Pathfinder - Docs

Table of Contents

- Problem
  - Premise
  - Input
  - Output
- Logic
  - Markov Decision Process
  - Bellman equation
  - Summary

Problem

Premise

Given an $(n \times m)$ map with empty, start, goal, and hazard positions, find the optimal path. A drone uses this information to deliver its payload to the goal position, beginning at the start position. Additionally, the drone can fly North, South, East, and West. When the drone is traveling, there is a 70% chance that the drone moves in the desired direction and a 30% chance the drone flies sideways (15% chance for each side). Finally, if the drone flies over a hazardous position, the drone crashes, and a negative utility of the drone repair cost is received.

Input

- Grid matrix of size $(n \times m)$;
- Start position;
- End position;
- Obstacles;
- Delivery price;
- Power cost;
- Repair cost;
- Discount (gamma value).

Output

What is the optimal next step the drone should take regardless of the current position. This can be used to determine the optimal path from the beginning to the end, as well as the optimal path from any position to the end if the drone is blown off course by the wind or its path is changed by another external source.

Logic

Markov Decision Process:

- Set of States $s \in S$
- Set of Actions $a \in A$
- Probability that a from s leads to s', i.e. $P(s'|s,a)$
- Reward Function $R(s,a,s')$
- Start State
- Terminal State (0.1% Convergence)

$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)\,[R(s,a,s') + \gamma V_k(s')]$

Bellman equation

This expression is the Bellman equation for the value iteration algorithm, which is used in reinforcement learning and Markov Decision Processes (MDPs) to find the optimal policy for an agent.

```
let s =
  0.7 * (-1 * powerCost + discount * values[s_posn[0]][s_posn[1]]) +
  0.15 * (-1 * powerCost + discount * values[w_posn[0]][w_posn[1]]) +
  0.15 * (-1 * powerCost + discount * values[e_posn[0]][e_posn[1]]);

let w =
  0.7 * (-1 * powerCost + discount * values[w_posn[0]][w_posn[1]]) +
  0.15 * (-1 * powerCost + discount * values[n_posn[0]][n_posn[1]]) +
  0.15 * (-1 * powerCost + discount * values[s_posn[0]][s_posn[1]]);

let n =
  0.7 * (-1 * powerCost + discount * values[n_posn[0]][n_posn[1]]) +
  0.15 * (-1 * powerCost + discount * values[w_posn[0]][w_posn[1]]) +
  0.15 * (-1 * powerCost + discount * values[e_posn[0]][e_posn[1]]);

let e =
  0.7 * (-1 * powerCost + discount * values[e_posn[0]][e_posn[1]]) +
  0.15 * (-1 * powerCost + discount * values[n_posn[0]][n_posn[1]]) +
  0.15 * (-1 * powerCost + discount * values[s_posn[0]][s_posn[1]]);
```

- $\sum_{s'}$: This is a summation over all possible next states s'.
- $P(s'|s,a)$: The transition probability function, which gives the probability of transitioning from state s to state s' given action a.
- $R(s,a,s')$: The reward function, representing the immediate reward received after transitioning from state s to state s' due to action a.

- γ: The discount factor, which is a number between 0 and 1 ($0 \leq \gamma < 1$). It determines the importance of future rewards. A higher γ makes future rewards more significant.
- $V_k(s')$: The value of state s' at iteration k. This represents the expected return from state s' if the agent follows the optimal policy thereafter.

```
let moves = [e, n, w, s];
let max_val = Math.max(...moves);
let max_move = moves.indexOf(max_val);
```

- $\max_a$: This denotes the maximum value over all possible actions a. The value iteration algorithm seeks the action aa that maximizes the expected return.

```
values[currPosn[0]][currPosn[1]] = max_val;
policies[currPosn[0]][currPosn[1]] = max_move + 1;
```

- $V_{k+1}(s)$: This represents the updated value of state ss at iteration $k+1$. The value function V gives the expected return (sum of rewards) starting from state s, considering the optimal actions taken thereafter.

Summary

The Bellman equation for value iteration updates the value of each state ss by considering the maximum expected return possible by taking any action a. This return is the sum of the immediate reward $R(s, a, s')$ and the discounted value $\gamma V_k(s')$ of the subsequent state s'. The value iteration process repeats this calculation iteratively until the values converge, which ultimately provides the optimal policy for decision making in the given environment.

Primary sorce