A Nondeterministic Dynamic Programming Model

1 Nondeterministic Dynamic Programming

A finite nondeterministic dynamic programming is defined by five-tuple:

$$\mathcal{N} \;=\; (\,N,\; X,\; \{U, U(\cdot)\},\; T,\; \{r, k, \beta\}\,),$$

where the definitions of each component are as follows.

1. $N(\geq 2)$ is an integer which means the *total number of stage*. The subscript n ranges $\{0, 1, \ldots, N\}$. It specifies the current number of stage.

2. X is a nonempty finite set which denotes a *state space*. Its elements $x_n \in X$ are called nth states. x_0 is an initial state and x_N is a terminal state.

3. U is a nonempty finite set which denotes an *action space*. Furthermore we also denote by U a mapping from X to 2^U and $U(x)$ is the set of all feasible actions for a state $x \in X$, where 2^Y denotes the following power set:

   $$2^Y = \{A | A \subset Y,\; A \neq \emptyset\}.$$

   After this, let $G_r(U)$ denote the graph of a mapping $U(\cdot)$:

   $$G_r(U) := \{(x, u) \,|\, u \in U(x),\; x \in X\} \subset X \times U.$$

4. $T : G_r(U) \to 2^X$ is a *nondeterministic transition law*. For each pair of a state and an action $(x, u) \in G_r(U)$, $T(x, u)$ means the set of all states appeared in the next stage. If an action u_n is chosen for a current state x_n, each $x_{n+1} \in T(x, u)$ will become a next state.

5. $r : G_r(U) \to R^1$ is a *reward function*, $k : X \to R^1$ is a *terminal reward function* and $\beta : G_r(T) \to [0, \infty)$ is a *weight function*. If an action u_n is chosen for a current state x_n, we get a reward $r(x_n, u_n)$ and each next state x_{n+1} will be appeared with a corresponding weight $\beta(x_n, u_n, x_{n+1})$ (≥ 0). For a terminal state x_N we get a terminal reward $k(x_N)$.

A mapping $\pi_n : X \to U$ $(n = 0, 1, \ldots, N-1)$ is called *nth decision function* if $\pi_n(x) \in U(x)$ for any $x \in X$. A sequence of decision functions:

$$\pi = \{\pi_0, \pi_1, \ldots \pi_{N-1}\}$$

is called a *Markov policy*. Let $\Pi(= \Pi(0))$ denotes the set of all Markov policies, which is called *Markov policy class*. If a decision-maker takes a Markov policy $\pi = \{\pi_0, \pi_1, \ldots \pi_{N-1}\}$, he chooses $\pi_n(x_n)$ $(\in U)$ for state x_n at nth stage. Then *total weighted value* is given by

$$\begin{aligned} V(x_0; \pi) \;\; := \;\; & r_0 + \sum_{x_1 \in X(1)} \beta_0 r_1 + \sum\sum_{(x_1, x_2) \in X(2)} \beta_0 \beta_1 r_2 + \cdots + \sum\sum \cdots \sum_{(x_1, \ldots, x_{N-1}) \in X(N-1)} \beta_0 \beta_1 \cdots \beta_{N-2} r_{N-1} \\ & + \sum\sum \cdots \sum_{(x_1, \ldots, x_N) \in X(N)} \beta_0 \beta_1 \cdots \beta_{N-1} k, \qquad x_0 \in X,\; \pi \in \Pi \qquad (1) \end{aligned}$$

where

$$r_n = r(x_n, \pi_n(x_n)), \quad \beta_n = \beta(x_n, \pi_n(x_n), x_{n+1}), \quad k = k(x_N),$$
$$X(m) = \{(x_1, \ldots, x_m) \in X \times \cdots \times X \,|\, x_{l+1} \in T(x_l, \pi_l(x_l)) \;\; 0 \leq l \leq m-1\,\}.$$

Thus the *nondeterministic dynamic programming problem* is formulated as a maximization problem :

$$\mathrm{P}_0(x_0) \qquad \text{Maximize} \quad V(x_0;\pi) \quad \text{subject to} \quad \pi \in \Pi.$$

The problem $\mathrm{P}_0(x_0)$ means an N-stage decision process starting at 0th stage with an initial state x_0.

Let $v_0(x_0)$ be the maximum value of $\mathrm{P}_0(x_0)$. A policy π^* is called *optimal* if

$$V(x_0;\pi^*) \geq V(x_0;\pi) \qquad \forall \pi \in \Pi,\ \ \forall x_0 \in X.$$

Similarly, we consider the $(N-n)$-stage process with a starting state $x_n (\in X)$ on nth stage. The Markov policy class for this process is

$$\Pi(n) = \{\pi = \{\pi_n, \pi_{n+1}, \dots \pi_{N-1}\} |\ \pi_l : X \to U,\ \pi_l(x) \in U(x),\ \ n \leq l \leq N-1\}.$$

Thus weighted value is given by

$$\begin{aligned} V_n(x_n;\pi) \ \ &:= \ \ r_n + \sum_{x_n \in X(n)} \beta_n r_{n+1} + \sum \sum_{(x_n, x_{n+1}) \in X(n+1)} \beta_n \beta_{n+1} r_{n+1} + \cdots \\ &\qquad + \sum \sum \cdots \sum_{(x_n, \dots, x_N) \in X(N)} \beta_n \beta_{n+1} \cdots \beta_{N-1} k, \qquad x_n \in X,\ \pi \in \Pi(n) \end{aligned}$$

where

$$X(m) \ = \ \{(x_n, \dots, x_m) \in X \times \cdots \times X \,|\, x_{l+1} \in T(x_l, \pi_l(x_l)),\ n \leq l \leq m-1\}.$$

Then for $n = 1, 2, \dots, N-1$ the *imbedded problem* is defined by

$$\mathrm{P}_n(x_n) \quad \text{Maximize} \quad V(x_n;\pi) \quad \text{subject to} \quad \pi \in \Pi(n),$$

and let $v_n(x_n)$ be the maximum value of $\mathrm{P}_n(x_n)$. For $n = N$ let $v_N(x_N) := k(x_N)$.

Then we have the following recursive equation:

Theorem 1 (nondeterministic)

$$\begin{aligned} v_N(x) \ &= \ k(x) \qquad x \in X, \\ v_n(x) \ &= \ \max_{u \in U(x)} \left[r(x,u) + \sum_{y \in T(x,u)} \beta(x,u,y) v_{n+1}(y) \right] \quad x \in X,\ 0 \leq n \leq N-1. \end{aligned}$$

Let $\pi_n^*(x) \in U(x)$ be a point which attains $v_n(x)$. Then we get the optimal Markov policy $\pi^* = \{\pi_0^*, \pi_1^*, \dots \pi_{N-1}^*\}$ in Markov class Π.

The following results are for other transition systems.

Corollary 1 (stochastic) *In case $\beta(x,u,y) = \beta \cdot p(y|x,u)$, $\beta \geq 0$ and $p = p(y|x,u)$ is a Markov transition law, $\mathrm{P}_0(x_0)$ is a stochastic dynamic programming problem. Then we have the following recursive equation:*

$$\begin{aligned} v_N(x) \ &= \ k(x) \qquad x \in X, \\ v_n(x) \ &= \ \max_{u \in U(x)} \left[r(x,u) + \beta \sum_{y \in T(x,u)} v_{n+1}(y) p(y|x,u) \right] \quad x \in X,\ 0 \leq n \leq N-1. \end{aligned}$$

Corollary 2 (deterministic) *In case $T(x,u)$ is a singleton, $\mathrm{P}_0(x_0)$ is a deterministic dynamic programming problem. Then we have the following recursive equation:*

$$\begin{aligned} v_N(x) \ &= \ k(x) \qquad x \in X, \\ v_n(x) \ &= \ \max_{u \in U(x)} [r(x,u) + \beta(x,u,T(x,u)) v_{n+1}(T(x,u))] \quad x \in X,\ 0 \leq n \leq N-1, \end{aligned}$$

where $\beta(x,u,\{y\}), v_n(\{y\})$ are equated with $\beta(x,u,y), v_n(y)$, respectively.

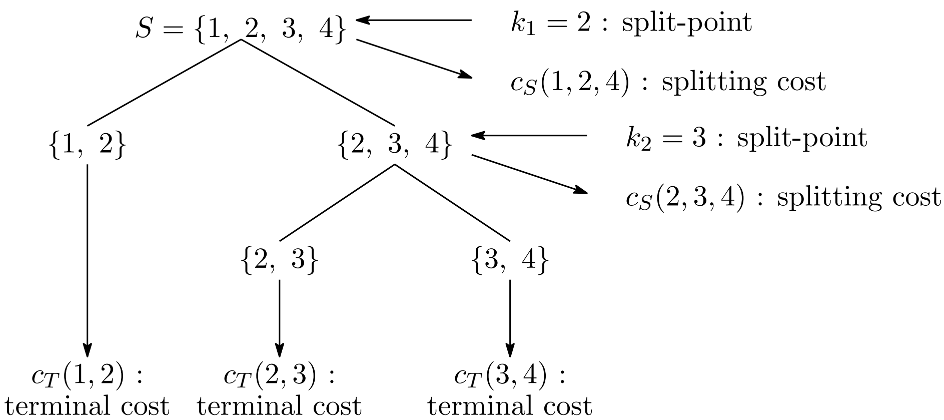

Figure 1:

2 Splitting Problem

In this section we formulate a splitting problem as a nondeterministic dynamic programming problem. An outline of the splitting problem is as follows.

Let S be an initial sequence $\{1, 2, \ldots, L\}$. We split S into two parts, both of which consist of consecutive numbers. The *split-point* belongs to both the parts. It costs $c_S(i, k, j)$ to split $\{i,\ i+1,\ \ldots,\ j\}$ into $\{i,\ i+1,\ \ldots,\ k\}$ and $\{k,\ k+1,\ \ldots,\ j\}$. We call c_S a *splitting cost function.* We continue splitting until any split part becomes a set of consecutive two numbers. It takes us $c_T(i, i+1)$ to reach terminal state $\{i, i+1\}$. We call c_T a *terminal cost function.* The problem is to find a sequence of splittings which minimizes the total sum of all splitting costs and of all terminal costs.

Example 1 Let $S = \{1, 2, 3, 4\}$ be an initial sequence. First we choose a split-point $k_1 = 2$. S is split into $\{1, 2\}$ and $\{2, 3, 4\}$ with splitting cost $c_S(1, 2, 4)$. Since $\{1, 2\}$ is a set of consecutive two numbers, it takes us terminal cost $c_T(1, 2)$. Next we choose a split-point $k_2 = 3$ for $\{2, 3, 4\}$. Then it is split into $\{2, 3\}$ and $\{3, 4\}$ with splitting cost $c_S(2, 3, 4)$. Finally it takes us terminal costs $c_T(2, 3)$ and $c_T(3, 4)$. Thus the total sum of costs for the strategy k_1, k_2 is

$$c_S(1, 2, 4) + c_S(2, 3, 4) + c_T(1, 2) + c_T(2, 3) + c_T(3, 4)$$

(see Fig. 1).

We consider the following nondeterministic dynamic programming problem:

$$\mathcal{N} \ = \ (\, L-2,\ X,\ \{U, U(\cdot)\},\ T,\ \{r, k, \beta\}\,),$$

where

$$\begin{aligned}
X &= \{\{i,\ i+1,\ \ldots,\ j\} \mid 1 \le i < j \le L\} \\
U &= \{2, 3, \ldots, L-1\} \\
U(x) &= \{i+1,\ i+2,\ \ldots,\ j-1\}, \quad x = \{i,\ i+1,\ \ldots,\ j\} \in X \\
T(x,u) &= \{\{i,\ \ldots,\ u\}, \{u,\ \ldots,\ j\}\}, \quad x = \{i,\ i+1,\ \ldots,\ j\} \in X,\ u \in U(x) \\
\beta(x,u,y) &= \begin{cases} 0 & x = \{i,\ i+1\} \\ 1 & \text{otherwise} \end{cases}, \qquad (x,u,y) \in G_r(T). \\
r(x,u) &= \begin{cases} c_T(i, i+1) & i+1 = j \\ c_S(i,k,j) & i+1 < j \end{cases}, \qquad (x,u) = (\{i, \ldots, j\}, k) \in G_r(U) \\
k(x) &= c_T(i, i+1), \qquad x = \{i, i+1\} \in X.
\end{aligned}$$

Furthermore let $v_0(x) = \cdots = v_{L-2}(x) = v(x)$, $x \in X$. Then the nondeterministic dynamic programming problem $\mathcal{N}$ expresses the splitting problem with an initial sequence (state) $x_0 = \{1, 2, \ldots, L\}$. The problem in Example 1 is interpreted as $\mathrm{P}_0(x_0)$ with $x_0 = S = \{1, 2, 3, 4\}$ and $N = 2$.

We give an application of the splitting problem in the next section.

3 Chained Matrix Products

We consider the problem on chained matrix products (see tutOR, http://www. tutor.ms.unimelb.edu.au/). When we compute the product of three matrices A, B and C, the result is independent of the product order, that is $A(BC) = (AB)C$. On the other hand the number of scalar products required for computing the product depends on the product order.

Example 2 Let A be $(r_A \times c_A)$-matrix, B $(r_B \times c_B)$-matrix and C $(r_C \times c_C)$-matrix $(c_A = r_B, c_B = r_C)$. The number of scalar products required for $A(BC)$ is not equal to that for $(AB)C$:

$$c_B \times (r_B \times c_C) + c_A \times (r_A \times c_C) \neq c_A \times (r_A \times c_B) + c_B \times (r_A \times c_C).$$

The purpose is to minimize the number of scalar products. We call this problem the chained matrix products problem.

We formulate the chained matrix products problem as a splitting problem defined in the previous section. Suppose that we have M matrices $A_1, A_2, \ldots, A_M$ to multiply and each matrix A_n has m_i rows and m_{i+1} columns. Then the splitting problem with

$$\begin{aligned}
&L = M + 1, \\
&c_S(i, k, j) = m_i m_k m_j, \\
&c_T(i, i + 1) = 0
\end{aligned}$$

denotes the chained matrix products problem. Hence we can get the optimal solutions by using Theorem 1.

Example 3 Let $M = 4, m_1 = 3, m_2 = 10, m_3 = 5, m_4 = 4$ and $m_5 = 16$. For example, $A_1(A_2(A_3A_4))$ involves

$$m_3m_4m_5 + m_2m_3m_5 + m_1m_2m_5 = 5 \cdot 4 \cdot 16 + 10 \cdot 5 \cdot 16 + 3 \cdot 10 \cdot 16 = 1600$$

scalar products. In the following, the recursive equation in Theorem 1 is applied to this problem.

First, since $c_T(i, i + 1) = 0, U(x) = \phi$ for $x = \{i, i + 1\} \in X$,

$$v(x) = 0, \quad x = \{i, i + 1\} \in X.$$

Next, since $r(\{i, j\}, k) = c_S(i, k, j) = m_i m_k m_j$ for $x = \{i, i + 1, \ldots, j\} \in X$ $(i + 1 < j)$,

$$\begin{aligned}
v(\{1, 2, 3\}) &= r(\{1, 2, 3\}, 2) + (v(\{1, 2\}) + v(\{2, 3\})) \\
&= m_1m_2m_3 + (0 + 0) = 150, \qquad \pi^*(\{1, 2, 3\}) = 2, \\
v(\{2, 3, 4\}) &= r(\{2, 3, 4\}, 3) + (v(\{2, 3\}) + v(\{3, 4\})) \\
&= m_2m_3m_4 + (0 + 0) = 200, \qquad \pi^*(\{2, 3, 4\}) = 3, \\
v(\{3, 4, 5\}) &= r(\{3, 4, 5\}, 4) + (v(\{3, 4\}) + v(\{4, 5\})) \\
&= m_3m_4m_5 + (0 + 0) = 320, \qquad \pi^*(\{3, 4, 5\}) = 4,
\end{aligned}$$

and

$$\begin{aligned}
v(\{1,2,3,4\}) &= \min\{r(\{1,2,3,4\},2) + (v(\{1,2\}) + v(\{2,3,4\})), \\
&\qquad\qquad r(\{1,2,3,4\},3) + (v(\{1,2,3\}) + v(\{3,4\}))\} \\
&= \min\{m_1 m_2 m_4 + (0 + 200), m_1 m_3 m_4 + (150 + 0)\} \\
&= \min\{120 + 200, 60 + 150\} = \min\{320, 210\} \\
&= 210, \qquad\qquad \pi^*(\{1,2,3,4\}) = 3,
\end{aligned}$$

$$\begin{aligned}
v(\{2,3,4,5\}) &= \min\{r(\{2,3,4,5\},3) + (v(\{2,3\}) + v(\{3,4,5\})), \\
&\qquad\qquad r(\{2,3,4,5\},4) + (v(\{2,3,4\}) + v(\{4,5\}))\} \\
&= \min\{1120, 840\} = 840, \qquad\qquad \pi^*(\{2,3,4,5\}) = 4.
\end{aligned}$$

Finally

$$\begin{aligned}
v(\{1,2,3,4,5\}) &= \min\{r(\{1,2,3,4,5\},2) + (v(\{1,2\}) + v(\{2,3,4,5\})), \\
&\qquad\qquad r(\{1,2,3,4,5\},3) + (v(\{1,2,3\}) + v(\{3,4,5\})), \\
&\qquad\qquad r(\{1,2,3,4,5\},4) + (v(\{1,2,3,4\}) + v(\{4,5\}))\} \\
&= \min\{1320, 710, 402\} = 402, \qquad\qquad \pi^*(\{1,2,3,4,5\}) = 4.
\end{aligned}$$

Thus we get the minimum of the number of scalar products $v(\{1,2,3,4,5\}) = 402$. The optimal sequence of splittings $\{k_1, k_2, k_3\}$ is given by

$$k_1 = \pi^*(\{1,2,3,4,5\}) = 4,\ \ k_2 = \pi^*(\{1,2,3,4\}) = 3,\ \ k_3 = \pi^*(\{1,2,3\}) = 2,$$

which means that the optimal product order is $((A_1A_2)A_3)A_4$.