

Dynamic Programming and Reinforcement Learning

Lecture 1: Intro to Dynamic Programming

Mengdi Wang

Operations Research and Financial Engineering
Princeton University

July 24-29, 2016

Approximate Dynamic Programming

Based on the graduate-level courses:

- MIT “Dynamic Programming and Optimal Control,”
- Princeton “Linear and Nonlinear Optimization”

Based on the books:

- “Neuro-Dynamic Programming,” by DPB and J. N. Tsitsiklis, Athena Scientific, 1996
- “Dynamic Programming and Optimal Control, Vol. II: Approximate Dynamic Programming,” by DPB, Athena Scientific, 2012
- “Abstract Dynamic Programming,” by DPB, Athena Scientific, 2013

For a fuller set of slides, see

<http://web.mit.edu/dimitrib/www/publ.html>

Approximate Dynamic Programming

BRIEF OUTLINE I

Our subject:

- Large-scale DP based on approximations and in part on simulation.
- This has been a research area of great interest for the last 25 years known under various names (e.g., reinforcement learning, neuro-dynamic programming)
- Emerged through an enormously fruitful cross-fertilization of ideas from artificial intelligence and optimization/control theory
- Deals with control of dynamic systems under uncertainty, but applies more broadly (e.g., discrete deterministic optimization)
- A vast range of applications in control theory, operations research, finance, robotics, computer games, and beyond (e.g., AlphaGo) ...
- The subject is broad with rich variety of theory/math, algorithms, and applications. Our focus will be mostly on algorithms ... less on theory and modeling

Approximate Dynamic Programming

BRIEF OUTLINE II

Our aim:

- A state-of-the-art account of some of the major topics at a graduate level
- Show how to use approximation and simulation to address the dual curses of DP: **dimensionality and modeling**

Our 5-lecture plan:

- 2 lectures on **exact DP** with emphasis on infinite horizon problems and issues of large-scale computational methods
- 1.5 lecture on **general issues of approximation and simulation** for large-scale problems: **approximate policy iteration, aggregation methods**
- 1.5 lecture on **Reinforcement Learning**
- **Computational experiments** on option pricing. Bring your laptop with matlab installed!

Today

- ① Introduction and Examples
- ② Finite-Horizon DP: Theory and Algorithms
- ③ Experiment: Option Pricing

Dynamic Programming

One powerful tool to solve certain types of optimization problems is called *dynamic programming* (DP).

- The idea is similar to *recursion*

To illustrate the idea of recursion, consider you want to compute $f(n) = n!$

- You know $n! = n \times (n - 1)!$. Therefore, if you know $f(n - 1)$ then you can compute $n!$ easily.
- Thus you only need to focus on computing $f(n - 1)$
- To compute $f(n - 1)$, you apply the same idea, you only need to compute $f(n - 2)$.. so on so forth
- And you know $f(1) = 1$

Factorial

Recursive way to compute factorial ($f(n)$):

$$f(n) = \begin{cases} 1 & n = 1 \\ n \times f(n-1) & n > 1 \end{cases}$$

```
function y = factorial_5531(n)
    if n == 1
        y = 1;
    else
        y = n * factorial_5531(n-1);
    end
```

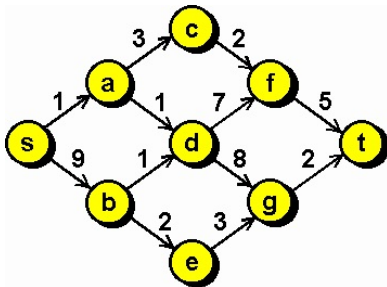
Dynamic Programming

- Basically, we want to solve a big problem that is hard
- We can first solve a few *smaller* but *similar* problems, if those can be solved, then the solution to the big problem will be easy to get
- To solve each of those smaller problems, we use the same idea, we first solve a few even smaller problems.
- Continue doing it, we will eventually encounter a problem we know how to solve

Dynamic programming has the same feature, the difference is that at each step, there might be some optimization involved.

Shortest Path Problem

You have a graph, you want to find the shortest path from s to t



Here we use d_{ij} to denote the distance between node i and node j

DP formulation for the shortest path problem

Let V_i denote the shortest distance between i to t .

- Eventually, we want to compute V_s
- It is hard to directly compute V_i in general
- However, we can just look one step

We know if the first step is to move from i to j , the shortest distance we can get must be $d_{ij} + V_j$.

- To minimize the total distance, we want to choose j to minimize $d_{ij} + V_j$

To write into a math formula, we get

$$V_i = \min_j \{d_{ij} + V_j\}$$

DP for shortest path problem

We call this the recursion formula

$$V_i = \min_j \{d_{ij} + V_j\} \text{ for all } i$$

We also know if we are already at our destination, then the distance is 0. I.e.,

$$V_t = 0$$

The above two equations are the DP formulation for this problem

Solve the DP

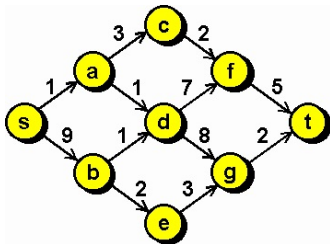
Given the formula, how to solve the DP?

$$V_i = \min_j \{d_{ij} + V_j\} \text{ for all } i, \quad V_t = 0$$

We use backward induction.

- From the last node (which we know the value), we solve the values of V 's backwardly.

Example



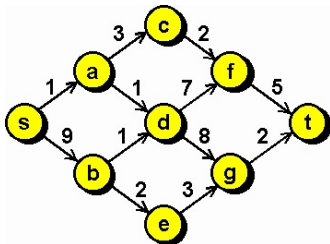
We have $V_t = 0$. Then we have

$$V_f = \min_{(f,j) \text{ is a path}} \{d_{fj} + V_j\}$$

Here, we only have one path, thus $V_f = 5 + V_t = 5$

- Similarly, $V_g = 2$

Example continued



We have $V_t = 0$, $V_f = 5$ and $V_g = 2$

Now consider c , d , e . For c and e there is only one path

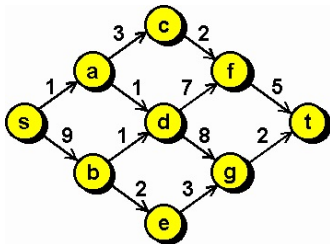
- $V_c = d_{cf} + V_f = 7$, $V_e = d_{eg} + V_g = 5$

For d , we have

$$V_d = \min_{(d,j) \text{ is a path}} \{d_{dj} + V_j\} = \min\{d_{df} + V_f, d_{dg} + V_g\} = 10$$

The optimal way to choose at d is go to g

Example continued



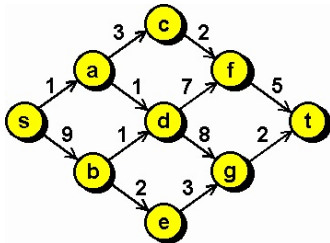
We got $V_c = 7$, $V_d = 10$, $V_e = 5$. Now we compute V_a and V_b .

$$V_a = \min\{d_{ac} + V_c, d_{ad} + V_d\} = \min\{3 + 7, 1 + 10\} = 10$$

$$V_b = \min\{d_{bd} + V_d, d_{be} + V_e\} = \min\{1 + 10, 2 + 5\} = 7$$

and the optimal path to go at a is to choose c , and the optimal path to go at b is to choose e .

Example continued



Finally, we have

$$V_s = \min\{d_{sa} + V_a, d_{sb} + V_b\} = \min\{1 + 10, 9 + 7\} = 11$$

and the optimal path to go at s is to choose a

Therefore, we found the optimal path is 11, and by connecting the optimal path, we get

$$s \rightarrow a \rightarrow c \rightarrow f \rightarrow t$$

Summary of the example

In the example, we saw that we have those V_i 's, indicating the shortest length to go from i to t .

- We call this V the **value function**

We also have those nodes s, a, b, \dots, g, t .

- We call them the **states** of the problem
- The value function is a function of the state

And the recursion formula

$$V_i = \min_j \{d_{ij} + V_j\} \text{ for all } i$$

connects the value function at different states. It is known as the **Bellman equation**

Dynamic Programming Framework

The above is the general framework of dynamic programming problems.

To formulate a problem into a DP problem, the first step is to define the states

- The state variables should be in some order (usually either time order or geographical order)
- In the shortest path example, the state is simply each node

We call the set of all the state the *state space*

- In this example, the state space is all the nodes

Defining the appropriate state space is the most important step in DP.

Definition

A state is a collection of variables that summarize all the (historical) information that is useful for (future) decisions. Conditioned on the state, the problem becomes Markov (i.e., memoryless).

DP: Actions

There will be an action set at each state

- At state x , we denote the set by $A(x)$
- For each action $a \in A(x)$, there is an immediate cost $r(x, a)$
- If one exerts action a , the system will go to some next state $s(x, a)$
- In the shortest path problem, the action at each state is the path it can take. There is a distance for each path which is the immediate cost. And after you take a certain path, the system will be at the next node

DP: Value Functions

There is a value function $V(\cdot)$ at each state. The value function denotes that if you choose the optimal action from this state and onward, what is the optimal value.

- In the shortest path example, the value function is the shortest distance between the current state to the destination

Then a recursion formula can be written to link the value functions:

$$V(x) = \min_{a \in A(x)} \{r(x, a) + V(s(x, a))\}$$

In order to be able to solve the DP, one has to know some terminal values (boundary values) of this V function

- In the shortest path example, we know $V_t = 0$
- The recursion for value functions is known as the Bellman equation.

Some more general framework

The above framework is to minimize the total cost. In some cases, one wants to maximize the total profit. Then the $r(x, a)$ can be viewed as the immediate reward.

The DP in those cases can be written as

$$V(x) = \max_{a \in A(x)} \{r(x, a) + V(s(x, a))\}$$

with some boundary conditions

Stochastic DP

In some cases, when you choose action a at x , the next state is not certain (e.g., you decide a price, but the demand is random).

- There will be $p(x, y, a)$ probability you move from x to y if you choose action $a \in A(x)$

Then the recursion formula becomes:

$$V(x) = \min_{a \in A(x)} \{r(x, a) + \sum_y p(x, y, a) V(y)\}$$

or if we choose to use the expectation notation:

$$V(x) = \min_{a \in A(x)} \{r(x, a) + \mathbb{E}V(s(x, a))\}$$

Example: Stochastic Shortest Path Problem

Stochastic setting:

- One no longer controls which exact node to jump to next
- Instead one can choose between different actions $a \in \mathcal{A}$
- Each action a is associated with a set of transition probabilities $p(j \mid i; a)$ for all $i, j \in \mathcal{S}$.
- The arc length may be random w_{ija}

Objective:

- One needs to decide on the action for every possible current node. In other words, one wants to find a **policy** or **strategy** that maps from \mathcal{S} to \mathcal{A} .

Bellman Equation for Stochastic SSP:

$$V(i) = \min_a \sum_{j \in \mathcal{S}} p(j \mid i; a)(w_{ija} + V(j)), \quad i \in \mathcal{S}.$$

Bellman equation continued

We rewrite the Bellman equation

$$V(i) = \min_a \sum_{j \in \mathcal{S}} p(j \mid i; a)(w_{ija} + V(j)), \quad i \in \mathcal{S}.$$

in a vector form:

$$V = \min_{\mu: \mathcal{S} \mapsto \mathcal{A}} g_{\mu} + P_{\mu} V$$

where

- average transitional cost starting from i to the next state:

$$g_{\mu}(i) = \sum_{j \in \mathcal{S}} p(j \mid i, \mu(i)) w_{ij, \mu(i)}$$

- transition probabilities

$$P_{\mu}(i, j) = p(j \mid i, \mu(i))$$

- $V(i)$ is the expected length of stochastic shortest path starting from i , also known as the value function or **cost-to-go** vector
- V^* is the optimal value function or cost-to-go vector that solves the Bellman equation

Example: Game of picking coins

Assume there is a row of n coins of values v_1, \dots, v_n where n is an even number. You and your opponent takes turn to

- Either pick the first or last coin in this row, obtain that value. That coin is then removed from this row
- Both player wants to maximize his total value



Example

Example: 1, 2, 10, 8

- If you choose 8, then your opponent will choose 10, then you choose 2, your opponent chooses 1. You get 10 and your opponent gets 11.
- If you choose 1, then your opponent will choose 8, you choose 10, your opponent chooses 2. You get 11 and your opponent gets 10

When there are many coins, it is not very easy to find the optimal strategy

- As you have seen, greedy strategy doesn't work

Dynamic programming formulation

Given a sequence of numbers v_1, \dots, v_n . Let the state be the remaining positions that yet to be chosen.

- That is, the state is a pair of i and j with $i \leq j$, meaning the remaining coins are v_i, v_{i+1}, \dots, v_j

The value function $V(i, j)$ denotes the maximum values you can take if the game starts when the coins are v_i, v_{i+1}, \dots, v_j (and you are the first one to move)

The action at state (i, j) is easy:

- Either take i or take j

The immediate reward function

- If you take i , then you get v_i
- If you take j , then you get v_j

What will be the next state if you choose i at state (i, j) ?

Example continued

Consider the current state (i, j) if you picked i .

- Your opponent will either choose $i + 1$ or j .
- When he chooses $i + 1$, the state will become $(i + 2, j)$
- When he chooses j , the state will become $(i + 1, j - 1)$
- He will choose to make most value of the remaining coins, i.e., leave you with the least value of the remaining coins
- Similar argument if you take j

Therefore, we can write down the recursion formula

$$V(i, j) = \max\{v_i + \min\{V(i + 2, j), V(i + 1, j - 1)\}, \\ v_j + \min\{V(i + 1, j - 1), V(i, j - 2)\}\}$$

And we know $V(i, i + 1) = \max\{v_i, v_{i+1}\}$ (if there are only two coins remaining, you pick the larger one)

Dynamic programming formulation

Therefore, the DP for this problem is:

$$V(i, j) = \max\{v_i + \min\{V(i+2, j), V(i+1, j-1)\}, \\ v_j + \min\{V(i+1, j-1), V(i, j-2)\}\}$$

with $V(i, i+1) = \max\{v_i, v_{i+1}\}$ for all $i = 1, \dots, n-1$.

How to solve this DP?

- We know $V(i, j)$ when $j - i = 1$
- Then we can easily solve $V(i, j)$ for any pair with $j - i = 3$
- Then all the way we can solve the initial problem

Code

If you were to code this game in Matlab:

```
function [value, strategy] = game(x)
    [~, b] = size(x);           % Compute the number of remaining coins
    if b <= 2                   % If less than 2 left
        [value, strategy] = max(x); % Your value is the larger one
    else
        [value1, ~] = game(x(3:b)); % Otherwise, follow the DP recursion
        [value2, ~] = game(x(2:(b-1)));
        [value3, ~] = game(x(1:(b-2)));
        [value, strategy] = max([x(1) + min([value1, value2]), x(b) + min([value2, value3])]);
    end
```

- It is very short.
- Doesn't depend on the input size
- All of these thanks to the recursion formula

Summary of this example

The state variable is a two-dimensional vector, indicating the “remaining range”

- This is one typical way to set up the state variables

We used DP to solve the optimal strategy of a two-person game

- In fact, for computers to solve games, DP is the main algorithm

For example, to solve a go or chess game. One can define the state space of the location of each piece/stone.

- The value function of a state is simply whether this is a win state or loss state
- When it is at certain state, the computer will consider all actions. The next state will be the most adversary state the opponent can give you after your move and his move
- The boundary states are the checkmate states

DP for games

DP is roughly how computer plays games.

- In fact, if you just use the DP and code it, the code will probably be no more than a page (excluding interface of course)

However, there is one major obstacle - the state space is huge

- For chess, each piece could take one of the 64 spots (and also could have been removed), there are roughly 32^{65} states
- For Go (weiqi), each spot on the board (361 spots) could be occupied by white, black or neither. There are 3^{361} states

It is impossible for current computers to solve that large problem

- This is called the “curse of dimensionality”
- To solve it, people have developed approximate dynamic programming techniques to get approximate good solutions (both the approximate value function and optimal strategy)

More Applications of (Approximate) DP

- Control of complex systems:
 - Unmanned vehicle/aircraft
 - Robotics
 - Planning of power grid
 - Smart home solution
- Games:
 - 2048, Go, Chess
 - Tetris, Poker
- Business:
 - Inventory and supply chain
 - Dynamic pricing with demand learning
 - Optimizing clinical pathway (healthcare)
- Finance:
 - Option pricing
 - Optimal execution (especially in dark pools)
 - High-frequency trading

- 1 Introduction and Examples
- 2 Finite-Horizon DP: Theory and Algorithms
- 3 Experiment: Option Pricing

Abstract DP Model

Discrete-time System

State transition:

$$x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, \dots, N - 1$$

- x_k : **state**; summarizing past information that is relevant for future optimization
- u_k : **control/action**; decision to be selected at time k from a given set U_k
- w_k : **random disturbance** or noise
- $g_k(x_k, u_k, w_k)$: **state transitional cost** incurred at time k given current state x_k and control u_k
- For every k and every x_k , we want an optimal action. We look for **a mapping μ from states to actions**.

Abstract DP Model


Objective - control the system to minimize overall cost

$$\min_{\mu} \mathbf{E} \left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) \right\},$$

subject to $u_k = \mu(x_k), \quad k = 0, \dots, N - 1.$

We look for a policy/strategy μ , which is a mapping from states to actions.

An Example in Revenue Management: Airfare Pricing

Nonstop flights from \$303					
from \$303 5 tickets at this price Select 	Depart: 10:42 p.m. Sat., Jan. 1, 2013 San Francisco, CA (SFO)	Arrive: 4:45 a.m. +1 Day Sun., Jan. 2, 2013 Chicago, IL (ORD - O'Hare)	Travel Time: 4 hr 3 mn	Distance: 1,846 miles	Flight: UA214 Aircraft: Boeing 737-200 Fare Class: United Economy (W) Meal: Snacks For Purchase No Special Meal Offered. See On-Time Performance View Seats
from \$343 1 ticket at this price Select	Depart: 4:30 p.m. Sat., Jan. 1, 2013 San Francisco, CA (SFO)	Arrive: 10:39 p.m. Sat., Jan. 2, 2013 Chicago, IL (ORD - O'Hare)	Travel Time: 4 hr 9 mn	Distance: 1,846 miles	Flight: UA705 Aircraft: Airbus A319 Fare Class: United Economy (V) Meal: Meals For Purchase No Special Meal Offered. See On-Time Performance View Seats
from \$343 2 tickets at this price Select	Depart: 11:10 p.m. Sat., Jan. 1, 2013 San Francisco, CA (SFO)	Arrive: 5:20 a.m. +1 Day Sun., Jan. 2, 2013 Chicago, IL (ORD - O'Hare)	Travel Time: 4 hr 10 mn	Distance: 1,846 miles	Flight: UA1193 Aircraft: Boeing 737-800 Fare Class: United Economy (V) Meal: Snacks For Purchase No Special Meal Offered. See On-Time Performance View Seats
from \$373 1 ticket at this price Select	Depart: 9:30 a.m. Sat., Jan. 1, 2013 San Francisco, CA (SFO)	Arrive: 3:40 p.m. Sat., Jan. 1, 2013 Chicago, IL (ORD - O'Hare)	Travel Time: 4 hr 10 mn	Distance: 1,846 miles	Flight: UA195 Aircraft: Boeing 737-800 Fare Class: United Economy (Q) Meal: Meals For Purchase No Special Meal Offered. See On-Time Performance View Seats

The price corresponding to each fare class rarely changes (this is determined by other department), however, the RM department determines when to close low fare classes

- From the passenger's point of view, when the RM system closes a class, the fare increases
- Closing fare class achieves **dynamic pricing**

Fare classes

And when you make booking, you will frequently see messages like

Shanghai SHA 12:00 Beijing PEK 14:20 Duration 2h20m Direct
China Southern Airlines CZ3908
Lowest fare/person CNY 540 Economy Select
+CNY 160 taxes & fees
Flight details All fares Only 3 seats left

\$179 Delta SFO 4:00p ► JFK 12:15a 5h 15m nonstop
Only 2 seats left at this price
Select Delta \$179 Coach Wi-Fi
Details Fares

This is real. It means there are only that number of tickets at that fare class (there is one more sale that will trigger the next protection level)

- You can try to buy one ticket with *only one remaining*, and see what happens

Dynamic Arrival of Consumers

Assumptions

- There are T periods in total indexed forward (the first period is 1 and the last period is T)
- There are C inventory at the beginning
- Customers belong to n classes, with $p_1 > p_2 \dots > p_n$
- In each period, there is a probability λ_i that a class i customer arrives
- Each period is small enough so that there is at most one arrival in each period

Decisions:

- When at period t and when you have x inventory remaining, which fare class should you accept (if such a customer comes)
- Instead of finding a single optimal price or reservation level, we now seek for a **decision rule**, i.e., a mapping from (t, x) to $\{I \mid I \subset \{1, \dots, n\}\}$.

Dynamic Arrival - a T -stage DP problem

- State: Inventory level x_k for stages $k = 1, \dots, T$
- Action: Let $\mathbf{u}^{(k)} \in \{0, 1\}^n$ to be the decision variable at period k

$$u_i^{(k)} = \begin{cases} 1 & \text{Accept class } i \text{ customer} \\ 0 & \text{Reject class } i \text{ customer} \end{cases}$$

decision vector $\mathbf{u}^{(k)}$ at stage k , where $u_i^{(k)}$ decides whether to accept the i th class

- Random disturbance: Let $w_k \in \{0, \dots, T\}$ denotes the type of new arrival during the k th stage (type 0 means no arrival). Then $P(w_k = i) = \lambda_i$ for $i = 1, \dots, T$ and $P(w_k = 0) = 1 - \sum_{i=1}^n \lambda_i$.

Value Function: A Rigorous Definition

- State transition cost:

$$g_k(x_k, u^{(k)}, w_k) = u_{w_k}^{(k)} p_{w_k}$$

where we take $p_0 = 0$. Clearly $\mathbf{E} [g_k(x_k, u^{(k)}, w_k) \mid x_k] = \sum_{i=1}^n u_i^{(k)} p_i \lambda_i$

- State transition dynamics:

$$x_{k+1} = \begin{cases} x_k - 1 & \text{if } u_{w_k}^{(k)} w_k = 1 \text{ (with probability } \sum_{i=1}^n u_i^{(k)} \lambda_i) \\ x_k & \text{otherwise (with probability } 1 - \sum_{i=1}^n u_i^{(k)} \lambda_i) \end{cases}$$

The overall revenue is

$$\max_{\mu_1, \dots, \mu_T} \mathbf{E} \left[\sum_{k=0}^T g_k(x_k, \mu_k(x_k), w_k) \right]$$

subject to the $\mu_k : x \mapsto \{\mathbf{u}\}$ for all k .

A Dynamic Programming Model

- Let $V_t(x)$ denote the optimal revenue one can earn (by using the optimal policy onward) starting at time period t with inventory x

$$V_t(x) = \max_{\mu_t, \dots, \mu_T} \mathbf{E} \left[\sum_{k=t}^T g_k(x_k, \mu_k(x_k), w_k) \mid x_t = x \right]$$

- We call $V_t(x)$ the **value function** (a function of stage t and state x)
- Suppose that we know the optimal pricing strategy from time $t + 1$ for all possible inventory levels x .
- More specifically, suppose that we know $V_{t+1}(x)$ for all possible **state** x .
- Now let us find the best decisions at time t .

Prove the Bellman Equation

We derive the Bellman equation from the definition of value function:

$$\begin{aligned} V_t(x) &= \max_{\mu_t, \dots, \mu_T} \mathbf{E} \left[\sum_{k=t}^T g_k(x_k, \mu_k(x_k), w_k) \mid x_t = x \right] \\ &= \max_{\mu_t, \dots, \mu_T} \mathbf{E} \left[g_t(x_t, \mu_t(x_t), w_t) + \sum_{k=t+1}^T g_k(x_k, \mu_k(x_k), w_k) \mid x_t = x \right] \\ &= \max_{\mu_t, \dots, \mu_T} \mathbf{E} \left[g_t(x, \mu_t(x), w_t) + \mathbf{E} \left[\sum_{k=t+1}^T g_k(x_k, \mu_k(x_k), w_k) \mid x_{t+1} \right] \mid x_t = x \right] \\ &= \max_{\mu_t} \mathbf{E} \left[g_t(x, \mu_t(x), w_t) + \max_{\mu_{t+1}, \dots, \mu_T} \mathbf{E} \left[\sum_{k=t+1}^T g_k(x_k, \mu_k(x_k), w_k) \mid x_{t+1} \right] \mid x_t = x \right] \\ &= \max_{\mu_t} \mathbf{E} [g_t(x, \mu_t(x), w_t) + V_{t+1}(x_{t+1}) \mid x_t = x] \end{aligned}$$

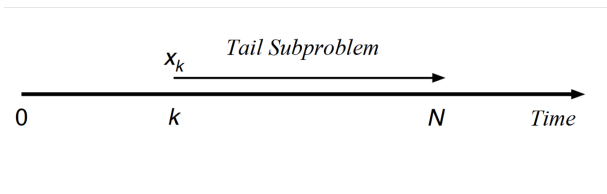
The maximization is attained at the optimal policy μ_t^* for the t th stage.

Tail Optimality

Bellman Equation:

$$V_t(x) = \max_{\mu_t} \mathbf{E} [g_t(x, \mu_t(x), w_t) + V_{t+1}(x_{t+1}) \mid x_t = x]$$

Key Property of DP: A strategy μ_1^*, \dots, μ_T^* is optimal, if and only if every tail strategy μ_t^*, \dots, μ_T^* is optimal for the tail problem starting at stage t .



Bellman's Equation for Dynamic Arrival Model

We just proved the Bellman's equation. In the airfare model, Bellman's equation is

$$V_t(x) = \max_{\mathbf{u}} \left\{ \sum_{i=1}^n \lambda_i (p_i u_i + V_{t+1}(x - u_i)) + (1 - \sum_{i=1}^n \lambda_i u_i) V_{t+1}(x) \right\}$$

with $V_{T+1}(x) = 0$ for all x and $V_t(0) = 0$ for all t .

We can rewrite this as

$$V_t(x) = V_{t+1}(x) + \max_{\mathbf{u}} \sum_{i=1}^n \lambda_i u_i (p_i + V_{t+1}(x - 1) - V_{t+1}(x))$$

For every (t, x) , we have an equality and an unknown. The Bellman equation bears a unique solution.

Dynamic Programming Analysis

$$V_t(x) = V_{t+1}(x) + \max_{\mathbf{u}} \sum_{i=1}^n \lambda_i u_i (p_i - \Delta V_{t+1}(x))$$

Therefore the optimal decision at time t with inventory x should be

$$u_i^* = \begin{cases} 1 & p_i \geq \Delta V_{t+1}(x) \\ 0 & p_i < \Delta V_{t+1}(x) \end{cases}$$

This is also called bid-price control policy

- The bid-price is $\Delta V_{t+1}(x)$
- If the customer pays more than the bid-price, then accept
- Otherwise reject

Dynamic Programming Analysis

Of course, to implement this strategy, we need to know $\Delta V_{t+1}(x)$

- We can compute all the values of $V_{t+1}(x)$ backwards
- Computational complexity is $O(nCT)$
- With those, we can have a whole table of $V_{t+1}(x)$. And we can execute based on that

Proposition (Properties of the Bid-prices)

For any x and t

- ① $\Delta V_t(x+1) \leq \Delta V_t(x)$
- ② $\Delta V_{t+1}(x) \leq \Delta V_t(x)$

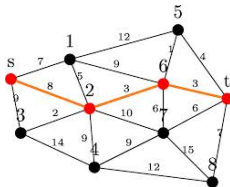
Intuitions:

- Fixed t , the value of the inventory has decreasing marginal returns
- The more time one has, the more valuable an inventory worth
- Proof by induction using the DP formula

From DP to Shortest Path Problem

Theorem

- Every deterministic DP is a SSP
- Every stochastic DP is a stochastic SSP



Shortest Path Problem (SSP): Given a graph $G(V, E)$

- V is the set of nodes $i = 1, \dots, n$ (node = state)
- E is the set of arcs with length $w_{ij} > 0$ if $(i, j) \in E$
(arc = state transition from i to j)
(arc length = state transition cost g_{ij})

Find the shortest path from a starting node s to a termination node t .
(Minimize the total cost from the first stage to the end)

Finite-Horizon: Optimality Condition = DP Algorithm

Principle of optimality

The tail part of an optimal policy is optimal for the tail subproblem

DP algorithm

Start with $J_N(x_N) = g_N(x_N)$, and go backwards for $k = N - 1, \dots, 0$ using

$$J_k(x_k) = \min_{u_k \in U_k} \mathbf{E}_{w_k} \{g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k))\}.$$

Proof by induction that the principle of optimality is always satisfied.

This DP algorithm is also known as **value iteration**.

Finite-Time DP Summary

Dynamic programming is a very useful tool for solving complex problems by breaking them down into simpler subproblems

- The recursion idea gives a very neat and efficient way to compute the optimal solution
- Finding the states is the key, you should have a basic understanding of it and once the states are given, be able to write down the DP formula.
- It is very important technique in modern decision making problems

Main Theory:

- Tail Optimality and Bellman equation
- Backward induction is value iteration

- 1 Introduction and Examples
- 2 Finite-Horizon DP: Theory and Algorithms
- 3 Experiment: Option Pricing

Option Pricing

Option is a common financial product written/sold by sellers.

Definition

An option provides the holder with the right to buy or sell a specified quantity of an underlying asset at a fixed price (called a strike price or an exercise price) at or before the expiration date of the option.

- Since it is a right and not an obligation, the holder can choose not to exercise the right and allow the option to expire.
- Option pricing means to find the intrinsic expected value of the right.
- There are two types of options - **call options** (right to buy) and **put options** (right to sell).
- The seller needs to set a fair price to the option so that no one can take advantage of misprice.

Call Options

A call option gives the buyer of the option the right to buy the underlying asset at a fixed price (strike price or K). The buyer pays a price for this right.

- At expiration,
 - If the value of the underlying asset (S) $>$ Strike Price(K) Buyer makes the difference: $S - K$
 - If the value of the underlying asset (S) $<$ Strike Price (K) Buyer does not exercise
- More generally,
 - the value of a call increases as the value of the underlying asset increases
 - the value of a call decreases as the value of the underlying asset decreases

European Options vs. American Options

An American option can be exercised at any time prior to its expiration, while a European option can be exercised only at expiration.

- The possibility of early exercise makes American options more valuable than otherwise similar European options.

Early exercise is preferred in many cases, e.g.,

- when the underlying asset pays large dividends.
- when an investor holds both the underlying asset and deep in-the-money puts on that asset, at a time when interest rates are high.

Valuing European Call Options

Variables

- Strike Price: K
- Time till Expiration: T
- Price of underlying asset: S
- Volatility: σ

Valuing European options involves solving a stochastic calculus equation, e.g., the Black-Scholes model.

In the simplest case, the option is priced as a conditional expectation relating to an exponentiated normal distribution:

$$\begin{aligned}\text{Option Price} &= \mathbf{E}[(S_T - K)\mathbf{1}_{S_T \geq K}] \\ &= \mathbf{E}[(S_T - K) \mid S_T \geq K] \text{Prob}(S_T \geq K),\end{aligned}$$

where

$$\log \frac{S_T}{S_0} \sim \mathcal{N}(0, \sigma\sqrt{T}).$$

Valuing American Call Options

Variables

- Strike Price: K
- Time till Expiration: T
- Price of underlying asset: S
- Volatility, Dividends, etc.

Valuing American options requires the solution of an **optimal stopping problem**:

$$\text{Option Price} = S(t^*) - K,$$

where

t^* = optimal exercising time.

If the option writers do not solve t^* correctly, the option buyers will have an arbitrage opportunity to exploit the option writers.

DP Formulation

- Dynamics of underlying asset: for example, exponentiated Browning motion

$$S_{t+1} = f(S_t, w_t)$$

- state: S_t , price of the underlying asset
- control: $u_t \in \{\text{Exercise, Hold}\}$
- transition cost: $g_t = 0$

Bellman Equation

When $t = T$, $V_t(S_T) = \max\{S_T - K, 0\}$, and when $t < T$

$$V_t(S_t) = \max\{S_t - K, \mathbf{E}[V_{t+1}(S_{t+1})]\},$$

where the optimal cost vector $V_t(S)$ is the option price at the t th day when the current stock price is S .

A Simple Binomial Model

We focus on American call options.

- Strike price: K
- Duration: T days
- Stock price of t th day: S_t
- Growth rate: $u \in (1, \infty)$
- Diminish rate: $d \in (0, 1)$
- Probability of growth: $p \in [0, 1]$

Binomial Model of Stock Price

$$S_{t+1} = \begin{cases} uS_t & \text{with probability } p \\ dS_t & \text{with probability } 1-p \end{cases}$$

As the discretization of time becomes finer, the binomial model approaches the Brownian motion model.

DP Formulation for Binomial Model

- Given S_0, T, K, u, r, p .
- State: S_t , finite number of possible values
- Cost vector: $V_t(S)$, the value of option at the t th day when the current stock price is S .

Bellman equation for binomial option

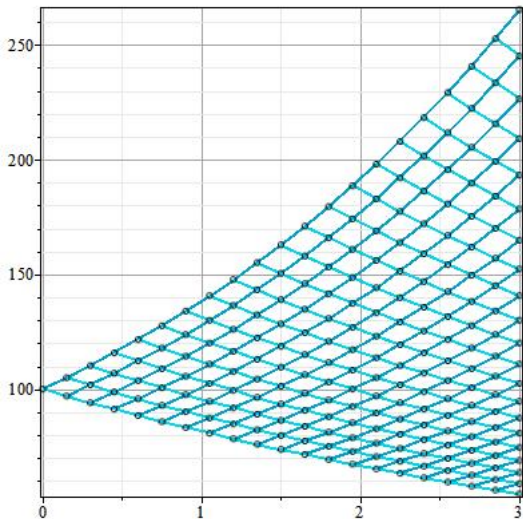
$$V_t(S_t) = \max \left\{ S_t - K, \quad pV_{t+1}(uS_t) + (1 - p)V_{t+1}(dS_t) \right\},$$
$$V_t(S_T) = \max \{ S_T - K, 0 \}.$$

Use Exact DP to Evaluate Options

Exercise 1

Use exact dynamic programming to price an American call option. The program should be a function of S_0, T, p, u, d, K .

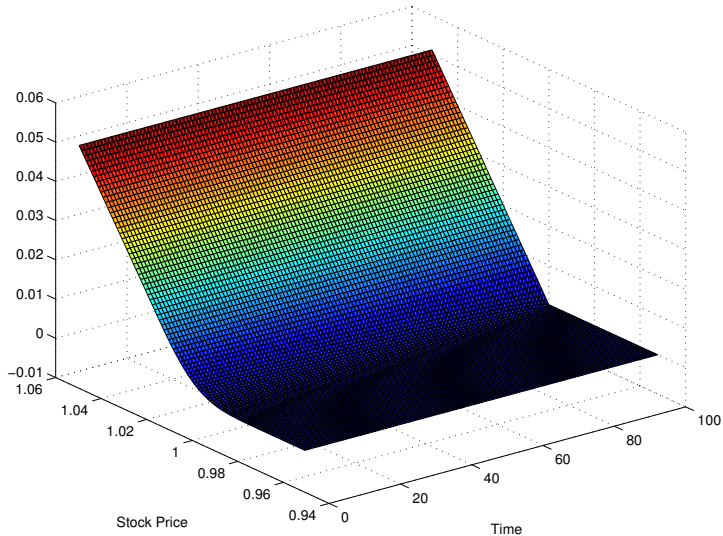
Algorithm Structure: Binomial Tree



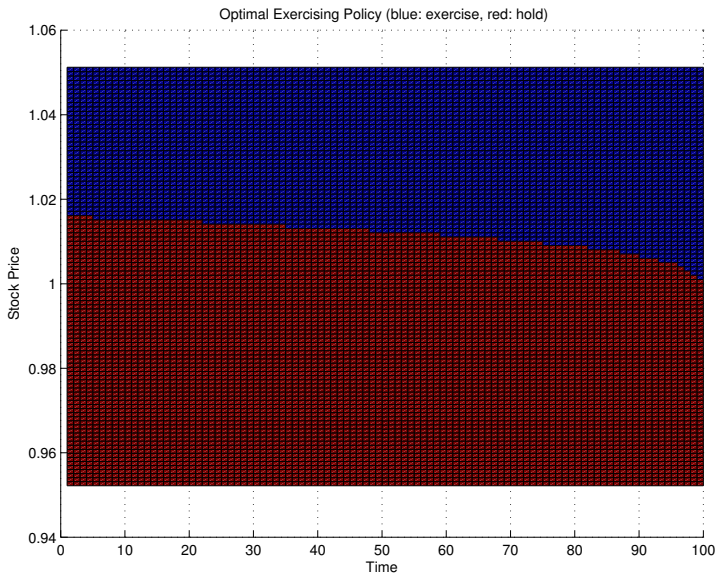
DP algorithm is backward induction on the binomial tree

Computation Results - Option Prices

Optimal Value Function (Option Price at given time and stock price)



Computation Results - Exercising strategy



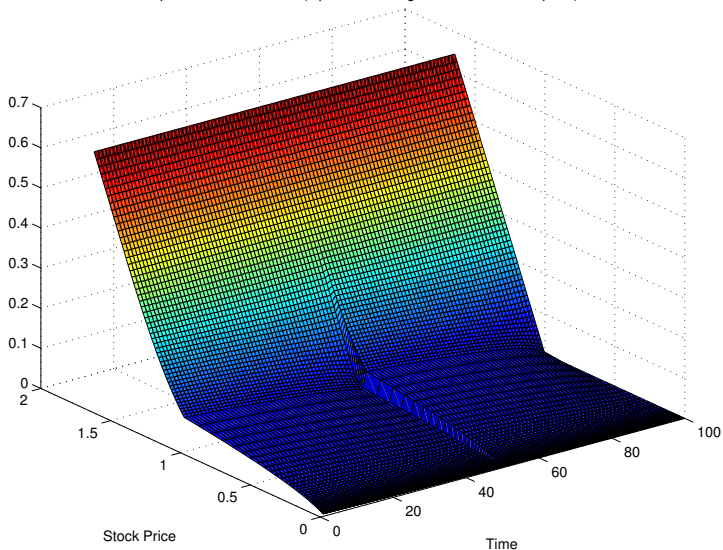
Exercise 2

Option with Dividend

Assume that at time $t = T/2$, the stock will yield a dividend to its shareholders. As a result, the stock price will decrease by $d\%$ at time t . Use this information to modify the program and price the option.

Option prices when there is dividend

Optimal Value Function (Option Price at given time and stock price)



Exercising strategy when there is dividend

