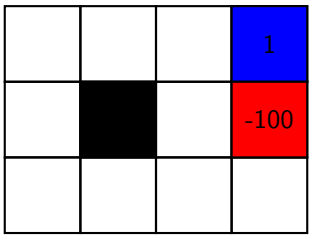
COMP3702/COMP7702 Artificial Intelligence
Semester 2, 2020
Tutorial 9

Note: *Tutorial 8 said we would work through solving the fruit stall MDP problem using value iteration, policy iteration and MCTS. Instead, we will work through the gridworld example below, the same that was used in the Week 9 lecture.*

Example domain

Consider the gridworld below:



States in this environment are the positions on the tiles. The world is bounded by a boundary wall, and there is one obstacle, at [1,1] (using python or zero indexing starting from the bottom left corner). In addition, there are two terminal states, indicated by the coloured squares.

Terminal states: In practice, we would say that the two coloured tiles are *terminal states* of the MDP. For mathematical convenience, we often prefer to deal with an infinite horizon MDP (see more below). To convert this model to an infinite horizon MDP, we will add an extra pseudo-state to the list of MDP states called *exited*, which is absorbing (the agent cannot leave it irrespective of its action, i.e. $T(exited, a, exited) = 1$ for all a).

Actions and Transitions: In this world, an agent can generally choose to move in four directions — *up*, *down*, *left* and *right* (which we might sometimes refer to as ^, v,< and >, respectively, as in the LaserTank environment). However, the agent moves successfully with only $p = 0.8$, and moves perpendicular to its chosen direction with $p = 0.1$ in each perpendicular direction. If it hits a wall or obstacle, the agent stays where it is. In addition, once the agent arrives on a coloured square with a value, it has only one special action available to it; that is, to *exit* the environment.

Rewards: The values stated on the coloured squares are the reward for *exiting* the square and the environment, so the reward is not repeatedly earned; that is, $R([3, 2], exit) = 1$, and $R([3, 1], exit) = -100$. All other states have 0 reward.

Discount factor: $\gamma = 0.9$.

Exercises

Comment: It looks like a lot of questions below, but most of these are small steps on the path to implementing VI, PI and MCTS.

Exercise 9.1. Define and code the transition function for this problem.

a) What is the one-step probability of arriving in state [1,0] from each state s after taking action a, i.e what is $P([1, 0]|a, s)\ \forall\ (a, s)$?

b) What is the one-step probability of arriving in each state s' when starting from [0,0] for each a, i.e what is $P(s'|a, [0, 0])\ \forall\ a, s'$?

c) Write a function that can compute the probabilities of arriving in any state s' from an initial state s after taking action a, i.e. $P(s'|a, s)$. Rather than hard-coding the probabilities, check if each action moves to a neighbour state or if it results in any collisions with the boundary or the obstacle. Then use this information to compute the transition probabilities. *Hint:* You may find some of the tricks for 8-puzzle from Tutorial 2 useful, e.g. using linear indexing and logical tests for collisions. You may also wish to parameterise this function so that p can be varied too (useful for the assignment).

d) Write a new function to processes the gridworld and store all of the transition probabilities, using the transition-probability computing function you have developed above. The aim is to have functions that can be used on an arbitrary gridworld (i.e. do not hard-code your function just for this problem instance!).

Exercise 9.2. Implement VI for this problem, using: $V[s] \leftarrow \max_a \sum_{s'} P(s' \mid s, a)\,(R(s, a, s') + \gamma V[s'])$.

Note that $R(s, a, s') = R(s)$ is the reward for landing on a square, which is non-zero for only the red and blue squares at [3,1] and [3,2], respectively.

a) What is the value function estimate after 4 iterations? What is the policy according to the value function estimate after 4 iterations?

b) What is the value function estimate after 10 iterations? What is the policy according to the value function estimate after 10 iterations?

c) What is the value function estimate after 1000 iterations? What is the policy according to the value function estimate after 1000 iterations?

d) What is the largest difference in the value function estimates for any state at iteration 999 and 1000? Based on this, can you say that the algorithm has converged?

e) How long does 1000 iterations of VI take?

Exercise 9.3.

a) Let $P^\pi \in R^{|S|\times|S|}$ be a matrix containing probabilities for each transition under the policy π, where:

$$P^\pi_{ij} = P(s_{t+1} = j \mid s_t = i, a_t = \pi(s_t))$$

What is the size of P^π in this gridworld, when the special pseudo-state *exited* is included?

b) Set the policy to move *right* everywhere, $\pi(s) => \ \forall\, s \in S$. Calculate the row of $P^>$ corresponding to an initial state at the bottom left corner, [0,0] of the gridworld.

c) Write a function that calculate the row of $P^>$ corresponding to an initial state at the bottom left corner, [0,0] of the gridworld for any action or deterministic $\pi([0, 0])$.

d) Turn this into a function that computes P^π for any deterministic π. Note that $P([3, 1], a, existed) = P([3, 2], a, existed) = P(exited, a, existed) = 1$ for all a, so the rows of P^π for these states s are all zeros except for a 1 corresponding to the transition to $s' = existed$.

e) Compute

$$V^> = (I - \gamma P^>)^{-1} r.$$

To do this, define r as the reward for landing on a square, which holds because $R(s, a, s') = R(s)$ for the red and blue squares at [3,1] and [3,2], respectively. (*Hint*: Rather than explicitly computing the matrix inverse, you may want to use `numpy.linalg.solve()` for large $|S|$. This implements the LU decomposition to solve for V^π using forward and backward substitution, so can result in a big speed-up.)

f) Using the policy evaluation above, implement PI for this problem, following:

- Set $\pi(s) => \ \forall\, s \in S$, and let $iter = 0$
- Repeat:
  1. Solve for $V^{\pi_i}(s)$ (or $Q^{\pi_i}(s, a)$):

$$V^{\pi_i}(s) = \sum_{s' \in \mathcal{S}} P(s' \mid \pi_i(s), s)\,[R(s, \pi_i(s), s') + \gamma V^{\pi_i}(s')] \quad \forall s \in S$$

  2. Update policy:

$$\pi_{i+1}(s) \leftarrow \arg\max_a \sum_{s' \in \mathcal{S}} P(s' \mid a, s)\,[R(s, a, s') + \gamma V^{\pi_i}(s')]$$

  3. $iter = iter + 1$
- until $\pi_i(s) = \pi_{i-1}(s)$

g) How many iterations does PI take to converge? How long does each iteration take?

Monte Carlo tree search

This material is adapted from: C. B. Browne et al., "A Survey of Monte Carlo Tree Search Methods," in IEEE Transactions on Computational Intelligence and AI in Games, vol. 4, no. 1, pp. 1-43, March 2012, http://www.diego-perez.net/papers/MCTSSurvey.pdf. See the paper for more detail, and note that we will cover bandit sampling, UCB and UCT in detail in Week 10's lecture.

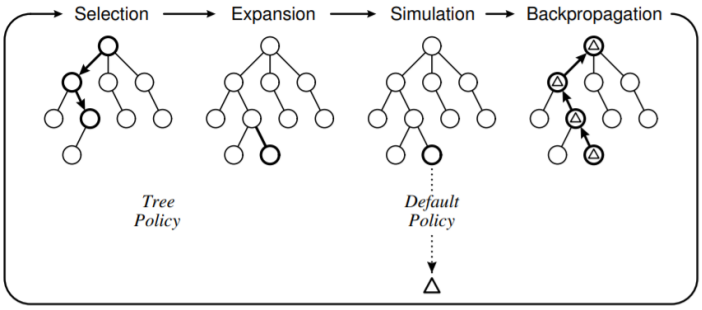
MCTS is a family of algorithms for fast planning, typically in online settings. The basic MCTS algorithm involves iteratively building a search tree until some predefined computational budget – typically a time, memory or iteration constraint, or sometimes all three – is reached, at which point the search is halted and the best-performing root action returned. MCTS is particularly useful because it is *anytime* – the algorithm can be interrupted and it will still return a solution, and the quality of the solution tends to increase with more iterations.

As the name suggests, MCTS involves search on a tree. Like in the search problems we addressed earlier in the course, each node in the search tree represents a state of the problem, and directed links to child nodes represent actions leading to subsequent states. However, unlike blind or heuristic search, MCTS interleaves random sampling of new states and backpropagation of values in order to direct the search process.

Four steps are applied per search iteration:

1. **Selection**: Starting at the root node, a child selection policy (i.e. sampling) is recursively applied to descend through the tree until the most urgent expandable node is reached. A node is expandable if it represents a nonterminal state and has unvisited (i.e. unexpanded) children. the measure of urgency is typically given by a function of the estimated mean and variance of the value of the state, which is incorporated into a bandit sampling policy.
2. **Expansion**: One (or more) child nodes are added to expand the tree, according to the available actions.
3. **Simulation**: A simulation is run from the new node(s) according to the default policy (e.g. choose a node uniformly at random) to produce an outcome.
4. **Backpropagation**: The simulation result is "backed up" (i.e. backpropagated) through the selected nodes to update their statistics. Backpropagation involves passing estimated mean and variance values back up the tree while selecting the highest value child (i.e. $\max_{s'} V(s')$) at each node to add to the instantaneous reward of the current node s, as is used in dynamic programming for finite horizon MDPs. Note that the backpropagation step does not use a policy itself (like VI), but updates node value estimates that inform future tree policy decisions.

These steps are illustrated in the figure below:



You can find pseudocode for the UCT algorithm, a particular variant of MCTS, on page 9 pf the paper above.

Exercise 9.4. Please discuss how you might implement an MCTS algorithm for this problem.

- Start by defining the four components: Selection, expansion, simulation and backpropagation.
- Note that selection is driven by a sampling policy, and is ϵ-greedy is a simple-to-implement bandit sampling policy that can work effectively.
- Also note that backpropagation is effectively an asynchronous value iteration algorithm running in reverse order through the search tree's state nodes, from the leaf nodes back to the root node.
- `https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/` is another useful resource on MCTS, however, it discusses MCTS applied to complex games like go and chess, which are typically harder problems to solve than MDPs.