Appendix D

Dynamic programming

D.1 The problem

Dynamic programming is an approach that allows systems of the form shown in figure D.1 to be optimized (for example this could be an unfolded dynamic control system). Dynamic programming selects the variables $x_1 \ldots x_n$ to minimize (or, with trivial modifications, maximize) the cost function

$$C = \sum_{i=1}^{n} c_i \tag{D.1}$$

where

$$y_{i+1} = T_i(x_i, y_i) \tag{D.2}$$

$$c_i = f_i(x_i, y_i) \tag{D.3}$$

Usually either y_1 or y_{n+1} is known beforehand. Note that the x_i and y_i can be scalars or vectors, and the c_i values are scalars.

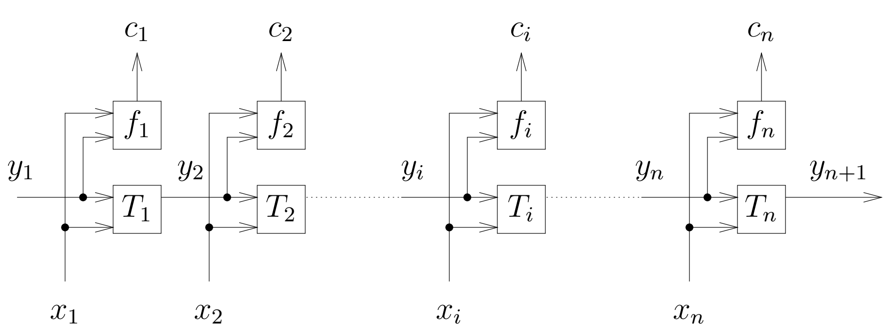

Figure D.1: *A system that can be optimized by dynamic programming.*

D.2 The solution

The system is decomposed using "Bellman's principle of optimality". This principle states that "an optimal solution is made up of optimal sub-solutions". This means that, if the variables $x_k \cdots x_n$ have been selected (with y_k fixed) to optimize the cost $\sum_{i=k}^{n} c_i$, then the variables $x_{k+1} \cdots x_n$ (with y_{k+1} keeping its same value) will be guaranteed to optimize the cost $\sum_{i=k+1}^{n} c_i$. This principle is implemented by defining the functions p and q:

- $p_i(y_i)$ is the value of x_i that is required for the given y_i to produce an optimal cost ($c_i + \cdots + c_n$), assuming that the variables $x_{i+1} \cdots x_n$ are chosen optimally.

- $q_i(y_i)$ is the cost $c_i + \cdots + c_n$ for the given y_i, assuming that $x_i \cdots x_n$ are chosen optimally.

Dynamic programming solves the problem backwards using recurrence relations. That is, given $p_{i+1}(y_{i+1})$ and $q_{i+1}(y_{i+1})$, $p_i(y_i)$ can be determined as the value of x_i that minimizes c_i plus the optimum subsequent cost $q_{i+1}(y_{i+1})$. The minimum cost[1] that is found is $q_i(y_i)$. Thus:

$$p_i(y_i) = \min_{(x_i)} \left[f_i(x_i, y_i) + q_{i+1}(T_i(x_i, y_i)) \right] \tag{D.4}$$

$$q_i(y_i) = f_i(p_i(y_i), y_i) + q_{i+1}(T_i(p_i(y_i), y_i)) \tag{D.5}$$

To use these recurrence relations, first define $q_{n+1}(y_{n+1})$. A backwards pass is made for $i = n \cdots 1$: first $p_i(y_i)$ is computed and then $q_i(y_i)$ is computed. Then a forward pass is made for $i = 1 \cdots n$: first $x_i = p_i(y_i)$ is computed and then $y_{i+1} = T(x_i, y_i)$ is computed. When the functions p and q are computed, what is actually calculated is a set of parameters that determine the functions. What these parameters actually represent will depend on the problem being solved. The $p(\cdot)$ parameters are needed on the forward pass, so they must be saved by the backwards pass. The $q(\cdot)$ parameters are not needed on the forward pass, so their storage can be re-used on the backwards pass.

D.3 A linear controller example

The above principles will now be applied to find the optimal control inputs for a simple discrete-time linear control system:

$$\mathbf{y}_{i+1} = A\mathbf{y}_i + Bx_i \tag{D.6}$$

$$c_i = (C\mathbf{y}_i - d_i)^2 + Dx_i^2 \tag{D.7}$$

where $\mathbf{y}_i$ is the $n_y \times 1$ system state vector, x_i is the (scalar) control force input and d_i is the desired (scalar) value for an element of $\mathbf{y}_i$ at each time step. The control goal specified by the cost function is to get an element of $\mathbf{y}_i$ to follow the reference d_i without making the control effort x_i too large. A and B are the system matrices, C is a $1 \times n_y$ vector that selects an element of $\mathbf{y}$, and D is a scalar constant that quantifies the relative importance of minimizing $|x_i|$. Define

$$q_{n+1} = (C\mathbf{y}_{n+1} - d_{n+1})^2 \tag{D.8}$$

and assume that the function q has the form

$$q_i(\mathbf{y}_i) = \mathbf{y}_i^T E_i \mathbf{y}_i + F_i \mathbf{y}_i \tag{D.9}$$

[1] Note that $\min_{(x)} f(x)$ is defined to be the value of x that minimizes $f(x)$.

where E_i is a $n_y \times n_y$ matrix and F_i is a $1 \times n_y$ vector. Thus:

$$\begin{aligned} p_i(\mathbf{y}_i) &= \min_{(x_i)} \Big[f_i(x_i, \mathbf{y}_i) + q_{i+1}(T_i(x_i, \mathbf{y}_i)) \Big] & \text{(D.10)} \\ &= \min_{(x_i)} \Big[(C\mathbf{y}_i - d_i)^2 + Dx_i^2 + & \text{(D.11)} \\ &\quad (A\mathbf{y}_i + Bx_i)^T E_{i+1}(A\mathbf{y}_i + Bx_i) + F_{i+1}(A\mathbf{y}_i + Bx_i) \Big] \\ &= Q_{i+1}\mathbf{y}_i + R_{i+1} & \text{(D.12)} \end{aligned}$$

where

$$\begin{aligned} Q_i &= \frac{-B^T E_i A}{D + B^T E_i B} & \text{(D.13)} \\ R_i &= -\frac{F_i B}{2(D + B^T E_i B)} & \text{(D.14)} \end{aligned}$$

Now,

$$\begin{aligned} q_i(\mathbf{y}_i) &= f_i(p_i(\mathbf{y}_i), \mathbf{y}_i) + q_{i+1}(T_i(p_i(\mathbf{y}_i), \mathbf{y}_i)) & \text{(D.15)} \\ &= (C\mathbf{y}_i - d_i)^2 + D(Q_{i+1}\mathbf{y}_i + R_{i+1})^2 + & \text{(D.16)} \\ &\quad (A\mathbf{y}_i + B(Q_{i+1}\mathbf{y}_i + R_{i+1}))^T E_{i+1}(A\mathbf{y}_i + B(Q_{i+1}\mathbf{y}_i + R_{i+1})) + \\ &\quad F_{i+1}(A\mathbf{y}_i + B(Q_{i+1}\mathbf{y}_i + R_{i+1})) \\ &= \mathbf{y}_i^T E_i \mathbf{y}_i + F_i \mathbf{y}_i & \text{(D.17)} \end{aligned}$$

where

$$\begin{aligned} E_i &= C^T C + DQ_{i+1}^T Q_{i+1} + ((A + BQ_{i+1})^T E_{i+1}(A + BQ_{i+1})) & \text{(D.18)} \\ F_i &= -2d_i C + 2DR_{i+1}Q_{i+1} + ((A + BQ_{i+1})^T E_{i+1}(BR_{i+1}))^T + & \text{(D.19)} \\ &\quad ((BR_{i+1})^T E_{i+1}(A + BQ_{i+1})) + F_{i+1}A + F_{i+1}BQ_{i+1} \end{aligned}$$

Thus the algorithm for computing the optimal control strategy is:

- Set $E_{n+1} = C^T C$ and $F_{n+1} = 2d_{n+1}C$.
- For $i = n \cdots 1$:
  - Set Q_{i+1} and R_{i+1} according to equation D.13 and equation D.14.
  - Set E_i and F_i according to equation D.18 and equation D.19.
- For $i = 1 \cdots n$:
  - Set $x_i = Q_{i+1}\mathbf{y}_i + R_{i+1}$
  - Compute y_{i+1} from equation D.6.

Despite the simplicity and power of the dynamic programming approach, it is not practical to directly implement it in a real controller. It is not an on-line algorithm because it requires a backwards pass before the control inputs can be computed. Also, it requires exact knowledge of the system dynamics (the matrices A and B).