ECE 276B
Project 1

Markov Processes and Dynamic Programming

Due Date: Feb 30, 2018

Professor: Nikolay Atanasov

Chuqiao Song

A53239614

Introduction

In tic-tac-toe problem, the dynamic programming method is being used to solve this 3*3 board game. To find the maximal wining probability of this chess problem, it should be considered backwards. Meanwhile, link the children layer to its father layer, and inherit its father's rewards, say link step 9 to step8. Finally, do this procedure recursively, and at the last layer, first step, the maximal reward of this layer should be the maximal wining probability for this problem. There are four conditions in this problem: 'me' going first condition and opponent going first condition, with whether considering losing or not

In short path problem, the dynamic programming algorithm also being used. To find the shortest path of given graph between two points, it is does not matter consider backwards or forwards. In each iteration, every edge cost of given starting points should be consider; find up to now costs for each possible path. Meanwhile, the optimal paths should be selected for next iteration. At terminal point, check the cost of each possible path to select minimal cost path as the shortest path solution.

Problem Formulation

1) Tic-Tac-Toe

Terminal and stage cost: This board game can be solved using DP algorithm by first defining terminal cost and stage cost. It should be considered in two conditions one is consider losing, and the other only consider winning and losing does not matter.

Noise: The opponent in this game can be treated as noise, and the rule opponent follows to put its chess onto the left 3*3 chess board space is a noise for decision, and it has to pre-define.

Optimal Policy: Using Markov Decision Processes to find optimal policy is an important step to maximize the wining probability. To get this policy, it has to think backwards and consider sub-problem sub-problem maximize the reward function $J_t^{\pi}(x_t)$ from given final stage to its first stage.

Motion model: For each people going step, people have to use decided optimal policy combining with motion model based on present situation to find the grid on board which can maximize it winning probability; it is an action step.

2) Shortest Path

Terminal and stage cost: Given the directed edge cost table, the goal is to use this table to find the shortest path from starting point to ending point.

Optimal Policy: Using principle of optimality, consider subproblem by subproblem to get optimal policy to minimize the path cost.

Motion model: It is just an action step, combined with decided optimal policy to go from starting point to end point using shortest path. Additionally, the noise should not be considered, since it is a deterministic problem, given directed edge cost table.

Technical Approach

1) Tic-Tac-Toe

Considering the chess board like this,

1	2	3
4	5	6
7	8	9

the number represents the position. It is winning when 123, 456, 789, 369, 147, 258, 159, and 357 has three same marks.

A. Terminal and Stage cost

In this problem, there are two conditions: the first one is only considering win, and that is terminal cost $g_T = 1$ for winning, and $g_T = 0$ for losing and drawn; the second one is both considering winning and losing, and that is $g_T = 1$ for winning, g_T = -1 for losing, and g_T=0 for drawn. Moreover, the stage cost for this board problem is set to 0 to let the first step winning probability is the reward inherited from final step.

B. Noise

The noise for this problem comes from opponent, and the opponent choose randomly from all available positions. Therefore, in this problem there are only two turns: when in a 'me' turn, it is the deterministic problem and the 'me' would choose an available position to maximize the winning probability; when in an opponent turn, the probability of winning for 'me' is unknown.

C. Optimal Policy

Considering me-first condition, to find optimal policy for each 'me' turn, the value function has to be defined and maximizing. To some extent, it is the instance of principle of optimality, by maximizing subproblem value function. In this problem, the discount factor and the last stage cost terms are set to be 1 and 0 respectively.

$$J_t^{\pi}(x_t) = \mathbb{E}_{x_{t+1:T}}\left[\gamma^{T-t} g_T(x_T) + \sum_{\tau=t}^{T-1} \gamma^{\tau-t} g_\tau(x_t, \pi_t(u_t)) \,\middle|\, x_t\right]$$

Basically, it is a finite-horizon dynamic programming problem, so there are total 9 states and 4 subproblem value functions for 'me', when consider from backwards. By maximizing these value function, the optimal policy for 'me' can be found.

To realize this value function, the inherit function is defined in this project, and it inherits rewards from step 9 to each previous step to step 1. Thinking it backwards from final winning arrangements and assign all these possible outcomes with reward 1; then generate step 8 all possible arrangements and find the relationship between step 8 arrangements and step 9 arrangements, who can inherit the reward from its 'father' in step 8. The father and children term here mean: the one father in step 9 generates some arrangements in step 8 called children if picking one associated chess out from board randomly. more specifically, if step 8 to step 9 is 'me' turn taking out 'x' from step 9 to step8, otherwise taking out 'o' when it is opponent turn. To some extent, in other step, one child in step k would have more than one fathers in step k+1. Finally, do this procedure recursively, the rewards of first step with 9 kinds of arrangement can be found. ('-------x-', '--x------', 'x--------', '-----x---', '------x--', '---x--

---', '-x-------', '--------x', '----x----') In this problem for any event A, $\mathbb{P}(A) = \mathbb{E}\{\mathbb{1}(A)\}$, therefore the maximal final reward in these 9 arrangements is the maximal winning probability under 'me' going first condition. Specifically, the inheriting rule in this function is that: if it is 'me' turn, this step would inherit maximal reward from next step with associated 'children-father relationship'; for opponent turn checking either 'me' win or not, and if 'me' already win give this child arrangement reward 1, otherwise give this child average associated fathers rewards. To determine the optimal policy, all of above content is considering backwards from final layer to first layer, and the procedure above is similar when consider opponent-first condition.

D. Motion model

When running the game, it is a forward procedure, and if it is me turn the maximal wining probability location can be chosen based on decide optimal policy, and so on next me turn step. Basically, the optimal policy is this problem can be generated as a table with associated rewards for each 'me' turn step. To winning the game, I only have to check the table find the maximal reward chess position to put my chess on.

$$\begin{aligned} \text{s.t. } & x_{t+1} \sim p_f(\cdot \mid x_t, \pi_t(x_t)), \\ & x_t \in \mathcal{X}, \quad \pi_t(x_t) \in \mathcal{U}(x_t) \end{aligned}$$

2) Shortest Path

Given starting point and end point, thinking from backwards or forwards using dynamic programming method does not matter the solution, as either one can be treated as end.

A. Terminal and Stage cost

In this problem, there is no terminal cost, and the stage cost is given by the input file, with directed edge cost table. In the running algorithm, to calculate the value function, we just have to look up the table

B. Optimal Policy

To find the optimal policy, the dynamic programming algorithm is used in this problem, as following procedure, where $V_T(x_T) = 0$, $Q_t(x_t)$ is the value function for each iteration, $\pi_t(x_t)$ is the optimal policy for this iteration.

Algorithm 1 Dynamic Programming

Input: MDP $(\mathcal{X}, \mathcal{U}, p_f, g, \gamma)$, initial state $x_0 \in \mathcal{X}$, and horizon T

$V_T(x) = g_T(x), \quad \forall x \in \mathcal{X}$

for $t = (T-1) \dots 0$ **do**

  $Q_t(x,u) \leftarrow g_t(x,u) + \gamma \mathbb{E}_{x' \sim p_f(\cdot|x,u)}[V_{t+1}(x')], \quad \forall x \in \mathcal{X}, u \in \mathcal{U}(x)$

  $V_t(x) = \min_{u \in \mathcal{U}(x)} Q_t(x,u), \qquad \forall x \in \mathcal{X}$

  $\pi_t(x) = \arg\min_{u \in \mathcal{U}(x)} Q_t(x,u), \qquad \forall x \in \mathcal{X}$

return policy $\pi_{0:T-1}$ and value function V_0

To find the shortest path, each iteration the function goes further possible edge from 'father nodes' to all possible children nodes, and it refresh the cost function by adding corresponding edge cost from root node (if forward it is starting point) to the children nodes, and it updates the children as next iteration father nodes. Meanwhile, I implement two filtering function here to shrink the path from root node to children nodes. For first filter function, it unique all children nodes by choosing corresponding minimal cost path, so that one node corresponding one path and it is minimal cost path for next iteration. For second filter function, it loops all shrink present children nodes and check whether this node already appears in previous path, if it is check the cost for this present path and the cost of previous path from the root node to this same node. If the present path is larger than the other, than delete this node and this path and its path cost, other otherwise keep the nodes and path and its path cost. Finally, if one path arrives the goal point, and its path cost is minimal between other cost, then break the iteration; otherwise keep the iteration.

C. Motion model

It is same as previous problem, given the starting point and end point, the motion model will follow the policy based on present node to keep on the shortest path.

Result

1) Tic-Tac-Toe

Calculation	Not consider losing	Consider losing
Me first	0.99479	0.99479
Opponent first	0.9386	0.9386

Iteration = 10000

Simulation	Not consider losing	Consider losing
Me first	0.9914	0.9914
Opponent first	0.936	0.936

Basically, the simulation has the same results as the results from dynamic programming calculation. The probability of wining does not matter whether I consider losing or not, and it is actually same. Intuitively, it is because, the algorithm always chooses maximum rewards to inherit to next step. However, the opponent first or me first really influence the wining probability, since the first step is undecided, where I have to put my chess on depends on where opponents put. Generally, if I play first, I will always win or tie the game and never losing.

2) Shortest Path

For input1 file the result is following:

Path [43.0, 52.0, 60.0, 69.0, 70.0, 71.0, 80.0, 81.0, 82.0, 83.0, 84.0, 85.0, 86.0, 87.0, 88.0, 99.0, 110.0]
Cost [16.0]
Optimal value [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

For input2 file, the result is following:

[12.0, 23.0, 34.0, 45.0, 56.0, 67.0, 68.0, 69.0, 70.0, 71.0, 72.0, 73.0, 84.0, 85.0, 86.0, 97.0, 98.0, 109.0, 110.0, 121.0]
[82.518007]
[0, 3, 6, 10, 14, 18, 24, 32, 41, 50, 58, 64, 70, 74, 77, 79, 80, 81, 82, 82]

Comparing with the output file it is same, the dynamic programming algorithm really perform well. It searches the path from the starting nodes and finds the minimal cost for each iteration and keep it. The main difference between these two input files is that: the edge cost in first one is always one; however, in second file, the edge cost is different. Therefore, the final optimal cost for two files are different.