SEEM 3470: Dynamic Optimization and Applications **2013–14 Second Term**

Handout 4: Deterministic Systems and the Shortest Path Problem

Instructor: Shiqian Ma January 27, 2014

Suggested Reading: Bertsekas' Lecture Slides on Dynamic Programming; Sections 2.1 and 2.2 of Chapter 2 of Bertsekas, *Dynamic Programming and Optimal Control: Volume I (3rd Edition)*, Athena Scientific, 2005.

1 Introduction

In this handout, we will focus on deterministic finite–state systems, i.e., systems in which the number of possible states in each time period is finite, and the parameter w_k in each time period k can only take on one value. Note that both the N–stage resource allocation problem and the operations scheduling problem are examples of deterministic systems. However, the former is not a finite–state system, while the latter is.

2 Finite–State Systems and Shortest Paths

2.1 Formulating a Deterministic Finite–State Problem as a Shortest Path Problem

Consider now a deterministic problem in which the number of possible states in each time period k is finite. Then, at any state S_k, a control x_k can be regarded as a transition from S_k to the state $\Gamma_k(S_k, x_k, w_k)$ at a cost $\Lambda_k(S_k, x_k, w_k)$. In particular, we can use a graph to represent such a system. Each node corresponds to a possible state of the system, and an arc (or a directed edge) corresponds to a transition between states at successive stages. Furthermore, each arc is associated with a cost. To take care of the final stage, an artificial terminal node t is added and each node corresponding to a final–stage state S_N are connected to t via an arc of cost $\Lambda_N(S_N)$. See Figure 1 for an illustration.

With the above setup, it is not hard to see that a control sequence $x_0, x_1, \ldots, x_{N-1}$ traces out a path originating at the initial state s and terminating at a node corresponding to the final stage in the transition diagram. Moreover, the cost associated with the control sequence is simply the sum of the costs on the arcs from s to t. Thus, if we view the costs on the arcs as distances, then the problem of finding a control sequence or policy that minimizes the cost function is equivalent to finding the shortest path from s to t in the transition diagram.

Formally, let $\mathcal{S}_k$ be the set of possible states in time period k. Let a_{ij}^k denote the cost of transition at stage k from state $i \in \mathcal{S}_k$ to state $j \in \mathcal{S}_{k+1}$, and let $a_{it}^N = \Lambda_N(i)$ be the terminal cost of state $i \in \mathcal{S}_N$. Here, we adopt the convention that if there is no transition from state $i \in \mathcal{S}_k$ to state $j \in \mathcal{S}_k$, then $a_{ij}^k = \infty$. Using these notations, the dynamic programming algorithm for the deterministic finite–state system takes the following form:

$$\begin{aligned} J_N(i) &= a_{it}^N && \text{for all } i \in \mathcal{S}_N, \\ J_k(i) &= \min_{j \in \mathcal{S}_{k+1}} \left\{ a_{ij}^k + J_{k+1}(j) \right\} && \text{for all } i \in \mathcal{S}_k,\ k = 0, 1, \ldots, N-1. \end{aligned} \tag{1}$$

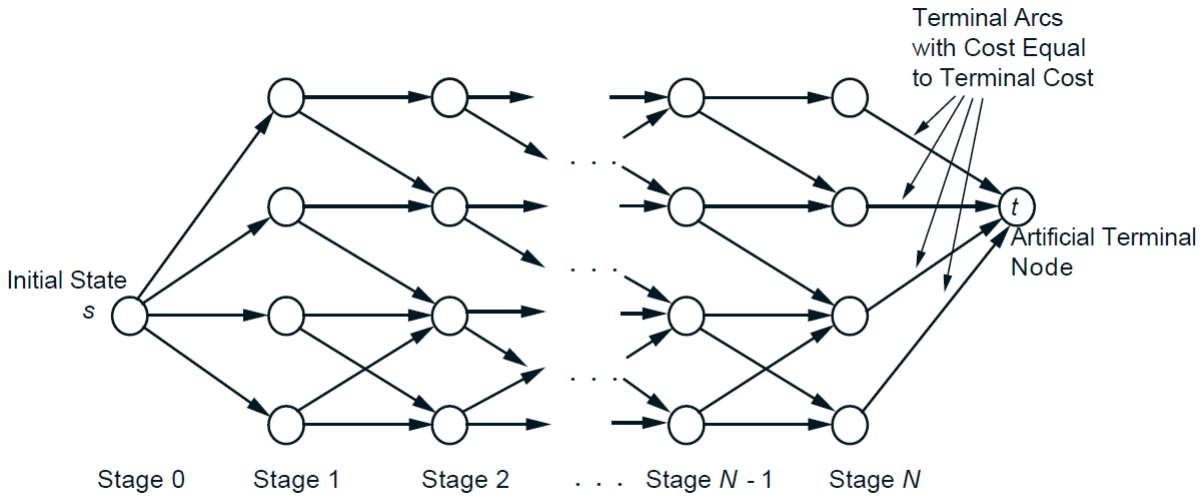

Figure 1: Transition Diagram of a Deterministic Finite–State System

The optimal cost is just $J_0(s)$ and is equal to the length of the shortest path from s to t.

Just as in the standard dynamic programming algorithm, the above algorithm proceeds backward in time. However, it is easy to convert it into an algorithm that proceeds forward in time. The crucial observation is that an optimal path from s to t is also an optimal path from t to s if we reverse all the directions of the arcs in the transition graph. Specifically, the algorithm for this reverse problem starts from the set of states $\mathcal{S}_1$ in stage 1, then proceeds to the set of states $\mathcal{S}_2$ in stage 2 and so on, until the states $\mathcal{S}_N$ in stage N are reached. Formally, the forward algorithm for the deterministic finite–state system is as follows:

$$\begin{aligned} \tilde{J}_1(i) &= a_{si}^0 && \text{for all } i \in \mathcal{S}_1, \\ \tilde{J}_k(i) &= \min_{j \in \mathcal{S}_{k-1}} \left\{ a_{ji}^{k-1} + \tilde{J}_{k-1}(j) \right\} && \text{for all } i \in \mathcal{S}_k,\ k = 2, \ldots, N. \end{aligned} \tag{2}$$

The optimal cost is then given by

$$\tilde{J}_{N+1}(t) = \min_{i \in \mathcal{S}_N} \left\{ a_{it}^N + \tilde{J}_N(i) \right\}.$$

Note that since the forward optimal path should coincide with the backward optimal path, we must have

$$J_0(s) = \tilde{J}_{N+1}(t).$$

To further understand the forward algorithm, recall that $J_k(i)$ is the optimal cost–to–go from state $i \in \mathcal{S}_k$ to state t. Hence, we may interpret $\tilde{J}_k(i)$ as the optimal cost–to–arrive to state $i \in \mathcal{S}_k$ from state s.

One of the advantages of the forward algorithm is that it does not require knowledge about the problem data in time periods $k+1, \ldots, N$ when making the decision for time period k. Later, we will see how this comes to play in applications.

2.2 Formulating a Shortest Path Problem as a Deterministic Finite–State Problem

In the last sub–section, we have seen that a deterministic finite–state problem can be formulated as a special type of shortest path problem, in which the graph has no cycles. As it turns out, a general shortest path problem can also be formulated as a deterministic finite–state problem. Consequently, one can apply the dynamic programming algorithm to solve the shortest path problem. To prove this result, let us introduce some preliminaries. Let $V = \{1, 2, \ldots, N, t\}$ be the set of nodes of a graph, and let a_{ij} be the cost of moving between nodes i and j. We assume that $a_{ij} = a_{ji}$, i.e., the cost of moving between nodes i and j does not depend on the direction. Moreover, we set $a_{ij} = \infty$ if one cannot move between nodes i and j directly. The node t is designated the destination. The goal of the problem is to find a shortest path from each node i to the node t.

In order for the problem to be well–defined, we need to assume that there is no negative cycles in the graph, i.e., there does not exist a sequence of nodes $j_1, \ldots, j_k$ such that

$$a_{j_1 j_2} + a_{j_2 j_3} + \cdots + a_{j_{k-1} j_k} + a_{j_k j_1} < 0.$$

Under this assumption, all cycles have non–negative costs, and it is clear that a shortest path need not take more than N moves. This motivates us to formulate the shortest path problem as an N–stage dynamic programming problem, where each stage corresponds to a move in the graph, and we allow degenerate moves of the form $i \to i$, whose associated cost is $a_{ii} = 0$. Now, let

$$J_k(i) = \text{optimal cost of getting from } i \text{ to } t \text{ in } N - k \text{ moves.}$$

Then, the optimal cost of the path from i to t is $J_0(i)$.

To apply the dynamic programming algorithm to this problem, we simply observe that

$$\begin{aligned} J_k(i) &= \min_{j \in \{1,\ldots,N\}} \{a_{ij} + J_{k+1}(j)\} \quad \text{for } i = 1, \ldots, N,\ k = 0, 1, \ldots, N-2 \\ J_{N-1}(i) &= a_{it} \qquad\qquad\qquad\qquad\quad \text{for } i = 1, \ldots, N. \end{aligned} \tag{3}$$

As an example, consider the following shortest path problem:

Here, we have $N = 4$ and node 5 is the destination node. From the dynamic programming equations (3), we compute

$$J_3(1) = 2,\ J_3(2) = 7,\ J_3(3) = 5,\ J_3(4) = 3,$$

$$J_2(1) = 2,\ J_2(2) = 5.5,\ J_2(3) = 4,\ J_2(4) = 3,$$

$$J_1(1) = 2,\ J_1(2) = 4.5,\ J_1(3) = 4,\ J_1(4) = 3,$$

$$J_0(1) = 2,\ J_0(2) = 4.5,\ J_0(3) = 4,\ J_0(4) = 3.$$

The optimal path from, e.g., node 3 to node 5 can then be read off by tracing the above computation:

$$\begin{aligned} J_0(3) &= a_{33} + J_1(3) \quad \text{node on path} = 3 \\ &= a_{33} + J_2(3) \quad \text{node on path} = 3 \\ &= a_{34} + J_3(4) \quad \text{node on path} = 4 \\ &= 1 + 3 \qquad\quad \text{node on path} = 5, \end{aligned}$$

i.e., the optimal path is $3 \to 4 \to 5$.

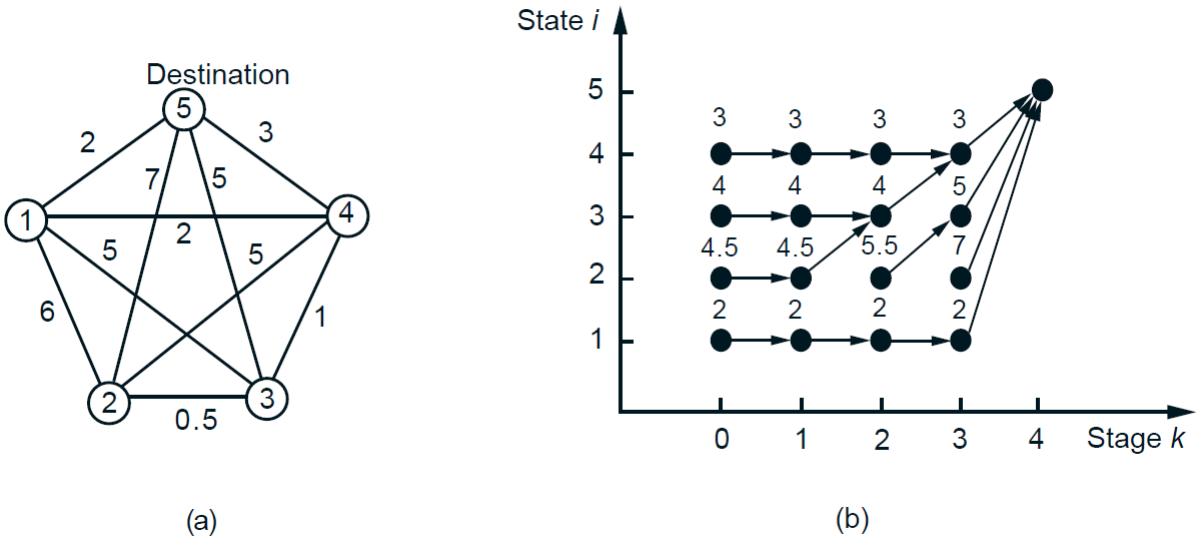

Figure 2: A Shortest Path Problem with $N = 4$ and $t = 5$

3 The Critical Path Analysis

Consider the problem of arranging a large project. There are many tasks to complete before the entire project can be completed. Each task has a duration to finish. Different tasks may have complicated precedence relationships. We may construct a graph where each arc represents a task, with the duration being the weight on the arc.

Also, there is a task symbolizing the start of the project; let the node be s. And similarly there is a task symbolizing the finish of the project; let the node be t. Suppose that (i, j) is a task (arc), and the duration of the task is t_{ij}. The *longest path* from s to t is called a *critical path*. All the jobs on critical paths are called *critical tasks*. The length of critical paths is the minimum total completion time of the project. Let us denote it to be C.

The *longest path* from s to i is the *earliest start time* when the task (i, j) must be started without affecting the completion of the entire project. Let this time be E_i. Similarly, the *longest path* from j to t is the *latest time* when the task must be finished without affecting the completion of the entire project. Let it be L_j. If $C = E_i + L_j + t_{ij}$, then the task (i, j) is *critical*. In general, $C - (E_i + L_j + t_{ij})$ is the *slack* time for the task (i, j). It is clear that computing E_i can be done by forward dynamic programming; and computing L_j can be done by backward dynamic programming.

4 Hidden Markov Models

In many applications, the actual state may not be exactly observable. Instead, one may receive some signals, suggesting the likelihood of the possible states. Let $X_N = \{x_0, x_1, ..., x_N\}$ be the true states undergone. As a transition takes place, a signal will be transmitted. Suppose that the observed signals are $Z_N = \{z_1, ..., z_N\}$. The question is: how can we determine the true states x_N from the observed signals z_N? Suppose that the probability of a transition from x_i to x_j given z_k is

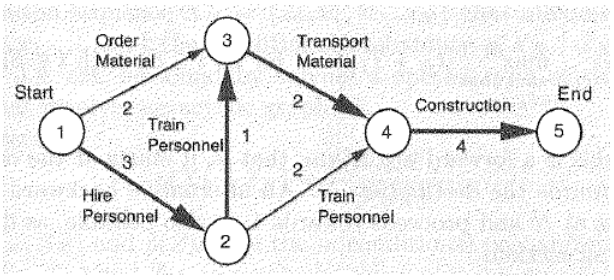

Figure 3: Critical Path Analysis

$r(z_k; x_i, x_j)$. Suppose the probability of the initial state is $P(x_0) = \pi_{x_0}$. So what is the maximum likelihood of X_N, given Z_N? Clearly,

$$P(X_N \mid Z_N) = \frac{P(X_N, Z_N)}{P(Z_N)}.$$

To maximize the likelihood, we need to find X_N maximizing $P(X_N, Z_N)$. In fact we can establish

$$P(X_N, Z_N) = \pi_{x_0} \prod_{k=1}^{N} p_{x_{k-1},x_k} r(z_k; x_{k-1}, x_k).$$

Now

$$\ln P(X_N, Z_N) = \ln \pi_{x_0} + \sum_{k=1}^{N} \left(\ln p_{x_{k-1},x_k} + \ln r(z_k; x_{k-1}, x_k) \right).$$

The most likely sequence of the hidden states can be found by forward dynamic programming (to find the longest path).

This approach is known as the *Viterbi* algorithm proposed by Andrew Viterbi in 1967.

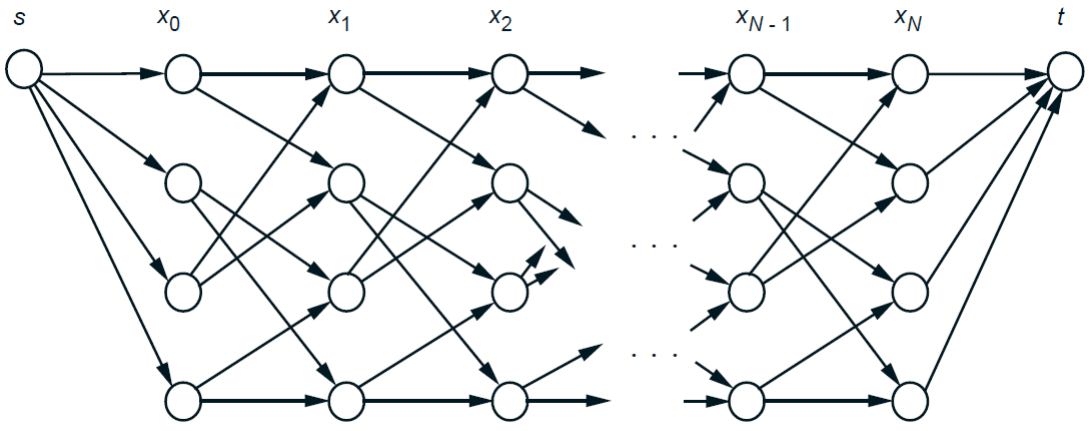

Figure 4: Hidden Markov Models