ITCS 6150
Intelligent Systems

Lecture 3

Uninformed Searches
Outline

**Problem Solving Agents**
- Restricted form of general agent

**Problem Types**
- Fully vs. partially observable, deterministic vs. stochastic

**Problem Formulation**
- State space, initial state, successor function, goal test, path cost

**Example Problems**

**Basic Search Algorithms**
Example: Poland

On holiday in Poland; currently in Nowy Sacz.

Flight leaves tomorrow from Gdansk.

Formulate Goal:

• be in Gdansk

Formulate Problem:

• states: various cities
• actions: drive between cities

Find Solution:

• Sequence of cities, e.g., Nowy Sacz, Radom, Warsaw, Gdansk
Example: Poland
Problem Types

**Deterministic, fully observable ➔ single-state problem**
- Agent knows exactly what state it will be in; solution is a sequence

**Non-observable ➔ conformant problem**
- Agent may have no idea where it is; solution (if any) is a sequence

**Non-deterministic and/or partially observable**
- Percepts provide new information about current state
- Solution is a tree or policy
- Often interleave search, execution

**Unknown state space ➔ exploration problem ("online")**
Example: Vacuum World

*Single-state, start in #5.*

- Solution??
Example: Vacuum World

**Single-state, start in #5.**
- Solution: [Right, Suck]

**Conformant, start in \{1,2,3,4,5,6,7,8\}**
- E.g., right goes to \{2,4,6,8\}
- Solution??
Example: Vacuum World

**Single-state, start in #5.**
- Solution: [Right, Suck]

**Conformant, start in {1,2,3,4,5,6,7,8}**
- E.g., right goes to {2,4,6,8}
- Solution: [Right, Suck, Left, Suck]

**Contingency, start in #5**
- Murphy’s Law: Suck can dirty a clean carpet
- Local sensing: dirt, location only
- Solution??
Example: Vacuum World

**Single-state, start in #5.**
- Solution: [Right, Suck]

**Conformant, start in \{1,2,3,4,5,6,7,8\}**
- E.g., right goes to \{2,4,6,8\}
- Solution: [Right, Suck, Left, Suck]

**Contingency, start in #5**
- Murphy’s Law: Suck can dirty a clean carpet
- Local sensing: dirt, location only
- Solution: [Right, if dirt then Suck]
Single-state problem formation

**A problem is defined by four items:**

**Initial state**
- E.g., “at Nowy Sacz”

**Successor function** \( S(x) = \text{set of action-state pairs} \)
- E.g., \( S(\text{Arad}) = \{<\text{Nowy Sacz} \rightarrow \text{Krakow}, \text{Krakow}>, <\text{Nowy Sacz} \rightarrow \text{Tarnow}, \text{Tarnow}>, \ldots \} \)

**Goal test**
- “at Gdansk”

**Path cost (additive)**
- E.g., a sum of distances, number of actions executed, etc.
- \( C(x,a,y) \) is the step cost, assumed to be non-negative

**A solution is a sequence of actions leading from the initial state to a goal state**
State Space

Real world is absurdly complex ➔ state space must be abstracted for problem solving

- (Abstract) state = set of real states
- (Abstract) action = complex combination of real actions, e.g., “Nowy Sacz → Gdansk” represents a complex set of possible routes, detours, rest stops, etc.
- (Abstract) solution = set of real paths that are solutions in the real world

Each abstract action should be “easier” than the original problem!
Example: Vacuum World state space graph

States?  Actions?  Goal test?  Path cost?
Example: Vacuum World state space graph

**States:**
- Integer dirt and robot locations (ignore dirt amounts)

**Actions:**
- Left, Right, Suck, NoOp

**Goal test:**
- No dirt

**Path cost:**
- 1 per action (0 for NoOp)
Other Examples

Eight puzzle [http://mypuzzle.org/sliding]

Patient Treatment [procedure path]

States?  Actions?  Goal test?  Path cost?
Tree Search Algorithms

**Basic idea:**

- Offline, simulated exploration of state space by generating successors of already explored states (AKA expanding states)

```plaintext
function TREE-SEARCH(problem, strategy) returns a solution, or failure
initialize the search tree using the initial state of problem
loop do
  if there are no more candidates for expansion then return failure
  choose a leaf node for expansion according to strategy
  if the node contains a goal state then return the corresponding solution
  else expand the node and add the resulting nodes to the search tree
end
```
Implementation: states vs. nodes

State
- (Representation of) a physical configuration

Node
- Data structure constituting part of a search tree
  - Includes parent, children, depth, path cost $g(x)$

States do not have parents, children, depth, or path cost!
Search strategies

A strategy is defined by picking the order of node expansion

Strategies are evaluated along the following dimensions:

- Completeness – does it always find a solution if one exists?
- Time complexity – number of nodes generated/expanded
- Space complexity – maximum nodes in memory
- Optimality – does it always find a least cost solution?

Time and space complexity are measured in terms of:

- $b$ – maximum branching factor of the search tree
- $d$ – depth of the least-cost solution
- $m$ – maximum depth of the state space (may be infinite)
Uninformed Search Strategies

Uninformed strategies use only the information available in the problem definition

- Breadth-first search
- Uniform-cost search (related to breadth-first search; determines a path to the goal state that has the lowest weight)
- Depth-first search
- Depth-limited search
- Iterative deepening search (depth-first search is run repeatedly with increasing depth limits until the goal is found)
Breadth-first search

Expand shallowest unexpanded node

Implementation:

- *Fringe* is a FIFO queue, i.e., new successors go at end
- Execute first few expansions of *Eight Puzzle* using Breadth-first search
Properties of breadth-first search

**Complete??** Yes (if b is finite)

**Time??** \(1 + b + b^2 + \ldots + b^d + b(b^d-1) = O(b^{d+1}), \text{ i.e.,}\)
\[\text{exp in d}\]

**Space??** \(O(b^{d+1})\) (keeps every node in memory)

**Optimal??** If cost = 1 per step, not optimal in general

**Space is the big problem; can easily generate nodes at 10 MB/s, so 24hrs = 860GB!**
Uniform-cost search

**Expand least-cost unexpanded node**

**Implementation:**
- *Fringe* = queue ordered by path cost

Equivalent to breadth-first if...

Complete?? If step cost $\geq \epsilon$

Time?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{C/\epsilon})$, where $C$ is the cost of the optimal solution

Space?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{C/\epsilon})$

Optimal?? Yes–nodes expanded in increasing order of $g(n)$

Execute first few expansions of *Eight Puzzle* using Uniform-first search
Depth-first search

Expand deepest unexpanded node

Implementation:

- Stack
- Execute first few expansions of Eight Puzzle game using Depth-first search
Depth-first search

**Complete??**
- No: fails in infinite-depth spaces, spaces with loops.
- Can be modified to avoid repeated states along path → complete in finite spaces

**Time??**
- \(O(b^m)\): terrible if \(m\) is much larger than \(d\), but if solutions are dense, may be much faster than breadth-first

**Space??**
- \(O(bm)\), i.e., linear space!

**Optimal??**
- No
Iterative deepening search

function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution

inputs: problem, a problem

for depth ← 0 to ∞ do

    result ← DEPTH-LIMITED-SEARCH(problem, depth)

    if result ≠ cutoff then return result

end
Summary

All tree searching techniques are more alike than different

Breadth-first has space issues, and possibly optimality issues

Uniform-cost has space issues

Depth-first has time and optimality issues, and possibly completeness issues

Depth-limited search has optimality and completeness issues

Iterative deepening is the best uninformed search we have explored

Next class we study informed searches