Uninformed Search Methods:
- Depth-First Search
- Breadth-First Search
- Uniform-Cost Search

Informed Search:
- A* or “A star”.

Sirius? Brightest star in sky.
No! search! Main idea: Heuristics.

Admissible.

Graph Search: Consistent.

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.

State Space Graphs vs. Search Trees

Each NODE in search tree is an entire PATH in state space graph.
We construct both on demand – and we construct as little as possible.

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

Tree Search: example.
**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?
- **Time complexity?**
- **Space complexity?**

Sketch of search tree:
- $b$ is the branching factor
- $m$ is the maximum depth
- Solutions at various depths

Number of nodes in entire tree?

$1 + b + b^2 + \cdots = O(b^m)$

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

- Which nodes expanded?
- 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

Strategy: expand a shallowest node first
Implementation: Fringe is a FIFO queue

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

DFS vs BFS

When will BFS outperform DFS? When will DFS outperform BFS?
[Demo]
- dfs
  /  
bfs
maze water (L2D6)
Space versus Time or Quality of Solution.

Video of Demo Maze Water DFS/BFS (part 1)

Video of Demo Maze Water DFS/BFS (part 2)
Iterative Deepening

Idea: get DFS's space advantage with BFS's time / shallow-solution advantages

- Run a DFS with depth limit 1.
  - If no solution...
   - Run a DFS with depth limit 2.
     - If no solution ...
       - Run a DFS with depth limit 3. ....

Isn't that wastefully redundant?
Generally most work happens in the lowest level searched, so not so bad!

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.

How?

Uniform Cost Search (Dijkstra's algorithm.)

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

Uniform Cost Search

What nodes does UCS expand?
All nodes cheaper than solution!
Solution cost $C^*$ and arc cost $\geq \epsilon$: depth $O(C^*/\epsilon)$.
Time: $O(b^{C^*/\epsilon})$.
How much space does the fringe take?
Last Tier Space: $O(b^{C^*/\epsilon})$.
Is it complete?
Finite solution cost/positive arc weights? Then yes.
Is it optimal?
Yes.

Uniform Cost Search (UCS) Properties

- Depth: $O(C^*/\epsilon)$.
- Time: $O(b^{C^*/\epsilon})$.
- Last Tier Space: $O(b^{C^*/\epsilon})$.
- Is it complete?
  - Finite solution cost/positive arc weights?
  - Then yes.
- Is it optimal?
  - Yes.
Uniform Cost Issues

Remember: UCS explores increasing cost contours.
The good:
UCS is complete and optimal!
The bad:
Goes in every “direction”.
The ugly?
Huh?
No information about goal location.
We’ll fix that soon!

Video of Demo Empty UCS

Video of Demo Maze with Deep/Shallow Water — DFS, BFS, or UCS? (part 1)

Video of Demo Maze with Deep/Shallow Water — DFS, BFS, or UCS? (part 2)

Video of Demo Maze with Deep/Shallow Water — DFS, BFS, or UCS? (part 3)

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 (state spaces) 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...
Recap: Search

Search problem:
States (configurations of the world)
Actions and costs
Successor function (world dynamics)
Start state and goal test

Search tree:
Nodes: represent plans for reaching states
Plans have costs (sum of action costs)

Search algorithm:
Systematically builds a search tree
Orders the fringe (unexplored nodes)
Complete: finds a plan.
Optimal: finds least-cost plans

Example: Pancake Problem
State space graph with costs as weights.

Cost: Number of pancakes flipped.

Example: Pancake Problem

General Tree Search

Action: flip top two
Cost: 2
Action: flip all four
Cost: 4
Path to reach goal:
Flip four, flip three
Total cost: 7

Example: Pancake Problem

Example: Pancake Problem

The One Queue

Our search algorithms are the same except for fringe strategies
▶ All fringes are priority queues: states with priorities.
▶ DFS, BFS a bit faster using simple stack/queues.
▶ Can code one implementation with variable queuing object.
Uninformed Search

Uniform Cost Search

Strategy: expand lowest path cost.
The good: Complete and optimal!
The bad: Explores options in every "direction" No information about goal location

Informed Search

Search Heuristics

A heuristic is:
▶ A function that estimates how close a state is to a goal
▶ Designed for a particular search problem
▶ Examples: Euclidean distance for pathing. Manhattan distance.

Example: Heuristic Function

Heuristic: the number of the largest pancake that is still out of place

Example: Heuristic Function

Heuristics

11.2
5
10

A heuristic is:
▶ A function that estimates how close a state is to a goal
▶ Designed for a particular search problem
▶ Examples: Euclidean distance for pathing. Manhattan distance.

Example: Heuristic Function

Heuristic: the number of the largest pancake that is still out of place