Blind (Uninformed) Search

Russell and Norvig:
Chap. 3, Sect. 3.3 – 3.4

Slides from Jean-Claude Latombe at Stanford University
(used with permission)
Simple Agent Algorithm

Problem-Solving-Agent

1. formulate: (abstraction!)
   1. $S_0 = \text{initial-state} \downarrow \text{sense/read state}$
   2. $\text{GOAL?} = \text{goal test} \downarrow \text{select/read goal test}$
   3. actions $\downarrow$ select/read action models
   4. transition model $\downarrow$ select/read model
   5. problem $\downarrow$ ($S_0$, GOAL?, actions, transition model)

2. solution $\downarrow$ search(problem)

3. perform(solution)
Search Tree

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

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:

- Applying each legal action on STATE($N$)
- Generating a child of $N$ for each new accessible state

node generation ≠ node expansion!
Frontier of Search Tree

The **frontier** is the set of all search nodes that haven’t been expanded yet.
Is the frontier identical to the set of leaves?
Search Strategy

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

1. If GOAL?(S₀) then return S₀
2. INSERT(N₀, FRONTIER)
3. Repeat:
   a. If EMPTY?(FRONTIER) then return failure
   b. N = POP(FRONTIER)
   c. s = STATE(N)
   d. For every state s’ in SUCCESSORS(s)
      i. Create a new node N’ as a child of N
      ii. If GOAL?(s’) then return path or goal state
      iii. INSERT(N’, FRONTIER)

SUCCESSORS(s) returns set of states reachable by single legal actions from s
Performance Measures

- **Completeness**
  A search algorithm is complete if it finds a solution whenever one 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 FRONTIER. They only exploit the positions of the nodes in the search tree.

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

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

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

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)

Arc cost = 1

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

New nodes are inserted at the end of FRONTIER

FRONTIER = (1)
Breadth-First Strategy

New nodes are inserted at the end of FRONTIER

FRONTIER = (2, 3)
Breadth-First Strategy

New nodes are inserted at the end of FRONTIER

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

New nodes are inserted at the end of FRONTIER

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

- Maximum number of successors of any state
  - \textbf{branching factor }$b$ of the search tree

- Minimal length (≠ cost) of a path between the initial and a goal state
  - \textbf{depth }$d$ of the shallowest goal node in the search tree
BF Evaluation

\[ b: \text{branching factor} \]
\[ d: \text{depth of shallowest goal node} \]

Breadth-first search is:

- Complete? Not complete?
- Optimal? Not optimal?
BF 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:
BF 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:
- \[ 1 + b + b^2 + \ldots + b^d = O(b^d) \]

→ Time and space complexity is \( O(b^d) \)
## Time and Memory Use

<table>
<thead>
<tr>
<th>$d$</th>
<th># Nodes</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>110</td>
<td>.11 msec</td>
<td>107 kilobytes</td>
</tr>
<tr>
<td>4</td>
<td>11,110</td>
<td>11 msec</td>
<td>10.6 megabytes</td>
</tr>
<tr>
<td>6</td>
<td>$\sim 10^6$</td>
<td>1.1 sec</td>
<td>1 gigabyte</td>
</tr>
<tr>
<td>8</td>
<td>$\sim 10^8$</td>
<td>2 min</td>
<td>103 gigabytes</td>
</tr>
<tr>
<td>10</td>
<td>$\sim 10^{10}$</td>
<td>3 hours</td>
<td>10 terabytes</td>
</tr>
<tr>
<td>12</td>
<td>$\sim 10^{12}$</td>
<td>13 days</td>
<td>1 petabyte</td>
</tr>
<tr>
<td>14</td>
<td>$\sim 10^{14}$</td>
<td>3.5 years</td>
<td>99 petabytes</td>
</tr>
</tbody>
</table>

Assumptions: $b = 10$; 1 million nodes/sec; 1000 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 frontier queues: FRONTIER1 and FRONTIER2

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

- New nodes are inserted at the front of FRONTPIER

FRONTPIER = (1)
Depth-First Strategy

New nodes are inserted at the front of FRONTIER

FRONTIER = (2, 3)
Depth-First Strategy

New nodes are inserted at the front of FRONTIER

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

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

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

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

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

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

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

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

New nodes are inserted at the front of FRONTIER
DF Evaluation

- $b$: branching factor
- $d$: depth of shallowest goal node
- $m$: maximal depth of a leaf node

Depth-first search is:

- Complete?
- Optimal?
**DF 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 \(O(bm)\)

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

**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
ID Evaluation

- Iterative deepening search is:
  - Complete
  - Optimal if step cost = 1

- Time complexity is:
  \[ db + (d-1)b^2 + \ldots + (1)b^d = O(b^d) \]

- Space complexity is: \( O(bd) \)
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.
Each arc has some cost $c \geq \epsilon > 0$
- The cost of the path to each node $N$ is
  $$g(N) = \sum \text{costs of arcs}$$
- The goal is to generate a solution path of minimal cost
- The nodes $N$ in the queue FRONTIER are sorted in increasing $g(N)$

Need to modify search algorithm
Search Algorithm #2

1. INSERT($N_0$,FRONTIER)
2. Repeat:
   a. If EMPTY?(FRONTIER) then return failure
   b. $N = \text{POP}(\text{FRONTIER})$
   c. $s = \text{STATE}(N)$
   d. If GOAL?(s) then return path or goal state
   e. For every state $s'$ in SUCCESSORS(s)
      i. Create a new node $N'$ as a child of $N$
      ii. INSERT($N'$,FRONTIER)

The goal test is applied to a node when this node is expanded, not when it is generated.