**Intelligent Agents**

**Agent:**
Anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

**Agent Function:**
Agent behavior is determined by the agent function that maps any given percept sequence to an action.

**Agent Program:**
The agent function for an artificial agent will be implemented by an agent program.

---

**A Simple Reflex Agent**

**Agent Function:**
Agent behavior is determined by the agent function that maps any given percept sequence to an action.

**Agent Program:**
The agent function for an artificial agent will be implemented by an agent program.

---

**Agent with Model and Internal State**

**Agent Function:**
Agent behavior is determined by the agent function that maps any given percept sequence to an action.

**Agent Program:**
The agent function for an artificial agent will be implemented by an agent program.

---

**Goal-Based Agent**

**Agent Function:**
Agent behavior is determined by the agent function that maps any given percept sequence to an action.

**Agent Program:**
The agent function for an artificial agent will be implemented by an agent program.

---

**Problem Solving as Search**

- **Search is a central topic in AI**
  - Automated reasoning is a natural search task
  - More recently: Given that almost all AI formalisms (planning, learning, etc.) are NP-complete or worse, some form of search is generally unavoidable (no “smarter” algorithm available).
Defining a Search Problem

State space - described by
- initial state - starting state
- actions - possible actions available
- successor function; operators - given a particular state $x$, returns a set of
  $<$ action, successor $>$ pairs
Goal test - determines whether a given state is a goal state.
Path cost - function that assigns a cost to a path

The 8 Puzzle

**Initial State**

```
5 4
6 1
7 3
```

**Goal State**

```
1 2 3
8
7 6 5
```

Cryptarithmetic

```
SEND
+ MORE
------
MONEY
```

Find (non-duplicate) substitution of digits for letters such that the resulting sum is arithmetically correct.

Each letter must stand for a different digit.

Solving a Search Problem: State Space Search

**Input:**
- Initial state
- Goal test
- Successor function
- Path cost function

**Output:**
- Path from initial state to goal state.
- Solution quality is measured by the path cost.

Generic Search Algorithm

```
L = make-list(initial-state)
loop
  node = remove-front(L) (node contains path of how the algorithm got there)
  if goal-test(node) == true then
    return(path to node)
  S = successors (node)
  insert (S$\setminus$L)
until L is empty
return failure
```

Search procedure defines a search tree

**Search tree**
- root node - initial state
- children of a node - successor states
- fringe of tree - L: states not yet expanded

**Search strategy** - algorithm for deciding which leaf node to expand next.
- stack: Depth-First Search (DFS).
- queue: Breadth-First Search (BFS).
Solving the 8-Puzzle

Start State

Goal State

What would the search tree look like after the start state was expanded?

Node Data Structure

Evaluate a Search Strategy

Completeness:
Is the strategy guaranteed to find a solution when there is one?

Time Complexity:
How long does it take to find a solution?

Space Complexity:
How much memory does it need?

Optimality:
Does strategy always find a lowest-cost path to solution? (this may include different cost of one solution vs. another).

Uninformed search: BFS

Consider paths of length 1, then of length 2, then of length 3, then of length 4,...

Time and Memory Requirements for BFS – \(O(b^{d+1})\)

Let \(b\) = branching factor, \(d\) = solution depth, then the maximum number of nodes generated is:

\[b + b^2 + \ldots + b^d + (b^{d+1} - b)\]

<table>
<thead>
<tr>
<th>Depth</th>
<th>Nodes</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1100</td>
<td>.11 sec</td>
<td>1 meg</td>
</tr>
<tr>
<td>4</td>
<td>111,100</td>
<td>11 sec</td>
<td>106 meg</td>
</tr>
<tr>
<td>6</td>
<td>(10^6)</td>
<td>19 min</td>
<td>10 gig</td>
</tr>
<tr>
<td>8</td>
<td>(10^8)</td>
<td>31 hrs</td>
<td>1 tera</td>
</tr>
<tr>
<td>10</td>
<td>(10^{10})</td>
<td>129 days</td>
<td>101 peta</td>
</tr>
<tr>
<td>12</td>
<td>(10^{12})</td>
<td>35 yrs</td>
<td>10 peta</td>
</tr>
<tr>
<td>14</td>
<td>(10^{14})</td>
<td>3523 yrs</td>
<td>1 exa</td>
</tr>
</tbody>
</table>
Uniform-cost Search

Use BFS, but always expand the lowest-cost node on the fringe as measured by path cost \( g(n) \).

Uninformed search: DFS

DFS vs. BFS

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>Optimal</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>YES</td>
<td>YES</td>
<td>( O(b^d) )</td>
<td>( O(b^d) )</td>
</tr>
<tr>
<td>DFS</td>
<td>Finite</td>
<td>NO</td>
<td>( O(b^d) )</td>
<td>( O(bm) )</td>
</tr>
</tbody>
</table>

\( m \) is maximum depth.

Time

- \( m = d \): DFS typically wins
- \( m > d \): BFS might win
- \( m \) is infinite: BFS probably will do better

Space

- DFS almost always beats BFS

Which search should I use?

Depends on the problem.

- If there may be infinite paths, then depth-first is probably bad.
- If goal is at a known depth, then depth-first is good.
- If there is a large (possibly infinite) branching factor, then breadth-first is probably bad.

(Could try nondeterministic search. Expand an open node at random.)

Iterative Deepening [Korf 1985]

Idea:

Use an artificial depth cutoff, \( c \).

- If search to depth \( c \) succeeds, we're done.
- If not, increase \( c \) by 1 and start over.

Each iteration searches using depth-limited DFS.
Cost of Iterative Deepening

<table>
<thead>
<tr>
<th>$b$</th>
<th>ratio of IDS to DFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1.5</td>
</tr>
<tr>
<td>10</td>
<td>1.2</td>
</tr>
<tr>
<td>25</td>
<td>1.08</td>
</tr>
<tr>
<td>100</td>
<td>1.02</td>
</tr>
</tbody>
</table>

Time: $O(b^d)$ as in DFS,

Comparing Search Strategies

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Iterative Deepening</th>
<th>Bidirectional (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>$b^{d/2}$</td>
<td>$b^*$</td>
<td>$b^d$</td>
<td>$b^{d/2}$</td>
<td></td>
</tr>
<tr>
<td>Space</td>
<td>$b^{d/2}$</td>
<td>$b^*$</td>
<td>$bm$</td>
<td>$bd$</td>
<td>$b^{d/2}$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Complete?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

***Note that many of the "yes's" above have caveats, which we discussed when covering each of the algorithms.