Deterministic dynamic programming

Fabian Bastin
fabian.bastin@cirrelt.ca
University of Montréal – CIRRELT

Summer School 2015

A brief definition of dynamic programming (DP)

Sequential decision process:

- Sequence of decisions to take, with an objective to optimize.
- Discrete time: at each step, one takes a decision, depending on the available information.
- Interactions between the decisions.
- It is possible to get additional information at each step, and there could be some randomness between the decision steps.
- Continuous time: the sequence of decisions if replaces by a control, function of time.

General definitions

Dynamic programming:
Set of mathematical and algorithmic tools designed to study sequential decision processes and calculate optimal strategies (exact or approximative).

A policy (or strategy) is a decision rule that, for each possible system configuration (system state), gives the decision (or action) to take in order to optimize a global objective function.

A deterministic PD model

At step k, the system is in the state $x_k \in X_k$.

A decision make observes x_k and take a decision (action) $u_k \in U_k(x_k)$.
She endures a cost $g_k(x_k, u_k)$ during this step, then the system transits to a new state $x_{k+1} = f_k(x_k, u_k)$ at step $k+1$.

At step k, we have:

- $X_k =$ state space;
- $U_k(x) =$ set of admissible decision in the state x;
- $g_k =$ cost function;
- $f_k =$ transition function;
- $x_k =$ system state at step k;
- $u_k =$ decision at step k.

A deterministic PD model (II)

One aims to minimize the sum of the costs from step 0 to step N:

$$\begin{aligned} \min \quad & g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k) \\ \text{s.t.} \quad & u_k \in U_k(x_k) \text{ and } x_{k+1} = f_k(x_k, u_k),\ k = 0, \ldots, N-1; x_0 \text{ fixed.} \end{aligned}$$

$$\begin{array}{llll} x_0 & \to u_0 = \mu_0(x_0), & w_0 & \to x_1 = x_0 + u_0 - w_0 \\ x_1 & \to u_1 = \mu_1(x_1), & w_1 & \to x_2 = x_1 + u_1 - w_1 \\ x_2 & \to u_2 = \mu_2(x_2), & w_2 & \to x_3 = x_2 + u_2 - w_2 \\ x_3 & \cdots & & \end{array}$$

Mathematical program

$$\begin{aligned} \min \quad & g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k) \\ \text{s.l.c.} \quad & u_k \in U_k(x_k) \text{ et } x_{k+1} = f_k(x_k, u_k), \quad k = 0, \dots, N-1 \\ & x_0 \text{ fixed.} \end{aligned}$$

The expressions for g_k and f_k can be complicated (nonlinear, nonconvex,...).

If $g_k(x_k, u_k)$ represents a gain instead of a cost, one replacee "min" by "max" (or "inf" by "sup").

An admissible policy (or strategy) is a suite of functions $\pi = (\mu_0, \dots, \mu_{N-1})$ such that $\mu_k : X_k \to U_k$ and $\mu_k(x) \in U_k(x)$ for all $x \in X_k$, $0 \leq k \leq N-1$. The decision at step k is $\mu_k(x_k)$.

Recurrence equations – Bellman equations

Given a policy π, let

$$
\begin{aligned}
J_{\pi,k}(x) &= \text{total cost from steps } k \text{ to } N \text{ given state } x \\
&\quad \text{at step } k, \text{ is one follows the policy } \pi \\
&= g_N(x_N) + \sum_{n=k}^{N-1} g_n(x_n, u_n) \\
\text{s.t.} \quad & u_n = \mu_n(x_n) \text{ and } x_{n+1} = f_n(x_n, u_n), \quad n = k, \ldots, N-1 \\
& x_k = x \text{ (fixed).}
\end{aligned}
$$

These values satisfy the following recurrence equations (or functional equations):

$$
\begin{aligned}
J_{\pi,N}(x) &= g_N(x) \quad \forall x \in X_N \\
J_{\pi,k}(x) &= g_k(x, \mu_k(x)) + J_{\pi,k+1}(f_k(x, \mu_k(x))) \text{ for all } x \in X_k, \\
&\quad \text{for } k = N-1, N-2, \ldots, 1, 0.
\end{aligned}
$$

Optimal policy

For $0 \leq k \leq N$ and $x \in X_k$,

$$
\begin{aligned}
J_k(x) &= \text{optimal cost for steps } k \text{ to } N \text{ give state } x \text{ at step } k \\
&= \min_{u_k,\ldots,u_{N-1}} \left(g_N(x_N) + \sum_{n=k}^{N-1} g_n(x_n, u_n) \right) \\
\text{s.t.} \quad & u_n \in U_n(x_n) \text{ and } x_{n+1} = f_n(x_n, u_n), \quad n = k, \ldots, N-1 \\
& x_k = x \text{ (fixed)}.
\end{aligned}
$$

An admissible policy $\pi^* = (\mu_0^*, \ldots, \mu_{N-1}^*)$ such that

$$J_{\pi^*,0}(x) = J_0(x) \overset{def}{=} J(x) \qquad \text{for all } x$$

is called optimal policy.

Backtracking

One has the recurrence equations (Bellman equations):

$$\begin{aligned} J_N(x) &= g_N(x) \quad \text{for all } x \in X_N \\ J_k(x) &= \min_{u \in U_k(x)} \{g_k(x,u) + J_{k+1}(f_k(x,u))\} \text{ for all } x \in X_k, \\ &\qquad \text{for } k = N-1, N-2, \ldots, 1, 0. \end{aligned}$$

One looks for $J_0(x_0)$ given the initial state x_0.

One can solve the program using backtracking:
Compute $J_{N-1}(x)$ for all $x \in X_{N-1}$, then $J_{N-2}(x)$ for all $x \in X_{N-2}$, etc.

How to find back the optimal solution?
During the computations, one memorizes, for each k and $x \in X_k$,

$$\mu_k^*(x) = \arg \min_{u \in U_k(x)} \{g_k(x,u) + J_{k+1}(f_k(x,u))\},$$

which is the optimal decision to take in state x at step k.

Dynamic programming algorithm

Backward chaining procedure

For all $x \in X_N$, $J_N(x) \leftarrow g_N(x)$;

For $k = N-1, \ldots, 0$ do

For all $x \in X_k$ do

$$J_k(x) \leftarrow \min_{u \in U_k(x)} \{g_k(x,u) + J_{k+1}(f_k(x,u))\};$$

$$\mu_k^*(x) \leftarrow \arg\min_{u \in U_k(x)} \{g_k(x,u) + J_{k+1}(f_k(x,u))\};$$

Bellman optimality principle

If $\pi^* = (\mu_0^*, \ldots, \mu_{N-1}^*)$ is an optimal policy for the initial problem and if $0 \leq i \leq j \leq N-1$, then the truncated policy $\pi_{i,j}^* = (\mu_i^*, \ldots, \mu_j^*)$ is an optimal policy for the subproblem

$$\min_{u \in U_k(x)} \sum_{k=i}^{j} g_k(x_k, u_k)$$

given x_i and x_j (setting $g_N(x_N, u_N) \equiv g_N(x_N)$).

Major assumptions: discrete time and additive costs.

If the cost is not additive, the optimality principle does not necessarily hold.
For instance, one replaces the sum by the maximum operator:

$$\max \left[g_k(x_k, u_k), \ldots, g_{N-1}(x_{N-1}, u_{N-1}), \, g_N(x_N) \right].$$

Example: allocation of medical teams

One can allocate 5 medical teams to 3 countries. Each additional team allocated to a country increase the life expectation in this country, but this increase is nonlinear.

Life expectation increase (in thousands of persons–years			
Allocated teams, u_k	Country 0	Country 1	Country 2
0	0	0	0
1	45	20	50
2	70	45	70
3	90	75	80
4	105	110	100
5	120	150	130

Example: allocation of medical teams II

Let u_k (an integer) be the number of allocated teams to country k. One aims to maximize the total increase of life expectation, for all the 3 countries.

Sequential DP: $N = 3$. At step $k \leq 2$, one still has $u_k, \ldots, u_2$ to fix, x_k teams still being available, and $J_k(x_k)$ is the optimal revenue for countries $k, \ldots, 2$.

$g_k(x_k, u_k) = g_k(u_k)$ are given in the table (here, they depend on u_k only, not on x_k.)

The node (k, x) corresponds to the state x_k: it remains x teams for the countries $k, \ldots, 2$. The backtracking chaining is equivalent to look for the longest path from $(0, 5)$ to $(3, 0)$.

Deterministic PD and shortest path

Given x_0, one aims to solve

$$\min_{\mu_0,\dots,\mu_{N-1}} \quad g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k))$$

$$\text{s.t.} \quad \mu_k(x_k) \in U_k(x_k) \text{ et } x_{k+1} = f_k(x_k, \mu_k(x_k)), \quad k = 0, \dots, N-1$$

which is equivalent to

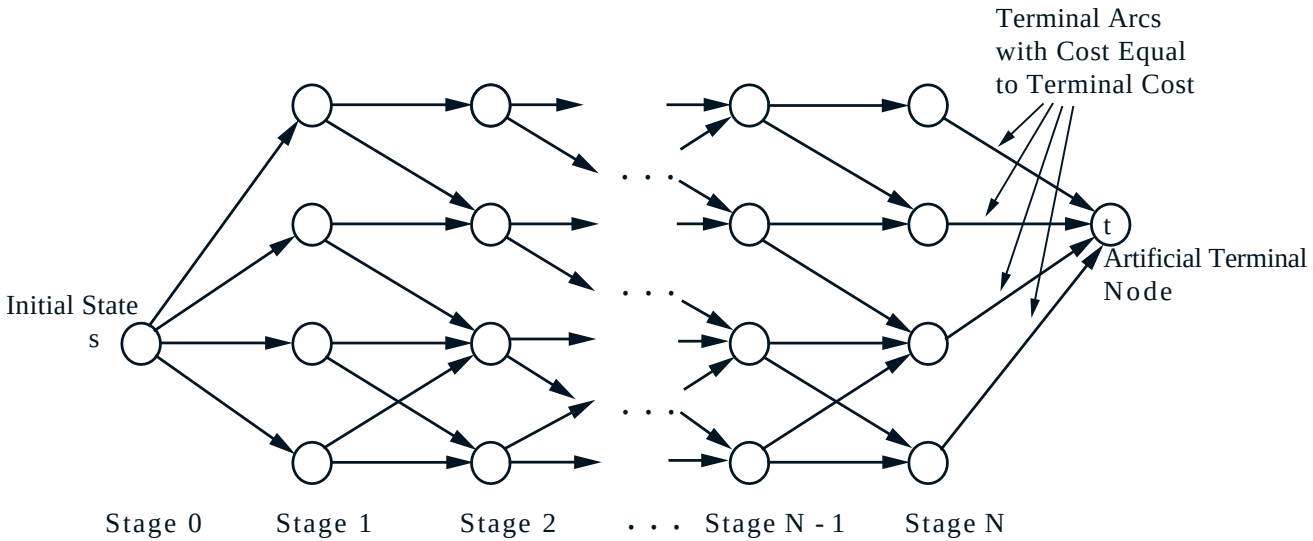
$$\min_{u_0,\dots,u_{N-1}} \quad g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$$

$$\text{s.t.} \quad u_k \in U_k(x_k) \text{ et } x_{k+1} = f_k(x_k, u_k), \quad k = 0, \dots, N-1.$$

In this case, one can compute the optimal decisions $u_0, \dots, u_{N-1}$ from the beginning, as no new information is obtained at a later stage.

Shortest path

If X_k et U_k are finite, solving this problem is equivalent to find a shortest path in a network, where the nodes are the states (k, x_k), for $0 \leq k \leq N$ and $x_k \in X_k$, augmented with an artificial nodel t that corresponds to a state where everything is over (stage $N+1$). For each node (k, x_k), $k < N$, and each decision $u_k \in U_k(x_k)$, there is a link of length $g(x_k, u_k)$ starting from node (k, x_k) to node $(k+1, x_{k+1})$, with $x_{k+1} = f_k(x_k, u_k)$. Each node (N, x_N) is connected to the node t by a link of length $g(x_N)$. One is looking for the shortest path from $s = (0, x_0)$ to t.



Shortest path (II)

If one numbers the nodes layer by layer, in ascending order value of stage k, one obtains a network without cycle and topologically ordered (i.e., a link (i, j) can exist only if $i < j$).

When it is not necessary to memorize the stage index, one can simplify the network by aggregating the nodes. The resulting network is not necessarily topologically ordered.

All shortest path problem in a network can also be formulated as a deterministic DP program.

Computation of the shortest path

Notation:

J_i = minimum distance from node i to node t;
D_i = minimum distance from mode 0 to node i;
u_i^* = the next node to visit, starting from i.

If one finds all the u_i^*, one has an optimal path.

One has

$$J_t = D_0 = 0; \qquad J_0 = \min_{1 \leq i \leq t} (D_i + J_i)$$

for all i, but this equation does not tell us how to solve the problem.

Backward chaining

Topological order: link (i, j) exists $\Rightarrow$ $i < j$.

Recurrence: $J_t = 0$ and

$$\begin{aligned} J_i &= \min_{\{j|j>i\}} \left\{a_{ij} + J_j\right\}, \quad i < t; \\ u_i^* &= \arg \min_{\{j|j>i\}} \left\{a_{ij} + J_j\right\}. \end{aligned}$$

PROCEDURE Backward Chaining;
    $J_t \leftarrow 0$;
    FOR $i \leftarrow t - 1$ GOING DOWN TO 0 DO
        $J_i \leftarrow \infty$;
        FOR $j \leftarrow i + 1$ TO t DO
            IF $a_{ij} + J_j < J_i$ THEN $J_i \leftarrow a_{ij} + J_j$ AND $u_i^* \leftarrow j$.

One computes the optimal path from each node i to the node t.
This formulation is called the initial value problem.

Forward chaining

Let D_j = minimum distance from node 0 to node j;
v_j^* = the previous node j on the optimal path from 0 to j.

Recurrence: $D_0 = 0$ and

$$\begin{aligned} D_j &= \min_{\{i|i<j\}} \left\{ D_i + a_{ij} \right\}; \\ v_j^* &= \arg \min_{\{i|i<j\}} \left\{ D_i + a_{ij} \right\}. \end{aligned}$$

PROCEDURE Forward Chaining;
$D_0 \leftarrow 0$;
FOR $j \leftarrow 1$ TO t DO
$D_j \leftarrow \infty$;
FOR $i \leftarrow 0$ TO $j - 1$ DO
IF $D_i + a_{ij} < D_j$ DO $D_j \leftarrow D_i + a_{ij}$ AND $v_j^* \leftarrow i$.

This formulation is called the final value problem.

Accession method

Same recurrence equations as in forward chaining, but one permutes the loops “FOR”.

PROCEDURE Accession;
$D_0 \leftarrow 0$;
FOR $j \leftarrow 1$ TO t DO $D_j \leftarrow \infty$;
FOR $i \leftarrow 0$ TO $t-1$ DO
FOR $j \leftarrow i+1$ TO t DO
IF $D_i + a_{ij} < D_j$ THEN $D_j \leftarrow D_i + a_{ij}$ AND $v_j^* \leftarrow i$.

The accession method becomes advantageous if one can eliminate node i during the computation process, for instance, if D_i is fixed and is greater than the current (temporary) value of D_t.

Labeling algorithms

Do not require that the network is topologically ordered.

There could be cycles and links of negative length, but no cycle of negative length.

One here assumes that $0 \leq a_{ij} \leq \infty$. Let

$$
\begin{aligned}
T &= \text{set of nodes with temporary labels.} \\
d_j &= \text{length of shortest path from 0 to } j \text{ without visiting } T \\
&= \min_{i \notin T} \left(d_i + a_{ij} \right). \\
v_j &= \text{node preceding } j \text{ on the best path from 0 to } j \text{ to date.}
\end{aligned}
$$

At the beginning, all the nodes are put in T, except 0.
At each iteration, one removes one node i from T and one gives a permanent label: $D_i = d_i$. When T is empty, stop.

Dijkstra algorithm

PROCEDURE Accession; // Dijkstra algorithm

$d_0 \leftarrow 0$; FOR $j \leftarrow 1$ TO t DO $d_j \leftarrow a_{0j}$; $v_j \leftarrow 0$;

$T \leftarrow \{1, 2, \ldots, t\}$;

WHILE $T \neq \phi$ DO

$i \leftarrow \arg\min_{j \in T} d_j$; $T \leftarrow T - \{i\}$;

// Invariant: $d_i = D_i$.

FOR EACH $j \in T$ DO

IF $d_i + a_{ij} < d_j$ THEN $d_j \leftarrow d_i + a_{ij}$ AND $v_j \leftarrow i$.

Proposition.

If it exists a path from 0 to t, at the end of the algorithm, one has $T = \phi$, $d_j = D_j$, and $v_j = v_j^*$ for all j.

One thus has a shortest path from 0 to each of the other node.

Label correction algorithm

Let

d_j = length of the shortest path from 0 to j to date.
v_j = node preceding j on the best path to date from 0 to j.
U = UPPER
= upper bound on the length of the shortest path.
O = OPEN
= nodes i whose label has been modified, but one still has to adjust the labels of their successors j, by checking if $d_i + a_{ij} < d_j$.

A the beginning, O contains the origin only.
At every iteration, one removes one node i from O and we try to reduce the d_j of the successors j from i, by checking if $d_i + a_{ij} < d_j$. When one lowers d_j, one put j in O. When O is empy, stop.

Label correction algorithm (II)

PROCEDURE LABEL CORRECTION;

$U \leftarrow \infty$; $O \leftarrow \{0\}$; $d_0 \leftarrow 0$;

FOR $j \leftarrow 1$ TO t DO $d_j \leftarrow \infty$;

WHILE $O \neq \phi$ DO

Choose i in O; $O \leftarrow O - \{i\}$;

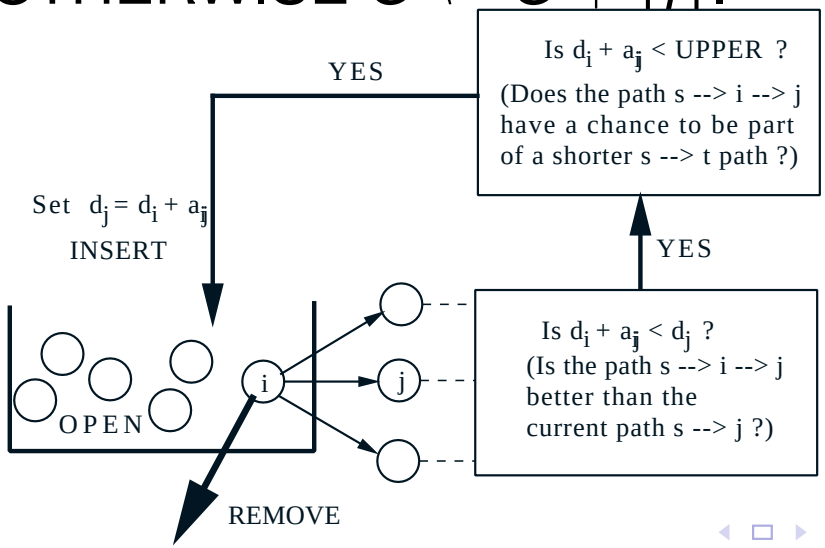
FOR EACH j such that $a_{ij} < \infty$ DO

IF $d_i + a_{ij} < \min(d_j, U)$ THEN

$d_j \leftarrow d_i + a_{ij}$; $v_j \leftarrow i$;

IF $j = t$ THEN $U \leftarrow \min(U, d_t)$

OTHERWISE $O \leftarrow O + \{j\}$.



Label correction algorithm (III)

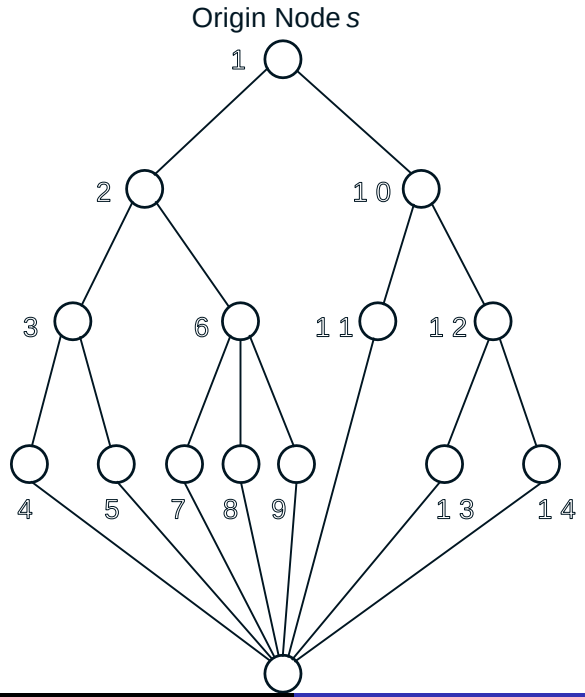
Strategies: one aims to reduce U (find good paths) rapidly, Keep O small (try to first remove the smallest d_j's), and do this with a minimum of overhead.

Proposition If it exists a path from 0 to t, the algorithm stop with $U = d_t = D_t$ and one has $v_j = v_j^*$ on the optimal path. Otherwise, the algorithm stops with $U = \infty$.

Choice of i in O

FIFO: O is managed as a waiting queue, first come first served. Bellman-Ford Algorithm. Search is width: the nodes are treated layer by layer. If the network is topologically ordered, one finds back the first accession procedure.

LIFO: O is managed as a stock: last come first served. Search in depth. One tries to reach t the fastest possible. Less memory usage than FIFO and reduce U faster.



Choice of i in O (II)

Smallest d_j first (Dijkstra).
O is sorted by considering the values d_j's. It is equivalent to Dijkstra algorithm, where O only contains the nodes of T that have a finite label.

D'Esposo-Pape method.
O is managed as a queue. When one inserts a node j in O, one put it at the beginning of the queue if it was previously in O, and at the end of the queue if it enters O for the first time.
SLF ("small-label-first"). When one inserts j in O, one puts it at the beginning of the queue O if its d_j is the than d_i, where i is the first node of O, and at the end of the queue otherwise.
One can combine it with LLL ("large-label-last"): each time that the label d_i of the first node i of O is greater than the average of the labels in O, one put i at the end of the queue.

Label correction with links of negative lengths

Assumption: no cycle of negative length.

Assume that a scalar u_j is known for every node j, that is a underestimation of the shortest distance from j to t (u_j can take the value $-\infty$ if no better underestimation in known).

The second step of the label correction algorithm is replaced by
FOR EACH j such that $a_{ij} < \infty$ DO
IF $d_i + a_{ij} < \min(d_j, U - u_j)$ THEN
$d_j \leftarrow d_i + a_{ij}$; $v_j \leftarrow i$;
IF $j = t$ THEN $U \leftarrow \min(U, d_t)$
OTHERWISE $O \leftarrow O + \{j\}$.

Infinite horizon

Infinite number of steps, stationary system: U_k, X_k, g_k, f_k, and $\mathbb{P}_k$ are the same for every k.

At step k, one observes the state x_k, and take a decision $u_k \in U(x_k)$, then a random variable w_k is generated according the law $\mathbb{P}(\cdot \mid x_k, u_k)$. One endures a cost of expectation $g(x_k, u_k)$, and the state at the next step is $x_{k+1} = f(x_k, u_k, w_k)$.

One has suppressed the parameter w_k from the function g. This is equivalent to the replacement of $g(x_k, u_k, w_k)$ by $\mathbb{E}[g(x_k, u_k, w_k)|x_k, u_k]$.

Total expected cost over infinite horizon

Discount factor $\alpha \leq 1$, for a policy $\pi = (\mu_0, \mu_1, \dots)$:

$$J_\pi(x_0) \quad = \quad \lim_{N \to \infty} \mathbb{E} \left[\sum_{k=0}^{N-1} \alpha^k g(x_k, u_k) \right]$$

where $u_k = \mu_k(x_k)$ and $x_{k+1} = f(x_k, u_k, w_k)$ for all k.
If the interest rate par step is r, then $\alpha = 1/(1+r)$.
Optimal total expected cost:

$$J^*(x_0) \quad = \quad \inf_\pi J_\pi(x_0).$$

Various conditions exist to guarantee the existence of a finite $J^*(x_0)$ and a stationary optimal policy.

Average cost per step

Given a policy $\pi = (\mu_0, \mu_1, \dots)$:

$$J_\pi(x_0) \quad = \quad \lim_{N\to\infty} \mathbb{E}\left[\frac{1}{N}\sum_{k=0}^{N-1} g(x_k, u_k)\right].$$

Average optimal cost:

$$J^*(x_0) \quad = \quad \inf_\pi J_\pi(x_0).$$

The average can only be non-zero is the total expected cost is infinite.

A policy π is said stationary if it is of the form $\pi = (\mu, \mu, \mu, \dots)$. In this case, one often drops the step index, and one speaks about the policy μ, and one denotes J_π by J_μ.

A stationary policy μ is optimal if $J_\mu(x) = J^*(x)$ for all x, and ϵ-optimal if $J_\mu(x) \leq J^*(x) + \epsilon$ for all x.

Truncated horizon

One can also consider an anticipated optimal cost for a model with truncated: $J_0(x) = 0$ and

$$\begin{aligned} J_k(x) &= \text{expected cost if we are in stage } x \\ &\quad \text{and it only remains } k \text{ steps} \\ &= \min_{u \in U(x)} \mathbb{E}_w \left[g(x,u) + \alpha J_{k-1}(f(x,u,w)) \right] \qquad \text{pour } k > 0. \end{aligned}$$

We expect that $J_k \to J^*$ when $k \to \infty$, and that

$$\mu^*(x) = \arg \min_{u \in U(x)} \mathbb{E}_w \left[g(x,u) + \alpha J^*(f(x,u,w)) \right]$$

define an optimal policy.

True under some conditions.

References

- D. P. Bertsekas, Dynamic Programming and Optimal Control, Vol. 1 et 2, Athena Scientific, Belmont, Mass., 2005 and 2007.
- W. B. Powell, Approximate Dynamic Programming, Second Edition, Wiley, New York, 2011. Available online: `http://onlinelibrary.wiley.com/book/10.1002/9781118029176`
- P. Glasserman, Monte Carlo Methods in Financial Engineering, Springer, 2003.
- ...