ITCS45 Artificial Intelligence

Instructor

Teacher Thanapon Noraset

Solving OpenAI's Gym Classic environments using Hill-climbing search (sideway + restart)

By

Mr. Komsan Kongwongsupak 6288024

Miss Anyamanee Amatyakul 6288060

Mr. Kasidis Chokphaiboon 6288071

Miss Cholravee Kittimethee 6288079

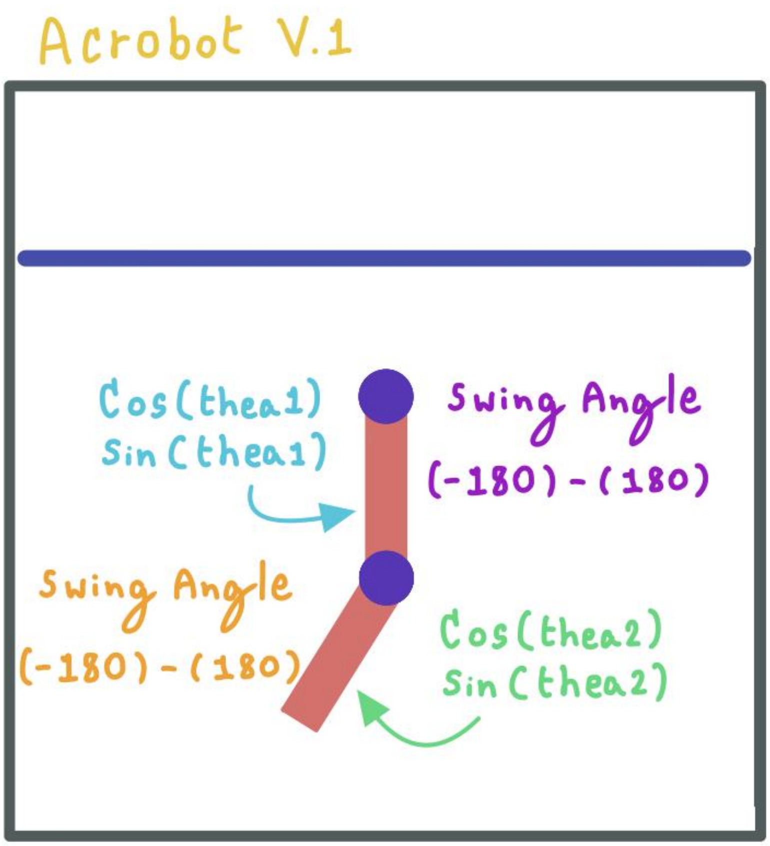
Faculty of Information and Communication Technology

Mahidol University

9 October 2021

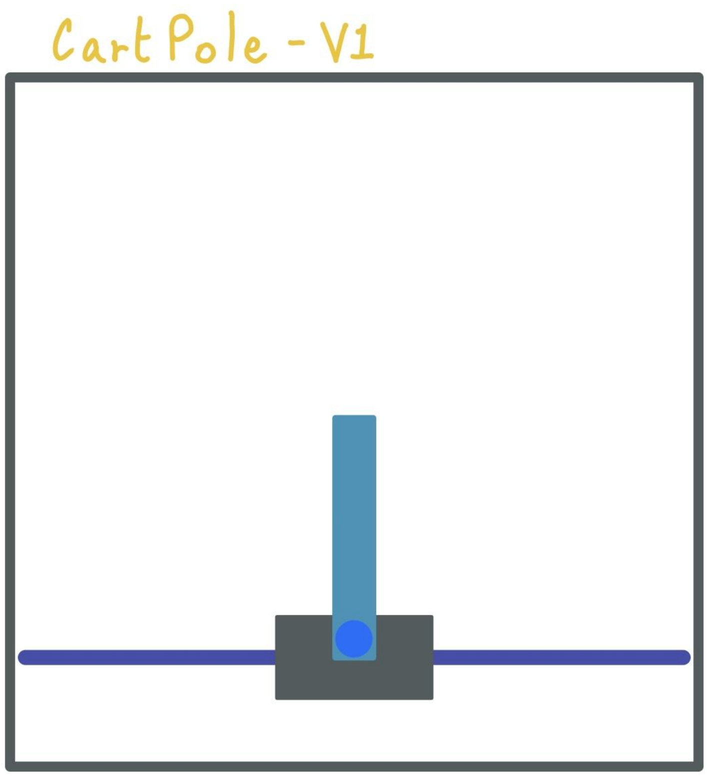
Acrobot V1

The acrobot environment includes two joints and two links, with the joint connecting the two links being operated. Initially, the links are hung downwards, and the goal is to swing the lower links end up to a specific height. The state is made up of the sin() and cos() of the two rotating joint angles, as well as the joint angular velocities: [cos(theta1), sin(theta1), cos(theta2), sin(theta2), V1, V2]. An angle of 0 corresponds to the first link pointing downwards. The second link's angle is relative to the first link's angle. An angle of 0 denotes that the two links have the same angle. A condition of [1, 0, 1, 0,...,...] indicates that both connections are pointing downward. If Sine is negative, the angle will be negative and shift left, but it will shift right if the angle is positive. Plus, if it is true, it will receive 0, and if it is false, it will receive -1 as a reward.



CartPole V0

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. Controlling the mechanism is applying a force of +1 or -1 to the cart. The goal is to keep the pendulum upright and from falling over. Every timestep that the pole remains upright results in a +1 reward. The episode terminates when the pole is more than 12 degrees from vertical or the cart is more than 2.4 units from the center. The state is made up of the car's position, the car's velocity, the angle of the pole, and velocity at the tip of the pole.

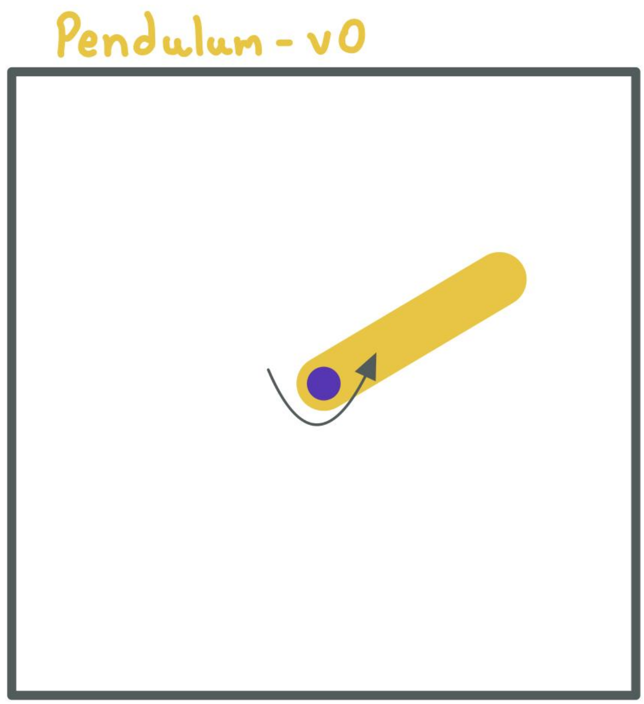


MountainCar V0

A car is on a one-dimensional track, positioned between two mountains. The goal is to drive up the mountain on the right; however, the car has no power to drive up the mountain at the one time. Therefore, it needs to drive back and forth to build up momentum. The less time it spends to reach the peak, the greater reward it will receive. The state is made up of the position of the car and the car's velocity.

Pendulum V0

The inverted pendulum swing-up problem is a classic problem in the control literature. The pendulum starts in a random position, and the goal is to swing it up so that it stays upright. The state is made up of [cos(theta), sin(theta), velocity].



Hill-Climbing Search Algorithm

```python
def hillclimb_sideway(env, agent, max_iters=10000, sideway_limit=10):
    """
    Run a hill-climbing search, and return the final agent.

    Parameters
    ----------
    env : OpenAI Gym Environment.
        A cart-pole environment for the agent.
    agent : CPAgent
        An initial agent.
    max_iters: int
        Maximum number of iterations to search.
    sideway_limit
        Number of sideway move to make before terminating.
        Note that the sideway count reset after a new better neighbor
        has been found.

    Returns
    ----------
    final_agent : CPAgent
        The final agent.
    history : List[float]
        A list containing the scores of the best neighbors of
        all iterations. It must include the last one that causes
        the algorithm to stop.

    """
    cur_agent = agent
    cur_r = simulate(env, [agent])[0]

    explored = set()
    explored.add(cur_agent)
    history = [cur_r]

    for __ in range(max_iters):
        env.render()
        # TODO 1: Implement hill climbing search with sideway move.
        neighbors = cur_agent.neighbors()
        _n = []
        for a in neighbors:
            if a not in explored:  # we do not want to move to previously explored ones.
                _n.append(a)
        neighbors = _n
        rewards = simulate(env, neighbors)
        best_i = np.argmax(rewards)
        history.append(rewards[best_i])
        if rewards[best_i] < cur_r:
            return cur_agent, history
        #   Sideway move
        elif rewards[best_i] == cur_r:
            for __ in range(sideway_limit):
                rewards = simulate(env, neighbors)
                equal_i = np.argmax(rewards)
                history.append(rewards[equal_i])
                if rewards[equal_i] < cur_r:
                    best_i = equal_i
                    break
            return cur_agent, history
        #
        cur_agent = neighbors[best_i]
        cur_r = rewards[best_i]

        # pass
    return cur_agent, history
```

According to the code in Hillclimb, it finds the highest neighbor and returns that agent. However, in the hillclimb_sideway, if it finds a higher neighbor, it will stop and walk sideways a few times. If the program walks sideway ten-time but cannot find the higher neighbor, it will return to the current agent.

```python

def hillclimb_restart(env, agent):
    """Run a hill-climbing search, and return the final agent."""
    best_agent, rewards = hillclimb_sideway(env, agent)
    best_reward = max(rewards)
    for __ in range(30):
        cur_agent = ACAgent()
        temp_agent, history = hillclimb_sideway(env, cur_agent)
        reward = max(history)
        if best_reward < reward:
            best_agent = temp_agent
            best_reward = reward

    return best_agent, history

```

In hillclimbing_restart, we run time first to collect the initial reward of the initial agent into our rewards log. Then we will run thirty more times, find the best result, and collect the higher agent into our log until we find the best agent. (It will not be the same agent) P.S. We run code thirty times to guarantee the goal.

Testing result

Acrobot (input-output-average)

	Cos (theta1)	Sin (theta1)	Cos (theta2)	Sin (theta2)	Velocity 1	Velocity 2	Bias	Rewards
Round 1	-0.0289	0.072	0.0391	-0.115	0.076	-0.000841	-0.0208	-298
Round 2	-0.000721	0.0536	0.00617	0.119	-0.083	-0.0434	-0.03	-298
Round 3	0.0332	0.0674	-0.163	-0.072	0.055	0.00925	-0.152	-298
Round 4	-0.217	-0.019	0.00338	-0.0812	-0.0149	0.0485	-0.049	-298
Round 5	-0.154	0.0454	0.0349	0.0571	0.0142	0.0286	-0.0451	-298
Average								-298

Cartpole (input-output-average)

	Cart position	Cart velocity	Pole angle	Pole velocity at tip	Bias	Rewards
Round 1	0.0999	-0.0877	0.085	0.0586	0.00779	149
Round 2	0.0565	0.0608	0.0865	0.686	-0.000163	500
Round 3	0.0113	-0.0268	0.134	0.105	0.00519	500
Round 4	0.0268	0.00757	0.237	0.152	-0.00223	500
Round 5	0.005	0.0757	0.154	0.00582	-0.0174	500
Average						429.8

MountainCar (input-output-average)

	Car position	Car velocity	Bias	Rewards
Round 1	0.209	0.127	0.0949	-120
Round 2	0.0354	-0.0331	-0.137	-120
Round 3	0.112	0.141	0.0778	-120
Round 4	0.0562	0.0141	-0.122	-120
Round 5	0.236	0.0377	-0.141	-120
Average				-120

Pendulum (input-output-average)

	cos(theta)	sin(theta)	Velocity	Bias	Rewards
Round 1	0.232	-0.0906	0.0237	-0.11	-568.33
Round 2	-0.17	-0.141	0.00426	-0.182	-617.69
Round 3	0.0631	-0.146	0.0258	-0.107	-566.45
Round 4	0.0473	0.0811	0.0266	-0.13	-500.71
Round 5	-0.0667	-0.113	0.0143	-0.0613	-623.87
Average					-575.41

Credits

The original code: Aj.Thanapon Noraset

Implemented code: Mr.Komsan Kongwongsupak and Miss Anyamanee Amatyakul

Environments: OpenAI Gym (Gym (openai.com))

Hill-Climbing Search Sideway and Random-Restart Algorithm: Mr.Komsan Kongwongsupak and Miss Anyamanee Amatyakul

Description and Testing: Mr.Kasidis Chokphaiboon and Miss Cholravee Kittimethee