CS 343: Artificial Intelligence

Search

Profs. Peter Stone and Yuke Zhu

The University of Texas at Austin

[These slides are based on slides 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.]
Good morning colleagues!

- Past due:
  - Python tutorial
  - 3 reading responses: AI100 report; Chapters 1,2; Chapter 3
- HW1: Search
  - Due Monday 2/8 at 11:59 pm
- P1: Search
  - Due Wednesday 2/10 at 11:59 pm
  - Pair work allowed
- Readings: Constraint Satisfaction and Local Search
  - **NOT** just Chapter 4
  - Due Monday 2/1 at 9:30 am
Programming Assignment 1

- P1: Search
  - Due Wednesday 2/10 at 11:59 pm
  - Pair work allowed
Textbook and other Resources

- Take your cue from how detailed things are
  - Important: heuristics for A* search; the concept of memory-bounded search
  - Just high-level ideas: IDA*, learning to search better
- Pseudocode can be useful
  - If something’s not clear, ask!
- Your reading responses were great!
- Monitor the class resources page
  - Links to more complete slide decks and lecture videos
Agents that Plan
Reflex Agents

- 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?
Video of Demo Reflex — Success
Planning Agents

- 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

- Optimal vs. complete planning
- Planning vs. replanning
- Planning vs. learning
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 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...

- This week:
  - Discrete
  - Deterministic
  - Fully observable
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: is (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?
    \[120 \times (2^{30}) \times (12^2) \times 4\ (> 74 \text{ trillion!})\]
  - States for pathing?
    120
  - States for eat-all-dots?
    \[120 \times (2^{30})\ (> 128 \text{ billion})\]
Quiz: Safe Passage

- Problem: eat all dots while keeping the ghosts perma-scared
- What does the state space have to specify?
  - (agent position, dot booleans, power pellet booleans, remaining scared time)
State Space Graphs and Search Trees
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 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
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

This is now / start
Possible futures
We construct both on demand – and we construct as little as possible.

Each NODE in the search tree is an entire PATH in the state space graph.
Consider this 4-state graph: How big is its search tree (from S)?

Important: Lots of repeated structure in the search tree!

So why would we ever use a search tree?

1) Cannot store “closed list” (previously visited nodes)
2) Graph happens to be a tree, so no reason to store closed list
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?
Tree Search: Extra Work!

- Failure to detect repeated states can cause exponentially more work.
Graph Search

- **Idea:** never expand a state twice

- **How to implement:**
  - Tree search + set of expanded states ("closed set")
  - Expand the search tree node-by-node, but...
  - Before expanding a node, check to make sure its state has never been expanded before
  - If not new, skip it, if new add to closed set

- **Important:** store the closed set as a set, not a list

- **Can graph search wreck completeness?** Why/why not?

- **How about optimality?**
The Main Search Algorithms

- **Uninformed Search:**
  - Breadth First Search (BFS)
  - Depth First Search (DFS)
  - Uniform Cost Search (UCS)  ~ [Dijkstra’s Algorithm]

- **Informed Search:**
  - Greedy Search  ~ [Best First Search]
  - A* Search

- Test your understanding!