1 Dynamic Programming

- Main Idea: Solve problem in multiple stages
- No exact standard form like LP
- Main idea: problems get solved in stages with decisions on how to move from one stage to the next

Typical example: Shortest path problem

![Graph of shortest path problem]

Calculate the distance to go (DTG) from the end point to the starting point.
DTG(I)=4, DTG(H)=3;
DTG(E)=4, DTG(F)=8, DTG(G)=7;
DTG(D)=9, DTG(C)=10, DTG(B)=11;
DTG(A)=12;
The local greedy option is not a good solution (shortest next step is a bad idea).
One option is to calculate all path lengths but it is computationally expensive and takes too long for large problems.

1.1 Solution: Dynamic Programming

- divide the decisions into stages (let \( n \) be the number of stages and \( S \) be the set of states)
- develop a decision policy for each stage
• if I’m in state A then do B
• optional policy minimizes total cost to go for any state

1.2 General Solution Idea

\[ f_n(S_n) = \max_{x_n} f_n(S_n, x_n) \]

(sometimes \( f_n(S_n) \) is minimum, which depends on problem)

where

\( S_n = \) State at time or stage \( n \)
\( x_n = \) decision at time or state \( n \)
\( f_n(S_n, x_n) = \) ”cost to go” of \( S_n \) and \( x_n \) (the cost associated with taking decision \( x_n \) is state \( S_n \)).

1.3 Deterministic DP Problems:
Decision + State \( \rightarrow \) exactly determines next state (this assumption is not always valid in practice, where sometimes there might be some degree of uncertainty).

1.4 Principle of Optimality

• Optimal policy for remaining stages is independent of previous stages

1.5 Solution technique

• Start at the end, move backwards
• Recursive policy:

\[ f_n^*(S_n) = \max_{x_n} \left( c_{xx_n} + f_{n+1}^*(x_n) \right) = \max_{x_n} f_n(S_n, x_n) \]

Where \( S_n = \) state
\( x_n = \) decision
\( f_n(S_n, x_n) = \) ”cost to go” of \( S_n \) and \( x_n \)

Deterministic DP: Decision and state exactly determines next state
Stochastic DP: Some uncertainty

1.6 Some examples

Deterministic:

• Shortest path
• Finite Limited resources with discrete options, e.g. split 5 teams over 3 countries. (common example in text)
• we have a value (cost) of sending \( x \) teams to country \( i \)

Objective: \( \sum_{i=1}^{n} p_i(x_i) \) s.t. \( \sum x_i = 5, x_i \epsilon \{0, 1, 2, ... 5\} \)
• \( P = \) total resource

3 stages, i.e. split \( P \) into 3 groups
Stages: 3 stages
State = remaining # resources

Solving the problem consists of taking 3 separate decisions. It is very similar to a shortest path problem.

1.6.1 Inventory control

\( x_k = \) stock available at time \( k \)
\( u_k = \) stock ordered at time \( k \)
\( w_k = \) demand in \( k \)th period (maybe uncertain)
\( r(x_k) = \) holding cost
\( C(u_k) = \) cost to order
\( R(x_N) = \) terminating cost

Objective: \( \min E[R(x_n) + \sum_{k=0}^{N-1} [r(x_k) + c(u_k)]] \)

Control Policy: given \( x_k \) at time \( k_1 \), what should \( u_k \) be?
Open loop: Make a decision now
Closed loop: Reconsider each stage
how much = control policy \( \mu_k \)
Works out to be optimal policy:
\[
\mu_k(x_k) = \begin{cases} 
S_k - x_k & \text{if } x_k < s_k \\
0 & \text{otherwise}
\end{cases}
\]

1.6.2 Machine Replacement

I own an expensive machine and it slowly degrades over time. When should I replace it?

Machine in one of many states \( i \)
\( g(1) \leq g(2) \leq \ldots \leq g(n) \)
\( g(i) = \) cost to operate machine when in state \( i \)
\( P_{ij} = P[\text{Next state} = j \mid \text{current} = i] \) (= 0 for \( j < i \))

Question: In what state should I replace the machine?
Solution = threshold policy, i.e. if state worst than \( i* \) then replace
1.6.3 Game Strategy

Consider a game of chess
Tournament between you and opponent

- 2 games
- Winner: most points (1 = win, 0.5 = draw)

If tied after 2 games then sudden death: First to win a game wins tournament
You have two strategies: Timid/Bold
Timid: Draw with probability $p_d$ else lose
Bold: Win with probability $p_w$ else lose
Consider open loop strategy options
- After 2 games play bold
"cost to go" ↔ probability of winning

After two games:
$P(\text{win}) = 1$ if score is 2-0 or 1.5-0.5
$p_w$ if score 1-1
0 otherwise
Options: TT → $p_d p_d p_w$
TB → $p_w p_d + p_w^2 (1 - p_d)$
BT → $p_w p_d + p_d^2 (1 - p_d)$
BB → $p_w^2 + p_w (1 - p_w) p_w$

Tie + win: $p_d p_w$
Lose + win: $(1 - p_d) p_w p_w$
Total $P(\text{win}) = p_w p_d + p_w^2 (1 - p_d)$
Say $p_w = 0.45$ and $p_d = 0.9$
Best strategy = TB or BT as long as $p_d > p_w$
Closed loop:
Play 1 game then decide
- Play Bold if win

Win with prob $p_w + (1 - p_w) p_w = 2p_w - p_w^2$
Win = $p_d + (1 - p_d) p_w = p_d + p_w - p_d p_w$

$2p_w - p_w^2 > p_d + p_w - p_d p_w$
$p_w - p_w^2 > p_d - p_d p_w$
If you lose

Bold: lose then play bold 2nd game, extra $p_w^2$ chance

Optimal policy:
Bold:

- Timid if ahead
- Bold if behind

Total win probability: $P_{win} = p_d^2(1 - w_p) + p_w(1 - p_w)p_d$

with our numbers = 0.53

"Value of information"
open loop-closed loop = 0.105

Moral: We thought TB and BT were equivalent on first pass. In fact BT is better because we get more info as long as $p_d > p_w$

Why didn’t we get this right the first time? We have extra information. If we’re losing we change. e.g. Play bold and lose then play bold again.

2 Solving Practical Dynamic Programming Problems

The general idea is to find an optimal decision per state considering the cost to go for each state.

$$f^*(S_n) = \min_{x_n} \max f_n(x_n, S_n)$$

The right hand side of the equation above is sometimes broken into an immediate cost and a cost to go.

This can be transformed into a problem where we are essentially looking for a fixpoint of a set of equations. This can be written as:

$$F^* = TF^*$$

In the equation above, $T$ is a transformation matrix and $F^*$ is a vector containing functions for the cost to go to a state. To emphasize the iterative solving of the problem, the equation above can be rewritten as: $F_N = TF_{N+1}$.

Possible ways to solve such a problem:
• keep updating $F_{N+1}$ as a function of $F_N$ in the equation above ($T$ encodes decisions at time $N$)
• policy iteration: updating $T$ based on estimates of $F$
• linear program: feasibility problem for the equation describing the dynamic programming problem.

3 Summary
"Finite horizon" Dynamic Programming have a finite set of decisions and finite number of steps.

• Solve as we have seen, using steps from last decision to first.
• Becomes difficult if large number of options or stages.
• Using Dynamic Programming instead of testing all options reduces from $o(D^N)$ to $o(DN)$ , where

\[ D = \# \text{ decision options} \]
\[ N = \# \text{ stages} \]

More often, we have "infinite horizon" problems

• Finite set of options.
• Infinite number of times to make decision

End up looking for solution where

\[ f_n(S_n) = f_{n+1}(S_n) \]
\[ f_n^*(S_n) = \max_{x_n} (c(x_n) + f_{n+1}^*(x_n)) \]

• Recursion of optimal policies and cost to go.
  ie. The same decision and cost remaining must apply for every state.

"Policy Iteration" = Keep updating the policy (decision) until the optimal is constant.
"Value Iteration" = Keep updating the values of $f_n(S_n)$ until the values are constant.

3.1 Example
Claim: Can win $\frac{2}{3}$ of the time
Challenge: Go from 3 chips to 5 chips in 3 plays
Q: What is optimal play for 3 rounds and what is probability of gaining from 3 to 5.
Rule is: Bet $n$ chips and gain $n$ if win, lose $n$ if lose.

Decision = how many to bet in each round
Let $x_n = \text{number of chips to bet in round } n$
$S_n = \text{number of chips available of state of round } n$

• Solve stage 3 first:
  $S_3$ could be 0, 1, 2, ..., 12
  $f_3^*(0) = 0$ (lost)
  $f_3(2) = f_3^*(1) = 0$ (lost)
\[ f_3^*(3) = \frac{2}{3}, \quad x_3^*(3) = \text{Play all chips} \]
\[ f_3^*(4) = \frac{2}{3}, \]
\[ f_3^*(5) = 1, \quad x_3^*(5) = \text{Bet 0} \]
\[ f_3^*(n > 5) = 1, \quad x^* = \text{Bet 0} \]

- Now look at stage 2:
  \[ f_2(S_2, x_2) = \frac{1}{4} f_3^*(S_2 - x_2) + \frac{3}{4} f_3^*(S_2 + x_2) \]

<table>
<thead>
<tr>
<th>( S_2 )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>( x^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2 × \frac{2}{3} × \frac{2}{3} × \frac{2}{3} × \frac{2}{3} × \frac{2}{3} = 1 or 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>\frac{2}{3}</td>
<td>\frac{4}{9}</td>
<td>\frac{2}{3}</td>
<td>\frac{2}{7}</td>
<td>-</td>
<td>0, 2 or 3</td>
</tr>
<tr>
<td>4</td>
<td>\frac{2}{3}</td>
<td>\frac{8}{9}</td>
<td>\frac{2}{3}</td>
<td>\frac{2}{3}</td>
<td>\frac{2}{3}</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

- Then stage 1:
  \[ f_1(S_1, x_1) = \frac{1}{3} f_2^*(S_1 - x_1) + \frac{2}{3} f_2^*(S_1 + x_1) \]

start with 3,

<table>
<thead>
<tr>
<th>( S_1 )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>( x^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>\frac{2}{3}</td>
<td>\frac{1}{3} \times \frac{2}{3} \times \frac{2}{3} \times \frac{2}{3} = \frac{20}{27}</td>
<td>\frac{2}{3}</td>
<td>\frac{2}{3}</td>
<td>x^* = 1, \quad f^* = \frac{20}{27}</td>
</tr>
</tbody>
</table>