Chapter 2 Dynamic Programming

2.1 Closed-loop optimization of discrete-time systems: inventory control

We consider the following **inventory control problem**:
The problem is to minimize the expected cost of ordering quantities of a certain product in order to meet a stochastic demand for that product. The ordering is only possible at discrete time instants $t_0 < t_1 < \cdots < t_{N-1}, N \in \mathbb{N}$.
We denote by x_k, u_k, and $w_k, 0 \leq k \leq N-1$, the available **stock**, the **order** and the **demand** at t_k. Here, $w_k, 0 \leq k \leq N-1$, are assumed to be independent random variables. Excess demand (giving rise to negative stock) is backlogged and filled as soon as additional inventory becomes available.
Consequently, the stock evolves according to the following **discrete-time system**

$$x_{k+1} \;=\; x_k \;+\; u_k \;-\; w_k \quad , \quad 0 \leq k \leq N-1 \; . \tag{2.1}$$

The **cost** at time t_k invokes a penalty $r(x_k)$ for either **holding cost** (excess inventory) or **shortage cost** (unsatisfied demand) and a **purchasing cost** in terms of a cost c per ordered unit. Moreover, there is **terminal cost** $R(x_N)$ for left over inventory at the final time instant t_{N-1}. Therefore, the **total cost** is given by

$$E\Big(R(x_N) \;+\; \sum_{k=0}^{N-1} \big(r(x_k) \;+\; cu_k\big)\Big) \; . \tag{2.2}$$

A natural constraint on the variables is $u_k \geq 0, 0 \leq k \leq N-1$.
We distinguish between **open-loop minimization** and **closed-loop minimization** of the cost:
In **open-loop minimization**, the decisions on the ordering policy $u_k, 0 \leq k \leq N-1$, are made once and for all at the initial time t_0 which leads to the minimization problem

$$\text{minimize } E\Big(R(x_N) \;+\; \sum_{k=0}^{N-1} \big(r(x_k) \;+\; cu_k\big)\Big) \; , \tag{2.3a}$$

$$\text{subject to } x_{k+1} \;=\; x_k \;+\; u_k \;-\; w_k \quad , \quad 0 \leq k \leq N-1 \; , \tag{2.3b}$$

$$u_k \geq 0 \; , \; 0 \leq k \leq N-1 \; . \tag{2.3c}$$

In **closed-loop minimization**, the orders $u_k, k > 0$, are placed at the last possible moment, i.e., at t_k, to take into account information about the development of the stock until t_k. In mathematical terms, this amounts to determine a **policy** or **control rule** $\pi := \{\mu_k\}_{k=0}^{N-1}$

described by functions $\mu_k = \mu_k(x_k), 0 \leq k \leq N-1$, where the function value is the amount of units of the product to be ordered at time t_k, if the stock is x_k. Hence, given a fixed initial stock x_0, the associated minimization problem reads as follows

$$\text{(2.4a)} \quad \text{minimize } J_\pi(x_0) \;:=\; E\Big(R(x_N) \;+\; \sum_{k=0}^{N-1} \big(r(x_k) \;+\; c\mu_k(x_k)\big)\Big) \;,$$

$$\text{(2.4b)} \quad \text{subject to } x_{k+1} \;=\; x_k \;+\; \mu_k(x_k) \;-\; w_k \quad , \quad 0 \leq k \leq N-1 \;,$$

$$\text{(2.4c)} \qquad \mu_k(x_k) \geq 0 \;,\; 0 \leq k \leq N-1 \;.$$

Dynamic Programming (DP) is concerned with the efficient solution of such closed-loop minimization problems.

Remark: We note that minimization problems associated with deterministic discrete-time dynamical systems can be considered as well. Such systems will be dealt with in more detail in Chapter 2.3.

2.2 Dynamic Programming for discrete-time systems

2.2.1 Setting of the problem

For $N \in \mathbb{N}$, we consider sequences $\{S_k\}_{k=0}^N, \{C_k\}_{k=0}^{N-1}$, and $\{D_k\}_{k=0}^{N-1}$ of (random) **state spaces** $S_k, 0 \leq k \leq N$, **control spaces** $C_k, 0 \leq k \leq N-1$, and (random) **disturbance spaces** $D_k, 0 \leq k \leq N-1$.
Given an **initial state** $x_0 \in S_0$, we assume that the states $x_k \in S_k, 0 \leq k \leq N$, evolve according to the **discrete-time dynamic system**

$$x_{k+1} \;=\; f_k(x_k, u_k, w_k) \quad , \quad 0 \leq k \leq N-1 \;,$$

where $f_k : S_k \times C_k \times D_k \to S_{k+1}, 0 \leq k \leq N-1$. The controls u_k satisfy

$$u_k \in U_k(x_k) \subset C_k \quad , \quad 0 \leq k \leq N-1 \;,$$

i.e., they depend on the state $x_k \in S_k$, and the random disturbances $w_k, 0 \leq k \leq N-1$, are characterized by a **probability distribution**

$$P_k(\cdot|x_k, u_k) \quad , \quad 0 \leq k \leq N-1 \;,$$

which may explicitly depend on the state x_k and on the control u_k, but not on the prior disturbances $w_j, 0 \leq j \leq k-1$.
At the time instants $0 \leq k \leq N-1$, **decisions** with respect to the choice of the controls have to be made by means of **control laws**

$$\mu_k : S_k \to S_k \quad , \quad u_k \;=\; \mu_k(x_k) \;,\; 0 \leq k \leq N-1 \;,$$

leading to a **control policy**

$$\pi \;=\; \{\mu_0, \mu_1, \cdots, \mu_{N-1}\} \;.$$

We refer to Π as the set of **admissible control policies**.
Finally, we introduce **cost functionals**

$$g_k : S_k \times C_k \times D_k \to \mathbb{R} \quad , \quad 0 \leq k \leq N-1 \;,$$

associated with the decisions at time instants $0 \leq k \leq N-1$, and a **terminal cost**

$$g_N : S_N \to \mathbb{R} \;.$$

We consider the **minimization problem**

$$\min_{\pi \in \Pi} J_\pi(x_0) \;=\; E\,[g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k)\,] \;, \tag{2.5a}$$

subject to

$$x_{k+1} \;=\; f_k(x_k, \mu_k(x_k), w_k) \quad , \quad 0 \leq k \leq N-1 \;, \tag{2.5b}$$

where the expectation E is taken over all random states and random disturbances.
For a given initial state x_0, the value

$$J^*(x_0) \;=\; \min_{\pi\in\Pi} J_\pi(x_0) \tag{2.6}$$

is referred to as the **optimal cost function** or the **optimal value function**. Moreover, if there exists an admissible policy $\pi^* \in \Pi$ such that

$$J_{\pi^*}(x_0) \;=\; J^*(x_0)\ , \tag{2.7}$$

then π^* is called an **optimal policy**.

2.2.2 Bellman's principle of optimality

Bellman's celebrated **principle of optimality** is the key for a constructive approach to the solution of the minimization problem (2.5) which readily leads to a powerful algorithmic tool: the **Dynamic Programming algorithm** (DP algorithm).
For **intermediate states** $x_k \in S_k, 1 \le k \le N-1$, that occur with positive probability, we consider the **minimization subproblems**

$$\min_{\pi_k\in\Pi_k} J_{\pi_k}(x_0) \;=\; E\,[g_N(x_N) + \sum_{\ell=k}^{N-1} g_\ell(x_\ell, \mu_\ell(x_\ell), w_\ell)\ ]\ , \tag{2.8a}$$

subject to

$$x_{\ell+1} \;=\; f_\ell(x_\ell, \mu_\ell(x_\ell), w_\ell) \quad , \quad k \le \ell \le N-1\ , \tag{2.8b}$$

where $\pi_k = \{\mu_k, \mu_{k+1}, \cdots, \mu_{N-1}\}$ and Π_k is the set of admissible policies obtained from Π by deleting the admissible control laws associated with the previous time instants $0 \le \ell \le k-1$.
Bellman's optimality principle states that if

$$\pi^* \;=\; \{\mu_0^*, \mu_1^*, \cdots, \mu_{N-1}^*\}$$

is an optimal policy for (2.5), then the **truncated policy**

$$\pi_k^* \;=\; \{\mu_k^*, \mu_{k+1}^*, \cdots, \mu_{N-1}^*\}$$

is an optimal policy for the minimization subproblem (2.8). We refer to

$$J_k^*(x_k) \;=\; J_{\pi_k^*}(x_k) \tag{2.9}$$

as the **optimal cost-to-go** for state x_k at the time instant k to the final time N. For completeness, we further set $J_N^*(x_N) = g_N(x_N)$.
The **intuitive justification** for Bellman's optimality principle is that if the truncated policy π_k^* were not optimal for subproblem (2.8), then

we would be able to reduce the optimal cost for (2.5) by switching to the optimal policy for (2.8) once x_k is reached.

2.2.3 The DP algorithm and its optimality

Bellman's optimality principle strongly suggests to solve the optimality subproblems (2.8) backwards in time, beginning with the terminal cost at final time instant N and then recursively compute the optimal cost-to-go for subproblems $k = N-1, \cdots, 0$. This leads to the so-called **backward DP algorithm** which is characterized by the **recursions**

$$J_N(x_N) = g_N(x_N) \ , \tag{2.10a}$$

$$J_k(x_k) = \min_{u_k \in U_k(x_k)} E_{w_k}[g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k))] \ , \ 0 \le k \le N-1 \ , \tag{2.10b}$$

where E_{w_k} means that the expectation is taken with respect to the probability distribution of w_k.

Theorem 2.1 (Optimality of the backward DP algorithm)
We assume that

- the random disturbance spaces $D_k, 0 \le k \le N-1$, are finite or countable sets,
- the expectations of all terms in the cost functionals in (2.10b) are finite for every admissible policy.

Then, there holds

$$J_k^*(x_k) = J_k(x_k) \quad , \quad 0 \le k \le N \ . \tag{2.11}$$

Moreover, if there exist optimal policies $\mu_k^*, 0 \le k \le N-1$, for (2.10b) such that $u_k^* = \mu_k^*(x_k)$, then $\pi^* = \{\mu_0^*, \mu_1^*, \cdots, \mu_{N-1}^*\}$ is an optimal policy for (2.5).

Proof. We note that for $k = N$ the assertion (2.11) holds true by definition. For $k < N$ and $\varepsilon > 0$, we define $\mu_k^\varepsilon(x_k) \in U_k(x_k)$ as the ε-suboptimal control satisfying

$$E_{w_k}[g_k(x_k, \mu_k^\varepsilon(x_k), w_k) + J_{k+1}(f_k(x_k, \mu_k^\varepsilon(x_k), w_k))] \le J_k(x_k) + \varepsilon \ . \tag{2.12}$$

We denote by $J_k^\varepsilon(x_k)$ the expected cost at state x_k and time instant k with respect to the ε-optimal policy

$$\pi_k^\varepsilon = \{\mu_k^\varepsilon(x_k), \mu_{k+1}^\varepsilon(x_{k+1}), \cdots, \mu_{N-1}^\varepsilon(x_{N-1})\}.$$

We show by induction on $N-k, k \geq 1$, that

$$J_k(x_k) \leq J_k^\varepsilon(x_k) \leq J_k(x_k) + (N-k)\varepsilon \ , \tag{2.13a}$$

$$J_k^*(x_k) \leq J_k^\varepsilon(x_k) \leq J_k^*(x_k) + (N-k)\varepsilon \ , \tag{2.13b}$$

$$J_k(x_k) = J_k^*(x_k) \ . \tag{2.13c}$$

(i) Begin of the induction $\mathbf{k = N-1}$: It follows readily from (2.12) that (2.13a) and (2.13b) hold true for $k = N-1$. The limit process $\varepsilon \to 0$ in (2.13a) and (2.13b) then yields that (2.13c) is satisfied for $k = N-1$.

(ii) Induction hypothesis: We suppose that (2.13a)-(2.13c) hold true for some $k+1$.

(iii) Proof for k : Using the definition of $J_k^\varepsilon(x_k)$, the induction hypothesis and (2.12), we get

$$\begin{aligned} &J_k^\varepsilon(x_k) = E_{w_k}[g_k(x_k, \mu_k^\varepsilon(x_k), w_k) + J_{k+1}^\varepsilon(f_k(x_k, \mu_k^\varepsilon(x_k), w_k))] \leq \\ &\leq E_{w_k}[g_k(x_k, \mu_k^\varepsilon(x_k), w_k) + J_{k+1}(f_k(x_k, \mu_k^\varepsilon(x_k), w_k))] + (N-k-1)\varepsilon \\ &\leq J_k(x_k) + \varepsilon + (N-k-1)\varepsilon = J_k(x_k) + (N-k)\varepsilon \ . \end{aligned}$$

On the other, using the induction hypothesis once more, we obtain

$$\begin{aligned} &J_k^\varepsilon(x_k) = E_{w_k}[g_k(x_k, \mu_k^\varepsilon(x_k), w_k) + J_{k+1}^\varepsilon(f_k(x_k, \mu_k^\varepsilon(x_k), w_k))] \geq \\ &\geq E_{w_k}[g_k(x_k, \mu_k^\varepsilon(x_k), w_k) + J_{k+1}(f_k(x_k, \mu_k^\varepsilon(x_k), w_k))] \geq \\ &\geq \min_{u_k \in U_K(x_k)} E_{w_k}[g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k))] = J_k(x_k) \ . \end{aligned}$$

The combination of the two preceding inequalities proves (2.13a) for k. Now, for every policy $\pi \in \Pi$, using the induction hypothesis and (2.12), we have

$$\begin{aligned} &J_k^\varepsilon(x_k) = E_{w_k}[g_k(x_k, \mu_k^\varepsilon(x_k), w_k) + J_{k+1}^\varepsilon(f_k(x_k, \mu_k^\varepsilon(x_k), w_k))] \leq \\ &\leq E_{w_k}[g_k(x_k, \mu_k^\varepsilon(x_k), w_k) + J_{k+1}(f_k(x_k, \mu_k^\varepsilon(x_k), w_k))] + (N-k-1)\varepsilon \\ &\leq \min_{u_k \in U_k(x_k)} E_{w_k}[g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k))] + (N-k)\varepsilon \leq \\ &\leq E_{w_k}[g_k(x_k, \mu_k(x_k), w_k) + J_{\pi_{k+1}}(f_k(x_k, \mu_k(x_k), w_k))] + (N-k)\varepsilon = \\ &= J_{\pi_k}(x_k) + (N-k)\varepsilon \ . \end{aligned}$$

Minimizing over π_k yields

$$J_k^\varepsilon(x_k) \leq J_k^*(x_k) + (N-k)\varepsilon \ .$$

On the other hand, by the definition of $J_k^*(x_k)$ we obviously have

$$J_k^*(x_k) \leq J_k^\varepsilon(x_k) \ .$$

Combining the preceding inequalities proves (2.13b) for k.
Again, (2.13c) finally follows from $\varepsilon \to 0$ in (2.13a) and (2.13b). $\square$

Remark: The general case (where the disturbance spaces are not necessarily finite or countable) requires additional structures of the spaces S_k, C_k, and D_k as well as measurability assumptions with regard to the functions f_k, g_k, and μ_k within the framework of measure-theoretic probability theory. We refer to [2] for details.

2.2.4 Applications of the DP algorithm

Example (Inventory control): We consider the following modification of the inventory control problem previously discussed in Chapter 2.1:
We assume that the states x_k, the controls u_k, and the demands w_k are non-negative integers which can take the values $0, 1$ and 2 where the demand w_k has the same probability distribution

$$p(w_k = 0) = 0.1 \quad , \quad p(w_k = 1) = 0.7 \quad , \quad p(w_k = 2) = 0.2$$

for all planning periods $(k, k+1)$.
We further assume that the excess demand $w_k - x_k - u_k$ is lost and that there is an upper bound of 2 units on the stock that can be stored. Consequently, the equation for the evolution of the stock takes the form

$$x_{k+1} \;=\; \max(0, x_k + u_k - w_k)$$

under the constraint

$$x_k \;+\; u_k \;\leq\; 2 \; .$$

For the holding costs and the terminal cost we assume

$$r(x_k) \;=\; (x_k + u_k - w_k)^2 \quad , \quad R(x_n) \;=\; 0 \; ,$$

and we suppose that the ordering cost is 1 per unit, i.e.,

$$c \;=\; 1 \; .$$

Therefore, the functions $g_k, 0 \leq k \leq N$, are given by

$$\begin{aligned} g_k(x_k, u_k, w_k) \;&=\; u_k + (x_k + u_k - w_k)^2 \quad , \quad 0 \leq k \leq N-1 \; , \\ g_N(x_N) \;&=\; 0 \; . \end{aligned}$$

Finally, we suppose $x_0 = 0$ and for the planning horizon we take $N = 3$ so that the recursions (2.8) of the backward DP algorithm have the form

$$\text{(2.14a)} \qquad J_3(x_3) = \; 0 \; ,$$

$$\text{(2.14b)} \qquad J_k(x_k) = \min_{0 \leq u_k \leq 2 - x_k} E_{w_k}[u_k + (x_k + u_k - w_k)^2 + \\ + \; J_{k+1}(\max(0, x_k + u_k - w_k))] \; , \; 0 \leq k \leq 2 \; .$$

We start with period 2, followed by period 1, and finish with period 0:

Period 2: We compute $J_2(x_2)$ for each possible value of the state x_2:

(i) $\mathbf{x_2 = 0}$: We have

$$\begin{aligned} J_2(0) &= \min_{u_2 \in \{0,1,2\}} E_{w_2}[u_2 + (u_2 - w_2)^2] = \\ &= \min_{u_2 \in \{0,1,2\}} [u_2 + 0.1u_2^2 + 0.7(u_2 - 1)^2 + 0.2(u_2 - 2)^2] \ . \end{aligned}$$

The computation of the right-hand side for the three different values of the control u_2 yields

$$\begin{aligned} u_2 = 0 \ &: \ E[\cdot] = 0.7 \cdot 1 + 0.2 \cdot 4 = 1.5 \ , \\ u_2 = 1 \ &: \ E[\cdot] = 1 + 0.1 \cdot 1 + 0.2 \cdot 1 = 1.3 \ , \\ u_2 = 2 \ &: \ E[\cdot] = 2 + 0.1 \cdot 4 + 0.7 \cdot 1 = 3.1 \ . \end{aligned}$$

Hence, we get

$$J_2(0) = 1.3 \quad , \quad \mu_2^*(0) = 1 \ .$$

(ii) $\mathbf{x_2 = 1}$: In view of the constraint $x_2 + u_2 \leq 2$ only $u_2 \in \{0, 1\}$ is admissible. Thus, we obtain

$$\begin{aligned} J_2(1) &= \min_{u_2 \in \{0,1\}} E_{w_2}[u_2 + (1 + u_2 - w_2)^2] = \\ &= \min_{u_2 \in \{0,1\}} [u_2 + 0.1(1 + u_2)^2 + 0.7u_2^2 + 0.2(u_2 - 1)^2] \ . \end{aligned}$$

The computation of the right-hand side gives

$$\begin{aligned} u_2 = 0 \ &: \ E[\cdot] = 0.1 \cdot 1 + 0.2 \cdot 1 = 0.3 \ , \\ u_2 = 1 \ &: \ E[\cdot] = 1 + 0.1 \cdot 4 + 0.7 \cdot 1 = 2.1 \ . \end{aligned}$$

It follows that

$$J_2(1) = 0.3 \quad , \quad \mu_2^*(1) = 0 \ .$$

(iii) $\mathbf{x_2 = 2}$: Since the only admissible control is $u_2 = 0$, we get

$$J_2(2) = E_{w_2}[(2 - w_2)^2] = 0.1 \cdot 4 + 0.7 \cdot 1 = 1.1 \ ,$$

whence

$$J_2(2) = 1.1 \quad , \quad \mu_2^*(2) = 0 \ .$$

Period 1: Again, we compute $J_1(x_1)$ for each possible value of the state x_1 taking into account the previously computed values $J_2(x_2)$:

(i) $\mathbf{x_1 = 0}$: We get

$$J_1(0) = \min_{u_1 \in \{0,1,2\}} E_{w_1}[u_1 + (u_1 - w_1)^2 + J_2(\max(0, u_1 - w_1))] \ .$$

The computation of the right-hand side results in

$$\begin{aligned} u_1 = 0 \ : \ E[\cdot] &= 0.1 \cdot J_2(0) + 0.7 \cdot (1 + J_2(0)) + \\ &\quad +0.2 \cdot (4 + J_2(0)) \ = \ 2.8 \ , \\ u_1 = 1 \ : \ E[\cdot] &= 1 + 0.1 \cdot (1 + J_2(1)) + 0.7 \cdot J_2(0) + \\ &\quad +0.2 \cdot (1 + J_2(0)) \ = \ 2.5 \ , \\ u_1 = 2 \ : \ E[\cdot] &= 2 + 0.1 \cdot (4 + J_2(2)) + 0.7 \cdot (1 + J_2(1)) + \\ &\quad +0.2 \cdot J_2(0) \ = \ 3.68 \ . \end{aligned}$$

We thus obtain

$$J_1(0) \ = \ 2.5 \quad , \quad \mu_1^*(0) \ = \ 1 \ .$$

(ii) $\mathbf{x_1 = 1}$: We have

$$J_1(1) \ = \ \min_{u_1 \in \{0,1\}} E_{w_1}[u_1 + (1 + u_1 - w_1)^2 + J_2(\max(0, 1 + u_1 - w_1))] \ .$$

Evaluation of the right-hand side yields

$$\begin{aligned} u_1 = 0 \ : \ E[\cdot] &= 0.1 \cdot (1 + J_2(1)) + 0.7 \cdot J_2(0) + \\ &\quad +0.2 \cdot (1 + J_2(0)) \ = \ 1.5 \ , \\ u_1 = 1 \ : \ E[\cdot] &= 1 + 0.1 \cdot (4 + J_2(2)) + 0.7 \cdot (1 + J_2(1)) + \\ &\quad +0.2 \cdot J_2(0) \ = \ 2.68 \ . \end{aligned}$$

Consequently, we get

$$J_1(1) \ = \ 1.5 \quad , \quad \mu_1^*(1) \ = \ 0 \ .$$

(iii) $\mathbf{x_1 = 2}$: Since the only admissible control is $u_1 = 0$, it follows that

$$\begin{aligned} J_1(2) &= E_{w_1}[(2 - w_1)^2 + J_2(\max(0, 2 - w_1))] \ = \\ &= 0.1 \cdot (4 + J_2(2)) + 0.7 \cdot (1 + J_2(1)) + 0.2 \cdot J_2(0) \ = \ 1.68 \ , \end{aligned}$$

whence

$$J_1(2) \ = \ 1.68 \quad , \quad \mu_1^*(2) \ = \ 0 \ .$$

Period 0: We only need to compute $J_0(0)$, since the initial state is $x_0 = 0$. We obtain

(i) $\mathbf{x_1 = 0}$: We get

$$J_0(0) \ = \ \min_{u_0 \in \{0,1,2\}} E_{w_0}[u_0 + (u_0 - w_0)^2 + J_1(\max(0, u_0 - w_0))] \ .$$

Computation of the right-hand side gives

$$\begin{aligned}
u_0 = 0 \ : \ E[\cdot] \ &= \ 0.1 \cdot J_1(0) + 0.7 \cdot (1 + J_1(0)) + \\
&\quad +0.2 \cdot (4 + J_1(0)) \ = \ 4.0 \ , \\
u_0 = 1 \ : \ E[\cdot] \ &= \ 1 + 0.1 \cdot (1 + J_1(1)) + 0.7 \cdot J_1(0) + \\
&\quad +0.2 \cdot (1 + J_1(0)) \ = \ 3.7 \ , \\
u_0 = 2 \ : \ E[\cdot] \ &= \ 2 + 0.1 \cdot (4 + J_1(2)) + 0.7 \cdot (1 + J_1(1)) + \\
&\quad +0.2 \cdot J_1(0) \ = \ 4.818 \ .
\end{aligned}$$

It follows that

$$J_0(0) \ = \ 3.7 \quad , \quad \mu_0^*(0) \ = \ 1 \ .$$

Conclusion: The optimal ordering policy is to order one unit for each period, if the stock is 0, and to order nothing, otherwise.

The results are visualized in Fig. 2.1-2.3 and Table 2.1. The numbers next to the arcs represent the transition probabilities.

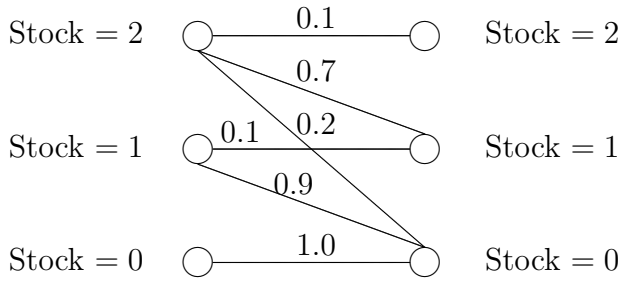

Fig. 2.1. Inventory control (purchased stock = 0)

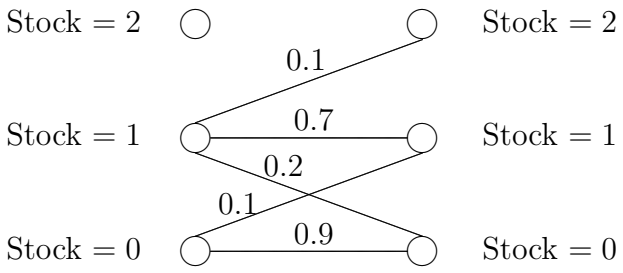

Fig. 2.2. Inventory control (purchased stock = 1)

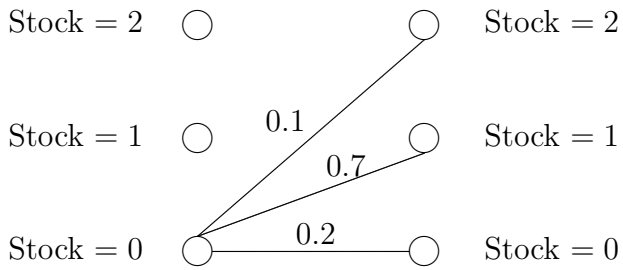

Fig. 2.3. Inventory control (purchased stock = 2)

Stock	Stage 0 Cost	Stage 0 Purchase	Stage 1 Cost	Stage 1 Purchase	Stage 2 Cost	Stage 2 Purchase
0	3.7	1	2.5	1	1.3	1
1	2.7	0	1.5	0	0.3	0
2	2.818	0	1.68	0	1.1	0

Table 2.1: Inventory control

Table 2.1 displays the optimal cost-to-go and optimal stock to be purchased for the three stages of the inventory control. Observe that due to the constraint $x_k + u_k \leq 2$, the control $u = 1$ is not available at state 2, whereas the control $u = 2$ is not available at states 1 and 2.

Example (Optimal chess match strategy): We consider a chess match between two players A and B according to the following rules:

- If a game has a winner, the winner gets 1 point and the loser gets 0 point,
- In case of a draw, both players get 1/2 point,
- If the score is $1-1$ at the end of the first two games, the match goes into sudden death, i.e., the players continue to play until the first time one of them wins a game.

We take the position of player A who has two playing strategies which he can choose independently at each game:

- **Timid play** with which he draws with probability $p_d > 0$ and looses with probability $1 - p_d$,
- **Bold play** which with he wins with probability $p_w < p_d$ and looses with probability $1 - p_w$.

Remark: We note that timid play never wins, whereas bold play never draws.

We will formulate a DP algorithm to determine the policy that maximizes player's A probability to win the match.
For that purpose, we consider an N-game match and define the **state** as the **net score**, i.e., the difference between the points of player A and the points of player B (e.g., a net score 0 indicates an even score). We further define T_k and B_k as the two strategies:

- $\mathbf{T_k}$: Timid play which keeps the score at x_k with probability p_d and changes x_k to $x_k - 1$ with probability $1 - p_d$,
- $\mathbf{B_k}$: Bold play which changes x_k to $x_k + 1$ with probability p_w or to $x_k - 1$ with probability $1 - p_w$.

The **DP recursion** begins with

$$J_N(x_N) = \begin{cases} 1 \ , \ x_N > 0 \\ p_w \ , \ x_N = 0 \\ 0 \ , \ x_N < 0 \end{cases} . \tag{2.15}$$

Note that $J_N(0) = p_w$, since due to the sudden death rule it is optimal to play bold, if the score is even after N games.
The **optimal cost-to-go function** at the beginning of the k-th game is given by the DP recursion

$$\begin{aligned} J_k(x_k) = \max_{T_k, B_k} [& p_d J_{k+1}(x_k) + (1-p_d) J_{k+1}(x_k - 1), \\ & p_w J_{k+1}(x_k + 1) + (1-p_w) J_{k+1}(x_k - 1)] \ . \end{aligned} \tag{2.16}$$

We remark that it is optimal to play bold, if

$$p_w J_{k+1}(x_k+1) + (1-p_w) J_{k+1}(x_k-1) \;\geq \tag{2.17}$$
$$\geq\; p_d J_{k+1}(x_k) + (1-p_d) J_{k+1}(x_k-1)\;.$$

or equivalently, if

$$\frac{p_w}{p_d} \;\geq\; \frac{J_{k+1}(x_k) - J_{k+1}(x_k-1)}{J_{k+1}(x_k+1) - J_{k+1}(x_k-1)}\;. \tag{2.18}$$

Using the recursions (2.15) and (2.16), we find

$$\begin{aligned}
J_{N-1}(x_{N-1}) &= 1 \text{ for } x_{N-1} > 1 \quad \text{Optimal play: either} \\
\\
J_{N-1}(1) &= \max(p_d + (1-p_d)p_w, p_w + (1-p_w)p_w) \\
&= p_d + (1-p_d)p_w \quad \text{Optimal play: timid} \\
\\
J_{N-1}(0) &= p_w \qquad \text{Optimal play: bold} \\
\\
J_{N-1}(-1) &= p_w^2 \qquad \text{Optimal play: bold} \\
\\
J_{N-1}(x_{N-1}) &= 0 \text{ for } x_{N-1} < -1 \quad \text{Optimal play: either}
\end{aligned}$$

Moreover, given $J_{N-1}(x_{N-1})$, and taking (2.16) and (2.18) into account, we find

$$\begin{aligned}
J_{N-2}(0) &= \max(p_d p_w + (1-p_d)p_w^2, p_w(p_d + \\
&\qquad +(1-p_d)p_w) + (1-p_w)p_w^2) \;= \\
&= p_w(p_w + (p_w+p_d)(1-p_w))\;,
\end{aligned}$$

i.e., if the score is even with two games still to go, the optimal policy is to play bold.

Conclusion: For a **2-game match**, the optimal policy for player A is to play timid if and only if player A is ahead in the score. The regions of pairs (p_w, p_d) for which player A has a better than $50-50$ chance to win the match is

$$\{(p_w,p_d) \in [0,1]^2 \mid J_0(0) = p_w(p_w + (p_w+p_d)(1-p_w)) > 1/2\}\;. \tag{2.19}$$

2.3 Finite-state systems and shortest path problems

2.3.1 The backward DP algorithm

We consider the following deterministic discrete-time system

$$x_{k+1} = f_k(x_k, u_k) \quad , \quad 0 \leq k \leq N-1 \ , \tag{2.20a}$$

$$u_k = \mu_k(x_k) \ , \tag{2.20b}$$

where x_0 is a fixed initial state and the state spaces $S_k, 1 \leq k \leq N$, are finite sets with card $S_k = n_k$. The control u_k provides a transition from the state $x_k \in S_k$ to the state $f_k(x_k, u_k) \in S_{k+1}$ at cost $g_k(x_k, u_k)$. Such a system is referred to as a **deterministic finite-state system**. We call $k \in \{0, \cdots, N\}$ the **stage** of the finite-state system. In particular, $k = 0$ is said to be the **initial stage** and $k = N$ is called the **final stage**. A finite-state system can be described by a **graph** consisting of **nodes** representing states and **arcs** representing transitions between states at successive stages. In particular, the nodes consist of an **initial node** s_0, **nodes** $s_i, 1 \leq i \leq n_k$, at stages $1 \leq k \leq N$, and a **terminal node** s_{N+1}. Each node $s_i, 1 \leq i \leq n_N$, representing a state x_N at the final stage N is connected to the terminal node s_{N+1} by an arc representing the **terminal cost** $g_N(x_N)$ (cf. Fig. 2.4).

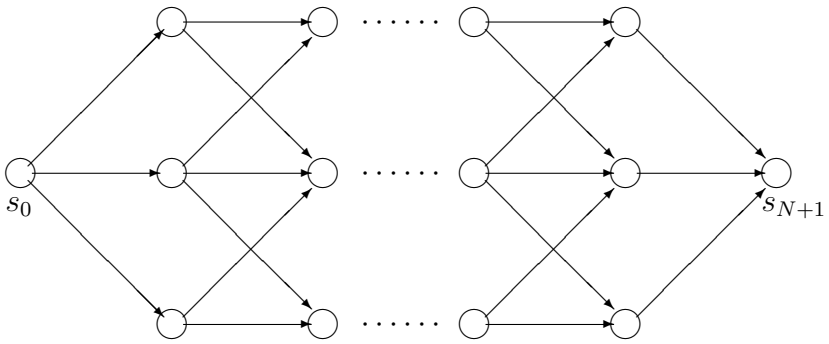

Fig. 2.4. Transition graph of a deterministic finite-state system

We denote by a_{ij}^k the transition cost for the transition from the state $x_i \in S_k$ to the state $x_j \in S_{k+1}$ with the convention that $a_{ij}^k = +\infty$, if there is no control that provides a transition from $x_i \in S_k$ to $x_j \in S_{k+1}$. The **DP algorithm** for the computation of the **optimal cost** is given

by

(2.21a)

$$J_N(s_i) = a^N_{i,s_{N+1}} \quad , \quad 1 \le i \le n_N \ ,$$

(2.21b)

$$J_k(s_i) = \min_{1\le j\le n_{k+1}} [a^k_{ij} + J_{k+1}(s_j)] \ , \ 1 \le i \le n_k \ , \ 0 \le k \le N-1 \ ,$$

where $J_0(s_0)$ represents the optimal cost.
The intermediate cost $J_k(s_i)$ represents the **optimal-cost-to-go** from state $x_i \in S_k$ to the terminal (fictitious) state x_{N+1} associated with the terminal node s_{N+1}.

Remark (shortest path problem): If we identify the transition costs a^k_{ij} with the lengths of the arcs connecting the nodes, the optimal cost corresponds to the length of the shortest path from the initial node s_0 to the terminal node s_{N+1}. Hence, problem (2.20),(2.21) can be interpreted as a **shortest path problem**.

2.3.2 The forward DP algorithm

In contrast to stochastic DP algorithms, in a deterministic framework there is an alternative to the backward-in-time algorithm, namely the **forward DP algorithm** which proceeds forward-in-time. It is based on the following observation with respect to the shortest path problem: The optimal path from the initial node s_0 to the terminal node s_{N+1} is also optimal for the so-called **reverse shortest path problem**, i.e., find the shortest path from the final node s_{N+1} to the initial node s_0 with a reverse order of the arcs but the same lengths (transition costs). The **forward DP algorithm** reads as follows:

(2.22a)

$$\tilde{J}_N(s_j) = a^0_{s_0,j} \quad , \quad 1 \le j \le n_1 \ ,$$

(2.22b)

$$\tilde{J}_k(s_j) = \min_{1\le i\le n_{N-k}} [a^{N-k}_{ij} + \tilde{J}_{k+1}(s_i)] \ , \ 1 \le j \le n_{N-k+1} \ , \ 1 \le k \le N-1 \ ,$$

with optimal cost

$$\tilde{J}_0(s_{N+1}) = \min_{1\le i\le n_N} [a^N_{i,s_{N+1}} + \tilde{J}_1(s_i)] \ . \tag{2.22c}$$

The backward DP algorithm (2.21a),(2.21b) and the forward DP algorithm (2.20a)-(2.22c) provide the same optimal cost

$$J_0(s_0) \ = \ \tilde{J}_0(s_{N+1}) \ . \tag{2.23}$$

The intermediate cost $\tilde{J}_k(j)$ can be interpreted as the **optimal-cost-to-arrive** from the initial node s_0 to the node s_j at stage k.

Remark (hidden Markov models): The forward DP algorithm will be applied for problems where the stage k problem data are unknown prior to stage k and become available to the controller just before decisions at stage k have to be made (hidden Markov models).

2.3.3 Shortest path problems as deterministic finite-state problems

In the previous sections, we have seen how to formulate a deterministic finite-state problem as a shortest path problem. Conversely, a shortest path problem can be cast in the framework of a deterministic finite-state problem:
We consider a graph consisting of $N+1$ nodes and arcs connecting node i with node j. The length of the arc from node i to node j is considered as the cost a_{ij} of moving from node i to node j. The final node $N+1$ is a special node which is dubbed **destination**. We set $a_{ij} = +\infty$, if there is no arc that connects node i and node j. The **shortest path problem** is to find the shortest path from each node i to the destination $N+1$. We formally require a total of N moves, but allow degenerate moves from a node i to itself at cost $a_{ii} = 0$. We consider N stages $k = 0, \cdots, N-1$ with $J_k(i)$ denoting the optimal cost of getting from node i to the destination $N+1$ in $N-k$ moves, i.e.,

(2.24a)
$$J_{N-1}(i) = a_{i,N+1} \quad , \quad 1 \le i \le N \ ,$$
(2.24b)
$$J_k(i) = \min_{1 \le j \le N} [a_{ij} + J_{k+1}(j)] \ , \ 1 \le i \le N \ , \ k = N-2, \cdots, 0 \ ,$$

with $J_0(i)$ being the cost of the optimal path. If the optimal path contains degenerate moves, we drop the degenerate moves which means that the optimal path may contain less than N moves.

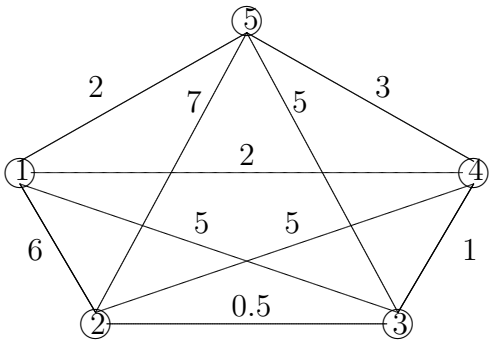

Fig. 2.5. Shortest path problem I (transition costs)

Example (shortest path problem I): We consider a graph with four nodes $i = 1, 2, 3, 4$ and destination 5. All nodes are interconnected and both directions are possible with equal costs as shown in Fig. 2.5. Fig. 2.6 contains the optimal costs-to-go $J_k(i)$ at stage k for each node $i = 1, 2, 3, 4$.

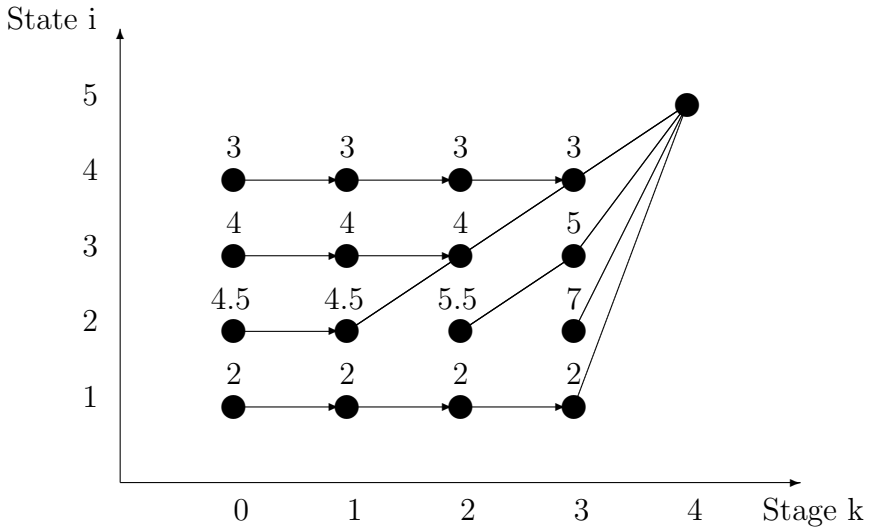

Fig. 2.6. Shortest path problem I (optimal costs-to-go)

Example (road network): We consider a network of roads connecting city 'A' (start) and 'J' (destination) via cities 'B'-'I' (cf. Fig. 2.7).

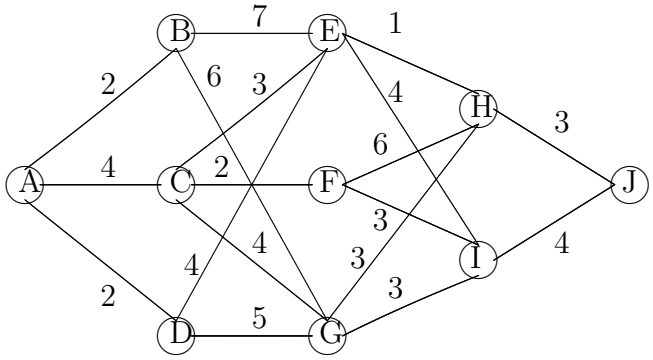

Fig. 2.7. Road network

The nodes S_k at stages $0 \leq k \leq 3$, and the terminal node S_4 are specified as follows

$$S_0 := \{A\} \quad , \quad S_1 := \{B, C, D\} \ ,$$
$$S_2 := \{E, F, G\} \quad , \quad S_3 := \{H, I\} \quad , \quad S_4 := \{J\} \ .$$

The costs a_{ij}^k for moving from node $i \in S_k$ to node $j \in S_{k+1}$ are indicated in Fig. 2.7 with the convention that $a_{ij} = +\infty$, if there is no arc connecting i and j. We recall the recursions from the backward DP algorithm

$$J_3(i) = a_{i,J} \quad , \quad i \in \{H, I\} \ ,$$
$$J_k(i) = \min_{j \in S_{k+1}} [a_{ij}^k + J_{k+1}(j)] \quad , \quad i \in S_k \ , \ k = 2, 1, 0 \ .$$

Following this recursion, at **stage 3** we find:

S_3	$J_3(\cdot)$	Decision go to
H	3	J
I	4	J

Table 2.2: Cost-to-go at stage 3

At **stage 2** we obtain

S_2	$a_{ij}^2 + J_3(j)$		$J_2(i)$	Decision go to
	H	I		
E	4	8	4	H
F	9	7	7	I
G	6	7	6	H

Table 2.3: Cost-to-go at stage 2

The **stage 1** computations reveal

S_1	$a_{ij}^1 + J_2(j)$			$J_1(i)$	Decision go to
	E	F	G		
B	11	11	12	11	E or F
C	7	9	10	7	E
D	8	8	11	8	E or F

Table 2.4: Cost-to-go at stage 1

Finally, at **stage 0** we get

S_0	$a_{ij}^0 + J_1(j)$			$J_0(i)$	Decision
	B	C	D		go to
A	13	11	11	11	C or D

Table 2.5: Cost-to-go at stage 0

Example (capital budgeting problem): A publishing company has \$ 5 million to allocate to its three divisions

$$i = 1 \text{ 'Natural Sciences'}$$
$$i = 2 \text{ 'Social Sciences'}$$
$$i = 3 \text{ 'Fine Arts'}$$

for expansion. Each division has submitted 4 proposals containing the cost c_i of the expansion and the expected revenue r_i (cf. Table 2.1).

Proposal	Division 1		Division 2		Division 3	
	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	0	0
4	0	0	4	12	0	0

Table 2.6: Proposals for investment

For the application of the forward DP algorithm we denote by

- **stage 1:** amount $x_1 \in \{0, 1, \cdots, 5\}$ of money allocated to division 1,
- **stage 2:** amount $x_2 \in \{0, 1, \cdots, 5\}$ of money allocated to divisions 1,2
- **stage 3:** amount $x_3 = 5$ of money allocated to divisions 1,2, and 3.

We further denote by

- $c(i_k)$: cost for proposal i_k at stage k,
- $r(i_k)$: revenue for proposal i_k at stage k,

and by

- $J_k(x_k)$: revenue of state x_k at stage k.

With these notations, the recursions for the forward DP algorithm are as follows:

$$\begin{aligned} J_1(x_1) &= \max_{i_1:c(i_1)\le x_1} [r(i_1)] \ , \\ J_k(x_k) &= \max_{i_k:c(i_k)\le x_k} [r(i_k) + J_{k-1}(x_k - c(i_k))] \ , \ k = 2, 3 \ . \end{aligned}$$

The **stage 1** computations yield

Capital x_1	Optimal Proposal	Revenue at stage 1
0	1	0
1	2	5
2	3	6
3	3	6
4	3	6
5	3	6

Table 2.7: Capital budgeting (stage 1)

As an example for the computations at stage 2 we consider the allocation of $x_2 = 4$ million US-Dollars:

- Proposal 1 gives revenue of 0, leaves 4 for stage 1, which returns 6. Total: 6;
- Proposal 2 gives revenue of 8, leaves 2 for stage 1, which returns 6. Total: 14;
- Proposal 3 gives revenue of 9, leaves 1 for stage 1, which returns 5. Total: 14;
- Proposal 4 gives revenue of 12, leaves 0 for stage 1, which returns 0. Total: 12.

Altogether, the **stage 2** computations result in

Capital x_2	Optimal Proposal	Revenue at stages 1,2
0	1	0
1	2	5
2	2	8
3	2	13
4	2 or 3	14
5	4	17

Table 2.8: Capital budgeting (stage 2)

At stage 3, the only amount of money to allocate is $x_3 = 5$ million US-Dollars. For division 3 we obtain:

- Proposal 1 gives revenue of 0, leaves 5. Previous stages give 17. Total: 17;
- Proposal 2 gives revenue of 4, leaves 4. Previous stages give 14. Total: 18.

This leaves for **stage 3**:

Capital x_3	Optimal Proposal	Revenue at stage 1,2,3
5	2	18

Table 2.9: Capital budgeting (stage 3)

Summarizing, the optimal allocation is: Implement proposal 2 at division 3, proposal 2 or 3 at division 2, and proposal 3 or 2 at division 1. The optimal revenue is $ 18 million.

2.3.4 Partially observable finite-state Markov chains and the Viterbi algorithm

Let $S = \{0, 1, \cdots, N\}$ be a finite number of states and

$$P = (p_{ij})_{i,j=1}^{N} \in \mathbb{R}^{N\times N} \tag{2.25}$$

be a **stochastic matrix**, i.e.,

$$p_{ij} \geq 0 \ , \ 1 \leq i,j \leq N \ , \quad \text{and} \quad \sum_{j=1}^{N} p_{ij} = 1 \ , \ 1 \leq i \leq N \ . \tag{2.26}$$

The pair $\{S, P\}$ is called a **stationary finite-state Markov chain**. We associate with $\{S, P\}$ a **process** with **initial state** $x_0 \in S$ with probability

$$P(x_0 = i) \ = \ r_0^i \quad , \quad 1 \leq i \leq N \ , \tag{2.27}$$

where for $k = 0, 1, \cdots, N-1$ transitions from the state x_k to the state x_{k+1} occur with probability

$$P(x_{k+1} = j | x_k = i) \ = \ p_{ij} \quad , \quad 1 \leq i,j \leq N \ , \tag{2.28}$$

i.e., if the state x_k is $x_k = i$, the probability that the new state x_{k+1} satisfies $x_{k+1} = j$ is given by p_{ij}.
The probability that $x_k = j$, if the initial state satisfies $x_0 = i$, is then given by

$$P(x_k = j | x_0 = i) \ = \ p_{ij}^k \quad , \quad 1 \leq i,j \leq N \ . \tag{2.29}$$

Denoting by

$$r_k \;=\; (r_k^1, \cdots, r_k^N)^T \in \mathbb{R}^N \tag{2.30}$$

the **probability distribution** of the state x_k after k transitions, in view of (2.27) and (2.29) we obtain

$$\begin{aligned} r_k \;&=\; r_o P^k \quad , \quad 1 \le k \le N \ , \text{ i.e.,} \\ r_k^j \;&=\; \sum_{i=1}^{N} P(x_k = j | x_0 = i) r_0^i \;=\; \sum_{i=1}^{N} p_{ij}^k r_0^i \ . \end{aligned} \tag{2.31}$$

We now focus on so-called **hidden Markov models** or **partially observable finite-state Markov chains**. The characteristic feature is that the states involved in the transition $x_{k-1} \to x_k$ are unknown, but instead an **observation** z_k of probability $r(z_k; x_{k-1}, x_k)$ is available. We introduce the following notations

$$\begin{aligned} X_N \;&=\; \{x_0, x_1, \cdots, x_N\} \quad \textbf{state transition sequence} \ , \\ Z_N \;&=\; \{z_1, z_2, \cdots, z_N\} \quad \textbf{observation sequence} \ , \end{aligned}$$

and

$$p(X_N | Z_N) \;=\; \frac{p(X_N, Z_N)}{p(Z_N)} \quad \textbf{conditional probability} \ , \tag{2.32}$$

where $p(X_N, Z_N)$ and $p(Z_N)$ are the (unconditional) probabilities of occurrence of (X_N, Z_N) and Z_N, respectively.
The **goal** is to provide an **optimal state estimation**

$$\hat{X}_N \;=\; \{\hat{x}_0, \hat{x}_1, \cdots, \hat{x}_N\} \tag{2.33}$$

in the sense that

$$p(\hat{X}_N | Z_N) \;=\; \max_{X_N} p(X_N | Z_N) \ . \tag{2.34}$$

Since $p(Z_N)$ is a positive constant, once Z_N is known, (2.34) reduces to

$$p(\hat{X}_N, Z_N) \;=\; \max_{X_N} p(X_N, Z_N) \ . \tag{2.35}$$

Lemma 2.2 (Probability of state transition/observation sequences)
Let X_N and Z_N be the state transition and observation sequences associated with a finite-state Markov chain and let r_{x_0} and $r(z_k; x_{k-1}, x_k)$

be the probability that the initial state takes the value x_0 and the probability of observation z_k at the k-th transition $x_{k-1} \to x_k$. Then, there holds

$$p(X_N, Z_N) \;=\; r_{x_0} \prod_{k=1}^{N} p_{x_{k-1},x_k} r(z_k; x_{k-1}, x_k) \;. \tag{2.36}$$

Proof. The proof is by induction on N. Indeed, for $N = 1$ we find

$$\begin{aligned} p(X_1, Z_1) \;&=\; p(x_0, x_1, z_1) \;= \\ &=\; r_{x_0} p(x_1, z_1 | x_0) \;=\; r_{x_0} p_{x_0,x_1} r(z_1; x_0, x_1) \;. \end{aligned}$$

The last step of the induction proof is left as an exercise. □

The maximization problem (2.35) can be equivalently stated as a shortest path problem by means of the so-called **trellis diagram** which can be generated by concatenating $N+1$ copies of the state space representing stages $0 \le k \le N$, an additional initial node s_0 and an additional terminal node s_{N+1} (cf. Fig. 2.9).

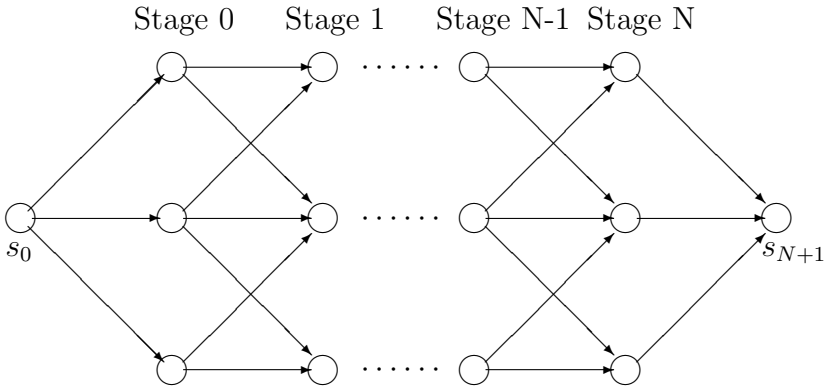

Fig. 2.9. Trellis diagram

An arc connects a node x_{k-1} of the k-th copy of the state space with a node x_k of the $(k+1)$-st copy, if the associated transition probability p_{x_{k-1},x_k} is positive.

Lemma 2.3 (Optimal state estimation as a minimization problem)
Using the same notations as in Lemma 2.2, the maximization problem (2.35) is equivalent to

$$\min_{X_N} \left[-\ln(r_{x_0}) \;-\; \sum_{k=1}^{N} \ln(p_{x_{k-1},x_k} r(z_k; x_{k-1}, x_k)) \right] . \tag{2.37}$$

Proof. The maximization of a positive objective functional is equivalent to the minimization of its logarithm. □

Corollary 2.4 (Associated shortest path problem)
The solution of the minimization problem (2.37) is given by the shortest path in the trellis diagram connecting the initial node s_0 with the terminal node s_{N+1}.

Proof. The proof follows directly from the following assignment of lengths to the arcs in the trellis diagram

$$\begin{aligned} (s_0, x_0) &\longmapsto -\ln(r_{x_0}) \ , \\ (x_{k-1}, x_k) &\longmapsto -\ln(p_{x_{k-1},x_k} r(z_k; x_{k-1}, x_k)) \ , \ 1 \le k \le N \ , \\ (x_N, s_{N+1}) &\longmapsto 0 \ . \end{aligned}$$

The shortest path in the trellis diagram thus gives the optimal state estimate $\hat{X}_N$. □

The above shortest path problem can be solved by the forward DP algorithm which in this special case is referred to as the **Viterbi algorithm**. Denoting by $\mathrm{dist}(s_0, x_k)$ the distance from the initial node s_0 to a state x_k, we have the recursion

$$\text{(2.38a)} \quad \begin{aligned} &\mathrm{dist}(s_0, x_{k+1}) \ = \\ &= \min_{x_k : p_{x_k,x_{k+1}} > 0} [\mathrm{dist}(s_0, x_k) - \ln(p_{x_k,x_{k+1}} r(z_{k+1}; x_k, x_{k+1}))] \ , \end{aligned}$$

$$\text{(2.38b)} \quad \mathrm{dist}(s_0, x_0) \ = \ -\ln(r_{x_0}) \ .$$

Example (Speech recognition: convolutional coding and decoding)
An important issue in **information theory** is the reliable **transmission of binary data over a noisy communication channel**, e.g, transmitting speech data from the cell phone of a sender via satellite to the cell phone of a receiver. This can be done by **convolutional coding** and **decoding**.

The **encoder/decoder scheme** is illustrated in Fig.2.10: We assume that we are given a source-generated **binary data sequence**

$$W_N = \{w_1, w_2, \cdots, w_N\} \quad , \quad w_k \in \{0, 1\} \ , \ 1 \le k \le N \ .$$

This sequence is converted into a **coded sequence**

$$\begin{aligned} &Y_N = \{y_1, y_2, \cdots, y_N\} \ , \ y_k = (y_k^1, \cdots, y_k^n)^T \ , \\ &y_k^i \in \{0, 1\} \ , \ 1 \le i \le n \ , \ 1 \le k \le N \ , \end{aligned}$$

where each **codeword** y_k is a vector in $\mathbb{R}^n$ with binary coordinates. The coded sequence is then transmitted over a noisy channel resulting in a **received sequence**

$$Z_N = \{z_1, z_2, \cdots, z_N\} \ ,$$

which has to be decoded such that the **decoded sequence**

$$\hat{W}_N = \{\hat{w}_1, \hat{w}_2, \cdots, \hat{w}_N\} \ ,$$

is as close to the original binary data sequence W_N as possible.

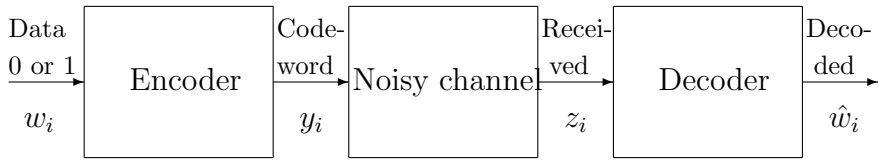

Fig. 2.10. Encoding and decoding in communications

Convolutional coding:
In convolutional coding, the vectors y_k of the coded sequence are related to the elements w_k of the original binary data sequence by means of a discrete dynamical system of the form

$$\begin{aligned} y_k &= Cx_{k-1} + dw_k \quad , \quad 1 \leq k \leq N \ , \\ x_k &= Ax_{k-1} + bw_k \ , \end{aligned} \tag{2.39}$$

where $x_0 \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times m}, C \in \mathbb{R}^{n \times m}$ as well as $b \in \mathbb{R}^m, d \in \mathbb{R}^n$ are given. The products and sums in (2.39) are evaluated using modulo 2 arithmetic. The sequence $\{x_1, \cdots, x_N\}$ is called the **state sequence**.

Example: Assume $m = 2, n = 3$ and

$$C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \quad , \quad A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \quad , \quad d = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad , \quad b = \begin{pmatrix} 0 \\ 1 \end{pmatrix} .$$

The evolution of the system (2.39) is shown in Fig. 2.11. The binary number pair on each arc is the data/codeword pair w_k/y_k. For instance, if $x_{k-1} = 01$, a zero data bit $w_k = 0$ results in a transition to $x_k = 11$ and generates the codeword $y_k = 001$.

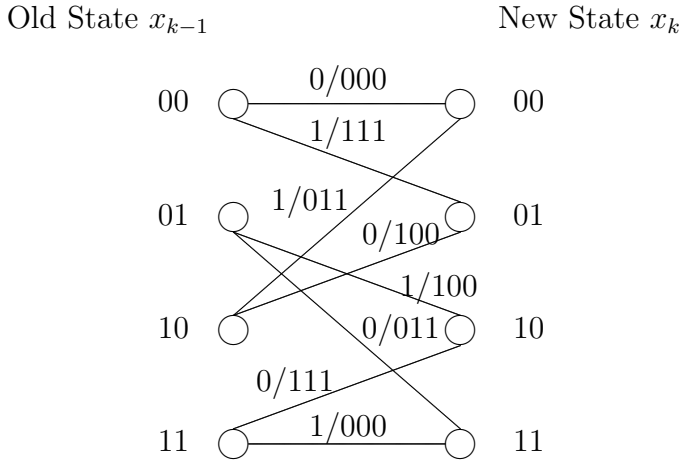

Fig. 2.11. State transition diagram for convolutional coding

In case $N = 4$ and $x_0 = 00$, the binary data sequence

$$\{w_1, w_2, w_3, w_4\} \;=\; \{1, 0, 0, 1\}$$

generates the state sequence

$$\{x_1, x_2, x_3, x_4\} \;=\; \{01, 11, 10, 00\}$$

and the codeword sequence

$$\{y_1, y_2, y_3, y_4\} \;=\; \{111, 011, 111, 011\}\ .$$

We assume that the noisy transmission over the channel is such that the codewords y_k are received as z_k with known **conditional probabilities** $P(z_k|y_k)$ and that the events are independent in the sense

$$P(Z_N|Y_N) \;=\; \prod_{k=1}^{N} P(z_k|y_k)\ . \tag{2.40}$$

The determination of an optimal code can be stated as a **maximum likelihood estimation problem**:
Find $\hat{Y}_N = (\hat{y}_1, \cdots, \hat{y}_N)$ such that

$$P(Z_N|\hat{Y}_N) \;=\; \max_{Y_N} P(Z_N|Y_N)\ , \tag{2.41a}$$

$$Y_N \text{ satisfies (2.39)}\ . \tag{2.41b}$$

In order to reformulate (2.41) as a shortest path problem, we construct a trellis diagram by concatenating N state transition diagrams as in Fig. 2.11 and introduce a dummy initial node s and a dummy terminal node t such that these dummy nodes are connected to x_0 and

x_N by zero-length arcs, respectively. Observing (2.40), the maximum likelihood estimation problem (2.41) can be rewritten as

$$\text{(2.42a)} \qquad \text{minimize } -\sum_{k=1}^{N} \ln(P(z_k|y_k)) \text{ over } Y_N \ ,$$

$$\text{(2.42b)} \qquad Y_N \text{ satisfies (2.39)} \ .$$

Considering the arc associated with the codeword y_k as being of length $-\ln(P(z_k|y_k))$, the solution of (2.42) is the shortest path from s to t. The maximum likelihood estimate $\hat{Y}_N$ can be determined by the **Viterbi algorithm** where the shortest distance $d_{k+1}(x_{k+1})$ from s to any state x_{k+1} is obtained by the dynamic programming recursion

$$\text{(2.43a)} \qquad d_{k+1}(x_{k+1}) \ = \ \min_{x_k} \Big(d_k(x_k) \ - \ \ln(P(z_{k+1}|y_{k+1})) \Big) \ ,$$

$$\text{(2.43b)} \qquad x_k \text{ is connected to } x_{k+1} \text{ by an arc } (x_k, x_{k+1}) \ ,$$

where y_{k+1} is the codeword associated with the arc (x_k, x_{k+1}).
Finally, the optimal decoded sequence $\hat{W}_N$ is retrieved from the maximum likelihood estimate $\hat{Y}_N$ by using the trellis diagram.

2.4 Label-correcting and branch-and-bound methods

2.4.1 Label-correcting methods

We consider a **graph** consisting of an **initial node** s, a **terminal node** t, **intermediate nodes** i ($i \neq s, i \neq t$), and **arcs** connecting the nodes. A node j is called a **child** of node i, if there exists an arc (i, j) connecting i and j of length $a_{ij} \geq 0$. The issue is to find the shortest path from the initial node s to the terminal node t.

Label correcting methods are shortest path algorithms that iteratively detect shorter paths from the initial node s to every other node i in the graph. They introduce d_i as an additional variable, called the **label** of node i whose value corresponds to the shortest path from s to i found so far. The label d_t of the terminal node t is stored in a variable called UPPER. We choose the following **initialization** of the labels:

- The label d_s of the initial node s is initialized by $d_s = 0$ and remains unchanged during the algorithm.
- All other labels $d_i, i \neq s$, are initialized by $d_i = +\infty$.

The algorithm also features a **list of nodes** dubbed OPEN (or **candidate list**) which contains nodes that are candidates for possible inclusion in the shortest path in a sense that will be made clear later. The **initialization** of OPEN is done as follows:

- OPEN is initialized by OPEN $= \{s\}$, i.e., at the beginning, OPEN only contains the initial node s.

The k-th step of the iteration ($k \geq 1$) consists of three substeps which are as follows:

Substep 1: Remove a node i from OPEN according to a strategy that will be described later. In particular, at the first iteration step, i.e., $k = 1$, the initial node s will be removed from OPEN, since initially OPEN $= \{s\}$.

Substep 2: For each child j of i test whether its label d_j can be reduced by checking

$$d_i + a_{ij} < \min(d_j, UPPER) . \tag{2.44}$$

If the test (2.44) is satisfied, set

$$d_j = d_i + a_{ij} \tag{2.45}$$

and call i the **parent** of j.
Moreover, update the list OPEN and the value of UPPER according

to

$$OPEN = OPEN \cup \{j\} \quad , \quad \text{if } j \neq t \ , \tag{2.46a}$$

$$UPPER = d_i + a_{it} \quad , \quad \text{if } j = t \ . \tag{2.46b}$$

Substep 3: If OPEN $= \emptyset$, terminate the algorithm. Otherwise, set $k = k + 1$ and go to Substep 1.

Theorem 2.2 (Finite termination property)
If there exists a path connecting the initial node s with the terminal node t, the label correcting method terminates after a finite number of steps with the shortest path given by the final value of UPPER. Otherwise, it terminates after a finite number of steps with UPPER $= +\infty$.

Proof: The algorithm terminates after a finite number of steps, since there is only a finite number of possible label reductions.

Case 1 (Non-existence of a path from s to t):
If there is no path from s to t, a node i connecting i and t by an arc (i,t) can never enter the list OPEN, since otherwise there would be a path from s to t in contradiction to the assumption. Consequently, UPPER can not be reduced from its initial value $+\infty$, and hence, the algorithm terminates with UPPER $= +\infty$.

Case 2 (Existence of a path from s to t):
If there is a path from s to t, there must be a shortest path, since there is only a finite number of paths connecting s and t. Assume that $(s, j_1, j_2, \cdots, j_k, t)$ is the shortest path of length d^{opt}. Each subpath $(s, j_1, \cdots, j_m), m \in \{1, \cdots, k\}$, of length d_{j_m} must then be the shortest path from s to j_m.
In contradiction to the assertion, we assume UPPER $> d^{opt}$. Then, for all $m \in \{1, \cdots, k\}$ we must also have UPPER $> d_{j_m}$. Consequently, the node j_k never enters the list OPEN, since otherwise UPPER will be set to d^{opt} in Substep 2 of the algorithm. Using the same argument backwards, it follows that the nodes $j_m, 1 \leq m < k$, never enter OPEN. However, since j_1 is a child of the initial node s, at the first iteration j_1 must enter OPEN and hence, we arrive at a contradiction. □

Example (The traveling salesman problem)
We consider a salesman who lives in a city A and has to visit a certain number $N-1$ of other cities for doing business before returning to city A. Knowing the mileage between the cities, the problem is to find the **minimum-mileage trip** that, starting from city A, visits each other city only once and then returns to city A.
The traveling salesman problem can be formulated as a shortest path

problem as follows: We construct a graph where the nodes are ordered sequences of $n \leq N$ distinct cities with the length of an arc denoting the mileage between the last two cities in the ordered sequence (cf. Fig. 2.x). The initial node corresponds to city A. We introduce an artificial terminal node t with an arc having the length equal to the mileage between the last city of the sequence and the starting city A. The shortest path from s to t represents the solution of the minimum-mileage problem.

In case of four cities A,B,C, and D, Fig. 2.12 illustrates the associated graph, whereas Table 2.10 contains the iteration history of the label correcting algorithm.

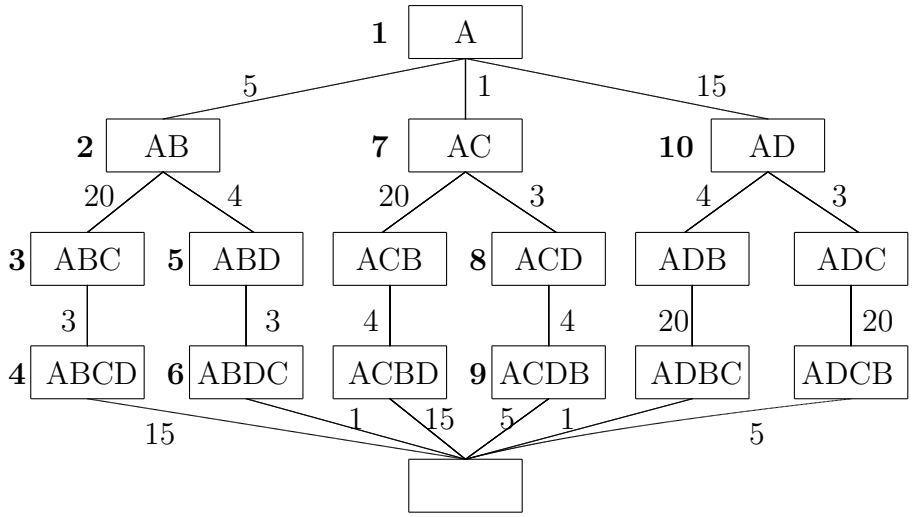

Fig. 2.12 Label correcting algorithm for the traveling salesman problem

Iteration	Node existing OPEN	OPEN at the end of the iteration	UPPER
0	–	1	∞
1	1	2,7,10	∞
2	2	3,5,7,10	∞
3	3	4,5,7,10	∞
4	4	5,7,10	43
5	5	6,7,10	43
6	6	7,10	13
7	7	8,10	13
8	8	9,10	13
9	9	10	13
10	10	$\emptyset$	13

Table 2.10: Iteration history of the label correcting algorithm

We finally discuss a variety of strategies for the removal of a node from the list OPEN in Substep 1 of the label correcting method. All of them except **Dijkstra's method** assume that the list OPEN is an ordered sequence of nodes (queue) with the first node being located at the 'top' of OPEN and the last node at the 'bottom'.

Breadth-first search (Bellman-Ford method):
Remove the node from the top of OPEN and place a new node at the bottom ('first-in/first-out policy').

Depth-first search:
Remove the node from the top of OPEN and place a new node at the top ('last-in/first-out policy').

D'Esopo-Pape method:
Remove the node from the top of OPEN and place a new node

- at the top of OPEN, if it has been in OPEN earlier,
- at the bottom of OPEN, otherwise.

Small-Label-First (SLF) method:
Remove the node from the top of OPEN and place a new node i

- at the top of OPEN, if its label d_i is less than or equal to the label d_j of the node at the TOP of OPEN,
- at the bottom of OPEN, otherwise.

Dijkstra's method (best-first search):
Remove from OPEN the node i with minimum label, i.e.

$$d_i = \min_{j \in OPEN} d_j \ . \tag{2.47}$$

Since this method does not necessarily assume an ordered list, a new node may enter OPEN somewhere.

2.4.2 Branch-and-bound methods

We consider the problem to minimize a function $f : X \to \mathbb{R}$ where the feasible set X consists of a finite number of feasible points, i.e., $\text{card}(X) = n < +\infty$:

$$\text{minimize } f(x) \quad \text{over } x \in X \ , \ \text{card}(X) = n \ . \tag{2.48}$$

The principle of the branch-and-bound methods is to split the feasible set X into smaller subsets with associated minimization subproblems and to compute bounds for the optimal value of these minimization subproblems in order to be able to exclude some feasible subsets from further consideration. To be more precise, let us assume that $Y_1 \subset X$ and $Y_2 \subset X$ are two subsets of the feasible set X and suppose that

$$\underline{f}_1 \ \leq \ \min_{x \in Y_1} f(x) \quad , \quad \overline{f}_2 \ \geq \ \min_{x \in Y_2} f(x) \ . \tag{2.49}$$

Then, if

$$\overline{f}_2 \ \leq \ \underline{f}_1 \ ,$$

the feasible points in Y_1 may be disregarded, since their cost can not be smaller than the cost associated with the optimal solution in Y_2.

In order to describe the branch-and-bound method systematically, we denote by $\mathcal{X}$ the collection of subsets of the feasible set X and construct an **acyclic graph** associated with $\mathcal{X}$ such that the nodes of the graph are elements of $\mathcal{X}$ according to the following **rules**:

(a) $X \in \mathcal{X}$,

(b) For each feasible point $x \in X$ we have $\{x\} \in \mathcal{X}$,

(c) Each set $Y \in \mathcal{X}$ with $\text{card}(Y) > 1$ into sets $Y_i \in \mathcal{X}, 1 \leq i \leq k$, such that

$$Y_i \ \neq \ Y \ , \ 1 \leq i \leq k \quad , \quad Y \ = \ \bigcup_{i=1}^{k} Y_i \ .$$

The set Y is called the **parent** of the sets $Y_i, 1 \leq i \leq k$, which are themselves referred to as the **children** of Y.

(d) Each set in $\mathcal{X}$ different from X has at least one parent.

We thus obtain an acyclic graph whose origin is the set X of all feasible points and whose terminal nodes are the sets $\{x\}$ of single feasible points. The arcs of the graph are those connecting the parents Y and their children Y_i (cf. Fig. 2.13). The length of an arc connecting Y

and Y_i will be computed as follows:

(i) Suppose we have at disposal an algorithm which for every parental node Y is able to compute upper and lower bounds for the associated minimization subproblem, i.e.,

$$\underline{f}_Y \;\leq\; \min_{x\in Y} f(x) \;\leq\; \overline{f}_Y \;.$$

(ii) Assume further that the bounds are exact in case of single feasible points so that formally

$$\underline{f}_{\{x\}} \;=\; f(x) \;=\; \overline{f}_{\{x\}} \;.$$

We define

$$\mathrm{arc}(Y, Y_i) \;=\; \underline{f}_{Y_i} \;-\; \underline{f}_Y \;.$$

According to this definition, the length of any path connecting the initial node X with any node Y is given by $\underline{f}_Y$. Since $\underline{f}_{\{x\}} = f(x), x \in X$, it follows that the solution of (2.48) corresponds to the shortest path from X to one of the nodes $\{x\}$.

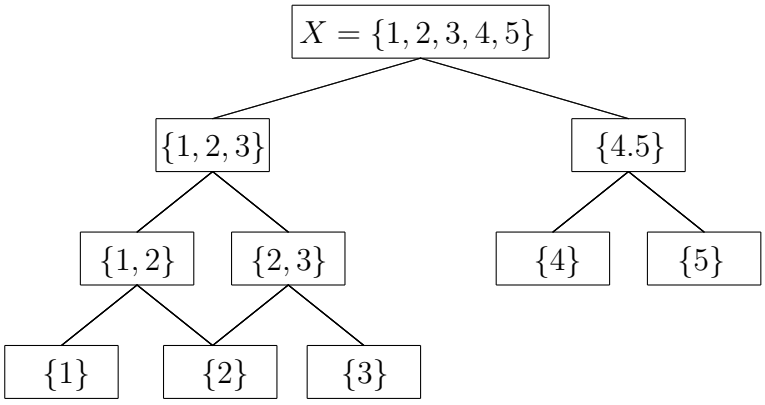

Fig. 2.13 Branch-and-bound method

The **branch-and-bound method** can be formulated as a variation of the **label correcting method** (see section 2.4.1) where the list OPEN and the value UPPER are initialized as follows:

$$\mathrm{OPEN} \;:=\; \{X\} \quad , \quad \mathrm{UPPER} \;:=\; \overline{f}_X \;.$$

The k-th iteration consists of the following substeps:

Substep 1: Remove a node Y from OPEN.

Substep 2: For each child Y_j of Y perform the test

$$\underline{f}_{Y_j} \; < \; \text{UPPER} \; .$$

If the test is satisfied, update OPEN according to

$$\text{OPEN} \; := \; \text{OPEN} \cup \{Y_j\}$$

and set

$$\text{UPPER} \; := \; \overline{f}_{Y_j} \quad , \quad \text{if } \overline{f}_{Y_j} \; < \; \text{UPPER} \; .$$

In case Y_j consists of a single feasible point, mark the solution as being the best solution found so far.

Substep 3: Terminate the algorithm, if OPEN $= \emptyset$. The best solution found so far corresponds to the optimal solution. Otherwise, set $k := k + 1$ and go to Substep 1.

It should be noted that the generation of the acyclic graph, i.e., the decomposition of the original minimization problem, is usually not done a priori, but performed during execution of the algorithm based on the available information. The lower bounds for the minimization subproblems are typically computed by methods from continuous optimization, whereas the upper bounds can be often obtained using appropriate heuristics. For details we refer to [3].

References

[1] D.P. Bertsekas; Dynamic Programming and Optimal Control. Vol. 1. 3rd Edition. Athena Scientific, Belmont, MA, 2005

[2] D.P. Bertsekas and S.E. Shreve; Stochastic Optimal Control: The Discrete-Time Case. Athena Scientific, Belmont, MA, 1996

[3] G.L. Nemhauser and L. Wolsey; Integer and Combinatorial Optimization. John Wiley & Sons, New York, 1988