Dynamic Programming: An overview

These notes summarize some key properties of the Dynamic Programming principle to optimize a function or cost that depends on an interval or stages. This plays a key role in routing algorithms in networks where decisions are discrete (choosing a particular link for a route).

1 Preliminaries: The basic principle underlying dynamic programming

Dynamic programming is a sequential procedure for optimizing a given objective function called a reward or cost function depending on whether we want to maximize or minimize the function. The variable with respect to which we optimize is called a decision or control.Underlying the the problem set-up is the notion of dynamics or stages. By this we mean the problem is broken up into stages or time points and the aim is at every stage or time point to select the optimal decision or control so that the objective over the total number of stages or time points is optimized. By selecting a particular decision at a given time or stage we can affect the way the process (which is being optimized) evolves for the next interval. This process done over the entire interval of interest (or over the total number of stages) then yields a trajectory which the process follows. Thus in essence dynamic programming is a technique by which we select the optimal trajectory from amongst all possible trajectories such that the given objective function (which in general depends on the trajectory followed and the decisions taken) is optimized. In the sequel we will always think of the optimization being a minimizing procedure since we know that minimizing a function corresponds to the mazimization of the negative of the function. In other words rewards can be thought of as negative costs.

Before we go into details we begin by stating the so-called *Principle of Optimality* bue to Richard Bellman which is called Bellmans principle of optimality.

Proposition 1.1 *(***Bellman's principle of Optimality)**

From any point on an optimal trajectory, the remaining trajectory is optimal for the corrseponding problem over the remaining number of stages or time interval initiated at that point.

1.1 Deterministic dynamic programming : the finite horizon case

Let us now begin by specifying the mathematical structure of the problems we will be treating in this course.

We now begin with specifying some notation.

The control variable at every time (or stage) t will be denoted by u_t.

Let $\mathcal{A}_t$ denote the set of possible values the control can take at time t i.e. for every t the control $u_t \in \mathcal{A}_t$.

For example $\mathcal{A}_t$ could be the set of non-negative integers (in which case it is the same for all t, or $\mathcal{A}_t = \{u_t : 0 \leq u_t \leq k(t)\}$ where $k(t)$ is a positive number which changes its value depending on t. Also note that u_t can be vector valued in which case $\mathcal{A}_t$ will specify the values the vector can take etc.

Let $\mathcal{U}_t$ denote the sequence of controls over a horizon $[0, t]$ i.e.

$$\mathcal{U}_t = (u_0, u_1, \ldots, u_t)$$

This is just the set of controls taken on the interval $[0, t]$. It is often called a *policy*.

Assuming we start at the point where our process has a value x, let the cost incurred in adopting the sequence $\{u_k\}_{k=0}^{t}$ or equivalently to adopting the policy $\mathcal{U}_t$ be denoted by

$$J(x, \mathcal{U}_t) = J(x, u_0, u_1, \cdots, u_t)$$

The above is often denoted $J^u_{(0,t)}(x)$. In words, it is the cost incurred over the horizon $[0,t]$ when starting from x and using the controls $u_0, u_1, \cdots, u_t$.

Now suppose the horizon of interest is $[0,T]$ i.e. we wish to stop at time or stage T.

Let $J_(0,T)^*(x)$ denote the optimal cost $\inf_{u_\cdot \in \mathcal{U}_{T-1}} J^u_{(0,T)}(x)$ where we have used inf for min since the minimum may not be defined without some conditions on $\mathcal{A}_t$. Note here we are specifying the control actions upto $T-1$ since the initial value and the controls upto $T-1$ determine where the trajectory of the system will be at time T and we have no interest to proceed further (as we terminate the problem at T).

Now by the application of the *Principle of Optimality* we obtain:

$$J^*_{(t,T)}(x) = \inf_{u_t \in \mathcal{A}_t} J^*_{(t,T)}(x, u_t, \mathcal{U}^*_{(t+1,T-1)}) \tag{1.1}$$

where $\mathcal{U}^*_{(t,s)}$ denotes the optimal values of the controls chosen in the interval $[t,s]$.

Thus, in order to be able to solve for $J^*_{(0,T)}(x)$, we need to specify the terminal cost $J^*_{(T,T)}(x)$. Once we have this we can work backwards. Note in the equation above x is just a variable which specifies that we start from the point x at time t.

Without further assumptions on the structure of the cost function we cannot say much more about the behaviour of J^* and hence determine $u^*_\cdot$.

In the sequel we will assume the following structure for the problem:

We will assume that the objective function or costs have an **additive** structure. By this we mean that the overall cost (over the interval (called horizon)) of interest is the sum of individual costs incurred at each stage or time point. The individual costs are called **running costs** or **stage costs**.

Before doing so we need another concept before we can write down the the equation for the optimal costs in a nice compact manner.

First note that in the definition of $J^*_{(0,t)}$ we carry the initial value x and the vector U_{t-1}. This is because specifying the initial condition and the set of control values over a given interval we specify the value of the trajectory at the end of the interval. Thus as t increases the vector $(x, u_0, u_1, \cdots, u_t)$ increases in dimension and this is very inconvenient. We can reduce the vector to define our process or system by the introduction of the concept of a *state* of a system.

Definition 1.1 (The state of a system)

The state of a system is a quantity which encapsulates the past of the system. Specifically the state of a system at time (or stage) t, denoted by x_t, is such that knowing x_t and the set of inputs $(u_t, u_{t+1}, \cdots, u_T)$ allows us to determine x_{T+1} completely.

In our context it is equivalent to saying that knowing $x_0 = x$ and u_0 determines x_1 at time 1. Knowing x_1 and u_1 determines x_2 and so on.

This suggests the following equation for the evolution of the system:

$$x_{k+1} = a_k(x_k, u_k);\ k = 0, 1, ... \tag{1.2}$$

with initial state $x_0 = x$ given.

With the above definition of the state, we will now define a general form for the additive costs we will treat:

$$J^u_{(0,T)}(x) = \sum_{k=0}^{T-1} c_k(x_k, u_k) + k_T(x_T) \tag{1.3}$$

where the terms $c_k(x_k, u_k)$ denote the **running** or stage costs (which depend on the time k, the state x_k and the control or decision u_k) and $k_T(x_T)$ denotes the **terminal** cost.

Remark: We have written the running costs in terms of the state at time k and the control used at time k. In terms of our prior discussion, if we did not use the concept of the state then each term $c_k(x_k, u_k)$ would be of the form $c_k(x, \mathcal{U}_k)$ which is a considerable saving both in terms of interpretation as well as the fact that (see below) the optimization at each stage is over the current control variable at that stage , i.e. u_k if the stage is k, instead of doing it over the whole vector $\mathcal{U}_k$.

With such an additive cost structure we can now rewite the equation that the optimal costs must satisfy in a much more convenient form. The equation which defines this relationship is called the **optimality equation**.

Proposition 1.2 (Optimality Equation)
For a system whose state is governed by the equation $x_k = a_k(x_k, u_k)$ and with an additive cost function of the form above, the optimal cost on the interval $[t, T]$ starting in state $x_t = x$ satisfies

$$J^*_{(t,T)}(x) = \inf_{u \in \mathcal{A}_t} \left[c_t(x, u) + J^*_{(t+1,T)}(a_t(x, u)) \right]; \quad t = 0, 1, \ldots, T-1 \tag{1.4}$$

with

$$J^*_{(T,T)}(x) = k_T(x)$$

This **optimality equation** is the very basic equation which will recur over and over again in the context of dynamic programming. Depending on the context we may denote it by J^* or V_* etc. But in all contexts the basic recursive form of the equation will be as it is above.

Note in the case above, $T < \infty$, and the problem is called a finite horizon problem. We will also consider the infinite horizon problem $T \to \infty$ but for that we need to assume that there exist sequences of controls $\{u_k\}$ for which the cost cunction $J^u_{(0,\infty)}(x)$ is defined.
Remark: If we are given a multiplicative cost structure with each component cost non-negative then we can convert it to an additive cost structure by taking the logarithm of the overall cost. If this cannot be done then one has to work with the original optimality form given in the beginning.

Let us make some preliminary remarks based on the optimality equation:

1) Note the optimality equation is a backward in time recursive equation i.e. we need to determine the optimal costs on $[t+1, T]$ before we can determine the optimal cost on $[t, T]$.

2) If we observe the righthand side (r.h.s) we see it is a function of x and u (and of course $\mathcal{A}_t$, $c_t(.,.)$ and $a_t(.,.)$ but these are fixed quantities and therefore optimizing with respect to (w.r.t) u will give u which actually denotes u_t as a function of $x_t = x$ or in otherwards the control is a feedback control. The actual form will depend of the precise form of the functions $c_t(.,.)$, $a_t(.,.)$ and the terminal cost $k_T(.)$

3) Note that in determining the optimal u_t we ignore the past actions since they are accounted for by the fact that we have assumed the state is at $x_t = x$ and so choosing $u_t = u$ only affects the next state $x_{t+1} = a_t(x_t, u_t)$ which appears as an argument in $J^*_{(t+1,T)}(.)$ since for the horizon $[t+1, T]$ we start at x_{t+1}.

Remarks: In finite horizon problems with fixed terminal time or stage T we can reduce our notation by calling $J^*_{(t,T)}(x)$ as $J^*_t(x)$ since T is fixed throughout.

Note in the case where $c_k(.,.)$ and $a_k(.,.)$ do not depend explicitly on the time k, we can rewrite the optimality equation in the forward direction by defining $J^*_{T-t} = V^*_t$ and now specifying the initial cost $V^*_0(x) = k(x)$. This is called the time-invariant or time-homogeneous case where everything depends only on the difference between T and t i.e. $T - t$ and so it makes no difference if we go forwards in time or backwards in time for calculation the optimal cost.

Corollary 1.1 *If the costs do not depend on time and the system is time invariant then the optimality equation can be written as:*

$$V^*_{t+1}(x) = \inf_{u \in \mathcal{A}} \{c(x, u) + V^*_t(a(x, u))\}; \quad t = 0, 1, ..., T-1 \tag{1.5}$$

with $V^*_0(x) = k(x)$. The optimal cost is then given by $V^*_T(x)$.

The basic optimality equation is all that is needed but given a problem we first need to determine what the control actions are and what constitute the states and then write the costs in terms of them.

1.2 Example: Shortest Path Problems

The class of shortest path problems are cononical examples of problems amenable to solution by dymamic programming.

In general a shortest path problem corresponds to finding the shortest path connecting two points on a directed graph. The arc length joining two adjacent points corresponds to the cost associated with choosing that arc.

So a general statement of the problem is given N nodes on a connected graph (it is possible to go from any node to any other node in the graph) with arc lengths specified by $c_{i,j}$ if i and j are neighbours (adjacent) provided an arc connects them, find the path of shortest length connecting any arbitrary nodes j and k on the graph?

In order to convert this statement into our formalism above we need to define our states and controls as well as define or notion of stages or time points. In this context the term stages is better.

Clearly in this context each node can be consiered a state and the choice of arcs corresponds to the control actions. This is obvious since knowing where we start and the arcs we choose to follow we can determine which node we end up at after a prescribed number of stages. Thus in order to define a dynamic programming algorithm it remains to calculate the horizon of the problem and also to guarantee that we terminate after some finite time so as not to have cycles. Our aim is to formulate a dynamic programming algorithm which does not depend on the specific nature of a given graph but is general enough to work on any graph. Thus in specific cases a simpler solution is possible but the algorithm given below will work for any connected graph and the output will give all pairs of shortest paths connecting any two nodes.

To this end suppose we are given a graph $\mathcal{G}$ with N nodes. Let $c_{i,j}$ denote the cost (in this case the length) of the arc connecting i and j. If there is no arc joining i and j we denote the length $c_{i,j} = \infty$.

Hence the total cost of going from any node i to any node j on the graph is denoted by:

$$l_{i,j} = \sum_{i=0}^{M} c_{j_i,j_{i+1}}$$

with the condition that $j_0 = i$ and $j_N = j$ and the assumption that there M possible stages to the problem.

We need to determine what M should be as well as we introduce the notation that $l_{i,i} = c_{i,i} = 0$ i.e. we can choose not to move and in which case the cost of remaining at a given state is 0.

Suppose our graph has N nodes then the maximum number of arcs in any path is bounded by $N-1$ (why?). Thus we need to choose $M = N-1$ or at worst there are $N-1$ stages to the problem without any further assumptions on the graph other than it is connected .

Then the shortest path algorithm for the graph can be directly obtained by dynamic programming as follows:

1. Call the terminal node desired to be t.

2. Let $J_k^*(i)$ denote the optimal cost of going from i to t in k stages (or moves) and therefore $J_1^*(i) = c_{i,t}$.

3. Compute:
$$J_{k+1}^*(i) = min_{j=1,2,..,N}\{c_{i,j} + J_k^*(j)\}; \quad fork = 0, 1, 2, ..., N-1$$
noting the value for j which minimizes the expression. Call it $j_k(i,t)$. Record $J_{N-1}(i)$ and $j_1(i,t), j_2(i,t), ..., j_{N-1}(i,t)$. These correspond to the indices of the arcs which constitute the

shortest path from i to t taken at every stage k i.e. the path is identified by the pairs j_k, j_{k+1} knowing $j_0 = i$ and $j_N = t$. $J_{N-1}(i)$ will be the total length of the shortest path from i to t.

4. Repeat steps 1,2 and 3 for all combinations of i and t.

Remarks: In the set-up of the algorithm, at every stage we can make the decision to remain at the particular node (with cost 0 incurred). These are called degenerate moves but helps in formulating the problem w.r.t a common horizon since not all possible paths will have the same number of arcs.

Another important application which can be posed in the context of shortest path problems is the critical path problem in the context of PERT type of networks. Here arc lengths are the times to complete a given activity and go to the "next" activity and hence we have a directed graph since certain acitivities can be started only after the preceeding activities have been completed and thus unlike the previous example we can go from nodes i to j but not in the reverse direction. The aim is to find the longest (critical path) from the start to the completion of the project since this defines the minimum time for the project to get done. Here we can pose it as a shortest path problem with negative arc costs (since we maximize). In this case the graph is acyclic (no cycles are possible).

1.3 Review of results

In this first part the key ideas are:

- *Bellman's principle of optimality*
- Notion of stages or horizons, controls or decisions and states
- Additive cost functions
- The optimality equation for dynamic programming for additive cost functions
- Backward recursive equations
- Optimal decisions (controls) are *feedback*.

1.4 References:

Some references for this part:

D. Bertsekas, Dynamic Programming: Deterministic and Stochastic Models, Prentice-Hall, N.J., 1976, Chapter 1

There are many other books on Dynamic Programming which can be found in the library.