L08 – Path search problems

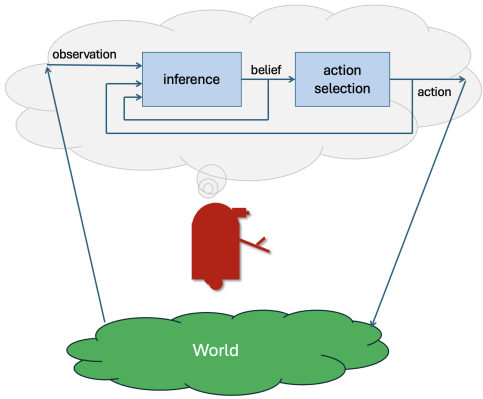
AIMA4e: Required: 3.1–3.4; 3.5.1–4; 3.6.1–2; 5.4

What you should know after this lecture

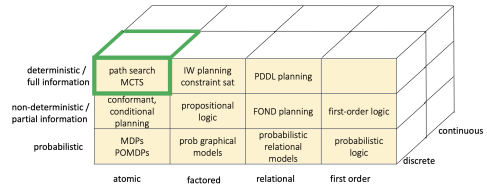
- State-space search
  - Minimizing additive path cost
  - Importance of avoiding redundant paths
- Uninformed search methods: Breadth first and Uniform cost
- Informed search methods: GBFS and A*
- Heuristics and where to find them
- Reward-formulation problems; relation to min-cost-path

Decision making!

- Given a current belief about the world
- And some objective
- What action should the agent take next?
- Apply the principle of rationality: select actions that will maximize your expected future utility



First problem setting: fully observable, deterministic



Atomic, discrete

- **Path-cost problem**
  - State set: $\mathcal{S}$
  - Initial state: s_0
  - Action set: $\mathcal{A}$
  - Transition model: $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$
  - Goal set: $G \subset \mathcal{S}$
  - Cost function $C : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$
- We need to find next action to take
- Find plan $a_1, \ldots, a_m$ from s_0 to some state in G such that $T(s_0, a_1) = s_1, \ldots, T(s_{m-1}, a_m) = s_m$ and $s_m \in G$
- Usually we try to minimize

$$\sum_i C(s_i, a_i, s_{i+1})$$

If path costs are not additive, then many algorithmic tricks don't apply and problem is much harder.

Reward-maximization formulation

Some problems are easier to formulate in terms of maximizing an amount of reward that gets accumulated over a trajectory of a fixed number of steps (horizon) H.

- Problem: $(\mathcal{S}, \mathcal{A}, T, R, H, s_0)$
- Reward instead of cost: $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- We want to find a length H path that maximizes

$$\sum_{t=0}^{H-1} R(s_t, a_t, s_{t+1})$$

- We can relax this fixed-horizon assumption later in the course, with a probabilistic model of termination.

Some formulation problems

The state should contain all and only the aspects of the world that can change and that are relevant for goal/reward.

- Trying to get to downtown Boston via walking, bike, T
- Solving our machine-shop problem
- Mars rover sample return mission
- Cleaning operations for all of United airlines

Measuring problem-solving performance

- Completeness: If there is a solution to your problem, is the algorithm guaranteed to find it?
- Cost optimality: If there is a solution, is the algorithm guaranteed to find the solution with the lowest cost?
- Computational complexity: As the size of the problem grows, how do the computation and time and space requirements grow? The answer to this depends on how we encode the input!
  - In CS algorithms tradition, problems are described as *graph search* problems, and complexity is characterized in terms of the number of vertices (states) and edges in the graph; usually nearly linear in the size of the input.
  - In our applications, we will often have a huge or even infinite S but it is not input to the algorithm. Instead, we provide s_0 and T, and incrementally expose the graph as we search. Characterize complexity in terms of branching factor $|A|$ and depth (also called "horizon" or "plan-length.") Usually exponential in the horizon.

Best-first search framework

- Critical to make a distinction between state (element of $\mathcal{S}$) and node of the search tree, which represents a path from s_0 to some state s. (Every search node has an associated state. It is possible to have multiple nodes with the same state (representing different paths to reach that state.)
- This framework takes a priority function f. Different values of f will yield different search algorithms.

Best-first search framework

```
BEST-FIRST-SEARCH(S, A, s_0, T, G, C, f)
 1  n = NODE(s_0)
 2  frontier = PRIORITYQUEUE(f)
 3  frontier.ADD(n)
 4  reached = {s_0 : n}
 5  while not frontier.EMPTY():
 6        n = frontier.POP()                          // Get node with lowest f value
 7        s = n.s
 8        if s ∈ G: return n
 9        for a ∈ A:                                  // Expand s
10              s' = T(s, a)
11              path_cost = n.path_cost + C(s, a, s')
12              if not s' ∈ reached or path_cost < reached[s'].path_cost:
13                    n' = NODE(s', n, a, path_cost)
14                    reached[s'] = n'                // visit s'
15                    frontier.ADD(n')
```

Redundant Paths

- Stupidest possible algorithm (SPA): enumerate all legal paths and pick the first one that reaches a goal state.
- There can be exponentially more paths than states!
- The *reached* data structure (sometimes called a visited list) and test in line 12 ensures that we never consider a path to a state that is higher in cost than the best one we've already found.
- In most of the searches we'll consider, in fact, we can prove that the first time we pop some path to a state off the frontier, we will have done so via a least-cost path, and so we never expand (consider the successor of) a state more than once.

Breadth-First Search

- Best-first with $f(n)$ **= number of steps in path** n
- First path found to s has fewest steps
- Can actually move the goal test earlier (from s in line 8 to s' just after line 10)
- Complete and optimal (in number of steps)
- Worst case time (and space) complexity $O(|\mathcal{A}|^d)$ where d is the length of the shortest path to the goal. This happens when there are no redundant paths.
- Note independence of $|\mathcal{S}|$
- But, complexity is also bounded by $O(|\mathcal{S}||\mathcal{A}|)$ which in some problems is smaller than $|\mathcal{A}|^d$.

Uniform Cost Search

- Best-first search with $f(n) = n.\mathit{path\_cost}$
- Assume costs are all positive[1]
- Like breadth-first, but pushes out frontier in equal-path-cost contours
- First path to s expanded has least cost.
- First path to s visited **does not necessarily have** least cost. See text for example of this, and why we cannot move the goal test earlier.
- Worst case time (and space) complexity

$$O(|\mathcal{A}|^{1+\lfloor C^*/\epsilon \rfloor})$$

  where C^* is the cost of the least-cost path and ϵ is the cost of the least-cost action.
- Sometimes called Dijkstra's Algorithm (although Dijkstra's is often used to compute shortest paths to all vertices in a given finite graph.)
- As with BFS, complexity is also bounded by $O(|\mathcal{S}||\mathcal{A}| \log |\mathcal{S}|)$.

[1] Or, at least, they are non-negative and that there are no zero-cost infinite paths

Informed state-space search methods

- Without any hints at all about how to make progress toward a goal state, we can't do better than uniform-cost search.
- A heuristic function $h : \mathcal{S} \to \mathbb{R}$ provides an estimate of the cost of the least-cost path from a state s to a goal state. (In AIMA, defined on nodes n, but really just applies to $n.s$).
- Standard example: Euclidean distance from s to a target destination in a route-finding problem.

Greedy best-first search (GBFS)

- Best-First-Search where

$$f(n) = h(n.s)$$

- Always take the path out of *frontier* that we estimate has gotten closest to the goal.
- Not guaranteed to find the least-cost path!
- Often finds a satisficing (goal-reaching) path much more quickly than uniform-cost search.

A*

- Best-First-Search where

$$f(n) = n.\mathit{path\_cost} + h(n.s)$$

- Always take the path out of *frontier* that we estimate has the cheapest sum of the length of the path so far and our estimate of how for from here to the goal.
- Guaranteed to find a least-cost path if h is admissible.
- Heuristic h is admissible iff

$$h(s) \leqslant h^*(s) \quad \text{for all } s \in \mathcal{S},$$

  where $h^*(s)$ is the actual least path cost from s to a goal state.
- If h is consistent we can slightly simplify our code
- Heuristic h is consistent iff

$$h(s) \leqslant c(s, a, s') + h(s')$$

More about A*

- Search contours are "stretched" in the direction of goal states.
- Let C^* be cost of optimal solution path:
  - A* expands all nodes reachable from s_0 on a path where every node on the path has $f(n) < C^*$
  - A* expands no nodes with $f(n) > C^*$
- If $h(s) = h^*(s)$ then A* will not expand any nodes that are not on an optimal path.
- If $h(s)$ is close to $h^*(s)$ then there will generally not be many nodes for which $f(n) \leqslant C^*$.
- If $h(s) = 0$ then h is admissible; in this case, A* degenerates into UCS.

Heuristic Functions

- A heuristic function, ideally, is:
  - Admissible and consistent
  - Close to h^*
  - Efficient to compute
- A good source of heuristics is problem relaxation: make your problem "easier" in two ways:
  - Solutions have lower cost in relaxed problem
  - Solutions are faster to find in relaxed problem
- Examples:
  - Relax problem of finding a path on a road-map to finding one that can go off-road.
  - Relax problem of finding a driving route that lets you keep the car fueled to one in which you ignore fuel.
- Another strategy: learn h (perhaps in the form of a neural network) using supervised or reinforcement-learning based on previous experience solving related problems.

Reward-maximization formulation

Some problems are easier to formulate in terms of maximizing an amount of reward that gets accumulated over a trajectory of a fixed number of steps (horizon) H.

- Problem: $(\mathcal{S}, \mathcal{A}, T, R, H, s_0)$
- Reward instead of cost: $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- We want to find a length H path that maximizes

$$\sum_{t=0}^{H-1} R(s_t, a_t, s_{t+1})$$

- We can relax this fixed-horizon assumption later in the course, with a probabilistic model of termination.

Reduction from reward maximization to min-cost-path problem

Given reward maximization problem $(\mathcal{S}, \mathcal{A}, T, R, H, s_0)$ we can generate min-cost-path problem $(\mathcal{S}', \mathcal{A}', T', G, C, s'_0)$ so that solution to the min-cost-path problem is a solution to the original reward-maximization problem.

- $\mathcal{S}' = \mathcal{S} \times \{0, \ldots, H\}$
- $\mathcal{A}' = \mathcal{A}$
- $s'_0 = (s_0, H)$ second component is "steps to go"
- $T'((s,t),a) = (T(s,a), t-1)$
- $G = \{(s,t) \mid t = 0\}$
- $C(s,a) = R_{max} - R(s,a)$ where $R_{max} = \max_{s,a} R(s,a)$

Note that costs are always non-negative.
We can solve using uniform-cost search!
Very hard to come up with a heuristic, since in principle, it might be possible for all the rest of your actions to pay off with R_{max} which would have a C of 0, meaning to be admissible, we need $h = 0$.

Reduction from min-cost-path to reward maximization

Given a min-cost-path problem $(\mathcal{S}, \mathcal{A}, T, G, C, s_o)$ we can generate a reward maximization problem $(\mathcal{S}', \mathcal{A}', T', R, H, s_0')$ so that solution to the min-cost-path problem is a solution to the original reward-maximization problem.

- $\mathcal{S}' = \mathcal{S} \times \{over\}$
- $\mathcal{A}' = \mathcal{A}$
- $s_0' = s_0$
- $$T'(s, a) = \begin{cases} T(s, a) & \text{if } s \notin G \text{ and } s \neq over \\ over & \text{otherwise} \end{cases}$$
- $R(s, a, s') = -C(s, a, s')$ if $s' \neq over$ else 0

Setting H is tricky:

- Could keep trying to re-solve with increasing H.
- You can do MCTS (or some other solution methods) on indefinite horizon problems, where instead of having a fixed horizon H, there are states marked as terminal and the "rollout" ends when one is reached (but you *still* need a max horizon in practice).