PROGRAMMAZIONE DINAMICA

1 – Cammini minimi

Sia $G = (N, E)$ un grafo orientato ($n = |N|$ e $m = |E|$) e s un nodo prefissato di G. Consideriamo solo cammini orientati che partono da s. Adotteremo la seguente notazione: se P è un cammino generico (con origine in s come ora assumeremo) e j è un nodo di P, indichiamo con P_j la restrizione di P da s a j. Per evidenziare il fatto che un cammino termina nel nodo i possiamo anche notarlo come P_i.

La lunghezza di un cammino P viene definita nella maggior parte dei modelli a partire da lunghezze (equivalentemente dette anche costi o distanze) di archi c_e, $e \in E$, come $V(P) = \sum_{e \in P} c_e$. Possiamo pensare di ridefinire la lunghezza di un cammino in modo ricorsivo come

$$V(P_s) = 0, \qquad V(P_j) = V(P_i) + c_{ij}, \quad \forall (ij) \in P \tag{1}$$

Si noti che la lunghezza $V(P_j)$ dipende solo dalla lunghezza di P_i e non già dal cammino stesso. L'obiettivo consiste nel trovare per ogni nodo j un cammino $\hat{P}_j$ che minimizza $V(P_j)$ (o massimizza, a seconda del problema) fra tutti i cammini P_i che terminano in i. Fondamentale a questo riguardo è il cosiddetto principio di ottimalità di immediata dimostrazione.

1 Definizione. (Principio di Ottimalità) *Se un cammino $\hat{P}$ è ottimo allora, per ogni nodo k su $\hat{P}$, $\hat{P}_k$ è ottimo fra tutti i cammini che terminano in k.*
Dimostrazione: Se $\hat{P}_k$ non è ottimo, esiste un cammino $\tilde{P}_k$ con $V(\tilde{P}_k) < V(\hat{P}_k)$. Il cammino ottenuto componendo $\tilde{P}_k$ con la restrizione di $\hat{P}$ da k a t è un cammino da s a t di lunghezza minore di $\hat{P}$, contraddicendo l'ipotesi. ∎

Il principio di ottimalità porta alla definizione dell'equazione ricorsiva, nota come *equazione di Bellman*, nelle variabili $V_1, \ldots, V_n$, (per problemi di minimo):

2 Definizione. (Equazione di Bellman)

$$V_s = 0, \qquad V_j = \min_{(ij) \in E} V_i + c_{ij} \qquad \forall j \neq s$$

∎

In base al principio di ottimalità i valori ottimi sono soluzioni dell'equazione di Bellman. Tuttavia vi possono essere soluzioni che non corrispondono a valori ottimi. Ad esempio $c_{s1} = 3$, $c_{s2} = 2$, $c_{s3} = 2$, $c_{12} = 0$, $c_{23} = 0$, $c_{31} = 0$, $c_{24} = 1$, $c_{34} = 1$. Ci sono infinite soluzioni dell'equazione di Bellman, ovvero: $V_s = 0$, $V_1 = V_2 = V_3 = \alpha$, $V_4 = 1 + \alpha$, per ogni $\alpha \leq 2$. Solo i valori con $\alpha = 2$ corrispondono a valori di cammini ottimi. Queste soluzioni spurie sono dovute al ciclo $1 \to 2 \to 3 \to 1$ di lunghezza nulla.

3 Teorema. *Se tutti i cicli sono strettamente positivi, le soluzioni dell'equazione di Bellman sono i valori dei cammini ottimi.*

Dimostrazione: Sia data una soluzione $\tilde{V}$ dell'equazione di Bellman. Sia $k(j)$ il nodo per il quale si ha il minimo nell'espressione

$$\tilde{V}_j = \min_{i:(ij)\in E} \tilde{V}_i + c_{ij} = \tilde{V}_{k(j)} + c_{k(j)j} \tag{2}$$

Se il minimo non è unico si scelga arbitrariamente il nodo fra quelli che rendono minima l'espressione. Si definisca l'insieme di archi

$$\tilde{E} := \{(k(j), j) : j \in N \setminus s\}$$

Affermiamo che $\tilde{E}$ è aciclico (prescindendo dall'orientazione). Infatti un ciclo in cui gli archi non sono tutti orientati nella medesima direzione ha necessariamente un nodo con due archi entranti. Però $\tilde{E}$ ha, per costruzione, un solo arco entrante su ogni nodo diverso da s e nessuno in s. Quindi se esiste un ciclo, questo deve essere orientato. Se esistesse un tale ciclo orientato C si avrebbe in base a (2)

$$\tilde{V}_h = \tilde{V}_h + \sum_{e\in C} c_e$$

per un nodo generico h del ciclo, da cui si avrebbe un ciclo nullo, caso escluso per ipotesi. Quindi, se consideriamo la successione di nodi j, $k(j)$, $k(k(j))$, ... la successione termina nella sorgente a partire da qualsiasi nodo j. Quindi l'insieme $\tilde{E}$ è connesso. Avendo $(n-1)$ archi è necessariamente un albero. Quindi per ogni nodo j il cammino da s a j in $\tilde{E}$ è unico e i valori $\tilde{V}_j$ sono le lunghezze di tali cammini. Resta da dimostrare che i valori $\tilde{V}_j$ sono ottimi. Sia P un cammino generico da s a j. Allora possiamo scrivere

$$V(P_h) = V(P_i) + c_{ih} \qquad (i,h) \in P \tag{3}$$

In base a (2) possiamo scivere

$$\tilde{V}_h \leq \tilde{V}_i + c_{ij} \qquad (i,h) \in P \tag{4}$$

Applicando ricorsivamente (3) e (4) a partire da $\tilde{V}_s = V(P_s) = 0$ lungo il cammino P, ricaviamo $\tilde{V}_j \leq V(P)$. ∎

Se invece esistono cicli negativi l'equazione di Bellman non ammette soluzioni, come si vede immediatamente da

$$V_j = \min_{i:(ij)\in E} V_i + c_{ij} \quad \Longrightarrow \quad V_j \leq V_i + c_{ij} \quad \forall (ij) \in E$$

e sommando le diseguaglianze lungo un ciclo negativo C si ottiene $0 \leq \sum_{e\in C} c_e$ (i valori V_j si annullano a due a due) che contraddice l'ipotesi di negatività del ciclo.

Quindi riassumendo: se tutti i cicli sono positivi c'è un'esatta equivalenza fra soluzioni dell'equazione di Bellman e valori ottimi. Se esiste almeno un ciclo negativo non vi sono cammini ottimi perché il problema è illimitato e non vi sono soluzioni dell'equazione di Bellman. Nel caso limite in cui non ci sono cicli negativi ma non tutti i cicli sono positivi non vi è equivalenza fra le soluzioni dell'equazione di Bellman e i valori ottimi. Per eliminare le soluzioni spurie si può far uso dei seguenti risultati.

4 Lemma. *Sia $\hat{P}$ un cammino minimo con valori $\hat{V}_i := V(\hat{P}_i)$ e sia $\tilde{V}$ una soluzione dell'equazione di Bellman. Allora $\tilde{V}_i \leq \hat{V}_i$ per ogni $i \in \hat{P}$.*

Dimostrazione: Dimostrazione per induzione. Si ha $\tilde{V}_s \leq \hat{V}_s$ (infatti $\tilde{V}_s = \hat{V}_s = 0$). Siano i e j due nodi adiacenti in $\hat{P}$ e si supponga $\tilde{V}_i \leq \hat{V}_i$. Allora

$$\tilde{V}_j = \min_{k:(kj)\in E} \tilde{V}_k + c_{kj} \leq \tilde{V}_i + c_{ij} \leq \hat{V}_i + c_{ij} = \hat{V}_j$$

∎

Trasformiamo ora l'equazione di Bellman nel seguente problema di PL

$$\begin{aligned} \max \quad & \sum_i V_i \\ & V_j \le V_i + c_{ij} \qquad \forall (i,j) \in E \\ & V_s = 0 \end{aligned} \tag{5}$$

5 Teorema. *Le soluzioni ottime di (5) soddisfano l'equazione di Bellman e corrispondono ai valori minimi.*
Dimostrazione: Sia V_i una soluzione ammissibile in (5). Se esiste un nodo j tale che $V_j < V_i + c_{ij}$ per ogni $i : (ij) \in E$, allora deve esistere un'altra soluzione ammissibile V', con $V'_i := V_i$ se $i \neq j$ e $V'_j := \min_{i:(ij)\in E} V_i + c_{ij} > V_j$. Certamente V' è ammissibile, poiché $V'_k = V_k \le V_j + c_{jk} \le V'_j + c_{jk}$. Quindi V non può essere ottimo e necessariamente $\hat{V}_j = \min_{i:(ij)\in E} \hat{V}_i + c_{ij}$ per ogni soluzione ottima. Inoltre ogni soluzione dell'equazione di Bellman è ammissibile in (5). Dal lemma 4 discende la tesi. ∎

Il modello che abbiamo finora presentato considera cammini uscenti da un nodo prefissato s. Possiamo però anche pensare ad un modello che consideri cammini entranti in un nodo prefissato t. Anche in questo caso possiamo adottare una notazione analoga alla precedente per cui, se P è un cammino generico (con destinazione in t questa volta) e j è un nodo di P indichiamo con P^j la restrizione da j a t di P. La lunghezza di un cammino P può essere ricorsivamente definita (a partire da t e operando a ritroso) come

$$V(P^t) = 0, \qquad V(P^i) = V(P^j) + c_{ij}, \quad \forall (ij) \in P \tag{6}$$

Si noti che, dato un cammino P da s a t, la somma $V(P_i)+V(P^i)$ è invariante per ogni i ed è la lunghezza del cammino. L'equazione di Bellman, nelle variabili $V^1, \ldots, V^n$ (valori di cammini da i a t) diventa ovviamente (per problemi di minimo)

$$V^t = 0, \qquad V^i = \min_{j:(ij)\in E} V^j + c_{ij} \qquad \forall i \neq t$$

e porta al seguente problema di PL

$$\begin{aligned} \max \quad & \sum_i V^i \\ & V^i \le V^j + c_{ij} \qquad \forall (i,j) \in E \\ & V^t = 0 \end{aligned} \tag{7}$$

Per risolvere l'equazione di Bellman, c'è il problema che i valori da determinare compaiono da entrambe le parti dell'equazione. L'algoritmo risolutivo, noto come algoritmo di Bellman-Ford, itera, usando valori provvisori a partire da $V_s := 0$, $V_i := +\infty$, $i \neq s$, il calcolo

$$V_j := \min \{V_j \, ; \, V_i + c_{ij}\} \qquad \forall (ij) \in E \tag{8}$$

Se, ad un certo punto i valori V_i non vengono più aggiornati, tali valori soddisfano chiaramente l'equazione di Bellman. Si tratta di capire sotto quali ipotesi l'iterazione (8) termina perché si sono raggiunti valori stazionari e se c'è un limite al numero di iterazioni necessarie. L'algoritmo è descritto in dettaglio nelle tabelline (nelle due versioni in avanti e all'indietro).

6 Teorema. *L'algoritmo di Bellman-Ford trova i cammini ottimi oppure stabilisce che non ci sono camini ottimi con complessità $O(n\,m)$.*
Dimostrazione: (versione in avanti) Sia V_j^k il valore di V_j dopo l'iterazione k-ma (V_j^0 è il valore iniziale). Se $V_j^{k+1} < V_j^k$, allora esiste un nodo i tale che $V_j^{k+1} = V_i^k + c_{ij}$. Al passo precedente, in base all'algoritmo, valeva $V_j^k \le V_i^{k-1} + c_{ij}$ (se $k \ge 1$). Le tre relazioni implicano

$$V_i^k + c_{ij} = V_j^{k+1} < V_j^k \le V_i^{k-1} + c_{ij}$$

Algoritmo di Bellman-Ford

(*versione in avanti*)
input(G, c)
$V_s := 0$; **for** all $i \neq s$ **do** $V_i := \infty$; $p(s) := s$;
for $k := 1$ **to** n **do**
for $(ij) \in E$ **do**
if $V_j > V_i + c_{ij}$
then begin
$V_j := V_i + c_{ij}$;
$p(j) := i$;
end
output(V, p).
	(*versione all'indietro*)
	input(G, c)
	$V^t := 0$; **for** all $i \neq t$ **do** $V^i := \infty$; $p(t) := t$;
	for $k := 1$ **to** n **do**
	for $(ij) \in E$ **do**
	if $V^i > V^j + c_{ij}$
	then begin
	$V^i := V^j + c_{ij}$;
	$p(i) := j$;
	end
	output(V, p).

da cui $V_i^k < V_i^{k-1}$, cioè il valore di V nel nodo i è stato aggiornato al passo precedente. Applicando ricorsivamente questo ragionamento esiste una successione di nodi $i_0, i_1, \ldots$ tali che $V_{i_h}^{k-h+1} < V_{i_h}^{k-h}$. Se $k > n$ nella successione c'è almeno un nodo ripetuto, ad esempio $r := i_{p-q} = i_p$, che crea il ciclo C definito dagli archi $e_1 := (i_p, i_{p-1})$, $e_2 := (i_{p-1}, i_{p-2})$,..., $e_q := (i_{p-q+1}, i_{p-q}) = (i_{p-q+1}, i_p)$ per il quale si ha

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

ed inoltre

$$V_r^{k-p+q+1} = V_r^{k-p+1} + \sum_{e \in C} c_e$$

Confrontando le due espressioni si ha $\sum_{e \in C} c_e < 0$. Quindi, se vi sono aggiornamenti dopo l'n-ma iterazione, esiste un ciclo negativo e quindi non esistono cammini ottimi. Questa affermazione significa anche che se non vi sono cicli negativi, non vi possono essere aggiornamenti dopo l'n-ma iterazione e non sono necessarie più di n iterazioni, da cui la complessità $O(m\,n)$. Se l'algoritmo termina entro n iterazioni i suoi valori soddisfano l'equazione di Bellman. Questo non basta a dire che si tratta dei valori ottimi perché ci potrebbero essere dei cicli nulli. Usando un argomento induttivo supponiamo che ad un'iterazione generica i valori V_j^k soddisfano le diseguaglianze $V_j^k \geq \hat{V}_j$ con $\hat{V}_j$ valori ottimi. Allora abbiamo

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

dove abbiamo sfruttato 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. La tesi segue dal Lemma 4. 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 $h = p(h)$. ∎

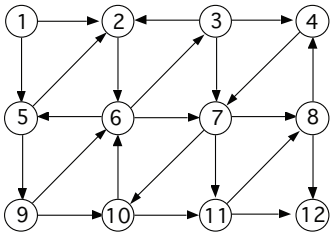
7 Esempio. Sia dato il seguente grafo orientato con i costi indicati degli archi:

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

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

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

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



Vogliamo calcolare il cammino minimo da 1 a 12. Applicando l'algoritmo di Bellman-Ford si ottengono i seguenti valori di V_i:

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).

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. ∎

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).

8 Teorema. *L'algoritmo Floyd-Warshall trova i cammini ottimi fra tutte le coppie di nodi oppure 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.

```
Algoritmo PDA (Programmazione Dinamica Aciclica)
input(G, c);
    V_s := 0; for all i ≠ s do V_i := ∞; p(s) := s;
    Q := {s}; n_j := |δ⁻(j)|, ∀j;
    while Q ≠ ∅ do
    begin
        let i ∈ Q; Q := Q \ {i};
        for all j ∈ δ⁺(i) do
        begin
            n_j := n_j − 1;
            if n_j = 0 then Q := Q ∪ {j};
            if V_j > V_i + c_ij
            then begin
                V_j := V_i + c_ij;
                p(j) := i;
            end
        end
    end
output(V, p).
```

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). ∎

Se il grafo è aciclico esistono sempre soluzioni ottime dato che tutti i cammini sono semplici ed in numero finito e l'equazione di Bellman fornisce necessariamente i valori ottimi. 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 $S = N$.

Algoritmicamente conviene procedere come nell'algoritmo PDA dove, non appena il valore V_i di un nodo i è definitivo, si calcola immediatamente $V_i + c_{ij}$ 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 confronti effettuati. Ovviamente tale numero è uguale al numero di archi del grafo. Pertanto possiamo concludere:

9 Teorema. *Se il grafo è aciclico 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. 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.

Algoritmo di Dijkstra
input(G, c);
    $V_s := 0$; **for** all $i \neq s$ **do** $V_i := \infty$;
    $S := \{s\}$; $k := s$; $p(s) = s$;
    repeat
        for $(kj) \in E$, $j \notin S$ **do**
            if $V_j > V_k + c_{kj}$
            then begin
                $V_j := V_k + c_{kj}$;
                $p(j) = k$
            end
        $k := \text{argmin}_{j \notin S} V_j$;
        $S := S \cup \{k\}$;
    until $(S = N) \vee (V_k = \infty)$
output(V,p)

Se i costi sono non negativi e il grafo è generico si può risolvere l'equazione di Bellman in modo meno dispendioso computazionalmente dell'Algoritmo di Bellman-Ford. Come la proprietà di grafo aciclico garantisce l'esistenza di valori definitivi di V_j, usabili quindi per risolvere esplicitamente l'equazione di Bellman, così anche la proprietà di costi non negativi garantisce l'esistenza di almeno un valore definitivo da usare nell'aggiornamento degli altri valori.

Tale garanzia 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 $S = N$. 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.

L'algoritmo che implementa il ragionamento induttivo sopra delineato è noto come Algoritmo di Dijkstra ed ha complessità $O(n^2)$ o $O(m \log n)$ a seconda dell'implementazione.

Quindi si è visto che il problema di trovare un cammino minimo con costi non negativi è un problema relativamente facile. Se i costi possono essere negativi ma non vi sono cicli negativi il problema continua ad essere facile ed è risolubile polinomialmente con l'algoritmo di Bellman-Ford (ma non con quello di Dijkstra).

Se sono ammessi solo cammini semplici (cioè senza ripetizioni di nodi) l'ipotesi di costi non negativi esclude automaticamente dai cammini minimi i cammini non semplici, dato che non è conveniente percorrere dei cicli e quindi gli algoritmi precedenti sono applicabili perché forniscono necessariamente cammini semplici.

Le cose sono più complicate se si vogliono cammini semplici e i costi possono anche essere negativi. In presenza di cicli negativi il problema diventa NP-hard e gli algoritmi risolutivi sono radicalmente diversi dai precedenti.

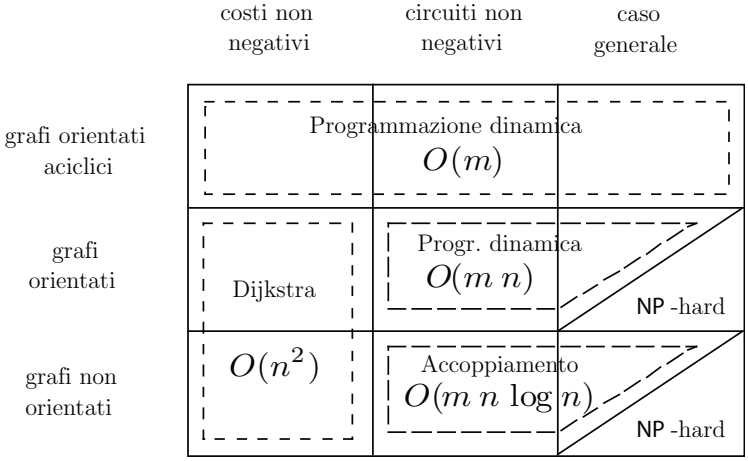
Se si deve risolvere un problema su un grafo non orientato con distanze non negative (in un grafo non orientato si può andare sia da i a j che da j a i al medesimo costo c_{ij}) è possibile trasformare il problema in uno definito su un grafo orientato semplicemente sostituendo ogni arco (i, j) con due 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.

Nel caso non orientato un ciclo ha (per definizione) almeno tre archi distinti. Non è cioè ammesso percorrere avanti e indietro lo stesso arco. Quindi se i costi possono essere negativi ma non vi sono cicli negativi (nel senso appena indicato) i cammini minimi esistono e sono semplici. Tuttavia 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 risolto in modo polinomiale trasformando il grafo e risolvendo un problema di accoppiamento con complessità $O(m\,n\,\log n)$. Tale tecnica verrà illustrata più avanti.

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



2 – Programmazione Dinamica e Programmazione Lineare

Si consideri il problema di trovare un cammino minimo fra due nodi prefissati s e t facendo uso dei modelli di PL (5) o (7). In particolare si prenda in esame il modello (7). Siccome siamo interessati solo ad un cammino (e non a tutti i cammini da i a t) cambiamo la funzione obiettivo di (7) nel seguente modo

$$\begin{array}{ll} \max & V^s \\ & V^i \le V^j + c_{ij} \qquad \forall (i,j) \in E \\ & V^t = 0 \end{array} \tag{9}$$

Il problema può essere riscritto come

$$\begin{array}{ll} \max & V^s - V^t \\ & V^i \le V^j + c_{ij} \qquad \forall (i,j) \in E \end{array} \tag{10}$$

dove le variabili sono le stesse che in (9) a meno di una costante additiva arbitraria. Il duale di (10) è il seguente problema nelle variabili x_{ij}, $(ij) \in E$:

$$\begin{aligned} \min \quad & \sum_{(ij)\in E} c_{ij}\, x_{ij} \\ & \sum_{j\in\delta^+(i)} x_{ij} - \sum_{j\in\delta^-(i)} x_{ji} = \begin{cases} 1 & \text{se } i = s \\ -1 & \text{se } i = t \\ 0 & \text{altrimenti} \end{cases} \qquad i \in N \\ & x_{ij} \geq 0 \end{aligned} \tag{11}$$

Possiamo interpretare ogni variabile x_{ij} come un flusso che percorre l'arco (ij). Allora i vincoli in (11) esprimono il fatto che, sui nodi diversi da s e t, il flusso 'entrante' nel nodo è uguale al flusso 'uscente', mentre nel nodo s il flusso uscente supera di 1 il flusso entrante e nel nodo t il flusso entrante supera di 1 il flusso uscente. Fra i valori ammissibili di x in (11), che in generale sono numeri reali, vi sono anche delle interessanti soluzioni di valore 0 oppure 1, che possono essere interpretate come un vettore d'incidenza di un particolare sottoinsieme di archi. Ad esempio sottoinsiemi corrispondenti a cammini da s a t sono soluzioni ammissibili di (10). Dimostriamo ora che le soluzioni di vertice possono assumere solo i valori 1 o 0.

Per provare questo fatto operiamo una decomposizione di una soluzione $\hat{x}$ ammissibile in (11) in un numero finito di cammini e circuiti. Per rendere il ragionamento più semplice, si immagini di aggiungere un arco (t, s) al grafo con flusso $x_{ts} = 1$. Su questo grafo esteso il flusso rispetta il vincolo di conservazione del flusso in tutti i nodi, cioè

$$\begin{aligned} & \sum_{j\in\delta^+(i)} x_{ij} - \sum_{j\in\delta^-(i)} x_{ji} = 0 \qquad i \in N \\ & x_{ij} \geq 0 \end{aligned}$$

Si consideri ora un arco qualsiasi con flusso strettamente positivo. Sia (i_0, i_1) tale arco e sia ξ_1 il valore del suo flusso. Siccome vi è del flusso positivo entrante in i_1, almeno uno degli archi uscenti da i_1 deve avere un valore di flusso strettamente positivo. Sia (i_1, i_2) tale arco con flusso ξ_2. Proseguendo in modo analogo si deve pervenire ad un nodo $i_p = i_q$, $q < p$, generando un circuito $C = \{(i_q, i_{q+1}), \ldots, (i_{p-1}, i_q))\}$. Sia $\xi := \min_{i=q+1,\ldots,p} \xi_i$. Trasformiamo la soluzione in

$$x_e := \begin{cases} x_e - \xi & \text{se } e \in C \\ x_e & \text{altrimenti} \end{cases}$$

Si noti che $x \geq 0$ (per la scelta di ξ), che $\sum_{j\in\delta^+(i)} x_{ij} - \sum_{j\in\delta^-(i)} x_{ji} = 0$, $i \in N$, (dato che si toglie la medesima quantià ξ sia in entrata che in uscita dai nodi del circuito C) e che il numero di archi con flusso strettamente positivo è diminuito almeno di uno (l'arco in cui il flusso vale proprio ξ).

La procedura viene ripetuta fino ad ottenere $x = 0$. Siccome ad ogni iterazione almeno un arco viene portato a flusso nullo il numero di iterazioni non può superare m. Quindi si ottengono al più m circuiti ad ognuno dei quali viene associato il flusso ξ corrispondente e la soluzione iniziale x può essere vista come ottenuta dalla sovrapposizione dei flussi dei circuiti. Se ora togliamo l'arco aggiunto, tutti i circuiti che vi passano diventano cammini da s a t con somma di flusso uguale a 1. Siano P_i, $i := 1, \ldots, p$, i cammini e C_i, $i := p+1, \ldots, q$ $(\leq m)$, i circuiti e siano ξ_i i valori di flusso corrispondenti. Siano inoltre $e(P_i)$ e $e(C_i)$ i corrispondenti vettori d'incidenza. Quindi

$$x = \sum_{i=1}^{p} \xi_i\, e(P_i) + \sum_{i=p+1}^{q} \xi_i\, e(C_i)$$

con $\sum_{i=1}^{p} \xi_i = 1$ per i vincoli sui nodi sorgente e destinazione e $\xi_i > 0$, $\forall i$. Si noti che posto $\varepsilon := \min\{\xi_i : i := p+1, \ldots, q\}$, le due soluzioni

$$x^+ := x + \varepsilon \sum_{i=p+1}^{q} e(C_i), \qquad x^+ := x - \varepsilon \sum_{i=p+1}^{q} e(C_i)$$

sono ammissibili e siccome $x = (x^+ + x^-)/2$, x non può essere un vertice. Affinché x sia un vertice non devono essere presenti circuiti nella decomposizione. Sia allora

$$x = \sum_{i=1}^{p} \xi_i \, e(P_i)$$

L'espressione indica che x è combinazione convessa di soluzioni ammissibili (i vettori d'incidenza sono soluzioni ammissibili). Quindi x è vertice solo se nella decomposizione compare un unico cammino e in questo caso $\xi_1 = 1$ (siccome $\sum_{i=1}^{p} \xi_i = 1$ e $p = 1$) e allora $x = e(P_1)$.

Quindi ogni soluzione di vertice è un cammino e il valore della funzione obiettivo $\sum_{(ij)\in E} c_{ij}\, x_{ij}$ calcolato su un vertice è la lunghezza del cammino corrispondente, da cui si vede che il problema (11) calcola i cammini minimi da s a t.

Il vantaggio di un modello di PL non consiste nella possibilità di trovare cammini minimi in questo modo anziché con i precedenti algoritmi, certamente computazionalmente più efficienti, ma nel fatto che la flessibilità della PL permette di aggiungere altri vincoli, per i quali invece i precedenti algoritmi diventano inutilizzabili. Nella sezione successiva vedremo un esempio.

Il problema dei cammini minimi da ogni nodo alla destinazione si può anche modellare con la PL. Basta pensare di inviare $(n-1)$ unità di flusso alla destinazione ciascuna in partenza da un nodo diverso, per cui si ha

$$\begin{aligned} \min \quad & \sum_{(ij)\in E} c_{ij}\, x_{ij} \\ & \sum_{j\in\delta^+(i)} x_{ij} - \sum_{j\in\delta^-(i)} x_{ji} = \begin{cases} -n+1 & \text{se } i = t \\ 1 & \text{altrimenti} \end{cases} \qquad i \in N \\ & x_{ij} \geq 0 \end{aligned} \tag{12}$$

In questo caso le soluzioni non saranno in generale 0-1 ma intere non negative. I valori interi definiscono un albero di supporto del grafo. Analogamente il problema dei cammini minimi dalla sorgente ad ogni nodo diventa:

$$\begin{aligned} \min \quad & \sum_{(ij)\in E} c_{ij}\, x_{ij} \\ & \sum_{j\in\delta^+(i)} x_{ij} - \sum_{j\in\delta^-(i)} x_{ji} = \begin{cases} n-1 & \text{se } i = s \\ -1 & \text{altrimenti} \end{cases} \qquad i \in N \\ & x_{ij} \geq 0 \end{aligned} \tag{13}$$

I problemi duali di (12) e (13) sono rispettivamente

$$\begin{aligned} \max \quad & \sum_i (V_i - V_t) \\ & V_i - V_j \leq c_{ij} \end{aligned} \qquad\qquad \begin{aligned} \max \quad & \sum_i (V_s - V_i) \\ & V_i - V_j \leq c_{ij} \end{aligned}$$

il primo dei quali coincide con (10) e il secondo è equivalente a (5), tenuto conto dell'inversione di segno sulle variabili V_i e del vincolo $V_s = 0$.

3 – Esempi combinatori di Programmazione Dinamica

Confronto di stringhe

Un frequente problema di biologia molecolare è quello di allineare sequenze diverse di genoma. Una sequenza di genoma è un frammento di DNA, rappresentabile come una stringa con i simboli dell'alfabeto di 4 lettere A (adenina), G (guanina), T (timina), C (citosina). Allineare due stringhe significa introdurre nelle

stringhe dei simboli vuoti "-" in modo tale che i due simboli in posizione corrispondente delle due stringhe sono uguali oppure uno dei due (ma non entrambi) è il simbolo vuoto. Naturalmente vi sono molti modi di allineare due stringhe. Se pensiamo che l'allineamento è tanto migliore quanto minori sono i simboli vuoti, il problema è quello di trovare l'allineamento migliore.

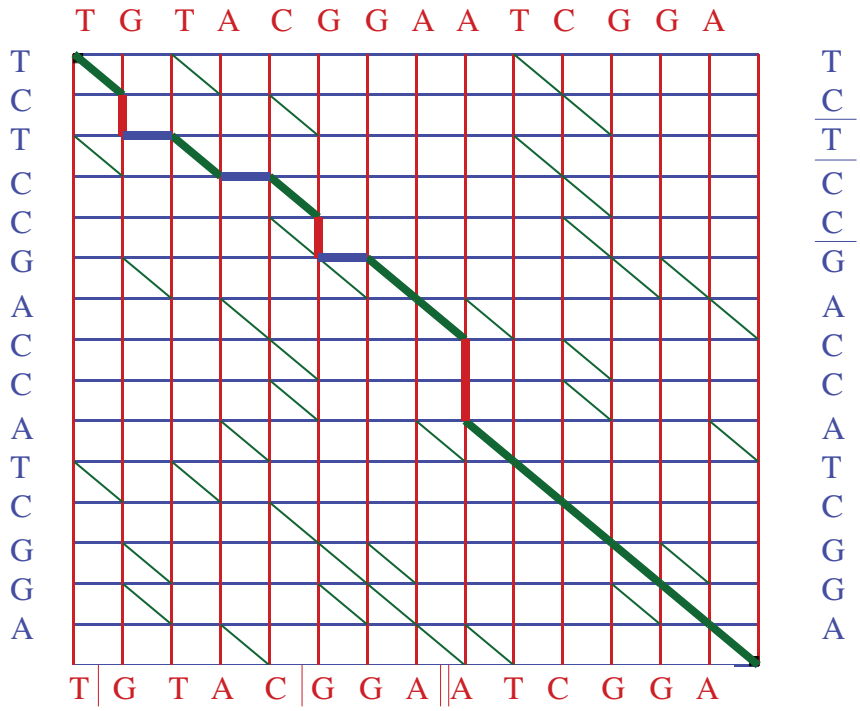
Sia $V(i,j)$ il costo ottimo per allineare le sottostringhe troncate all'i-mo e al j-mo simbolo rispettivamente. Allora si ha la seguente equazione di Bellman:

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

dove

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

Il problema si può rappresentare anche come un grafo a griglia con archi diagonali in corrispondenza di simboli uguali. Gli archi diagonali hanno costo 0 mentre gli archi orizzontali o verticali hanno costo 1. Si cerca il cammino minimo che collega i due vertici opposti del grafo con il massimo numero di archi diagonali.



Si abbia t =TGTACGGAATCGGA e s =TCTCCGACCATCGGA. Il cammino indicato in figura corrisponde all'allineamento

T - G T A C - G G A - - A T C G G A
T C - T - C C - G A C C A T C G G A

Prodotto di matrici

Il prodotto di una matrice $m_1 \times m_2$ con una matrice $m_2 \times m_3$ (il numero di colonne della prima deve essere uguale al numero di righe della seconda, affinché si possa eseguire il prodotto) richiede il calcolo di $m_1 \cdot m_3$ prodotti scalari di vettori di dimensione m_2. Ognuno di questi prodotti richiede il calcolo di m_2 prodotti e $m_2 - 1$ somme. Possiamo quindi assumere (con una leggera approssimazione) che il prodotto delle due matrici abbia costo proporzionale a $m_1 \cdot m_2 \cdot m_3$.

Se si devono moltiplicare fra loro tre matrici A_1, A_2 e A_3 di dimensione $m_1 \times m_2$, $m_2 \times m_3$ e $m_3 \times m_4$ rispettivamente, possiamo eseguire il calcolo nei due seguenti modi alternativi, dato che la legge associativa vale per il prodotto matriciale:

$$(A_1 \cdot A_2) \cdot A_3, \qquad A_1 \cdot (A_2 \cdot A_3)$$

Nel primo modo si eseguono $m_1 \cdot m_2 \cdot m_3$ operazioni per il prodotto $A_1 \cdot A_2$ e poi, siccome la matrice risultante $(A_1 \cdot A_2)$ ha m_1 righe e m_3 colonne, si eseguono $m_1 \cdot m_3 \cdot m_4$ operazioni. Nel secondo modo il numero di operazioni è $m_2 \cdot m_3 \cdot m_4 + m_1 \cdot m_2 \cdot m_4$. I due numeri $(m_1 \cdot m_2 \cdot m_3 + m_1 \cdot m_3 \cdot m_4)$ e $(m_2 \cdot m_3 \cdot m_4 + m_1 \cdot m_2 \cdot m_4)$ sono in generale diversi. Naturalmente siamo interessati ad eseguire i calcoli in modo da minimizzare il numero di operazioni. Se le matrici sono tre è facile scegliere il più piccolo dei due numeri. Se le matrici sono in generale n, il problema diventa più complesso.

Le matrici siano $A_1, A_2, \ldots A_n$. La matrice A_i ha m_i righe e m_{i+1} colonne. Definiamo allora come $V(i,j)$ il costo ottimo del prodotto del blocco di matrici dalla i-ma (compresa) alla j-ma (esclusa). Si noti che il prodotto del blocco (i,j)

$$(A_i \cdot A_{i+1} \cdots A_{j-2} \cdot A_{j-1})$$

è una matrice con m_i righe e m_j colonne e che si ottiene come prodotto dei due blocchi

$$(A_i \cdot A_{i+1} \cdots A_{k-2} \cdot A_{k-1}) \cdot (A_k \cdot A_{k+1} \cdots A_{j-2} \cdot A_{j-1})$$

per un opportuno valore di k, cioè quello che rende minima l'espressione

$$V(i,k) + V(k,j) + m_i \cdot m_k \cdot m_j$$

Allora l'equazione di Bellman è

$$V(i,j) = \min_{i<k<j} V(i,k) + V(k,j) + m_i \cdot m_k \cdot m_j$$

con valori iniziali $V(i,i+1) = 0$. La ricorsione si esegue per blocchi crescenti, ovvero:

$$V(i,i+h) = \min_{i<k<i+h} V(i,k) + V(k,i+h) + m_i \cdot m_k \cdot m_{i+h}\,, \qquad i := 1,\ldots,n+1-h, \qquad h := 2,\ldots,n$$

e il valore ottimo finale è $V(1,n+1)$. Inoltre bisogna memorizzare, per ogni blocco (i,j) il valore $k(i,j)$ che dà il modo migliore di spezzarlo nei due blocchi $(i,k(i,j))$ e $(k(i,j),j)$, per poter poi ricostruire l'ordine di esecuzione dei prodotti. Ad esempio siano date le matrici A_i, $i := 1,\ldots,9$, con i seguenti dati:

$$m = (\,3 \quad 5 \quad 6 \quad 2 \quad 4 \quad 3 \quad 8 \quad 4 \quad 5 \quad 9\,)$$

La ricorsione produce i seguenti valori $V(i,j)$

$i \setminus j$	3	4	5	6	7	8	9	10
1	90	90	114	132	204	250	296	410
2	0	60	100	114	212	236	286	416
3	0	0	48	60	168	184	236	374
4	0	0	0	24	72	136	176	266
5	0	0	0	0	96	144	216	396
6	0	0	0	0	0	96	156	291
7	0	0	0	0	0	0	160	468
8	0	0	0	0	0	0	0	180

e i valori $k(i,j)$

$i \setminus j$	3	4	5	6	7	8	9	10
1	2	2	4	4	6	4	4	4
2	0	3	4	4	4	4	4	4
3	0	0	4	4	4	4	4	4
4	0	0	0	5	6	7	8	9
5	0	0	0	0	6	6	6	9
6	0	0	0	0	0	7	8	9
7	0	0	0	0	0	0	8	8
8	0	0	0	0	0	0	0	9

dai quali si deduce il seguente modo ottimo di eseguire i prodotti:

$$((A_1 \cdot (A_2 \cdot A_3)) \cdot (((((A_4 \cdot A_5) \cdot A_6) \cdot A_7) \cdot A_8) \cdot A_9))$$

che comporta un costo di 410. Il prodotto delle matrici seguendo l'ordine diretto, cioè

$$(((((((A_1 \cdot A_2) \cdot A_3) \cdot A_4) \cdot A_5) \cdot A_6) \cdot A_7) \cdot A_8) \cdot A_9$$

ha un costo di

$$\sum_{i=2}^{n} m_1 \cdot m_i \cdot m_{i+1} = 549$$

mentre seguendo l'ordine inverso

$$A_1 \cdot (A_2 \cdot (A_3 \cdot (A_4 \cdot (A_5 \cdot (A_6 \cdot (A_7 \cdot (A_8 \cdot A_9)))))))$$

si ha un costo di 1377.

10 Esercizio. In quanti modi alternativi si possono moltiplicare n matrici? Trovare un metodo ricorsivo per calcolare questo numero. Esiste anche una formula chiusa che fornisce questo numero (ma non si chiede di dimostrarla), ed è (si tratta dei celebri numeri di Catalan)

$$\frac{1}{n} \binom{2\,n-2}{n-1}$$

La formula può essere usata per verificare il risultato della ricorsione. ∎

Problema dei k insiemi minimi

Siano assegnati n numeri non negativi $a_1, a_2, \ldots, a_n$. Il valore di un sottoinsieme $J \subset \{1, \ldots, n\}$ è definito come

$$v(J) := \sum_{j \in J} a_j$$

Vogliamo calcolare i primi k sottoinsiemi di valore minimo (incluso quello vuoto che ha valore nullo per definizione). I sottoinsiemi sono distinti ma non necessariamente disgiunti.

Sia $V(i,h)$ il valore dell'h-mo ottimo per sottoinsiemi di $\{1, \ldots, i\}$, con $h \leq \min\left\{k, 2^i\right\}$ (ci possono essre meno di k sottoinsiemi se k sottoinsiemi di fatto non esistono, se cioè $k > 2^i$, ovvero $\log_2 k > i$. Indichiamo con $V(i)$ sinteticamente la lista dei valori $V(i,h)$. Allora vale la relazione ricorsiva

$$V(i+1) = \min\left\{V(i)\ ;\ V(i) + a_{i+1}\right\}$$

dove $V(i) + a_{i+1}$ indica che si è sommato a_{i+1} ad ogni elemento della lista $V(i)$ e l'operazione di 'min' è in realtà un'operazione di fusione delle due liste indicate estraendo dalle liste i migliori k valori. L'operazione di fusione, se fatta su liste ordinate ha costo $O(k)$ e produce a sua volta una lista ordinata. Complessivamente quindi l'algoritmo ha complessità $O(k\,n)$. Se i valori a_i sono preventivamente ordinati in modo crescente, l'algoritmo può esssere fatto terminare prematuramente non appena il valore a_i da considerare è non migliore del peggiore sottoinsieme.

Sia ad esempio $a = \{1, 3, 4, 4, 8, 9, 11\}$ e sia $k = 7$. Inizialmente $V(0) = \{0\}$. Conviene anche memorizzare, per ogni elemento della lista, oltre al valore del sottoinsime anche il sottoinsieme stesso. Sia $S(i,h)$ l'insieme corrispondente a $V(i,h)$. Quindi inizialmente $S(0) = \{\emptyset\}$.

Al primo passo si ha

$$V(1) = \min\left\{V(0)\ ;\ V(0) + a_1\right\} = \min\left\{\{0\}, \{0+1\}\right\} = \{0, 1\}, \qquad S(1) = \{\emptyset, \{1\}\}$$

e successivamente

$$V(2) = \min\{V(1)\ ;\ V(1) + a_2\} = \min\{\{0,1\}, \{0+3, 1+3\}\} = \{0,1,3,4\},$$

$$S(2) = \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$$

$$V(3) = \min\{V(2)\ ;\ V(2) + a_3\} = \min\{\{0,1,3,4\}, \{0,1,3,4\} + 4\} = \{0,1,3,4,4,5,7\},$$

$$S(3) = \{\emptyset, \{1\}, \{2\}, \{1,2\}, \{3\}, \{1,3\}, \{2,3\}\}$$

$$V(4) = \min\{\{0,1,3,4,4,5,7\}, \{0,1,3,4,4,5,7\} + 4\} = \{0,1,3,4,4,4,5\},$$

$$S(4) = \{\emptyset, \{1\}, \{2\}, \{1,2\}, \{3\}, \{4\}, \{1,3\}\}$$

Siccome $a_5 = 8 > 5$ il calcolo può terminare.

4 – Problemi di Knapsack

Vi sono vari problemi cosiddetti di knapsack (zaino):

11 Definizione. *Si definisce* knapsack 0-1 *il seguente problema: dati interi positivi $v_1, \ldots, v_n$, (valori), $w_1, \ldots, w_n$, (pesi), K, (capacità), si trovi* $\max\{v(J) : w(J) \leq K,\ J \subset \{1, \ldots, n\}\}$. ∎

Nella definizione si è posto $v(J) := \sum_{i \in J} v_i$, $w(J) := \sum_{i \in J} w_i$.

12 Definizione. *Si definisce* knapsack intero *il seguente problema: dati interi positivi $v_1, \ldots, v_n$, (valori), $w_1, \ldots, w_n$, (pesi), K, (capacità), si trovino interi non negativi $x_1, \ldots, x_n$, tali da massimizzare*

$$\left\{ \sum_{i=1}^{n} v_i\, x_i : \sum_{i=1}^{n} w_i\, x_i \leq K \right\}$$

∎

13 Definizione. *Si definisce* knapsack a scelta multipla *il seguente problema: sono assegnati m insiemi disgiunti $T_1, \ldots, T_m$, e per ogni elemento $j \in T_i$ sono assegnati un valore v_{ij} e un peso w_{ij}. Inoltre è assegnata una capacità K. Bisogna trovare un insieme $J \subset \cup_i T_i$, tale che $|J \cap T_i| \leq 1$, per ogni i, $w(J) \leq K$ e $v(J)$ è massimo.* ∎

14 Definizione. *Si definisce* problema della partizione *il seguente problema: sono assegnati n interi positivi $a_1, \ldots, a_n$, tali che $\sum_i a_i$ è pari. Ci si chiede se esiste un sottoinsieme $J \subset \{1, \ldots, n\}$ tale che $\sum_{i \in J} a_i = \sum_{i \notin J} a_i$.* ∎

```
Algoritmo knapsack 0-1
input(v, w, K);
    for c := 0 to K do begin V(c) := 0; for i := 1 to n do x(i, c) := 0 ; end;
    for i := 1 to n do
    begin
        for c := K − w_i downto 0 do
        begin
            if V(c + w_i) < V(c) + v_i
            then begin
                V(c + w_i) := V(c) + v_i;
                x(i, c + w_i) := 1
            end
        end
        c := K
        for i := n downto 1 do
        begin
            x̂(i) := x(i, c);
            c := c − x̂(i) · w_i;
        end
    end
output(x̂).
```

Si tratta in tutti i casi di problemi **NP**-difficili, anche se, come vedremo, non sono particolarmente intrattabili. Per risolvere un problema di knapsack 0-1 si può far ricorso sia alla PL intera come alla programmazione dinamica.

L'approccio al problema knapsack 0-1 con la programmazione dinamica fa usa della seguente funzione

$$V(i,c) := \text{valore ottimo per l'istanza limitata ai primi } i \text{ oggetti e con capacità } c$$

I valori di $V(i,c)$ vengono inizializzati a $V(0,c) = 0$ per ogni $0 \le c \le K$. L'equazione di Bellman diventa allora

$$V(i,c) = \max\{V(i-1,c)\ ;\ v_i + V(i-1, c-w_i)\} \tag{14}$$

restando inteso che se $c - w_i < 0$ il secondo termine non esiste. L'iterazione viene effettuata per $i := 1, \ldots, n$ e, per ogni indice i, per $c := K, K-1, \ldots, 1$. Operando in qeusto modo si può usare un unico vettore $V(c)$ che viene costantemente aggiornato. Il valore ottimo del problema è oviamente $V(n,K)$. Per ricostruire la soluzione bisogna anche memorizzare per ogni coppia (i,c) un puntatore $x(i,c)$ posto uguale a 0 se il massimo in (14) è dato da $V(i-1,c)$ e posto uguale a 1 altrimenti. Dopodiché la soluzione ottima si ottiene ricorsivamente da

$$c := K\ ;\ \hat{x}_n := x(n,K)\ ;\ c := c - \hat{x}_i\, w_i\ ;\ \hat{x}_i := x(i,c)$$

L'equazione ricorsiva (14) può anche essere rappresentata da un grafo con $n+1$ livelli etichettati $0, 1, \ldots, n, n+1$, e $K+1$ nodi nei livelli $1, \ldots, n$, etichettati (i,c) con $i = 0, \ldots, n$, e $c = 0, 1, \ldots K$. 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 ('diagonale') fra $(i-1,c)$ e $(i, c+w_i)$, per ogni $1 \le c \le K - w_i$, di valore v_i ed un arco ('orizzontale') $(i-1,c)$ e (i,c), per ogni $1 \le c \le K$, di valore nullo. Infine tutti i nodi del livello n sono connessi a t con archi di valore nullo. Ogni cammino da s a t corrisponde ad una scelta ammissibile di un insieme J: se nel passaggio dal livello $i-1$ al livello i si usa un arco diagonale, questo corrisponde alla scelta dell'elemento i. Il cammino di valore massimo corrisponde all'ottimo.

Dalle precedenti considerazioni si vede che la complessità del knapsack è $O(n\,K)$ che è pseudopolinomiale, perché K viene codificato in input con $\log K$ simboli e quindi K è esponenziale rispetto alla stringa d'input.

Nel caso del problema di knapsack intero si fa uso della seguente ricorsione

$$V(c) := \text{valore ottimo per l'istanza con capacità } c$$

$$V(c) = \max_{1 \le i \le n} V(c - w_i) + v_i, \qquad c := 1, \ldots, K \tag{15}$$

Nell'implementazione di (15) bisogna ovviamente tener conto che $V(c-w_i)$ non è definito per $c < w_i$. Inoltre conviene inizializzare $V(c) := 0$ per ogni c. In questo modo si può tenere conto anche di soluzioni che non saturano la capacità (e ve ne possono essere di ottime).

Per rappresentare (15) con un grafo basta costruire un grafo orientato con $K+1$ nodi etichettati $0, \ldots, K$ ed archi $(c, c + w_i)$ di valore v_i per ogni i tale che $c + w_i \le K$ e per ogni $c := 0, \ldots, K - 1$. A questi archi si aggiungono archi $(c, c + 1)$ di valore 0 per ogni c. Quindi ogni cammino da 0 a K corrisponde ad una scelta di valori x_i, in quanto se il cammino usa l'arco di valore v_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 $(c, c + 1)$ corrisponde ad una scelta di diminuire la capacità disponibile senza inserzioni.

Dalle precedenti considerazioni si vede che anche la complessità del problema del knapsack intero è $O(n\,K)$.

15 Esercizio. Si progetti un algoritmo di programmazione dinamica per il problema del knapsack a scelta multipla di complessità $O(n\,K)$ con $n = \sum_j |T_j|$. ∎

Il problema della partizione può essere modellato come un problema di knapsack in cui i valori e i pesi concidono e sono uguali a a_i e la capacità è posta uguale a $\sum_i a_i/2$. Allora il problema della partizione ammette soluzione se e solo se l'ottimo di knapsack vale $\sum_i a_i/2$.

16 Esempio. Nelle elezioni presidenziali americane ogni stato dispone di un certo numero di voti elettorali che vanno tutti al candidato che nello stato ha ottenuto la maggioranza dei voti popolari. Esiste una possibilità in cui entrambi i candidati ottengono la stessa somma di voti elettorali? I dati sono i seguenti (elezioni 2004): AK-3; AL-9; AR-6; AZ-4; CA-55; CO-10; CT-7; DC-3; DE-3; FL-27; GA-15; HI-4; ID-3; IL-21; IN-11; IO-7; KS-6; KY-8; LA-9; MA-12; MD-10; ME-4; MI-17; MN-10; MO-11; MS-6; MT-9; NC-15; ND-3; NE-5; NH-4; NJ-15; NM-6; NV-5; NY-31; OH-20; OK-7; OR-7; PA-21; RI-4; SC-8; SD-3; TN-11; TX-34; UT-3; VA-13; VM-3; WA-11; WI-10; WV-8; WY-5.

Risolvendo con la PLI si trova che tale possibilità effettivamente esiste ed è ad esempio dovuta a questa scelta di stati: AL, CT, DC, FL, GA, IN, KY, LA, MA, MD, MI, NC, NJ, NY, OH, OR, PA, RI, SC, SD, WA, WV.

Come si può capire se esistono anche altre soluzioni? ∎

Nell'approccio con la PL intera il knapsack 0-1 viene modellato nel seguente modo:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} v_i\, x_i \\ & \sum_{i=1}^{n} w_i\, x_i \le K \\ & x_i \in \{0, 1\} \end{aligned}$$

e la risoluzione si basa sul seguente problema rilassato detto *knapsack continuo*:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} v_i\, x_i \\ & \sum_{i=1}^{n} w_i\, x_i \le K \\ & 0 \le x_i \le 1 \end{aligned} \tag{16}$$

La risoluzione di (16) si effettua rapidamente con il seguente algoritmo: si ordinano gli oggetti in base a rapporti v_i/w_i non crescenti. Poi si scelgono gli oggetti secondo l'ordine finché la capacità è soddisfatta.

Quando l'inserzione di un oggetto non può essere effettuata perché la capacità è violata, quell'oggetto viene scelto in modo frazionario fino a saturazione della capacità.

Per giustificare la correttezza dell'algoritmo si effettui la trasformazione di variabile $y_i = w_i\, x_i$, in modo da risolvere il seguente problema:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} \frac{v_i}{w_i}\, y_i \\ & \sum_{i=1}^{n} \quad y_i \leq K \\ & 0 \leq y_i \leq w_i \quad \forall i \end{aligned}$$

Si immagini ora di costruire la soluzione a partire da $y_i = 0$, $\forall i$. Siccome c'è simmetria fra le variabili per il vincolo $\sum_{i=1}^{n} y_i \leq K$, conviene assegnare il più grande valore positivo alla componente corrispondente al massimo rapporto v_i/w_i. Questo viene effettuato saturando tale variabile (cioè fino a $y_i = w_i$) oppure saturando il vincolo $\sum_{i=1}^{n} y_i \leq K$. Si prosegue assegnando il massimo valore positivo alla variabile corrispondente al secondo rapporto v_i/w_i e così di seguito fino alla saturazione del vincolo $\sum_{i=1}^{n} y_i \leq K$. La soluzione così ottenuta è un massimo. Infatti ogni altra soluzione si può ottenere diminuendo i valori di componenti con rapporto v_i/w_i più alto e aumentando quelli di componenti con rapporto v_i/w_i più basso.

Quindi la risoluzione del problema di knapsack continuo dà luogo ad al più una variabile frazionaria, che viene quindi utilizzata per la suddivisione nella procedura branch-and-bound, In ogni nodo dell'albero di ricerca viene risolto un problema di knapsack continuo con il vincolo aggiuntivo che alcune variabili sono vincolate a 0 ed altre a 1. Questo vincolo tuttavia non pregiudica l'esecuzione dell'algoritmo precedentemente delineato. Basta infatti iniziare la scansione dalle variabili vincolate a 1 e non considerare quelle vincolate a 0. Tenuto conto che l'ordinamento dei rapporti v_i/w_i viene effettuato inizialmente e non più ripetuto, in ogni nodo dell'albero di ricerca si esegue un calcolo di complessità $O(n)$ (la scansione delle variabili) e quindi il metodo è abbastanza veloce.