ECE 276B PR1: DPA for Autonomous Navigation

Varun Vupparige
Mechanical and Aerospace Engineering
University of California San Diego
La Jolla, USA

I. Introduction

The progress of robotic systems in the past decades provides robots with capabilities to operate in human populated environments. One of the major challenges on the way to obtain the objective of a robot co-worker is to ensure a safe and reliable operation of robots. Generally, a robotic system is composed of different subsystems such as vision, compute, localization, path planning and control algorithms. The objective of a path planning algorithm is to generate a safe path from an agent/robot's starting position to its goal.A working environment of an agent can be classified as a completely known environment, a partially known environment, a completely unknown environment and a dynamic environment.

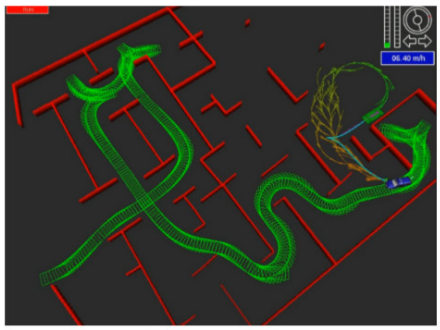

Fig. 1. A*: Popular planning algorithm

The planned path of the agent can be evaluated based on metrics such as path length, movement time, energy consumption and risk levels. Increased adoption of robotic systems in an human populated environment has posed particular challenges for safe path planning. One of the commonly used methods to find an optimal path from start to goal is Dynamic Programming. The term dynamic programming was originally used in the 1940s by Richard Bellman to describe the process of solving problems where one needs to find the best decisions one after another. By 1953, he refined this to the modern meaning, referring specifically to nesting smaller decision problems inside larger decisions. From an high level view, this algorithm involves formulating the control actions, state space and value function as a Markov decision process, indicating all the inter-state transitions (and their respective costs/reward) possible for a given environment and use techniques like value iteration, label correction, etc. to find the path that has the least cost or the highest reward.

In this project, a agent is trying to find an optimal path in an min grid environment consisting of start position, door, key and goal. A agent is capable of executing five control actions: Move Forward (MF), Turn Right (TR), Turn Left (TL), Pick Key (PK) and Unlock Door (UD). Parameters such as goal position, door position and key position are known to the agent in one set of environments and unknown in random environments. The main objective is to find a control policy to traverse the agent from its start position to goal position optimally.

II. Problem Formulation

In this section, the problem of autonomous navigation in a door and key environment is formulated as Markov Decision Process (MDP) by carefully explaining each parameter of MDP in detail:

A. Markov Decision Process

A Markov Decision process is a framework for high level decision making problems which is formulated based on the following Markov Assumption (in context of a robotics problem): "Your next state x_{t+1} is dependent only on the previous state x_t and control action u_t." This mathematical framework is used to solve for problem both probabilistic and deterministic in nature. A typical Markov Decision Process is composed of following parameters:

- X: Discrete/continuous set of states
- U: Discrete/continuous set of controls
- p_o: Prior probability density function/mass function defined on X
- $p_f(x_{t+1}|x_t, u_t)$: conditional pmf/pdf based on Markov Assumption
- T: Planning horizon
- $l(x, u)$: stage cost of applying control u on state x
- $q(x)$: Terminal cost of being in state x at time T
- y: Discount factor ranging from 0 to 1

Lets formulate the parameters of the Markov Decision Process in context of our project:

1) Environment: The environment is the surrounding with which the agent interacts (for eg: the house where the Robot moves). The agent cannot manipulate the environment; it can only control its own actions.

The red pointer denotes the current location of agent

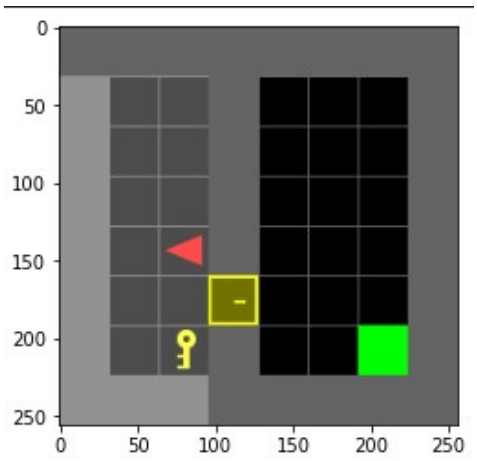

Fig. 2. Sample Mini-grid Environment

and its orientation. The key location, door location and goal is also depicted in the figure below. The agent can pick up key or unlock door only if the agent is infront and facing it.

2) State Space: In MDPs, the state defines the current situation of the agent. For example, it might specify information about agent's position, orientation. For our project, the state of the agent can be described by: agent's location (x, y), agent's orientation vector, door status, key status for the known environments. Let x_t be the state of the agent in grid environment which can be explained as follows:

$$\begin{Bmatrix} \text{x} & \text{y} & \theta & doorstatus & keystatus \end{Bmatrix} \tag{1}$$

For an 8x8 grid, the state space X consists of 1024 probable states. Thus the size of state space is equal to the product of sizes of grid width, grid height, orientation vector, door status and key status.

3) Control Space: The control space $\mathcal{U}$ is the set of given actions taken by the agent to reach the final goal (τ). The control is u is taken to reach these neighboring states. Five control actions: Move Forward (MF), Turn Right (TR), Turn Left (TL), Pick Key (PK) and Unlock Door (UD).

4) Motion Model: Based on the definition of state space X and control space U, we can now define the motion model. Given a robot's state x_t, and control input u_t, the state x_{t+1} can be modelled as a probabilistic/deterministic function given by,

$$x_{t+1} = f(x_t, u_t) = p_f(.|x_t, u_t) \tag{2}$$

Since our project is deterministic in nature, we can discard the probability mass/density function. For a particular state, there can be five possible next states based on the control actions.

5) Finite/Infinite Horizon: The trajectories which terminate at fixed T ¡ $infinity$ are termed as finite-horizon. In this project, since the DPA algorithm terminates when the agent reaches its goal, it is a finite horizon problem.

6) Stage/Terminal Cost: In DPA algorithm, there is cost for choosing any control u in state x. For our project we place a cost of one for each valid control actions given that it results in a valid state. The terminal cost is the state cost at stage/horizon when the DPA algorithm terminates.
In summary.....

III. Technical Approach

In this section, the implementation of dynamic programming algorithm to compute optimal policy for the agent in the door and key environments is explained in detail. This section also introduces the mathematical framework of dynamic programming algorithm to solve a finite horizon optimal control problem.

A. Dynamic Programming Algorithm

The approach of this algorithm is to divide a given problem into smaller sub-problems. From the principle of optimality which states that the optimal policy of a given problem is a subset of optimal policy of a sub problem. It obtains an optimal control policy for a MDP problem. Using the value function computation, the algorithm searches for optimal policies. The objective of this algorithm can mathematically expressed as:

$$\pi^* = \arg\min_{\pi} V_0^{\pi}(\mathbf{x}_0) \tag{3}$$

where,

- Control Policy(π): a function mapping from state space X to feasible control input u. An optimal policy pi^* is the one that corresponds to lowest cost.
- Value Function(V_t): expected long-term cost starting in state x at time t and following policy pi

If a control policy is sub optimal for the subproblem, then ere would exist a policy yielding a lower cost on at-least some portion of the state space. This algorithm is able to handle non-convex and non-linear problems.

In this algorithm, the value function at the goal states are set to zero or negative cost initially. Then iterating along the planning horizon, Q_t is calculated as shown in the eq(4) below. Since we are dealing with deterministic problem, we need not compute the expectation of the value function.

Then we find the minimum value function at the particular timestep and find the argument which minimises the value function as shown in eqn(5) which completes our objective of DPA algorithm as specified in eqn(3).

$$Q_t(\mathbf{x}, \mathbf{u}) = \ell(\mathbf{x}, \mathbf{u}) + \gamma \mathbb{E}_{\mathbf{x}' \sim p_f(\cdot \mid \mathbf{x}, \mathbf{u})}\left[V_{t+1}(\mathbf{x}')\right] \tag{4}$$

$$V_t(\mathbf{x}) = \min_{\mathbf{u}\in\mathcal{U}(\mathbf{x})} Q_t(\mathbf{x},\mathbf{u})$$
$$\pi_t(\mathbf{x}) = \arg\min_{\mathbf{u}\in\mathcal{U}(\mathbf{x})} Q_t(\mathbf{x},\mathbf{u}) \quad (5)$$

The algorithm terminates when the value function for two consecutive time steps are equivalent.

B. Project Implementation

In this project, we are required to design and implement the dynamic programming algorithm for the door-key environment. We need to construct each element of Markov Decision Process such as motion model, state space, cost matrix. By taking the information regarding the size of the map, possible orientation, key status and door status, we calculate the total number of possible states for a given environment.

For example, a 6x6 grid would contain 432 possible states. Each state is described by a dictionary containing 'position', 'orientation' and 'keydoor' as the keys. It is stored as a list of dictionaries. The states which is of wall type are filtered out and not included in the state space to save computational time of the DPA algorithm. For the part B, we include two more states called probable goal position and probable key position in our state space.

The motion model takes two inputs, previous state x_t and control action u_t and outputs the next state x_{t+1}. Since we don't have a well defined function to parameterise like differential drive model or bicycle model, the motion model was constructed based on heuristics. Our control space is composed of 5 possible action: MF, TL, TR, PK and UD. Thus, a particular state x_t has a possibility of 5 next states based on the control action. The following heuristics were used to construct the motion model:

- If the action is MF and the cell in front is a wall/locked door, then the next state will be same as the previous state.
- If the action is MF and the cell in front is free/unoccupied, then the next state is basically vector summation of previous state position and its orientation.
- If the action is TR/TL, then the agent's orientation is changed is irrespective of any other conditions.
- If the action is UD/PK and the front cell is door or key, then the status of keydoor will change or else it will remain the same.

In part B, we construct the motion model such that the door status and key status are checked in the multiple location based on same heuristics explained above. Using the motion model, we construct the cost matrix $c_{i,j}$ of size NxN where N = length of the state space. Each element of the cost matrix corresponds to cost of travelling from state i to state j. All the diagonal elements of the cost matrix is set to zero since cost of transitioning to itself is nil cost. Cost of possible states based on heuristics of the motion model is equal to one and everything else is set to infinity cost.

Now that we have constructed all the elements of MDP, we can go ahead with implementing the dynamic algorithm. The value function is a NxT matrix where N is the length of state space and T is the planning horizon and set to infinity except at the goal index which corresponds to terminal stage cost. Furthermore, the eqn(4) and eqn(5) are computed to derive the optimal policy and using the drawgif function provided in the utilis.py to visualise the results of the action sequence.

IV. Results

The optimal policy for each environment is listed below:

- Environment name: doorkey-6x6-direct: The value and Policy function graph is shown in fig 3 and 4. The control sequence output: $TR > TR > MF > MF$.
- Environment name: doorkey-6x6-normal: The value and Policy function graph is shown in fig 7 and 8. The control sequence output: $MF > TR > PK > TR > MF > TL > MF > MF > MF > TR > UD > MF > MF > TR > MF > MF > MF$
- Environment name: doorkey-8x8-direct: The value and Policy function graph is shown in fig 5 and 6. The control sequence output: $TL > MF > MF > MF$
- Environment name: doorkey-8x8-normal: The value and Policy function graph is shown in fig 9 and 10. The control sequence output: $TL > MF > TR > MF > MF > TR > MF > TL > PK > TL > MF > TL > MF > MF > MF > TR > UD > MF > MF > MF > TR > MF > MF > MF > MF > MF$

The algorithm performs fairly well for 6x6 grid but takes at least 10 seconds for 8x8 grids. One of the major reasons for computational inefficiency is high number of decision trees in constructing the motion model and cost matrix. Algorithm performed well for Part B as well since number of states are drastically reduced by using the given information about the wall.

GIF visualization of few environments can found in the following google drive link: click here

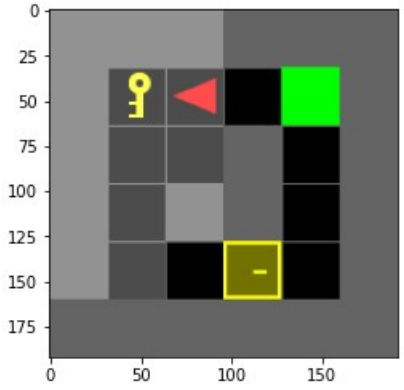

Fig. 3. Doorkey 6x6 direct: Mini-grid Environment

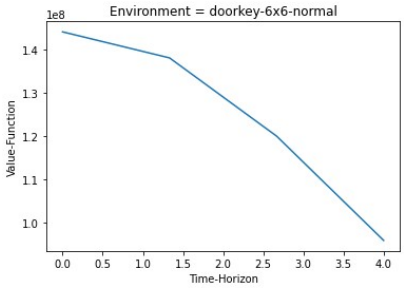

Fig. 4. Doorkey 6x6 direct: Value Function over Time

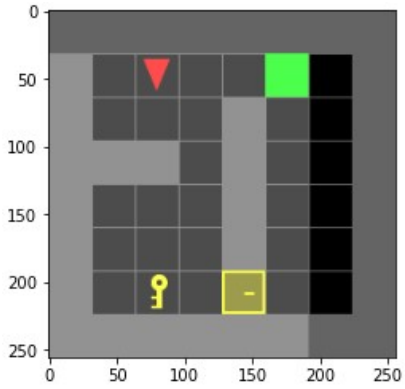

Fig. 5. Doorkey 8x8 direct: Mini-grid Environment

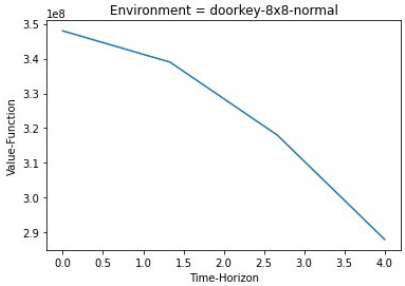

Fig. 6. Doorkey 8x8 direct: Value Function over Time

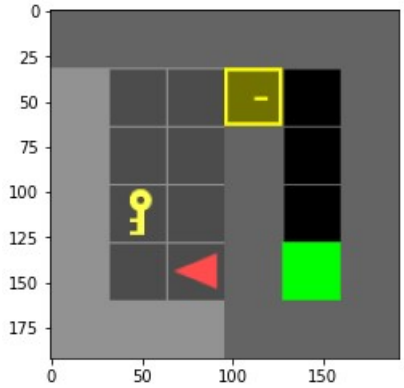

Fig. 7. Doorkey 6x6 normal: Mini-grid Environment

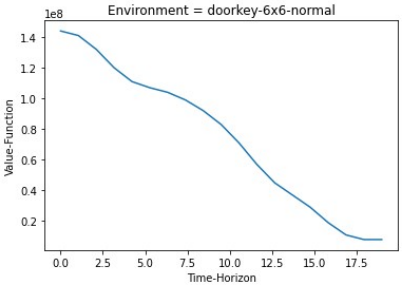

Fig. 8. Doorkey 6x6 normal: Value Function over Time

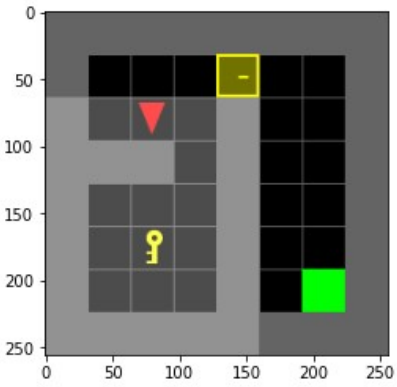

Fig. 9. Doorkey 8x8 normal: Mini-grid Environment

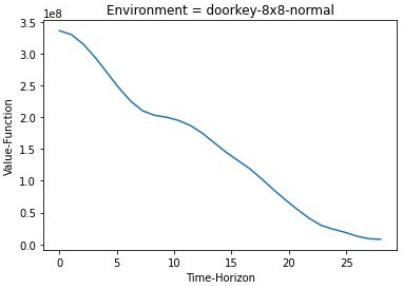

Fig. 10. Doorkey 8x8 normal: Value Function over Time