Intelligent Agents

agent: anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

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

The agent function for an artificial agent will be implemented by an agent program.

A “Cornell AI Student” Agent

Agent requires access to environment through sensors: visual, aural, touch, etc.

Available actions: talk, walk, do arithmetic, do boolean logic, programming skills, etc.

A Simple Reflex Agent

Agents with Internal State
Goal-Based Agents

Environment

Agent

Sensors

State

What the world evolves

How the world evolves

What the world is like now

What it will be like if I do action A

What my actions do

What action I should do now

Goals

Actuators

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

A path is any sequence of states connected by a sequence of actions.

Goal test — determines whether a given state is a goal state.

Path cost — function that assigns a cost to a path; relevant if more than one path leads to the goal, and we want the shortest path.

Assumption: cost of a path is the sum of the costs of the individual actions along the path; sum of the step costs, which must be non-negative.
The 8-Puzzle

States:
Initial state:
Goal test:
Successor function:
Path cost:

Cryptarithmetic

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

Find substitution of digits for letters such that the resulting sum is arithmetically correct.

Each letter must stand for a different digit.

Cryptarithmetic, cont.

States: an 8-tuple indicating a (partial) assignment of digits to letters.

Successor function: represents the act of assigning digits to letters.

Goal test: all letters have been assigned digits and sum is correct.

Path cost: ...all solutions are equally valid; step cost = 0.

Solving a Search Problem: State Space Search

Input:
• Initial state
• Goal test
• Successor function
• Path cost

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

State space is not stored in its entirety by the computer.
Generic Search Algorithm

L = make-queue(initial-state)
loop
    node = remove-front(L) (and save in order
to return as part of path to goal)
    if goal-test(node) = true return(path to goal)
    S = successors(node)
    insert(S,L)
until L is empty
return failure

Search procedure defines a search tree

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

stack: Depth-First Search (DFS).
queue: Breadth-First Search (BFS).

Search strategy — algorithm for deciding which leaf
node to expand next.

Node Data Structure

State: ACTION = right
DEPTH = 6
PATH-COST = 6

Solving the 8-Puzzle

What would the search tree look like after the start state
was expanded?
Evaluating 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 the strategy find the highest-quality solution when there are several different solutions?

---

**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:

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

---

<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>(10^7)</td>
<td>19 min</td>
<td>10 gig</td>
</tr>
<tr>
<td>8</td>
<td>(10^9)</td>
<td>31 hrs</td>
<td>1 tera</td>
</tr>
<tr>
<td>10</td>
<td>(10^{11})</td>
<td>129 days</td>
<td>101 tera</td>
</tr>
<tr>
<td>12</td>
<td>(10^{13})</td>
<td>35 yrs</td>
<td>10 peta</td>
</tr>
<tr>
<td>14</td>
<td>(10^{15})</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 $g(n)$.

Requirement: $g(\text{Successor}(n)) \geq g(n)$

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>$O(b^{d+1})$</td>
<td>$O(b^{d+1})$</td>
</tr>
<tr>
<td>DFS</td>
<td>finite depth</td>
<td>NO</td>
<td>$O(b^m)$</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 DFS.

Space requirements? Same as DFS. Each search is just a DFS.

Time requirements. Would seem very expensive!! BUT not much different from single BFS or DFS to depth \( d \).

Reason: Almost all work is in the final couple of layers. E.g., binary tree: 1/2 of the nodes are in the bottom layer. With \( b=10 \), 9/10th of the nodes in final layer!

So, repeated runs are on much smaller trees (i.e., exponentially smaller).

Example:
\( b=10 \), \( d=5 \), the number of nodes generated in a BFS:
\[
b + b^2 + \ldots + b^d + b^{d+1} - b =
10 + 100 + 1000 + 10,000 + 100,000 + 999,990 = 1,111,100
\]

For IDS:
\[
(d)b + (d-1)b^2 + \ldots + (1)b^d =
50 + 400 + 3,000 + 20,000 + 100,000 = 123,450
\]

Cost of repeating the work at shallow depths is not prohibitive.
Cost of Iterative Deepening

space: \( O(bd) \) as in DFS, time: \( O(b^d) \)

<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>

Bidirectional Search

- Search forward from the start state and backward from the goal state simultaneously and stop when the two searches meet the middle.
- If branching factor = \( b \) from both directions, and solution exists at depth \( d \), then need only \( O(2b^{d/2}) = O(b^{d/2}) \) steps.
- Example \( b = 10, d = 6 \) then BFS needs 1,111,111 nodes and bidirectional search needs only 2,222.
  - What does it mean to search backwards from a goal?
  - What if there is more than one goal state? (chess).

Comparing Search Strategies

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First Cost</th>
<th>Uniform-First Cost</th>
<th>Depth-First Cost</th>
<th>Iterative Deepening Cost</th>
<th>Bidirectional Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>( b^{d+1} )</td>
<td>( b^d )</td>
<td>( b^{d+1} )</td>
<td>( b^d )</td>
<td>( b^{d/2} )</td>
</tr>
<tr>
<td>Space</td>
<td>( b^{d+1} )</td>
<td>( b^d )</td>
<td>( bd )</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.***