Planning: Part 2
Probabilistic Planning

Computer Science 4766/6912

Department of Computer Science
Memorial University of Newfoundland

July 13, 2018

Introduction

In classical approaches to robot planning there is no uncertainty. The following assumptions were implicit in the classical paradigm:

- The robot's sensors reveal the structure of the world directly
- The robot's pose and the goal pose are known
- The results of robot actions can be predicted

Yet it turns out that there is uncertainty in sensor values, uncertainty in pose estimates, and uncertainty in robot actions!

Consider a classical planner taking the robot to the goal pictured below:

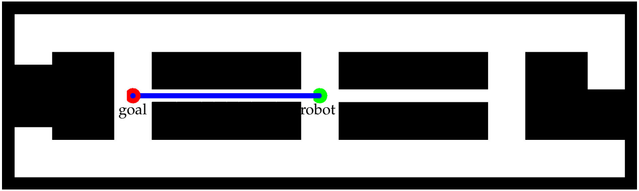
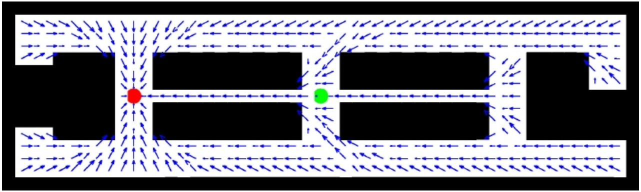


Figure 14.1 Near-symmetric environment with narrow and wide corridors. The robot starts at the center with unknown orientation. Its task is to move to the goal location on the left.



What is shown is a **control policy** which maps states into actions. Here it is assumed that these actions are *deterministic* (i.e. entirely predictable). However, movement down the central may be rather risky. It would be better if uncertainty in action could be taken into account.

Markov Decision Process

A Markov Decision Process (MDP) accounts for uncertainty in a robot's action. An MDP utilizes the Markov assumption, given before, that the state vector is complete and that knowing the current action allows us to completely determine the probability distribution of the next state.

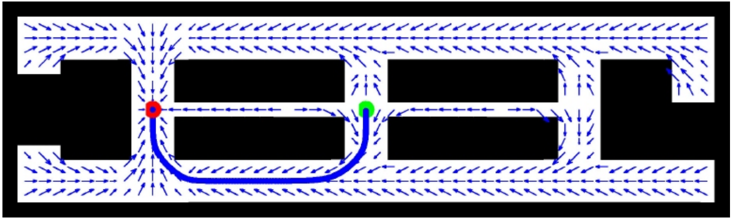
An MDP allows for uncertainty in action, but it does assume that the state is fully known. Uncertainty in state can be accounted for in a *Partially Observable Markov Decision Process* (POMDP).

POMDP's are important, but we will just cover MDP's.

We will be using the MDP framework and will therefore assume that the robot's state (pose) is known. This is reasonable if we assume that the robot has already localized itself using one of the previously studied localization techniques.

Since the robot's actions are uncertain, it is helpful to have a complete plan that covers the whole state space in case the robot wanders off the ideal route. A **control policy** gives the right action to perform in any state. A control policy is also known as a **universal plan**.

If we incorporate uncertainty in actions then this modifies the control policy:



Notice that the longer path is now preferred since it reduces the risk of running into a wall.

Value Iteration

We require an algorithm to evaluate the value of all possible states so that we can choose the best next state from any current state (i.e. the best movement). The algorithm used here is known as **value iteration**.

We first need to define a *payoff function* to get the immediate expected reward or penalty for any movement u in state x. e.g.

$$r(x, u) = \begin{cases} +100 & \text{if we expect } u \text{ to bring us to the goal state} \\ -1 & \text{otherwise} \end{cases}$$

We get a reward of 100 for reaching the goal state and a movement penalty of -1. We attempt to maximize the expected future payoff, which will hopefully lead us to the goal state while minimizing future penalties.

How do we measure the value of future payoffs? Define a **planning horizon** of T time steps and maximize payoffs within this time frame.

- $T = 1$: The robot is very short-sighted and attempts only to maximize the immediate payoff. This is known as the *greedy case*.
- $T > 1$: The robot attempts to maximize payoff over the finite planning horizon T. However, determining the best value for T may be difficult.
- $T = \infty$: Clearly we cannot make plans infinitely far in advance. Instead we define a **discount factor** $\gamma < 1$ which is used to discount future payoffs. Future payoffs are discounted by γ^τ where τ goes from $1 \to \infty$. In practice we only plan ahead up to the point where γ^τ becomes negligible.

The value iteration algorithm uses $T = \infty$ with future payoffs discounted by γ^τ.

Our fundamental goal is to define a control policy which is a mapping from states to actions:

$$\pi : x \rightarrow u$$

We wish to determine the policy π that maximizes future cumulative payoff. Consider first the optimal policy for $T = 1$:

$$\pi_1(x) = \underset{u}{\operatorname{argmax}}\, r(x, u)$$

($\underset{a}{\operatorname{argmax}}\, f(a)$ is the value for a at which $f(a)$ attains its maximum value. It could be a set if there are ties for the maximum.)

We define a value function $V_T(x)$ which gives the expected payoff for the current policy, discounted by γ. For $T = 1$ it is,

$$V_1(x) = \gamma \max_u r(x, u)$$

($\max_a f(a)$ is the maximum value for $f(a)$.)

We continue for larger planning horizons. For $T = 2$ we consider the best next move and then incorporate the expected value of all the places we might end up (x') after making the move u.

$$\pi_2(x) = \underset{u}{\text{argmax}} \left[r(x, u) + \int V_1(x')p(x'|u, x)dx' \right]$$

Remember that $r(x, u)$ gives the expected payoff for u, but the results of our actions our uncertain, which is why we must incorporate the expected value for all possible next states x'. The value function for $T = 2$ is,

$$V_2(x) = \gamma \max_u \left[r(x, u) + \int V_1(x')p(x'|u, x)dx' \right]$$

The discount factor γ is applied to the immediate reward $r(x, u)$ as before, as well as being applied within the definition of $V_1(x)$.

We can infer the general form for the optimal policy and associated value function for any T,

$$\begin{aligned}
\pi_T(x) &= \underset{u}{\operatorname{argmax}} \left[r(x,u) + \int V_{T-1}(x') p(x'|u,x) dx' \right] \\
V_T(x) &= \gamma \max_u \left[r(x,u) + \int V_{T-1}(x') p(x'|u,x) dx' \right]
\end{aligned}$$

Since $V_T(x)$ is defined recursively, you can see that the discount factor will be applied T times. γ is usually set to a value just less than one (e.g. 0.99). So γ^T will become vanishingly small as T increases. Thus, we expect $V_T(x)$ to converge to some finite value as $T \to \infty$. Hence,

$$V_\infty(x) = \gamma \max_u \left[r(x,u) + \int V_\infty(x') p(x'|u,x) dx' \right]$$

This expression leads to a concrete algorithm which progressively refines the value function for increasing values of T. Assume that the value function is stored in a discrete grid. $V(x_i)$ represents the value function for discrete state x_i.

Algorithm (VERSION 1)

discreteValueIteration()

```
for i = 1 to N
        $V(x_i) = r_{min}$
end for
repeat until convergence
        for i = 1 to N
                $V(x_i) = \gamma \max_u \left[ r(x_i, u) + \sum_{j=1}^{N} V(x_j) p(x_j | u, x_i) \right]$
        end for
end repeat
```

controlPolicy(x_i, V)

```
return $\underset{u}{\operatorname{argmax}} \left[ r(x_i, u) + \sum_{j=1}^{N} V(x_j) p(x_j | u, x_i) \right]$
```

There is an issue with this version (VERSION 1) of value iteration. Consider the value update equation:

$$V(x_i) = \gamma \max_u \left[r(x_i, u) + \sum_{j=1}^{N} V(x_j) p(x_j | u, x_i) \right]$$

Here, rewards come from executing actions in particular states—not from reaching states (e.g. the goal state). If it is arrival in a state that brings a reward, the following formulation is better:

$$V(x_i) = \gamma \max_u \left[\sum_{j=1}^{N} p(x_j | u, x_i) \left(r(x_j, u) + V(x_j) \right) \right]$$

Using this, and the similarly modified policy rule, gives us VERSION 2 of the algorithm...

Algorithm (VERSION 2)

discreteValueIteration()

```
for i = 1 to N
        $V(x_i) = r_{min}$
end for
repeat until convergence
        for i = 1 to N
                $V(x_i) = \gamma \max_u \left[\sum_{j=1}^{N} p(x_j|u, x_i)\left(r(x_j, u) + V(x_j)\right)\right]$
        end for
end repeat
```

controlPolicy(x_i, V)

```
return $\underset{u}{\text{argmax}} \left[\sum_{j=1}^{N} p(x_j|u, x_i)\left(r(x_j, u) + V(x_j)\right)\right]$
```

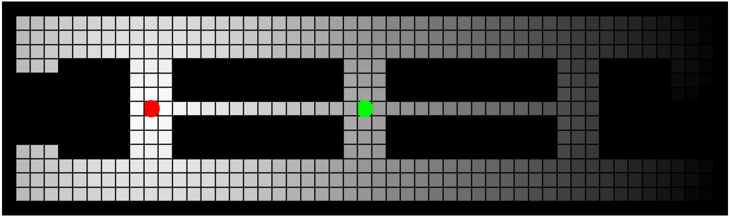


Figure 14.4 An example of an infinite-horizon value function T_∞, assuming that the goal state is an "absorbing state." This value function induces the policy shown in Figure 14.2a.

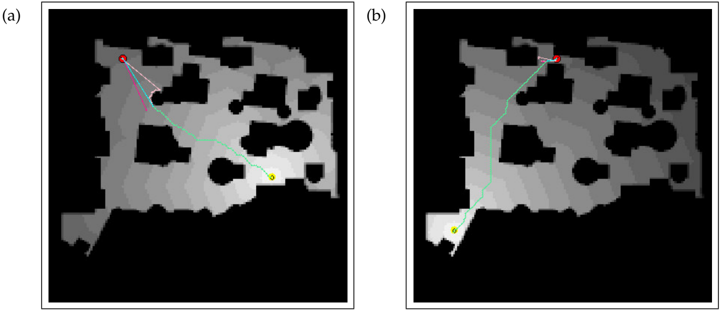


Figure 14.5 Example of value iteration over state spaces in robot motion. Obstacles are shown in black. The value function is indicated by the gray shaded area. Greedy action selection with respect to the value function lead to optimal control, assuming that the robot's pose is observable. Also shown in the diagrams are example paths obtained by following the greedy policy.

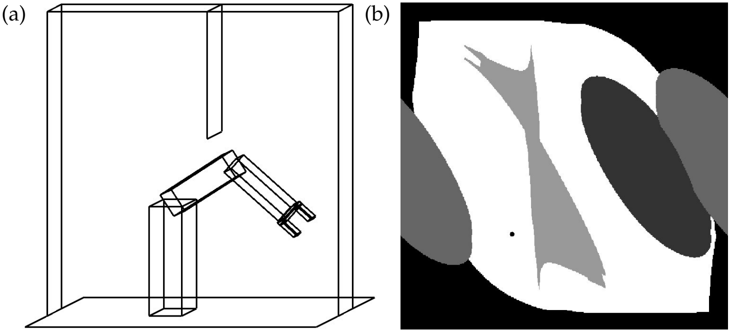


Figure 14.6 (a) 2-DOF robot arm in an environment with obstacles. (b) The *configuration space* of this arm: the horizontal axis corresponds to the shoulder joint, and the vertical axis to its elbow joint configuration. Obstacles are shown in gray. The small dot in this diagram corresponds to the configuration on the left.

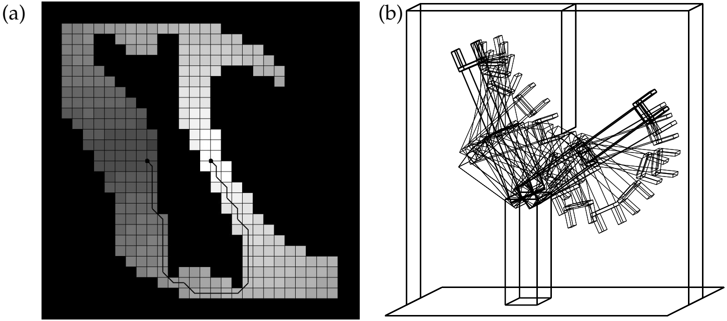


Figure 14.7 (a) Value iteration applied to a coarse discretization of the configuration space. (b) Path in workspace coordinates. The robot indeed avoids the vertical obstacle.