IEOR 265 - Lecture 2
Deterministic Dynamic Programming

1 Deterministic Dynamic Programming

We will begin our study of DP algorithms by first analyzing the deterministic case. Recall from Lecture 1 that a DP problem has a dynamics function as one of it's key components:

$$x_{k+1} = f_k(x_k, \mu_k(x_k), w_k), \forall k \in \{0, ..., N-1\} \tag{1}$$

In the **deterministic** case, we assume that the disturbance w_k take a single value (usually the zero vector) with probability one. This has an important consequence for algorithms that attempt to solve this problem. In particular, we can write the DP as a "single-shot" optimization problem:

$$\begin{aligned} J_0^*(x_0) = \min_{u_0,...,u_{N-1}} & g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k) \\ & \text{s.t.: } x_{k+1} = f_k(x_k, u_k) \\ & x_k \in \mathcal{X}_k \\ & u_k \in U_k(x_k) \end{aligned} \tag{2}$$

Optimization algorithms (such as Interior Point Method, or Active Set Methods, for example) can be used to solve the above problem if the state space and action spaces are continuous. Usually such methods only provide local optimality guarantees, and require additional assumptions such as convexity to ensure global optimality. In this lecture we will not focus on such methods, and we will defer their analysis for later.

Instead we will focus on the case where such methods are not so appealing: On the discrete(finite-set) state/action spaces. In this case, we will see that the DP problem can be cast as a **Shortest Path** problem on graphs. Our goal on this lecture is to study the most important shortest path algorithms and how they are used in practical problems.

1.1 Shortest-Path Problems and DP

Consider a deterministic DP, where the state space S_k is a finite set of elements for each stage k. Then, each control u_k is associated with a transition from state x_k to state x_{k+1} via the dynamics $f_k(x_k, u_k)$ and the cost $g_k(x_k, u_k)$. Let's define a directed graph $G = (V, E)$, where V is the set of nodes, where we will have one node for each possible state and for each stage; and E is the set of

arcs, where we will have one arc for each possible transition between states (as defined via the dynamics $f_k(x_k, u_k)$); lastly we assign each arc with a "weight" of value equal to $g_k(x_k, u_k)$. In addition we define the node s as the source node associated with the initial state x_0 and the terminal node t whose nodes corresponding to all possible states x_N are linked via arcs with cost $g_N(x_N)$. This representation gives birth to two important observations:

1. (1) Obtaining a sequence of controls $(u_0, ..., u_{N-1})$ is equivalent to selecting a single *path* from the source s to the terminal t. Hence, the problem of finding the sequence of controls with the smallest cost is equivalent of finding the path with the smallest weight, a.k.a the *Shortest Path*.

2. (2) Since the DP is deterministic, we can always map the optimal policy $\pi^* = \{\mu_0^*, ..., \mu_{N-1}^*\}$ to a sequence $(u_0^*, ..., u_{N-1}^*)$, such that $u_i^* = \mu_i^*(x_i)$. Hence, the closed-loop optimal policy and the open-loop optimal control sequence are equivalent.

Observation (2) is what separates deterministic problems and stochastic problems and allows to solve the DP problem in usual backward fashion, but also in the forward fashion. W.l.o.g we start be letting the states be defined by integers $i \in S_k$ and we define $g_{i,j}^k$ as the cost of the transition from state i to state j at stage k. In addition, if there are no control u_k that brings state i to state j we define $g_{i,j}^k = \infty$. Lastly we let $h_{i,t}^N = g_N(i)$ be the cost from transition to state i to the terminal t at the last stage N. We illustrate the graph G in the following figure:

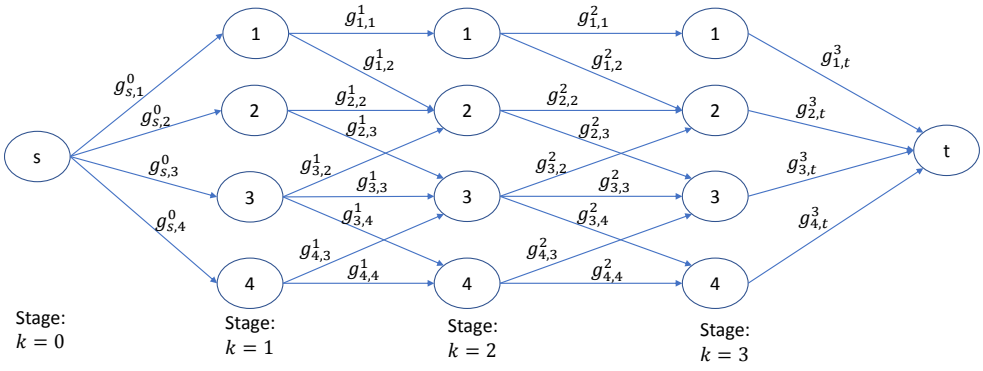

Figure 1: Schematic example of the graph G for the case with four states and $N = 3$.

Hence we can write the (backwards) DP equations:

$$J_N(i) = g_{i,t}^N, \forall i \in S_N \tag{3}$$

$$J_k(i) = \min_{j \in S_{k+1}} \left\{ g_{i,j}^k + J_{k+1}(j) \right\}, \forall i \in S_k, k \in \{0, ..., N-1\} \tag{4}$$

And the optimal cost $J_0(s)$ is both the optimal DP cost and the length of the shortest path from s to t. The forward DP algorithm follows from the somewhat

trivial observation that the shortest path from s to t is also the shortest path from t to s if we "reverse" the orientation of every arc in the graph G. Then we can write:

$$\tilde{J}_N(j) = g^0_{s,j}, \forall j \in S_1 \tag{5}$$

$$\tilde{J}_k(j) = \min_{i \in S_{N-k}} \left\{ g^{N-k}_{i,j} + \tilde{J}_{k+1}(i) \right\}, \forall j \in S_{N-k+1}, k \in \{1, ..., N-1\} \tag{6}$$

And the optimal cost is:

$$\tilde{J}_0(t) = \min_{i \in S_N} \left\{ g^N_{i,t} + \tilde{J}_1(i) \right\} \tag{7}$$

and it follows that: $J_0(s) = \tilde{J}_0(t)$.

The forward-version of the DP is important because in many real-time applications (for example in autonomous vehicles or in state estimation) the problem data for the stage k is only available when the system reaches stage k and not before. Hence doing the backward-version of DP is not possible. In addition, the shortest-path algorithms we will cover next are all based on the forward-version of DP.

2 The Viterbi Algorithm

We will discuss an important algorithm that is essential in may different fields, such as speech recognition, coding/decoding, graphical models, etc: **The Viterbi Algorithm**. This algorithm, and it's generalizations, are essentially deterministic DP algorithms, that is, they can be reduced to the problem of finding the shortest path in a particular directly acyclic graph (DAG).

2.1 Hidden Markov Models (HMM)

In the last lecture we saw the definition of Markov Decision Processes (MPD's). Here we will consider a Markov chain with a finite number of states $\{1, ..., p\}$ with the associate transition probabilities from each state i to state j:

$$p_{i,j} = p(x_{k+1} = j | x_k = i), \forall k = \{0, ..., N-1\} \tag{8}$$

Note that we are implicitly assuming that the probabilities do not change across stages and they are not impacted by any control/actions. Suppose that our goal is simply to observe this markov chain evolve with time. However we do not get access to the state values perfectly, instead we obtain a noisy observation z_k of the state x_k for every stage k. Given a sequence of imperfect observations $(z_0, ..., z_N)$, we would like to provide the "optimal" estimate (in some sense) for the underlying corresponding markov states $(\hat{x}_0, ..., \hat{x}_N)$. In addition, assume that we are given the probability q_i that the initial state of the Markov chain is $x_0 = i$. Lastly, suppose we are given the conditional probabilities $r(z|i,j) = r(z_{k+1} = z, x_k = i, x_{k+1} = j)$, that is the probability that we observe the imperfect measurement z given that the Markov chain transitioned from state i to j. Again, we made the simplifying assumption that the observation probabilities are time-invariant.

This type of model where the Markov chain transitions are imperfectly observed are called *Hidden Markov Models* or *Partially Observable Markov chains*.

By counterpart, if we allow control/actions to impact the probabilities, we would have a *Partially Observable Markov Decision Process* (POMDP's). We will study POMDP's later in the course.

For now let's focus on the HMM: Suppose we have obtained our imperfectly observed sequence $Z_N = (z_1, ..., z_N)$. Our optimality criterion will be to select a sequence $\{\hat{x}_0, ..., \hat{x}_N\}$ that maximizes the following probability:

$$\Pr((x_0, ..., x_N)|(z_1, ..., z_N)) = \Pr(X_N|Z_N) \tag{9}$$

Maximizing conditional probabilities is often challenging and we can leverage the Bayes rule:

$$\Pr(X_N|Z_N) = \frac{\Pr(X_N, Z_N)}{\Pr(Z_N)} \tag{10}$$

and the fact that since Z_N is a known vector for us, $\Pr(Z_N)$ is a known positive constant, to maximize the joint probability $\Pr(X_N, Z_N)$ instead. So we can write by conditioning:

$$\begin{gathered}\Pr(X_N, Z_N) = \Pr(x_0, ..., x_N, z_1, ..., z_N) = \\ q_{x_0} \Pr(x_1, ..., x_N, z_1, ..., z_N|x_0) = \\ q_{x_0} \Pr(x_1, z_1|x_0) \Pr(x_2, ..., x_N, z_1, ..., z_N|x_0, x_1, z_1) = \\ q_{x_0} p_{x_0,x_1} r(z_1|x_0, x_1) \Pr(x_2, ..., x_N, z_2, ..., z_N|x_0, x_1, z_1)\end{gathered} \tag{11}$$

Now we decompose the last term as follows:

$$\begin{gathered}\Pr(x_2, ..., x_N, z_2, ..., z_N|x_0, x_1, z_1) = \\ \Pr(x_2, z_2|x_0, x_1, z_1) \Pr(x_3, ..., x_N, z_3, ..., z_N|x_0, x_1, z_1, x_2, z_2) = \\ p_{x_1,x_2} r(z_2|x_1, x_2) \Pr(x_3, ..., x_N, z_3, ..., z_N|x_0, x_1, z_1, x_2, z_2)\end{gathered} \tag{12}$$

where we used the independence of observations, that is: $\Pr(z_2|x_0, x_1, x_2, z_1) = r(z_2|x_1, x_2)$. Combining Eq.11 with Eq.12 gives us:

$$\Pr(x_N, z_N) = q_{x_0} p_{x_0,x_1} r(z_1|x_0, x_1) p_{x_1,x_2} r(z_2|x_1, x_2) \Pr(x_3, ..., x_N, z_3, ..., z_N|x_0, x_1, z_1, x_2, z_2) \tag{13}$$

Now, by using the same conditioning argument on the last term of Eq.13 until we reach stage N gives us:

$$\Pr(x_N, z_N) = q_{x_0} \prod_{k=1}^{N} p_{x_{k-1},x_k} r(z_k|x_{k-1}, x_k) \tag{14}$$

Note that maximizing the above expression is not easy, since we have a product. In order to obtain an additive objective function, we use the fact that maximizing Eq.14 is equivalent to maximizing it's logarithm. In addition, it will suit our purpose to instead of maximize Eq.14 we will equivalently minimize it's negative. Therefore, we can write the following optimization problem:

$$\begin{gathered}\min - \ln(q_{x0}) - \sum_{k=1}^{N} \ln(p_{x_{k-1},x_k} r(z_k|x_{k-1}, x_k)) \\ \text{over all possible sequences } \{x_0, x_1, ..., x_N\}\end{gathered} \tag{15}$$

And our estimate of the most likely sequence of transitions is the optimal solution of the above problem.

2.2 The Algorithm

Finally we are in a position to frame this statistical inference problem as a shortest path problem on graphs. We will construct the graph called the *Trellis Diagram* as follows: We we will create $(N+1)$ copies of every single state $\{1, ..., p\}$, for a total of $(N+1)p$ nodes. In addition we will add two dummy nodes (a source s and a terminal t). The nodes of the k'th copy correspond to the states x_{k-1} at time $k-1$. We will connect a node x_{k-1} to a node x_k if the transition probability p_{x_{k-1},x_k} is positive and we will attribute the arc length to be $-\ln(p_{x_{k-1},x_k} r(z_k|x_{k-1}, x_k))$. Next we connect the source s to every node of the first copy, and every arc from s to x_0 will have length $-\ln(q_{x_0})$. Lastly, we connect every node of the last $(N+1)$ copy to the terminal t and all arcs from x_N to t will have length equal to 0. We illustrate the trellis diagram in the following figure:

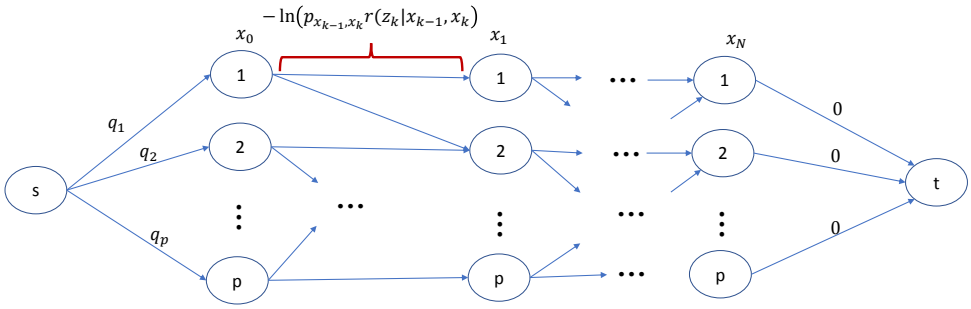

Figure 2: Schematic representation of the trellis diagram with the arc lengths highlighted: The arc lengths connecting the source s to nodes x_0 are given by the q_{x_0}; the intermediate arcs lengths are given by $-\ln(p_{x_{k-1},x_k} r(z_k|x_{k-1}, x_k))$, accordingly; lastly, the arcs from the last state x_N connected to the terminal t have zero length.

It is now clear that the shortest path from s to t will give us the optimal solution to the optimization problem in Eq.15. Moreover the solution is our desired most likely sequence $\{\hat{x}_0, ..., \hat{x}_N\}$.

The Viterbi Algorithm proceeds *forward* in time: It first compute the shortest distance from s to each node x_1, then to x_2, until we reach the terminal t. For instance, let $D_k(x_k)$ be the shortest distance from s to node x_k given the observation sequence $\{z_1, ..., z_k\}$. Then we can write the DP equations:

$$D_{k+1}(x_{k+1}) = \min_{\text{all } x_k \,:\, p_{x_k,x_{k+1}}>0} \left\{ D_k(x_k) - \ln(p_{x_k,x_{k+1}} r(z_{k+1}|x_k, x_{k+1})) \right\}$$

$$D_0(x_0) = -\ln(q_{x_0}) \tag{16}$$

And the optimal sequence $\{\hat{x}_0, ..., \hat{x}_N\}$ corresponds to the shortest path from s to t, thus minimizing $D_N(x_N)$ over the set of all possible states x_N. This Algorithm is attractive, since it allows real-time computations, that is: we do not need to wait to get the entire observation sequence Z_N, we can keep updating

the shortest distances D_k as we obtain new measurements z_k. In addition, the standard algorithm can be made more efficient, via a series of simple modifications, we refer to [cite] for possible extensions and improvements.

Next we illustrate this algorithm with a Convolutional Coding example.

2.3 Convolutional Coding Problem

The following example is taken from [1]. We will illustrate the Viterbi Algorithm with an application that is central to Information Theory and lies in the center in the fields of encoding/decoding techniques in Machine Learning. Suppose you would like to send a binary vector containing sensitive information to someone via a noisy communication channel. In order to protect your information you decide to use convolutions in order to codify your data vector. The main idea is to convert your data vector:

$$(w_1, w_2, ...), \ w_k \in \{0, 1\}, k \in \{1, 2, ...\} \tag{17}$$

into a coded sequence $(y_1, y_2, ...)$, where each y_k is an n-dimensional vector with binary coordinates, called a *codeword*:

$$y_k = \begin{bmatrix} y_k^1 \\ \vdots \\ y_k^n \end{bmatrix}, \ y_k^i \in \{0, 1\}, k \in \{1, 2, ...\} \tag{18}$$

The sequence $(y_1, y_2, ...)$ is transmitted over a noisy channel and gets transformed into a sequence $(z_1, z_2, ...)$, which is then decoded to yield the decoded data sequence $(\hat{w}_1, \hat{w}_2, ...)$. The objective is to design the encoder/decoder architecture so that the decoded sequence is as close to the original sequence as possible. Schematically we can draw the following figure:

Figure 3: Schematic representation of the encoder/decoder scheme.

One way of designing the encoder is to use convolutions. For instance, we use a linear-time invariant convolution of the following form:

$$y_k = Cx_{k-1} + dw_k, \ , k \in \{1, 2, ...\} \tag{19}$$

$$x_k = Ax_{k-1} + bw_k, \ , k \in \{1, 2, ...\}, \ x_0 : \text{ given} \tag{20}$$

where x_k is an m-dimensional vector with binary coordinates which is the state vector. In addition the matrices A and C together with the vectors d and b (of appropriate dimensions) also have entirely binary components. Note that since all the elements in Eq.19-20 are binary, one can use modulo 2 arithmetic to perform very fast computations. For instance suppose $m = 2$ and $n = 3$ and we have the following parameters:

$$C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \ A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, d = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{21}$$

Suppose that the initial state is $x_0 = 00$ and our data vector is:

$$(w_1, w_2, w_3, w_4) = (1, 0, 0, 1) \tag{22}$$

Then our generated states are:

$$(x_0, x_1, x_2, x_3, x_4) = (00, 01, 11, 10, 00) \tag{23}$$

and our codeword sequence is:

$$(y_1, y_2, y_3, y_4) = (111, 011, 111, 011) \tag{24}$$

This encoding is best illustrated with a figure:

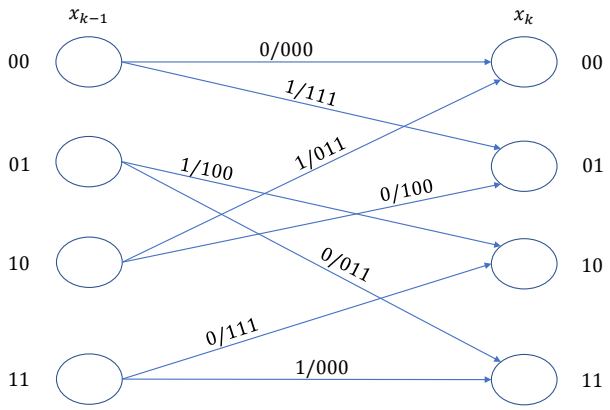

Figure 4: Schematic representation of state transition for the convolutional coding example: The binary number pair on each arc is the data/codeword pair w_l/y_k for the corresponding transition, So for example, when $x_k - 1 = 01$, a zero data bit ($w_k = 0$) effects a transition to $x_k = 11$ and generates the codeword 001.

Now, assuming that the a codeword y is actually received as z with some known probability $\Pr(z|y)$, where z is any n-bit binary number and each subsequent transmission is independent from the previous, then we can apply a similar argument used to obtain Eq.14 to write:

$$\Pr(Z_N|Y_N) = \prod_{k=1}^{N} \Pr(z_k|y_k) \tag{25}$$

where $Z_N = (z_1, ..., z_n)$ is the received sequence and $Y_N = (y_1, ..., y_N)$ is the transmitted sequence. As illustrated by figure 4 , each element y correspond to a state transition of the underlying convolution. Then our goal is to find the most likely sequence $\hat{Y}_N = (\hat{y}_1, ..., \hat{y}_N)$ that solves:

$$\Pr(Z_N|\hat{Y}_N) = \max_{Y_N} \left\{ \Pr(Z_N|Y_N) \right\} \tag{26}$$

Now we construct the trellis diagram in a similar way as before by concatenating N state transitions plus the dummy nodes s and t. In this case, however,

we let the arcs connecting s and t to the their respective nodes x_0 and x_N to have length equal to 0. Then, by applying the logarithm we have:

$$\Pr(Z_N|\hat{Y}_N) = \min\Big\{\sum_{k=1}^{N} -\ln(\Pr(z_k|y_k))\Big\}$$
$$\text{over all binary sequences } (y_1, ..., y_N) \tag{27}$$

And it follows directly, that this problem is equivalent of finding the shortest path in the trellis diagram where the transition associated with the codeword y_k is represented by an arc with length $-\ln(\Pr(z_k|y_k))$ connecting x_{k-1} and x_k. Hence the maximum likelihood estimate $\hat{Y}_N$ is given as the solution of the Viterbi Algorithm applied to this graph:

$$D_{k+1}(x_{k+1}) = \min_{\text{all } x_k \,:\, (x_x,\, x_{k+1}) \text{ is an arc}} \Big\{D_k(x_k) - \ln(\Pr(z_{k+1}|y_{k+1}))\Big\} \tag{28}$$

We conclude this example by observing that $\hat{y}_k$ is the estimated transition(codeword) which is then, itself, associated with a transition from x_{k-1} to x_k. With this information we can decode the estimated transmission into the estimated original data vector $\hat{w}_k$. Hence the estimated sequence $(\hat{w}_1, ..., \hat{w}_N)$ is taken as the decoded final data.

References

[1] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas, *Dynamic programming and optimal control*. Athena scientific Belmont, MA, 1995, vol. 1, no. 2.