IEOR E4007: Optimization Models and Methods

Dynamic Programming

Garud Iyengar

Columbia University

Industrial Engineering and Operations Research

Dynamic programming

- Simple example: Capital Budgeting
- Characteristics of dynamic programming
- Integer knapsack and its variants
- Shortest path problems and linear programming
- Stochastic dynamic programs
    - American option pricing
    - Trade execution
- Approximate dynamic programming

Capital budgeting problem

- Total budget for expansion: \$5 million
- Three (3) plants with the following proposals:

	Plant 1		Plant 2		Plant 3	
Proposal	c_1	r_1	c_2	r_2	c_3	r_3
1	0	0	0	0	0	0
2	1	5	2	8	1	4
3	2	6	3	9	–	–
4	–	–	4	12	–	–

- At most 1 proposal from each plant.

$$
\begin{aligned}
\text{max} \quad & (5x_{12} + 6x_{13}) + (8x_{22} + 9x_{23} + 12x_{24}) + (4x_{32}) \\
\text{s.t.} \quad & (x_{12} + 2x_{13}) + (2x_{22} + 3x_{23} + 4x_{24}) + (x_{32}) \leq 5, \\
& x_{12} + x_{13} \leq 1, \\
& x_{22} + x_{23} + x_{24} \leq 1, \\
& x_{ij} \in \{0, 1\}
\end{aligned}
$$

Stages, states and recursion

- Solve problem in 3 stages: in stage i allocate plants $j \leq i$
- $V_i(s)$ = optimal revenue using capital s in stage i
  - Recursively define $V_i(\cdot)$ in terms of $V_{i-1}(\cdot)$
  - Computing $V_1(\cdot)$ is easy!
- **Note**: Have to compute $V_i(s)$ for all possible values of s.

First stage solution

Stage 1: $V_1(s) = \max\{r_{1k} : c_{1k} \leq s\}$

s	$V_1(s)$	optimum k_1^*
0	0	1
1	5	2
2	6	3
3	6	3
4	6	3
5	6	3

Recursion

Stage 2: Bellman recursion

$$
\begin{aligned}
V_2(s) &= \max\{r_{1k} + r_{2l} : c_{1k} + c_{2l} \le s\} \\
&= \max\left\{r_{2l} + V_1(s - c_{2l}) : c_{2l} \le s\right\}
\end{aligned}
$$

For $s = 4$
$$
\begin{aligned}
V_2(4) &= \max\{0 + V_1(4), 8 + V_1(2), 9 + V_1(1), 12 + V_1(0)\} \\
&= \max\{6, 8 + 6, 9 + 5, 12\} = 14
\end{aligned}
$$

s	$V_2(s)$	optimum k_2^*
0	0	1
1	5	1
2	8	2
3	13	2
4	14	2/3
5	17	4

Recursion

Stage 3: Bellman recursion

$$V_3(5) = \max\{r_{3l} + V_2(5 - c_{3l}) : c_{3l} \leq 5\} = \max\{0 + 17, 4 + 14\} = 18$$

Optimal solution: $(k_3^*, k_2^*, k_1^*) = (2, 2, 3)$

- Optimal solution may not be unique!

Recursion

Stage 3: Bellman recursion

$$V_3(5) = \max\{r_{3l} + V_2(5 - c_{3l}) : c_{3l} \leq 5\} = \max\{0 + 17, 4 + 14\} = 18$$

Optimal solution: $(k_3^*, k_2^*, k_1^*) = (2, 2, 3)$

- Optimal solution may not be unique!

General recursion

$$\begin{array}{rcll} V_i(s) & = & \text{max} & \sum_{j=1}^{i} r(x_j) \\ & & \text{s.t.} & \sum_{j=1}^{i} c(x_j) \leq s \end{array}$$

Recursion

Stage 3: Bellman recursion

$$V_3(5) = \max\{r_{3l} + V_2(5 - c_{3l}) : c_{3l} \leq 5\} = \max\{0 + 17, 4 + 14\} = 18$$

Optimal solution: $(k_3^*, k_2^*, k_1^*) = (2, 2, 3)$

- Optimal solution may not be unique!

General recursion

$$\begin{aligned} V_i(s) &= \begin{array}{ll} \text{max} & \sum_{j=1}^{i} r(x_j) \\ \text{s.t.} & \sum_{j=1}^{i} c(x_j) \leq s \end{array} \\ &= \max_{x_i} \left\{ r(x_i) + \begin{array}{ll} \text{max} & \sum_{j=1}^{i-1} r(x_j) \\ \text{s.t.} & \sum_{j=1}^{i-1} c(x_j) \leq s - c(x_i) \end{array} \right\} \end{aligned}$$

Recursion

Stage 3: Bellman recursion

$$V_3(5) = \max\{r_{3l} + V_2(5 - c_{3l}) : c_{3l} \le 5\} = \max\{0 + 17, 4 + 14\} = 18$$

Optimal solution: $(k_3^*, k_2^*, k_1^*) = (2, 2, 3)$

- Optimal solution may not be unique!

General recursion

$$\begin{aligned} V_i(s) &= \begin{array}{rl} \text{max} & \sum_{j=1}^{i} r(x_j) \\ \text{s.t.} & \sum_{j=1}^{i} c(x_j) \le s \end{array} \\ &= \max_{x_i} \left\{ r(x_i) + \begin{array}{rl} \text{max} & \sum_{j=1}^{i-1} r(x_j) \\ \text{s.t.} & \sum_{j=1}^{i-1} c(x_j) \le s - c(x_i) \end{array} \right\} \\ &= \max_{x_i} \{ r(x_i) + V_{i-1}(s - c(x_i)) \} \end{aligned}$$

Dynamic project selection

Suppose n_t projects are possible at time $t = 1, \ldots, T$

- Cost of project i available at time t: c_{it}
- NPV value at time t for project i: r_{it}
- Can initiate only one project at each time t

Interest rate r. Total budget W.

Value function

$$\begin{aligned} V_\ell(s) &= \text{maximum NPV from projects initiated at times} \\ &\quad\; t = \ell, \ldots, T\text{, using at most budget } s \end{aligned}$$

Dynamic project selection

Suppose n_t projects are possible at time $t = 1, \ldots, T$

- Cost of project i available at time t: c_{it}
- NPV value at time t for project i: r_{it}
- Can initiate only one project at each time t

Interest rate r. Total budget W.

Value function

$$V_\ell(s) \quad = \quad \text{maximum NPV from projects initiated at times } t = \ell, \ldots, T, \text{ using at most budget } s$$

Recursion

$$V_\ell(s) \quad = \quad \max_{\{i : c_{i\ell} \le s\}} \left[r_{i\ell} + \tfrac{1}{1+r} V_{l+1}\Big((1+r)(s - c_{i\ell}) \Big) \right]$$

How does one solve this recursion?

Dynamic project selection

Suppose n_t projects are possible at time $t = 1, \ldots, T$

- Cost of project i available at time t: c_{it}
- NPV value at time t for project i: r_{it}
- Can initiate only one project at each time t

Interest rate r. Total budget W.

Value function

$$V_\ell(s) = \text{maximum NPV from projects initiated at times } t = \ell, \ldots, T, \text{ using at most budget } s$$

Recursion

$$V_\ell(s) = \max_{\{i : c_{i\ell} \leq s\}} \left[r_{i\ell} + \tfrac{1}{1+r} V_{l+1}\Big((1+r)(s - c_{i\ell}) \Big) \right]$$

How does one solve this recursion?

Start from $V_T(s)$ and work backwards to $V_1(W)$

Characteristics of dynamic programming

- Decision can be divided into stages
- Each stage has a number of states
  - state = complete information needed to solve the problem
  - given state, solution in subsequent/previous stages not required
- Each state has a number of available actions
  - Action a results in a reward/cost $r(s, a)$
  - Action a results in a state transition $\tilde{s}|a$
- $V_t(s)$ = optimal value in state s in stage t
- Use a recursive relation to compute V_t for all t

$$V_t(s) = \max_a \{r(s, a) + V_{t+1}(\tilde{s} \mid a)\}$$

Integer knapsack problem

- Optimization problem: v_j, w_j: integers

$$\begin{array}{rl} \text{max} & \sum_{j=1}^{N} v_j x_j \\ \text{s.t.} & \sum_{j=1}^{N} w_j x_j \leq W \\ & x_j \in \mathbf{Z}_+ \end{array}$$

- stages $i = 1, \ldots, N$: compute optimal solution for $j \leq i$

- state s $(\leq W)$: weight available for the objects $j \leq i$

- recursion

$$V_i(s) = \max\{V_{i-1}(s), V_i(s - w_i) + v_i\}$$

  - Can you make sense of this recursion?
  - Recursion relates V_i to itself. Does this make sense?

More on integer knapsack

What are the stages, states, and recursion for following?

$$
\begin{aligned}
\max \quad & \textstyle\sum_{i=1}^{n} U(x_i) \\
\text{s.t.} \quad & \textstyle\sum_{i=1}^{n} w_i x_i \leq W \\
& x \in \mathbb{Z}_+^n.
\end{aligned}
$$

More on integer knapsack

What are the stages, states, and recursion for following?

$$\begin{array}{rl} \text{max} & \sum_{i=1}^{n} U(x_i) \\ \text{s.t.} & \sum_{i=1}^{n} w_i x_i \le W \\ & x \in \mathbb{Z}_+^n. \end{array}$$

Stages: $i = 1, \ldots, n$

States: $s =$ budget for stages $j \le i$

$$V_i(s) = \max\{U(x_i) + V_{i-1}(s - w_i x_i)\}$$

More on integer knapsack

What are the stages, states, and recursion for following?

$$\begin{array}{rl} \text{max} & \sum_{i=1}^n U(x_i) \\ \text{s.t.} & \sum_{i=1}^n w_i x_i \le W \\ & x \in \mathbb{Z}_+^n. \end{array}$$

Stages: $i = 1, \ldots, n$

States: $s =$ budget for stages $j \le i$

$$V_i(s) = \max\{U(x_i) + V_{i-1}(s - w_i x_i)\}$$

$$\begin{array}{rl} \text{max} & \sum_{i=1}^n U(x_i) \\ \text{s.t.} & \sum_{i=1}^n w_i x_i \le W \\ & \sum_{i=1}^n v_i x_i \le V \\ & x \in \mathbb{Z}_+^n. \end{array}$$

More on integer knapsack

What are the stages, states, and recursion for following?

$$\begin{array}{rl} \text{max} & \sum_{i=1}^n U(x_i) \\ \text{s.t.} & \sum_{i=1}^n w_i x_i \le W \\ & x \in \mathbb{Z}_+^n. \end{array}$$

Stages: $i = 1, \ldots, n$

States: $s =$ budget for stages $j \le i$

$$V_i(s) = \max\{U(x_i) + V_{i-1}(s - w_i x_i)\}$$

$$\begin{array}{rl} \text{max} & \sum_{i=1}^n U(x_i) \\ \text{s.t.} & \sum_{i=1}^n w_i x_i \le W \\ & \sum_{i=1}^n v_i x_i \le V \\ & x \in \mathbb{Z}_+^n. \end{array}$$

Stages: $i = 1, \ldots, n$

States: $(s_w, s_v) =$ budgets for stages $j \le i$

$$\begin{aligned} V_i(s_w, s_v) \quad = \quad & \max\{U(x_i) + \\ & V_{i-1}(s_w - w_i x_i, s_v - v_i x_i)\} \end{aligned}$$

More on integer knapsack

$$
\begin{array}{rl}
\text{max} & \sum_{i=1}^{n-1} U(x_i, x_{i+1}) \\
\text{s.t.} & \sum_{i=1}^{n} w_i x_i \le W \\
& x \in \mathbb{Z}_+^n.
\end{array}
$$

More on integer knapsack

$$\begin{array}{rl} \text{max} & \sum_{i=1}^{n-1} U(x_i, x_{i+1}) \\ \text{s.t.} & \sum_{i=1}^{n} w_i x_i \le W \\ & x \in \mathbb{Z}_+^n. \end{array}$$

$$\begin{array}{rcl} V_i(x,s) & = & \text{Optimal utility from stages } j < i \text{ when} \\ & & \text{budget consumed is } s \text{ and } x_i = x \\ & = & \begin{array}{rl} \text{max} & \sum_{j=1}^{i-1} U(x_j, x_{j+1}) \\ \text{s.t.} & \sum_{j=1}^{i-1} w_j x_j \le s \\ & x_i = x \end{array} \\ & = & \max_{x_{i-1}} \left\{ U(x_{i-1}, x) + V_{i-1}\big(x_{i-1}, s - w_{i-1}x_{i-1}\big) \right\} \end{array}$$

How does one initialize this recursion? How does it end?

More on integer knapsack

$$\begin{array}{rl} \text{max} & \sum_{i=1}^{n-1} U(x_i, x_{i+1}) \\ \text{s.t.} & \sum_{i=1}^{n} w_i x_i \leq W \\ & x \in \mathbb{Z}_+^n. \end{array}$$

$$\begin{array}{rcl} V_i(x,s) & = & \text{Optimal utility from stages } j < i \text{ when} \\ & & \text{budget consumed is } s \text{ and } x_i = x \\ & = & \begin{array}{rl} \text{max} & \sum_{j=1}^{i-1} U(x_j, x_{j+1}) \\ \text{s.t.} & \sum_{j=1}^{i-1} w_j x_j \leq s \\ & x_i = x \end{array} \\ & = & \max_{x_{i-1}} \Big\{ U(x_{i-1}, x) + V_{i-1}\big(x_{i-1}, s - w_{i-1} x_{i-1}\big) \Big\} \end{array}$$

How does one initialize this recursion? How does it end?

- Recursion starts with $V_2(x,s)$.
- At the end of the recursion we need to compute $\max_x \{V_N(x, W)\}$

Shortest path problem

N nodes: c_{ij} = "length" of arc from node i to node j

Goal: Compute the shortest path from node s to all other nodes

path = sequence of nodes $s \to v_1 \to \ldots \to \ell \to j$ such that (v_i, v_{i+1}) is an edge in graph and no vertices are repeated.

- stage k: shortest paths with k or fewer edges
  - what is the largest k that one needs to consider ?

Shortest path problem

N nodes: c_{ij} = "length" of arc from node i to node j

Goal: Compute the shortest path from node s to all other nodes

path = sequence of nodes $s \to v_1 \to \ldots \to \ell \to j$ such that (v_i, v_{i+1}) is an edge in graph and no vertices are repeated.

- stage k: shortest paths with k or fewer edges
  - what is the largest k that one needs to consider ? N-1

Shortest path problem

N nodes: c_{ij} = "length" of arc from node i to node j

Goal: Compute the shortest path from node s to all other nodes

path = sequence of nodes $s \to v_1 \to \ldots \to \ell \to j$ such that (v_i, v_{i+1}) is an edge in graph and no vertices are repeated.

- stage k: shortest paths with k or fewer edges
  - what is the largest k that one needs to consider ? N-1
- state $j = 1, \ldots, N$
- $V_k(j)$ = length of shortest path from s to j using k or fewer edges

Shortest path problem

N nodes: c_{ij} = "length" of arc from node i to node j

Goal: Compute the shortest path from node s to all other nodes

path = sequence of nodes $s \to v_1 \to \ldots \to \ell \to j$ such that (v_i, v_{i+1}) is an edge in graph and no vertices are repeated.

- stage k: shortest paths with k or fewer edges
  - what is the largest k that one needs to consider ? N-1
- state $j = 1, \ldots, N$
- $V_k(j)$ = length of shortest path from s to j using k or fewer edges
- Recursion: Suppose the shortest path from $s \to v_1 \to \ldots \to \ell \to j$. Then $s \to v_1 \to \ldots \to \ell$ must be the shortest path from s to ℓ.

$$V_k(j) = \min\{V_{k-1}(i) + c_{ij} : i = 1, \ldots, N\}, \quad c_{ii} = 0$$

Shortest paths (contd)

- Will the recursion always work?

Shortest paths (contd)

- Will the recursion always work?
  Recursion fails when there is a negative cost cycle!

Shortest paths (contd)

- Will the recursion always work?
  Recursion fails when there is a negative cost cycle!

- How can one detect a problem, if any ?

Shortest paths (contd)

- Will the recursion always work?
  Recursion fails when there is a negative cost cycle!

- How can one detect a problem, if any ?
  Can detect it when $V_N(j) < V_{N-1}(j)$ for some node j

Shortest paths (contd)

- Will the recursion always work?
  Recursion fails when there is a negative cost cycle!

- How can one detect a problem, if any ?
  Can detect it when $V_N(j) < V_{N-1}(j)$ for some node j

- Possible to compute shortest path using DP when there are negative cycles?

Shortest paths (contd)

- Will the recursion always work?
  Recursion fails when there is a negative cost cycle!

- How can one detect a problem, if any ?
  Can detect it when $V_N(j) < V_{N-1}(j)$ for some node j

- Possible to compute shortest path using DP when there are negative cycles?

MATLAB code `shortestpath.m`

Maximize the minimum height

Consider a graph with n nodes and edge set $\mathcal{E}$

For $(i, j) \in \mathcal{E}$

- $h_{ij} =$ minimum height of bridges on the direct edge from i to j.

Goal: Find a path from node s to all other nodes that maximizes the minimum height of the bridge along the path.

$$V_k(t) \quad = \quad \text{the height along the path that maximizes the height among all } s - t \text{ paths using } k \text{ or fewer edges}$$

Recursion

$$V_k(t) \quad = \quad \max_{1 \leq j \leq n} \Big\{ \min\{V_{k-1}(j), h_{jt}\} \Big\}$$

where $h_{ij} = -\infty$ when $(i, j) \notin \mathcal{E}$, and $h_{jj} = \infty$ for all $j = 1, \ldots, n$

Dynamic programs are extremal path problems

Consider a minimization dynamic program with initial state $s_0 = s$

$$V_t(s_t) = \min_a \{c(s_t, a) + V_{t+1}(\tilde{s}|a)\}$$

Construct a graph as follows.

- For time $t = 0$, add one node $(0, s_0)$
- For time $t = 1, \ldots, T$, and states s_t at time t add a node (t, s_t).
- Suppose there is an action a that takes state (t, u) to the state $(t + 1, v)$
  - Insert a directed edge from (t, u) to $(t + 1, v)$
  - Assign the cost $c_t(u, a)$ to this edge
  - More than one actions taking state (t, u) to the state $(t + 1, v)$?
- Compute optimal path from $(0, s_0)$ to all nodes at time T

Capital Budgeting problem

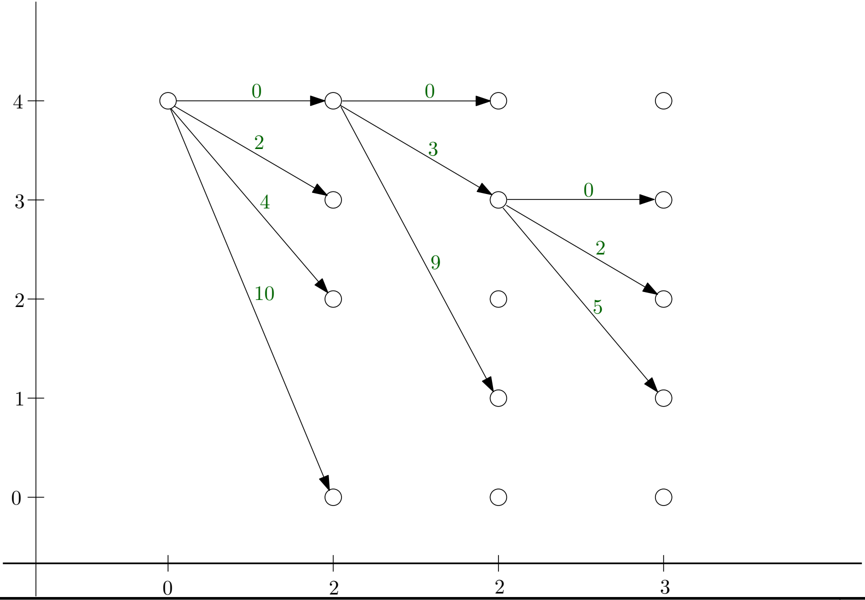
Budget $W = 4$

Three (3) plants with the following projects

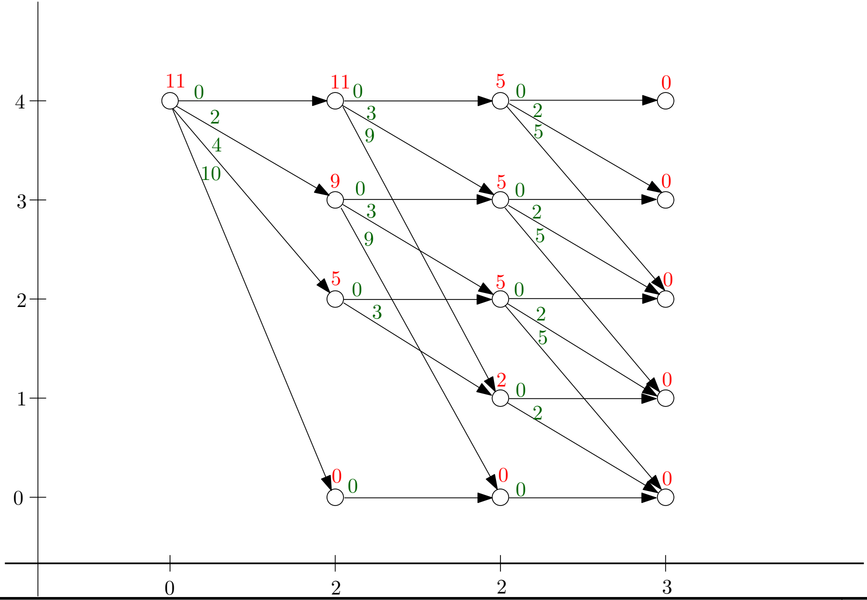
	Plant 1		Plant 2		Plant 3	
Projects	c_1	r_1	c_2	r_2	c_3	r_3
1	0	0	0	0	0	0
2	1	2	1	3	1	2
3	2	4	3	9	2	5
4	4	10	–	–	–	–

As before we must implement exactly one project from each plant.

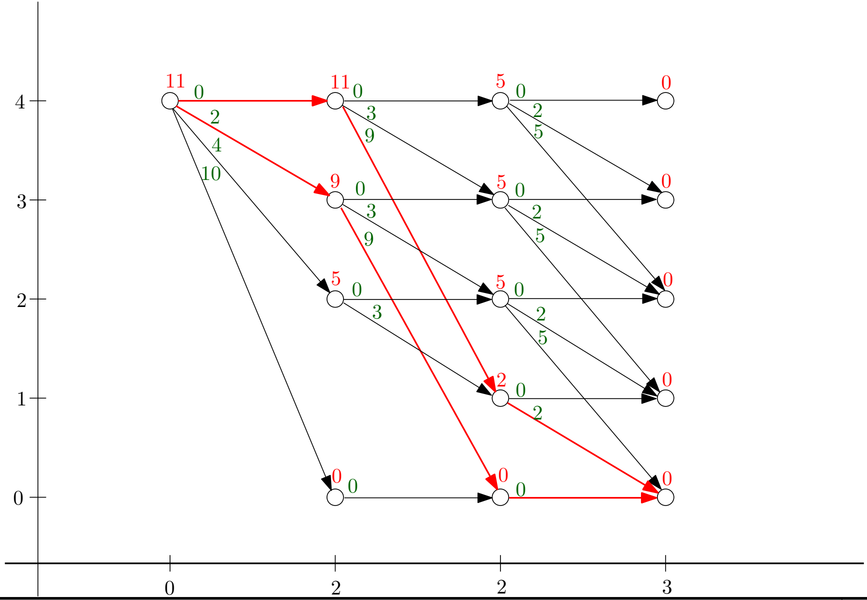
Graphical representation for capital budgeting



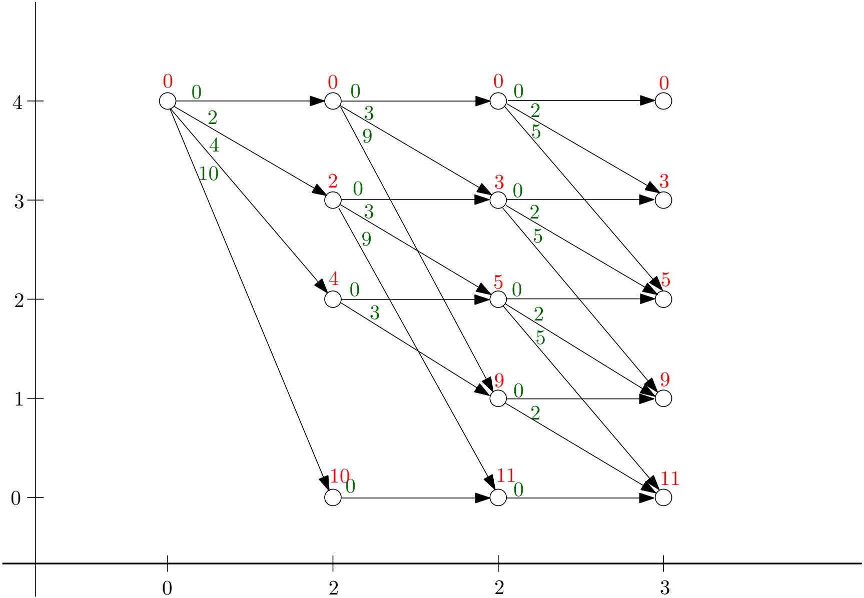
Graphical representation for capital budgeting



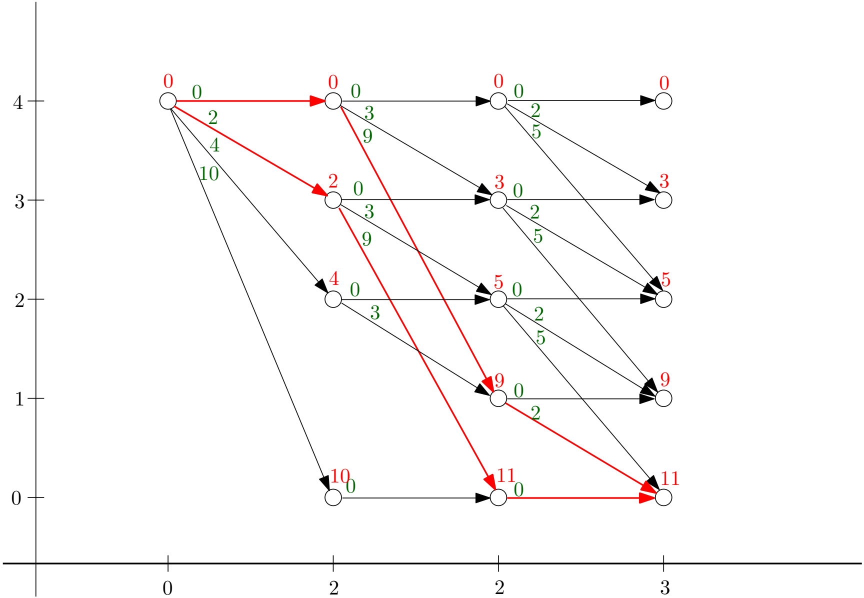
Graphical representation for capital budgeting



Another recursion



Another recursion



Max-min problem

Consider the following optimization

$$
\begin{array}{rl}
\max_x & \min_{1 \leq i \leq n} U_i(x_i), \\
\text{s.t.} & \sum_{i=1}^n x_i \leq W, \\
& x_i \geq 0, \quad i = 1, \ldots, i
\end{array}
$$

- Stage i: Compute optimal value of x_i
- State s: Budget available for the subproblem
- Value function

$$
\begin{array}{rrl}
V_i(s) = & \max_x & \min_{1 \leq j \leq i} U_j(x_j), \\
& \text{s.t.} & \sum_{j=1}^i x_j \leq s, \\
& & x_j \geq 0, \quad j = 1, \ldots, i
\end{array}
$$

Max-min problem (contd)

"Splitting" the problem suggests the following recursion

$$
\begin{aligned}
V_i(s) &= \max_{x_i \geq 0} \min \left\{ U_i(x_i),\ \begin{array}{rl} \max_x & \min_{1 \leq j \leq i-1} U_j(x_j) \\ \text{s.t.} & \sum_{j=1}^{i-1} x_j \leq s - x_i, \\ & x_j \geq 0, \quad j = 1, \ldots, i-1 \end{array} \right\} \\
&= \max_{x_i \geq 0} \min \left\{ U_i(x_i), V_{i-1}(s - x_i) \right\} \\
\pi_i(s) &= \operatorname{argmax}_{x_i \geq 0} \min \left\{ U_i(x_i), V_{i-1}(s - x_i) \right\}
\end{aligned}
$$

This recursion is initiated by computing $V_1(s) = \max_{x_1 \geq 0} U_1(x_1)$

Optimal solution of the max-min utility maximization problem:

$$
\begin{array}{lll}
x_n^* = \pi_n(W) & s_n = W - x_n^* & \\
x_i^* = \pi_i(s_{i+1}) & s_i = s_{i+1} - x_i^* & i = n-1, \ldots, 1.
\end{array}
$$

Shortest paths and linear programming

- l_i = shortest path from s to i. Then

$$\begin{aligned} l_i &= \min_{j=1,\ldots,N}\{l_j + c_{ji}\} \\ &\le l_j + c_{ji}, \quad \forall i, j \end{aligned}$$

- Then l_t is the solution of the linear program

$$\begin{array}{rl} \max & l_t \\ \text{s.t.} & l_i \le l_j + c_{ji}, \quad i, j = 1, \ldots, N \\ & l_s = 0 \end{array}$$

  This LP has n variables and n^2 variables.

- Any deterministic dynamic program has a linear programming representation

$$\begin{array}{rl} \max & \sum_{s_T} \pi(t, s_T)\ell(T, s_T) \\ \text{subject to} & \ell(t, s) \le c_a + \ell(t+1, \tilde{s} \mid a), \quad \forall t, a \end{array}$$

  $\pi(T, s_T) > 0$ for all nodes s_T at time T

Dual linear programs

- Primal linear program

$$
\begin{array}{rl}
\max & l_t \\
\text{s.t.} & l_i \le l_j + c_{ji}, \quad i, j = 1, \ldots, N \\
& l_s = 0
\end{array}
$$

- The dual of this linear program is given by

$$
\begin{array}{rl}
\min & \sum_{i,j=1}^N f_{ij} c_{ij}, \\
\text{s.t.} & \sum_{j=1}^N f_{ji} - \sum_{k=1}^N f_{ik} = 0, \quad \forall i \neq s, t \\
& \sum_{j=1}^N f_{jt} - \sum_{k=1}^N f_{tk} = 1 \\
& f_{ij} \ge 0
\end{array}
$$

  - Inflow $=$ 1 in node t
  - Inflow $=$ Outflow at all nodes $i \neq s, t$
  - Therefore outflow $=$ 1 at node s

- The dual problem is a min-cost flow problem
  - dual optimal solutions give optimal actions at the various nodes.

Stochastic dynamic programming

- State $\tilde{s}_{t+1}$ resulting from action a in state s at time t is random.
- $\mathbb{P}_t(\cdot|s,a)$: distribution of $\tilde{s}_{t+1}$ as a function of (s,a)
- $V_t(s) \equiv$ maximum achievable reward starting from state s at time t
- Bellman recursion
$$V_t(s) = \max_{a \in \mathcal{A}_t(s)} \left\{ r_t(s,a) + \beta \mathbb{E}\left[V_{t+1}(\tilde{s}_{t+1} \mid s,a \right] \right\}$$
where $\beta =$ discount factor
- Optimal policy: mapping from state s to the optimal action a^*

Binomial-tree model for asset price

- T period market
- Two assets
  - cash with interest rate r
  - stock dynamics

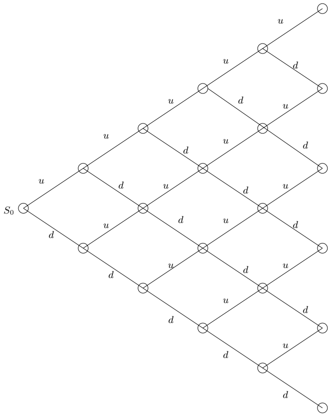
$$S_{t+1} = \begin{cases} uS_t & \text{prob } p \\ dS_t & \text{prob } q = 1 - p. \end{cases}$$

- Martingale measure

$$pu+(1-p)d = e^r \quad \Rightarrow \quad p = \frac{e^r - d}{u - d}$$

- Stages: $t = 0, 1, \ldots, T$
- States in stage t:

$$\mathcal{B}_t = \{S_0 d^i u^{t-i} : i = 0, \ldots, t\}$$



American option

- American put option with strike K and expritation T
  - right to a payoff $(K - S_\tau)^+$ for any $\tau \le T$

- Option price $P_t(S_t)$ as a function of stock price S_t

$$P_t(s) = \sup_{t \le \tau \le T} \{\mathbb{E}[e^{-r(\tau - t)}(K - S_\tau)^+ | S_t = s]\}$$

τ: stopping time

- Dynamic programming formulation
  - stages: $t = 0, \ldots, T$ and states: $\mathcal{B}_t \cup \chi \equiv$ stop state
  - actions, rewards and transitions
    - $s \in \mathcal{B}_t$: $a \in \{0, 1\}$, $r(s, 0) = (K - s)^+$, $\mathbb{P}(\chi | s, 0) = 1$; $r(s, 1) = 0$, $\mathbb{P}(uS | s, 1) = 1 - \mathbb{P}(dS | s, 1) = p$
    - χ: $a = 0$, $r(\chi, 0) = 0, \mathbb{P}(\chi | \chi, 0) = 1$

American Option Pricing (contd)

- $V_t(s)$ = price of American option in stage t and state s
- $V_t(\chi) = 0$ for all t
- Recursion

$$\begin{aligned} V_t(s) &= \max\left\{(K-s)^+, e^{-r}\mathbb{E}[V_{t+1} \mid S_t = s]\right\} \\ &= \max\left\{(K-s)^+, e^{-r}\left(pV_{t+1}(us) + (1-p)V_{t+1}(ds)\right)\right]\right\} \end{aligned}$$

- How does one start this recursion ?

Trade execution

- Have to liquidate N shares of a stock over T trading epochs.
- Stock price dynamics

$$S_{t+1} = S_t + \delta\tilde{\xi}_t - g(n_t)$$

  - S_t = price at the previous trading epoch
  - δ = price step and $\mathbb{P}(\xi = 1) = 1 - \mathbb{P}(\xi = -1) = \pi$
  - $g(\cdot)$ = permanent price impact function.

Trade execution

- Have to liquidate N shares of a stock over T trading epochs.
- Stock price dynamics

$$S_{t+1} = S_t + \delta\tilde{\xi}_t - g(n_t)$$

  - S_t = price at the previous trading epoch
  - δ = price step and $\mathbb{P}(\xi = 1) = 1 - \mathbb{P}(\xi = -1) = \pi$
  - $g(\cdot)$ = permanent price impact function.

- (Random) Revenue from selling n_t shares at time t

$$\tilde{r}(n_t) = (S_t - h(n_t)) \cdot n_t - \lambda x_t^{\beta}$$

  - $h(\cdot)$ = temp price impact function. only affect the revenue in time t.
  - x_t = inventory (unsold shares) at time t
  - λx_t^{β}: inventory cost $\equiv$ "delay" cost $\equiv$ "variance" of revenue

Trade execution

- Have to liquidate N shares of a stock over T trading epochs.
- Stock price dynamics

$$S_{t+1} = S_t + \delta\tilde{\xi}_t - g(n_t)$$

  - S_t = price at the previous trading epoch
  - δ = price step and $\mathbb{P}(\xi = 1) = 1 - \mathbb{P}(\xi = -1) = \pi$
  - $g(\cdot)$ = permanent price impact function.
- (Random) Revenue from selling n_t shares at time t

$$\tilde{r}(n_t) = (S_t - h(n_t)) \cdot n_t - \lambda x_t^\beta$$

  - $h(\cdot)$ = temp price impact function. only affect the revenue in time t.
  - x_t = inventory (unsold shares) at time t
  - λx_t^β: inventory cost $\equiv$ "delay" cost $\equiv$ "variance" of revenue
- Optimization problem:

$$\max_{\{n: n=(n_0,\ldots,n_{T-1}),\text{causal}\}} \left\{ \mathbb{E}\left[\sum_{t=1}^{T} \tilde{r}_t(n_t) \right] \right\}$$

  - causal $\equiv$ n_t is only a function of information at time t

Trade execution: linear permanent price impact

- State at time t
  - $x =$ unsold inventory
  - $s = \sum_{j=0}^{t-1} \mathbf{1}(\xi_j = 1) =$ cumulative state of the random walk

Trade execution: linear permanent price impact

- State at time t
  - $x =$ unsold inventory
  - $s = \sum_{j=0}^{t-1} \mathbf{1}(\xi_j = 1) =$ cumulative state of the random walk
- Action $A(x, s) = \{a : 0 \leq a \leq x\}$

Trade execution: linear permanent price impact

- State at time t
  - x = unsold inventory
  - $s = \sum_{j=0}^{t-1} \mathbf{1}(\xi_j = 1)$ = cumulative state of the random walk
- Action $A(x, s) = \{a : 0 \leq a \leq x\}$
- Price S_t as a function of state (x, s)

$$
\begin{aligned}
S_t &= S_0 + \delta \sum_{\tau=0}^{t-1} \xi_\tau - \sum_{\tau=0}^{t-1} g(n_\tau) \\
&= S_0 + \delta\big(s - (t - s)\big) - g\Big(\sum_{\tau=0}^{t-1} n_\tau\Big) \\
&= S_0 + \delta(2\delta s - t) - g(N - x)
\end{aligned}
$$

Trade execution: linear permanent price impact

- State at time t
  - x = unsold inventory
  - $s = \sum_{j=0}^{t-1} \mathbf{1}(\xi_j = 1)$ = cumulative state of the random walk
- Action $A(x, s) = \{a : 0 \leq a \leq x\}$
- Price S_t as a function of state (x, s)

$$\begin{aligned} S_t &= S_0 + \delta \sum_{\tau=0}^{t-1} \xi_\tau - \sum_{\tau=0}^{t-1} g(n_\tau) \\ &= S_0 + \delta\big(s - (t - s)\big) - g\Big(\sum_{\tau=0}^{t-1} n_\tau \Big) \\ &= S_0 + \delta(2\delta s - t) - g(N - x) \end{aligned}$$

- Revenue function

$$\begin{aligned} r(x, s, a) &= \big(S_t - h(a)\big) \cdot a - \lambda x^\beta \\ &= \Big(S_0 + \delta(2\delta s - t) - g(N - x) - h(a)\Big) \cdot a - \lambda x^\beta \end{aligned}$$

Trade Execution (contd.)

- State transition

$$(x_{t+1}, s_{t+1}) = \begin{cases} (x - a, s + 1), & \pi, \\ (x - a, s + 0), & 1 - \pi. \end{cases}$$

Trade Execution (contd.)

- State transition

$$(x_{t+1}, s_{t+1}) = \begin{cases} (x-a, s+1), & \pi, \\ (x-a, s+0), & 1-\pi. \end{cases}$$

- Recursion

$$V_t(x,s) = \max_{0 \le a \le x} \left\{ r(x,s,a) + \mathbb{E}\big[V_{t+1}(x-a, \tilde{s})\big] \right\}$$

Trade Execution (contd.)

- State transition

$$(x_{t+1}, s_{t+1}) = \begin{cases} (x - a, s + 1), & \pi, \\ (x - a, s + 0), & 1 - \pi. \end{cases}$$

- Recursion

$$V_t(x, s) = \max_{0 \le a \le x} \{ r(x, s, a) + \mathbb{E}[V_{t+1}(x - a, \tilde{s})] \}$$

- Optimal trade in state (x, s)

$$a_t^*(s, a) = \operatorname*{argmax}_{0 \le a \le x} \{ r(x, s, a) + \mathbb{E}[V_{t+1}(x - a, \tilde{s})] \}$$

- Code in MATLAB file `binimpact.m`

Utility maximization in a binomial tree

Market

- T period market
- Two assets
  - cash with interest rate r
  - stock dynamics

$$S_{t+1} = \begin{cases} uS_t & \text{prob } \pi \\ dS_t & \text{prob } 1-\pi \end{cases}$$

- Equivalent Martingale measure $q = \frac{e^r - d}{u-d}$

Optimization problem: Concave, non-decreasing utility function U

$$\begin{array}{rl} \max & \mathbb{E}[U(\tilde{w}_T)] \\ \text{s.t.} & \tilde{w}_T \text{ achievable using a self-financing} \\ & \text{trading strategy using initial wealth } w \end{array}$$

Dynamic programming approach

- Need a state (s, w): $s =$ stock price and $w =$ current wealth
- w continuous ... dynamic programming does not work.

Utility maximization (contd)

Define

$$V_t(s,w) \quad = \quad \text{Maximum terminal utility achievable with initial wealth } w \text{ when stock price is } s \text{ at time } t$$

Actions available in state (s,w)

$$\mathcal{A} = \Big\{(1-\phi,\phi) : \phi = \text{proportion invested in risky asset}\Big\}$$

Recursion

$$\begin{aligned} V_t(s,w) \quad &= \quad \max_{\phi\in\mathbb{R}} \Big\{\pi V_{t+1}\big(us, w\phi u + w(1-\phi)e^r\big) \\ &\qquad + (1-\pi)V_{t+1}\big(ds, w\phi d + w(1-\phi)e^r\big)\Big\} \end{aligned}$$

Utility maximization (contd)

Fix $\gamma > 0$ and $\gamma \neq 1$. Define the utility

$$U(w) = \begin{cases} \frac{w^{(1-\gamma)}}{1-\gamma} & w \geq 0 \\ -\infty & w < 0 \end{cases}$$

Suppose only long positions are allowed on each of the assets. Then the we are guaranteed that the wealth in any state is non-negative.

Then $V_t(s, w) = w^{1-\gamma} V_t(s, 1)$

- $V_T(s, w) = U(w) = w^{(1-\gamma)} U(1)$
- Suppose the statement is true for all $t \geq \tau + 1$. Then

$$\begin{aligned} V_\tau(s, w) &= \max_{\phi \in [0,1]} \Big\{ \pi V_{\tau+1}\big(us, w\phi u + w(1-\phi)e^r\big) \\ &\qquad + (1-\pi) V_{\tau+1}\big(ds, w\phi d + w(1-\phi)e^r\big) \Big\} \end{aligned}$$

Utility maximization (contd)

- Using the induction hypothesis, we get

$$\begin{aligned} V_\tau(s,w) &= \max_{\phi\in[0,1]}\Big\{w^{(1-\gamma)}\pi(\phi u+(1-\phi)e^r)^{(1-\gamma)}V_{\tau+1}(us,1) \\ &\qquad + w^{(1-\gamma)}(1-\pi)(\phi d+(1-\phi)e^r)^{(1-\gamma)}V_{\tau+1}(ds,1)\Big\} \\ &= w^{(1-\gamma)}\max_{\phi\in[0,1]}\Big\{\pi(\phi u+(1-\phi)e^r)^{(1-\gamma)}V_{\tau+1}(us,1) \\ &\qquad + (1-\pi)(\phi d+(1-\phi)e^r)^{(1-\gamma)}V_{\tau+1}(ds,1)\Big\} \\ &= w^{(1-\gamma)}V_\tau(s,1). \end{aligned}$$

In this special case, we do *not* have to use w as a state!
The function $V_t(s,1)$ can be computed by the recursion

$$\begin{aligned} V_t(s,1) &= \max_{\phi\in[0,1]}\Big\{\pi(\phi u+(1-\phi)e^r)^{(1-\gamma)}V_{t+1}(us,1) \\ &\qquad + (1-\pi)(\phi d+(1-\phi)e^r)^{(1-\gamma)}V_{t+1}(ds,1)\Big\} \end{aligned}$$

Utility maximization (contd)

Different approach: Characterize the set $\mathcal{W}_T$ of possible random wealths $\tilde{w}_T$ that can be generated using self-financing strategies.

$$\mathcal{W}_T = \left\{\tilde{w}_T : \mathbb{E}^*[\tilde{w}_T] \leq e^{rT} w_0\right\}$$

where $\mathbb{E}^*$ denote the expectation with respect to the risk-neutral (or equivalent Martingale) measure.

Binomial tree

- $T+1$ states: label states $k = 0, \ldots, T$ according to increasing stock price.
- Real world probability: $\pi_k = \pi^k (1-\pi)^{(T-k)}$
- Risk neutral probability: $q_k = q^k (1-q)^{(T-k)}$
- $w_T(k) =$ wealth in state k at time T

Utility maximization (contd)

Utility maximization problem

$$\begin{array}{rl} \text{max} & \sum_{k=0}^{T} \pi_k U(w_T(k)) \\ \text{s.t.} & \sum_{k=10}^{T} q_k w_T(k) \leq e^{rT} w_0 \end{array}$$

Simple convex optimization problem with one linear constraint.

Let $\{w_T^*(k) : k = 0, \ldots, T\}$ denote the optimal solution to this problem

- U non-decreasing implies that $\sum_{k=10}^{T} q_k w_T(k) = e^{rT} w_0$
- Binomial tree is a complete market so we can replicate any payoff
- The replication strategy gives the optimal trading strategy

Approximate dynamic programming

- Random option pay-off: $H_t = h(S_t)$ (function of stock price S_t)
- Exercise dates: $\{0, 1, \ldots, T\}$
- Price of the option: $V_t(s) = \sup_{\tau \geq t} \mathbb{E}[e^{-r(\tau-t)} h(S_\tau) | S_t = s]$
  - V_t is a function: maps the price S_t to option price
  - binomial tree $\equiv$ finite set of stock prices: explicit solution
- Alternative: Q-value iteration
  - $Q_t(s) = \mathbb{E}[e^{-r} V_{t+1}(S_{t+1}) | S_t = s]$: value from not exercising
  - Iteration: $Q_t = \mathbb{E}[e^{-r} \max\{h(S_{t+1}), Q_{t+1}(S_{t+1})\} | S_t = s]$
- Approximate Q_t: $Q_t(x) = \beta_1 \phi_1(x) + \beta_2 \phi_2(x) + \ldots \beta_L \phi_L(x)$
  - $\phi_i(x)$: known basis functions
  - need to compute the constants β_i

ADP continued

- Approximate Q-value iteration
  - Generate N paths of the stock price: $\{S_t^{(i)} : t = 1, \ldots, T\}$
  - Set $\tilde{Q}_T(s) \equiv 0$ for all s
  - For $t = T - 1 : -1 : 0$
    - Compute $\hat{Q}_t(S_t^{(i)}) = \max\left\{h(S^{(i)})_{t+1}, \tilde{Q}_{t+1}(S_{t+1}^{(i)})\right\}$
    - Compute constants $(\beta_1^{(t)}, \ldots, \beta_L^{(t)})$ by solving the least squares problem

$$\beta^{(t)} = \operatorname*{argmin}_{\beta} \sum_{i=1}^{N} \left(\hat{Q}_t(S_t^{(i)}) - \sum_{j=1}^{L} \beta_j \phi_j(S_t^{(i)})\right)^2$$

    - Set $\tilde{Q}_t(s) = \sum_{j=1}^{L} \beta_j^{(t)} \phi_j(s)$
  - Return $\tilde{V}_0 = \max\{h(S_0), \tilde{Q}_0(S_0)\}$
- Estimate: $\underline{V}_0 = \mathbb{E}[e^{-r\gamma} h(S_\gamma)]$, $\gamma = \min\{t : h_t \geq \tilde{Q}_t\}$

Information relaxation and duality

Consider a finite sample space $\Omega = \{\omega_1, \ldots, \omega_N\}$

An event $E \subset \Omega$, $\mathcal{E} =$ set all of events.

A filtration $\mathcal{F} = \{\mathcal{F}_t : t = 0, \ldots, T\}$ satisfies the following properties.

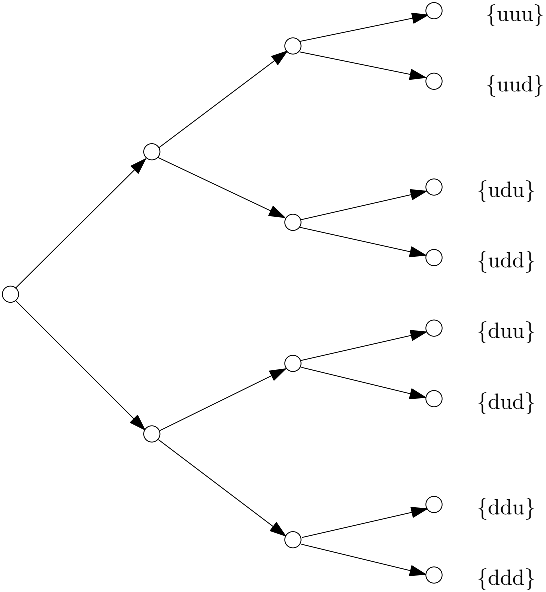
- $\mathcal{F}_t$ is a collection of events, i.e. $\mathcal{F}_t \subset \mathcal{E}$
- $\mathcal{F}_t$ is a partition of Ω
- $A_{t+1} \in \mathcal{F}_{t+1} \Rightarrow \exists\ A_t \in \mathcal{F}_t$ with $A_{t+1} \subseteq A_t$, i.e. $\mathcal{F}_{t+1} \subseteq \mathcal{F}_t$

$\mathcal{F}$ encodes the evolution of information

Binomial tree with 3 time steps

- $\Omega = \{uuu, uud, udu, udd, duu, dud, ddu, ddd\}$
- $\mathcal{F}_0 = \Omega \equiv$ one state in Ω will occur
- $\mathcal{F}_1 = \{\{uuu, uud, udu, udd\}, \{duu, dud, ddu, ddd\}\}$
- $\mathcal{F}_2 = \{\{uuu, uud\}, \{udu, udd\}, \{duu, dud\}, \{ddu, ddd\}\}$
- $\mathcal{F}_3 = \{\{uuu\}, \{uud\}, \{udu\}, \{udd\}, \{duu\}, \{dud\}, \{ddu\}, \{ddd\}\}$

Filtration tree



Actions and policies

Decision maker chooses a sequence of actions $a = (a_0, \ldots, a_T)$. Let A denote the set of all feasible action sequences.

A policy $\alpha : \Omega \mapsto A$. A policy $\alpha(\omega) = (a_0(\omega), \ldots, a_T(\omega))$ is adapted to the filtration $\mathcal{F}$ (i.e. $\alpha \in A_{\mathcal{F}}$) provided

- $a_t(\omega)$ is the same for all $\omega \in F \subset \mathcal{F}_t$ (adapted)
- The actions cannot use information that is not available!

Dynamic programming problem

$$V^* = \max_{\alpha \in A_{\mathcal{F}}} \mathbb{E}\Big[\underbrace{\sum_{t=0}^{T} r\big(a_t(\omega), \omega\big)}_{r(\alpha)} \Big]$$

- Heuristic policies give a lower bound
- Need an upper bound ... duality?

Information relaxation upper bound

A filtration $\mathcal{G}$ is called a relaxation of $\mathcal{F}$ if $\mathcal{G}_t \subseteq \mathcal{F}_t$, i.e.

$$G \in \mathcal{G}_t \quad \Rightarrow \quad \exists F \in \mathcal{F}_t \text{ with } G \subseteq F$$

- $\mathcal{G}$ has more information than $\mathcal{F}$
- Example: $\mathcal{G}_t \equiv \mathcal{F}_T$ for all t – all information is available at time $t = 0$
- How would the maximum change if a is adapted to $\mathcal{G}$?

Let $z : A \times \Omega \mapsto \mathbb{R}$ denote any function such that

$$\mathbb{E}\big[\underbrace{z(\alpha_F(\omega), \omega))}_{z(\alpha_F)}\big] \leq 0 \quad \text{for all } \mathcal{F}\text{-adapted policies } \alpha_F$$

Then

$$\begin{aligned} V^* &= \max_{\alpha_F \in A_{\mathcal{F}}} \mathbb{E}\big[r(\alpha_F)\big] \\ &\leq \max_{\alpha_F \in A_{\mathcal{F}}} \mathbb{E}\big[r(\alpha_F) - z(\alpha_F)\big] \\ &\leq \max_{\alpha_G \in A_{\mathcal{G}}} \mathbb{E}\big[r(\alpha_F) - z(\alpha_G)\big] \end{aligned}$$

Given a penalty and a relaxation, we have an upper bound!

Full information filtration

Suppose $\mathcal{G}_t \equiv \mathcal{F}_T = \Omega$ for all $t \geq 0$. Then

$$\max_{\alpha_G \in A_{\mathcal{G}}} \mathbb{E}\big[r(\alpha_G) - z(\alpha_G)\big] = \mathbb{E}\big[\max_{a \in A}\{r(a,\omega) - z(a,\omega)\}\big]$$

- The decision maker knows the state of nature ω at time $t = 0$
- Choose a sequence of actions a that optimize objective for each ω

Let $\hat{V}$ denote the value function for any feasible (heuristic) policy.

- A sequence of actions $a_0^t = (a_0, \ldots, a_t)$ results in the state

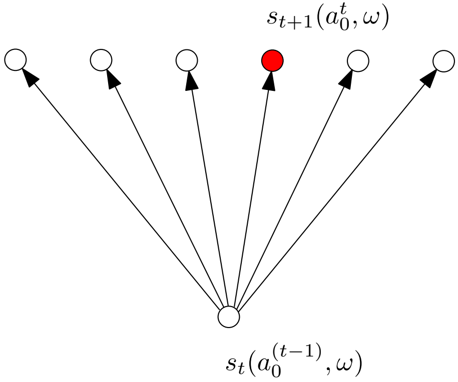
  $$s_{t+1}(a_0^t, \omega)$$

  at time $t + 1$
- Define

  $$z_t(a_0^t, \omega) = \hat{V}_{t+1}(s_{t+1}(a_0^t, \omega)) - \mathbb{E}[\hat{V}_{t+1}(\tilde{s}_{t+1}(a_0^t))|\mathcal{F}_t]$$

- $z(a,\omega) = \sum_{t=0}^{T} z_t(a,\omega)$ is a valid penalty.

Penalty for full information



$$z_t(a_0^t, \omega) \quad = \quad \hat{V}_{t+1}\big(s_{t+1}(a_0^t, \omega)\big) - \mathbb{E}\big[\hat{V}_{t+1}(s) | s_t(a_0^{t-1}, \omega), a_t\big]$$

Subtract the expected value to transform into a Martingale.

Implentation of full-information relaxation

Heuristic policy:

- Stop when $h_t(S_t) \geq \tilde{Q}(S_t)$
- Set $\tilde{V}_t(\chi) = 0$ for the "stop" state χ

Steps to generate an upper bound

- Generate sample paths of stock prices

$$S^{(k)} = \left(S_0^{(k)}, S_1^{(k)}, \ldots, S_T^{(k)}\right) \quad k = 1, \ldots, N$$

- Compute the penalty $z_t(a_0^t, k)$ as follows:

$$z_t(a_0^t, k) = \begin{cases} 0 & a_j = 0, \text{for some j} \\ \tilde{V}^{(t+1)}(S_k^{(t+1)}) - \mathbb{E}[\tilde{V}(S_{t+1})|S_t^{(k)}] & \end{cases}$$

- Compute the optimal actions for the k-th sample path

$$h^{(k)} = \max_{0 \leq \tau \leq T} \left\{h_\tau(S_\tau) + \sum_{t=1}^{\tau} z_t(\mathbf{1}, k)\right\}$$