MS-E2148 Dynamic optimization

Lecture 7

Contents

- DP-algorithm and deterministic problems with finite states
- Application: critical path analysis
- Application: comparing DNA sequences
- Application: stopping problems
- Inventory control with continuous state
- Material Bertsekas 2.1, 2.2.1 and 4.4

Recap

- The principle of optimality: if a-b-c is optimal route, then b-c is optimal subroute
- Cost-to-go: what is the cost starting from stage k to the last stage N
- DP-algorithm: compute the cost-to-go values backwards from $N-1$ to the initial stage

Deterministic, finite state space

- **Deterministic**: w_k get one value for each k
  - Problems that are truly deterministic
  - Problems that are stochastic but the noise can be assumed approximately constant
- **Finite state space**: S_k is *finite set* for all k
  - E.g. time table scheduling problems are finite
  - Are they finite in modeling altitude of 747 or the balance of bank account?
- At state x_k the control u_k means deterministic transition to state $f_k(x_k, u_k)$ with cost $g_k(x_k, u_k)$

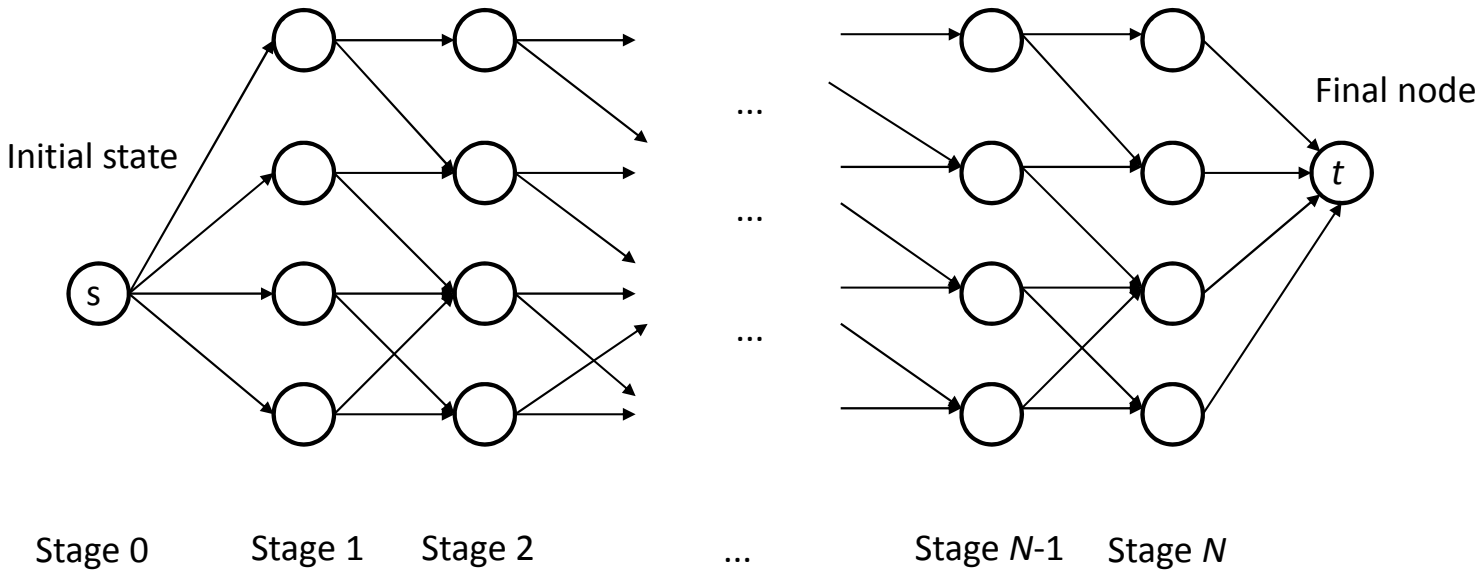
Deterministic, finite state space

- Special feature: feedback/closed-loop control does not give smaller cost (without stochastics, there is no benefit in observing the state)
- Minimizing cost over admissible controls $\{\mu_0, ..., \mu_{N-1}\} \in \Pi$ gives the same cost as minimizing cost over control vectors $\{u_0, ..., u_{N-1}\}$
- For some control law $\{\mu_0, ..., \mu_{N-1}\}$ and initial state x_0, the future states are perfectly predictable by the equation $x_{k+1} = f_k(x_k, \mu_k(x_k))$, $k = 0, 1, ..., N-1$, and the corresponding controls can be determined: $u_k = \mu_k(x_k)$
- This gives computational advantages

Deterministic, finite state space

- These problems can be presented as *graphs*:
  - **Nodes** correspond states
  - **Arcs** correspond transition between consecutive states
  - The cost of moving from state to another is given by the **length of the arc**
- Initial state: initial node s that is attached to the states of stage 1
- Final state: artificial end node t that is attached to each final state x_N with arc that has cost $g_N(x_N)$
- Controls: paths that start from s and end to some x_N

Deterministic, finite state space



Shortest path problem

- Transition cost $g_k(x_k, u_k)$ along the arc from state x_k to x_{k+1} is the length of the arc
- Let us denote: a_{ij} is the length of the arc from node i to j; if there is no arc then $a_{ij} = \infty$
- We want to find the **shortest path**, from every node i to the end node t

  $\Rightarrow$ we search for a sequence of controls that minimize the total cost of getting to t from each node $1, 2, ..., N$

Shortest path problem

- The solution exists if we require that the length of the *cycles* are non-negative
- Cycle: $(i, j_1), (j_1, j_2), ..., (j_k, i)$
- The optimal path is at most N length: formulate the problem as N-length but allow degenerate transitions from node i to itself so that $a_{ii} = 0$

- We denote for all $i = 1, ..., N$ and $k = 0, ..., N - 1$:
  $J_k(i) =$ optimal cost with $N - k$ steps from node i to end t

Shortest path problem

- The cost from i to t is then $J_0(i)$

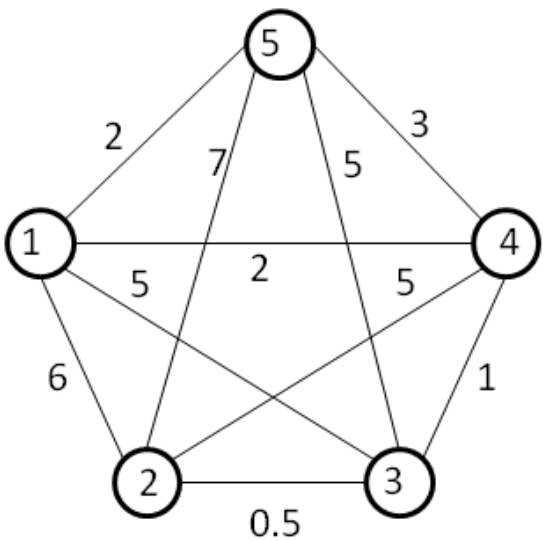
- **DP algorithm for the shortest path problem**: the optimal cost from i to t in $N-k$ steps is

$$J_k(i) = \min_{j=1,\ldots,N} [a_{ij} + J_{k+1}(j)], \quad k = 0, 1, \ldots, N-2,$$

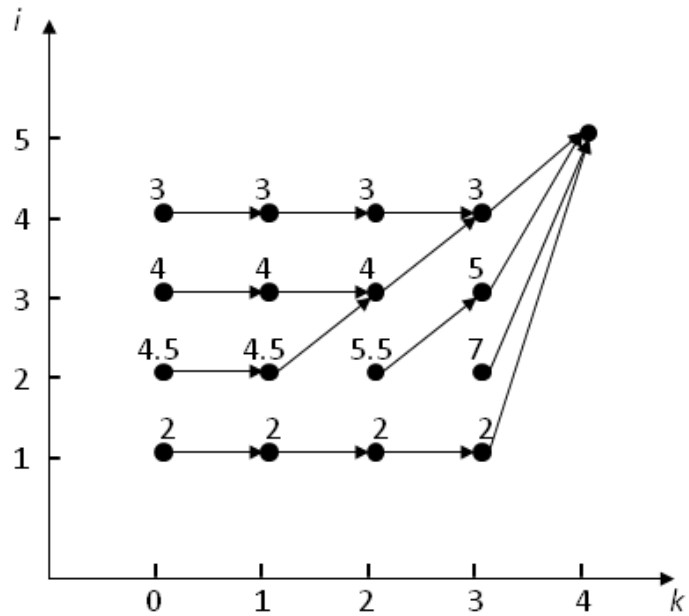
  where $J_{N-1}(i) = a_{it}$, $i = 1, 2, \ldots, N$.
- Note that the above $J_{k+1}(j)$ = optimal cost from j to t in $N-k-1$ steps

Shortest path problem: example

- The figure (a) is a graph where the costs a_{ij} are denoted on the arcs; figure (b) gives the cost-to-go $J_k(i)$ from each node i and stage k generated by DP
- Node 5 is the end node; $N = 4$, $k = 0, 1, 2, 3, 4$



(a)



(b)

Shortest path problem: example

- The cost-to-go values are computed with DP in the following way in figure (b)
  - $J_2(2) = \min\{5+3, 0.5+5, 6+2, ...\} = 5.5$ (2 steps)
  - $J_0(4) = \min\{0+0+0+3, 0+0+1+5, ...\} = 3$ (4 steps)
- Zero means staying in the node: $a_{ii} = 0$
- The optimal paths can be seen from the figure (b):

$$1 \to 5,$$
$$2 \to 3 \to 4 \to 5,$$
$$3 \to 4 \to 5,$$
$$4 \to 5$$

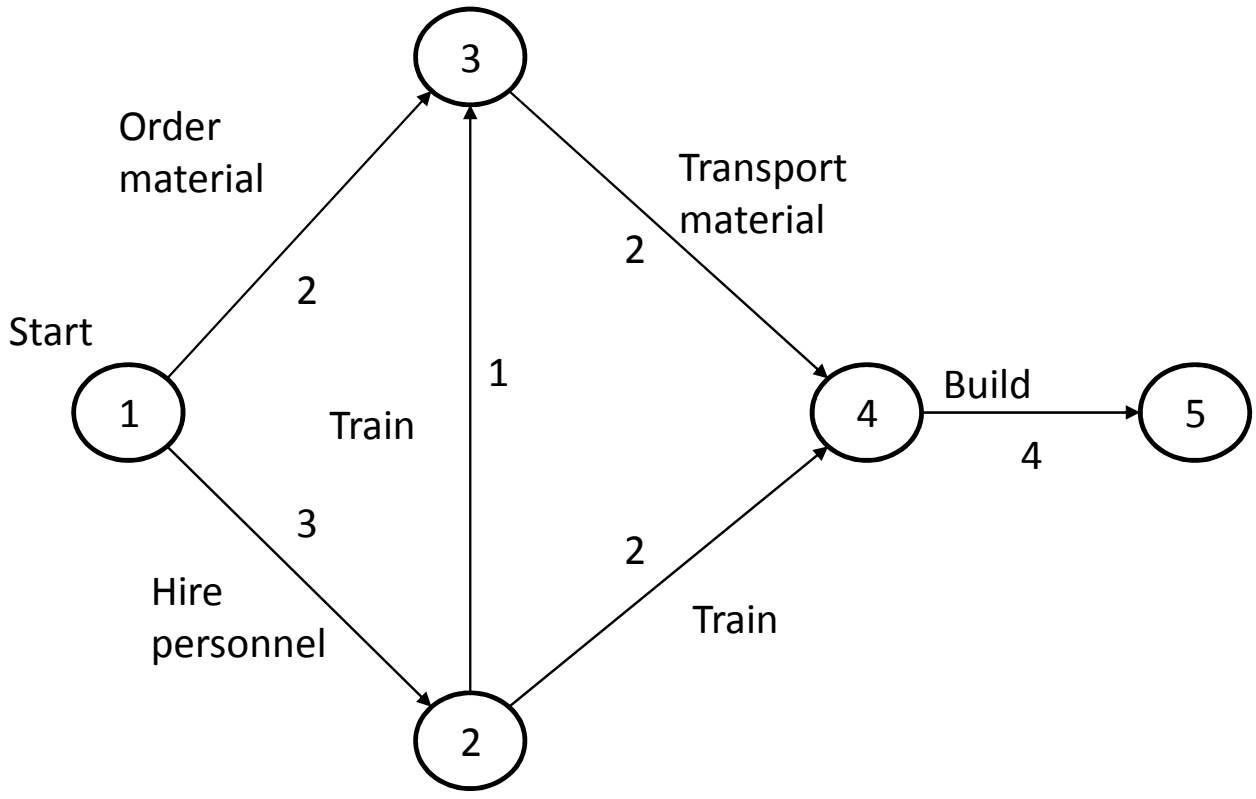
Application: critical path analysis

- Project has various activities and some of them need to be finished before starting the next ones
- How to schedule the project?
- What are the *critical activities*, which cause the whole project to delay if they are delayed?
- How long does it take to complete the project (in minimal time)?

Application: critical path analysis

- Node in the graph represents the ending of a stage
- Arc (i, j) describes activity that start when stage i is completed; arc length is the length of the activity $t_{ij} > 0$
- Stage j is done when all activities that end at node j (arcs (i, j)) are completed
- Nodes 1 and N are the start and end nodes
- Graph does not contain cycles

Application: critical path analysis



Application: critical path analysis

- $p = \{(1, j_1), (j_1, j_2), ..., (j_k, i)\}$ is path from start to node i
- D_p is the length of the path: $D_p = t_{1j_1} + t_{j_1 j_2} + ... + t_{j_k i}$
  - Esim: $D_{1\to 2\to 3} = 3 + 1 = 4$, $D_{1\to 3} = 2$
- Completion of stage i takes

$$T_i = \max_p D_p$$

  - E.g.: completion of stage 3 takes $\max\{4, 2\} = 4$

- To find T_i we need to find the *longest path* from 1 to i
  - Does the longest path exist? Yes since there are no cycles

Application: critical path analysis

- The length of the project is the longest path from 1 to N; this is called the *critical* path
- Delay in any of the activities on the critical path will delay the whole project with that amount

- Note: it is a shortest path problem if we multiply t_{ij} by -1

Application: critical path analysis

- Let S_1 be a group of stages (nodes) that do not depend on completion of any other stage
- Let S_k, when $k = 1, 2, ...,$ be a set
  $S_k = \{i|$ all paths from 1 to i have $\leq k$ arcs $\}$, where
  $S_0 = \{1\}$.
- The sets S_k are states of the DP, and DP-algorithm is

$$T_i = \min_{(j,i)}[t_{ji} + T_j], \quad \forall i \in S_k, j \in S_{k-1}, i \notin S_{k-1} \qquad (1)$$

- This is "*forward DP-algorithm*" that starts from 1 and ends at N; computation does not change from the regular DP

Application: critical path analysis

- For the problem in the figure

$$S_0 = \{1\}, \quad S_1 = \{1, 2\}, \quad S_2 = \{1, 2, 3\}$$
$$S_3 = \{1, 2, 3, 4\}, \quad S_4 = \{1, 2, 3, 4, 5\}$$

- Using the algorithm (1) we get

$$T_1 = 0, T_2 = 3, T_3 = 4, T_4 = 6, T_5 = 10.$$

- Critical path is $1 \to 2 \to 3 \to 4 \to 5$

Application: comparison of DNA sequences

- The task is to compare two DNA sequences and find how similar they are
- Sequence consists of nucleotides that are named by their bases **G**(uanine), **A**(denine), **C**(cytosine), **T**(ymine)
- Sequence may contain (*gaps*), but there can be no gaps at the start or at the end

- Widely used Needleman-Wunsch algorithm is based on DP-algorithm in bioinformatics
- The following example is based on: `http://www.avatar.se/molbioinfo2001/dynprog/dynamic.html`

Application: comparison of DNA sequences

- Let us compare two sequences:

```
(1) G A A T T C A G T T A
(2) G G A T C G A
```

- They can be aligned the following way:

```
(1) G A A T T C A G T T A
(2) G G A - T C - G - - A
```

- Sequence (2) has gaps
- Sequence (2) may develop to sequence (1) e.g. by mutation when one base A is transformed to G

Application: comparison of DNA sequences

- To find out similarity of the sequences, we weight different alignments:
  $S_{i,j} = 1$ if base in sequence 1 at place j matches the base of sequence 2 at place i; otherwise $S_{i,j} = 0$;
  $w = 0$ is a penalty for having a gap
- The alignment on previous slide gives:

$$S_{G,G} + S_{G,A} + S_{A,A} + w + S_{T,T} + S_{C,C} + w + S_{G,G} + 2w + S_{A,A} = 1 + 0 + 1 + 0 + 1 + 1 + 0 + 1 + 0 + 1$$

Application: comparison of DNA sequences

- The task is to find the alignment with highest score
- We design an alignment matrix whose elements are:

$$M_{i,j} = \max \left\{ \begin{array}{ll} M_{i-1,j-1} + S_{i,j}, & \text{(match/no match)} \\ M_{i,j-1} + w, & \text{(gap in sequence 2)} \\ M_{i-1,j} + w, & \text{(gap in sequence 1)} \end{array} \right\}$$

- This is recursion of "*forward-DP*", the text in the bracket gives the "control" how the sequence (2) is built
- Also assume that $i, j = 1, 2, \ldots$ and $M_{0,0} = M_{i,0} = M_{0,j} = 0$

Application: comparison of DNA sequences

- $M_{1,1} = \max\{0 + 1, 0 + 0, 0 + 0\} = 1$

	G	A	A	T	T	C	A	G	T	T	A
G	1										
G											
A											
T											
C											
G											
A											

- (Sequences cannot start with gaps)

Application: comparison of DNA sequences

- $M_{1,2} = \max\{M_{0,1} + S_{1,2}, M_{1,1} + w, M_{0,2} + w\} = 1 + 0$
- $M_{1,3} = \max\{M_{0,2} + S_{1,3}, M_{1,2} + w, M_{0,3} + w\} = 1 + 0$
- $M_{2,1} = \max\{M_{1,0} + S_{2,1}, M_{2,0} + w, M_{1,1} + w\} = 0 + 1$

	G	A	A	T	T	C	A	G	T	T	A
G	1	1	1	1	1	1	1	1	1	1	1
G	1										
A	1										
T	1										
C	1										
G	1										
A	1										

Application: comparison of DNA sequences

- $M_{2,2} = \max\{M_{1,1} + S_{2,2}, M_{2,1} + w, M_{1,2} + w\} = 1 + 0$
- $M_{3,2} = \max\{M_{2,1} + S_{3,2}, M_{3,1} + w, M_{2,2} + w\} = 1 + 1$

	G	A	A	T	T	C	A	G	T	T	A
G	1	1	1	1	1	1	1	1	1	1	1
G	1	1									
A	1	2									
T	1	2									
C	1	2									
G	1	2									
A	1	2									

Application: comparison of DNA sequences

	G	A	A	T	T	C	A	G	T	T	A
G	1	1	1	1	1	1	1	1	1	1	1
G	1	1	1	1	1	1	1	2	2	2	2
A	1	2	2	2	2	2	2	2	2	2	3
T	1	2	2	3	3	3	3	3	3	3	3
C	1	2	2	3	3	4	4	4	4	4	4
G	1	2	2	3	3	4	4	5	5	5	5
A	1	2	3	3	3	4	5	5	5	5	6

- Maximum score is 6

Application: comparison of DNA sequences

- Now we find the alignment that gives score 6 by *traceback*-algorithm, where we move from the element of maximum score to the elementh $M_{1,1}$
- **Traceback**: find the predecessor of i, j:
  - If $M(i,j) = M(i,j-1)$, seq. (2) has gap at place i
  - If $M(i,j) = M(i-1,j)$, seq. (1) has gap at place j
  - If $M(i,j) > M(i-1,j), M(i,j-1), M(i-1,j-1)$, then element j of sequence (1) matches with element i of sequence (2)
  - If $M(i,j) > M(i-1,j-1)$, but is equal to $M(i,j-1)$ or $M(i-1,j)$, then element j of sequence (1) does not match with element i of sequence (2)
- The example has many possible alignments

Application: comparison of DNA sequences

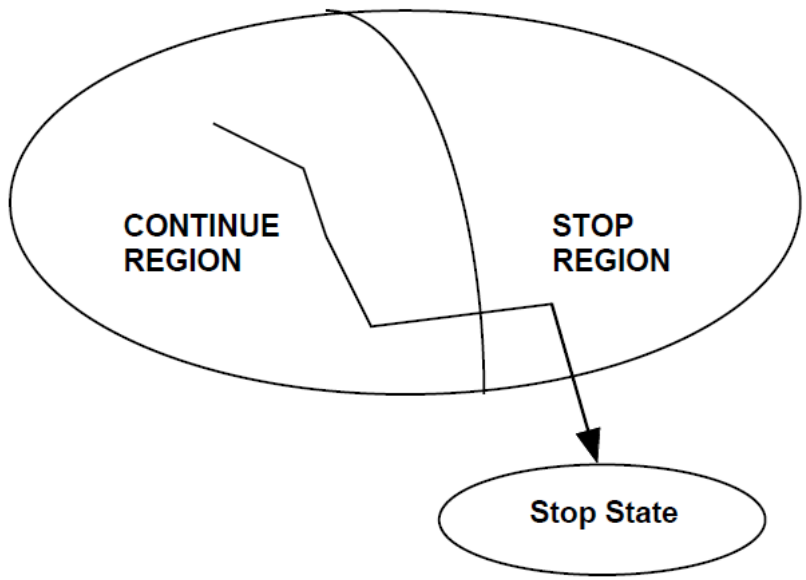
- Alignment and scoring matrix

```
(1) G A A T T C A G T T A
(2) G G A - T C - G - - A
```

	G	A	A	T	T	C	A	G	T	T	A
G	**1**	1	1	1	1	1	1	1	1	1	1
G	1	**1**	1	1	1	1	1	2	2	2	2
A	1	2	**2**	**2**	2	2	2	2	2	2	3
T	1	2	2	3	**3**	3	3	3	3	3	3
C	1	2	2	3	3	**4**	**4**	4	4	4	4
G	1	2	2	3	3	4	4	**5**	**5**	**5**	5
A	1	2	3	3	3	4	5	5	5	5	**6**

Application: Stopping problems

- Two controls: stop (stopping cost and absorption) or continue (state equation and cost)
- Control law splits the states into two regions: stopping and continuing states



Application: Selling goods

- A person is selling assets and gets an offer w_k at stage $k = 0, 1, \ldots, N-1$
- The offer can be accepted and the money can be invested into fixed rate r or rejected (wait for the next offer)
- Offer w_{N-1} must be accepted at the end

Application: Selling goods

- DP algorithm: (offer x_k, end state T)

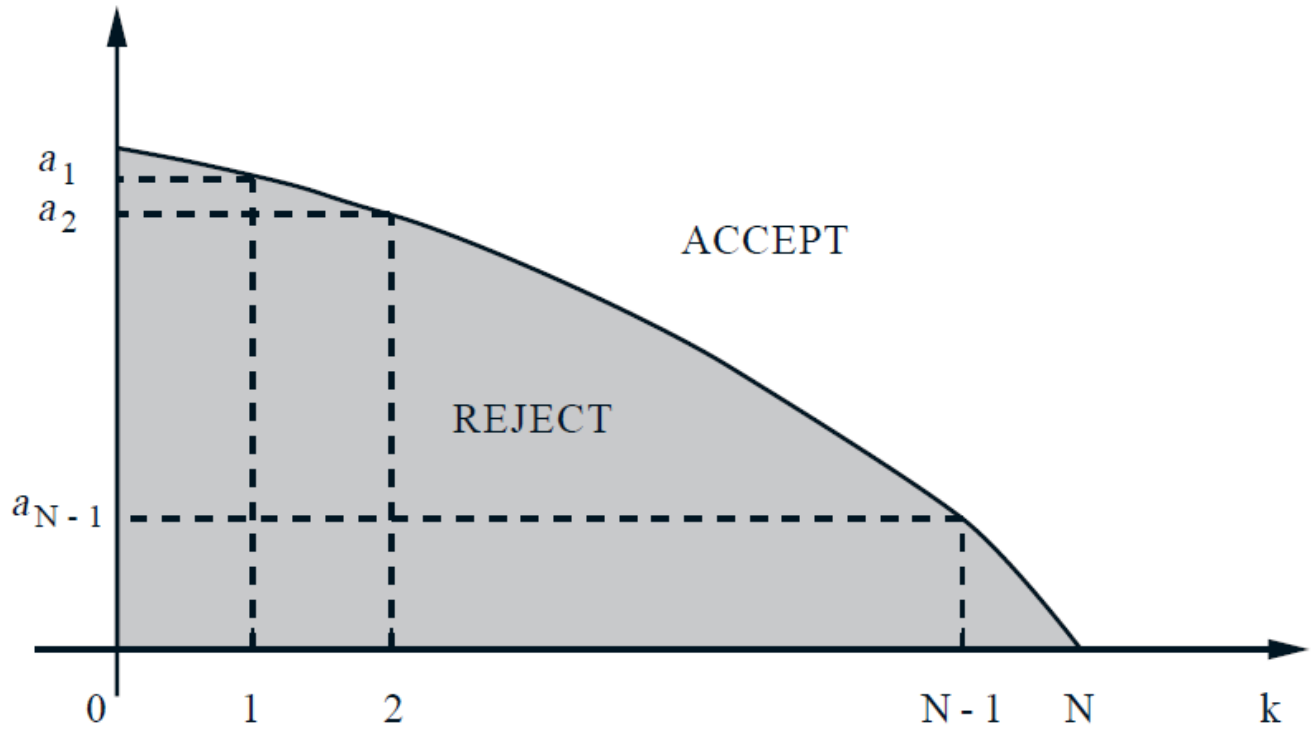
$$J_N(x_N) = \begin{cases} x_N, & \text{if } x_N \neq T \\ 0, & \text{if } x_N = T \end{cases}$$

$$J_k(x_k) = \begin{cases} \max\left[(1+r)^{N-k}x_k, E(J_{k+1}(w_k))\right], & \text{if } x_k \neq T \\ 0, & \text{if } x_k = T \end{cases}$$

- Optimal control law:
  accept if $x_k \geq \alpha_k$ and reject if $x_k < \alpha_k$, where

$$\alpha_k = \frac{E(J_{k+1}(w_k))}{(1+r)^{N-k}}$$

Application: Selling goods



- It can be shown that $\alpha_k \geq \alpha_{k+1}$ for all k
- Also $\alpha_k \to \bar{a}$ when $k \to \infty$ and the infinite horizon problem is stationary.

Example: inventory control

- Order the product in N stages so that the stochastic demand is satisfied and the expected cost is minimized

- x_k is the amount of product at the beginning of stage k
- u_k is the order quantity at stage k that arrives immediately and satisfies $u_k \geq 0$
- w_k is the stochastic demand at stage k; $w_0, w_1, \ldots, w_{N-1}$ are independent of each other, but the demand may depend on the state and/or the control

Example: inventory control

- The equation for the inventory is

$$x_{k+1} = x_k + u_k - w_k$$

- Negative x_k means that the demand is not met, positive means there is extra inventory

Example: inventory control

- The total cost over N periods is

$$E\left\{\sum_{k=0}^{N-1}(cu_k + r(x_k + u_k - w_k))\right\}$$

- The cost is made of two components:
  a. Ordering cost cu_k
  b. Inventory cost $r(x_k + u_k - w_k)$, which can be positive or negative
- How do you interpret the parameters c and r?
- What about the final cost $g_N(x_N)$?

Example: inventory control

- The control law $\pi = \{\mu_k(x_k)|k = 0, 1, ..., N-1\}$ gives a description how to act in different cases: ”if you observe the inventory x_k at stage k, order $\mu_k(x_k)$”
- Is this closed or open-loop control?

Example: inventory control

- The task is to find the best control law that minimizes the cost:
  $\pi^* = \arg\min_\pi \{J_\pi(x_0)\}$ where

$$J_\pi(x_0) = E\left\{ \sum_{k=0}^{N-1} (cu_k + r(x_k + u_k - w_k)) \right\}$$

s.t.

$$x_{k+1} = x_k + u_k - w_k$$

DP-algorithm: inventory control

- The subproblems of length one: at stage $N-1$ the inventory is x_{N-1}. Choose control u_{N-1} that minimizes the ordering cost and the expected inventory cost:

$$J_{N-1}(x_{N-1}) = \min_{u_{N-1} \geq 0} \left[cu_{N-1} + E_{w_{N-1}} \{ r(x_{N-1} + u_{N-1} - w_{N-1} \} \right]$$

- When this control is determined, we get the last part of the optimal control law $\mu^*_{N-1}(x_{N-1})$

DP-algorithm: inventory control

- The subproblems of length two: at stage $N-2$ the inventory is x_{N-2}. Choose control u_{N-2} to minimize:

  (the cost at stage $N-2$) + (the cost at stage $N-1$ s.t. we use the optimal control for the subproblem μ^*_{N-1}), i.e.,

$$J_{N-2}(x_{N-2}) = \min_{u_{N-2}\geq 0}\Big[cu_{N-2} + E_{w_{N-2}}\{r(x_{N-2}+u_{N-2}-w_{N-2}) + J_{N-1}(x_{N-2}+u_{N-2}-w_{N-2})\}\Big]$$

- Note: $x_{N-1} = x_{N-2} + u_{N-2} - w_{N-2}$
- When this control is determined, we also get the second last part of the optimal control law $\mu^*_{N-2}(x_{N-2})$

DP-algorithm: inventory control

- The subproblems of length $N-k$: at stage k the inventory is x_k. Choose control u_k that minimizes:

$$J_k(x_k) = \min_{u_k \geq 0} \Big[cu_k + E_{w_k}\{ r(x_k + u_k - w_k) + J_{k+1}(x_k + u_k - w_k) \} \Big] \tag{2}$$

- The functions $J_k(x_k)$ describe the optimal expected cost for subproblems that *start* at stage k; these functions are computed recursively backwards in time starting from stage $N-1$ and ending to stage 0
- The optimal cost for the problem is $J_0(x_0)$. The optimal control law will be determined for the minimization problem (2) for each k

DP-algorithm

- To solve the optimal $J_k^*(x_k)$ and $u_k^* = \mu_k^*(x_k)$, the right-hand side minimization problem should have a solution
- In the inventory control, this is satisfied, e.g., when the penalty term is function $r(\cdot) = (x_k + u_k - w_k)^2$
- Minimization problem $\min_{u_k}[L(x_k, u_k)]$ is solved using the **first-order conditions**: the partial derivative is set to zero: $\partial L / \partial u_k = 0$ and from this we solve $u_k^* = \mu_k^*(x_k)$
- In DP-algorithm we have N of these minimization problems

DP-algorithm

Formulation

- For each initial state x_0 of the basic problem, the optimal cost $J^*(x_0)$is $J_0(x_0)$, which we get at the last step of the computation:

$$J_N(x_N) = g_N(x_N), \tag{3}$$

$$J_k(x_k) = \min_{u_k \in U_k(x_k)} E_{w_k}\Big\{g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k))\Big\}, \tag{4}$$

  $k = 0, 1, ..., N-1$

- If $u_k^* = \mu_k^*(x_k)$ minimizes the right-hand side of (4) for each x_k and k, the control law $\pi^* = \{\mu_0^*, ..., \mu_{N-1}^*\}$ is optimal

Summary

- Deterministic problem
- Finite state space
- Shortest path problem
- DP-algorithm in shortest path problem