CS 232: Artificial Intelligence

Search
September 3rd, 2015

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu]

Today

- Agents that Plan Ahead
- Search Problems
- Uninformed Search Methods
  - Depth-First Search
  - Breadth-First Search
  - Uniform-Cost Search

Agents that Plan

- Reflex agents:
  - Choose action based on current percept (and maybe memory)
  - May have memory or a model of the world's current state
  - Do not consider the future consequences of their actions
  - Consider how the world IS

- Can a reflex agent be rational?

Reflex Agents

[Demo: reflex optimal (L2D1)]
[Demo: reflex optimal (L2D2)]
Planning Agents:
- Ask "what if"
- Decisions based on (hypothesized) consequences of actions
- Must have a model of how the world evolves in response to actions
- Must formulate a goal (test)
- Consider how the world WOULD BE if a certain action is performed

Optimal vs. complete planning
- Optimal: find the best solution
- Complete planning: finds a solution

Planning vs. replanning
- Generates one single plan vs. replans at every step of the way

Video of Demo Reflex Optimal

Video of Demo Reflex Odd

Search Problems
Search Problems

- A search problem consists of:
  - A state space
  - A successor function (with actions, costs)
  - A start state and a goal test
  - A solution is a sequence of actions (a plan) which transforms the start state to a goal state

Search Problems Are Models

Important: Find the right level of abstraction, so that your model is a good model of the world.

Example: Traveling in Romania

- State space:
  - Cities
- Successor function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?

What’s in a State Space?

The world state includes every last detail of the environment

A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: (x,y)=END

- Problem: Eat-All-Dots
  - States: [(x,y), dot booleans]
  - Actions: NSEW
  - Successor: update location and possibly a dot boolean
  - Goal test: dots all false
State Space Sizes?

- **World state:**
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

- **How many**
  - World states? 120x(2^30)x(12!)x4
  - States for pathing? 120
  - States for eat-all-dots? 120x(2^31)

---

Quiz: Safe Passage

- Problem: eat all dots while keeping the ghosts perma-scared
- What does the state space have to specify?
  - Solution:

---

State Space Graphs and Search Trees

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arrows represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)

- In a state space graph, each state occurs only once!

- We can rarely build this full graph in memory (it’s too big), but it’s a useful idea

---

State Space Graphs
State Space Graphs

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent successors (action results)
  - The goal test is a set of goal nodes (maybe only one)
- In a search graph, each state occurs only once!
- We can rarely build this full graph in memory (it’s too big), but it’s a useful idea

Search Trees

- A search tree:
  - A “what if” tree of plans and their outcomes
  - The start state is the root node
  - Children correspond to successors
  - Nodes show states, but correspond to PLANS that achieve those states
  - For most problems, we can never actually build the whole tree

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

How big is its search tree (from S)?

Important: Lots of repeated structure in the search tree!
Searching with a Search Tree

- **Search:**
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

General Tree Search

**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 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

- **Important ideas:**
  - Fringe
  - Expansion
  - Exploration strategy

- **Main question:** which fringe nodes to explore?
**Depth-First Search**

**Strategy:** expand a deepest node first

**Implementation:** Fringe is a LIFO stack

---

**Search Algorithm Properties**

- **Complete:** Guaranteed to find a solution if one exists?
- **Optimal:** Guaranteed to find the least cost path?
- **Space complexity?**
- **Cartoon of search tree:**
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths
- **Number of nodes in entire tree?**
  - \( 1 + b + b^2 + \ldots + b^m = O(b^m) \)

---

**Depth-First Search (DFS) Properties**

- **What nodes DFS expand?**
  - Some left prefix of the tree.
  - Could process the whole tree!
  - If m is finite, takes time \( O(b^m) \)
- **How much space does the fringe take?**
  - Only has siblings on path to root, so \( O(bm) \)
- **Is it complete?**
  - \( m \) could be infinite, so only if we prevent cycles (more later)
- **Is it optimal?**
  - No, it finds the "leftmost" solution, regardless of depth or cost
**Breadth-First Search (BFS) Properties**

- **What nodes does BFS expand?**
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be \( s \)
  - Search takes time \( O(b^s) \)

- **How much space does the fringe take?**
  - Has roughly the last tier, so \( O(b^s) \)

- **Is it complete?**
  - \( s \) must be finite if a solution exists, so yes!

- **Is it optimal?**
  - Only if costs are all 1 (more on costs later)

---

**Quiz: DFS vs BFS**

- When will BFS outperform DFS?
- When will DFS outperform BFS?

[Demo: dfs/bfs maze water (L2D6)]
Cost-Sensitive Search

BFS finds the shortest path in terms of number of actions. It does not find the least-cost path. We will now cover a similar algorithm which does find the least-cost path.

Uniform Cost Search

Strategy: expand a cheapest node first:
Fringe is a priority queue (priority: cumulative cost)

Uniform Cost Search (UCS) Properties

- **What nodes does UCS expand?**
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs $C^*$ and arcs cost at least $\varepsilon$, then the "effective depth" is roughly $C^*/\varepsilon$
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)

- **How much space does the fringe take?**
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$

- **Is it complete?**
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!

- **Is it optimal?**
  - Yes! (Proof next lecture via A*)
### Uniform Cost Issues

- **Remember:** UCS explores increasing cost contours

- **The good:** UCS is complete and optimal!

- **The bad:**
  - Explores options in every “direction”
  - No information about goal location

- **We’ll fix that soon!**

![Uniform Cost Issues Diagram](image)

### The One Queue

- **All these search algorithms are the same except for fringe strategies**
  - Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
  - Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
  - Can even code one implementation that takes a variable queuing object

### Search and Models

- **Search operates over models of the world**
  - The agent doesn’t actually try all the plans out in the real world!
  - Planning is all “in simulation”
  - Your search is only as good as your models...

### Search Gone Wrong?

- ![Search Gone Wrong Image](image)