at the front of FRINGE
Evaluation

- \( b \): branching factor
- \( d \): depth of shallowest goal node
- \( m \): maximal depth of a leaf node
- Depth-first search is:
  - Complete?
  - Optimal?
Evaluation

- \( b \): branching factor
- \( d \): depth of shallowest goal node
- \( m \): maximal depth of a leaf node
- Depth-first search is:
  - Complete only for finite search tree
  - Not optimal
- Number of nodes generated (worst case):
  \[ 1 + b + b^2 + \ldots + b^m = O(b^m) \]
- Time complexity is \( O(b^m) \)
- Space complexity is \( O(bm) \) [or \( O(m) \)]

[Reminder: Breadth-first requires \( O(b^d) \) time and space]
Depth-Limited Search

- Depth-first with depth cutoff $k$ (depth at which nodes are not expanded)

- Three possible outcomes:
  - Solution
  - Failure (no solution)
  - Cutoff (no solution within cutoff)
Iterative Deepening Search

Provides the best of both breadth-first and depth-first search

Main idea: Totally horrifying!

IDS
For $k = 0, 1, 2, \ldots$ do:
- Perform depth-first search with depth cutoff $k$
  (i.e., only generate nodes with depth $\leq k$)
Iterative Deepening
Iterative Deepening
Iterative Deepening
Performance

- Iterative deepening search is:
  - Complete
  - Optimal if step cost = 1
- Time complexity is:
  \[(d+1)(1) + db + (d-1)b^2 + \ldots + (1) b^d = O(b^d)\]
- Space complexity is: \(O(bd)\) or \(O(d)\)
Calculation

db + (d-1)b^2 + ... + (1) b^d
= b^d + 2b^{d-1} + 3b^{d-2} + ... + db
= (1 + 2b^{-1} + 3b^{-2} + ... + db^{-d}) \times b^d
\leq (\sum_{i=1,...,\infty} ib^{(1-i)}) \times b^d = b^d \left(\frac{b}{(b-1)}\right)^2
**Number of Generated Nodes**  
(Breadth-First & Iterative Deepening)

\[ d = 5 \text{ and } b = 2 \]

<table>
<thead>
<tr>
<th>BF</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(1 \times 6 = 6)</td>
</tr>
<tr>
<td>2</td>
<td>(2 \times 5 = 10)</td>
</tr>
<tr>
<td>4</td>
<td>(4 \times 4 = 16)</td>
</tr>
<tr>
<td>8</td>
<td>(8 \times 3 = 24)</td>
</tr>
<tr>
<td>16</td>
<td>(16 \times 2 = 32)</td>
</tr>
<tr>
<td>32</td>
<td>(32 \times 1 = 32)</td>
</tr>
<tr>
<td>63</td>
<td>120</td>
</tr>
</tbody>
</table>

\[ \frac{120}{63} \approx 2 \]
## Number of Generated Nodes (Breadth-First & Iterative Deepening)

*d = 5 and b = 10*

<table>
<thead>
<tr>
<th>BF</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>100</td>
<td>400</td>
</tr>
<tr>
<td>1,000</td>
<td>3,000</td>
</tr>
<tr>
<td>10,000</td>
<td>20,000</td>
</tr>
<tr>
<td>100,000</td>
<td>100,000</td>
</tr>
<tr>
<td><strong>111,111</strong></td>
<td><strong>123,456</strong></td>
</tr>
</tbody>
</table>

\[
\frac{123,456}{111,111} \approx 1.111
\]
Comparison of Strategies

- Breadth-first is complete and optimal, but has high space complexity.
- Depth-first is space efficient, but is neither complete, nor optimal.
- Iterative deepening is complete and optimal, with the same space complexity as depth-first and almost the same time complexity as breadth-first.

Quiz: Would IDS + bi-directional search be a good combination?
Revisited States

No  Few  Many

search tree is finite  search tree is infinite

8-queens  assembly planning  8-puzzle and robot navigation
Avoiding Revisited States

- Requires comparing state descriptions
- Breadth-first search:
  - Store all states associated with generated nodes in VISITED
  - If the state of a new node is in VISITED, then discard the node
Avoiding Revisited States

- Requires comparing state descriptions
- Breadth-first search:
  - Store all states associated with *generated* nodes in **VISITED**
  - If the state of a new node is in **VISITED**, then discard the node

Implemented as hash-table or as explicit data structure with flags
Avoiding Revisited States

- Depth-first search:
  Solution 1:
  - Store all states associated with nodes in current path in VISITED
  - If the state of a new node is in VISITED, then discard the node

→ ??
Avoiding Revisited States

- **Depth-first search:**
  
  **Solution 1:**
  - Store all states associated with nodes in current path in VISITED
  - If the state of a new node is in VISITED, then discard the node

  → Only avoids loops

  **Solution 2:**
  - Store all generated states in VISITED
  - If the state of a new node is in VISITED, then discard the node

  → Same space complexity as breadth-first!
Uniform-Cost Search

- Each arc has some cost $c \geq \varepsilon > 0$
- The cost of the path to each node $N$ is $g(N) = \sum$ costs of arcs
- The goal is to generate a solution path of minimal cost
- The nodes $N$ in the queue FRINGE are sorted in increasing $g(N)$

- Need to modify search algorithm
Search Algorithm #2

SEARCH#2
1. INSERT(initial-node,FRINGE)
2. Repeat:
   a. If empty(FRINGE) then return failure
   b. N ← REMOVE(FRINGE)
   c. s ← STATE(N)
   d. If GOAL?(s) then return path or goal state
   e. For every state s' in SUCCESSORS(s)
      i. Create a node N' as a successor of N
      ii. INSERT(N',FRINGE)

The goal test is applied to a node when this node is expanded, not when it is generated.
Avoiding Revisited States in Uniform-Cost Search

- For any state \( S \), when the first node \( N \) such that \( \text{STATE}(N) = S \) is expanded, the path to \( N \) is the best path from the initial state to \( S \)

\[
g(N) \leq g(N') \\
g(N) \leq g(N'')
\]
Avoiding Revisited States in Uniform-Cost Search

- For any state $S$, when the first node $N$ such that $\text{STATE}(N) = S$ is expanded, the path to $N$ is the best path from the initial state to $S$

- So:
  - When a node is expanded, store its state into \text{CLOSED}
  - When a new node $N$ is generated:
    - If $\text{STATE}(N)$ is in \text{CLOSED}, discard $N$
    - If there exits a node $N'$ in the fringe such that $\text{STATE}(N') = \text{STATE}(N)$, discard the node — $N$ or $N'$ — with the highest-cost path