ZUM - Path finding in environment with obstacles using Markov Decision Processes

Nikita Bardatskii

2024, Summer Semester

Briefing

This survey presents a solution to a problem where a drone must navigate through a cave filled with obstacles (stalagmites and stalactites) to safely reach a finish point. The task is a pathfinding problem in a **fully observable 2D environment**, using **Markov Decision Processes** to model the problem and the **Value Iteration** algorithm to solve it. The notebook interactively demonstrates solutions for different hyperparameter values, allowing to observe the effects of tweaking these hyperparameters.

MDP class

There is a base class, `MDP`, from which the `GridMDP` class is derived to work with the 2D environment of the cave with obstacles.

The `MDP` class does not include the algorithm for the solution; it is purely a data class to encode the problem into an MDP representation.

To encode the problem into an MDP, we define:

- **state space** - a set of states where the drone can be.
- **transition model** - which describes the outcome of each action in each state.
- **list of actions** - a set of actions that can be made in each state.
- **rewards** - from each state we can transition to another state, and each transition is associated with a reward value.
- **action probability distribution** - each action has the probability to do certain outcomes. For example, if the probability of moving forward action is **0.8**, then the probability that our drone will be unstable and move in another direction instead is **0.2**. Probability distribution regulates how the drone is cautious and aware of its surroundings, as there is a chance it may not move as intended due to mistakes or external factors, like weather instability or wind.

Value Iteration

The Value Iteration algorithm basically does information propagation. We know that there is some positive reward at the finish state, so by each iteration we propagate that information about positive or negative rewards until equilibrium is reached and each next iteration does almost nothing.

The gamma parameter describes how far the information is propagated. For small gamma the information propagation horizon is short, for higher gamma the horizon is farther, for the maximum value of gamma = **1.0**, agent always considers long-term rewards without regard to how far they are.

We start by calculating utility value for each of the states. The value of each state is the expected sum of discounted future rewards. Then we iterate over and over until utilities start to converge. The Bellman equation is used to calculate and update utility values for each state for every iteration:

$$U_{i+1}(s) = R(s) + \gamma \max_{a \epsilon A(s)} \sum_{s'} P(s' \mid s, a) U_i(s')$$

Value Iteration starts with rewards initial values for the utilities (though it can be any arbitrary values), calculates the right side of the Bellman equation and plugs it into the left hand side, thereby

updating the utility of each state from the utilities of its neighbors. This is repeated until equilibrium is reached.

It works on the principle of *Dynamic Programming* - using precomputed information to simplify the subsequent computation.

The concept of *contraction* successfully explains the convergence of value iteration. In the algorithm, we calculate a value *delta* that measures the difference in the utilities of the current time step and the previous time step.

$$\delta = \max\left(\delta, |U_{i+1}(s) - U_i(s)|\right)$$

This value of delta decreases as the values of U_i converge. We terminate the algorithm if the δ value is less than a threshold value determined by the hyperparameter *epsilon*.

$$\delta < \epsilon \frac{(1-\gamma)}{\gamma}$$

Results

Visualizations are presented as two plots - grid plot, and action probability distribution plot. Interactive sliders and button to print current hyperparameters are also available.

On the grid plot are shown utility values and best policy actions for specific iteration, by default for the last iteration where it converged close enough or exceeded iterations limit. Darker tiles represent obstacles - stalagmites and stalactites, green tile is starting state, blue tile is finish state. Blue arrows represent the best policy action plan from the start state.

On the right plot we see the action distribution. Given an action going in some direction, there is a chance it will go to the intended direction, with some small chance it may go sideways because of some instability/uncertainty of the drone/agent.