Ricerca Operativa

Problemi di ottimizzazione su reti di flusso e algoritmi per il problema dei cammini minimi

L. De Giovanni

AVVERTENZA: le note presentate di seguito non hanno alcuna pretesa di completezza, né hanno lo scopo di sostituirsi alle spiegazioni del docente. Il loro scopo è quello di fissare alcuni concetti presentati in classe. Le note contengono un numero limitato di esempi ed esercizi svolti. Questi rappresentano una parte fondamentale nella comprensione della materia e sono presentati in classe.

Contents

1 Problemi di flusso di costo minimo

Molti problemi di ottimizzazione combinatoria possono essere modellati ricorrendo ai grafi. Consideriamo il seguente esempio.

Esempio 1 (Un problema di distribuzione di energia) *Una società di produzione di energia elettrica dispone di diverse centrali di produzione e distribuzione, collegate tra loro con cavi reofori. Ogni centrale i può:*

- *produrre p_i kW di energia elettrica ($p_i = 0$ se la centrale non produce energia);*
- *distribuire energia elettrica su una sottorete di utenti la cui domanda complessiva è di d_i kW ($d_i = 0$ se la centrale non serve utenti);*
- *smistare l'energia da e verso altre centrali.*

I cavi che collegano una centrale i ad una centrale j hanno una capacità massima di u_{ij} kW e costano c_{ij} euro per ogni kW trasportato. La società vuole determinare il piano di distribuzione dell'energia elettrica di costo minimo, sotto l'ipotesi che l'energia complessivamente prodotta sia pari alla domanda totale delle sottoreti di utenti.

Per la modellazione matematica del problema consideriamo un grafo orientato costruito come segue. Sia $G = (N, A)$ un grafo i cui nodi corrispondono alle centrali e i cui archi corrispondono ai collegamenti tra le centrali. Per ogni nodo $v \in N$ definiamo il parametro $b_v = d_v - p_v$ che rappresenta la differenza tra la domanda che la centrale deve soddisfare e l'offerta di energia che è in grado di generare. Si noti che:

- $b_v > 0$ se la centrale deve soddisfare una domanda superiore alla capacità produttiva (la centrale deve far arrivare energia da altre centrali);
- $b_v < 0$ se nella centrale c'è un eccesso di offerta (la produzione in eccesso deve essere convogliata verso altre centrali);
- $b_v = 0$ se la domanda e l'offerta di energia si equivalgono o (come caso particolare) se la centrale svolge semplici funzioni di smistamento ($p_v = d_v = 0$).

In generale possiamo dire che b_v è la *richiesta* del nodo $v \in N$ (una richiesta negativa rappresenta un'offerta messa a disposizione della rete) e distinguere:

- nodi domanda (richiesta $b_i > 0$);
- nodi offerta (richiesta $b_i < 0$);
- nodi di transito (richiesta $b_i = 0$).

Le variabili decisionali sono definite sugli archi del grafo e sono relative alla quantità x_{ij} da far fluire sull'arco $(i,j) \in A$. Il modello è il seguente:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} \\ s.t. \quad & \sum_{(i,v)\in A} x_{iv} - \sum_{(v,j)\in A} x_{vj} = b_v \quad \forall v \in N \\ & x_{ij} \leq u_{ij} \quad \forall (i,j) \in A \\ & x_{ij} \in \mathbb{R}_+ \end{aligned}$$

La funzione obiettivo minimizza il costo complessivo di distribuzione mentre i vincoli definiscono le x_{ij} come *flusso ammissibile* sulla rete:

- il primo insieme di vincoli, uno per ogni nodo, è detto *vincolo di bilanciamento dei nodi* e assicura che la differenza tra tutto il flusso (di energia elettrica) entrante in un nodo v (archi (i,v)) e tutto il flusso uscente dal nodo stesso (archi (v,j)) sia esattamente pari alla richiesta del nodo stesso (positiva per i nodi domanda, negativa per i nodi offerta e nulla per i nodi di transito);
- il secondo insieme di vincoli, uno per ogni arco, è il vincolo di *capacità degli archi* e limita la quantità che può fluire in ogni arco.

A questo punto disponiamo di un modello di programmazione lineare per il problema e pertanto, per la sua soluzione, possiamo utilizzare, ad esempio, il metodo del simplesso.

Il modello presentato per questo problema di distribuzione di energia elettrica può essere facilmente generalizzato a diversi altri problemi di distribuzione su reti: reti di distribuzione di beni materiali (il flusso è relativo al trasporto dei beni e i nodi rappresentano punti di produzione, consumo e smistamento), reti di telecomunicazioni (il flusso rappresenta la banda da riservare su ogni collegamento), reti di trasporto (il flusso rappresenta i veicoli sulla rete) etc. In generale, il problema prende il nome di *problema del flusso su reti di costo minimo* e può essere utilizzato per modellare e risolvere svariati problemi di ottimizzazione combinatoria (anche non direttamente modellabili su una rete di flusso).

2 Flusso di costo minimo e matrici totalmente unimodulari

Consideriamo ora l'esempio in Figura 1 (i numeri sugli archi rappresentano un flusso ammissibile).

Se trascuriamo i vincoli di capacità degli archi, il modello di flusso di costo minimo presenta i seguenti vincoli, scritti per esteso:

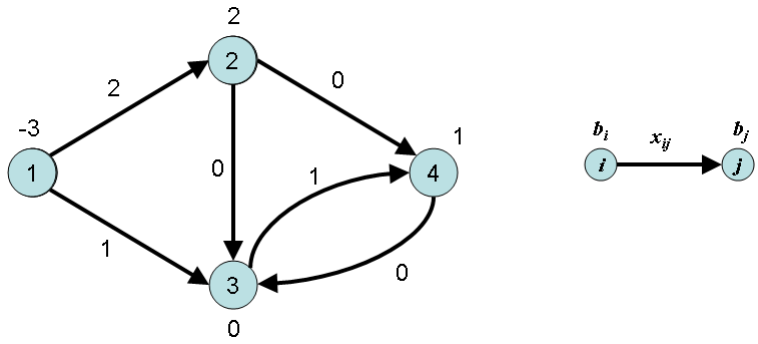

Figure 1: Esempio di flusso su rete.

$$
\begin{array}{rrrrrrrrrrrrl}
- & x_{12} & - & x_{13} & & & & & & & = & -3 & \text{bilanciamento del nodo 1} \\
+ & x_{12} & & & - & x_{23} & - & x_{24} & & & = & 2 & \text{bilanciamento del nodo 2} \\
 & & + & x_{13} & + & x_{23} & & & - & x_{34} + x_{43} & = & 0 & \text{bilanciamento del nodo 3} \\
 & & & & & & + & x_{24} & + & x_{34} - x_{43} & = & 1 & \text{bilanciamento del nodo 4}
\end{array}
$$

e la corrispondente matrice E dei vincoli è la seguente

x_{12}	x_{13}	x_{23}	x_{24}	x_{34}	x_{43}
-1	-1	0	0	0	0
$+1$	0	-1	-1	0	0
0	$+1$	$+1$	0	-1	$+1$
0	0	0	$+1$	$+1$	-1

Abbiamo una colonna per ogni variabile e, quindi, per ogni arco $(i, j) \in A$ e una riga per ogni nodo $v \in N$. Per ogni colonna, ci sono esattamente due numeri diversi da 0 e, in particolare un $+1$ e un -1. In effetti, il flusso x_{ij} su ogni arco (i, j) contribuisce al flusso di soli due nodi: viene aggiunto al bilanciamento del flusso del nodo in cui entra ($+1$ nella riga j) e viene sottratto dal bilanciamento del flusso del nodo da cui esce (-1 sul nodo i).

> Quindi, in generale, la matrice corrispondente ai vincoli di bilanciamento di flusso ha, per ogni colonna, un $+1$, un -1 e tutti gli altri elementi a 0.

Essendo la somma delle righe di E pari al vettore nullo, il rango della matrice E non è massimo. Si può in effetti dimostrare che:

$$\rho(E) = |N| - 1$$

Di conseguenza, affinché il sistema di equazioni $Ex = b$ relativo al bilanciamento di flusso abbia soluzione, è necessario che $\rho(E) = \rho(E|b) = |N| - 1$ e cioè che anche la somma di tutti i b_i sia 0:

$$\sum_{i \in N} b_i = 0$$

Tale condizione corrisponde a una *rete di flusso bilanciata*. Sotto tale condizione possiamo eliminare uno qualsiasi dei vincoli del sistema $Ex = b$, in quanto ridondante.

Se proviamo a calcolare il determinante di sottomatrici quadrate (di dimensioni 1×1, 2×2, 3×3 e 4×4), otteniamo come risultati sempre $+1$, -1 oppure 0. Il risultato è generalizzabile, grazie alla semplicità della matrice dei vincoli di bilanciamento. Si può infatti dimostrare la seguente proprietà.

Proprietà 1 *Sia E la matrice corrispondente ai vincoli di bilanciamento di flusso e D una qualsiasi sottomatrice quadrata (di qualsiasi dimensione). Allora* $\det(D) \in \{-1, 0, +1\}$.

Matrici con tale proprietà si dicono *matrici totalmente unimodulari (TUM)*. Quindi, la matrice relativa ai vincoli di bilanciamento di flusso è TUM.

Le matrici TUM hanno particolare rilevanza nell'ambito della programmazione lineare. Ricordiamo che una base B della matrice E è una sottomatrice quadrata di E di rango massimo ($|N|-1$). Comunque sia scelta una base B della matrice E, avremo che $det(B) \in \{-1, +1\}$, essendo lo 0 escluso dal fatto che la matrice B ha rango massimo (ed è quindi non singolare e invertibile). Si ricorda che

$$B^{-1} = \frac{1}{det(B)} B^{*T}$$

dove B^* è la matrice dei cofattori o complementi algebrici di B:

$$B^* = \begin{pmatrix} \text{cof}(B, B_{1,1}) & \dots & \text{cof}(B, B_{1,j}) \\ \vdots & \ddots & \vdots \\ \text{cof}(B, B_{i,1}) & \dots & \text{cof}(B, B_{i,j}) \end{pmatrix}^T$$

dove $B_{i,j}$ è l'elemento di B in posizione i, j. Il cofattore in posizione i, j è definito come:

$$\text{cof}(B, B_{i,j}) = (-1)^{i+j} \cdot \det(\text{minor}(B, i, j))$$

dove $minor(B, i, j)$ rappresenta il minore di B che si ottiene cancellando la riga $i-$esima e la colonna $j-$esima. Ora, E è TUM, e quindi:

- $B_{i,j} \in \{-1, 0, +1\}$, visto che ogni elemento è una particolare sottomatrice quadrata di E di ordine 1 il cui determinante (e quindi l'elemento stesso) è -1, 0 o $+1$;
- $\det(\text{minor}(B, i, j)) \in \{-1, 0, +1\}$ visto che $\text{minor}(B, i, j)$ è una sottomatrice di E.

Di conseguenza, ogni elemento di B^{-1} è -1, 0 o $+1$. Consideriamo ora una soluzione di base del sistema $Ex = b$, che rappresenta i vincoli di bilanciamento di flusso. Questa sarà ottenuta come:

$$x = \begin{bmatrix} x_B \\ x_E \end{bmatrix} = \begin{bmatrix} B^{-1}\tilde{b} \\ 0 \end{bmatrix}$$

dove B è una sottomatrice quadrata non singolare di E di dimensione $|N| - 1$[1] e $\tilde{b}$ è un sottovettore di b corrispondente all'eliminazione di un vincolo ridondante. In particolare, se $b \in \mathbb{Z}_+$, tutte le soluzioni di base avranno coordinate intere, essendo le variabili in base ottenute dal prodotto di una matrice intera (in particolare di -1, 0 o $+1$) per un vettore intero, e le variabili fuori base pari a 0 (numero intero).

Il risultato è generalizzabile ad un qualsiasi problema di programmazione lineare.

Proprietà 2 *Sia dato un problema di programmazione lineare in forma standard* $\min\{c^T x : Ax = b, x \geq 0\}$. *Se A è TUM e $b \in Z_+^m$, allora tutte le soluzione di base hanno coordinate intere.*

Un'importante conseguenza è che, se risolviamo il problema con l'algoritmo del simplesso, *che si applica solo quando le variabili sono definite reali*, ma che esplora solo soluzioni di base, otterremo una soluzione ottima di base *a coordinate intere*.

> Sia dato il problema di programmazione lineare ***intera*** $\min\{c^T x : Ax = b, x \in \mathbb{Z}_+^n\}$. Se A è TUM, il problema può essere risolto con l'algoritmo del simplesso.

Tornando al problema del flusso di costo minimo, possiamo anche dimostrare che la proprietà di totale unimodularità della matrice dei vincoli è conservata, anche se si aggiungono i vincoli di capacità degli archi. Supponiamo adesso che i beni che circolano sulla rete di flusso non siano divisibili (frigoriferi, passeggeri etc.). Il modello del flusso di costo minimo sarebbe:

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} \\
s.t. \quad & \sum_{(i,v)\in A} x_{iv} - \sum_{(v,j)\in A} x_{vj} = b_v \quad \forall v \in N \\
& x_{ij} \leq u_{ij} \quad \forall (i,j) \in A \\
& x_{ij} \in \mathbb{Z}_+
\end{aligned}$$

in cui si utilizzano variabili intere. Ma, per le osservazioni sopra riportate, e sotto l'ipotesi (ragionevole) che i fattori di bilanciamento b_i siano interi e l'ipotesi di utilizzare l'algoritmo del simplesso per la soluzione del problema, possiamo in modo equivalente utilizzare lo stesso modello con variabili reali.

[1] Per ottenere una base B bisogna sempre escludere una riga di E, corrispondente ad un vincolo ridondante sotto l'ipotesi di bilanciamento della rete di flusso

3 Il problema del cammino minimo

Uno dei problemi classici sui grafi è il problema del cammino minimo, così definito. Sia dato un grafo $G = (N, A)$ con un costo c_{ij}, $\forall (i,j) \in A$, un nodo origine $s \in N$ e un nodo destinazione $d \in N$. Si vuole trovare il cammino P da s a d il cui costo (somma dei $c_{ij} : (i,j) \in P$) sia minimo. Introduciamo delle variabili decisionali binarie in corrispondenza degli archi del grafo:

$$x_{ij} = \begin{cases} 1, & \text{l'arco } (i,j) \text{ è sul cammino minimo;} \\ 0, & \text{altrimenti.} \end{cases}$$

Un possibile modello è il seguente:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} \\ \text{s.t.} \quad & \\ & \sum_{(i,v)\in A} x_{iv} - \sum_{(v,j)\in A} x_{vj} = \begin{cases} -1, & v = s; \\ +1, & v = d; \\ 0, & v \in N \setminus \{s,d\}. \end{cases} \\ & x_{ij} \in \{0,1\} \quad \forall\, (i,j) \in A \end{aligned}$$

Le variabili x_{ij} selezionano un insieme di archi a costo minimo. I vincoli stabiliscono che si può selezionare un solo arco uscente dall'origine; si può selezionare un solo arco entrante nella destinazione; per gli altri nodi, il numero di archi entranti selezionati deve essere uguale al numero di archi uscenti. Di fatto, le x_{ij} rappresentano un flusso unitario da s a d e il problema è stato formalizzato come un caso particolare del problema di flusso di costo minimo: si vuole far partire un'unità di flusso dall'origine s ($b_s = -1$) e farla arrivare alla destinazione d ($b_s = +1$) facendole compiere un percorso di costo minimo attraverso gli altri nodi della rete ($b_i = 0,\ \forall i \in N \setminus \{s,d\}$). Osserviamo che, anche eliminando il vincolo di dominio $x_{ij} \in \{0,1\}$, una soluzione ottima non avrà mai flussi superiori a 1, altrimenti si pagherebbe di più in funzione obiettivo. Pertanto, si potrebbe scrivere $x_{ij} \in \mathbb{Z}_+$. Inoltre, per le osservazioni sulla totale unimodularità della matrice dei vincoli, possiamo risolvere il problema del cammino minimo risolvendo il seguente modello di programmazione lineare:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} \\ \text{s.t.} \quad & \\ & \sum_{(i,v)\in A} x_{iv} - \sum_{(v,j)\in A} x_{vj} = \begin{cases} -1, & v = s; \\ +1, & v = d; \\ 0, & v \in N \setminus \{s,d\}. \end{cases} \\ & x_{ij} \in \mathbb{R}_+ \quad \forall\, (i,j) \in A \end{aligned}$$

Visto che le variabili sono dichiarate come variabili reali e positive, possiamo applicare l'algoritmo del simplesso per risolvere il problema del cammino minimo.

4 Algoritmo label correcting per il problema del cammino minimo

La possibilità di usare un modello di programmazione lineare (non intera) per il problema del cammino minimo non ha solo la conseguenza di poter applicare l'algoritmo del simplesso, ma ha conseguenze più importanti. A questo punto, infatti, è possibile applicare al problema del cammino minimo la teoria della dualità in programmazione lineare (che non sarebbe applicabile se le variabili fossero intere) e derivare delle proprietà che possono essere sfruttate per la messa a punto di algoritmi di soluzione più efficienti. Il simplesso, infatti, ha complessità "esponenziale", mentre, vedremo, il problema del cammino minimo può essere risolto in tempo polinomiale.

Per applicare la teoria della dualità, scriviamo innanzitutto il duale del problema del cammino di costo minimo. Introduciamo una variabile π_i per ogni vincolo del primale e, quindi, per ogni nodo $i \in N$. Il problema duale avrà quindi la seguente funzione obiettivo:

$$\max \pi_d - \pi_s$$

e un vincolo per ogni variabile x_{ij}, cioè per ogni arco $(i, j) \in A$. La scrittura del vincolo duale relativo alla variabile x_{ij} considera la colonna della variabile stessa che, come abbiamo visto, presenta solo due elementi diversi da 0: $+1$ in corrispondenza del vincolo del nodo j (variabile duale π_j) e -1 in corrispondenza del vincolo del nodo i (variabile duale π_i). Il vincolo duale sarà quindi del tipo:

$$\pi_j - \pi_i \leq c_{ij}$$

Complessivamente, il duale del problema del cammino minimo è il seguente:

$$\begin{aligned} \max \quad & \pi_d - \pi_s \\ \text{s.t.} \quad & \pi_j - \pi_i \leq c_{ij} \quad \forall\, (i,j) \in A \\ & \pi_v \in \mathbb{R} \quad \forall\, v \in N \end{aligned}$$

Notiamo che i vincoli del problema duale sono particolarmente semplici. Si potrebbe quindi pensare di implementare un algoritmo che stabilisca dei valori ammissibili per le variabili π. Il valore π_i è anche detto *etichetta* (o *label*) del nodo i. Se, in corrispondenza di etichette ammissibili, riusciamo a costruire una soluzione del problema primale (variabili x) che sia ammissibile primale e con ugual valore della funzione obiettivo (oppure in scarti complementari con π), allora avremmo ottenuto una soluzione ottima per il problema del

cammino minimo[2]. Un algoritmo molto semplice che permette di ottenere delle etichette ammissibili è il seguente, detto *label correcting* perché procede *correggendo* iterativamente le etichette che non soddisfano il vincolo duale:

Algoritmo *label correcting* generico

π_s := 0; set $p(s) = \wedge$;
for each $v \in N - s$ { set $\pi_v(v)$:=$+\infty$, set $p(v) = \wedge$; }
while ($\exists\,(i,j) \in A : \pi_j > \pi_i + c_{ij}$) do {
    set π_j := $\pi_i + c_{ij}$
    set $p(j)$:= i;
}

I primi passi sono di inizializzazione. Si fa notare che se aggiungiamo una stessa costante (sia positiva che negativa) a *tutte* le etichette, non cambia niente in termini di valore della funzione obiettivo duale e di soddisfazione dei vincoli duali (la stessa costante viene sempre prima sommata e poi sottratta). La soluzione duale è invariante rispetto all'aggiunta di una costante e quindi, possiamo fissare una variabile duale ad un valore arbitrario. Scegliamo di fissare $\pi_s = 0$. Si noti che tale osservazione corrisponde all'esistenza di un vincolo primale ridondante, e che fissare $\pi_s = 0$ corrisponde ad eliminare il vincolo del bilanciamento del nodo origine (la variabile π_s non compare più nel duale). Per gli altri nodi, scegliamo un valore iniziale molto elevato, con la convenzione che

$$+\infty \pm cost. = +\infty$$

Per il resto, l'algoritmo è un semplice ciclo che, ad ogni iterazione, controlla se un *qualsiasi* vincolo duale non è rispettato e, nella stessa iterazione, fa in modo che lo stesso vincolo sia rispettato all'uguaglianza, aggiornando opportunamente l'etichetta della testa dell'arco corrispondente. L'algoritmo, inoltre, mantiene un vettore di puntatori p, uno per ogni nodo. Ogni volta che l'etichetta di un nodo j viene aggiornata, il puntatore punta al nodo i che è coda dell'arco che ha causato l'aggiornamento. Prima di dimostrare la correttezza dell'algoritmo, vediamo un esempio di applicazione.

Esempio 2 *Si applichi l'algoritmo* label correcting generico *al grafo in Figura 2, per calcolare un insieme di etichette ammissibili duali con nodo origine $s = 1$ e nodo destintazione $d = 6$.*

Inizializzazioni: $\pi_1 = 0$; $\pi_2 = \pi_3 = \pi_4 = \pi_5 = \pi_5 = +\infty$;
Iterazione 1: arco $(1, 2)$: $\pi_2 = 7$; $p(2) = 1$.
Iterazione 2: arco $(2, 4)$: $\pi_4 = 11$; $p(4) = 2$.

[2]In realtà, questa tecnica è utilizzata molto spesso, per derivare degli algoritmi risolutivi di problemi di ottimizzazione combinatoria che siano modellabili come problemi di programmazione lineare.

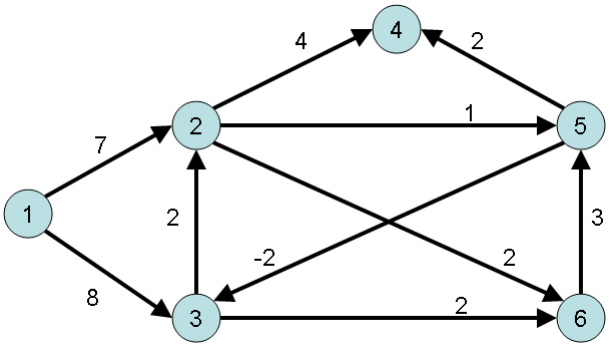

Figure 2: Rete di flusso dell'Esempio 2.

Iterazione 3: arco $(1,3)$: $\pi_3 = 8$; $p(3) = 1$.
Iterazione 4: arco $(3,6)$: $\pi_6 = 10$; $p(6) = 3$.
Iterazione 5: arco $(6,5)$: $\pi_5 = 13$; $p(5) = 6$.
Iterazione 6: arco $(2,5)$: $\pi_5 = 8$; $p(5) = 2$.
Iterazione 7: arco $(5,3)$: $\pi_3 = 6$; $p(3) = 5$.
Iterazione 8: arco $(5,4)$: $\pi_4 = 10$; $p(4) = 5$.
Iterazione 9: arco $(3,6)$: $\pi_6 = 8$; $p(6) = 3$.

A questo punto, tutti gli archi soddisfano i vincoli duali e l'algoritmo termima.

Si fa notare che, seguendo a ritroso i puntatori memorizzati nelle variabili $p(i)$, a partire dal nodo d, si ottiene un cammino da s a d:

$$6 \leftarrow 3 \leftarrow 5 \leftarrow 2 \leftarrow 1$$

e che il costo di questo cammino è proprio π_6: $7 + 1 - 2 + 2 = 8$.

4.1 Proprietà dell'algoritmo

La dimostrazione di correttezza dell'algoritmo usa le condizioni di complementarietà primale duale. Per la loro applicazione, dobbiamo dimostrare il seguente Lemma.

Lemma 1 *Alla fine di ogni iterazione, considerati i valori correnti delle etichette π, per ogni nodo $j \in N : \pi_j < +\infty$ si ha:*

- *esiste un cammino P dal nodo origine s a j;*
- *$p(j)$ è il predecessore di j in tale cammino P;*
- *il costo del cammino P è pari a π_j.*

Dimostrazione: Dimostriamo l'asserto per induzione[3]. Dimostriamo che la proposizione è vera all'iterazione $k = 1$. Alla fine dell'iterazione 1, esistono al più due nodi $j : \pi_j < +\infty$. Il primo nodo è l'origine s. Per tale nodo è immediato verificare che: esiste un cammino da s (formato dal solo nodo s); il predecessore di s in questo cammino è NULL, come stabilito da $p(s)$; il costo di questo cammino è 0, come riportato da π_s. Sia $w \neq s$ il secondo nodo tale che $\pi_w < +\infty$. Se $\pi_w < +\infty, w \neq s$ alla fine della prima iterazione, allora π_w è stato aggiornato, cioè $\exists\ i : \pi_w > \pi_i + c_{iw}$ e, di conseguenza, $\pi_i < +\infty$. Ma all'inizio della prima iterazione solo $\pi_s < +\infty$, cioè $i = s$. Di conseguenza, abbiamo posto $\pi_w = c_{sw}$ e $p(w) = s$. Anche per w è quindi immediato che: esiste un cammino $s \to w$ da s a w; il predecessore di w in questo cammino è $s = p(w)$; il costo del cammino è $c_{sw} = \pi_w$. È quindi vero che, all'iterazione 1, le tre condizioni sono vere per ogni nodo $j : \pi_j < +\infty$.

Assumiamo ora (ipotesi induttiva) che la proposizione sia vera all'iterazione k, e cioè che, per ogni $v \in N : \pi_v < +\infty$, si abbia un cammino $P = s \to v_1 \to v_2 \to \cdots \to p(v) \to v$ di lunghezza pari a π_v (vedi Figura 3).

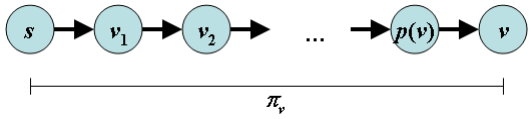

Figure 3: Ipotesi induttiva.

All'iterazione $k + 1$ si aggiorna un $\pi_j : \pi_j > \pi_i + c_{ij}$. Se tale condizione è rispettata, deve essere $\pi_i < +\infty$ (all'iterazione k) e, per ipotesi induttiva, esiste un cammino $P = s \rightsquigarrow p(i) \to i$ di lunghezza pari a π_i. Dopo aver aggiornato $\pi_j = \pi_i + c_{ij}$ e $p(j) = i$, possiamo dire che (vedi Figura 4):

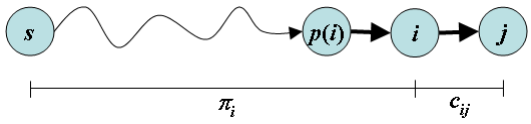

Figure 4: Passo induttivo.

esiste un cammino $P' = P \to j = s \rightsquigarrow p(i) \to i \to j$ da s a j; il predecessore di j su questo cammino è $i = p(j)$; il costo del cammino è $\pi_i + c_{ij} = \pi_j$. Le tre condizione valgono quindi per il nodo j la cui etichetta è stata aggiornata al passo $k + 1$ e continua a valere

[3]Il principio di induzione afferma che, data una proposizione Π definita su un numero naturale k allora:

$$\begin{cases} \Pi(1) = true \\ \Pi(k) = true \Rightarrow \Pi(k+1) = true \end{cases} \quad \Rightarrow \quad \Pi(n) = true,\ \forall\ n \geq 1$$

$\Pi(k) = true$ è detta *ipotesi induttiva*.

per gli altri nodi con etichetta finita, visto che tali etichette e i relativi puntatori non sono stati modificati. Per induzione, quindi, la proprietà vale per tutte le iterazioni $n \geq 1$. ■

4.2 Convergenza e complessità

Per il Lemma 1, se $\pi_j < +\infty$, π_j è la lunghezza di un cammino ammissibile da s a j. In ogni caso, π_j rappresenta sempre un limite superiore (upper bound) alla lunghezza del cammino minimo da s a j. In altri termini, π_j NON può scendere mai sotto la lunghezza del cammino minimo. Inoltre, se l'algoritmo itera, allora un'etichetta viene diminuita (strettamente). Consideriamo adesso due casi, relativi all'esistenza o meno di cicli di costo negativo nel grafo.

Se esiste un ciclo negativo, per i nodi j nel ciclo sarà sempre possibile diminuire le relative etichette, come risulta evidente dall'esempio in Figura 5 (troviamo sempre un arco (i, j) : $\pi_j > \pi_i + c_{ij}$).

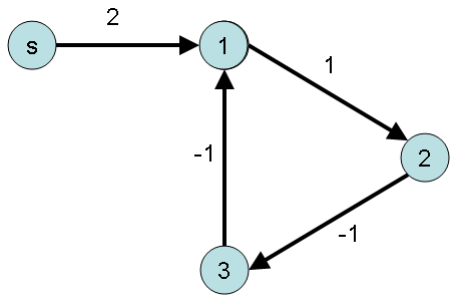

Figure 5: Esempio di flusso su rete.

Inizializzazioni: $\pi_s = 0$; $\pi_1 = \pi_2 = \pi_3 = +\infty$;
Iterazione 1: arco $(s, 1)$: $\pi_1 = 2$; $p(1) = s$.
Iterazione 2: arco $(1, 2)$: $\pi_2 = 3$; $p(2) = 1$.
Iterazione 3: arco $(2, 3)$: $\pi_3 = 2$; $p(3) = 2$.
Iterazione 4: arco $(3, 1)$: $\pi_1 = 1$; $p(1) = 3$.
Iterazione 5: arco $(1, 2)$: $\pi_2 = 2$; $p(2) = 1$.
Iterazione 6: arco $(2, 3)$: $\pi_3 = 1$; $p(3) = 2$.
Iterazione 7: arco $(3, 1)$: $\pi_1 = 0$; $p(1) = 3$.
Iterazione 8: arco $(1, 2)$: $\pi_2 = 1$; $p(2) = 1$.
Iterazione 9: arco $(2, 3)$: $\pi_3 = 0$; $p(3) = 2$.
Iterazione 10: arco $(3, 1)$: $\pi_1 = -1$; $p(1) = 3$.
...

Quindi, in presenza di un ciclo negativo, l'algoritmo non converge.

Consideriamo adesso il caso in cui non esistono cicli di costo negativo. In questo caso, il costo di un cammino minimo da s a d, se d è raggiungibile dall'origine s, è comunque limitato ($> -\infty$). Inoltre, se un nodo non è raggiungibile da s, allora la sua etichetta non viene mai aggiornata rimane a $+\infty$ (se così non fosse, per il Lemma 1 dovrebbe esistere un cammino da s, cadendo in contraddizione). Tali nodi non raggiungibili, quindi, non fanno aumentare il numero di iterazioni.

Finché l'algoritmo itera, almeno un π_j viene diminuito. Se l'algoritmo iterasse all'infinito, questo π_j arriverebbe a scendere sotto il costo di un cammino ammissibile da s a j, il che non è possibile, per il Lemma 1. Pertanto, l'algoritmo converge in un numero finito di iterazioni.

Per quanto riguarda la complessità computazionale asintotica, nel caso di non esistenza di cicli negativi, bisogna considerare il numero di iterazioni nel caso peggiore. Supponiamo che i costi sugli archi siano interi[4]: ad ogni iterazione, un π_j scende di almeno una unità.

Osserviamo inoltre che:

Osservazione 1 *Se non esistono cicli negativi, un cammino minimo contiene al più* $|N|-1$ *archi.*

Infatti, se ci fossero più archi, necessariamente un nodo sarebbe toccato più di una volta; esisterebbe quindi un ciclo nel cammino di costo > 0, che contraddirrebbe l'ottimalità, o di costo $= 0$, che può essere eliminato senza perdere in ottimalità.

Sotto questa osservazione, è possible stabilire sia un limite superiore $\overline{M}$, sia un limite inferiore $\underline{M}$ per il costo del cammino minimo da s verso un qualsiasi nodo j. Ad esempio:

$$\overline{M} = (n-1) \max_{(i,j)\in A} c_{ij}$$

$$\underline{M} = (n-1) \min_{(i,j)\in A} c_{ij}$$

Possiamo pertanto porre convenzionalmente $+\infty = \overline{M}$ e, in questo modo, il numero massimo di iterazioni sarebbe di $\overline{M} - \underline{M}$ per ogni etichetta ($|N| - 1$ etichette in tutto). Ad ogni iterazione si seleziona un arco (si potrebbe fare con un massimo $|A|$ operazioni di confronto e scegliendo a caso uno degli archi che violano il vincolo duale) e si effettuano due operazioni di assegnazione (tempo costante). Abbiamo pertanto dimostrato che:

Proprietà 3 *Se il grafo non presenta cicli di costo negativo, l'algoritmo* label correcting generico *converge in* $O((|N|-1)\,(\overline{M} - \underline{M})\,|A|)$.

Si fa notare che la complessità dipende da $\overline{M} - \underline{M}$, a sua volta dipendente dai costi sugli archi, che sono *parametri* (e non dimensione) del problema. Tale differenza potrebbe essere molto elevata e, di conseguenza, la convergenza potrebbe essere molto lenta. Si tratta infatti di una complessità computazionale *pseudo-polinomiale*.

[4]Se i costi sono razionali, basta moltiplicarli tutti per il minimo comune multiplo. Se fossero irrazionali, potrebbero esserci problemi numerici di convergenza.

4.3 Correttezza dell'algoritmo label correcting

Dimostriamo adesso la correttezza dell'algoritmo label correcting, facendo ricorso alla teoria della dualità.

Proprietà 4 *L'algoritmo label correcting risolve il problema del cammino minimo da un nodo origine s a un nodo destinazione d.*

Dimostrazione: Alla fine dell'algoritmo label correcting abbiamo a disposizione una soluzione ammissibile duale π e dei puntatori per ogni nodo i, che costruiscono un cammino di costo π_i da s a i. Consideriamo una soluzione del problema primale ottenuta ponendo $x_{ij} = 1$ se (i, j) è un arco del cammino P da s a d ottenuto partendo da d e seguendo a ritroso la catena dei puntatori ai nodi come definiti dall'algoritmo label correcting; $x_{ij} = 0$ per tutti gli altri archi. Le x_{ij} così definite soddisfano i vincoli di bilanciamento dei flussi. Abbiamo quindi a disposizione due soluzioni x e π ammissibili primale e duale, rispettivamente. Inoltre, il valore della funzione obiettivo del problema primale è:

$$\sum_{(i,j)\in A} c_{ij}x_{ij} = \sum_{(i,j)\in P} c_{ij} = \pi_d = \pi_d - 0 = \pi_d - \pi_s$$

Pertanto abbiamo una coppia di soluzioni primale-duale ammissibili con valori della funzione obiettivo coincidenti. Le sue soluzioni sono pertanto ottime (vedi Corollario 1 della teoria della dualità in programmazione lineare) e, in particolare, P è un cammino minimo da s a d. ■

4.4 Albero dei cammini minimi

Abbiamo visto il modello del cammino minimo da un'origine s a una destinazione d

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} \\
\text{s.t.} \quad & \\
& \sum_{(i,v)\in A} x_{iv} - \sum_{(v,j)\in A} x_{vj} = \begin{cases} -1, & v = s; \\ +1, & v = d; \\ 0, & v \in N \setminus \{s, d\}. \end{cases} \\
& x_{ij} \in \mathbb{R}_+ \quad \forall\, (i,j) \in A
\end{aligned}$$

ed il corrispondente duale

$$\begin{aligned}
\max \quad & \pi_d - \pi_s \\
\text{s.t.} \quad & \pi_j - \pi_i \leq c_{ij} \quad \forall\, (i,j) \in A \\
& \pi_v \in \mathbb{R} \quad \forall\, v \in N
\end{aligned}$$

Se consideriamo il problema del cammino mimino dalla stessa origine s alla destinazione d', si ottiene la seguente coppia di modelli del primale

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} \\ \text{s.t.} \quad & \\ & \sum_{(i,v)\in A} x_{iv} - \sum_{(v,j)\in A} x_{vj} = \begin{cases} -1, & v = s; \\ +1, & v = d'; \\ 0, & v \in N \setminus \{s, d'\}. \end{cases} \\ & x_{ij} \in \mathbb{R}_+ \quad \forall\, (i,j) \in A \end{aligned}$$

e del duale

$$\begin{aligned} \max \quad & \pi_{d'} - \pi_s \\ \text{s.t.} \quad & \pi_j - \pi_i \leq c_{ij} \quad \forall\, (i,j) \in A \\ & \pi_v \in \mathbb{R} \quad \forall\, v \in N \end{aligned}$$

In particolare, i vincoli dei due problemi duali sono esattamente gli stessi e, di conseguenza:

> *Lo stesso algoritmo label correcting fornisce delle etichette ammissibili duali, per* tutte *le possibili destinazioni del cammino minimo.*

Inoltre, per il Lemma 1, alla fine dell'algoritmo sono disponibili, attraverso i puntatori $p(\cdot)$, cammini da s verso *tutti* i nodi $j \in N$. Quindi, è possibile partire da d' e costruire (a ritroso), un cammino da s a d', di costo $\pi_{d'}$. Questo cammino, con le stesse motivazioni utilizzate per dimostrare la correttezza dell'algoritmo label correcting, è un cammino minimo da s a d' e $\pi_{d'}$ è la sua lunghezza. Abbiamo pertanto dimostrato che

Proprietà 5 *L'algoritmo label correcting fornisce la lunghezza π_j di un cammino minimo da un'origine s verso* tutti *i nodi $j \in N$ e, attraverso i puntatori $p(\cdot)$, un cammino minimo da s, verso* tutti *gli altri nodi.*

È facile osservare come i puntatori $p(\cdot)$ definiscano una struttura gerarchica tra i nodi e, in particolare, un albero con radice in s (tranne s, ogni nodo ha un solo predecessore-padre). L'unico percorso nell'albero tra s e un nodo j definisce un cammino minimo da s verso j, il cui costo è π_j. Tale albero è detto *albero dei cammini minimi*. Possiamo pertanto affermare che

> Dato un grafo pesato e un nodo origine s, l'algoritmo label correcting permette di costruire l'albero dei cammini minimi con radice in s.

Esempio 3 *Con riferimento al grafo in Figura 2, l'albero dei cammini minimi è raffigurato in Figura 6.*

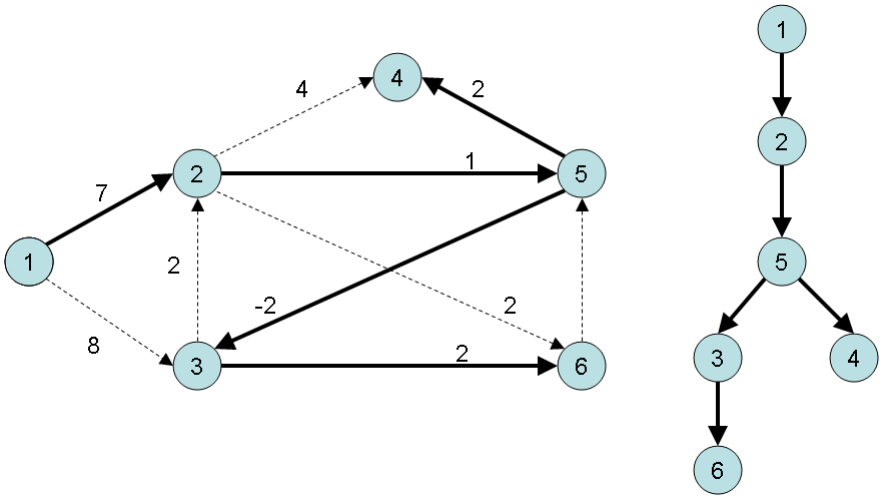

Figure 6: Albero dei cammini minimi per il grafo in Figura 2.

4.5 Grafo dei cammini minimi e cammini minimi alternativi

Attraverso la memorizzazione dei predecessori $p(\cdot)$, l'algoritmo label correcting ci permette di ottenere *uno* dei cammini minimi da s verso gli altri nodi del grafo. In realtà, ricorrendo alla teoria della dualità, è possibile ricavare *tutti* i cammini minimi da s verso un qualsiasi altro nodo.

Dato un grafo $G = (N, A)$ e le etichette ottime π derivate dall'algoritmo label correcting in relazione ad un nodo origine s, si costruisca il grafo $G_{s,\pi} = (N, A_{s,\pi})$ dove $A_{s,\pi} = \{(i, j) \in A : \pi_j = \pi_i + c_{ij}\}$.

Esempio 4 *, sia G il grafo in Figura 7a, dove sono indicate anche le etichette ottime relative all'origine* 1*, determinate dall'algoritmo label correcting. È subito evidente che esistono due cammini minimi equivalenti da* 1 *a* 4*. Il corrispondente grafo $G_{1,\pi}$ è indicato in Figura 7b. Si noti che entrambi i cammini minimi da* 1 *a* 4 *appartengano a $G_{1,\pi}$.*

Se consideriamo un qualsiasi cammino P da s verso un altro nodo $j \in N$ sul grafo $G_{s,\pi}$, possiamo costruire una soluzione primale del problema del cammino minimo da s a j ponendo $x_{ij} = 1$ per gli archi $(i, j) \in P$ e $x_{ij} = 0$ altrimenti. Tale soluzione è ammissibile primale e, inoltre, valgono le condizioni di complementarietà primale duale. Infatti

$$(\pi_j - \pi_i - c_{ij})x_{ij} = 0, \ \forall \ (i, j) \in A$$

visto che $x_{ij} = 1$ solo su archi tali che $(\pi_j - \pi_i - c_{ij}) = 0$, e che, se $(\pi_j - \pi_i - c_{ij}) < 0$, allora $x_{ij} = 0$. Abbiamo quindi a disposizione una coppia di soluzioni ammissibili primale-duale e in scarti complementari. Le due soluzioni sono pertanto ottime e, in particolare:

Un qualsiasi *cammino da s a j su $G_{s,\pi}$ è un cammino minimo da s a j.*

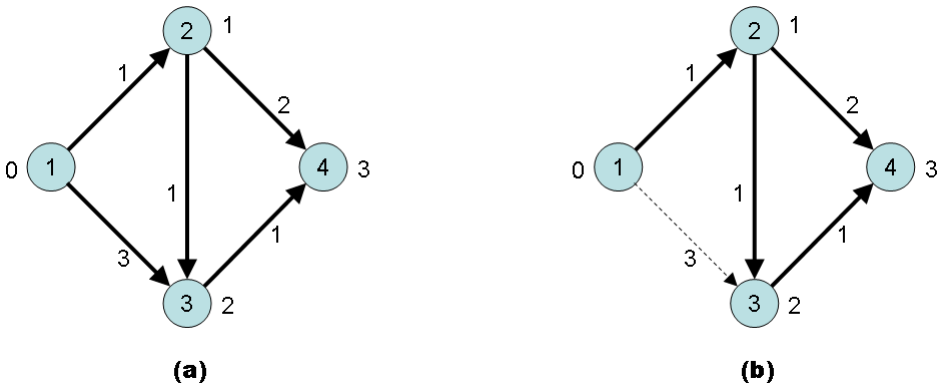

Figure 7: Un esempio di grafo dei cammini minimi.

In effetti, essendo il teorema degli scarti complementari una condizione necessaria e sufficiente, è possibile dimostrare anche il contrario e cioè che se P è un cammino minimo da s a j, allora P è contenuto in $G_{s,\pi}$. $G_{s,\pi}$ è pertanto detto *grafo dei cammini minimi* con origine in s e si ha:

Proprietà 6 *Dato un grafo dei cammini minimi rispetto a un nodo origine s, i cammini su tale grafo da s verso un altro nodo j sono* tutti e soli *i cammini minimi da s a j.*

5 L'algoritmo di Bellman-Ford

L'algoritmo label correcting generico presenta una complessità pseudo-polinomiale. Ciò è dovuto al fatto che non possono essere fatte assunzioni sull'ordine nel quale gli archi vengono controllati e le relative etichette aggiornate. In effetti, dando sistematicità ai controlli e agli aggiornamenti è possibile tenere sotto controllo il numero di iterazioni. Un possibile ordine è dato dal seguente algoritmo.

Algoritmo di *Bellman-Ford*

π_s := 0; set $p(s) = \wedge$;
for each $v \in N - s$ { set π_v:=$+\infty$, set $p(v)$:= $\wedge$ }
for h = 1 to $|N|$ {
    set π' := π; set flag_aggiornato := false;
    for all ($(i,j) \in A : \pi_j > \pi'_i + c_{ij}$) do {
        set π_j := $\pi'_i + c_{ij}$
        set $p(j)$:= i;
        set flag_aggiornato := true;
    }
    if (not flag_aggiornato) then { STOP: π è ottima }
}
STOP: $\exists$ ciclo di costo negativo.

Si fa notare che si tratta di una particolare implementazione dell'algoritmo label correcting generico: si tratta di eseguire i controlli sugli archi e gli aggiornamenti delle etichette in un particolare ordine, e cioè il controllo h-esimo dello stesso arco può essere effettuato solo se tutti gli altri archi sono stati controllati almeno $h-1$ volte. Infatti, ad ogni iterazione con h fissato, vengono controllati *tutti* gli archi. Inoltre, il passo di inizializzazione è lo stesso, così come il criterio di arresto `not flag_aggiornato`. Si noti che, ad ogni iterazione, viene creata una copia delle etichette dell'iterazione precedente (π'): gli aggiornamenti delle etichette correnti (π_j) vengono effettuati sulla base dei valori precedenti (π'_i). Tale operazione non influenza le caratteristiche della soluzione finale ottenuta: la condizione di uscita con etichette ottime rimane la stessa e, in questo caso, $\pi' = \pi$ (l'ultima iterazione lascia inalterate le etichette). Tale accorgimento sarà però utile, come vedremo, nella discussione della convergenza complessiva dell'algoritmo. In ogni caso, le etichette ottenute con l'algoritmo di Bellman-Ford (almeno se si esce con `not flag_aggiornato`) godono delle stesse proprietà viste in precedenza (ottimalità, possibilità di costruire l'albero e il grafo dei cammini minimi etc.).

Esiste tuttavia una nuova condizione di arresto (che permette all'algoritmo di individuare dei cicli negativi) e il numero di iterazioni è limitato dal ciclo `for` esterno. In effetti, tale limite, vedremo, non influenza la possibilità di ottenere le etichette ottime.

5.1 Convergenza, correttezza e complessità

Per dimostrare la convergenza, la complessità e la correttezza dell'algoritmo in generale ci si basa sulla seguente proprietà.

Lemma 2 *Alla fine dell'iterazione con $h = \bar{h}$, π_j rappresenta il costo di un* cammino minimo *da s a j con* al più $\bar{h}$ archi.

La dimostrazione formale, che qui omettiamo, si ottiene per induzione sul numero di iterazioni, ribadendo che, ad ogni iterazione, gli aggiornamenti vengono effettuati sulla base delle etichette dell'iterazione precedente. Ad esempio, quando $h = 1$, tutti gli archi vengono controllati e si aggiornano le etichette di *tutti* i successori di s nel grafo (e solo quelli). Chiaramente, le etichette così ottenute sono relative ai cammini minimi da s verso tutti gli altri nodi che usano esattamente un arco (le etichette rimangono a $+\infty$ per i nodi non direttamente collegati a s). Con $h = 2$ sono aggiornate le etichette dei nodi collegati ai successori dei nodi aggiornati nella prima iterazione (si considerano i π'). Essendo considerati *tutti* i possibili archi, tutti i possibili miglioramenti delle etichette ottenibili con cammini di al più 2 archi sono considerati, e così via.

Abbiamo visto come, in assenza di cicli di costo negativo, un cammino minimo contiene al più $|N| - 1$ archi. Pertanto, per il Lemma 2, l'ultimo aggiornamento di una etichetta avviene al più all'iterazione $h = |N| - 1$ e, con l'iterazione al più $h = |N|$ si verifica semplicemente che nessuna etichetta è cambiata (`flag_aggiornato = false`) e l'algoritmo termina con le etichette ottime. Se invece esistono cicli negativi, all'iterazione $h = N$ si continuano a migliorare le etichette (`flag_aggiornato = true`) e pertanto si uscirà normalmente dal ciclo `for` e si segnalerà la presenza di un ciclo negativo. In questo caso, un ciclo negativo si ottiene percorrendo a ritroso i puntatori a partire da un nodo la cui etichetta è cambiata nell'iterazione $h = |N|$.

L'algoritmo di Bellman-Ford, quindi, converge anche in presenza di cicli di costo negativo e il numero massimo di iterazioni è pari al numero di nodi $|N|$. Ad ogni iterazione, esattamente $|A|$ controlli con eventuali aggiornamenti di π e $p(\cdot)$ sono effettuati in tempo costante, per una complessità computazionale di $O(|N| \cdot |A|)$ che è polinomiale. Abbiamo quindi la seguente proprietà:

Proprietà 7 *Dato un grafo pesato $G = (N, A)$ e un nodo origine $s \in N$, l'algoritmo di Bellman-Ford fornisce la soluzione ottima del problema del cammino minimo dal nodo origine s verso tutti gli altri nodi o individua un ciclo di costo negativo in $O(|N| \cdot |A|)$.*

5.2 Esempi e implementazione efficiente

Esempio 5 *Si consideri il grafo in Figura 8 e si calcoli il costo dei cammini minimi da* 1 *verso tutti gli altri nodi applicando l'algoritmo di Bellman-Ford.*

Per seguire il corretto ordine di controllo degli archi e aggiornamento delle etichette (evitare, ad esempio, di aggiornare le etichette sulla base delle π invece che delle π'),

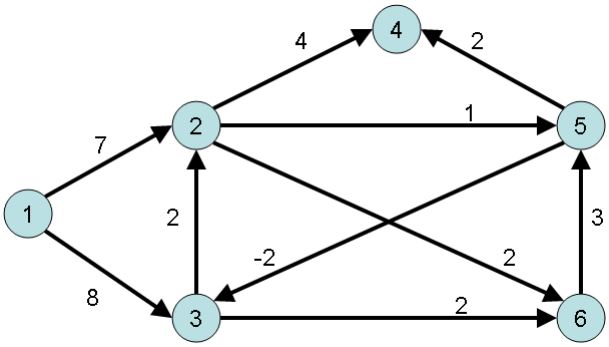

Figure 8: Rete di flusso dell'Esempio 5.

è conveniente organizzare le iterazioni in una tabella. In ogni cella si riportano il valore di π e il nodo predecessore (tra parentesi).

iterazione	nodo 1	nodo 2	nodo 3	nodo 4	nodo 5	nodo 6
$h=0$	$0_{(\wedge)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$ $_{(\wedge)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$
$h=1$	$0_{(\wedge)}$	$+7_{(1)}$	$+8_{(1)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$
$h=2$	$0_{(\wedge)}$	$+7_{(1)}$	$+8_{(1)}$	$+11_{(2)}$	$+8_{(2)}$	$+9_{(2)}$
$h=3$	$0_{(\wedge)}$	$+7_{(1)}$	$+6_{(5)}$	$+10_{(5)}$	$+8_{(2)}$	$+9_{(2)}$
$h=4$	$0_{(\wedge)}$	$+7_{(1)}$	$+6_{(5)}$	$+10_{(5)}$	$+8_{(2)}$	$+8_{(3)}$
$h=5$	$0_{(\wedge)}$	$+7_{(1)}$	$+6_{(5)}$	$+10_{(5)}$	$+8_{(2)}$	$+8_{(3)}$

Attenzione: ad ogni iterazione i controlli e gli aggiornamenti si devono basare sulle etichette dell'iterazione precedente (riga sopra) e non su quelle dell'iterazione corrente (stessa riga). Ad esempio, all'iterazione $h=2$, le etichette dei nodi 4, 5 e 6 restano a $+\infty$, anche se gli archi, ad esempio, $(2,4)$, $(2,5)$ e $(3,6)$ vengono controllati dopo l'arco $(1,2)$, che ha permesso di aggiornare π_2 per l'iterazione corrente, mentre π'_2 è rimasto a $+\infty$. Analogamente, all'iterazione $h=3$, π_6 resta 9.

L'algoritmo termina con la convergenza delle etichette, e pertanto le etichette nell'ultima (e penultima) riga sono ottime (costi dei cammini minimi). Attraverso i puntatori è inoltre possibile costruire l'albero (o il grafo) dei cammini minimi, mostrato in figura 9.

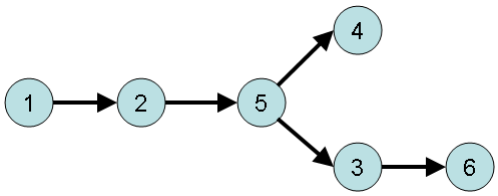

Figure 9: Un albero dei cammini minimi per l'Esempio 5.

Si fa notare come, ad ogni iterazione, sia inutile verificare le condizioni per gli archi uscenti da nodi le cui etichette non vengono aggiornate. Pertanto, è possibile migliorare l'efficienza computazionale media (*ma NON quella del caso peggiore*) andando a memorizzare, ad ogni iterazione, la lista dei nodi le cui etichette vengono aggiornate (Lista `Aggiornati`). All'iterazione successiva basterà controllare gli archi $(i, j) \in A : i \in$ `Aggiornati`.

Esempio 6 *Si risolva il problema dei cammini minimi con origine nel nodo* 1 *per il grafo in Figura 10.*

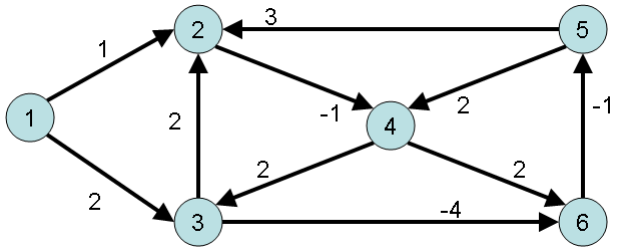

Figure 10: Grafo per l'Esempio 6.

iterazione	nodo 1	nodo 2	nodo 3	nodo 4	nodo 5	nodo 6	`Aggiornati`
$h = 0$	0 $(\wedge)$	$+\infty$ $(\wedge)$	$+\infty$ $(\wedge)$	$+\infty$ $(\wedge)$	$+\infty$ $(\wedge)$	$+\infty$ $(\wedge)$	1
$h = 1$	0 $(\wedge)$	$+1$ (1)	$+2$ (1)	$+\infty$ $(\wedge)$	$+\infty$ $(\wedge)$	$+\infty$ $(\wedge)$	$2, 3$
$h = 2$	0 $(\wedge)$	$+1$ (1)	$+2$ (1)	0 (2)	$+\infty$ $(\wedge)$	-2 (3)	$4, 6$
$h = 3$	0 $(\wedge)$	$+1$ (1)	$+2$ (1)	0 (2)	-3 (6)	-2 (3)	5
$h = 4$	0 $(\wedge)$	0 (5)	$+2$ (1)	-1 (5)	-3 (6)	-2 (3)	$2, 4$
$h = 5$	0 $(\wedge)$	0 (5)	$+1$ (4)	-1 (5)	-3 (6)	-2 (3)	3
$h = 6$	0 $(\wedge)$	0 (5)	$+1$ (4)	-1 (5)	-3 (6)	-3 (3)	6

L'algoritmo ha terminato con `flag_aggiornato = true` e pertanto esiste un ciclo negativo. Tale ciclo è individuato partendo dal nodo 6, (uno dei nodi) che ha subito un aggiornamento all'ultima iterazione, e procedendo a ritroso attraverso i puntatori $p(\cdot)$: $6 \leftarrow 3 \leftarrow 4 \leftarrow 5 \leftarrow 6$ (di costo $-1 + 2 + 2 - 4 = -1$).

5.3 Cammini minimi con massimo numero di archi

Esistono applicazioni di notevole importanza in cui si è interessati ai cammini minimi con un numero limitato di archi (ad esempio, nelle reti di telecomunicazione, per limitare la probabilità di errore di trasmissione). Si parla in questo caso di cammini minimi con un massimo numero di archi (*max-hop*, per le reti di telecomunicazione). Il Lemma 2 ci permette di risolvere tale problema con l'algoritmo di Bellman-Ford: basta fermarsi all'iterazione $h = \bar{h}$, dove $\bar{h}$ è il limite richiesto sul numero di archi.

Esempio 7 *Si calcolino i cammini minimi con al più 3 archi per il grafo in Figura 2.*

Applicando l'algoritmo di Bellman-Ford limitato a 3 iterazioni si ottiene:

iterazione	nodo 1	nodo 2	nodo 3	nodo 4	nodo 5	nodo 6	`Aggiornati`
$h=0$	$0_{(\wedge)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)\ (\wedge)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$	1
$h=1$	$0_{(\wedge)}$	$+7_{(1)}$	$+8_{(1)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$	$2,3$
$h=2$	$0_{(\wedge)}$	$+7_{(1)}$	$+8_{(1)}$	$+11_{(2)}$	$+8_{(2)}$	$+9_{(2)}$	$4,5,6$
$h=3$	$0_{(\wedge)}$	$+7_{(1)}$	$+6_{(5)}$	$+10_{(5)}$	$+8_{(2)}$	$+9_{(2)}$	$3,4$

Si noti che, anche in presenza di cicli negativi, i cammini minimi con un massimo numero di archi sono ben definiti, visto che non è possibile percorrere i cicli negativi un numero di volte illimitato.

Esempio 8 *Si calcolino i cammini minimi con al più 4 archi per il grafo in Figura 10.*

Applicando l'algoritmo di Bellman-Ford limitato a 4 iterazioni si ottiene:

iterazione	nodo 1	nodo 2	nodo 3	nodo 4	nodo 5	nodo 6	`Aggiornati`
$h=0$	$0_{(\wedge)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$	1
$h=1$	$0_{(\wedge)}$	$+1_{(1)}$	$+2_{(1)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$	$2,3$
$h=2$	$0_{(\wedge)}$	$+1_{(1)}$	$+2_{(1)}$	$0_{(2)}$	$+\infty_{(\wedge)}$	$-2_{(3)}$	$4,6$
$h=3$	$0_{(\wedge)}$	$+1_{(1)}$	$+2_{(1)}$	$0_{(2)}$	$-3_{(6)}$	$-2_{(3)}$	5
$h=4$	$0_{(\wedge)}$	$0_{(5)}$	$+2_{(1)}$	$-1_{(5)}$	$-3_{(6)}$	$-2_{(3)}$	$2,4$

Si fa infine notare che, in presenza del massimo numero di archi, è possibile costruire solo l'albero dei cammini minimi, attraverso i puntatori ai predecessori di ogni nodo. Infatti, il grafo dei cammini minimi, anche se costruito sulla base delle etichette ottenute limitando il numero di iterazioni, NON rappresenta tutti e soli i cammini minimi vincolati: alcuni cammini sul grafo potrebbero superare il limite di archi. Ad esempio, le etichette in Figura 11 sono relative a cammini minimi con al massimo due archi (sono ottenibili con due iterazioni dell'algoritmo di Bellman-Ford), mentre il corrispondente grafo dei cammini minimi contiene un cammino minimo $1 \to 2 \to 3 \to 4$ con 3 archi.

5.4 Cammini minimi e principio di sub-ottimalità (nota storica)

La correttezza degli algoritmi label correcting (tra cui l'algoritmo di Bellman-Ford) si basa sul seguente principio di ottimalità, che abbiamo dimostrato ricorrendo alla teoria della dualità in programmazione lineare:

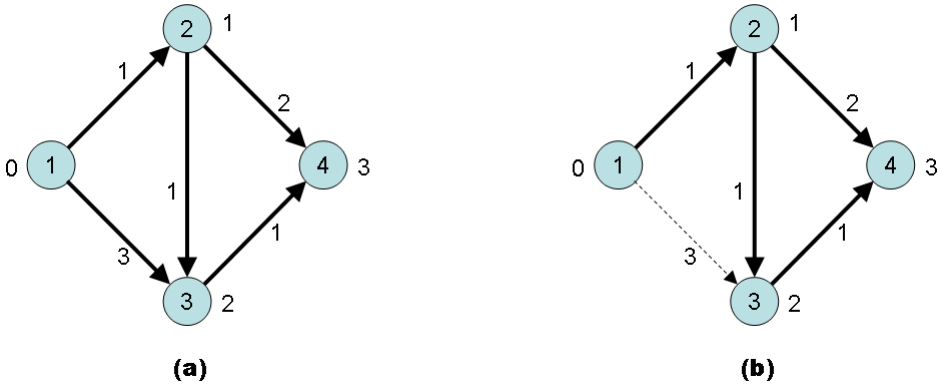

Figure 11: Grafo dei cammini minimi e massimo numero di archi.

Proprietà 8 (Condizioni di ottimalità dei cammini minimi) *Dato un grafo $G = (N, A)$ con costi c sugli archi e un nodo origine $s \in N$, e date delle etichette π_v per ogni nodo $v \in N$, queste rappresentano i costi dei cammini minimi da s verso v* se e solo se *π_v è la lunghezza di un cammino ammissibile da s a v e $\pi_j - \pi_i \leq c_{ij}$, $\forall\, (i, j) \in A$.*

Tali condizioni sono note come *Condizioni di ottimalità di Bellman* e sono le condizioni storicamente sfruttate da Bellman e Ford negli anni '60 per proporre l'algoritmo sopra esposto. Tale algoritmo nasceva come un'applicazione della programmazione dinamica. In effetti, le condizioni di ottimalità possono essere desunte dal seguente principio di sub-ottimalità dei cammini minimi, alla base dell'approccio di programmazione dinamica.

Proprietà 9 (Principio di sub-ottimalità dei cammini minimi) *P è un cammino minimo da s a d se e solo se, per ogni coppia di nodi i e j nel cammino, il sotto-cammino da i a j $P_{ij} \subseteq P$ è un cammino minimo.*

La proprietà è facilmente dimostrabile, considerando che, se P_{ij} non fosse minimo, allora esisterebbe un altro cammino da i a j con costo inferiore. Sostituendo tale cammino a P_{ij} nel cammino P, si otterrebbe un nuovo cammino da s a d migliore di P, contraddicendo l'ottimalità di P.

6 Algoritmo di Dijkstra per il problema del cammino minimo

L'algoritmo di Dijkstra può essere applicato per calcolare i cammini minimi di un grafo $G = (N, A)$ i cui costi soddisfino la seguente condizione:

$$c_{ij} \geq 0 \; \forall \; (i,j) \in A$$

Si tratta di uno schema che calcola delle etichette π ammissibili (e ottime) per il problema duale con una caratteristica che lo differenzia dagli algoritmi *label correcting* sopra esposti e che gli permette di raggiungere una migliore complessità computazionale. Gli algoritmi label correcting, infatti, mantengono delle etichette π che sono fissate come ottime tutte insieme e solo all'ultima iterazione. L'algoritmo di Dijkstra, invece, sotto l'ipotesi di costi non negativi, è in grado di *fissar*e, ad ogni iterazione, il valore ottimo di un'etichetta e, in questo modo, riduce la complessità computazionale per il calcolo del cammino minimo su grafi con costi tutti ≥ 0. Si tratta pertanto di un algoritmo *label setting*, che termina quando viene fissata l'ultima etichetta.

L'algoritmo di Dijkstra determina i valori ottimi delle etichette π_j, $\forall \; j \in N$ mantenendo, ad ogni iterazione, una partizione dei nodi N in due sottoinsiemi S e $\bar{S}$: i nodi in S sono i nodi con etichetta fissata (già ottima) e i nodi in $\bar{S}$ sono quelli la cui etichetta deve essere ancora fissata. Ad ogni iterazione un nodo in $\bar{S}$ viene trasferito in S (*label setting*) e l'algoritmo termina quando $\bar{S} = \emptyset$ (e $S = N$). L'algoritmo procede per aggiornamenti successivi delle π, con conseguente aggiornamento dei consueti puntatori $p(j)$, $\forall \; j \in N$. Nell'algoritmo considereremo un nodo origine s, e indicheremo con $\Gamma_i \subset N$ l'insieme dei nodi successori di un nodo $i \in N$ nel grafo G, ossia $\Gamma_i = \{j \in N : (i,j) \in A\}$.

Algoritmo di *Dijkstra*

0) π_s := 0; set $p(s)$:= $\wedge$; set S := $\{s\}$; set $\bar{S}$:= $S \setminus \{s\}$;
   for each $j \in \Gamma_s$ { set π_j := c_{sj}; set $p(j)$:= s }
   for each $v \in N \setminus \Gamma_s \setminus \{s\}$ { set π_v:=$+\infty$, set $p(v)$:= $\wedge$ }
1) set $\hat{v}$:=$\arg\min_{i \in \bar{S}} \{\pi_i\}$
   set S := $S \cup \{\hat{v}\}$; set $\bar{S}$:= $\bar{S} \setminus \{\hat{v}\}$;
   if $\bar{S} = \emptyset$ then STOP: π è ottimo.
2) for all ($j \in \Gamma_{\hat{v}} \cap \bar{S} : \pi_j > \pi_{\hat{v}} + c_{\hat{v}j}$) do {
     set π_j := $\pi_{\hat{v}} + c_{\hat{v}j}$
     set $p(j)$:= $\hat{v}$;
   }
   go to 1)

Prima di discutere le proprietà dell'algoritmo, vediamo un esempio di applicazione.

Esempio 9 *Si applichi l'algoritmo di Dijkstra per il calcolo dei cammini minimi nel grafo in Figura 12 a partire dal nodo 1.*

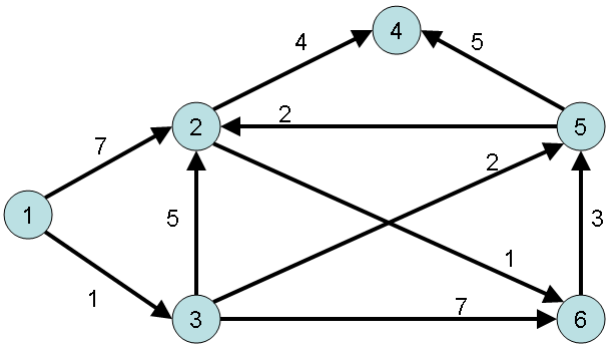

Figure 12: Grafo per l'Esempio 9.

Anche in questo caso possiamo organizzare i calcoli delle varie iterazioni in una tabella.

it.	nodo 1	nodo 2	nodo 3	nodo 4	nodo 5	nodo 6	$\bar{S}$	$\hat{v}$
0	$0^{*}_{(\wedge)}$	$7_{(1)}$	$1_{(1)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$	$+\infty_{(\wedge)}$	$2,3,4,5,6$	
1		$6_{(3)}$	*		$3_{(3)}$	$8_{(3)}$	$2,4,5,6$	3
2		$5_{(5)}$		$8_{(5)}$	*		$2,4,6$	5
3		*		$-$		$6_{(2)}$	$4,6$	2
4					$\times$	*	4	6
5				*			$\emptyset$	4

$*$: etichetta fissata.
$-$: etichetta controllata ma non aggiornata.
$\times$: etichetta non controllata perché il nodo è già fissato.

Dopo l'inizializzazione, ad ogni iterazione:

1) si considerano le etichette correnti e si sceglie quella più piccola; il corrispondente nodo $\hat{v}$ viene estratto da $\bar{S}$ e inserito in S;

2) si procede con le verifiche della condizione di Bellman (vincolo duale) sugli archi che escono da $\hat{v}$ e portano a nodi in $\bar{S}$.

Si noti che, al passo 2), le etichette dei nodi in S non sono più messe in discussione, e risultano pertanto fissate nel momento in cui un nodo viene inserito in S. Ad esempio, al passo 4, l'unico arco uscente da $\hat{v} = 6$ è $(6, 5)$, ma la verifica non viene effettuata, essendo $5 \notin \bar{S}$. Si noti anche che, al passo 3, l'etichetta del nodo 4 viene controllata ($4 \in \Gamma_2$) ma non viene aggiornata ($\pi_4 = 8 < 5 + 4 = \pi_2 + c_{24}$)[5].
L'algoritmo termina quando non ci sono più nodi in $\bar{S}$.

Anche in questo caso, è possibile derivare l'albero (risp. il grafo) dei cammini minimi attraverso i puntatori ai predecessori (risp. la verifica della saturazione dei vincoli duali sugli archi). Nell'esempio, l'albero e il grafo dei cammini minimi coincidono e sono rappresentati in Figura 13.

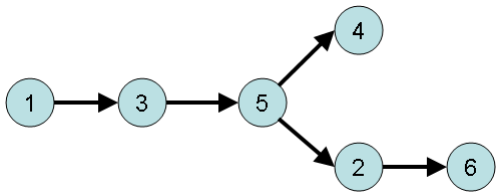

Figure 13: Albero/Grafo dei cammini minimi per l'Esempio 9.

6.1 Convergenza, correttezza e complessità

Per quanto riguarda la convergenza dell'algoritmo di Dijkstra, questa è garantita dal fatto che, ad ogni iterazione, esattamente un nodo lascia l'insieme $\bar{S}$. Saranno pertanto sufficienti $|N|-1 = O(|N|)$ iterazioni per raggiungere la condizione di terminazione $\bar{S} = \emptyset$.

Rispetto ad un algoritmo label correction, l'algoritmo di Dijkstra limita le verifiche della condizione di Bellman (vincoli duali) ai soli archi uscenti dal nodo correntemente selezionato ed entranti in un nodo correntemente nell'insieme $\bar{S}$. Per dimostrare che ciò è lecito, utilizziamo il seguente risultato.

Lemma 3 *Alla fine di ogni iterazione, per ogni nodo $i \in N$:*

- *se $i \in S$, π_i è il costo di un cammino minimo dal nodo origine s a i, $p(i)$ è il predecessore di i su tale cammino, e tale cammino utilizza solo nodi in S;*
- *se $i \in \bar{S}$, π_i è il costo di un cammino minimo dal nodo origine s a i* che utilizza solo nodi in S *per arrivare a i, e $p(i)$ è il predecessore di i su tale cammino.*

[5]Le condizioni sono segnalate in tabella con i simboli $\times$ e $-$ rispettivamente, in modo da verificare facilmente (in fase di svolgimento di un esercizio) che tutti gli archi uscenti da $\hat{v}$ siano stati correttamente presi in considerazione per effettuare o non effettuare la verifica ed eventualmente aggiornare un'etichetta.

Dimostrazione: Utilizziamo l'induzione sul numero di iterazioni. Alla fine della prima iterazione, S è composto dal nodo s (per il quale la verifica della proprietà è immediata) e dal nodo $\hat{v}$, scelto al passo `1)`. Il cammino $s \to \hat{v}$ è un cammino minimo da s a v a costo $c_{s\hat{v}} = \pi_{\hat{v}}$ e non è possibile migliorarlo: ogni altro percorso dovrebbe passare da un terzo nodo v e sarebbe del tipo $s \to \overbrace{v \rightsquigarrow \hat{v}}^{P}$ il cui costo è $c_{sv} + c(P)$. Ma $c_{sv} \geq c_{s\hat{v}}$ (per scelta di $\hat{v}$) e $c(P) \geq 0$, essendo P composto da *archi* di lunghezza ≥ 0. Abbiamo quindi che $\pi_{\hat{v}}$ è il costo di un cammino minimo da s a $\hat{v}$ e che $s = p(\hat{v})$ è il predecessore di $\hat{v}$ su tale cammino, che utilizza solo nodi in S. Per quanto riguarda i nodi in $j \in \bar{S}$, questi sono di 4 tipi (vedi Figura 14):

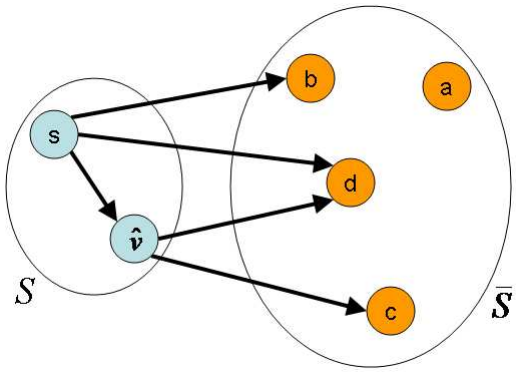

Figure 14: Primo passo induttivo per il Lemma 3.

a) j non è raggiungibile né da s, né da $\hat{v}$: non esistono cammini da s a j con soli nodi in S prima di j; l'etichetta di j non è stata finora aggiornata e rimane $\pi_j = +\infty$ e $p(j) = \wedge$;

b) j è raggiungibile solo da s: l'unico cammino da s a j con soli nodi in S prima di j è $s \to j$ di costo c_{sj}; π_j non viene modificata dall'iterazione e rimane $\pi_j = c_{sj}$ e $p(j) = s$;

c) j è raggiungibile solo da $\hat{v}$: l'unico cammino da s a j con soli nodi in S prima di j è $s \to \hat{v} \to j$ di costo $c_{s\hat{v}} + c_{\hat{v}j}$; l'etichetta π_j era $+\infty$ prima dell'iterazione e diventa $\pi_j = \pi_{\hat{v}} + c_{\hat{v}j}$ con $p(j) = \hat{v}$;

d) j è raggiungibile sia da s che da $\hat{v}$: esistono due cammini da s a j con soli nodi in S prima di j, $P_1 = s \to j$ e $P_2 = s \to \hat{v} \to j$; prima dell'iterazione, π_j e $p(j)$ erano relativi al cammino P_1; dopo l'iterazione, se il costo di P_2 è migliore del costo di P_1, π_j e $p(j)$ vengono modificati di conseguenza.

In ogni caso, l'asserto è verificato per l'iterazione 1.

Consideriamo ora l'iterazione $k+1$ sotto l'ipotesi induttiva che l'asserto sia vero all'iterazione k. Sia $\hat{w}$ il nodo selezionato al passo `1)` dell'iterazione $k+1$ per il passaggio da $\bar{S}$ a S. Osserviamo che, avendo trasferito $\hat{w}$ in S, adesso $\pi_{\hat{w}}$ è il costo di un cammino tutto appartenente ad S (per ipotesi induttiva tutti i restanti nodi del cammino erano già in S). Bisogna dimostrare che $\pi_{\hat{w}}$ è il costo di un cammino *minimo* da s a $\hat{w}$. Se così non fosse, visto che, per ipotesi induttiva, $\pi_{\hat{w}}$ è il costo di un cammino minimo da s a $\hat{w}$ con soli nodi di S prima di w, dovrebbe esistere un cammino P^* da s a $\hat{w}$ che comprende nodi di $\bar{S}$, cioè, con riferimento alla Figura 15, $P^* = P_1 \cup (p(i), i) \cup P_2 \cup P_3$ di costo $c(P_1) + c_{p(i)i} + c(P_2) + c(P_3) < \pi_{\hat{w}}$. Ora, P_1 utilizza solo archi in S prima di i e, per ipotesi induttiva, $c(P_1) + c_{p(i)i} \geq \pi_i$; per come è stato scelto $\hat{w}$, si ha anche $\pi_i \geq \pi_{\hat{w}}$. Inoltre, essendo i cammini P_2 e P_3 composti da *archi* con costo non negativo, $c(P_2) \geq 0$ e $c(P_3) \geq 0$. Quindi $c(P^*) \geq \pi_{\hat{w}}$, che contraddice l'ipotesi su P^*. Pertanto, $\pi_{\hat{w}}$ è l'etichetta ottima di $\hat{w}$.

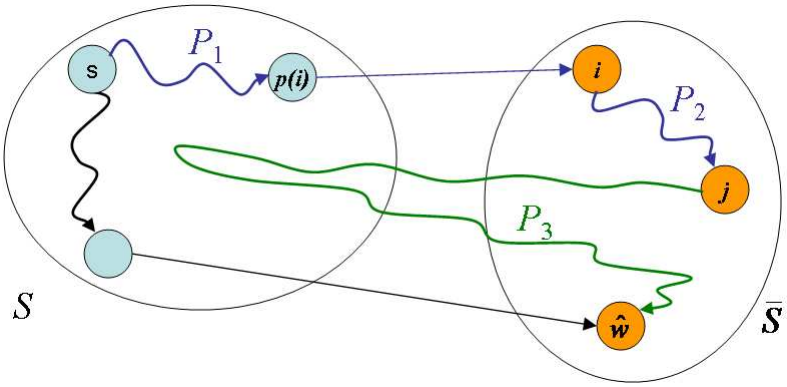

Figure 15: Induzione per il Lemma 3: nodi in S.

Per quanto riguarda i nodi $j \in \bar{S}$, l'inserimento di $\hat{w}$ in S potrebbe portare a ridurre l'etichetta π_j, visto che $\hat{w}$ potrebbe diventare un nodo interno ad un cammino da s a j. Distinguiamo due casi a seconda che j sia o meno successore di $\hat{w}$ (vedi Figura 16). Se $j \notin \Gamma_{\hat{w}}$, allora la sua etichetta potrebbe essere migliorata da un cammino del tipo $P^* = s \rightsquigarrow \hat{w} \rightsquigarrow i \rightarrow j$, con $i \neq \hat{w}$. Osserviamo innanzitutto che, per ipotesi induttiva, non può essere $\pi_j > \pi_i + c_{ij}$, altrimenti avremmo un cammino da s a j con soli nodi in S prima di j a costo inferiore di π_j. Pertanto $\pi_j <= \pi_i + c_{ij}$. Per quanto riguarda il costo di P^*, abbiamo $c(P^*) = c(s \rightsquigarrow \hat{w}) + c(\hat{w} \rightsquigarrow i) + c_{ij}$. Per ipotesi induttiva, π_i è il costo di un cammino minimo da s a i e pertanto, $c(s \rightsquigarrow \hat{w}) + c(\hat{w} \rightsquigarrow i) \geq \pi_i$, da cui $c(P^*) \geq \pi_i + c_{ij} \geq \pi_j$. Quindi, l'etichetta di un nodo $j \notin \Gamma_{\hat{w}}$ non può essere migliorata. Nel caso $j \in \Gamma_{\hat{w}}$, oltre alla possibilità esclusa dalla discussione precedente, potremmo migliorare π_j con un cammino $P^* = s \rightsquigarrow \hat{w} \rightarrow j$, il cui costo è $\pi_{\hat{w}} + c_{\hat{w}j}$. In effetti, il passo `3)` dell'iterazione $k+1$ contempla proprio questa possibilità, aggiornando opportunamente π_j e $p(j)$. ■

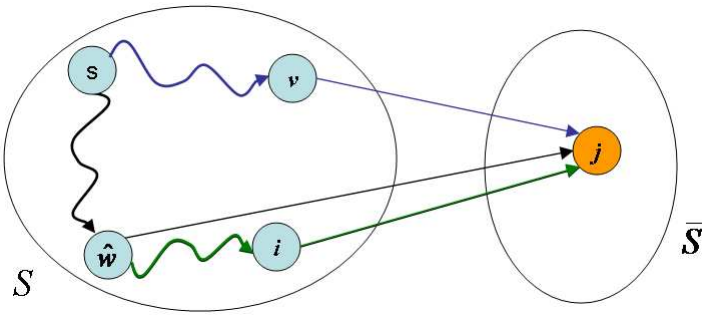

Figure 16: Induzione per il Lemma 3: nodi in $\bar{S}$.

Dal Lemma 3 è immediato ricavare la correttezza dell'algoritmo.

Proprietà 10 *Dato un grafo pesato $G = (N, A)$ e un nodo origine $s \in N$, l'algoritmo di Dijkstra risolve il problema del cammino minimo dall'origine s verso tutti gli altri nodi in tempo $O(|N|^2)$.*

Dimostrazione: L'algoritmo parte con $S = \{s\}$ e trasferisce un nodo da $\bar{S}$ in S ad ogni iterazione. Dopo $|N| - 1$ iterazioni quindi, $\bar{S} = \emptyset$ e l'algoritmo termina. All'ultima iterazione, tutti i nodi sono in S e, pertanto, π_i rappresenta il costo del cammino minimo da s verso i, per tutti i nodi $i \in N$ (e $p(i)$ il relativo predecessore). Inoltre, il passo `0)` comprende $O(N)$ operazioni di inizializzazione effettuabili in tempo costante. Il passo `2)` consiste nella ricerca del minimo in un insieme di al più $|N| - 1$ elementi. La ricerca del minimo si può effettuare scandendo i nodi in $\bar{S}$ ed effettuando $|\bar{S}|$ confronti, quindi in tempo $O(|N|)$. Per quanto riguarda il passo `3)`, consideriamo la sua complessità *ammortizzata*, ossia la complessità cumulativa di tutte le iterazioni. Ad ogni iterazione si considerano gli archi uscenti da un solo nodo. Nel corso delle iterazioni si considerano tutti i nodi (esclusa l'origine, considerata però al passo `0)`) e, di conseguenza, si verificano i vincoli duali su tutti gli archi. L'operazione di verifica ed eventuale aggiornamento di un etichetta e di un puntatore prende tempo costante e pertanto, la complessità ammortizzata del passo `3)` è $O(|A|)$. In totale, l'algoritmo converge alla soluzione ottima in tempo $O(|N| + (|N| - 1)|N| + |A|) = O(|N|^2)$. ■

> Si fa notare che, nella dimostrazione di correttezza, abbiamo utilizzato il Lemma 3, che usa l'ipotesi che *tutti i costi* siano non negativi: non basta che non esistano cicli negativi. **Si ribadisce che, per la corretta applicazione dell'algoritmo di Dijkstra, è necessario che *tutti i costi* siano** ≥ 0.

Ad esempio, si consideri il grafo in Figura 17 dove esiste un costo negativo ma NON esistono cicli di costo negativo. L'applicazione dell'algoritmo di Dijkstra porterebbe al risultato errato di fissare il costo del nodo 3 a 13 alla prima iterazione (mentre il costo corretto, calcolabile con l'algoritmo di Bellman-Ford, è 12).

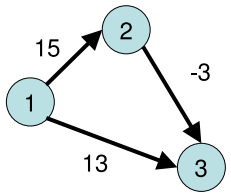

Figure 17: Grafo con costo negativo: non si può applicare Dijkstra.

6.2 Implementazioni efficienti

Abbiamo visto come la complessità dell'algoritmo di Dijkstra dipenda dall'operazione di selezione del minimo nell'insieme $\bar{S}$. Sappiamo che la ricerca del minimo in un insieme di n elementi può essere effettuata in tempi migliori di $O(n)$, al costo di mantenere l'insieme come una struttura dati con proprietà particolari. Per migliorare l'efficienza dell'algoritmo di Dijkstra, quindi, possiamo memorizzare $\bar{S}$ (cioè tutte le etichette π_j per i nodi j da fissare) in una struttura dati a forma di *heap*[6]. Tra le operazioni che possono essere effettuate su un heap, quelle che interessano l'algoritmo di Dijkstra sono le seguenti:

- *create*: crea un heap vuoto, usata in fase di inizializzazione;
- *insert*: aggiunge un elemento allo heap, usata in fase di inizializzazione;
- *find-min*: restituisce l'elemento minimo, usata al passo `2)` per selezionare $\hat{v}$;
- *delete-min*: elimina l'elemento minimo dall'heap, usata al passo `2)` per eliminare da $\bar{S}$ l'elemento $\hat{v}$;
- *decrease-key*: diminuisce il valore di un elemento, usata al passo `3)` ogni volta che si migliora un'etichetta.

Ricordiamo che tutte le operazioni devono preservare le caratteristiche dell'heap stesso (relazione tra gli elementi padre-figli e bilanciamento). La complessità delle operazioni dipende dal modo in cui l'heap è implementato. Ad esempio, con un heap binario (ogni nodo ha due figli), e ricordando che l'heap memorizza al massimo $|N|$ etichette, si ha[7]:

- *create*: complessità $O(1)$;
- *insert*: complessità $O(\log |N|)$;
- *find-min*: complessità $O(1)$ (basta leggere la radice dell'albero);

[6]Ricordiamo che un *d-heap* è una struttura dati a forma di albero in cui ogni nodo padre ha d figli; il valore del nodo padre è non maggiore dei nodi figli; i nodi figli (fratelli) sono ordinati in modo che un fratello abbia valore non maggiore dei fratelli a destra. Un heap è sempre *bilanciato*, in modo tale da minimizzare la profondità dell'albero, che così è $\lfloor \log_d n \rfloor + 1$, se n è il numero di elementi da memorizzare.

[7]Non entriamo nei dettagli, si rimanda alle nozioni di algoritmi e strutture dati.

- *delete-min*: complessità $O(\log |N|)$ (una volta eliminata la radice, bisogna selezionare la nuova radice e percorrere l'albero per un numero di passi pari alla profondità dell'albero stesso per ristabilire le proprietà dell'heap);
- *decrease-key*: complessità $O(\log |N|)$ (bisogna ricercare un elemento e ristabilire le proprietà dell'heap, con un numero di passi che dipendono dalla profondità).

Pertanto, al passo `0)` vengono effettuate $O(|N|)$ operazioni di inizializzazione, un'operazione di creazione dell'heap vuoto ($O(1)$) e $|N|-1$ operazioni di inserimento ($O((|N|-1)\log|N|)$); al passo `2)`, un'operazione di ricerca (trova $\hat{v}$) e un'operazione di eliminazione dell'elemento minimo (trasferisci $\hat{v}$ da $\bar{S}$ a S) sono ripetute per $|N|-1$ volte ($O(2(|N|-1)\log|N|)$); al passo `3)`, ammortizzato, si effettuano $|A|$ confronti e al massimo $|A|$ operazioni di diminuzione di un elemento e di aggiornamento del predecessore ($O(|A|(1+\log|N|+1))$. Quindi, in tutto, la complessità dell'algoritmo di Dijkstra implementato con heap binomiale è $O(|A|\log|N|)$.

Citiamo inoltre il fatto che l'implementazione più efficiente dell'algoritmo di Dijkstra fa uso di *heap di Fibonacci* e raggiunge una complessità di $O(|A|+|N|\log|N|)$.