Math 441 Notes on Dynamic Programming Richard Anstee from Robert Israel

Dynamic Programming determines optimal strategies among a range of possibilities typically putting together 'smaller' solutions. In some cases it is little more than a careful enumeration of the possibilities but can be organized to save effort by only computing the answer to a small problem once rather than many times.

In Graph Theory, the dynamic programming idea yields an effective algorithm for shortest (or indeed longest) paths or walks in a graph (or directed graph) $G = (V, E)$ even those with negative edge weights. Let $w(i, j)$ denote the length of the edge (i, j). Let $d^k(i, j)$ denote the length of the shortest i-j-walk from i to j that uses at most k edges. The base case can either be for $k = 0$

$$d^0(i, i) = 0 \text{ and } d^0(i, j) = \infty \text{ for } i \neq j$$

or for $k = 1$

$$d^1(i, j) = \begin{cases} w(i, j) & \text{for } (i, j) \in E \\ \infty & \text{for } (i, j) \notin E \end{cases}$$

The recurrence becomes for $k \geq 2$:

$$d^k(i, j) = \min_p \{d^{k-1}(i, p) + w(p, j) \ : \ (p, j) \in E\}$$

The parameter k representing the number of arcs is crucial. In some graph theory applications and many Dynamic programming applications the graph is stratified say with vertices partitioned $V = V_0 \cup V_1 \cup V_2 \cup V_3 \cup \cdots \cup V_t$ where V_i denotes vertices at distance (in terms of number of edges) from V_0 and with no edges between vertices of V_i. Thus a vertex of V_i is joined only to vertices from $V_{i-1} \cup V_{i+1}$.

In our applications, the number of edges/arcs k will be replaced by the number of stages which often are time periods (i.e. V_i corresponds to vertices at time i). Typically we will determine the optimal solutions for "i time periods to go" from the optimal solutions for "$i - 1$ time periods to go".

Some notation is helpful, rather fearsome at first glance, but useful in its generality.

Stages: these are often the time periods. Typically they are indexed as $i = n, n - 1, \ldots 1, 0$ where i measures the number of stages left to go before the end of the process. This typically $i = 0$ refers to the final stage.

• **States** Each state is a set of information describing the process at each stage. For many of our problems this can be a single integer.

• s_i The state occupied by the process at stage i.

• $D_i(s_i)$ The set of decisions that are allowed to be taken when the propcess is in state s_i at stage i.

• d_i is the decision taken when the process is at stage i.

• t_i is the transition function, defined on the decisions and states available at stage i. We have the equation

$$s_{i-1} = t_i(d_i, s_i)$$

that gives the outcome of the process at stage $i - 1$ (with $i - 1$ stages to go) after decision d_i is taken when in state s_i. The function depends only on d_i, s_i and not on how the process got to state s_i with i stages to go. We need this for dynamic programming to work.

We need some notion of cost or revenue.

• $f_i(d_i, s_i)$ is the immediate or short term revenue resulting from taking decision d_i while in state s_i with i stages to go.

• $v_i(s_i)$ denotes the long term revenue that can be derived (by selecting decisions optimally) in the remaining i stages given that we are in state s_i with i stages to go.

For most of our problems the function $f_i(d_i, s_i)$ is quite straightforward.

The first problem we consider is the planned building of power plants to meet projected demand. This comes from *Applied Mathematical Programming* by Bradley, Hax, and Magnanti, 1977 . The relevant computations are in a file dynamicbuildnew.xls (on the first sheet) located on our website. We are asked to plan the construction over a number of years. The construction costs for a power plant depend on the year of construction and we are allowed to construct up to three plants in a given year. Moreover if we are doing any construction in a given year, we incur an extra administrative cost of \$1.5M. For us stages are years and states give the number of plants constructed.

stage	year	cumulative demand	cost per plant
6 →			
	1981	1	5.4
5 →			
	1981	2	5.6
4 →			
	1982	4	5.8
3 →			
	1983	6	5.7
2 →			
	1984	7	5.5
1 →			
	1985	8	5.2

Thus for example at stage 4 we must have $s_4 \geq 2$ and the possible decisions are those that result in $s_3 \geq 4$. The decision d_i refers to the number of plants constructed in that year so that $t_i(d_i, s_i) = d_i + s_i$. The 'value' (or in this case 'cost') is

$$f_4(d_4, s_4) = \begin{cases} 0 & d_4 = 0 \\ 5.8d_4 + 1.5 & d_4 > 0 \end{cases}$$

You should note that $v_6(0)$ represents the solution for the entire propblem since it is the cost of being in state 0 (i.e. with no power plants constructed) with 6 years to go.

What follows is the general algorithm (quite easy!) whose main drawback is the potential for too many states and too many stages. This is sometimes called *the combinatorial explosion* although that term often refers to even greater growth. The notion of *memoization* from Computer Science is used to help with these algorithms.

Bellman's Equations.

$$v_i(s_i) = \max\{f_i(d_i, s_i) + v_{i-1}(t_i(d_i, s_i)) \ : \ d_i \in D_i(s_i)\}.$$

We are maximizing the cumulative revenue (or minimizing the cost)

$$f_n(d_n, s_n) + f_{n-1}(d_{n-1}, s_{n-1}) + \cdots + f_0(d_0, s_0) \ : \ d_i \in D_i(s_i) \text{ for } i = n, n-1, \ldots, 1.$$

This simple formulation adapts to many situations and also generalizations. One simple generalization considers the discount rate which while unfamiliar to students is a business practice. Roughly speaking each enterprise can estimate its own discount rate as $e^{-\mu} = \beta < 1$ so that \$1 one year from now is worth $\$e^{-\mu}$=$\$\beta$ today (i.e. slightly less). One can view this as a consequence of interest rates but it occurs as a result of the investment opportunities of a business. Thus the discount rate varies depending on the business or individual.

Bellman's Equations with discounted future returns.

$$v_i(s_i) = \max\{f_i(d_i, s_i) + e^{-\mu} v_{i-1}(t_i(d_i, s_i)) \ : \ d_i \in D_i(s_i)\}.$$

We are maximizing the cumulative revenue (or minimizing the cost)

$$f_n(d_n, s_n) + e^{-\mu} f_{n-1}(d_{n-1}, s_{n-1}) + \cdots + e^{-n\mu} f_0(d_0, s_0) \ : \ d_i \in D_i(s_i) \text{ for } i = n, n-1, \ldots, 1.$$

Again in the file dynamicbuildnew.xls (on the second sheet) we can see the power plant example worked out with discounting. The file is set up so that you could adjust the discount rate. I have set things up so the optimal decisions at each stage are computed so you can change the data.

Bellman's Equations for stochastic dynamic programming. The return (or cost) of a strategy may depend on the outcome of a stochastic event(s). The value of a strategy is measured as an expected value.

$$v_i(s_i) = \max\{E(f_i(d_i, s_i)) + E(v_{i-1}(t_i(d_i, s_i))) \ : \ d_i \in D_i(s_i)\}.$$

An obvious instance of a stochastic event is the sales of a product. A worked example is in the file dynamicsales.xls