Project 1: Markov Processes and Dynamic Programming

Ibrahim Akbar, University of California, San Diego
ECE276B: Planning & Learning in Robotics, Winter 2018

***Abstract*—It is important for many systems to understand what the optimal decision is under given circumstances. This allows for the reduction of errors, an increase in the life span of the system, and overall improvement in performance. The difficulty that lies in understanding how to act optimal is both in understanding how to models the system's states, controls, and observations that affect the decision and the way optimal is defined. In this report I will cover how to define a model using Markov Decision Processes in order for a method known as Dynamic Programming to be defined and allow us to determine an optimal policy for a tic tac toe game or for a deterministic shortest path problem.**

I. Introduction

The state of a system or the environment of the system should be taken into consideration when determining what the best possible control is in that state. In order to consider even making a decision we first have to understand how to model these states, the transitions between states, and the controls available. Then we can determine how to represent a value for the possible controls at various states. A common process used to model the variations of a system's state is a Markov Process which provides us with the Markov assumption of "memorylessness" that eases the computational complexity of the model. With this process we can define a state space in which it resides and a function that corresponds to the transitioning of the process to different states in the space. Now if we consider this environment or system to accept inputs in the form of controls then this process becomes a Markov Decision Process (MDP) which means that our function for transitioning has an another parameter in the form of the control input. If asked to determine the optimal controls at each state given a certain optimality measure we could use an algorithm called Dynamic Programming to produce an optimal policy for this MDP. In this report I am going to consider the scenario where we have a stochastic and deterministic MDP for which we would like to develop optimal policies. First I will determine what problem we are trying to solve for both the tic tac toe game and shortest path problem. Then I will discuss my approach to these problems through the formulation of an MDP and applying Dynamic Programming onto the process. Finally I will present my results and elaborate on them.

II. Problem Formulations

A. Tic Tac Toe

We are given that we want to determine the optimal policy for playing tic tac toe against a player who has controls that can be modeled with a discrete uniform distribution. Thus we want,

$$\pi^*(x) \;\text{ s.t. }\; J_0^{\pi^x} < J_0^{\pi} \;\forall \pi(x),\; x \in \mathcal{X}$$

where J_0^{π} denotes the value of a policy. To determine a policy we first need to know how the MDP behaves. Thus we need to define a state space $\mathcal{X}$ such that the control space $\mathcal{U}$ and transition function $f(x, u, w)$ are well defined. Where $x \in \mathcal{X}$, $u \in \mathcal{U}$, and $w \sim U\{\cdot\}$. Once we have developed a model that accurately represents the process we can consider how to define optimality. Optimality can be defined through the use of a cost/reward function $g(x, u)$ that penalizes/rewards states and controls that adhere to the system specified constraints. How we define $g(x, u)$ will determine what our policy believes to be optimal for various states.

B. Shortest Path

Given a deterministic directed graph, a start node, and goal node using dynamic programming we would like to find the optimal policy for reaching this goal node. Since this is a deterministic setting we are capable of transforming a shortest path problem into a deterministic finite state (DFS) control problem. Thus we can apply a similar question as with the tic tac toe problem and ask how do we need to represent this graph in order for the process to be well defined and allow us to apply a cost/reward function. Given a set of nodes $n \in \mathcal{V}$ and a set of edges $e \in \mathcal{E}$ where we want to go from n_0 to n_τ, how can we express this as a DFS that allows us to define a $g(x, u)$ and apply dynamic programming.

III. Technical Approaches

A. Tic Tac Toe Model Definition

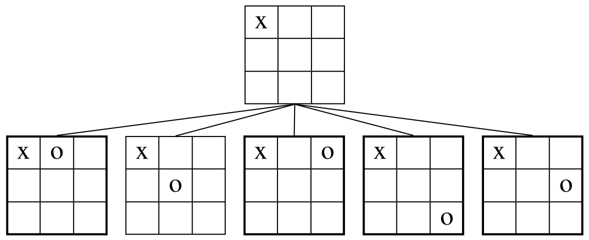
I will make the assumption that all games always start from the beginning rather than any possible state, therefore we can say, $p_{0|0} = 1$. Let's also denote the initial state x_0 as an empty board.

$$x_0 = \{0\}^{3\times 3}$$

$$x_t \in \{-1, 0, 1\}^{3\times 3} = \mathcal{X}$$

In order to know size of our space $\mathcal{X}$ we have to determine when the game would end. A game can only be comprised of 9 moves as such we know that our horizon is $\tau = 10$. We can then say that the game ends if the horizon is reached or an end game condition is achieved. An end game condition is defined as having three identical marks in a row either vertically, horizontally, or diagonally. Thus we know that $\mathcal{X}$

should contain all states from x_0 to x_τ, with the exception of states that are only attained after reaching an end condition. Since this game is a two player game where player's play sequentially; states where $|m_1(t+1)| > |m_2(t)| + 1$ can also be discarded. Here $|m_1(t)|$ is the number of moves made by the first player and $|m_2(t)|$ is the number of moves made by the second player. To decrease the state space even further we can rely on symmetry to explain a few variations,



The first state that is only occupied by an X is enough to describe all other state with a single X in the corner by a simple rotation about an axis through the center square. The same ideas applies to next possible states; however now the axis may be different. Using this idea of symmetry we can reduce the amount of states when $t < 4$.
Given our complete state space $\mathcal{X}$ we can now think about the controls that are possible. In general a control may be one of the players making a mark in one of the locations of the board. That means that each player would have a total of 9 possible controls. Since we know that player 2's controls can also be modeled with a uniform distribution I will denote their controls as $w(t)$. Hence our control space reduces to player 1's controls. If we denote the control of player 1 as 1 and for player 2 as -1, player 1 control space would be,

$$\mathcal{U}_1 = \{1_1, 1_2, 1_3, 1_4, 1_5, 1_6, 1_7, 1_8, 1_9\}$$

With these controls we can derive a function that acts as a transition function between states. We know that players move sequential through time thus if we assume player 1 starts the game,

$$f(x(t), u(t), w(t)) = \begin{cases} x(t) + u(t) & t \in \{1,3,5,7,9\} \\ x(t) + w(t) & t \in \{2,4,6,8\} \end{cases}$$

$$u(t) \in \mathcal{U}_1(x(t)),\ x(t) \in \mathcal{X},\ w(t) \sim U\{\mathcal{U}_2(x(t))\}$$

The times t can be switched to indicate player 2 starting the game. With this transfer function I am also stating that, $\mathbb{P}(u(t) = 1_n | x(t)) = 1$, where n is the desired location. This is under the assumption that their is no noise involved. While for player 2

$$\mathbb{P}(w(t) = -1_n | x(t)) = \frac{1}{|\mathcal{U}_2(x(t))|}$$

Thus for each subset of the control space $\mathcal{U}_1$ and $\mathcal{U}_2$ we can write a transition matrix P^1_{ij} where i, j denotes transitioning from state j to state i and the superscript is the subset of the control space this transition matrix belongs to.

B. Tic Tac Toe Cost & Value Function

Having determined a model for this process we can now elaborate on the idea of how we want to determine an optimal policy. This requires defining a cost function $g(x(t), u(t))$. Since this is a game there should be no intermediate costs as a system is only concerned with achieving a winning state. Thus we are able to write,

$$g_t(x(t), u(t)) = 0,\ \forall t < \tau$$

$$g_\tau(x(\tau), u(\tau)) = \begin{cases} -1 & x(t) \in \mathcal{X}_w \\ 0 & x(t) \in \mathcal{X}_d \\ 1 & x(t) \in \mathcal{X}_l \end{cases}$$

where $\mathcal{X}_w$ is the subspace containing all winning end states, $\mathcal{X}_d$ contains all drawing end states, and $\mathcal{X}_l$ contains all losing end states. If we do not differentiate between losing and drawing we can assign the same cost of either 0 or 1 to both subspace. Now we can define our value function as,

$$V_t(x(t)) = \mathbb{E}\left[g_\tau(x(\tau), u(\tau)) + \sum_{n=t+1}^{\tau-1} g_n(x(n), u(n)) | x(t) \right]$$

which due to intermediate states having no cost reduces to,

$$V_t(x(t)) = \mathbb{E}[g_\tau(x(\tau), u(\tau))]$$

This implies that our value function for determining the optimal policy is simply computing the expected cost of the possible terminal states at each intermediate state. Since our transition matrix is deterministic when it is the system's turn this equates to propagating the minimum expected value from the future state. Whereas when the system's noise is playing we are required to find an expectancy over all possible terminal states. Using this definition for a model, cost, and value function I am able to apply the dynamic programming algorithm to determine the optimal policy by minimizing over the expected value calculated from the value function,

$$V_t^*(x(t)) = \min_{u(t) \in \mathcal{U}} \mathbb{E}[g_\tau(x(\tau), u(\tau))]$$

C. Shortest Path Model Definition

In order to define a model that allows us to apply dynamic programming we first need to determine how we can define the transformation from a shortest path problem to a DFS control problem. That means we again need to define $\mathcal{X}$, $\mathcal{U}$, $f(x(t), u(t))$ in terms of the current graph definition. The nodes within the graph would constitute all the potential states of our system, meaning that,

$$\mathcal{X} = \mathcal{V}$$

As we require a initial state and goal state, I will also define those as,

$$x_0 := n_0 \quad x_\tau := n_\tau$$

With this definition we can also define a subspace of $\mathcal{X}$ as,

$$\mathcal{X}_t = \mathcal{V} \backslash \{n_\tau\},\ t < \tau$$

where $\tau = |\mathcal{V}| - 1$ since in the worst case scenario a system could reach the goal node in the total amount of edges of the graph. As this is a graph and there are no direct controls, I will interpret the controls as being a teleportation from one state to another. This also means that we can define the control space as being the same as the node space.

$$\mathcal{U} = \mathcal{V}$$

It is important to note that at time $\tau - 1$ the only possible control is to move to the goal state,

$$u_{\tau-1} = n_\tau \quad \mathcal{U}_t = \mathcal{V} \backslash \{n_\tau\},\ t < \tau - 1$$

As the control is identical to the state space it also implies that our transition function would simply be the control stating which state is the next one.

$$f(x(t), u(t)) = u(t)$$

This is a deterministic problem and therefore we are not concerned with any noise and can state such a direct transition function.

D. Shortest Path Cost & Value Function

This time in contrast to the game scenario if we are applying Dynamic Programming as a backwards pass the goal state would have no cost because you have arrived at where you would like to be and there is no cost to stay, while all the intermediate states would have a cost. We can denote this cost as the weight that corresponds to edge connecting two different nodes,

$$g_\tau(x(\tau)) = 0$$

$$g_t(x(t), u(t)) = e_{ij}$$

where e_{ij} is the edge weight between n_i and n_j. With this cost we can again have the same value function which reduces to the other term,

$$V_t(x(t)) = \mathbb{E}\left[\sum_{n=t}^{\tau-1} g(x(n), u(n))\right]$$

Using the Dynamic program algorithm we again would like to minimize the cost that is required to reach the goal node.

$$V_t^*(x(t)) = \min_{u(t)\in\mathcal{U}} \mathbb{E}\left[\sum_{n=t}^{\tau-1} g(x(n), u(n))\right]$$

IV. Results

A. Tic Tac Toe

Optimal Policy Probabilities: Do not care about losing

Start Time	DP: Max. Prob.	Sim: Max. Prob.
t = 1	0.9917	0.9923
t = 2	0.9127	0.9192

Optimal Policy Probabilities: Do care about losing

Start Time	DP: Max. Prob.	Sim: Max. Prob.
t = 1	0.9917	0.9909
t = 2	0.8778	0.8749

Simulation Probabilities: Do not care about losing

Simulation Runs	t = 1	t = 2
r = 10	0.9	1.0
r = 100	0.98	0.93
r = 500	0.996	0.906
r = 1000	0.99	0.913
r = 10000	0.9923	0.9107

Simulation Probabilities: Do care about losing

Simulation Runs	t = 1	t = 2
r = 10	1.0	0.8
r = 100	1.0	0.89
r = 500	0.996	0.892
r = 1000	0.993	0.877
r = 10000	0.9935	0.8806

B. Shortest Path

Path	Cost
43	16
44	15
45	14
54	13
62	12
71	11
80	10
81	9
82	8
83	7
84	6
85	5
86	4
87	3
88	2
99	1
110	0

Table I
INPUT #1

Path	Cost
12	82.51800700000003
23	79.30526500000002
34	75.62901200000002
45	71.82755700000001
56	67.97264200000001
67	63.687595
68	57.904296
69	50.220302000000004
70	41.354928
71	32.400658
72	24.334032
73	17.588478000000002
84	12.190735
85	8.036792
86	4.993704
97	2.906599
98	1.552351
109	0.725438
110	0.252471
121	0

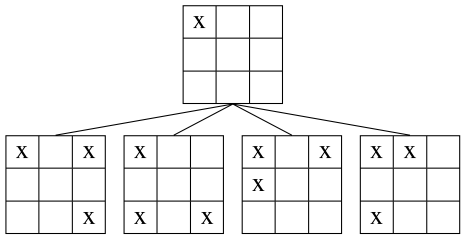
Table II
INPUT #2

V. Discussion

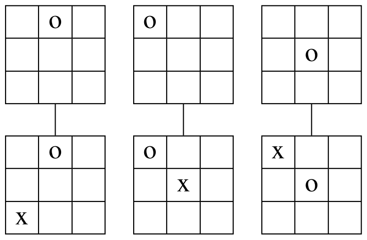
A. Tic Tac Toe

When computing the maximum probability it corresponds to the likelihood of encountering the desired output. This means that $\mathbb{P}(A) = \mathbb{E}[\mathbb{1}(A)]$, where $\mathbb{1}(\cdot)$ is the indicator function. The way I devised to obtain this probability is through setting the winning states with a value of 1 and all other states with a value of 0 when losing and drawing were equal. Thus since the value function is taking an expectancy over the future values, the value function is determining the optimal value which corresponds to the maximum probability of winning. For the four scenarios that are considered each of the optimal policies differs, this is because either starting first or second already dictates which possible states the system will now be able to apply a control to and these are disjoint respective of starting either first or second. On top of that variation the option to differentiate losing and drawing cause a change to the understanding of preference ranking.

When there is no differentiation all that is optimal is winning, thus either control for going to a losing or drawing state is treated equally. While when the system differentiates between the two it considers drawing a priority over losing so now when those are the only options that exist given the current state it would choose to draw. Which is verified based on the simulation results as the system has 0 probability of losing and in the worst case scenario goes to a draw state. The initial control when starting first is to always place the mark in the corner. Intuitively, this makes sense since you would like to achieve a state where no matter the noise input the desired output would remain the same.



Given any of these states because the other player only is allowed one control per time step the opportunity to win is not diminished. Whereas if the control were to be in the center square: 5 then it would not have this guarantee of achieving a desired output irrespective of noise towards the end.
When starting second the policy is still attempting to achieve this sort of state but now it has to be aware that noise is being introduced first so it could start in a variety of states. The noise is capable of placing a control on the locations of either: corner, center, or side. Thus given these scenarios the system would determine what the likelihood of each state is and have an optimal control for those situations.



From the initial states that the system could be in these are the optimal controls that policy generates for the system's first input when not differentiating between drawing and losing. In terms of maximum probability of winning from highest to lowest they are ordered from left to right. The state on the right is still able to generate states similar to conditions mentioned above where the system will win irrespective of noise, while the other two states are not capable of this anymore because there is a mark in position 5. The system prefers the center condition over the left condition because it allows the system to dictate what control the noise should ideally make; however since it is noise, there is a likelihood that it will not choose said control allowing for the system to win. Since the system is guiding the noise's potential moves it can also enforce a draw if the noise happens to follow those control inputs. Whereas for the left condition it is the opposite and the system is counteracting each control input from the noise till it has chance to win or draws.
Given that the system differentiates between drawing and losing there is no difference in the initial control when the system is starting first, but when it starts second it makes the control in the previous most preferable state to also be in position 5. My understanding is that since the system prefers to draw over losing and playing in the center square still guarantees a chance at winning it makes this move because it reduces the chance of losing to zero by now being able to draw the game in all scenarios.

B. Shortest Path

Since the shortest path problem was in deterministic setting this problem reduced to propagating the costs based on the goal node throughout the various states till you had a cost for each node in the graph saying how much is the cost to arrive. It served as a good practice to see how transferable the understanding between a DFS and shortest path problem are when deterministic.