Capitolo 9
Programmazione dinamica

La programmazione dinamica fu proposta da R. Bellman nel 1953 per risolvere in modo efficiente problemi di decisione di tipo sequenziale (si veda ad esempio la monografia Bellman [1957]). In questo tipo di problemi le decisioni si attuano periodicamente ed influenzano le grandezze del modello. A loro volta queste influenzano le decisioni future. Il principio di ottimalità, enunciato da Bellman, permette, attraverso un'intelligente applicazione, di affrontare in modo efficiente la complessità intrinseca dell'interazione fra le decisioni e le grandezze del modello. Le tecniche della programmazione dinamica furono fin dall'inizio applicate inoltre a problemi in cui non è presente alcun aspetto temporale o sequenziale. Tuttavia, anche se teoricamente la programmazione dinamica può essere applicata ad una vasta gamma di problemi fornendo virtualmente un comune modello astratto, dal punto di vista pratico molti problemi richiedono modelli di tali dimensioni da precludere, allora come oggi, ogni approccio computazionale. Questo inconveniente fu chiamato allora 'maledizione della dimensionalità' e fu un'anticipazione, in termini ancora informali, di concetti di complessità computazionale.

I maggiori successi della programmazione dinamica sono stati ottenuti nell'ambito dei modelli di decisione sequenziali, specialmente di tipo stocastico (quali i processi markoviani di decisione), ma anche in alcuni particolari modelli di tipo combinatorio. In questa sede, anziché presentare la programmazione dinamica attraverso i consueti concetti di 'stadio' ('stage') e 'stato', si è preferito fornire un modello più generale che fa uso solamente del concetto di grafo orientato, che effettivamente è sufficiente a definire la programmazione dinamica.

9.1. Il principio di ottimalità

In senso generale la programmazione dinamica risolve problemi di cammino minimo (o massimo) su grafi orientati $G = (N, E)$ (sia $n = |N|$ e $m = |E|$) con costi associati ai cammini di tipo particolare. In molte applicazioni i grafi sono aciclici e questo fatto permette di avere generalmente algoritmi più veloci. I problemi di programmazione dinamica possono ricevere una formulazione in avanti, oppure all'indietro, oppure ambedue. Consideriamo inizialmente la formulazione in avanti.

Sia $P(i) : s \to i$ un generico cammino orientato da un nodo fissato s ad un generico nodo i. Se j è un nodo sul cammino $P(i)$ indichiamo con $P_j(i)$ la restrizione del cammino $P(i)$ fino al nodo j (di fatto è un generico cammino $P(j)$). La notazione può essere ambigua se il cammino non è semplice. Infatti, se il cammino passa per un nodo j più volte, non è chiaro a quale restrizione $P_j(i)$ si riferisce. Se non c'è pericolo di confusione intendiamo con $P_j(i)$ una generica restrizione. Se non è indispensabile indicare il nodo terminale del cammino, questo verrà omesso nella notazione.

Sia $V(P)$ il costo associato al cammino P. $V(P)$ viene calcolato nel seguente modo: sia i il nodo che precede il nodo j nel cammino P, allora $V(P_j)$ si esprime come funzione solamente del costo $V(P_i)$, cioè $V(P_j) := f_{ij}(V(P_i))$, ed è indipendente dal cammino che ha condotto nel nodo i. In altri termini l'unica informazione necessaria per determinare il costo futuro del cammino è il costo corrente con cui si è raggiunto un nodo e non ha importanza quale sia stato il cammino precedente. Per questo motivo si usa spesso, nei problemi di programmazione dinamica, la parola stato (secondo la terminologia dei sistemi dinamici) al posto di nodo. Quindi definiamo ricorsivamente per ogni cammino P

$$\begin{aligned} &V(P_s) := 0 \\ &V(P_j) := f_{ij}(V(P_i)) \qquad \forall (i,j) \in P \end{aligned} \tag{9.1}$$

Come conseguenza di (9.1) il costo del cammino $P(t) : s \to i \to j \ldots \to k \to t$ è dato da

$$V(P(t)) := f_{kt}(\ldots(\, f_{ij}(\, f_{si}(0))))$$

Nella maggior parte dei casi le funzioni definite in (9.1) sono di tipo additivo, cioè

$$f_{ij}(V) := V + c_{ij} \qquad \forall (i,j) \in E \tag{9.2}$$

L'obiettivo consiste nel trovare, per ogni nodo j, cammini ottimi $\hat{P}(j) : s \to j$ che minimizzano (o massimizzano a seconda dei casi) $V(P(j))$. A questo fine, è fondamentale nella programmazione dinamica il seguente principio.

9.1 Definizione. (Principio di ottimalità - formulazione in avanti) *Un cammino ottimo $\hat{P}$ soddisfa il principio di ottimalità se, per ogni nodo k sul cammino $\hat{P}$, $\hat{P}_k$ minimizza $V(P(k))$.* ∎

L'ipotesi essenziale che viene fatta sulle funzioni f_{ij} perché valga il principio di ottimalità è che siano strettamente monotone crescenti in V (cioè $V' > V'' \Longrightarrow f(V') > f(V'')$), ipotesi soddisfatta ad esempio quando sono additive (con qualsiasi valore di c_{ij}). Infatti si ha:

9.2 Teorema. *Siano f_{ij} strettamente monotone. Allora vale il principio di ottimalità per ogni cammino ottimo.*

Dimostrazione. Sia $\hat{P}$ un cammino minimo (massimo). Se esistesse un cammino $R(k)$ tale che $V(R(k)) < V(\hat{P}_k)$ $(V(R(k)) > V(\hat{P}_k))$ allora il cammino che si otterrebbe collegando $R(k)$ al sottocammino di $\hat{P}$ da k a t avrebbe un costo inferiore (superiore) a causa della stretta monotonia delle funzioni f_{ij} contraddicendo l'ottimalità di $\hat{P}$. ∎

Se l'ipotesi di stretta monotonia viene rilassata a semplice monotonia (cioè $V' > V'' \Longrightarrow f(V') \geq f(V'')$) vi possono essere cammini ottimi che non soddisfano il principio di ottimalità. Si consideri il seguente esempio: $f_{s1}(V) = V + 1$, $f_{s2}(V) = V + 1$, $f_{21}(V) = V + 1$, $f_{13}(V) = \max\{V, 3\}$. Vi sono solo due cammini $P(3) : s \to 1 \to 3$, $R(3) : s \to 2 \to 1 \to 3$ e sono ambedue minimi dato che $V(P(3)) = V(R(3)) = 3$. Se consideriamo $R_1(3)$, è chiaro che $R_1(3)$ non è minimo e quindi $R_1(3)$ non soddisfa il principio di ottimalità. Tuttavia un cammino che lo soddisfa comunque esiste ($P(3)$ nella fattispecie) e quindi sembrerebbe che la semplice monotonia possa garantire la validità del principio per almeno un cammino ottimo anziché per tutti. Ma non è così come si vede dal seguente esempio: $f_{s1}(V) = V + 3$, $f_{12}(V) = \min\{V, 1\}$, $f_{21}(V) = 2$, $f_{s2}(V) = V + 2$. Consideriamo i quattro cammini: $P^{(1)} : s \to 1 \to 2$, $P^{(2)} : s \to 1 \to 2 \to 1$, $P^{(3)} : s \to 2 \to 1$, $P^{(4)} : s \to 2 \to 1 \to 2$. Ogni altro cammino (di almeno due archi) si ottiene percorrendo più volte il ciclo $1 \to 2 \to 1$. Si

ottiene $V(P^{(1)}) = 1$, $V(P^{(1)}) = 1$, $V(P^{(2)}) = 2$, $V(P^{(3)}) = 2$, $V(P^{(4)}) = 1$. Sia $P^{(1)}$ che $P^{(4)}$ sono minimi fra i cammini che terminano nel nodo 2. Però $V(P_1^{(1)}) = 3 > V(P^{(3)}) = 2$ e $V(P_2^{(4)}) = 2 > V(P^{(1)}) = 1$. Analogamente $V(P_2^{(3)}) = 2 > V(P^{(1)}) = 1$. Nessun cammino ottimo soddisfa quindi il pincipio di ottimalità. Nel caso le funzioni f_{ij} siano soltanto monotone, bisogna introdurre quindi un'ipotesi alternativa alla stretta monotonia. Si definisce *superlineare* una funzione per cui $f(V) \geq V$ e *sublineare* una funzione per cui $f(V) \leq V$. Allora abbiamo:

9.3 Lemma. *Siano f_{ij} monotone e superlineari (sublineari). Allora esiste un cammino minimo (massimo) da s ad ogni nodo j.*

Dimostrazione. (Per problemi di minimo). In ogni cammino non semplice esiste un ciclo. Sia F_C la composizione delle f_{ij} lungo il ciclo. La superlinearità implica $F_C(V) \geq V$. Quindi, se dal cammino non semplice si toglie il ciclo, si ottiene un cammino di valore non superiore (per la monotonia). Per trovare i cammini minimi basta quindi considerare i cammini semplici e, essendo questi in numero finito, c'è la garanzia di esistenza dei minimi. Si ragiona in modo analogo per i cammini massimi. ∎

9.4 Lemma. *Siano le funzioni f_{ij} monotone e superlineari (sublineari). Allora, per ogni nodo k in un cammino minimo (massimo) semplice $P(j)$, esiste un cammino minimo (massimo) $P(k)$ che non contiene il nodo j.*

Dimostrazione. (Per problemi di minimo). La dimostrazione procederà per contraddizione. Supponiamo che il cammino minimo $P(k)$ contenga j e si abbia $V(P(k)) < V(P_k(j))$. Si indichi per semplicità di notazione $V^j := V(P(j))$, $V^k := V(P(k))$, $V_k := V(P_k(j))$ e $V_j := V(P_j(k))$. Quindi $V^k < V_k$.

Per definizione si ha $V^j = F_{kj}(V_k)$ e $V^k = F_{jk}(V_j)$. Per l'ottimalità si ha $V^j \leq V_j$ e $V^k \leq V_k$. Per la superlinearità si ha $V_k \leq F_{kj}(V_k)$ e $V_j \leq F_{jk}(V_j)$. Quindi

$$V^j \leq V_j \leq F_{jk}(V_j) = V^k \leq V_k \leq F_{kj}(V_k) = V^j$$

da cui $V^k = V_k$ contraddicendo l'ipotesi. ∎

9.5 Teorema. *Siano le funzioni f_{ij} monotone e superlineari (sublineari). Allora esiste, per ogni nodo j, un cammino minimo (massimo) $\hat{P}(j)$ per il quale vale il principio di ottimalità.*

Dimostrazione. Dato un cammino ottimo $\hat{P}(j)$ sia k il nodo che precede j su $\hat{P}(j)$. Sia $\hat{P}(k)$ un cammino ottimo che non contiene j. Il lemma precedente garantisce l'esistenza di un tale cammino. Si ridefinisca $\hat{P}(j)$ come $\hat{P}(k) \cup (k, j)$. Si consideri ora il nodo h che precede k sul nuovo $\hat{P}(j)$. In base al lemma esiste un cammino $P(h)$ che non contiene né k né j. Si ridefinisce $\hat{P}(j)$ come $\hat{P}(h) \cup (h, k) \cup (k, j)$. Si procede ricorsivamente fino ad arrivare al nodo s. Non potendoci essere ripetizioni di nodi, in base al lemma, c'è la garanzia che il cammino cercato esiste. ∎

L'importanza del principio di ottimalità risiede nella possibilità di sfruttare la seguente equazione ricorsiva nelle variabili $V_1, \ldots, V_n$, detta *equazione di Bellman*, che per problemi di minimo è:

$$V_j = \min_{i \in N_j^-} f_{ij}(V_i) \qquad V_s = 0$$

e per problemi di massimo

$$V_j = \max_{i \in N_j^-} f_{ij}(V_i) \qquad V_s = 0$$

dove $N_j^- := \{i \in N : (i,j) \in E\}$. Vale il seguente teorema che discende immediatamente dal principio.

9.6 Teorema. *Se il principio di ottimalità vale per almeno un cammino ottimo $P(i)$, per ogni nodo i, allora i valori ottimi $V_i := V(\hat{P}(i))$ soddisfano l'equazione di Bellman.*

Dimostrazione. (Per problemi di minimo). Sia k il nodo che precede j nel cammino ottimo $\hat{P}(j)$ per il quale assumiamo valga il principio di ottimalità. Allora $V_j = f_{kj}(V_k)$ (per il principio di ottimalità). Sia i un qualsiasi nodo predecessore di j e sia $\hat{P}(i)$ il cammino ottimo da s a i con valore ottimo V_i. Estendendo $\hat{P}(i)$ fino a j si ha che $f_{ij}(V_i) \geq V_j$ per l'ottimalità di V_j e quindi si ha la tesi. ∎

L'equazione di Bellman è soddisfatta anche dai valori ottimi di cammini senza che sia necessariamente richiesto il principio di ottimalità, bastando la semplice monotonia (si verifichi che i precedenti controesempi soddisfano l'equazione di Bellman).

9.7 Teorema. *Siano f_{ij} monotone ed esistano cammini ottimi $\hat{P}(j)$ per ogni nodo j. Allora i valori ottimi V_j soddisfano l'equazione di Bellman.*

Dimostrazione. (Per problemi di minimo). Sia k il nodo che precede j su $\hat{P}(j)$. Allora si ha $V_k \leq V(\hat{P}_k(j))$ per l'ottimalità e, dalla monotonia, $f_{kj}(V_k) \leq f_{kj}(V(\hat{P}_k(j))) = V_j$. Per ogni predecessore i di j si formino i cammini $\hat{P}(i) \cup (i,j)$ i cui valori sono $f_{ij}(V_i)$. Per l'ottimalità $f_{ij}(V_i) \geq V_j$ per ogni i e in particolare $f_{kj}(V_k) \geq V_j$. Dalla precedente diseguaglianza $f_{kj}(V_k) = V_j$ e l'equazione di Bellman è soddisfatta. ∎

Ci si può chiedere se l'equazione di Bellman possa essere soddisfatta dai valori ottimi quando non esistano soluzioni ottime (e il problema non sia illimitato). Si può dimostrare il seguente risultato:

9.8 Teorema. *Siano f_{ij} monotone e continue ed esistano valori ottimi limitati. Allora i valori ottimi soddisfano l'equazione di Bellman.*

Dimostrazione. (Per problemi di minimo). Sia $\hat{P}^h(j)$ una successione di cammini $s \to \ldots k \to j$ tali che $\lim_{h\to\infty} V(\hat{P}^h(j)) = V_j$. Esiste sempre un nodo k per cui una tale successione esiste. Sia $\bar{P}^h(k)$ una successione di cammini $s \to \ldots k$ tali che $\lim_{h\to\infty} V(\bar{P}^h(k)) = V_k$. Restringendo ogni cammino $\hat{P}^h(j)$ a k, si ha $V_k \leq V(\hat{P}^h_k(j))$. Quindi, per la monotonia, $f_{kj}(V_k) \leq f_{kj}(V(\hat{P}^h_k(j))) = V(\hat{P}^h(j))$ per ogni h, da cui, passando al limite, $f_{kj}(V_k) \leq V_j$. Sia i un qualsiasi nodo predecessore di j e sia $\bar{P}^h(i)$ una generica successione di cammini al valore ottimo V_i. Estendendo ogni cammino $\bar{P}^h(i)$ fino a j si ha, per definizione di valore ottimo, $f_{ij}(V(\bar{P}^h(i))) \geq V_j$. Passando al limite e sfruttando la continuità si ha $f_{ij}(V_i) \geq V_j$ per ogni i e in particolare $V_j = f_{kj}(V_k)$. ∎

L'ipotesi di continuità è necessaria. Si consideri il seguente esempio: $f_{s1}(V) = V + 2$, $f_{12}(V) = \sqrt{V}$, $f_{23}(V) = \sqrt{V}$, $f_{31}(V) = \sqrt{V}$, $f_{2t}(V) = V$ se $V > 1$ e $f_{2t}(V) = V - 1$ se $V \leq 1$. Si ottiene $V_t = 1$, $V_2 = 1$, però $f_{2t}(V_2) = 0 < V_t$.

I precedenti teoremi fanno vedere come i valori ottimi dei cammini vadano cercati fra le soluzioni dell'equazione di Bellman. Ci possiamo però chiedere se fra le soluzioni dell'equazione di Bellman vi siano valori che non hanno niente a che fare con i valori dei cammini ottimi. Questo può effettivamente succedere. Consideriamo il seguente esempio in cui tutte le funzioni sono additive: $f_{s1}(V) := V + 3$, $f_{s2}(V) := V + 2$, $f_{s3}(V) := V + 2$, $f_{12}(V) := V$, $f_{23}(V) := V$, $f_{31}(V) := V$, $f_{24}(V) := V + 1$, $f_{34}(V) := V + 1$. Vi sono infinite soluzioni dell'equazione di Bellman (minimo), cioè: $V_s = 0$, $V_1 = V_2 = V_3 = \alpha$, $V_4 = 1 + \alpha$, per ogni $\alpha \leq 2$. Di questi valori solo quelli per $\alpha = 2$ corrispondono ai valori dei cammini minimi. La

causa dell'esistenza di tali soluzioni spurie è essenzialmente dovuta al ciclo $1 \to 2 \to 3 \to 1$ per il quale la composizione delle f_{ij} è la funzione identica.

L'esempio può essere variato ridefinendo $f_{12}(V) := V^2/4$, $f_{23}(V) := V + \sqrt{V}/2$, $f_{31}(V) = V + 1/2$. Si vede che $F_C(V) := f_{31}(f_{23}(f_{12}(V))) = V^2/4 + V/4 + 1/2$ e che $F_C(V) = V$ per $V = 1$ e $V = 2$. L'equazione di Bellman è soddisfatta da tre soluzioni diverse: $V_s = 0$, $V_1 = 1$, $V_2 = 1/4$, $V_3 = 1/2$, $V_4 = 5/4$, oppure $V_s = 0$, $V_1 = 2$, $V_2 = 1$, $V_3 = 3/2$, $V_4 = 2$, oppure $V_s = 0$, $V_1 = 5/2$, $V_2 = 2$, $V_3 = 2$, $V_4 = 3$. Delle tre soluzioni solo l'ultima corrisponde ai valori dei cammini ottimi.

In entrambi gli esempi le soluzioni dell'equazione di Bellman corrispondenti ai valori dei cammini ottimi sono quelle con i valori più alti. Si può effettivamente dimostrare che deve essere così. Infatti:

9.9 Lemma. *Sia $\hat{P}$ un cammino minimo (massimo) con valori $\hat{V}_i := V(\hat{P}_i)$ e sia $\tilde{V}$ una soluzione dell'equazione di Bellman. Allora $\tilde{V}_i \le \hat{V}_i$ ($\tilde{V}_i \ge \hat{V}_i$) per ogni $i \in \hat{P}$ se le funzioni f_{ij} sono monotone.*

Dimostrazione. (Per problemi di minimo). La dimostrazione procede per induzione. Si ha $\tilde{V}_s \le \hat{V}_s$ (infatti $\tilde{V}_s = \hat{V}_s = 0$). Siano i e j due nodi successivi su $\hat{P}$ e si supponga $\tilde{V}_i \le \hat{V}_i$. Allora

$$\tilde{V}_j = \min_{k \in N_j^-} f_{kj}(\tilde{V}_k) \le f_{ij}(\tilde{V}_i) \le f_{ij}(\hat{V}_i) = \hat{V}_j$$

∎

Le soluzioni spurie possono essere eliminate trasformando l'equazione di Bellman nel seguente problema di ottimizzazione (per problemi di minimo):

$$\begin{aligned} \max \quad & \sum_i V_i \\ & V_j \le f_{ij}(V_i) \qquad \forall (i,j) \in E \\ & V_s = 0. \end{aligned} \tag{9.3}$$

Si noti che (9.3) diventa un problema di programmazione lineare, in particolare un problema di massima tensione, se le funzioni f sono additive.

9.10 Teorema. *Le soluzioni ottime di (9.3) soddisfano l'equazione di Bellman e corrispondono ai valori dei cammini ottimi.*

Dimostrazione. Sia V_i una soluzione ammissibile in (9.3). Se esiste un nodo j tale che $V_j < f_{ij}(V_i)$ per ogni $i \in N_j^-$, allora esiste un'altra soluzione ammissibile V', con $V'_i := V_i$ se $i \ne j$ e $V'_j := \min_{i \in N_j^-} f_{ij}(V_i) > V_j$. La monotonia delle funzioni f garantisce l'ammissibilità di V' in quanto $V_k = V'_k \le f_{jk}(V_j) \le f_{jk}(V'_j)$ per ogni $(j,k) \in E$. Quindi V non può essere ottimo. Necessariamente $\hat{V}_j := \min_{i \in N_j^-} f_{ij}(\hat{V}_i)$ per ogni soluzione ottima. Inoltre ogni soluzione dell'equazione di Bellman è ammissibile in (9.3). In base al lemma 9.9 l'ottimo è la soluzione corrispondente ai cammini ottimi. ∎

L'ipotesi di monotonia delle funzioni f_{ij} è, come si è visto, fondamentale nel poter applicare il principio di ottimalità, che, come vedremo nella prossima sezione, permetterà la costruzione di algoritmi particolarmente efficienti. Per così dire, se un problema si può modellare in modo che ci sia la monotonia (o il principio di ottimalità) questo significa che si tratta di un problema 'ben condizionato'. Infatti, se l'ipotesi di monotonia non dovesse valere, anche solo per un arco, trovare un cammino minimo può essere **NP**-difficile. Si consideri una istanza del problema della partizione con dati $a_1, \ldots, a_n$ (sia $b := \sum_i a_i/2$) e si

costruisca il seguente grafo orientato: $N = \{0, 1, 2, \ldots, n, n+1\}$ ed archi $e_i^0 = (i-1, i)$, $e_i^1 = (i-1, i)$, $i := 1, \ldots, n$, $e_* = (n, n+1)$. Sia

$$f_{e_i^0}(V) := V \qquad f_{e_i^1}(V) := V + a_i \qquad f_{e_*}(V) := |V - b|$$

Si vede subito che esiste un cammino di costo zero se e solo se l'istanza della partizione è ammissibile (se il cammino percorre gli archi e_i^0 il numero i-mo non è scelto, se invece percorre gli archi e_i^1 il numero i-mo è scelto, per cui il costo del cammino nel nodo n è pari alla somma dei numeri scelti, e l'ultimo arco 'decide' quale sia il cammino che partiziona i numeri).

Nella formulazione all'indietro si considerano cammini $P(i) : i \to t$ orientati da un un generico nodo i ad un nodo fissato t. Se j è un nodo sul cammino $P(i)$ indichiamo con $P^j(i)$ la restrizione del cammino $P(i)$ dal nodo j a t. Se il nodo che segue i nel cammino P è il nodo j, il costo $U(P^i)$ di raggiungere la destinazione t si esprime come funzione solamente del costo $U(P^j)$, cioè $U(P^i) = g_{ij}(U(P^j))$, ed è indipendente dal cammino che da j porta a t. Quindi definiremo per ogni cammino $P : i \to t$

$$\begin{aligned} &U(P^t) = 0 \\ &U(P^i) = g_{ij}(U(P^j)) \qquad \forall (i,j) \in P \end{aligned} \tag{9.4}$$

Nella maggior parte dei casi le funzioni definite in (9.4) sono di tipo additivo, cioè

$$U(P^i) = c_{ij} + U(P^j) \qquad \forall (i,j) \in E \tag{9.5}$$

Come conseguenza di (9.4) il costo del cammino $P : s \to k \to \ldots i \to\!\!\to j \to t$ è dato da

$$U(P^s) = g_{sk}(\ldots(\, g_{ij}(\, g_{jt}(0))))$$

e l'obiettivo consiste nel trovare un cammino $P : s \to t$ che minimizza (o massimizza) $U(P^s)$. Analogamente alla formulazione in avanti abbiamo i seguenti risultati:

9.11 Definizione. (Principio di ottimalità - formulazione all'indietro) *Un cammino ottimo $\hat{P}$ soddisfa il principio di ottimalità se, per ogni nodo k sul cammino $\hat{P}$, $\hat{P}^k$ minimizza $U(P(k))$.* ∎

9.12 Teorema. *Siano g_{ij} strettamente monotone. Allora vale il principio di ottimalità per ogni cammino ottimo.* ∎

9.13 Lemma. *Siano g_{ij} monotone e superlineari (sublineari). Allora esiste un cammino minimo (massimo) da ogni nodo j a t.* ∎

9.14 Lemma. *Siano le funzioni g_{ij} monotone e superlineari (sublineari). Allora, per ogni nodo k in un cammino minimo (massimo) semplice $P(j)$, esiste un cammino minimo (massimo) $P(k)$ che non contiene il nodo j.* ∎

9.15 Teorema. *Siano le funzioni g_{ij} monotone e superlineari (sublineari). Allora esiste, per ogni nodo j, un cammino minimo (massimo) $\hat{P}(j)$ per il quale vale il principio di ottimalità.*

L'equazione di Bellman è, nella formulazione all'indietro, per problemi di minimo:

$$U^i = \min_{j \in N_i^+} g_{ij}(U^j) \qquad U_t = 0 \tag{9.6}$$

e per problemi di massimo

$$U^i = \max_{i \in N_j^+} g_{ij}(U^j) \qquad U_t = 0$$

dove $N_j^+ := \{i \in N : (i,j) \in E\}$.

9.16 Teorema. *Se il principio di ottimalità vale per almeno un cammino ottimo $\hat{P}(i)$, per ogni nodo i, allora i valori ottimi $\hat{U}^i := U(\hat{P}(i))$ soddisfano l'equazione di Bellman.* ∎

9.17 Teorema. *Siano g_{ij} monotone ed esistano cammini ottimi $\hat{P}(j)$ da ogni nodo j. Allora i valori ottimi U^j soddisfano l'equazione di Bellman.* ∎

9.18 Teorema. *Siano g_{ij} monotone e continue ed esistano valori ottimi limitati. Allora i valori ottimi soddisfano l'equazione di Bellman.* ∎

9.19 Lemma. *Sia $\hat{P}$ un cammino minimo (massimo) con valori $\hat{U}^i := U(\hat{P}^i)$ e sia $\tilde{U}$ una soluzione dell'equazione di Bellman. Allora $\tilde{U}^i \leq \hat{U}^i$ ($\tilde{U}^i \geq \hat{U}^i$) per ogni $i \in \hat{P}$ se le funzioni g_{ij} sono monotone.* ∎

Anche nella formulazione all'indietro le soluzioni spurie possono essere eliminate trasformando l'equazione di Bellman nel seguente problema di ottimizzazione (per problemi di minimo):

$$\begin{aligned} \max \quad & \sum_i U^i \\ & U^i \leq g_{ij}(U^j) \qquad \forall (i,j) \in E \\ & U^t = 0. \end{aligned}$$

È importante rilevare che la formulazione in avanti e quella all'indietro rappresentano problemi diversi a meno che le funzioni siano additive come in (9.2) e (9.5). In questo caso le formulazioni sono equivalenti come espresso dal seguente teorema:

9.20 Teorema. *Siano s e t due nodi fissati e sia P un generico cammino $s \to t$. La condizione $V(P_k) + U(P^k) = V(P_t) = U(P^s)$ è soddisfatta per ogni nodo k in P se e solo se f e g sono funzioni additive.*

Dimostrazione. La sufficienza è ovvia. Per la necessità si supponga che $V(P_i) + U(P^i) = V(P_j) + U(P^j)$ con (i,j) arco di P. Allora $V(P_i) + g_{ij}(U(P^j)) = f_{ij}(V(P_i)) + U(P^j)$, cioè $f_{ij}(V(P_i)) = V(P_i) + g_{ij}(U(P^j)) - U(P^j)$. La funzione f_{ij} non può dipendere dal cammino successivo al nodo j, e quindi l'espressione $g_{ij}(U(P^j)) - U(P^j)$ deve essere invariante rispetto a P^j e dipendere solo dall'arco (i,j), cioè $g_{ij}(U(P^j)) - U(P^j) =: c'_{ij}$. In modo simile $f_{ij}(V(P_i)) - V(P_i) =: c''_{ij}$ e, confrontando le espressioni $c'_{ij} = c''_{ij} =: c_{ij}$. ∎

Algoritmo Bellman-Ford
input(G, f)
    $V_s := 0$; **for** all $i \neq s$ **do** $V_i := \infty$; $p(s) := s$;
    for $k := 1$ **to** n **do**
        for $j := 1$ **to** n **do**
        begin
            $h := \operatorname{argmin}\{f_{ij}(V_i) : i \in N_j^-\}$;
            $V_j := \min\{V_j\ ;\ f_{hj}(V_h)\}$;
            if $V_j = f_{hj}(V_h)$ **then** $p(j) := h$;
        end
output(V, p).

9.2. Algoritmi di programmazione dinamica

L'equazione di Bellman, vista come un'equazione di punto fisso suggerisce l'algoritmo noto con il nome di *Bellman-Ford*.

9.21 Teorema. *Se le funzioni f_{ij} sono strettamente monotone l'algoritmo Bellman-Ford in tempo $O(n\,m)$ o trova tutti i cammini ottimi $s \to i$, per ogni i, oppure scopre che non ci sono cammini ottimi.*

Dimostrazione. Si indichi con V_j^k il valore di V_j dopo la k-esima iterazione dell'algoritmo (V_j^0 è il valore iniziale). Se $V_j^{k+1} < V_j^k$, allora esiste un nodo i tale che $V_j^{k+1} = f_{ij}(V_i^k)$. In base all'algoritmo $V_j^k \leq f_{ij}(V_i^{k-1})$ per $k \geq 1$. Le tre relazioni implicano $f_{ij}(V_i^k) < f_{ij}(V_i^{k-1})$, da cui, per la monotonia delle funzioni (basta la monotonia semplice), $V_i^k < V_i^{k-1}$. Applicando questo ragionamento ricorsivamente esiste una sequenza di nodi $i_0, i_1, \ldots$ tali che $V_{i_h}^{k-h+1} < V_{i_h}^{k-h}$. Se $k > n$ ci deve essere almeno un nodo ripetuto $r := i_{p-q} = i_p$ nella sequenza e per tale nodo si ha

$$V_r^{k-p+q+1} < V_r^{k-p+q} \leq V_r^{k-p+1} < V_r^{k-p}$$

I nodi $i_p, i_{p-1}, \ldots, i_{p-q}$ formano un ciclo orientato C. Si indichi con F_C la composizione delle funzioni f_{ij} lungo il ciclo. Quindi si ha $F_C(V_r) < V_r$. La stretta monotonia delle f_{ij} implica la stretta monotonia di F_C e quindi $F_C(F_C(V_r)) < F_C(V_r) < V_r$. Di conseguenza esiste una successione $P_r(q)$ di cammini ognuno dei quali percorre q volte il ciclo C e tali che $V(P_r(q)) < V(P_r(q-1))$. Quindi non c'è soluzione e il problema è illimitato se la successione $V(P_r(q))$ non ha una limitazione inferiore (ad esempio con funzioni additive).

Quindi se esiste una soluzione ottima i valori V_j non possono essere aggiornati dopo l'n-esima iterazione. La complessità computazionale è ovvia. Resta però ancora da dimostrare che i valori ottenuti sono ottimi. Finora si è solo dimostrato che soddisfano l'equazione di Bellman. Procedendo per induzione si supponga che ad un generico passo d'iterazione i valori V_j^k soddisfino le diseguaglianze $V_j^k \geq \hat{V}_j$ con $\hat{V}_j$ valori ottimi. Allora si ha

$$V_j^{k+1} = \min\{V_j^k\ ;\ f_{ij}(V_i^k)\} \geq \min\left\{\hat{V}_j\ ;\ f_{ij}(\hat{V}_i)\right\} = \hat{V}_j$$

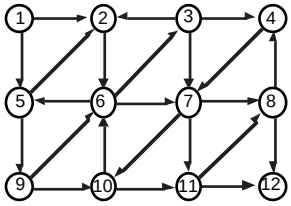
dove si è sfruttata la monotonia e il fatto che i valori ottimi soddisfano l'equazione di Bellman. Quindi l'ipotesi induttiva è vera anche all'iterazione successiva e siccome è vera al passo iniziale è vera sempre. Da questa, usando il lemma 9.9, discende la tesi. I valori $p(i)$ sono puntatori all'indietro necessari per ricostruire il cammino ottimo partendo da t, ponendo $h := t$ e iterando $h := p(h)$ fino a $h = p(h)$. ∎

Senza l'ipotesi di stretta monotonia l'algoritmo di Bellman Ford potrebbe richiedere un numero arbitrariamente elevato di iterazioni prima di trovare la soluzione ottima. Ad esempio se $f_{st}(V) := \max\{V-1, -3\,K\}$, $f_{t1}(V) := V-1$, $f_{1s}(V) := V-1$, sono necessarie $K\,n$ iterazioni. Tuttavia, introducendo l'ipotesi di superlinearità (cioè $f(V) \geq V$), soddisfatta in molti casi, che garantisce l'esistenza di cammini ottimi, si ottiene la validità dell'algoritmo di Bellman-Ford.

9.22 Teorema. *Se le funzioni f_{ij} sono monotone e superlineari l'algoritmo Bellman-Ford trova tutti i cammini ottimi $s \to i$, per ogni i, in tempo $O(n\,m)$.*

Dimostrazione. La dimostrazione procede come nel teorema 9.21 fino alla diseguaglianza $F_C(V_r) < V_r$. Ma, per la superlinearità, tale diseguaglianza non può verificarsi e quindi l'algoritmo deve trovare gli ottimi entro l'n-esima iterazione. ∎

9.23 Esempio. Sia dato il seguente grafo orientato con i costi indicati degli archi:



$c_{1,2} = -1,\ c_{1,5} = 10,\ c_{2,6} = -1,\ c_{3,2} = 8,\ c_{3,4} = 4,\ c_{3,7} = 10,$

$c_{4,7} = 0,\ c_{5,2} = 7,\ c_{5,9} = 0,\ c_{6,3} = -3,\ c_{6,5} = 3,\ c_{6,7} = 7,$

$c_{7,8} = 5,\ c_{7,10} = 6,\ c_{7,11} = 8,\ c_{8,4} = 0,\ c_{8,12} = -5,\ c_{9,6} = -2,$

$c_{9,10} = 3,\ c_{10,6} = 3,\ c_{10,11} = 0,\ c_{11,8} = 10,\ c_{11,12} = -4$

Vogliamo calcolare il cammino minimo da 1 a 12. Quindi le funzioni f_{ij} sono di tipo additivo $f_{ij}(V) = V + c_{ij}$ e sono strettamente monotone (anche se $c_{ij} < 0$) e quindi si può applicare l'algoritmo di Bellman-Ford. L'algoritmo dà i seguenti valori di V_i:

V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}	V_{11}	V_{12}
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
0	-1	-5	10	1	-2	5	10	10	11	11	5
0	-1	-5	-1	1	-2	-1	4	1	4	4	-1
0	-1	-5	-1	1	-2	-1	4	1	4	4	-1

Il vettore di puntatori è $\{0, 1, 6, 3, 6, 2, 4, 7, 5, 9, 10, 8\}$ che fornisce il cammino minimo $1 \to 2 \to 6 \to 3 \to 4 \to 7 \to 8 \to 12$. Come si vede i valori diventano definitivi già alla terza iterazione. Infatti avviene spesso che, se l'istanza non è illimitata, bastano meno delle n iterazioni teoriche. Questo succede perché i valori aggiornati di V_i vengono usati già all'interno dell'iterazione. Se il valore di $c_{7,10}$ viene modificato da 6 in -5 si ottengono invece i seguenti valori:

V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}	V_{11}	V_{12}
0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
0	-1	-5	10	1	-2	5	10	10	0	0	-4
0	-1	-5	-1	1	-3	-1	4	1	-6	-6	-10
0	-1	-6	-1	0	-3	-1	4	1	-6	-6	-10
0	-1	-6	-2	0	-4	-2	3	0	-7	-7	-11
0	-1	-7	-2	-1	-4	-2	3	0	-7	-7	-11
0	-1	-7	-3	-1	-5	-3	2	-1	-8	-8	-12
0	-1	-8	-3	-2	-5	-3	2	-1	-8	-8	-12
0	-1	-8	-4	-2	-6	-4	1	-2	-9	-9	-13
0	-1	-9	-4	-3	-6	-4	1	-2	-9	-9	-13
0	-1	-9	-5	-3	-7	-5	0	-3	-10	-10	-14
0	-1	-10	-5	-4	-7	-5	0	-3	-10	-10	-14
0	-2	-10	-6	-4	-8	-6	-1	-4	-11	-11	-15
0	-2	-11	-6	-5	-8	-6	-1	-4	-11	-11	-15

e quindi dobbiamo concludere che il problema è illimitato. Infatti il ciclo $3 \to 4 \to 7 \to 10 \to 6 \to 3$ ha lunghezza negativa. ∎

9.24 Esempio. Si consideri il seguente problema: una stazione s deve trasmettere un messaggio ad un'altra stazione t. Il messaggio, per essere trasmesso, deve essere inviato ad un satellite. Successivamente il messaggio può essere spedito da un satellite ad un altro satellite oppure da un satellite alla stazione t. Il problema sorge dal fatto che, a causa della rivoluzione dei satelliti attorno alla Terra, non sempre i satelliti sono visibili fra loro oppure dalle due stazioni a terra s e t. Si producono quindi dei ritardi nella trasmissione dovuti alle attese delle finestre di visibilità. Possiamo supporre istantanea la trasmissione di un messaggio, qualora vi sia visibilità. L'obiettivo è quello di trovare una rotta per il messaggio che minimizzi il ritardo.

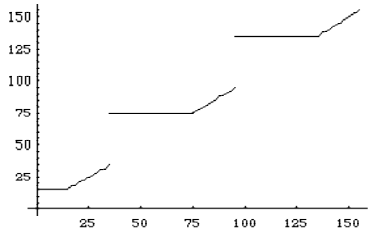
Supponiamo che la rivoluzione di ogni satellite duri esattamente un'ora e che le finestre di visibilità si ripetano periodicamente ogni ora. Quindi per ogni coppia di satelliti viene assegnato un intervallo espresso in minuti che corrisponde alla finestra di visibilità. Supponiamo che vi siano 6 satelliti, numerati da 2 a 7; numeriamo come 1 la stazione s e come 8 la stazione t. Il messaggio parte da 1 al minuto t_0. I dati relativi alle finestre temporali sono i seguenti:

	1	2	3	4	5	6	7	8
1	[∗,∗]	[44,51]	[26,32]	[39,44]	[11,18]	[43,48]	[45,53]	[∗,∗]
2	[44,51]	[∗,∗]	[5,11]	[27,32]	[14,21]	[13,21]	[26,34]	[3,12]
3	[26,32]	[5,11]	[∗,∗]	[10,18]	[26,36]	[4,12]	[2,9]	[46,52]
4	[39,44]	[27,32]	[10,18]	[∗,∗]	[23,33]	[48,53]	[27,36]	[38,44]
5	[11,18]	[14,21]	[26,36]	[23,33]	[∗,∗]	[14,22]	[37,42]	[2,11]
6	[43,48]	[13,21]	[4,12]	[48,53]	[14,22]	[∗,∗]	[49,57]	[20,25]
7	[45,53]	[26,34]	[2,9]	[27,36]	[37,42]	[49,57]	[∗,∗]	[33,39]
8	[∗,∗]	[3,12]	[46,52]	[38,44]	[2,11]	[20,25]	[33,39]	[∗,∗]

Il problema può essere convenientemente modellato con la programmazione dinamica. I nodi del grafo sono le due stazioni e i 6 satelliti (numerati come sopra) e gli archi sono costituiti da tutte le coppie ordinate (i, j) tranne $(1, 8)$ e $(8, 1)$. La lunghezza V del cammino può essere identificata con la differenza fra l'istante di arrivo del messaggio nel nodo e t_0 per cui le funzioni f_{ij} sono

$$f_{i,j}(V) := \begin{cases} V & \text{se } (t_0 + V - a_{ij}) \bmod 60 \leq b_{ij} - a_{ij} \\ V + ((a_{ij} - t_0 - V) \bmod 60) & \text{altrimenti} \end{cases}$$

dove $[a_{ij}, b_{ij}]$ è la finestra temporale per l'arco (i, j). Si veda la figura dove si è scelto $t_0 := 10$, $a := 25$, $b := 45$.



Si noti che la funzione non è strettamente monotona, ma è superlineare e quindi l'algoritmo di Bellman-Ford può essere applicato. Si ottengono i seguenti valori di V:

V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8
0	∞	∞	∞	∞	∞	∞	∞
0	44	26	39	11	43	45	∞
0	14	26	23	11	14	43	20
0	14	26	23	11	14	26	20
0	14	26	23	11	14	26	20

e i valori dei puntatori sono $p = \{0, 5, 1, 5, 1, 5, 2, 6\}$ per cui il cammino ottimo è $1 \to 5 \to 6 \to 8$. I ritardi che si accumulano su questo cammino sono quindi: 11 nella stazione di partenza, 3 nel satellite 5 e 6 nel satellite 6. Come si deve risolvere il problema se il ritardo accumulato nella stazione di terra non conta? ∎

Nel caso additivo è possibile derivare dall'algoritmo di Bellman-Ford un algoritmo che trova al medesimo tempo i cammini minimi $i \to j$ per *tutte* le coppie di nodi i e j. Un'applicazione ingenua dell'algoritmo Bellman-Ford richiederebbe un tempo $O(n^2 m)$ (scegliendo come sorgente ad ogni ripetizione dell'algoritmo un nodo diverso). Tuttavia si può notare come molte operazioni verrebbero replicate diverse volte con notevole spreco di tempo. Ad esempio, se il cammino ottimo $s \to j$ passa per k, l'algoritmo calcolerebbe i tre cammini $s \to j$, $s \to k$ e $k \to j$, quando invece solo un cammino deve essere calcolato esplicitamente a causa del principio di ottimalità. Razionalizzando l'algoritmo di Bellman-Ford nel caso di calcolo dei cammini minimi per tutte le coppie si ottiene l'algoritmo sviluppato indipendentemente da Floyd [1962] e Warshall [1962].

9.25 Teorema. *L'algoritmo Floyd-Warshall o trova i cammini ottimi fra tutte le coppie di nodi o trova un ciclo di lunghezza negativa (e quindi un'istanza illimitata) in tempo $O(n^3)$.*

Dimostrazione. La complessità computazionale è ovvia. Per ciò che riguarda la correttezza dell'algoritmo conviene usare l'induzione sui valori k del ciclo più esterno dell'algoritmo. L'ipotesi dell'induzione è che al passo k , $V_j(i)$ rappresenta il valore ottimo fra tutti i cammini $i \to j$ *con il vincolo di usare come nodi intermedi solo i nodi* $1, \ldots, k$ e $V_i(i)$ rappresenta il valore ottimo fra tutti i cicli passanti per il nodo i e *con il vincolo di usare come altri nodi solo i nodi* $1, \ldots, k$. Si indichi con $V_j^k(i)$ il valore $V_j(i)$ al passo k-esimo.

L'ipotesi è ovviamente vera per $k = 0$. Quindi supponiamo sia vera per $k-1$. Aggiungendo il nodo k ai nodi ammissibili si permette a tutti i cammini $i \to j$ (ed a tutti i cicli per i) di passare per k. Il cammino ottimo $i \to j$ vincolato a passare per k (e in modo simile il ciclo ottimo oer i e k) consiste di due sottocammini $i \to k$ e $k \to j$ ($k \to i$ per i cicli). Per il principio di ottimalità questi sottocammini devono essere ottimi fra i sottocammini che usano solo i nodi $1, \ldots k-1$ e i loro valori ottimi sono $V_k^{k-1}(i)$ e $V_j^{k-1}(\ k)$ ($V_i^{k-1}(k)$ per i cicli). Quindi il valore ottimo $V_j^k(i)$ viene calcolato confrontando la loro somma con $V_j^{k-1}(i)$ (in modo simile si calcola il ciclo ottimo). ∎

Algoritmo Floyd-Warshall
input(c);
  $V_j(i) := c_{ij}$, $\forall i \neq j$; $V_i(i) := +\infty$, $\forall i$;
  for $k := 1$ to n **do**
    for $i := 1$ to n **do**
      for $j := 1$ to n **do**
        $V_j(i) := \min\{V_j(i)\ ;\ V_k(i) + V_j(k)\}$;
output(V).

Algoritmo PDA (Programmazione Dinamica Aciclica)
input(G, f);
  $Q := \{s\}$; $p(s) := s$; $n_j := |N_j^-|$, $\forall j$;
  while $t \notin Q$ **do**
  begin
    sia $i \in Q$; $Q := Q \setminus \{i\}$;
    for all $j \in N_i^+$ **do**
    begin
      $V_j := \min\{V_j\ ;\ f_{ij}(V_i)\}$;
      if $V_j = f_{ij}(V_i)$ **then** $p(j) := i$;
      $n_j := n_j - 1$;
      if $n_j = 0$ **then** $Q := Q \cup \{j\}$;
    end
  end
output(V, p).

Se il grafo è aciclico esistono sempre soluzioni ottime dato che tutti i cammini sono semplici ed in numero finito. Quindi in base al teorema 9.5 basta la monotonia semplice delle funzioni f_{ij} a garantire la validità del principio di ottimalità per almeno un cammino e quindi l'applicabilità dell'equazione di Bellman. Inoltre l'aciclicità permette algoritmi più veloci.

In un grafo aciclico la risoluzione dell'equazione di Bellman non presenta problemi in quanto i valori di sinistra possono sempre essere calcolati da valori di destra già calcolati. Infatti inizialmente è noto (per definizione) $V_s = 0$. In un grafo aciclico esiste sempre almeno un nodo senza predecessori. Supporremo, in modo naturale, che il nodo s sia senza predecessori e anche che sia l'unico nodo senza predecessori. Sia inizialmente $S := \{s\}$ l'insieme dei nodi per i quali il valore di V è definitivo. Consideriamo il grafo $G(S) := (N \setminus S, E(N \setminus S))$ che risulta anch'esso aciclico. I nodi senza predecessori in $G(S)$ (e ne esiste sempre almeno uno) hanno predecessori in S e quindi il loro valore ottimo è calcolabile esplicitamente dall'equazione di Bellman ed è definitivo. Si aggiungano questi nodi ad S e si proceda ricorsivamente finché $t \in S$.

Algoritmicamente conviene procedere come nell'algoritmo PDA dove, non appena il valore V_i di un nodo i è definitivo, si calcola immediatamente $f_{ij}(V_i)$ e lo si confronta con un valore provvisorio V_j, eventualmente aggiornando quest'ultimo. Per sapere quando il valore V_j diventa definitivo è efficiente decrementare di uno ad ogni confronto un contatore che viene inizializzato al numero di predecessori di j.

La complessità computazionale è data dal numero di volte le funzioni f_{ij} vengono calcolate e i confronti vengono effettuati. Ovviamente tale numero è uguale al numero di archi del grafo. Pertanto possiamo concludere (nell'ipotesi che il calcolo di f_{ij} abbia complessità costante e tenendo conto anche di alcuni ragionamenti fatti nel teorema 9.21) :

9.26 Teorema. *Se le funzioni f_{ij} sono monotone l'algoritmo PDA trova tutti i cammini ottimi $s \to i$, per ogni i, in tempo $O(m)$.* ∎

In molti modelli di programmazione dinamica il grafo è a livelli, come ad esempio in tutte le formulazioni in termini di stadi e di stati. In questi casi il grafo è ovviamente aciclico e per di più è già noto a priori quando i valori di V_i diventano definitivi. Basta infatti eseguire i calcoli livello per livello.

9.3. Cammini minimi rivisitati

La programmazione dinamica permette di trovare velocemente cammini minimi non necessariamente elementari su grafi orientati, anche nel caso di lunghezze negative. Sorge naturale la domanda se si possono ottenere risultati analoghi nei casi in cui si imponga il vincolo di cammino elementare, oppure nei casi in cui il grafo è non orientato.

Si è visto che l'esistenza di un ciclo a lunghezza negativa costituisce una 'trappola' per tutti i cammini in quanto ogni convoluzione lungo il ciclo produce un miglioramento nella lunghezza del cammino. Gli algoritmi visti precedentemente non sono in grado di evitare cammini non elementari quando questi si rivelino migliori. Infatti imporre la condizione aggiuntiva che i cammini debbano essere elementari trasforma il problema in uno molto più difficile. Un'istanza con $c_{ij} = -1$ per tutti gli archi corrisponde a trovare il cammino più lungo nel grafo, e, a seconda della soluzione, siamo in grado di dire se il grafo ammette un cammino hamiltoniano oppure no. Siccome trovare un cammino hamiltoniano è **NP**-difficile, trovare cammini minimi elementari in presenza di costi negativi è ugualmente **NP**-difficile.

Se si deve risolvere un problema su un grafo non orientato con distanze non negative (in un grafo non orientato si può andare sia da i a j che da j a i al medesimo costo c_{ij}) è possibile trasformare il problema in uno definito su un grafo orientato semplicemente sostituendo ogni arco (i, j) con due archi antiparalleli (i, j) e (j, i). Ovviamente questa trasformazione funziona per distanze non negative giacché nessun cammino trova conveniente inserire un ciclo $i \to j \to i$. Quindi al più uno dei due archi (i, j) e (j, i) verrà usato e la soluzione ottenuta per il grafo orientato può essere reinterpretata per il grafo non orientato originario. Tuttavia l'applicazione dell'algoritmo di Bellman-Ford a questa rete trasformata porterebbe ad una complessità $O(n\,m)$, che risulta troppo elevata.

L'algoritmo di Dijkstra, di complessità $O(n^2)$ o $O(m \log n)$ a seconda dell'implementazione, può esser visto anche come un algoritmo di programmazione dinamica che fa uso di un intelligente aggiornamento dei valori V_j, applicabile però soltanto a costi additivi con distanze non negative. Come in un grafo aciclico alcuni valori di V_j devono essere definitivi e quindi usabili per risolvere l'equazione di Bellman, così in questo caso, pur non essendo il grafo aciclico, ci sono nodi i cui valori V_j sono definitivi.

L'esistenza di una tale proprietà si basa sul seguente ragionamento induttivo: durante l'iterazione sia noto un insieme S di nodi i cui valori V_j, $j \in S$, sono definitivi e siano noti dei valori V_j, $j \notin S$, che rappresentano i costi dei cammini ottimi da s a j con il vincolo di usare come nodi intermedi soltanto nodi di S (se il cammino non esiste $V_j := \infty$). Ora sia k tale che $V_k = \min_{j \notin S} V_j$. V_k è il valore finale ottimo per i cammini $s \to k$. Infatti ogni altro cammino dovrebbe passare per qualche altro nodo non in S ad un costo più elevato a causa della scelta di k e delle distanze non negative. A questo punto si può aggiornare S ponendo $S := S \cup \{k\}$. Per aggiornare V_j, $j \notin S$, bisogna tener conto della nuova opzione di passare per k. Sfruttando il principio di ottimalità si pone quindi $V_j := \min\{V_j\,;\, V_k + c_{kj}\}$. La proprietà è ora verificata per un insieme più grande di nodi e si può iterare quindi fino a che $t \in S$. La proprietà è certamente vera inizialmente per $S = \{s\}$ e $V_j = c_{sj}$ se esiste l'arco (s, j) e $V_j = \infty$ altrimenti. Si è quindi dimostrata la correttezza dell'algoritmo di Dijkstra per una via diversa da quella usata nel capitolo 8.

Se ammettiamo costi negativi ma anche supponiamo che non esistano cicli negativi, allora nel caso orientato l'algoritmo di Bellman-Ford è quanto di meglio si possa fare. Nel caso non orientato il trucco di sostituire ogni arco non orientato con la coppia di archi orientati ed opposti non funziona, perché si vengono a creare piccoli cicli negativi in corrispondenza di ogni arco negativo e questo fatto rende illimitata in ogni caso l'istanza sul grafo orientato.

Il problema di scoprire cicli elementari negativi in un grafo non orientato può essere invece affrontato tramite la seguente trasformazione da un grafo non orientato con n nodi e m archi

in un grafo non orientato con $2n + 2m$ nodi e $5m + n$ archi:

– ogni nodo i produce una coppia di nodi, etichettati i e i', ed un arco, etichettato (i, i');

– ogni arco (i, j) produce due nodi, etichettati iij e ijj, e 5 archi, etichettati (i, iij), (i', iij), (iij, ijj), (ijj, j) e (ijj, j');

– i costi degli archi sono assegnati come $c(i, iij) = c(i', iij) = c(ijj, j) = c(ijj, j') = c_{ij}$ e $c(i, i') = c(iij, ijj) = 0$.

Sul grafo trasformato si risolve un problema di accoppiamento di minimo costo. Si vede che ogni accoppiamento corrisponde ad un inseme di circuiti nel grafo originale con costo due volte quello dei circuiti. Se non c'è un ciclo di lunghezza negativa l'accoppiamento minimo corrisponde agli archi di costo zero (nel grafo trasformato), mentre se c'è un ciclo negativo esiste anche un accoppiamento di costo negativo che include almeno un ciclo negativo. Si veda la figura 9.1 dove sono stati disegnati un grafo con un ciclo negativo e il corrispondente grafo trasformato. Si è anche evidenziato l'accoppiamento ottimo (le coppie di nodi i e i' sono disegnate in nero).

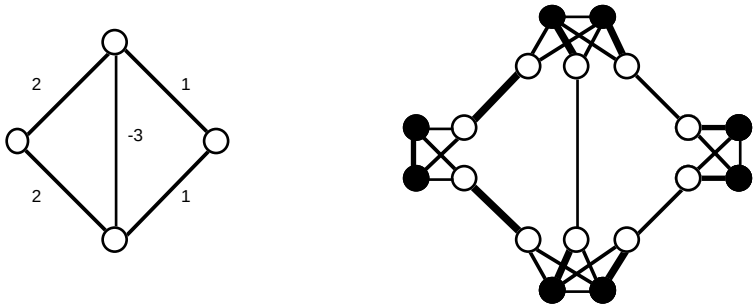


FIGURA 9.1

Si può estendere questa costruzione al problema di trovare un cammino minimo in una rete con costi generici ma senza cicli negativi. Si indichi con TC la precedente trasformazione e con TP quella che verrà ora definita. La differenza fra TC e TP riguarda i nodi s e t e i loro archi. I nodi s e t non vengono duplicati cosicché gli archi (s, i) sono trasformati in (s, ssi), (ssi, sii), (sii, i) e (sii, i') (e similmente per t). Ora ogni accoppiamento nel grafo trasformato si traduce nel grafo originale in un cammino $s \to t$ più eventualmente dei cicli. Tuttavia, se non vi sono cicli negativi, si ottiene soltanto un cammino.

Se si rilassa la condizione sulle lunghezze di ciclo ammettendo un valore qualsiasi, il problema diventa **NP**-difficile se si richiede un cammino elementare, ma rimane polinomiale se ammettiamo cammini qualsiasi. In questo caso bisogna dapprima scoprire se esistono cicli negativi usando la trasformazione TC. Se ne esiste uno l'istanza è illimitata, altrimenti si applica la trasformazione TP. Quindi si tratta di risolvere due problemi d'accoppiamento. Risolvere soltanto un problema d'accoppiamento con la trasformazione TP non fornisce la risposta corretta in generale come si vede nell'esempio in figura 9.2, che risulta illimitato ammettendo cammini non elementari, però la trasformazione TP fornisce il cammino minimo elementare (indicato in figura). Nella figura 9.3 si vede il corrispondente grafo trasformato con l'accoppiamento.

In figura 9.4 viene riassunta la complessità computazionale di diversi problemi di cammino minimo. La prima colonna della tabella si riferisce al caso di costi non negativi. Nella seconda colonna si assume che i costi siano qualsiasi ma non ci siano cicli negativi. Infine nella terza colonna non si fa nessuna ipotesi. Le righe si riferiscono a tipi diversi di grafo. Nella prima riga si considerano grafi orientati aciclici, nella seconda grafi orientati e nella terza grafi non orientati. Nei due riquadri divisi diagonalmente si considerano separatamente i casi

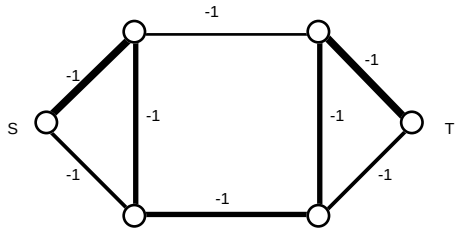


FIGURA 9.2

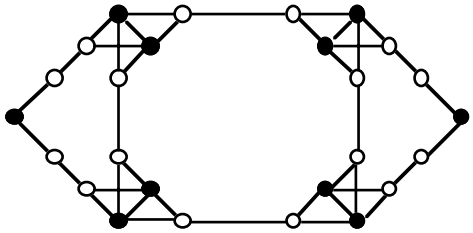


FIGURA 9.3

di cammini qualsiasi (in alto a sinistra) e di cammini elementari (in basso a destra). La distinzione è ovviamente necessaria solo in questi casi.

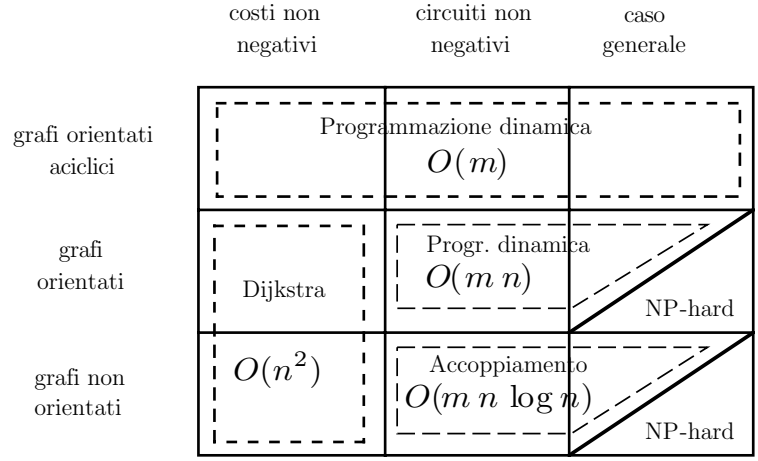
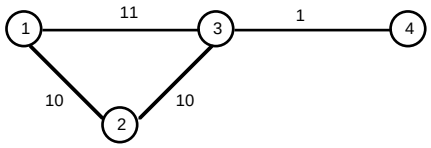


FIGURA 9.4

Può essere interessante far vedere dei casi di problemi di cammino minimo in cui il principio di ottimalità non può essere applicato direttamente. Si supponga di cercare il più lungo (o più corto) cammino medio fra due nodi. Questo tipo di obiettivo potrebbe essere necessario ad esempio se il grafo modella una rete di trasporto e ad ogni arco è associato il profitto dovuto al trasporto di passeggeri fra i nodi (città) corrispondenti. Supponiamo inoltre che sia richiesto un giorno di viaggio fra ogni coppia di nodi. Una volta arrivata a destinazione la corriera continua a compiere il servizio ritornando indietro e così di seguito. La società che gestisce il servizio è interessata al massimo profitto medio, dato che l'orizzonte temporale è molto lungo. Si consideri il seguente esempio:



Il cammino che massimizza il profitto medio è $1 \to 2 \to 3 \to 4$ dato che $(10 + 10 + 1)/3 = 7 > (11 + 1)/2 = 6$. Può essere sorprendente notare che il sottocammino $1 \to 2 \to 3$ *non* è ottimo se restringiamo il servizio al nodo 3. In questo caso il miglior cammino è $1 \to 3$ con profitto medio $11/1 = 11 > (10 + 10)/2 = 10$.

Il motivo per cui non si può applicare direttamente il principio di ottimalità è perché il valore di un cammino non può essere calcolato da un nodo all'altro semplicemente conoscendo il valore accumulato nel nodo. Bisogna anche sapere quanti archi hanno condotto in un particolare nodo. Quindi una corretta formulazione di programmazione dinamica richiede una rete a livelli in cui ogni livello rappresenta il numero di archi del cammino e i nodi del grafo sono ripetuti in ogni livello. Archi da un livello all'altro corrispondono a possibili viaggi giornalieri. Possiamo non ammettere archi da un nodo in un livello al nodo omologo del livello successivo perché tali archi non fanno aumentare il profitto. Inoltre abbiamo bisogno di altri archi che connettano il nodo 4 di ogni livello ad un nodo terminale t (in questo modo la rete risultante non è veramente a livelli, ma ciò non è essenziale). Questi archi sono necessari per rappresentare differenti condizioni di fermata al nodo 4 dopo $1, 2, \ldots$ passi. Se non ammettiamo cicli, la rete ha 4 livelli (n in generale) ed è raffigurata in figura 9.5. Indicando un nodo con i due indici k per il livello e i per il nodo nel livello le funzioni $f_{(ik)(jh)}(V)$ sono definite da:

$$f_{(ik)(j,k+1)}(V) := \frac{kV + c_{ij}}{k+1} \qquad \text{e} \qquad f_{(ik)(tn)}(V) := V$$

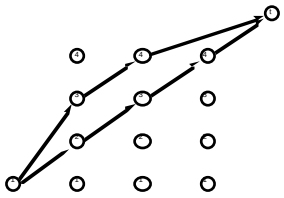


FIGURA 9.5

9.27 ESERCIZIO. Si calcolino i cammini medi più lunghi (e anche quelli più corti) degli esempi in figura 9.6. Sono possibili altri approcci? (Suggerimento: si pensi a cosa succede modificando ogni costo della medesima costante). ∎

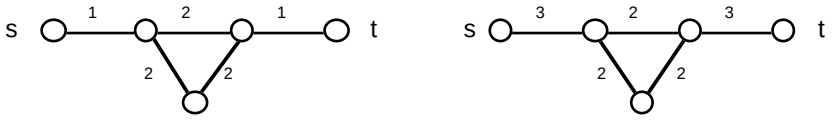


FIGURA 9.6

9.4. Modelli di allocazione

Un'applicazione tipica della programmazione dinamica riguarda i modelli di allocazione. In questi modelli si devono allocare delle risorse scarse all'interno di vincoli prescritti e con un obiettivo definito. L'approccio tramite la programmazione dinamica richiede, come è del resto tipico di questo metodo, la discretizzazione delle variabili da determinare, in modo da poter usare lo schema generale di grafo orientato. Se da un lato la discretizzazione permette una grande flessibilità, in quanto variabili continue e intere vengono trattate alla stessa stregua, dall'altro il prezzo che si paga è un aumento a volte esponenziale nel numero di

nodi del grafo, rendendo quindi proibitivo ogni approccio computazionale. In pratica quindi solo problemi con un numero limitato di vincoli, uno o due, possono essere affrontati.

Siano $x_i \in Z$, $i = 1, \ldots, n$, variabili intere eventualmente vincolate come $l_i \leq x_i \leq u_i$. Ad ogni variabile siano associate una funzione di costo $v_i(x_i)$ ed una funzione di peso a valori interi $w_i(x_i) \geq 0$. Inoltre sia b una capacità specificata. Il modello di allocazione che vogliamo risolvere richiede la massimizzazione di $\sum_i v_i(x_i)$ con il vincolo che $\sum_i w_i(x_i) \leq b$.

Si indichi con $V_k(y)$ l'allocazione ottima delle prime k variabili con il vincolo di capacità y. Allora l'equazione di Bellman può essere scritta come:

$$V_k(y) = \max_{l_k \leq x_k \leq u_k} V_{k-1}(y - w_k(x_k)) + v_k(x_k)$$

dove si pone $V_0(0) := 0$ e $V_0(y) := -\infty$ se $y > 0$.

A questo problema si può associare il seguente grafo a livelli. Ci sono $n+2$ livelli etichettati $0, 1, \ldots, n+1$, e $b+1$ nodi nei livelli $1, \ldots, n$, etichettati (k, i), con $k := 1, \ldots, n$ e $i := 0, 1, \ldots b$. Al livello 0 c'è l'unico nodo $s = (0, 0)$ e al livello $n+1$ c'è l'unico nodo $t = (n+1, 0)$. C'è un arco fra $(k-1, i)$ e (k, j), $1 \leq k \leq n$, se $j - i = w_k(x_k)$ per qualche $l_k \leq x_k \leq u_k$ con una funzione di costo additiva $V + v_k(x_k)$. Il nodo t è collegato a tutti i nodi del livello n a costo zero.

9.28 Esempio. Si consideri il seguente problema:

$$\begin{aligned} \min \quad & 2\,x_1^2 + x_2^2 + 3\,x_3^2 + 4\,x_4^2 \\ & x_1 + x_2 + \quad x_3 + \quad x_4 = 14 \\ & x_i \geq 0 \qquad x_i \text{ intero} \end{aligned}$$

Quindi vi sono 4 livelli e 15 nodi per livello. Gli archi sono del tipo $((k, i), (k+1, j))$ con $j - i \geq 0$ e con funzione

$$f_{((k,i),(k+1,j))}(V) := V + c_k\,(j - i)^2$$

dove c_k è il k-esimo coefficiente della funzione obiettivo. Il calcolo fornisce i seguenti valori ottimi V_i^k:

i =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
V^1 =	0	2	8	18	32	50	72	98	128	162	200	242	288	338	392
V^2 =	0	1	3	6	11	17	24	33	43	54	67	81	96	113	131
V^3 =	0	1	3	6	9	14	20	27	36	45	55	66	79	93	108
V^4 =	0	1	3	6	9	13	18	24	31	40	49	59	70	82	95

da cui l'ottimo $x = \{3, 7, 2, 2\}$ può essere dedotto usando l'informazione contenuta nella seguente tabella di puntatori all'indietro:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	2	2	2	3	3	3	4	4	4	5
0	1	2	2	3	4	5	6	6	7	8	9	10	11	11
0	1	2	3	4	4	5	6	7	8	9	10	11	11	12
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

È importante notare comunque che il calcolo fornisce come sottoprodotto anche le soluzioni ottime per tutti i valori di b fra 0 e 14. Ecco le diverse soluzioni ottime:

b	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
x_1	0	0	1	1	1	1	1	2	2	2	2	3	3	3	3
x_2	0	1	1	1	2	2	3	3	4	4	5	5	6	6	7
x_3	0	0	0	1	1	1	1	1	1	2	2	2	2	2	2
x_4	0	0	0	0	0	1	1	1	1	1	1	1	1	2	2

∎

9.29 Esempio. Si supponga di dover costruire degli impianti per fornire un servizio ad un'area geografica ad un livello prescritto b di servizio (possiamo ad esempio pensare che gli impianti siano centrali elettriche e che b sia l'energia giornaliera richiesta). Un certo numero di siti sono stati selezionati e si deve decidere su quali effettivamente costruire gli impianti e quanto grandi questi debbano essere per soddisfare la domanda al minimo costo.

Indichiamo con x_i il livello richiesto di servizio per l'impianto i. Il costo di costruzione e di gestione di un impianto capace di fornire il servizio al livello x_i non è normalmente una funzione lineare in x_i. Per esempio il costo di costruzione è nullo se non si costruisce l'impianto, ma diventa immediatamente molto elevato anche per un piccolo impianto. Poi vi sono ritorni marginali decrescenti al crescere di x_i. I costi di gestione possono tipicamente avere dei ritorni marginali decrescenti per piccoli valori di x_e, poi ritorni costanti ed infine crescenti dato che grandi impianti tendono ad essere inefficienti. Infine si dovrebbero valutare i costi di gestione su un orizzonte temporale ben definito (con possibili fattori di sconto) e aggiungere questi costi a quelli di costruzione. La funzione che ne risulta può essere molto complessa e priva di quelle proprietà matematiche, quali la convessità, che rendono più agevole la soluzione analitica.

Possiamo infine supporre che i costi di gestione di un impianto non dipendano dagli altri impianti. Se quest'ipotesi non è soddisfatta, il problema diventa molto più difficile e va risolto con algoritmi specifici.

A patto di discretizzare le variabili di decisione la programmazione dinamica è un potente strumento per risolvere problemi di questo genere. Si consideri la seguente istanza: ci sono $n = 6$ possibili siti. I valori di servizio richiesti sono discretizzati e normalizzati in valori interi. Sia $b = 25$ il livello globale di servizio richiesto. I costi di costruzione sono definiti nella tabella 1 (dove un costo infinito significa che l'impianto di quel tipo non può essere costruito), mentre i costi di gestione proiettati sull'orizzonte temporale sono riportati in tabella 2.

È conveniente calcolare gli ottimi anche per valori più elevati di $b = 25$. Siccome i dati in uscita forniscono i costi ottimi per valori qualsiasi di b (si veda in figura 9.7 il costo in funzione dell'offerta), risolvere per valori più alti (e più bassi) permette un'utile analisi di sensibilità sulla decisione finale. Dalla tabella 3 si vede che se la domanda si rivela più grande di 25 ma inferiore a 27 la soluzione è robusta (nel senso che basta ristrutturare gli impianti esistenti ma non costruirne di nuovi). Comunque per una domanda maggiore o uguale a 27 bisogna costruire un nuovo impianto. Si può vedere dalla figura 9.7 che vi è un più marcato aumento del costo ogni qualvolta si verifica un cambio strutturale dovuto alla costruzione di un nuovo impianto. Può essere interessante notare un comportamento 'non monotono' delle variabili di decisione rispetto alla domanda. Decidere a quale livello costruire gli impianti in modo da non incorrere in più elevati costi di espansione in futuro certamente dipende da un'accurata previsione della domanda sull'orizzonte temporale. ∎

9.5. Problemi di knapsack

I problemi di knapsack sono particolari problemi di allocazione (vedi anche esempio 1.72). Tuttavia, data la loro speciale importanza, è opportuno trattarli a parte. Ripetiamo la seguente definizione:

9.30 Definizione. *Si definisce* knapsack 0-1 *il seguente problema: dati interi positivi* $a_1, \ldots, a_n$, $c_1, \ldots, c_n$, b, *si trovi* $\max\{c(J) : a(J) \leq b,\ J \subset \{1, \ldots, n\}\}$. ∎

Tabella 1

	1	2	3	4	5	6	7	8	9	10
1	20	25	28	∞	∞	∞	∞	∞	∞	∞
2	40	45	50	54	57	60	62	63	64	65
3	23	26	29	32	∞	∞	∞	∞	∞	∞
4	35	40	44	48	51	53	∞	∞	∞	∞
5	50	60	70	∞	∞	∞	∞	∞	∞	∞
6	20	24	28	32	35	38	41	43	45	47

Tabella 2

	1	2	3	4	5	6	7	8	9	10
1	40	60	90	∞	∞	∞	∞	∞	∞	∞
2	12	16	20	23	26	30	34	40	47	60
3	25	30	33	35	∞	∞	∞	∞	∞	∞
4	18	22	25	28	32	36	∞	∞	∞	∞
5	10	13	16	∞	∞	∞	∞	∞	∞	∞
6	45	48	51	53	55	57	60	64	70	76

Tabella 3

b	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
$V(b)$	248	285	293	301	315	323	337	374	382	390	404	463	476	490	550
x_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
x_2	10	8	8	9	8	9	10	8	8	9	10	9	9	10	10
x_3	0	4	4	4	0	0	0	4	4	4	4	4	4	4	4
x_4	0	0	0	0	6	6	6	6	6	6	6	6	6	6	6
x_5	0	0	0	0	0	0	0	0	0	0	0	2	3	3	3
x_6	10	9	10	10	10	10	10	9	10	10	10	10	10	10	10

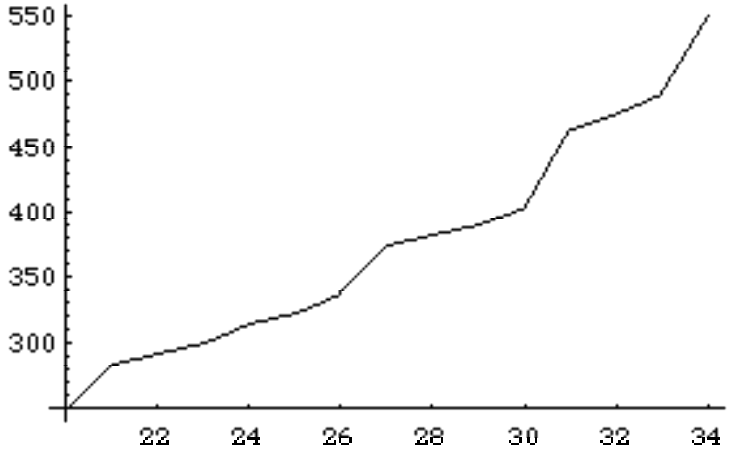


FIGURA 9.7

```
Algoritmo knapsack 0-1
input(c, a, b);
    for j := 0 to b do begin V(j) := 0; J(j) := ∅; end;
    for i := 1 to n do
    begin
        for j := b − a_i downto 0 do
        begin
            if V(j + a_i) < V(j) + c_i
            then begin
                V(j + a_i) := V(j) + c_i;
                J(j + a_i) := J(j) ∪ {i}
            end
        end
    end
output(J(b)).
```

Nella definizione si è posto $c(J) := \sum_{i \in J} c_i$, $a(J) := \sum_{i \in J} a_i \leq b$. Come già osservato, un problema di knapsack 0-1 contiene come caso particolare il problema della partizione (esempio 1.70), quando cioè $a_i = c_i$, $\forall i$, e $b = \sum_i c_i/2$. Quindi si tratta di un problema **NP**-difficile, dato che il problema della partizione è **NP**-completo (esempio 3.51).

Per risolvere un problema di knapsack 0-1 si può costruire il medesimo grafo a livelli usato per problemi generali di allocazione. Quindi ci sono $n+2$ livelli etichettati $0, 1, \ldots, n, n+1$, e $b+1$ nodi nei livelli $1, \ldots, n$, etichettati (k, j) con $k = 0, \ldots, n$, e $j = 0, 1, \ldots b$. Al livello 0 c'è l'unico nodo $s = (0, 0)$ e al livello $n+1$ c'è l'unico nodo t. C'è un arco fra $(k-1, i)$ e (k, j), $1 \leq k \leq n$, con una funzione di costo additiva $V + c_k$, se $j - i = a_k$ oppure con funzione di costo V se $j = i$. In un'effettiva implementazione dell'algoritmo di programmazione dinamica applicato al knapsack 0-1 non c'è ovviamente bisogno di costruire il grafo. Tale processo viene simulato implicitamente. Dato che il grafo ha due archi uscenti da ogni nodo e il numero di nodi è $O(n\,b)$ abbiamo dimostrato che:

9.31 Teorema. *L'algoritmo knapsack 0-1 K trova l'ottimo in tempo $O(n\,b)$.* ∎

Potrebbe quindi sembrare che si sia trovato un algoritmo polinomiale per un problema **NP**-completo. Tuttavia, come si è visto nel capitolo 3, la complessità di un algoritmo si misura rispetto alle dimensioni dell'istanza e, con la codifica normale, b è esponenziale rispetto alla sua codifica. Quindi, a ben vedere, $O(n\,b)$ è una complessità di tipo esponenziale. Adottando la codifica unaria l'algoritmo diventerebbe polinomiale e quindi si tratta di un algoritmo pseudopolinomiale (definizione 3.8). Questa comporta a sua volta che il problema di knapsack è debolmente **NP**-difficile (definizione 3.40).

Si possono scambiare i ruoli dei coefficienti c e a nell'algoritmo appena visto. In altre parole si può costruire un grafo con $n+2$ livelli e C nodi in ogni livello dove $C := \sum_k c_k$, e vi sono archi $((k-1, i), (k, i + c_k))$ di costo a_k e archi $((k-1, i), (k, i))$ di costo 0. In questo caso la funzione $V(k, j)$ rappresenta il peso ottimo di sottoinsiemi di valore uguale a j fino al livello k. Ogni qualvolta due sottoinsiemi abbiano lo stesso valore j al livello k, ovvero corrispondono a due cammini che terminano al livello k nello stesso nodo, viene selezionato quello con un valore migliore di peso, cioè quello più leggero, perché ci sono più possibilità di aggiungere elementi mantenendo l'ammissibilità. Questa discussione giustifica quindi l'algoritmo knapsack 0-1 C e il seguente teorema:

9.32 Teorema. *L'algoritmo knapsack 0-1 C trova l'ottimo in tempo $O(n\,C)$.* ∎

Algoritmo knapsack 0-1 C
input(a, c, b);
    $V(0) := 0$; $J(0) := \emptyset$;
    for $j := 1$ **to** C **do begin** $V(j) := +\infty$; $J(j) := \emptyset$; **end**;
    for $i := 1$ **to** n **do**
    begin
        for $j := C - c_i$ **downto** 0 **do**
        begin
            if $V(j + c_i) > V(j) + a_i$
            then begin
                $V(j + c_i) := V(j) + a_i$;
                $J(j + c_i) := J(j) \cup \{i\}$;
            end;
        end;
    end;
    sia $h : V(h) \leq b$ e $V(j) > b, \forall j > h$;
output$(J(h))$.

Algoritmo knapsack intero
input(a, c, b);
    for $k := 0$ **to** b **do begin** $V(k) := 0$; $p(k) := 0$; $q(k) := 0$ **end**;
    for $i := 1$ **to** n **do** $x_i := 0$;
    for $k := 0$ **to** $b - 1$ **do**
    begin
        if $V(b) < V(k)$
            then begin
                $V(b) := V(k)$; $p(b) := k$; $q(b) := 0$;
            end;
        for $i := 1$ **to** n **do**
        begin
            if $(k + a_i \leq b) \ \wedge \ (V(k + a_i) < V(k) + c_i)$
            then begin
                $V(k + a_i) := V(k) + c_i$; $p(k + a_i) := k$; $q(k + a_i) := i$;
            end
        end
    end;
    if $q(b) = 0$ **then** $k := p(b)$ **else** $k := b$;
    repeat
        $i := q(k)$; $x_i := x_i + 1$; $k := p(k)$;
    until $k = 0$;
output(x);

Si consideri ora un'altra versione del problema di knapsack:

9.33 Definizione. *Si definisce* knapsack intero *il seguente problema: dati interi positivi* $a_1, \ldots, a_n$, $c_1, \ldots, c_n$, b, *si trovi* $\max\{\sum_{i=1}^n c_i\, x_i : \sum_{i=1}^n a_i\, x_i \leq b,\ x_i \in \mathbb{Z}_+,\ \forall i\}$. ∎

In questo caso basta costruire una rete molto semplice con $b+1$ nodi etichettati $0, \ldots, b$ ed archi $(k, k+a_i)$ di costo c_i per ogni i tale che $k + a_i \leq b$ e per ogni $k := 0, \ldots, b-1$. A questi archi si aggiungono archi (k, b) di costo 0 per ogni k. Quindi ogni cammino da 0 a b corrisponde ad una scelta di valori x_i, in quanto se il cammino usa l'arco di costo c_i questo equivale ad incrementare x_i di uno (a partire ovviamente da una situazione iniziale in cui sono tutti nulli). L'uso di un arco (k, b) corrisponde alla scelta di terminare senza incrementare le variabili. In questo grafo gli archi incidenti in ogni nodo sono $O(n)$. Quindi possiamo concludere:

9.34 Teorema. *L'algoritmo knapsack intero trova l'ottimo in tempo* $O(n\, b)$. ∎

9.35 Esercizio. Un'altra versione ancora del problema di knapsack è data dal cosiddetto *knapsack a scelta multipla*: sono dati m insiemi $J_1, J_2, \ldots J_m$; ad ogni elemento $i \in J_j$ sono associati due numeri positivi c_{ij} e a_{ij}. Vogliamo selezionare al più un elemento da ogni insieme in modo che la somma dei costi degli elementi selezionati sia massima e la somma degli stessi sia non superiore ad una limitazione b. Si progetti un algoritmo di programmazione dinamica simile a quello per il knapsack 0-1, di complessità $O(nb)$ con $n = \sum_j |J_j|$. ∎

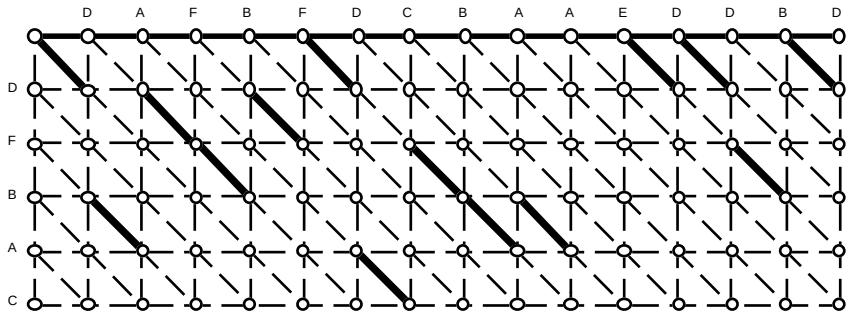
Per uno studio approfondito dei problemi di knapsack e dei loro vari metodi di risoluzioni si veda la monografia Martello e Toth [1990].

9.6. Problemi combinatori

Vi sono diversi casi in cui problemi combinatori di ottimizzazione possono essere risolti sfruttando un'equazione ricorsiva quale quella di Bellman. In un tipico modello combinatorio risolvibile con la programmazione dinamica è data una struttura combinatoria dipendente da uno o più parametri $n, m, \ldots$, e il calcolo del valore ottimo della struttura per $n+1, m+1 \ldots$, si può ricavare facilmente dal valore ottimo per $n, m, \ldots$. Assegnati i valori iniziali della ricorsione, si esegue il calcolo esplicito della ricorsione. Tuttavia tale calcolo si può effettivamente eseguire solo se lo spazio degli stati sottinteso dalla ricorsione è limitato polinomialmente rispetto ai dati d'ingresso. Purtroppo non sono molte le applicazioni in cui questa circostanza si verifica. Qui di seguito forniamo alcuni esempi fra quelli disponibili in letteratura.

9.36 Esempio. Confronto di stringhe
Sono dati un testo $t = t_1 t_2 \ldots t_n$ ed una stringa $s = s_1 s_2 \ldots s_m$ rispettivamente di n e m simboli da un alfabeto finito. Tipicamente si ha $n >> m$. Si vuole trovare un pezzo di testo che sia uguale alla stringa. La situazione ideale sarebbe che esistesse veramente un tale sottotesto, ma in molti casi avviene che non esiste e quindi dobbiamo accontentarci di un sottotesto 'quasi' uguale alla stringa.

Questa è una situazione tipica ad esempio in biologia molecolare, dove il testo è un lungo frammento di DNA, con i simboli dell'alfabeto di 4 lettere A (adenina), G (guanina), T (timina), C (citosina), e la stringa è un frammento corto di DNA che si suppone provenire



$V(i,0)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$V(i,1)$	1	0	1	1	1	1	0	1	1	1	1	1	0	0	1	0
$V(i,2)$	2	1	1	1	2	1	1	1	2	2	2	2	1	1	1	1
$V(i,3)$	3	2	2	2	1	2	2	2	1	2	3	3	2	2	1	2
$V(i,4)$	4	3	2	3	2	2	3	3	2	1	2	3	3	3	2	2
$V(i,5)$	5	4	3	3	3	3	3	3	3	2	2	3	4	4	3	3

FIGURA 9.8

dal frammento lungo, tramite però una serie di modificazioni che ne rendono impossibile l'esatta coincidenza.

Per modellare formalmente il problema diremo che due stringhe a e b sono alla distanza d se sono necessarie d operazioni per rendere uguali a e b. Un'operazione può essere: cancellazione di un simbolo da a, cancellazione di un simbolo da b, sostituzione di un simbolo con un altro in a. Allora il problema consiste nel trovare un pezzo di testo che minimizza la distanza con la stringa.

Si costruisca allora un grafo con $(n+1)\,(m+1)$ nodi. Indichiamo con s^j la stringa troncata alla posizione j e con t^i il testo troncato alla posizione i. Un nodo viene etichettato con la coppia di indici (i,j) e rappresenta una situazione in cui è stata ottenuta una esatta coincidenza fra una stringa a ottenuta da s^j tramite una serie di operazioni (eventualmente nessuna) e gli ultimi simboli di una stringa b ottenuta da t^i tramite una serie di operazioni (eventualmente nessuna).

Ci sono tre tipi di archi. Uno spostamento da (i,j) a $(i,j+1)$ significa che si sta cancellando il $(j+1)$-esimo simbolo dalla stringa per poter mantenere l'esatta coincidenza. Similmente uno spostamento da (i,j) a $(i+1,j)$ significa che si sta cancellando il $(i+1)$-esimo simbolo dal testo. Il costo di questi archi è 1 perché è richiesta un'operazione di cancellazione. Uno spostamento da (i,j) a $(i+1,j+1)$ significa che si sta cercando di aggiungere alle due stringhe a e b i simboli $(i+1)$-esimo e $(j+1)$-esimo rispettivamente. Il costo di questo arco può essere 0 se i due simboli sono uguali oppure 1 se non sono uguali e quindi viene richiesta una sostituzione.

Un cammino da un qualsiasi nodo $(i,0)$ ad un altro nodo (i',m), $i' > i$ corrisponde ad una sequenza di operazioni o sulla sottostringa del testo dalla posizione $i+1$ a i' o sulla stringa s tali da far coincidere le risultanti sottostringhe.

Sia $V(i,j)$ il costo ottimo per ottenere una coincidenza fa gli ultimi simboli di t^i e s^j. Allora si ha la seguente equazione di Bellman:

$$V(i+1,j+1) := \min\{1+V(i+1,j)\ ;\ 1+V(i,j+1)\ ;\ c_{ij}+V(i,j)\}$$

dove

$$c_{ij} := \begin{cases} 0 & \text{if } t_i = s_j \\ 1 & \text{altrimenti} \end{cases}$$

Si abbia $t = DAFBFDCBAAEDDBD$ e $s = DFBAC$. L'istanza è allora rappresentata dal grafo in figura 9.8 dove gli archi tratteggiati hanno costo 1 e gli altri costo 0. Vi sono due cammini ottimi di lunghezza 2 che partono ambedue dalla seconda F del livello superiore e che arrivano alla seconda e terza A del livello inferiore. Quindi ci sono due sottostringhe del testo per cui la distanza dalla stringa è 2, e cioè $DCBAA$ (due sostituzioni), $DCBA$ (una sostituzione e una cancellazione dalla stringa). ∎

9.37 Esempio. Suddivisione ordinata

L'esempio verrà illustrato in un caso particolare. Nel caso generale si tratta di dividere un insieme ordinato di oggetti in un numero variabile di sottoinsiemi mantenendo invariato l'ordine all'interno di questi e minimizzando una opportuna funzione della suddivisione.

Si supponga di dover decidere dove disporre le fermate di una linea di autobus. Sulla linea ci sono n isolati e ad ogni isolato c'è una fermata potenziale. Però è possibile avere solo $m < n$ fermate (capolinea inclusi). Per decidere quali fermate effettivamente attivare si valuta il disagio che deriva ai residenti da una assegnata disposizione delle fermate e si cerca quella di minimo disagio.

Supponiamo di aver valutato che nell'isolato i vi sono p_i residenti che prendono l'autobus. Se vi è una fermata in un isolato, i residenti dello stesso isolato non hanno bisogno di camminare, altrimenti devono camminare fino alla fermata più vicina alla sinistra o alla destra del loro isolato. Supponiamo per semplicità che la lunghezza di ogni isolato sia uno. Allora se le fermate più vicine all'isolato i si trovano agli isolati h e k con $h < i < k$, la distanza percorsa dai residenti dell'isolato i è $p_i \min\{i-h; k-i\}$. Si vuole minimizzare la distanza totale percorsa da tutti i residenti.

Per modellare il problema con la programmazione dinamica supponiamo di conoscere il valore ottimo quando vi siano i fermate (incluse quelle terminali), i capolinea sono agli isolati 1 (sempre fisso) e j e solo gli abitanti dall'isolato 1 all'isolato j vengono serviti. Si indichi con $V(i,j)$ tale costo ottimo. Se si vuole aumentare di uno il numero di fermate e vogliamo quindi calcolare $V(i+1,j)$ possiamo ragionare come segue: la linea da 1 a j può essere vista come formata da due parti, la prima con i fermate fino ad un isolato k, $i \le k < j$ e poi il tratto da k a j senza fermate. Gli abitanti fra gli isolati k e j si serviranno delle fermate k e j ad un costo globale

$$c_{kj} = \sum_{h=k+1}^{j-1} p_h \min\{h-k; j-h\}$$

Gli abitanti fra l'isolato 1 e quello k si serviranno delle i fermate fra 1 e k come se si trattasse di una linea con capolinea in k, e quindi con valore ottimo $V(i,k)$. Il modo ottimo di aggiungere una fermata è di calcolare il disagio globale (fino all'isolato j) per tutti i valori di k, $i \le k < j$, e poi prendere il migliore. Quindi l'equazione ricorsiva è:

$$V(i+1,j) = \min_{i \le k \le j-1} V(i,k) + c_{kj}$$

con condizione iniziale $V(2,j) := c_{1,j}$ per ogni j.
Si consideri il seguente esempio con 21 isolati e 6 fermate:

$$p = \{4,6,3,2,6,4,8,8,2,1,4,3,7,7,2,2,4,5,8,2,9\}$$

Allora i valori ottimi $V(i,j)$ sono:

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
2	0	6	9	14	22	34	52	78	100	123	146	172	197	229	255	283	313	348	390	434
3	0	0	3	5	13	18	26	38	44	47	52	60	74	95	114	135	157	177	205	228
4	0	0	0	2	5	9	17	26	28	29	34	41	55	66	69	73	79	90	107	125
5	0	0	0	0	2	5	9	17	19	20	25	32	39	48	50	54	60	71	86	89
6	0	0	0	0	0	2	5	9	11	12	17	23	30	39	41	43	49	58	67	70

con puntatori all'indietro

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	2	2	2	2	5	5	6	7	7	7	7	7	7	7	7	7	8	8	8
4	1	2	3	3	5	5	5	7	7	8	8	8	8	12	13	13	13	13	13	14
5	1	1	3	4	5	6	7	7	7	8	8	8	11	12	13	13	13	13	17	18
6	1	1	1	4	5	6	7	8	8	8	8	11	11	12	13	14	14	14	17	18

da cui si ricava che le fermate devono essere situate negli isolati $(1, 5, 8, 13, 18, 21)$. ∎

9.38 Esempio. Schedulazione di una macchina

In questo problema vi sono n lavori che devono essere eseguiti su una macchina. La macchina può eseguire solo un lavoro alla volta, ed una volta iniziata l'esecuzione di un lavoro non è ammessa l'interruzione. I lavori sono contraddistinti da una durata $p_i > 0$ e da una funzione di penalità $f_i(C_i)$ che dipende dal tempo di completamento C_i del lavoro. Si assume che le funzioni f_i siano monotone. La penalità globale viene calcolata come $\max_i f_i(C_i)$. Una schedulazione viene definita a partire da una permutazione dei lavori. Essendo le funzioni f_i monotone non c'è nessun vantaggio nel ritardare l'esecuzione dei lavori, per cui i lavori vengono eseguiti uno di seguito all'altro a partire dall'istante zero e la permutazione definisce immediatamente i tempi di completamento di ogni lavoro. Il problema consiste allora nel trovare la permutazione che minimizza la penalità globale $\max_i f_i(C_i)$.

Usando la programmazione dinamica si può definire come $V(S)$ il costo di una schedulazione ottima se i lavori da schedulare sono solo quelli dell'insieme S. Ciò che dobbiamo calcolare è pertanto $V(N)$ con $N = \{1, \ldots, n\}$. Si indichi $T(S) := \sum_{i \in S} p_i$ il tempo di completamento dell'ultimo dei lavori in S. Si noti che questo valore è invariante rispetto alla permutazione con cui vengono schedulati i lavori in S. Pertanto vale la seguente equazione ricorsiva

$$V(S) = \min_{i \in S} \max \{V(S \setminus \{i\}), f_i(T(S))\} \tag{9.7}$$

in cui si esprime semplicemente il fatto che la schedulazione ottima di S si ottiene 'provando' a sistemare in ultima posizione a turno ogni lavoro di S e schedulando in ottimalità gli altri lavori e poi prendendo la migliore di queste possibilità. Il problema che sorge nell'usare la ricorsione (9.7) è che il numero di stati è esponenziale. Tuttavia alcune semplici osservazioni permettono di ridurre il numero di stati utili al calcolo dell'ottimo a solo n. Innanzitutto vale

$$V(S) \geq \min_{i \in S} f(T(S)) \tag{9.8}$$

in quanto il costo di ogni permutazione è maggiore o uguale alla penalità dell'ultimo lavoro schedulato e quindi anche il valore ottimo deve essere maggiore o uguale al più piccolo di questi valori. La seconda osservazione sfrutta la monotonia delle funzioni di penalità per cui

$$V(S) \geq V(S \setminus \{i\}) \quad \forall i \tag{9.9}$$

Ora si consideri (9.7) quando $S = N$ e quindi $T(S) = T(N)$ è noto, e sia k tale che $f_k(T(N)) = \min_{i \in N} f_i(T(N))$

$$V(N) = \min_{i \in N} \max \{V(N \setminus \{i\}), f_i(T(N))\} \leq \max \{V(N \setminus \{k\}), f_k(T(N))\}$$

e da (9.8) e (9.9) si ha

$$V(N) \geq \max \{V(N \setminus \{k\}), f_k(T(N))\}$$

e quindi

$$V(N) = \max \{V(N \setminus \{k\}), f_k(T(N))\}$$

Questa uguaglianza implica che fra le schedulazioni ottime ce n'è una con il lavoro k schedulato per ultimo e quindi l'unico insieme di $n-1$ elementi da considerare è $N \setminus \{k\}$. A questo punto si procede ricorsivamente. Solo quando si sia trovata la schedulazione ottima si può calcolare il valore ottimo. La complessità computazionale è $O(n^2)$ dovuta al fatto che bisogna calcolare n volte il minimo di n valori (nell'ipotesi di costo costante per le valutazioni delle funzioni f_i).

Siano dati 5 lavori con

$$p = (4, 6, 11, 8, 7)$$

$$f_1(C) := \begin{cases} 0 & C \leq 10 \\ +\infty & C > 10 \end{cases} \qquad f_2(C) := \begin{cases} C & C \leq 8 \\ 4C - 24 & C \geq 8 \end{cases} \qquad f_3(C) := 2C$$

$$f_4(C) := \begin{cases} 0 & C \leq 10 \\ 30 & 10 < C \leq 20 \\ 80 & C > 20 \end{cases} \qquad f_5(C) := \begin{cases} 3C & C \leq 30 \\ +\infty & C > 30 \end{cases}$$

$T(N) = 36$ e $f_1(36) = \infty$, $f_2(36) = 120$, $f_3(36) = 72$, $f_4(36) = 80$, $f_5(36) = \infty$ e quindi l'ultimo lavoro è il lavoro 3. Si passa ora a considerare $S = \{1, 2, 4, 5\}$ e $T(S) = 25$. Allora $f_1(25) = \infty$, $f_2(25) = 76$, $f_4(25) = 80$, $f_5(25) = 75$ e si schedula per ultimo il lavoro 5. Ora $S = \{1, 2, 4\}$ e $T(S) = 18$. Allora $f_1(18) = \infty$, $f_2(18) = 48$, $f_4(18) = 30$ e si schedula per ultimo il lavoro 4. Ora $S = \{1, 2\}$ e $T(S) = 10$. Allora $f_1(10) = 0$, $f_2(10) = 16$ e si schedula per ultimo il lavoro 1. Infine $f_2(6) = 6$. A questo punto si può calcolare $V(N) = \max\{6; 0; 30; 75; 72\} = 75$ e il lavoro più critico risulta essere il lavoro 5. ∎

9.39 Esempio. TSP

Fra i vari approcci al TSP, ce n'è uno che fa uso della programmazione dinamica. In generale non è molto raccomandabile computazionalmente. Tuttavia in alcune circostanze lo spazio degli stati può essere ridotto ad una dimensionale maneggevole, soprattutto se sono presenti altri vincoli, quali precedenze e finestre temporali,

Un nodo del grafo di programmazione dinamica è identificato dall'etichetta (S, j) dove S è un sottoinsieme di nodi non contenente il nodo 1 e contente il nodo j e corrisponde ad un cammino elementare che parte dal nodo 1, passa attraverso tutti i nodi di S terminando in j. Se $V(S, j)$ è il costo ottimo di un tale cammino vale la seguente equazione ricorsiva:

$$V((S, j)) := \min_{i \in S \setminus \{j\}} (V(S \setminus \{j\}, i) + c_{ij})$$

Il numero di archi di questa rete è $2(n-1) + \sum_{k=1}^{n-2} k(n-k-1)\binom{n-1}{k}$, il che porta ad una complessità $O(n^2 2^n)$ che non è praticamente accettabile, se non si fa ricorso ad accorgimenti opportuni. Si veda in figura 9.9 il tipo di grafo che viene creato con $n = 5$. I numeri nei nodi indicano gli insiemi S e l'ultimo numero indica il nodo j, ultimo visitato dal cammino hamiltoniano. ∎

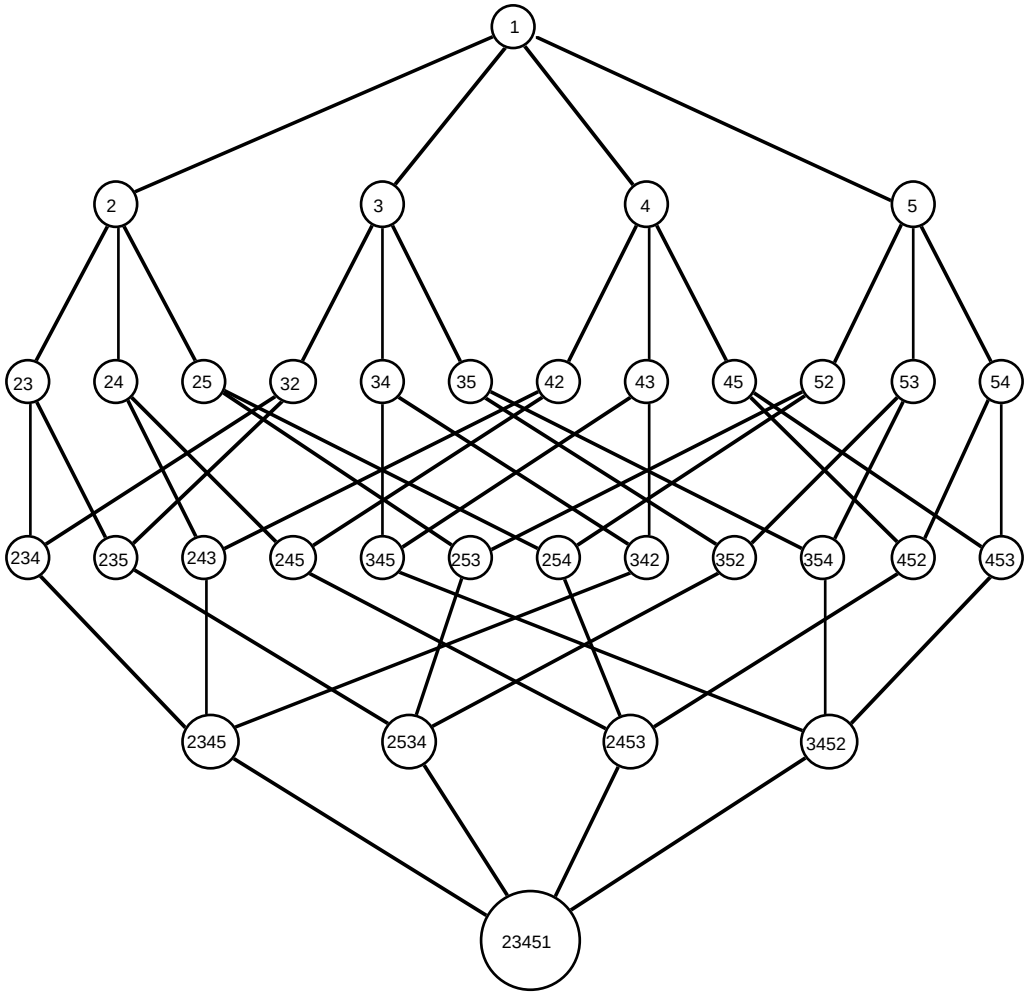


FIGURA 9.9

9.40 ESERCIZIO (da Papadimitriou e Steiglitz [1982] p. 451). Sia da eseguire il prodotto di n matrici rettangolari $A^1, A^2, \ldots, A^n$. Sia a_i il numero di righe della matrice i-esima e il numero di colonne della matrice $i-1$-esima. Il costo computazionale del prodotto delle due matrici $A^{i-1}\,A^i$ è $a_{i-1}\,a_i\,a_{i+1}$. Valendo la proprietà associativa del prodotto si può eseguire il calcolo in molti modi alternativi di complessità computazionale diversa, in generale. Si trovi il modo migliore di eseguire il prodotto usando una formulazione di programmazione dinamica. ∎

9.41 ESERCIZIO. Si modelli con la programamzione dinamica il problema di dividere un paragrafo in un numero variabile di linee in modo che la divisione del testo sia la migliore possibile. Vari tipi di funzione obiettivo possono venir formulati. Si modelli il problema secondo l'approccio della suddivisone ordinata. ∎

9.7. Controllo ottimo

I principi della programmazione dinamica si possono applicare anche a grafi virtuali con un numero infinito di nodi e di archi. Vi sono infatti molti problemi in cui i cammini sono curve qualsiasi di $\mathbb{R}^n$ e quindi il grafo associato a questi problemi ha come nodi tutti i punti di $\mathbb{R}^n$ e come archi tutti i vettori infinitesimi dx. Naturalmente non è possibile in questi casi risolvere un problema in modo algoritmico come nel caso di nodi finiti e bisogna invece ricorrere ad una formulazione analitica. Non ci occupiamo in questa sede di questi particolari problemi di programmazione dinamica che richiederebbero un tipo di approccio che esula dall'ambito della programmazione matematica.

Tuttavia vi sono alcuni problemi dinamici più semplici a causa della discretizzazione che viene operata sul tempo. Questo corrisponde ad avere un grafo a livelli, associati agli istanti di tempo, in cui i nodi di ogni livello sono i punti di $\mathbb{R}^n$ e gli archi fra livelli successivi sono tutte le coppie $\mathbb{R}^n \times \mathbb{R}^n$. La risoluzione di problemi di questo genere è possibile se si riesce a pervenire ad un'espressione analitica per quel che riguarda gli stati, mentre per il tempo, dato che è stato discretizzato, si può procedere per via algoritmica.

Una tipica applicazione di questo genere è costituita dal controllo ottimo di un sistema lineare. Un sistema dinamico lineare discreto è rappresentato dalla seguente equazione di evoluzione:

$$x^{k+1} = A\,x^k + B\,u^{k+1} \tag{9.10}$$

dove $x^k \in \mathbb{R}^n$ è lo stato al tempo k e $u^k \in \mathbb{R}^m$ è il controllo (oppure una forza esterna) applicata al tempo k. La matrice A rappresenta l'evoluzione libera del sistema in assenza di forze esterne applicate e la matrice B rappresenta l'interazione fra lo stato e le forze esterne.

Il problema del controllo consiste nel trovare una sequenza u^k tale che, a partire da uno stato iniziale fissato x^0, si riescano a ottenere delle condizioni prescritte sugli stati e/o sui controlli per un numero finito oppure infinito di istanti temporali.

Consideriamo solo un caso particolare, il cosiddetto controllo quadratico. Sia N il tempo finale. Siano P^k delle matrici $n \times n$ e Q^k delle matrici $m \times m$ definite per $k = 1, \dots N$. Si vuole trovare un controllo u^k, $k = 1, \dots, N$, tale che:

$$\sum_{k=1}^{N} x^k\,P^k\,x^k + u^k\,Q^k\,u^k$$

sia minimo. Chiaramente ha senso considerare solo matrici P^k e Q^k simmetriche. Inoltre si considerano normalmente soltanto matrici semidefinite positive con l'ulteriore requisito che P^N e Q^k, $k = 1, \dots N$ siano positive definite.

Si noti che in assenza di forze esterne l'origine è un punto d'equilibrio. Tale equilibrio può essere stabile, indifferente o instabile a seconda del raggio spettrale $\rho(A) < 1$, $\rho(A) = 1$ e $\rho(A) > 1$ rispettivamente. Quindi la parte della funzione obiettivo relativa allo stato realizza lo scopo di mantenere lo stato il più possibile in equilibrio all'origine. Tuttavia, siccome controllare ha un costo (in molte applicazioni pratiche d'interesse vi è una limitazione superiore alla norma del controllo), bisogna minimizzare anche il controllo e questo viene realizzato nella parte della funzione obiettivo relativa al controllo. La matrice P^N riveste un ruolo particolare in quanto rappresenta il costo dello stato finale. Se si vuole raggiungere l'origine al tempo N, questo si può ottenere con un valore molto grande di P^N.

La programmazione dinamica permette di trovare il controllo ottimo in modo semplice ed elegante. Il problema viene formulato all'indietro in quanto permette di ottenere un controllo sotto forma di controreazione sullo stato. In altri termini si vogliono trovare matrici R^k tali che il controllo ottimo si esprima come $u^{k+1} := R^k\,x^k$ e quindi il controllo venga direttamente

calcolato a partire dal valore corrente dello stato. Questo tipo di controllo viene chiamato a *controreazione* oppure *in catena chiusa*. Controllo in catena aperta significa che la sequenza u^k è precalcolata e poi applicata senza necessariamente valutare gli stati correnti.

Mentre in un modello astratto e puramente deterministico non c'è differenza fra controllo in catena chiusa o aperta, in pratica è di gran lunga più raccomandabile il controllo in catena cbiusa perché tiene automaticamente conto di possibili disturbi ed è autocorrettivo.

Sia $U^k(x)$ il costo ottimo a partire dal tempo k in poi, essendo il sistema nello stato x. Stabiliamo inoltre che $U^k(x)$ includa il costo dello stato al tempo k ma non quello del controllo applicato al tempo k. Allora $U^N(x)$ dipende solo dallo stato finale siccome non viene applicato alcun controllo dopo il tempo N. Quindi $U^N(x) = x^T\, P^N\, x$. Ora vogliamo far vedere che $U^k(x)$ ha la seguente forma

$$U^k(x) = x^T\, S^k\, x$$

per una matrice S^k positiva semidefinita. L'affermazione verrà provata per induzione. È certamente vera per $k = N$, con $S^N = P^N$. Quindi dobbiamo dimostrare che se è vera al tempo k è vera anche al tempo $k-1$. Questo verrà fatto sfruttando l'equazione di Bellman all'indietro (9.6).

$$\begin{aligned} U^{k-1}(x) = \min_u x^T\, P^{k-1}\, x + u^T\, Q^k\, u + U^k(Ax + Bu) = \qquad (9.11)\\ \min_u x^T\, P^{k-1}\, x + u^T\, Q^k\, u + (A\, x + B\, u)^T\, S^k\, (A\, x + B\, u) = \\ \min_u u^T\, (Q^k + B^T\, S^k\, B)\, u + 2\, x^T\, A^T\, S^k\, B\, u + x^T\, (P^{k-1} + A^T\, S^k\, A)\, x \end{aligned}$$

Il minimo si ottiene derivando rispetto ad u e eguagliando a zero, cioè:

$$2\, (Q^k + B^T\, S^k\, B)\, u + 2\, B^T\, S^k\, A\, x = 0.$$

Poiché Q^k è positiva definita anche $(Q^k + B^T\, S^k\, B)$ è positiva definita e può essere invertita. Allora

$$u = -(Q^k + B^T\, S^k\, B)^{-1}\, B^T\, S^k\, A\, x$$

e

$$R := -(Q^k + B^T\, S^k\, B)^{-1}\, B^T\, S^k\, A$$

è la matrice di controreazione. Sostituendo in (9.11) si ottiene

$$\begin{aligned} U^{k-1}(x) = x^T\, P^{k-1}\, x + x^T\, R^T\, Q^k\, R\, x + (A\, x + B\, R\, x)^T\, S^k\, (A\, x + B\, R\, x) = \\ = x^T\, (P^{k-1} + R^T\, Q^k\, R + (A + B\, R)^T\, S^k\, (A + B\, R))\, x \end{aligned}$$

e quindi

$$S^{k-1} := P^{k-1} + R^T\, Q^k\, R + (A + B\, R)^T\, S^k\, (A + B\, R)$$

e l'induzione è provata. Dal punto di vista computazionale si tratta di eseguire il seguente schema iterativo: si inizializza $S := P^N$, poi si calcola

$$R^k := -(Q^k + B^T\, S\, B)^{-1}\, B^T\, S\, A \qquad (9.12)$$

e

$$S := P^{k-1} + R^{kT}\, Q^k\, R^k + (A + B\, R^k)^T\, S(A + B\, R^k) \qquad (9.13)$$

per $k = N, N-1, \ldots, 1$. Alla fine si applica il controllo in controreazione

$$u^k = R^k\, x^{k-1} \qquad \text{per } k = 1, 2, \ldots N$$

9.42 Esempio. Consideriamo le seguenti matrici

$$A = \begin{bmatrix} 1.1 & 0.2 \\ -0.2 & 1.1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Il sistema è instabile. In figura 9.10 si vede l'evoluzione libera a partire da $x^0 = \{1, 1\}$. Vogliamo guidare il sistema nell'origine entro 3 passi. A questo scopo applichiamo un peso altissimo allo stato finale, mentre gli stati intermedi ricevono peso nullo e il controllo riceve un peso piccolo. Quindi fissiamo

$$P^N := \begin{bmatrix} 1000 & 0 \\ 0 & 1000 \end{bmatrix}; \qquad P^k := \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad k < N; \qquad Q^k := [1] \quad k \leq N$$

Il controllo risultante forza lo stato alla traiettoria in figura 9.11.

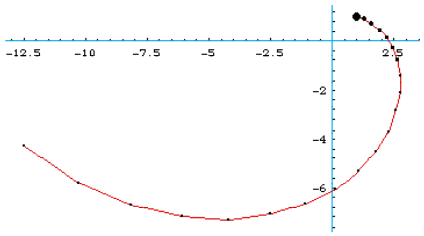


Figura 9.10

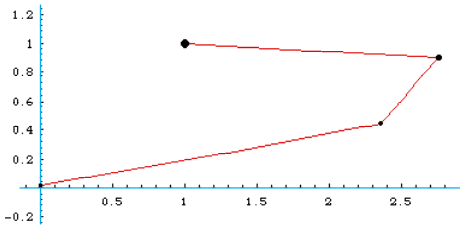


Figura 9.11

Se vogliamo una traiettoria che raggiunga l'origine in modo più uniforme senza picchi intermedi nei valori di stato, allora dobbiamo assegnare del peso anche agli stati intermedi. Quindi fissiamo

$$P^N := \begin{bmatrix} 1000 & 0 \\ 0 & 1000 \end{bmatrix}; \qquad P^k := \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix} \quad k < N; \qquad Q^k := [10] \quad k \leq N \quad (9.14)$$

e abbiamo il diagramma di figura 9.12

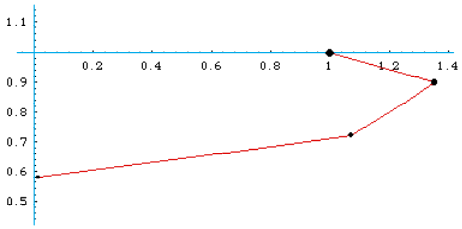


Figura 9.12

L'approccio all'origine è ora più uniforme, ma l'origine non viene raggiunta entro 3 passi. Certamente dobbiamo aumentare il numero di passi se vogliamo raggiungere dolcemente l'origine. Portando N a 20 con gli stessi valori di (9.14) si ha la traiettoria di figura 9.13, i cui valori u^k sono raffigurati in figura 9.14. ∎

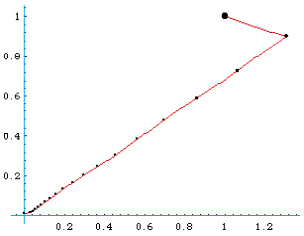


FIGURA 9.13

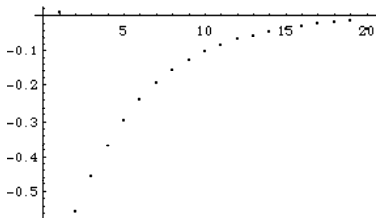


FIGURA 9.14

9.43 ESEMPIO. Come secondo esempio si consideri il problema di spostare una massa m lungo una linea retta da una posizione di riposo ad un'altra posizione di riposo. Possiamo pensare che la massa sia un'automobile da muovere da un posto di parcheggio ad un altro su una strada rettilinea. Modelliamo inizialmente il problema usando il tempo continuo e poi lo approssimeremo discretizzando il tempo.

Sono necessarie due variabili di stato per rappresentare la posizione e la velocità, che indicheremo con x_1 e x_2. Il controllo è una forza che agisce sulla velocità. Quindi si ha $dx_1(t)/dt = x_2(t)$ e $dx_2(t)/dt = K\,u(t)$ con $K = 1/m$. Il modello lineare continuo è allora

$$\frac{d}{dt}\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ K \end{bmatrix} u(t) \tag{9.15}$$

Il modello (9.15) può essere approssimato nel seguente modo:

$$\frac{1}{\Delta t}\left(x(t+\Delta t) - x(t)\right) = A\,x(t) + B\,u(t)$$

dove Δt è un intervallo di tempo fissato. Allora si ha

$$x(t+\Delta t) = (I + \Delta t\,A)\,x(t) + \Delta t\,B\,u(t) \tag{9.16}$$

da dove si vede che (9.16) è l'approssimazione di primo ordine dell'espansione in serie di potenze dell'esponenziale di $\Delta t\,A$. Usando l'indice k per lo stato al tempo $t + k\,\Delta t$ e l'indice $k+1$ per il controllo al tempo $t + k\,\Delta t$ possiamo scrivere

$$x^{k+1} = (I + \Delta t\,A)\,x^k + \Delta t\,B\,u^{k+1}$$

ovvero per il problema dell'esempio

$$\begin{bmatrix} x_1^{k+1} \\ x_2^{k+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x_1^k \\ x_2^k \end{bmatrix} + \begin{bmatrix} 0 \\ \Delta t\,K \end{bmatrix} u^{k+1}$$

Per risolvere numericamente l'esempio si supponga che la massa sia un'automobile da 1 t da spostare di 20 m in 5 s. Per discretizzare il tempo possiamo prendere $\Delta t = 0.1$ s. Usando le unità di misura MKS abbiamo $K := 10^{-3}$ e $x^0 := \{-10, 0\}$. Siamo interessati soltanto a raggiungere lo stato finale di riposo in esattamente 5 s. Quindi assegniamo peso soltanto allo stato finale. Q deve essere non zero per ragioni numeriche. Così abbiamo

$$Q := [\,10^{-5}\,]\,; \qquad P^k := \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad k < N; \qquad P^N := \begin{bmatrix} 10^6 & 0 \\ 0 & 10^6 \end{bmatrix}$$

La traiettoria di stato risultante si vede in figura 9.15 e il controllo in figura 9.16. Come si può vedere si raggiunge una velocità massima di circa 3 m/s (10.8 km/h) a metà strada e viene richiesta una forza massima (in valore assoluto) alla partenza e alla fermata. Questa forza è di circa $2,2 \cdot 10^3$ N che risulta essere circa un quarto di quella esercitata sull'automobile dalla gravità ($9,8 \cdot 10^3$ N). ∎

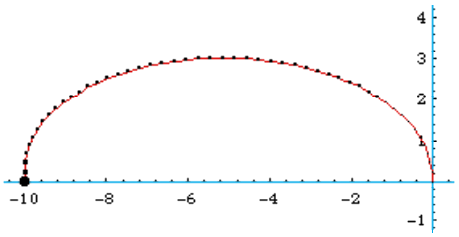


FIGURA 9.15

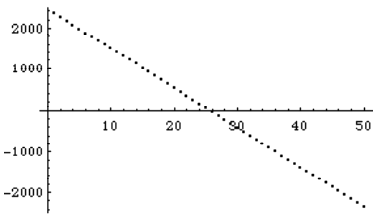
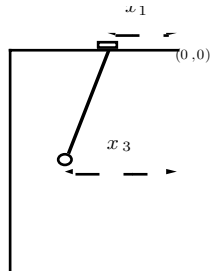


FIGURA 9.16

9.44 ESEMPIO. Questo esempio è simile al precedente ma più complesso. Si supponga di voler controllare il movimento di una gru. Più esattamente si vuole spostare un carico appeso alla gru da una posizione ad un'altra in un intervallo di tempo fissato in modo che al tempo finale il carico non oscilli più. Si vuole inoltre che le oscillazioni durante la manovra siano minime. Si veda in figura una rappresentazione schematica della gru.



I valori esatti dell'accelerazione orizzontale dell'estremo superiore del cavo (a_1) e delle accelerazioni orizzontale (a_2) e verticale (a_3) dell'estremo inferiore del cavo sono dati dalle seguenti formule (valide se il cavo è sotto tensione come ragionevolmente supponiamo), dove α è l'angolo fra la verticale ed il cavo, m_0 è la massa del motore all'estremo superiore del cavo, m_1 è la massa del carico, u è la forza orizzontale impressa dal motore, L è la lunghezza del cavo e g è la costante di gravità:

$$
\begin{aligned}
a_1 &= \frac{u}{m_0} + \frac{g\, m_1 \cos(\alpha) \sin(\alpha)}{m_0 + m_1 \sin^2(\alpha)} - \frac{u\, m_1 \sin^2(\alpha)}{m_0\,(m_0 + m_1 \sin^2(\alpha))} \\
a_2 &= -\frac{g\, m_0 \cos(\alpha)\sin(\alpha)}{m_0 + m_1 \sin^2(\alpha)} + \frac{u \sin^2(\alpha)}{m_0 + m_1 \sin^2(\alpha)} \\
a_3 &= -g + \frac{g\, m_0 \cos^2(\alpha)}{m_0 + m_1 \sin^2(\alpha)} - \frac{u \cos(\alpha) \sin(\alpha)}{m_0 + m_1 \sin^2(\alpha)}
\end{aligned}
$$

Per angoli α piccoli possiamo usare le seguenti approssimazioni di primo ordine $\sin(\alpha) \approx \alpha$, $\cos(\alpha) \approx 1$, $\sin^2(\alpha) \approx 0$ che danno luogo a

$$
\begin{aligned}
a_1 &= \frac{u}{m_0} + \frac{g\, m_1\, \alpha}{m_0} \\
a_2 &= -g\,\alpha \\
a_3 &= -\frac{u\,\alpha}{m_0}
\end{aligned}
$$

Si noti che α può essere approssimato dalla differenza fra le posizioni orizzontali degli estremi superiore ed inferiore del cavo divisa per la lunghezza del cavo. Siccome non siamo

interessati al moto verticale del carico (che inoltre non ha effetto sui moti orizzontali) quattro variabili di stato sono sufficienti, due per la posizione e velocità orizzontali dell'estremo superiore del cavo e due per la posizione e velocità orizzontali dell'estremo inferiore. Allora il modello continuo è descritto da

$$\frac{d}{dt}\begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -H & 0 & H & 0 \\ 0 & 0 & 0 & 1 \\ K & 0 & -K & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} + \begin{bmatrix} 0 \\ J \\ 0 \\ 0 \end{bmatrix} u(t) \tag{9.17}$$

dove $H = g\,m_1/(m_0\,L)$, $K = g/L$, $J = 1/m_0$ e si è usata l'approssimazione $\alpha = (x_3 - x_1)/L$. Si considerino i seguenti valori numerici: $m_0 = 50$ kg, $m_1 = 200$ kg, $L = 5$ m, $g = 9.8$ m/s^2 e supponiamo di dover muovere il carico di 10 m in 5 s, discretizzati ponendo $\Delta t = 0.1$ s. Come nel caso precedente siamo solo interessati a raggiungere lo stato finale e quindi i costi sono fissati a

$$Q := [\,10^{-4}\,]; \qquad P^k := \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad k < N; \qquad P^N := \begin{bmatrix} 10^5 & 0 & 0 & 0 \\ 0 & 10^5 & 0 & 0 \\ 0 & 0 & 10^5 & 0 \\ 0 & 0 & 0 & 10^5 \end{bmatrix}$$

Con questi valori il controllo ottimo risultante è raffigurato nelle figure 9.17, 9.18, 9.19 e 9.20. In figura 9.17 sono rappresentate le due variabili di posizione e in figura 9.20 i loro valori in funzione del tempo. Si può vedere che un'oscillazione di quasi 2 m ha luogo subito dopo la partenza e subito prima dell'arrivo. Ovviamente le velocità dei due estremi del cavo differiscono considerevolmente come illustrato in figura 9.18. La legge di controllo si vede in figura 9.19.

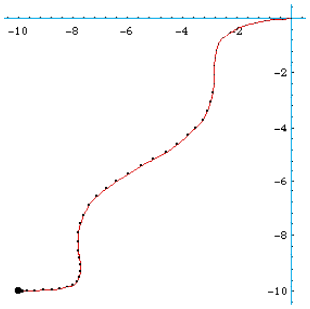


FIGURA 9.17

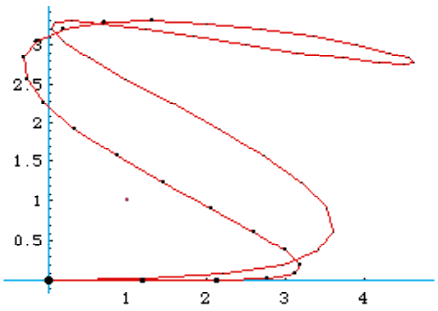


FIGURA 9.18

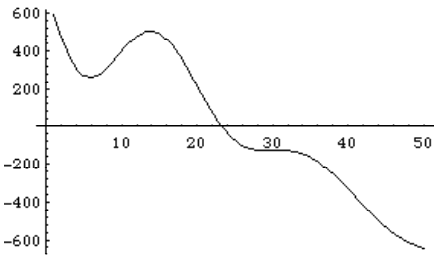


FIGURA 9.19

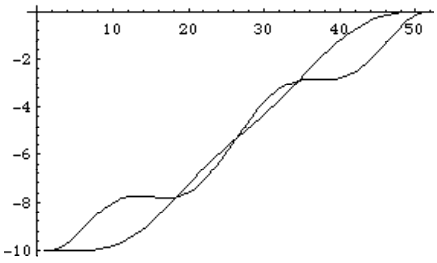


FIGURA 9.20

Se vogliamo ridurre le oscillazioni durante la manovra dobbiamo pesare la differenza $x_1 - x_3$ e se vogliamo ridurre la velocità del carico dobbiamo pesare anche x_4. Questo si può ottenere dando a P^k i valori

$$P^k := \begin{bmatrix} w & 0 & -w & 0 \\ 0 & 0 & 0 & 0 \\ -w & 0 & w & 0 \\ 0 & 0 & 0 & v \end{bmatrix}$$

per opportuni pesi w e v. Possiamo ad esempio usare $w := 10^4$, $v := 10^3$ e modificare il valore di Q a 10^{-6} e i valori diagonali di P^N a 10^6. La nuova legge di controllo si vede nelle figure 9.21 (x_1 vs. x_3), 9.22 (x_2 vs. x_4) e 9.23 (u in funzione del tempo). Le oscillazioni sono naturalmente minori e la legge di controllo è ora alquanto complicata con dei violenti picchi all'inizio e alla fine. I valori non in figura sono dell'ordine di 7,000 N. ∎

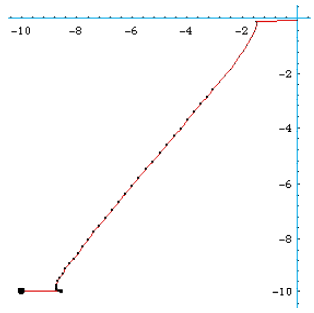


FIGURA 9.21

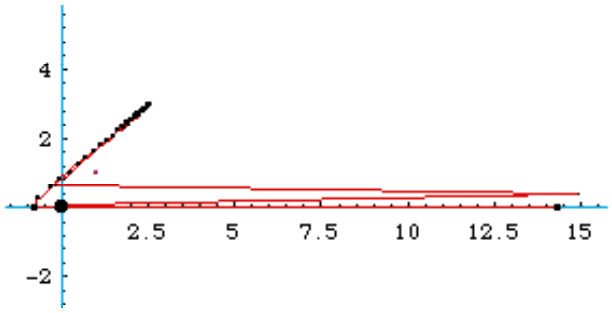


FIGURA 9.22

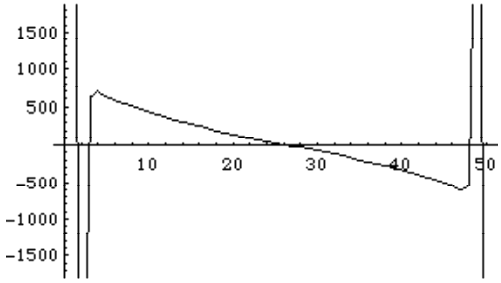


FIGURA 9.23