Summary Notes on Stochastical Optimal Control

Caden Lee

November 2023

Introduction

Stochastic Optimal Control vs Reinforcement Learning

There is a subtle difference in Stochastic Optimal Control and Reinforcement Learning, particularly in the problem each seeks to solve, the terminology, and the mathematical system.

	Stochastic Optimal Control	Reinforcement Learning
Problem	Cost Minimization	Reward Maximization
System Terminology	- Decision Maker/ Controller - Decision/ Control -Dynamic System	- Agent - Action - Environment
Methods Terminology	- Solving problem using simulation - model-based vs model-free simulation	- Learning - Planning vs Learning Distinction
Notational Systems	Transition Probability $\mathbb{P}(s, a, s')$	Discrete-time system equation $X_{k+1} = f(x_k, u_k, w_k)$

On Dynamic Programming

Dynamic Programming is a generic term that describes an approach that involves breaking down complex problems into simpler overlapping subproblems to be solved.

It may be referred to as an algorithm in the context of computer science. In the context of reinforcement learning and stochastic optimal control, we describe any optimization problem that can be solved by dynamic programming as a *"Dynamic Programming Problem"*, or *"Dynamic Programming"* in short. It is important to note that any mention of dynamic programming here thus refers to the *optimization problem* and not the algorithm per se. This is an important disclaimer to make to avoid confusion with dynamic programming algorithm problems.

1 Deterministic Dynamic Programming

1.1 Finite Horizon Problem

In this section, we consider a finite horizon problem, i.e. there exists an endpoint to our problem. A finite horizon problem can be described in the following system in Eq. 1.1.

$$x_{k+1} = f_k(x_k, u_k), \quad k \in \{0, 1, ..., N-1\} \tag{1.1}$$

where:

k is the time index

x_k is the state of the system at time k

u_k is the control or decision variable, $u_k \in U_k(x_k)$ is from a set of decision variables (*control space*) that is dependent on the state at time k

f_k is a function of (x_k,$u =_k$) that describes how the state is updated from k to $k+1$

N is the horizon, i.e. the number of times control is applied / decision is being made

Figure 1 illustrates how the control space $U_k(x_k)$ may vary depending on the state in which the control is applied. A discrete optimization problem can be represented in a graph, where each node represents a state, and the connecting lines (arcs) represent the controls. Collectively, the node and the arc represent a state-control pair. The *state space* refers to the set of possible states $\{x_{k+1}, x'_{k+1}, x"_{k+1}\}$ that one could branch into.

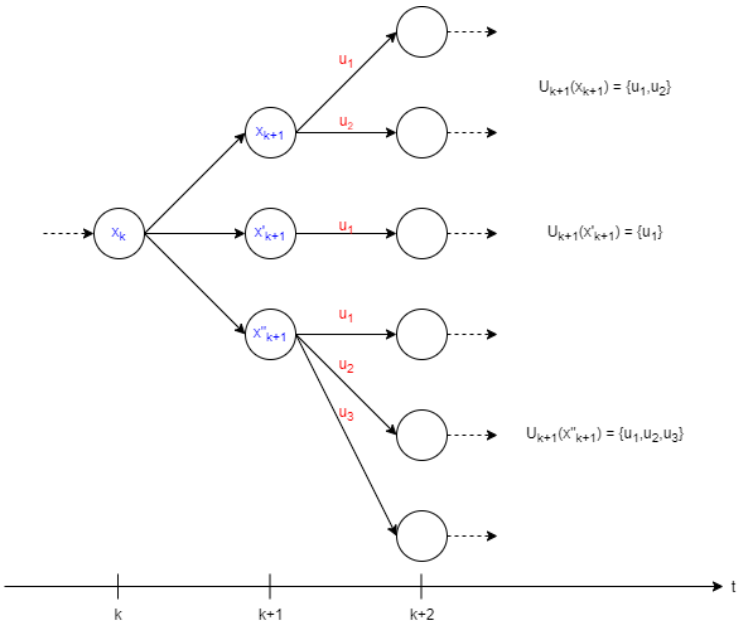

Figure 1: This graph shows the set of control variables that exists for each possible state at $k+1$ that leads into the next time step $k+2$.

When one progresses from one state to another, i.e. from x_k to x_{k+1}, one would incur a (incremental) cost from making that decision. This cost, denoted as $g_k(x_k, u_k)$, is a function of the current state x_k and the decision made u_k.
Spanning across the entire horizon, one may identify many possible sequences of decisions. This is also known as a *control sequence*. Each control sequence connects the initial state x_0 to the N-th stage at state x_N, as shown in Fig. 2.

One is to note that in the optimization problem, the state x_N is not the endpoint, but rather the last stage one arrives at after making the N-th decision u_N. In this optimization problem, all initial states will arrive at the same endpoint. So we create an artificial terminal node to complete the graph. One is to note that no decision is required to be made since there is only one route connecting state x_N to the terminal node, the incremental cost (or here we call it the terminal cost) is simply a function of the terminal state x_N which it came from, i.e. $g_N(x_N)$.

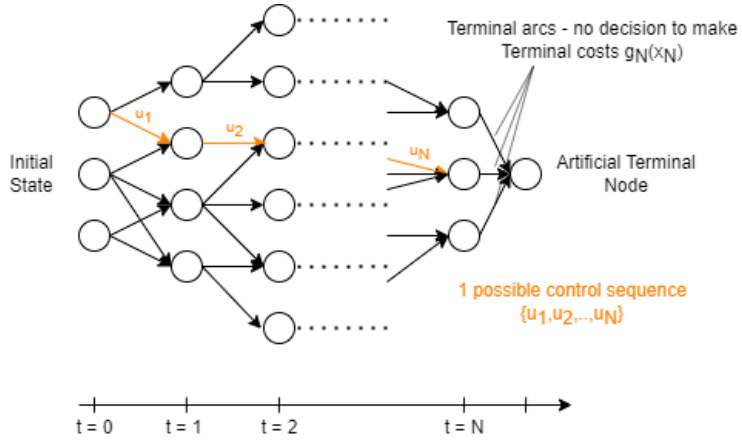

Figure 2: The orange line shows a possible control sequence connecting an initial state to the end state.

The cost incurred over a control sequence from an initial state is denoted in Eq. 1.2.

$$J(x_0; u_1, u_2, ..., u_N) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k) \tag{1.2}$$

More generally, the cost incurred over a control sequence from state x_m can be represented as Eq. 1.3.

$$Jm(x_m; u_{m+1}, ..., u_N) = g_N(x_N) + \sum_{k=m}^{N-1} g_k(x_k, u_k) \tag{1.3}$$

Our objective function (Eq. 1.4) of the dynamic programming problem is to find the optimal control sequence that minimizes the total cost incurred over all sequences.

$$J^*(x_0) = \min_{\substack{u_k \in U_k(x_k) \\ k \in \{0,...,N-1\}}} J(x_0; u_1, u_2, ..., u_N) \tag{1.4}$$

1.2 Principle of Optimality

> Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. (See Bellman, 1957, Chap. III.3.)

Let $\{u_0^*, ..., u_{N-1}^*\}$ be an optimal control sequence, which together with x_0 determines the corresponding state sequence $\{x_1^*, ..., x_N^*\}$ given Eq. 1.1. The optimal control sequence of a tail subproblem, defined at a start point x_k^* (which is a state within the optimal state sequence) which we seek to minimize the "cost-to-go" from time k to time N, is the truncated optimal control sequence from time k, as described in Eq. 1.5.

$$\begin{aligned} J_k^*(x_k^*) &= J_k(x_k^*; u_k^*, ..., u_{N-1}^*) \\ &= g_k(x_k^*, u_k) + \min_{\substack{u_m \in U_m(x_m) \\ m \in \{m+1,...,N-1\}}} \sum_{m=k+1}^{N-1} g_m(x_m, u_m) + g_N(x_N) \end{aligned} \tag{1.5}$$

Figure 3 illustrates the relationship between the full optimal control sequence and the optimal solution of the sub tailproblem.

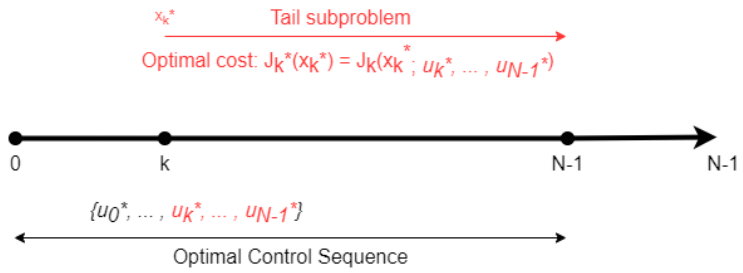

Figure 3: An illustration of the principle of optimality. The tail of the optimal control sequence is the optimal control sequence to the tail subproblem.

The principle of optimality is an important property as it suggests that the optimization problem can be solved piecewise through backward propagation

1. Compute the optimal cost function for the "tail subproblem" involving the last stage
2. Backpropagate to the preceding stage to solve for the optimal cost function for the new "tail subproblem" involving the last two stages
3. continue backpropagating iteratively until one solves for the optimal cost function involving across the entire time period of analysis

1.3 DP Algorithm for Deterministic Finite Horizon Problem

The "Offline Play"

Start with the terminal stage N, we get the optimal cost

$$J_N^*(x_N) = g_N(x_N), \quad \text{for all } x_N$$

Iterate backwards $k = N-1, N-2, ..., 0$, solve for

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} [g_k(x_k, u_k) + J_{k+1}^*(f_k(x_k, u_k))], \quad \text{for all } x_k$$

Figure 4 illustrates the algorithm in the shaded box above pictorially. Starting from the endpoint, we regress towards the preceding state and calculate the "cost-to-go" from that preceding state to the forward states. We assign the least "cost-to-go" as the optimal cost at state k. We continue the same step by regressing further backward. By the time we arrive at the initial state, we will derive the most optimal cost incurred from the initial stage to the terminal stage.

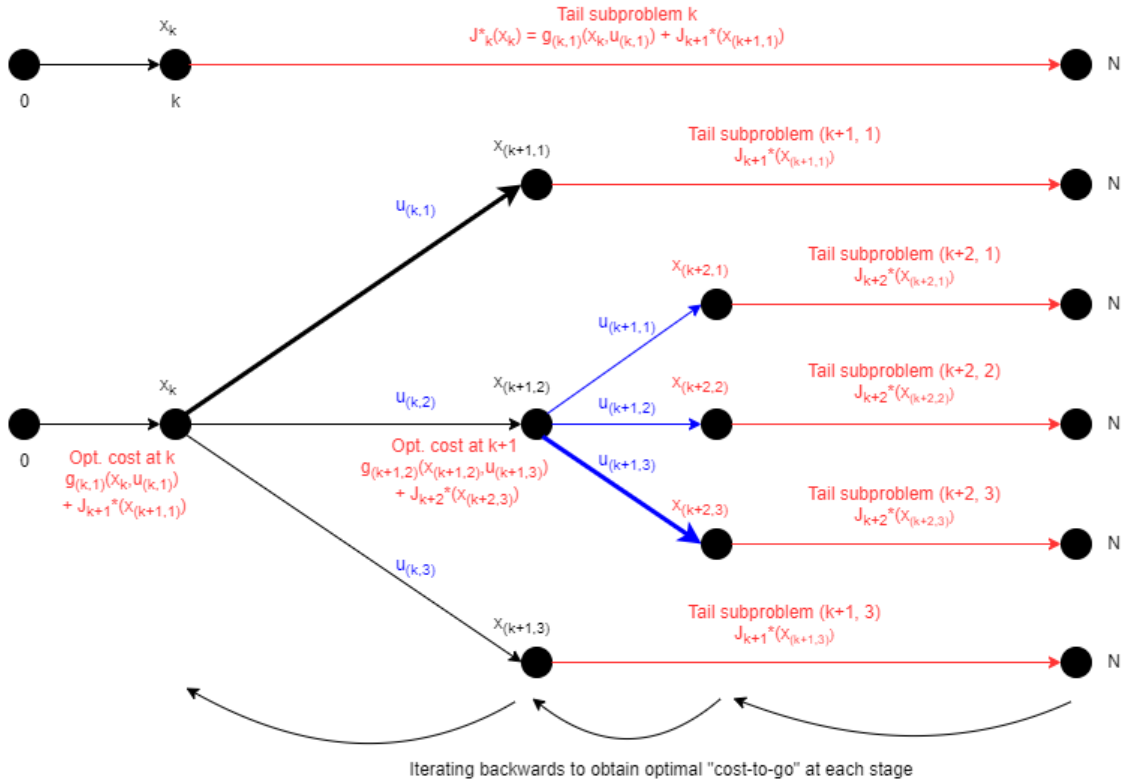

Figure 4: This example reveals the DP algorithm with two consecutive iterations.

1.4 Construction of Optimal Control Sequence

At this point, we managed to derive the optimal cost but we also need to determine the optimal control sequence, the sequence of decisions to be made to achieve the optimal cost. This comes the second part of the problem-solving, where we will - with the optimal costs derived at each stage - forward propagate through the path of "least cost". Remember, the optimal cost tagged to the initial node is the total optimal "cost-to-go" across the entire sequence. So in the forward propagation, one needs to travel *the path of reducing cost.*

The "Online Play"

To determine the optimal control sequence $\{u_0^*, ..., u_{N-1}^*\}$

Set

$$u_0^* \in arg \min_{u_0 \in U_0(x_0)} \Big[g_0(x_0, u_0) + J_1^*(f_0(x_0, u_0)) \Big]$$

and

$$x_1^* = f_0(x_0, u_0^*)$$

Sequentially, moving forward $k = 1, 2, ..., N-1$, set

$$u_k^* \in arg \min_{u_k \in U_k(x_k^*)} \Big[g_k(x_k^*, u_k) + J_{k+1}^*(f_k(x_k^*, u_k)) \Big]$$

and

$$x_{k+1}^* = f_k(x_k^*, u_k^*)$$

Note: *arg min* denotes finding the smallest value given constraints.

1.5 Q-Factors

Bridging the concepts between stochastic optimal control and reinforcement learning is the use of the optimal cost-to-go function J_k^*. A class of reinforcement learning - the *Q-learning* - uses the Q-factors, which is expressed as Eq. 1.6.

$$Q_k^*(x_k, u_k) = g_k(x_k, u_k) + J_{k+1}^*(f_k(x_k, u_k)) \tag{1.6}$$

Essentially, a Q-factor is the *pre-optimized cost-to-go function at state* k. The optimal cost-to-go $J_k^*(x_k)$ can be recovered by optimizing the Q-factor.

1.6 Solving with Value Space Approximation and Rollout

An issue with the forward optimal control sequence construction is that it is prohibitively time-consuming, as it requires all optimal cost-to-go to be computed by DP for all x_k and k. To reduce the computation time required, one may instead apply the forward algorithm on an approximated optimal cost-to-go $\tilde{J}_k$ (aka approximation in value space in Reinforcement Learning). In value space approximation, the suboptimal control sequence $\{\tilde{u}_0, ..., \tilde{u}_{N-1}\}$ is constructed instead. No backward propagation is required here.

One way to obtain the approximated values $\tilde{J}_k$ is through a *rollout* with a heuristic control scheme.

- Rollout: to achieve cost improvement from the base heuristic/ policy
- Heuristic: "trial-and-error"