Ricerca Operativa
Modulo 2
Teoria dei Grafi: Parte 2

Marco A. Boschetti

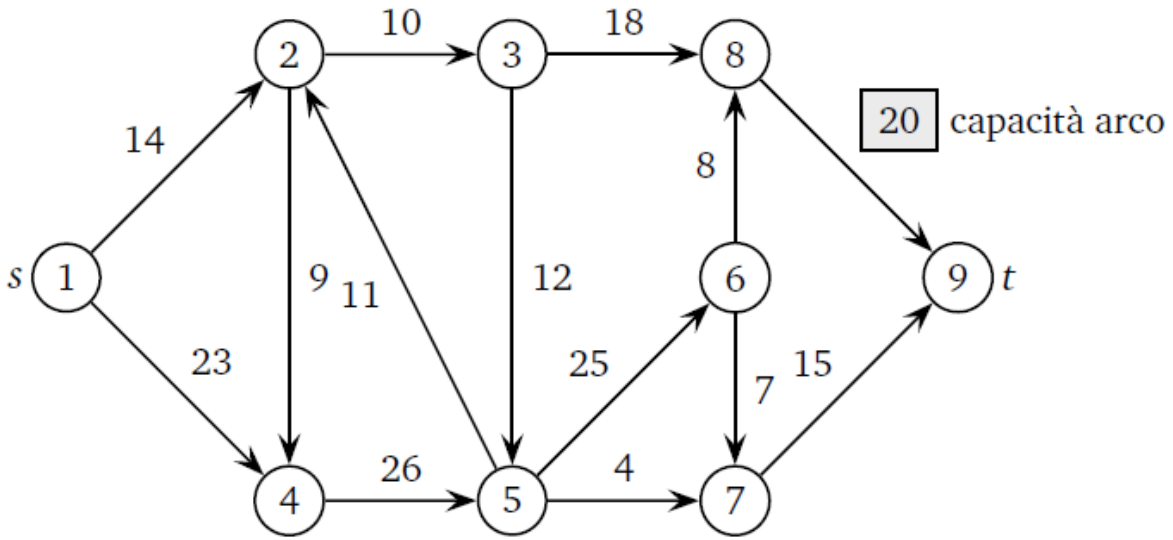
Università degli Studi di Bologna
Dipartimento di Matematica
marco.boschetti@unibo.it

Outline

Outline (2)

Flusso Massimo: Definizione

- Sia dato un grafo direzionato $G = (N, A)$, in cui ad ogni arco $(i, j) \in A$ è associata una capacità $u_{ij} \geq 0$.
- Si vuole determinare il flusso massimo che può essere inviato dal vertice origine $s \in N$ al vertice destinazione $t \in N$.
- Nell'esempio è riportata la capacità u_{ij} per ogni arco $(i, j) \in A$. Inoltre, l'origine è $s = 1$ e la destinazione è $t = 9$.



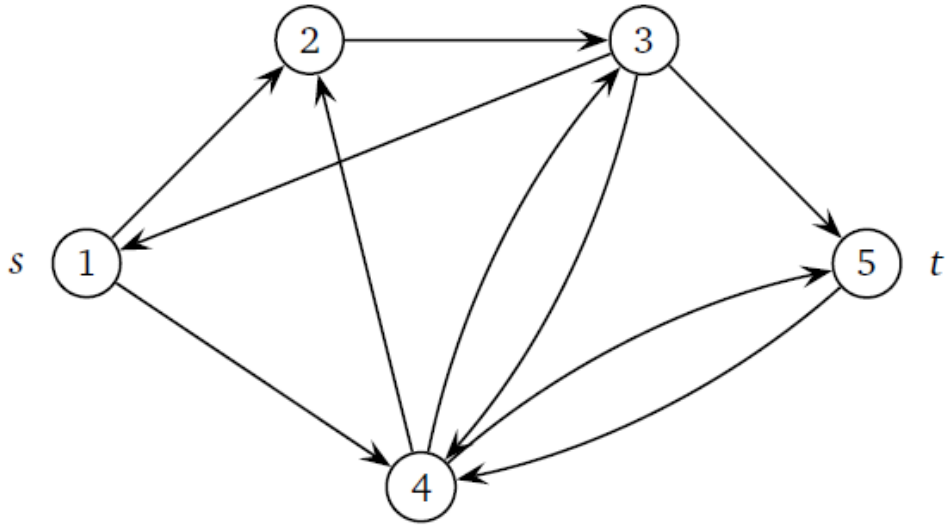
Flusso Massimo: Formulazione Matematica

- Per formulare il problema del flusso massimo sono necessarie le seguenti variabili decisionali:
  - x_{ij}: quantità di flusso assegnata a ciascun arco $(i,j) \in A$;
  - v: flusso introdotto all'origine s e ricevuto alla destinazione t.
- Il problema del flusso massimo può essere formulato come segue:

$$
\begin{aligned}
Max \;\; z(P) = \;\; & v \\
s.t. \;\; & \sum_{j \in \Gamma_i} x_{ij} - \sum_{j \in \Gamma_i^{-1}} x_{ji} = \begin{cases} v, & i = s \\ -v, & i = t \\ 0, & i \neq s,t \end{cases} && i \in N \\
& 0 \leq x_{ij} \leq u_{ij}, && (i,j) \in A
\end{aligned}
$$

dove $\Gamma_i = \{j : (i,j) \in A\}$ e $\Gamma_i^{-1} = \{j : (j,i) \in A\}$.

Flusso Massimo: Formulazione Matematica



Vertice	x_{12}	x_{14}	x_{23}	x_{31}	x_{34}	x_{35}	x_{42}	x_{43}	x_{45}	x_{54}		v
$s=1$	x_{12}	$+ x_{14}$		$- x_{31}$							$=$	v
2	$- x_{12}$		$+ x_{23}$				$- x_{42}$				$=$	0
3			$- x_{23}$	$+ x_{31}$	$+ x_{34}$	$+ x_{35}$		$- x_{43}$			$=$	0
4		$- x_{14}$			$- x_{34}$		$+ x_{42}$	$+ x_{43}$	$+ x_{45}$	$- x_{54}$	$=$	0
$t=5$						$- x_{35}$			$- x_{45}$	$+ x_{54}$	$=$	$-v$

Assunzioni e Definizioni

Assunzioni

- Il grafo $G(N,A)$ è orientato;
- Tutte le capacità u_{ij} sono intere non negative (i.e., $u_{ij} \geq 0$);
- Il grafo non contiene un cammino orientato dal vertice s al vertice t composto da soli archi di capacità infinita.
- Quando un arco $(i,j) \in A$ allora anche $(j,i) \in A$ (si noti che uno dei due potrebbe avere capacità nulla).

Parametri

- $n = |N|$ è il numero di nodi del grafo G.
- $m = |A|$ è il numero di archi del grafo G.
- $U = max\{u_{ij} : (i,j) \in A\}$.

Assunzioni e Definizioni (2)

Grafo Residuo

- Il grafo residuo $G(x)$ corrispondente al flusso x nel grafo G è ottenuto sostituendo ogni arco $(i,j) \in A$ con un arco "*diretto*" (i,j) e un arco "*inverso*" (j,i) con una *capacità residua* pari a $r_{ij} = u_{ij} - x_{ij}$ e $r_{ji} = x_{ij}$.

Cammino aumentante

- Dato un flusso x nel grafo G, un *cammino aumentante* è un cammino da s a t nel grafo residuo $G(x)$.

Cammino aumentante (definizione equivalente)

- Dato un flusso x nel grafo G, un *cammino aumentante* è un cammino non orientato da s a t nel grafo G, in cui si può aumentare il flusso di almeno una unità negli archi percorsi nella direzione originaria e diminuire il flusso di almeno una unità negli archi percorsi nella direzione contraria rispetto a quella originaria.

Taglio *s-t*

Taglio *s-t*

Si definisce *Taglio s-t* una partizione dei vertici N del grafo G in due sottoinsiemi S e $\bar{S} = N \setminus S$ tali che $s \in S$ e $t \in \bar{S}$.

Capacità di un Taglio *s-t*

La capacità di un taglio *s-t* è data da:

$$u(S,\bar{S}) = \sum_{i \in S} \sum_{j \in \bar{S}} u_{ij} = \sum_{(i,j) \in \Gamma(S,\bar{S})} u_{ij}$$

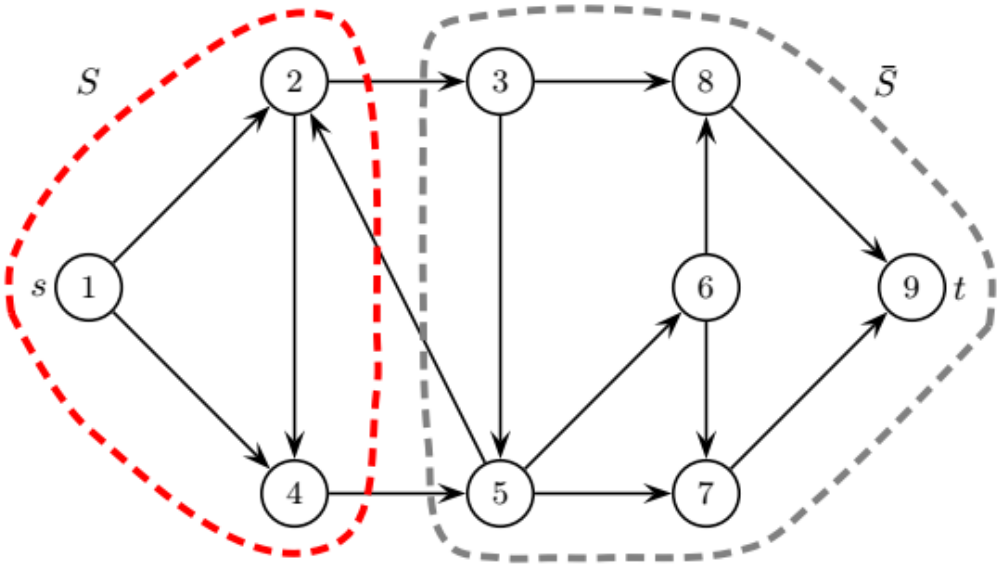
dove $\Gamma(S,\bar{S}) = \{(i,j) \in A : i \in S, j \in \bar{S}\}$.

Si noti che il flusso che attraversa gli archi del taglio appartenenti all'insieme $\Gamma(\bar{S},S) = \{(i,j) \in A : i \in \bar{S}, i \in S\}$ *riduce* il flusso da s a t.

Taglio s-t (2)

Esempio di Taglio s-t

Nell'esempio la partizione è $S = \{1, 2, 4\}$ e $\bar{S} = \{3, 5, 6, 7, 8, 9\}$:



Nell'esempio $\Gamma(S, \bar{S}) = \{(2, 3), (4, 5)\}$ e $\Gamma(\bar{S}, S) = \{(5, 2)\}$.

Condizioni di Ottimalità

Teorema

Dato un flusso ammissibile x pari a v, per ogni taglio s-t $(S, \bar{S})$ si ha:

$$v = \sum_{(i,j)\in\Gamma(S,\bar{S})} x_{ij} - \sum_{(i,j)\in\Gamma(\bar{S},S)} x_{ij} \quad \textit{(flusso attraverso il taglio)}$$

Dimostrazione.

Sommando i vincoli di conservazione del flusso per tutti i nodi di S:

$$v = \sum_{h\in S}\left[\sum_{j\in\Gamma_h} x_{hj} - \sum_{j\in\Gamma_h^{-1}} x_{jh}\right] = \sum_{h\in S}\sum_{j\in\Gamma_h} x_{hj} - \sum_{h\in S}\sum_{j\in\Gamma_h^{-1}} x_{jh}$$

$$= \left[\sum_{(i,j)\in A(S)} x_{ij} + \sum_{(i,j)\in\Gamma(S,\bar{S})} x_{ij}\right] - \left[\sum_{(i,j)\in A(S)} x_{ij} + \sum_{(i,j)\in\Gamma(\bar{S},S)} x_{ij}\right] = \sum_{(i,j)\in\Gamma(S,\bar{S})} x_{ij} - \sum_{(i,j)\in\Gamma(\bar{S},S)} x_{ij}$$

dove $A(S) = \{(i,j) \in A : i \in S, j \in S\}$. □

Condizioni di Ottimalità (2)

Teorema

Il valore v di ogni flusso ammissibile x in G è minore o uguale alla capacità $u(S,\bar{S})$ di un qualunque taglio s-t.

Dimostrazione.

Per il Teorema precedente il valore v del flusso x è dato da:

$$v = \sum_{(i,j)\in\Gamma(S,\bar{S})} x_{ij} - \sum_{(i,j)\in\Gamma(\bar{S},S)} x_{ij}$$

Un limite superiore al valore di v si ottiene sostituendo x_{ij} con u_{ij}, per ogni $(i,j) \in \Gamma(S,\bar{S})$, e x_{ij} con 0, per ogni $(i,j) \in \Gamma(\bar{S},S)$. Per cui:

$$v \leq \sum_{(i,j)\in\Gamma(S,\bar{S})} u_{ij} = u(S,\bar{S})$$

In particolare, anche il valore del flusso massimo è minore o uguale alla capacità del taglio s-t di capacità minima. □

Condizioni di Ottimalità (3)

Teorema (Augmentig Path Theorem)

Sia x un flusso ammissibile in G da s a t e sia $G(x)$ il corrispondente grafo residuo. Il flusso x è massimo se e solo se non esiste in $G(x)$ un cammino da s a t (i.e., non esiste un cammino aumentante da s a t in G).

Dimostrazione: Parte (a).

- Se il flusso x è massimo, allora non esiste un cammino P in $G(x)$.
  Se per assurdo esistesse un cammino aumentante P in $G(x)$ tale che $\delta = \min\{r_{ij} : \forall (i,j) \in P\} > 0$, allora sarebbe possibile aggiornare il flusso per ogni arco $(i,j) \in P$:
  - $x_{ij} = x_{ij} + \delta$, se (i,j) è un arco diretto;
  - $x_{ji} = x_{ji} - \delta$, se (i,j) è un arco inverso;

  Il nuovo flusso aggiornato x è un flusso ammissibile di valore $v + \delta$, per cui il flusso x prima dell'aggiornamento non era ottimo. □

Condizioni di Ottimalità (4)

Dimostrazione: Parte (b).

- Se non esiste un cammino P in $G(x)$, allora il flusso x è massimo.
  Se non esiste un cammino P in $G(x)$, allora esiste un taglio $(S, \bar{S})$, dove $\bar{S} = V \setminus S$, nel grafo residuo $G(x)$ tale che $\Gamma(S, \bar{S}) = \emptyset$. Nella rete originale si ha allora che:
  - ogni arco $(i,j) \in \Gamma(S, \bar{S})$ è saturo (i.e., $x_{ij} = u_{ij}$);
  - ogni arco $(i,j) \in \Gamma(\bar{S}, S)$ è scarico (i.e., $x_{ij} = 0$).

  Quindi, il valore del flusso per il taglio $(S, \bar{S})$ è uguale a:

$$v = \sum_{(i,j)\in\Gamma(S,\bar{S})} x_{ij} - \sum_{(i,j)\in\Gamma(\bar{S},S)} x_{ij} = \sum_{(i,j)\in\Gamma(S,\bar{S})} u_{ij} = u(S, \bar{S})$$

  e dal teorema precedente si ha che il flusso x è ottimo e $(S, \bar{S})$ è il taglio s-t di capacità minima. □

Condizioni di Ottimalità (5)

Un ulteriore importante teorema che ne deriva è il seguente:

Teorema (Max-Flow Min-Cut Theorem)

Il flusso massimo da s a t in G è uguale alla capacità del taglio s-t di capacità minima.

Inoltre, una importante proprietà del problema del flusso massimo è definita dal seguente teorema.

Teorema (Integrality Theorem)

Se tutti gli archi hanno una capacità u_{ij} intera, allora il problema del flusso massimo ha una soluzione intera.

Un algoritmo per risolvere il problema del flusso massimo è l'Algoritmo di Ford-Fulkerson che è basato sul *Augmenting Path Theorem*.

Algoritmo di Ford-Fulkerson (versione 1)

Step 1. [Inizializza]

Poni $x_{ij} = 0$, per ogni $(i,j) \in A$, e $v = 0$.

Step 2. [Determina Cammino Aumentante]

Costruisci il grafo residuo $G(x)$.

Trova un cammino da s a t in $G(x)$.

Se non esiste un cammino da s a t, allora il flusso x è massimo, quindi STOP.

Step 3. [Aumenta il Flusso x]

Sia P il cammino da s a t trovato e poni $\delta = \min\{r_{ij} : (i,j) \in P\}$.

Per ogni $(i,j) \in P$:

(i) Se (i,j) corrisponde all'arco $(i,j) \in A$: $x_{ij} = x_{ij} + \delta$;

(ii) Se (i,j) corrisponde all'arco $(j,i) \in A$: $x_{ji} = x_{ji} - \delta$;

Aggiorna il flusso $v = v + \delta$.

Ritorna allo Step 2.

Algoritmo di Ford-Fulkerson (versione 2)

- L'algoritmo di Ford-Fulkerson può essere riformulato senza usare il grafo residuo $G(x)$ esplicitamente.
- Per consentire il calcolo del cammino aumentante nel grafo residuo senza doverlo generare si possono usare delle etichette $[pred[j], \delta_j]$ per ciascun vertice $j \in N$.
- $pred[j]$ è il "predecessore" del vertice j nel cammino aumentante:
  - Se $pred[j] = i > 0$, allora nell'arco $(i,j) \in A$ il flusso potrebbe essere aumentato di δ_j.
  - Se $pred[j] = -i < 0$, allora nell'arco $(j,i) \in A$ il flusso potrebbe essere diminuito di δ_j.
- Ciascun δ_j indica il massimo aumento consentito nel cammino da s fino a j. Mentre, l'effettivo aumento del flusso nel cammino da s a t è determinato da δ_t.

Algoritmo di Ford-Fulkerson (versione 2)

Step 1. [Inizializza]
Poni $x_{ij} = 0$, per ogni $(i,j) \in A$, e $v = 0$.

Step 2. [Determina Cammino Aumentante]
Dichiara il vertice s non espanso con etichetta $[+s, \infty]$, mentre dichiara tutti gli altri vertici non etichettati (e non espansi).

(i) Se tutti i nodi etichettati sono già espansi, allora il flusso v è massimo (i.e, la soluzione x è ottima), quindi STOP.

(ii) Se esiste un vertice i etichettato ma non ancora espanso, provvedi a espanderlo:

- Per ogni vertice $j \in \Gamma_i$ non etichettato per cui $x_{ij} < u_{ij}$ assegna l'etichetta $[+i, \delta_j]$, dove $\delta_j = \min\{u_{ij} - x_{ij}, \delta_i\}$;
- Per ogni vertice $j \in \Gamma_i^{-1}$ non etichettato per cui $x_{ji} > 0$ assegna l'etichetta $[-i, \delta_j]$, dove $\delta_j = \min\{x_{ji}, \delta_i\}$.

(iii) Se t non risulta etichettato torna a passo (i).

Algoritmo di Ford-Fulkerson (versione 2)

Step 3. [Aumenta il Flusso x]

Sia P il cammino aumentante dal vertice s al vertice t (ricostruito usando le etichette).

Per ogni $(i,j) \in P$:

(i) Se l'arco (i,j) è percorso nel suo verso originario (da i a j):
$x_{ij} = x_{ij} + \delta_t$;

(ii) Se l'arco (i,j) è percorso nel verso contrario (da j a i):
$x_{ij} = x_{ij} - \delta_t$.

Aggiorna il flusso $v = v + \delta_t$.

Annulla tutte etichette e ritorna allo Step 2.

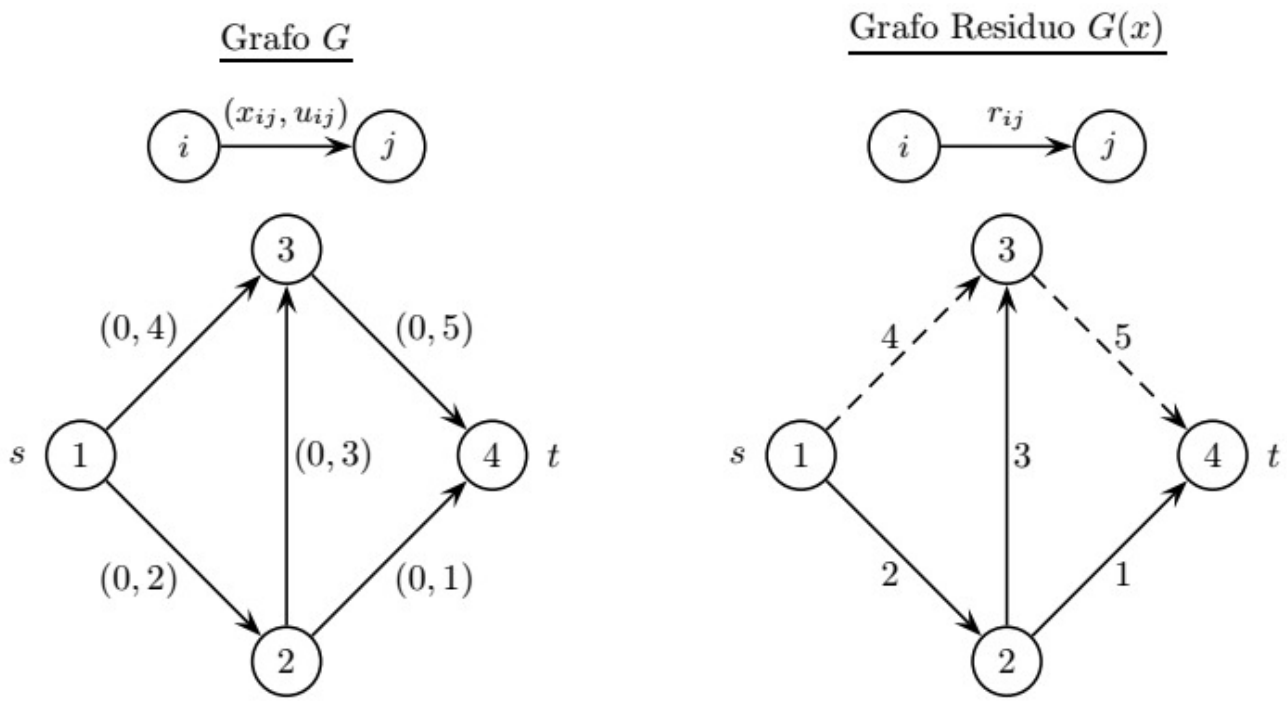
NOTA: Quando l'Algoritmo di Ford-Fulkerson termina, l'insieme dei vertici etichettati S e quello dei vertici non etichettati $\bar{S}$ costituiscono un *taglio di capacità minima*.

Algoritmo di Ford-Fulkerson: Complessità

- L'algoritmo di Ford-Fulkerson ha complessità $O(nmU)$, perché:
  - ad ogni iterazione il flusso aumenta di $\delta \geq 1$ e il valore del flusso è limitato superiormente da nU;
  - il calcolo di un cammino aumentante ha complessità $O(m)$.
- Edmonds e Karp (1972) hanno proposto alcune varianti dell'algoritmo di Ford-Fulkerson:
  - calcolando il cammino aumentante di capacità massima la complessità è pari a $O(nm \log U)$;
  - calcolando il cammino aumentante di cadinalità minima la complessità è pari a $O(n^2 m)$.
- Esistono altri algoritmi con "minore" complessità, per esempio:
  - FIFO preflow-push con complessità $O(n^3)$;
  - Highest-label preflow push con complessità $O(n^2\sqrt{m})$;
  - Excess scaling con complessità $O(nm + n^2 \log U)$.

Algoritmo di Ford-Fulkerson: Esempio 1

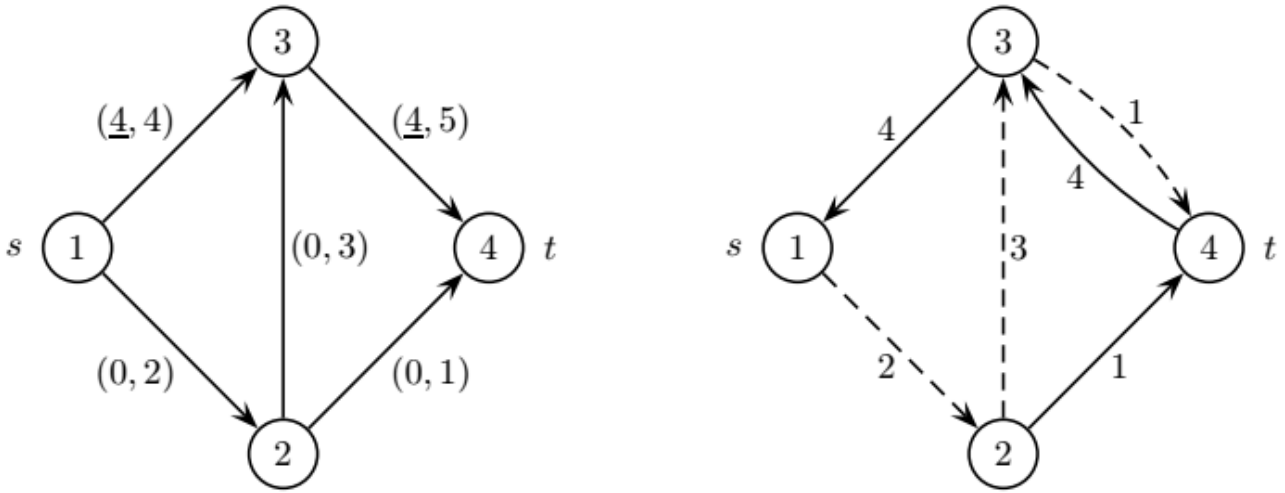
Risolviamo usando il grafo residuo impostando la soluzione e il flusso iniziale a $x = 0$ e $v = 0$:



Il flusso può essere aumentato di $\delta = \min\{r_{13}, r_{34}\} = 4$ unità sul cammino aumentante $P = (1, 3, 4)$ e $v = v + \delta = 0 + 4 = 4$.

Algoritmo di Ford-Fulkerson: Esempio 1 (2)

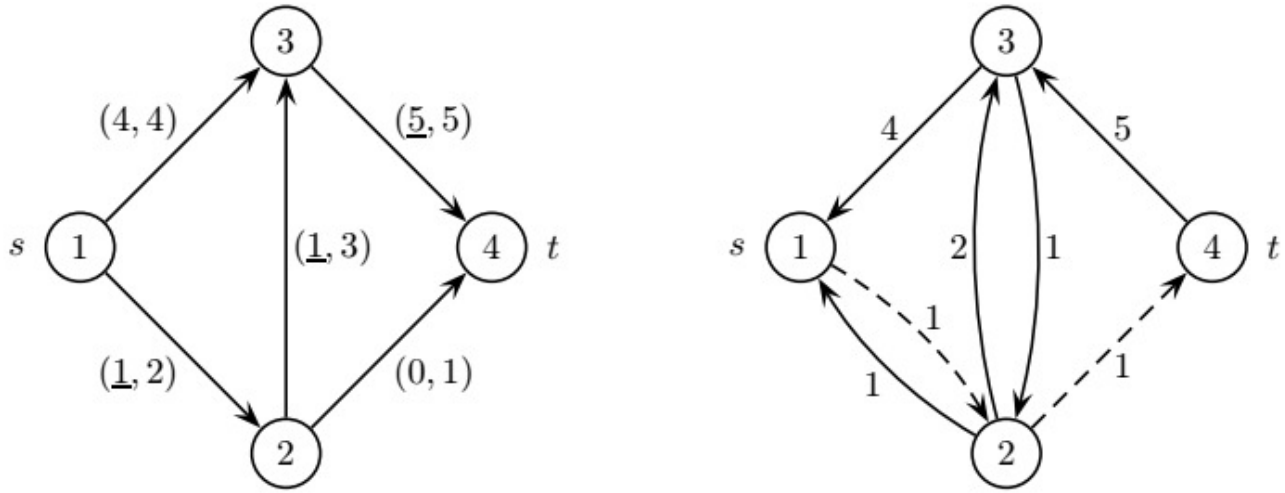
Si aggiorna il grafo residuo $G(x)$ e si cerca un nuovo cammino aumentante:



Il flusso può essere aumentato di $\delta = \min\{r_{12}, r_{23}, r_{34}\} = 1$ unità sul cammino aumentante $P = (1, 2, 3, 4)$ e $v = v + \delta = 4 + 1 = 5$.

Algoritmo di Ford-Fulkerson: Esempio 1 (3)

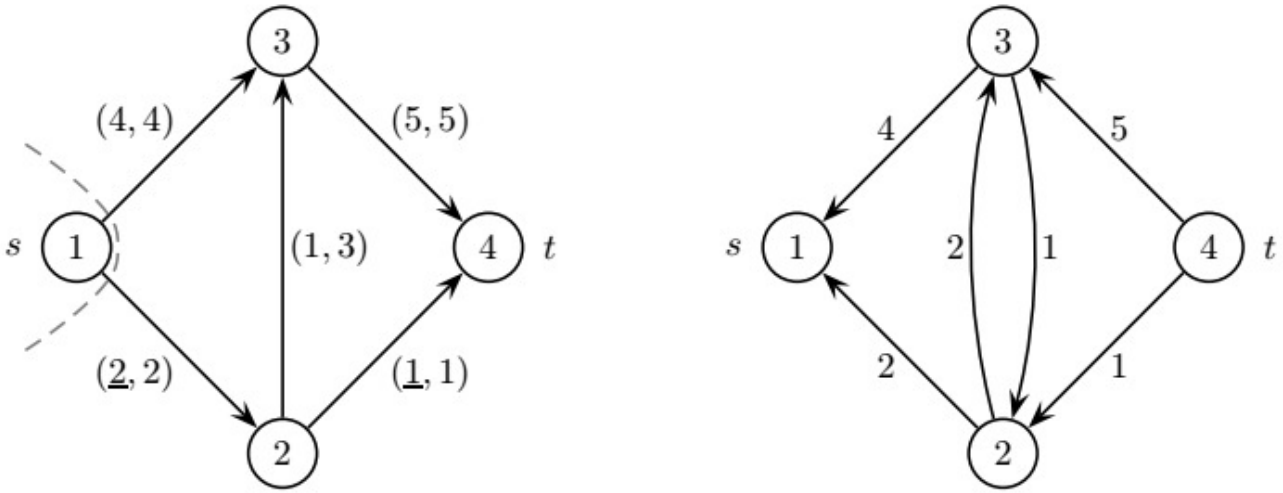
Si aggiorna nuovamente il grafo residuo $G(x)$ e si cerca un nuovo cammino aumentante:



Il flusso può essere aumentato di $\delta = \min\{r_{12}, r_{24}\} = 1$ unità sul cammino aumentante $P = (1, 2, 4)$ e $v = c + \delta = 5 + 1 = 6$.

Algoritmo di Ford-Fulkerson: Esempio 1 (4)

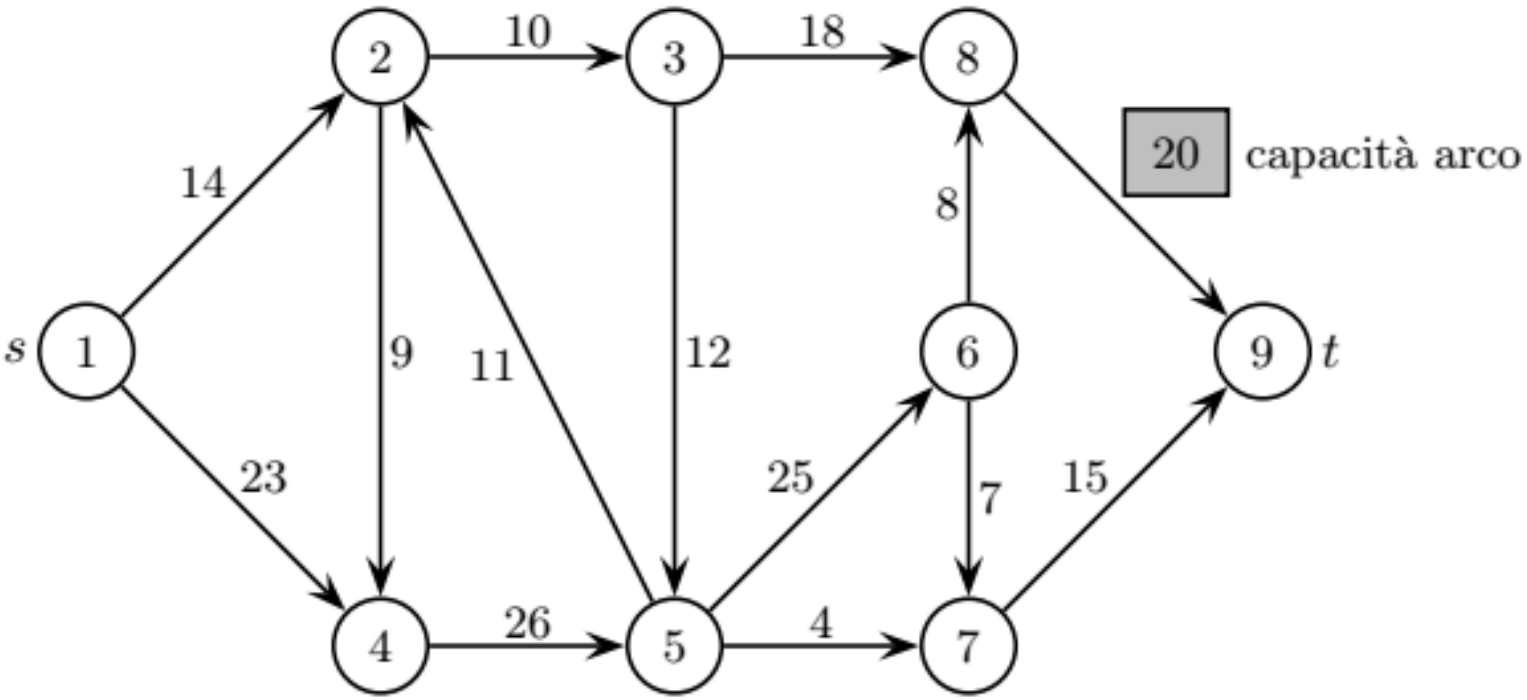
Si aggiorna un'ultima volta il grafo residuo $G(x)$:



- Il grafo residuo non contiene alcun cammino aumentante.
- Il flusso massimo è $v = 6$.
- Il taglio di capacità minima è definito da $S = \{1\}$ e $\bar{S} = \{2, 3, 4\}$; inoltre, si può notare che $v = \sum_{i \in S} \sum_{j \in \bar{S}} u_{ij} = 6$.

Algoritmo di Ford-Fulkerson: Esempio 2

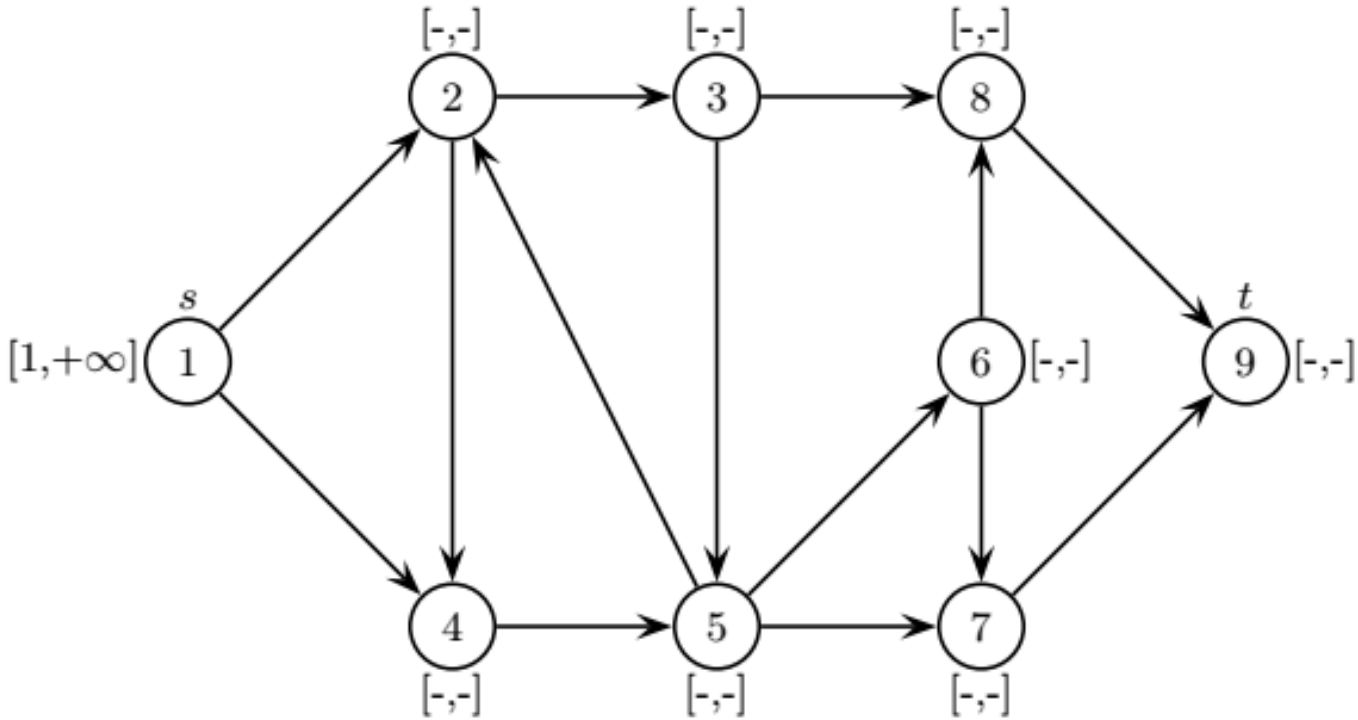
Consideriamo il seguente grafo:



Risolviamo usando la versione con le etichette impostando la soluzione e il flusso iniziale a $x = 0$ e $v = 0$.

Algoritmo di Ford-Fulkerson: Esempio 2 (2)

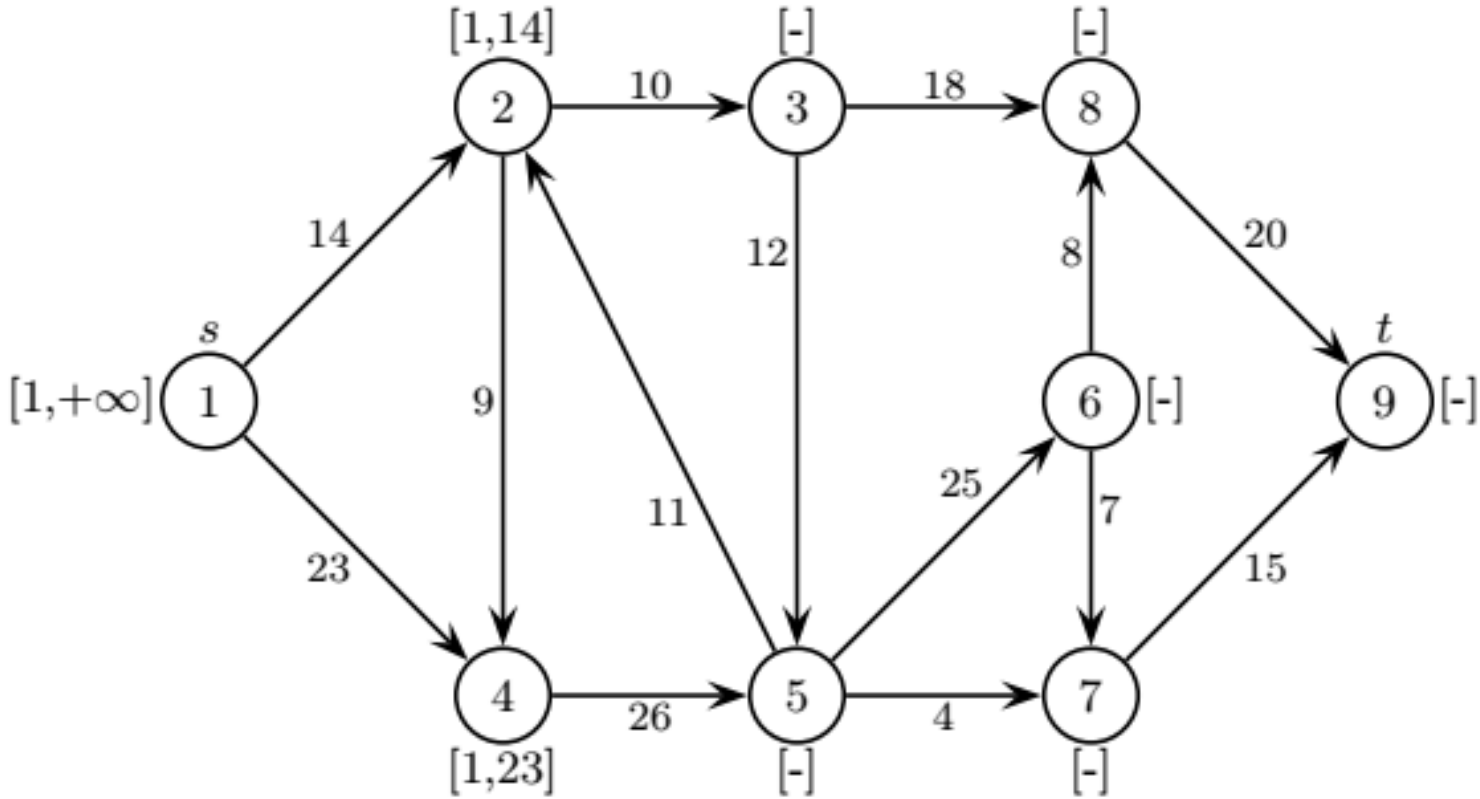
1^a iterazione (di cui vediamo tutti i passaggi):



Viene etichettato il nodo sorgente $s = 1$, che ora deve essere espanso.

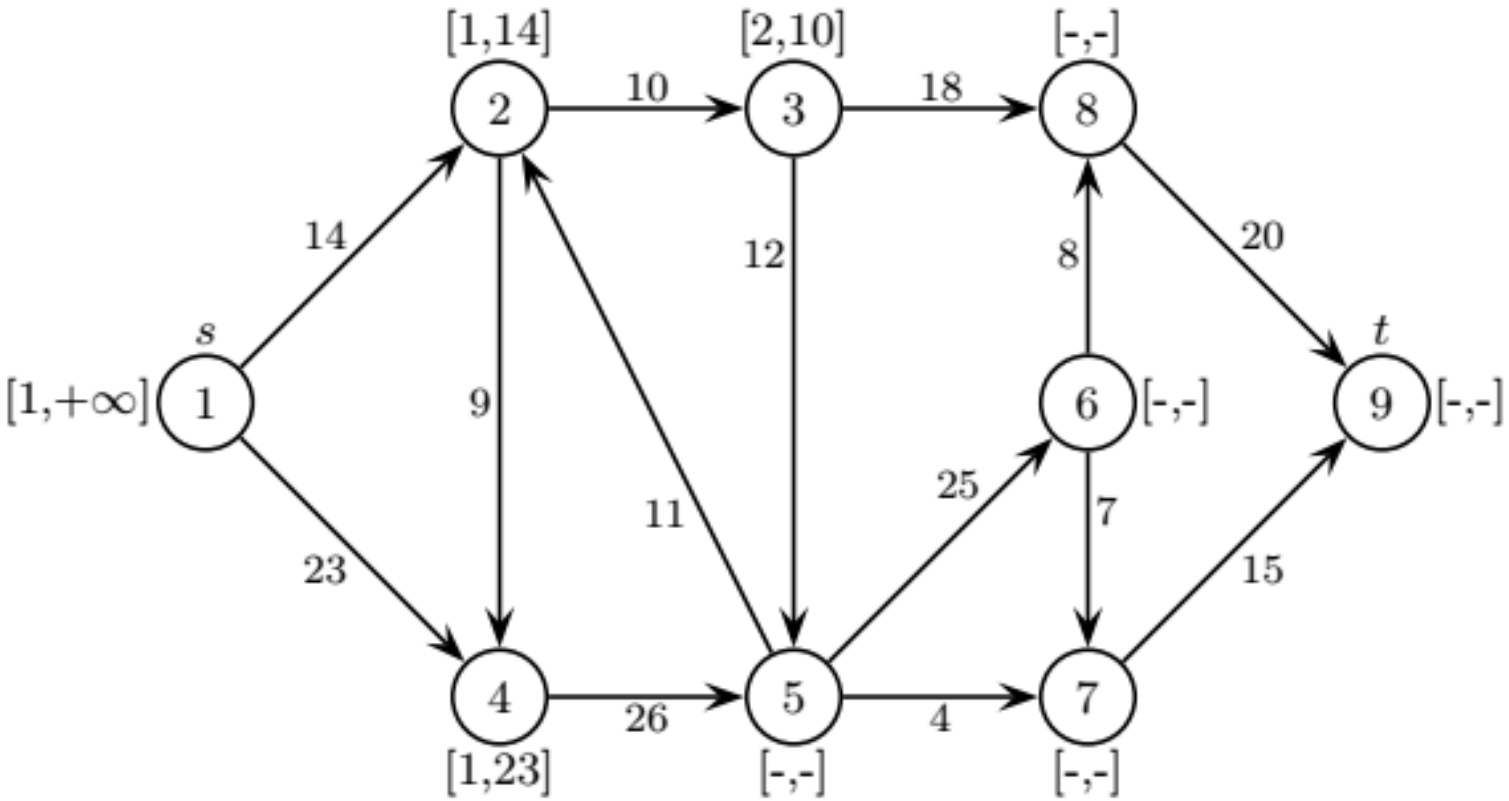
Algoritmo di Ford-Fulkerson: Esempio 2 (3)

Il nodo $s = 1$ viene espanso considerando i nodi adiacenti 2 e 4:



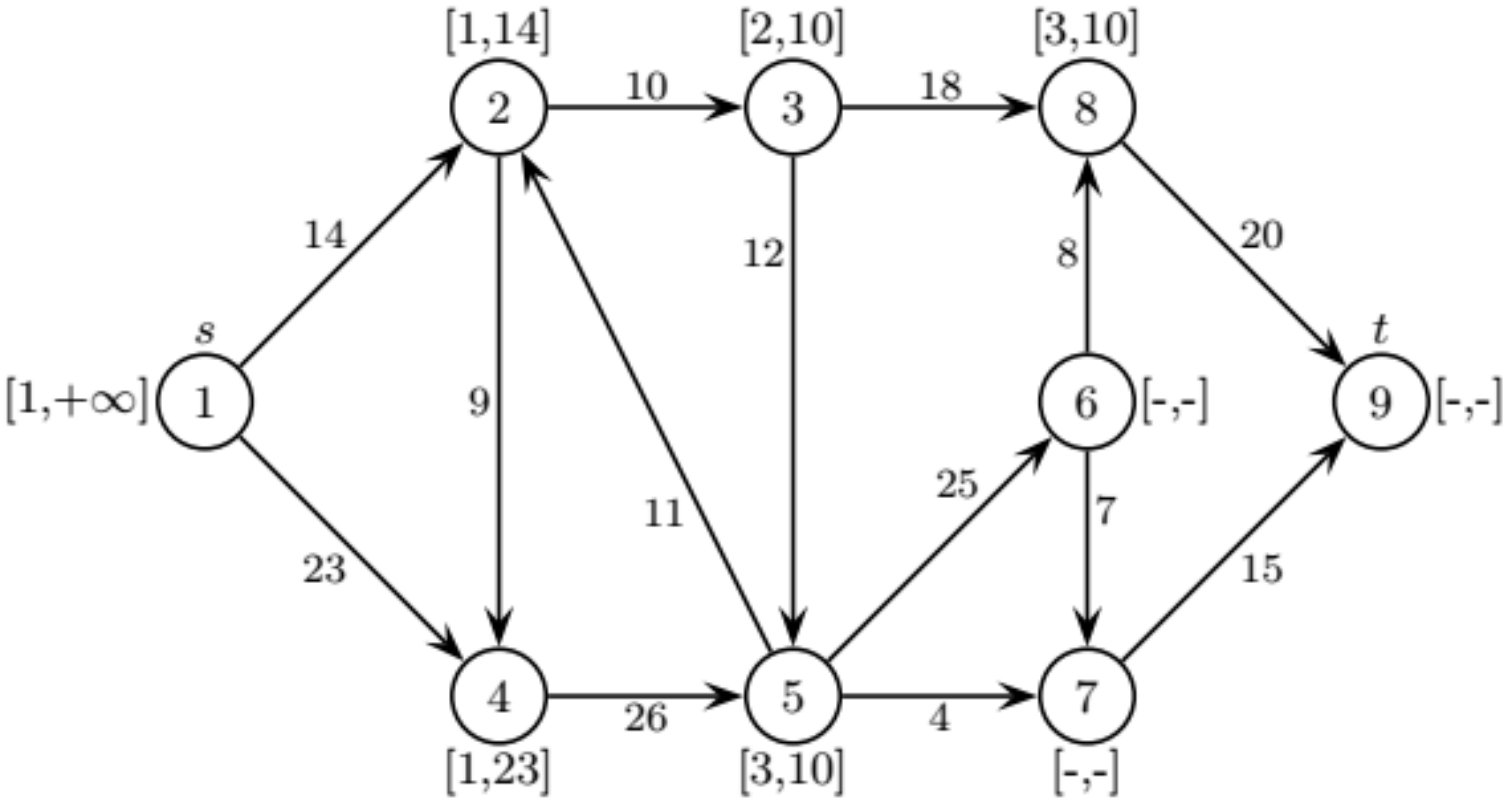
Algoritmo di Ford-Fulkerson: Esempio 2 (4)

Tra i nodi etichettati e non espansi (i.e., 2 e 4), viene espanso il nodo 2 considerando i nodi adiacenti 3 e 4 (che però è già stato etichettato):



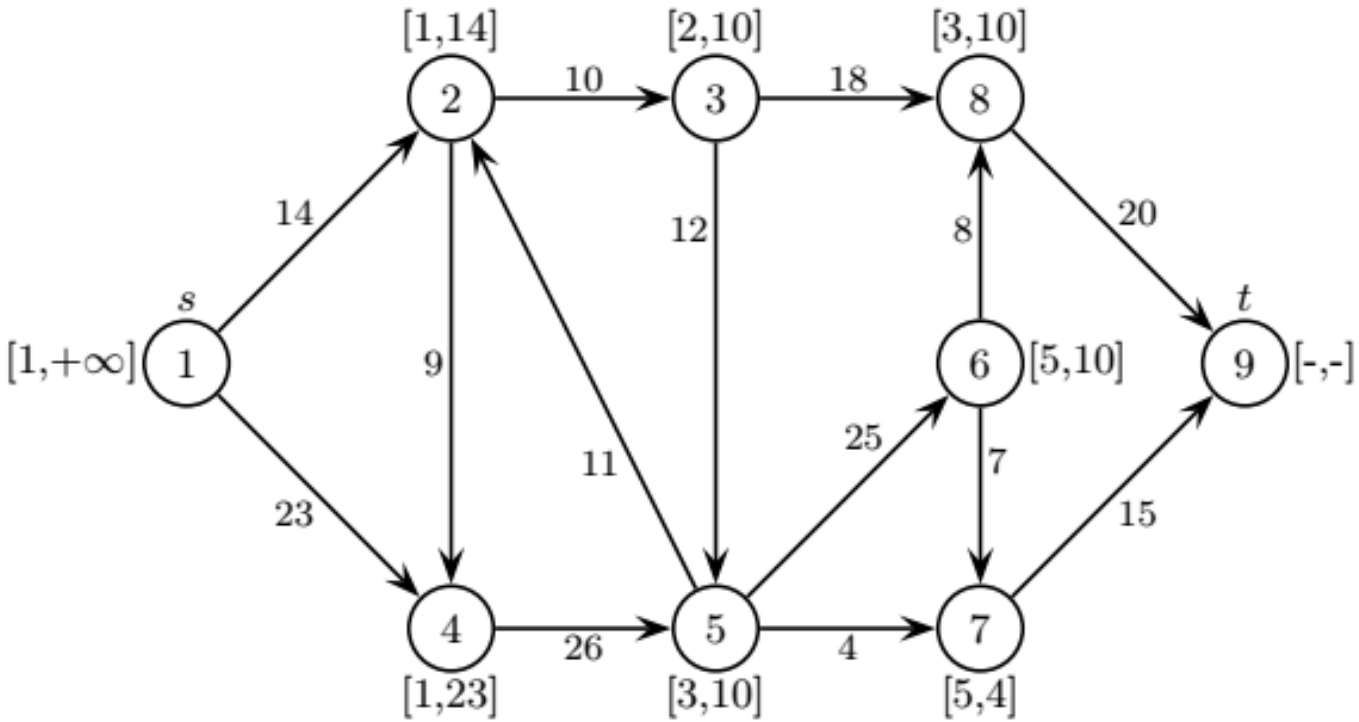
Algoritmo di Ford-Fulkerson: Esempio 2 (5)

Tra i nodi etichettati e non espansi (i.e., 3 e 4), viene espanso il nodo 3 considerando i nodi adiacenti 5 e 8:



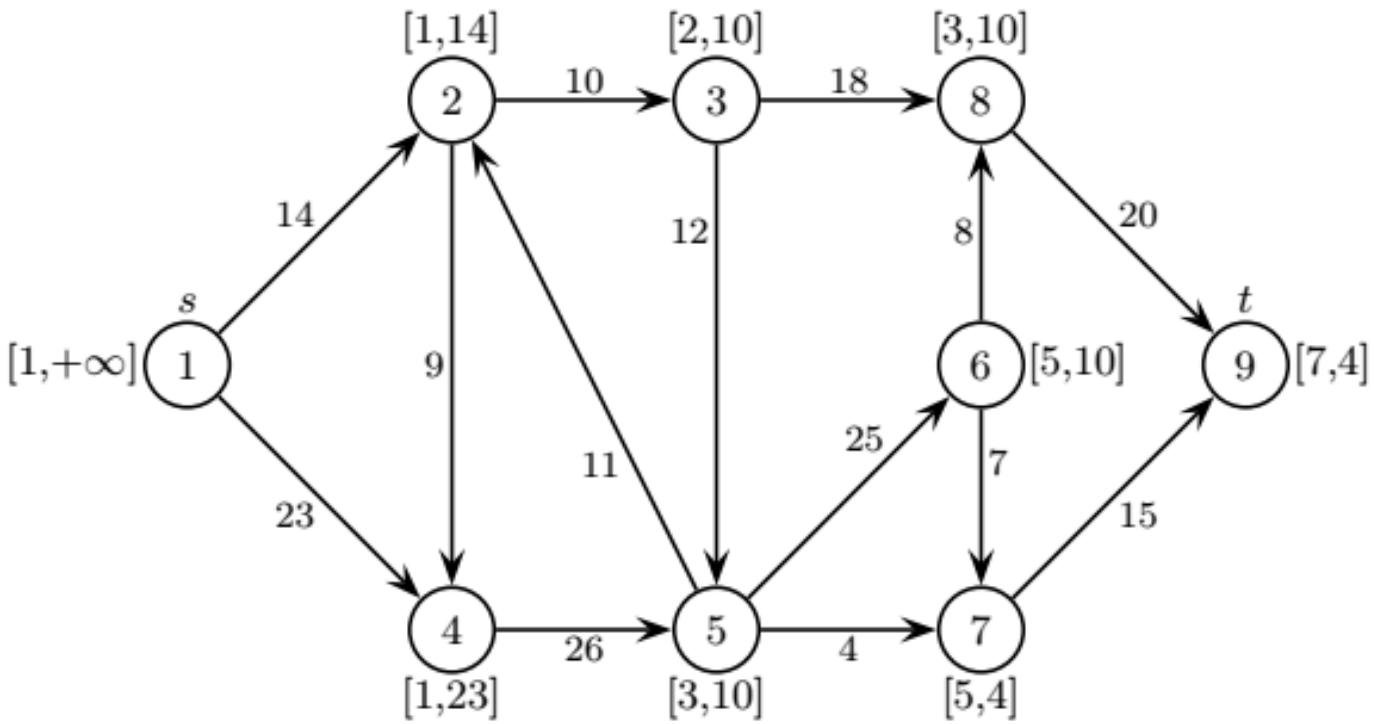
Algoritmo di Ford-Fulkerson: Esempio 2 (6)

Tra i nodi etichettati e non espansi (i.e., 4, 5 e 8), viene espanso il nodo 4 ma il nodo adiacente 5 è già stato etichettato. Quindi, passiamo a espandere il nodo 5 che ha i nodi adiacenti 2 (già etichettato), 6 e 7:



Algoritmo di Ford-Fulkerson: Esempio 2 (7)

Tra i nodi etichettati e non espansi (i.e., 6, 7 e 8), viene espanso il nodo 6 ma i nodi adiacenti 7 e 8 sono già etichettati. Quindi, passiamo a espandere il nodo 7 che ha il nodo adiacenti $t = 9$:

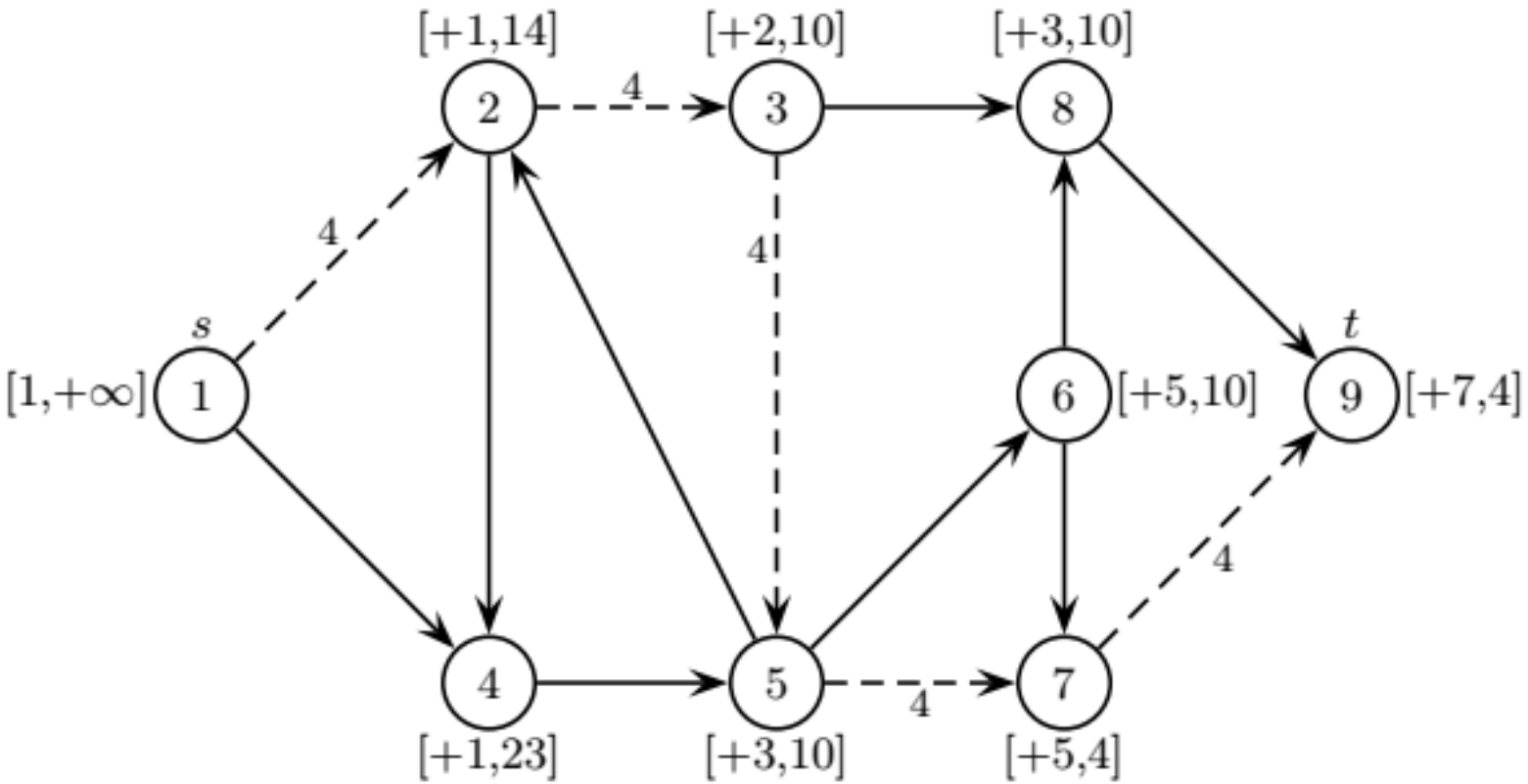


Algoritmo di Ford-Fulkerson: Esempio 2 (8)

- Siccome il nodo $t = 9$ è stato etichettato con $\delta_t = 4$, possiamo aumentare il flusso di 4 unità lungo il cammino aumentante.
- Il cammino aumentante $P = (1, 2, 3, 5, 7, 9)$ è stato ricostruito utilizzando le etichette a partire dal nodo $t = 9$.
- Il flusso viene aggiornato: $v = v + \delta_t = 0 + 4 = 4$.
- Il flusso degli archi (i.e., la soluzione x) si modifica come segue:
  - arco (7,9): $x_{79} = x_{79} + \delta_t = 0 + 4 = 4$;
  - arco (5,7): $x_{57} = x_{57} + \delta_t = 0 + 4 = 4$;
  - arco (3,5): $x_{35} = x_{35} + \delta_t = 0 + 4 = 4$;
  - arco (2,3): $x_{23} = x_{23} + \delta_t = 0 + 4 = 4$;
  - arco (1,2): $x_{12} = x_{12} + \delta_t = 0 + 4 = 4$.

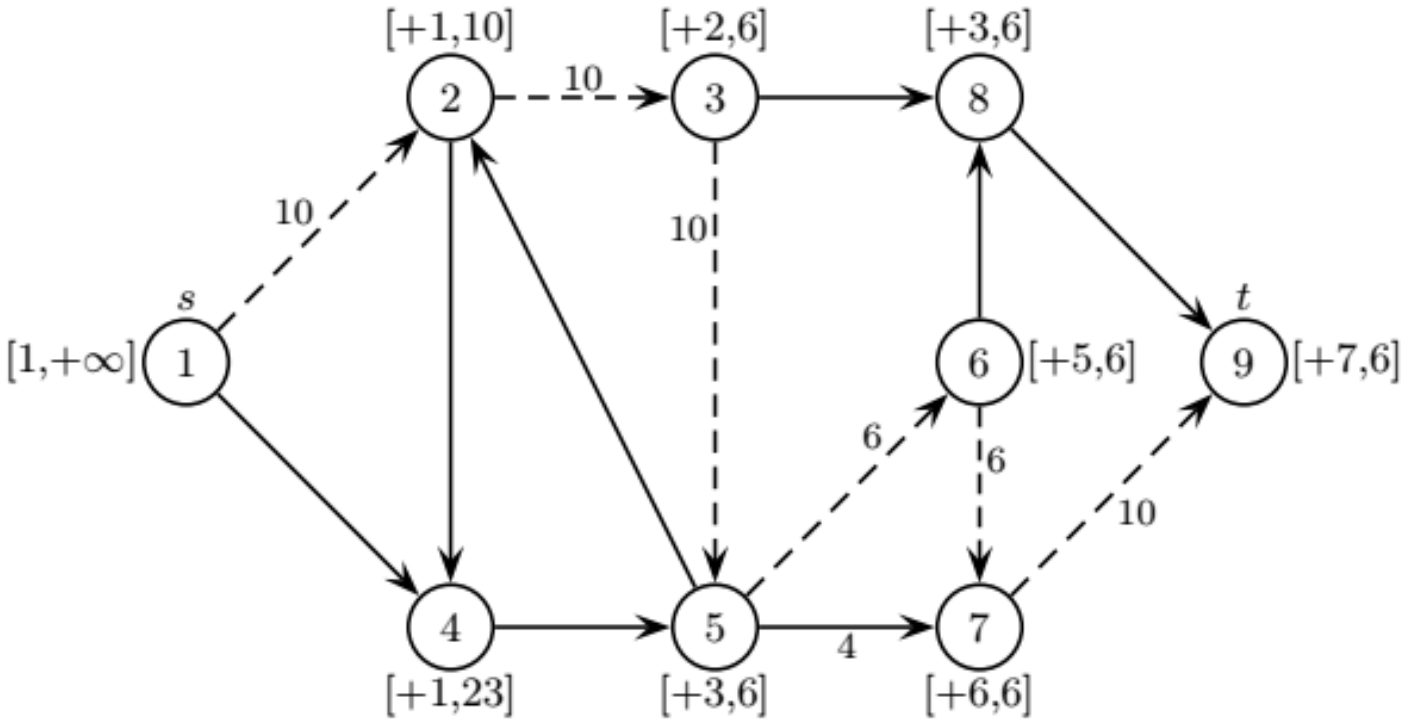
Algoritmo di Ford-Fulkerson: Esempio 2 (9)

Al termine della 1^a iterazione abbiamo la seguente situazione:



Algoritmo di Ford-Fulkerson: Esempio 2 (10)

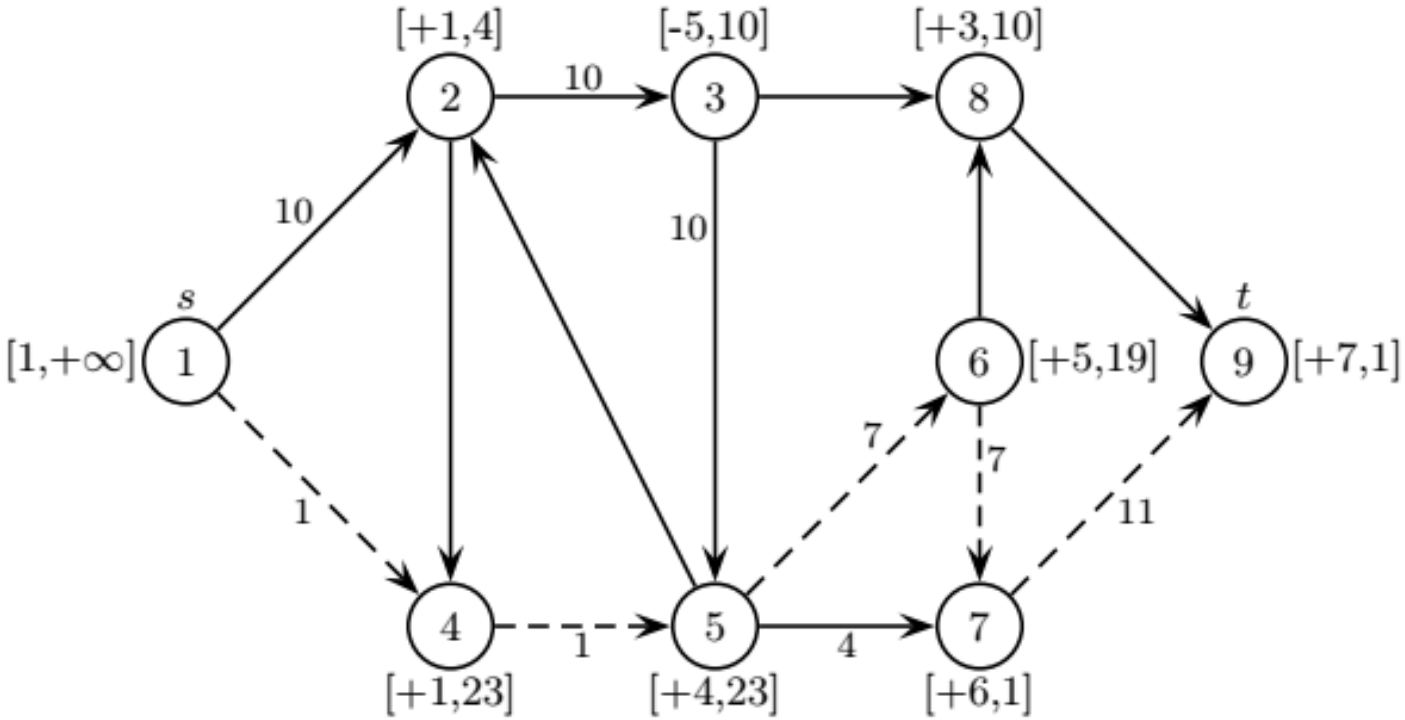
2^a iterazione:



Al termine dell'etichettamento possiamo aumentare il flusso di $\delta_t = 6$ unità lungo il cammino aumentante $P = (1, 2, 3, 5, 6, 7, 9)$ e il nuovo flusso è $v = v + \delta_t = 4 + 6 = 10$.

Algoritmo di Ford-Fulkerson: Esempio 2 (11)

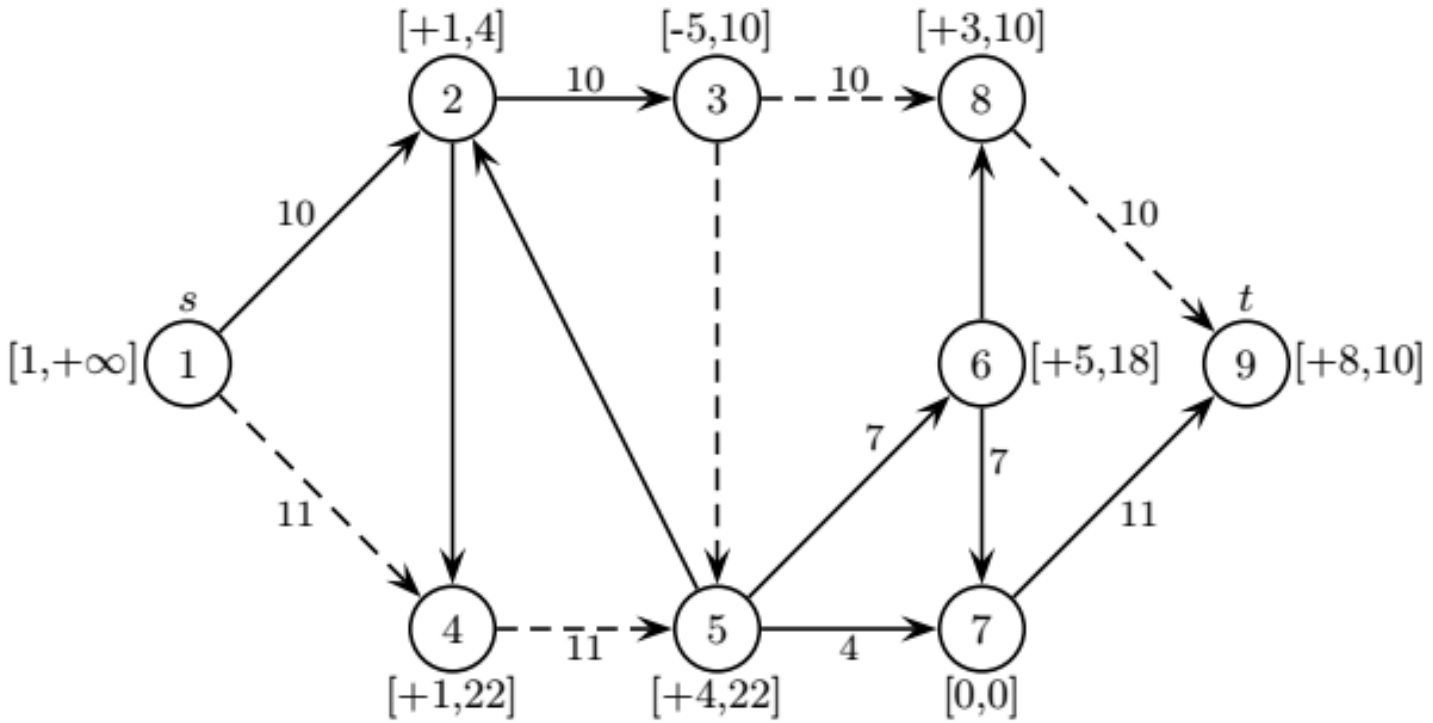
3^a iterazione:



Al termine dell'etichettamento possiamo aumentare il flusso di $\delta_t = 1$ unità lungo il cammino aumentante $P = (1, 4, 5, 6, 7, 9)$ e il nuovo flusso è $v = v + \delta_t = 10 + 1 = 11$.

Algoritmo di Ford-Fulkerson: Esempio 2 (12)

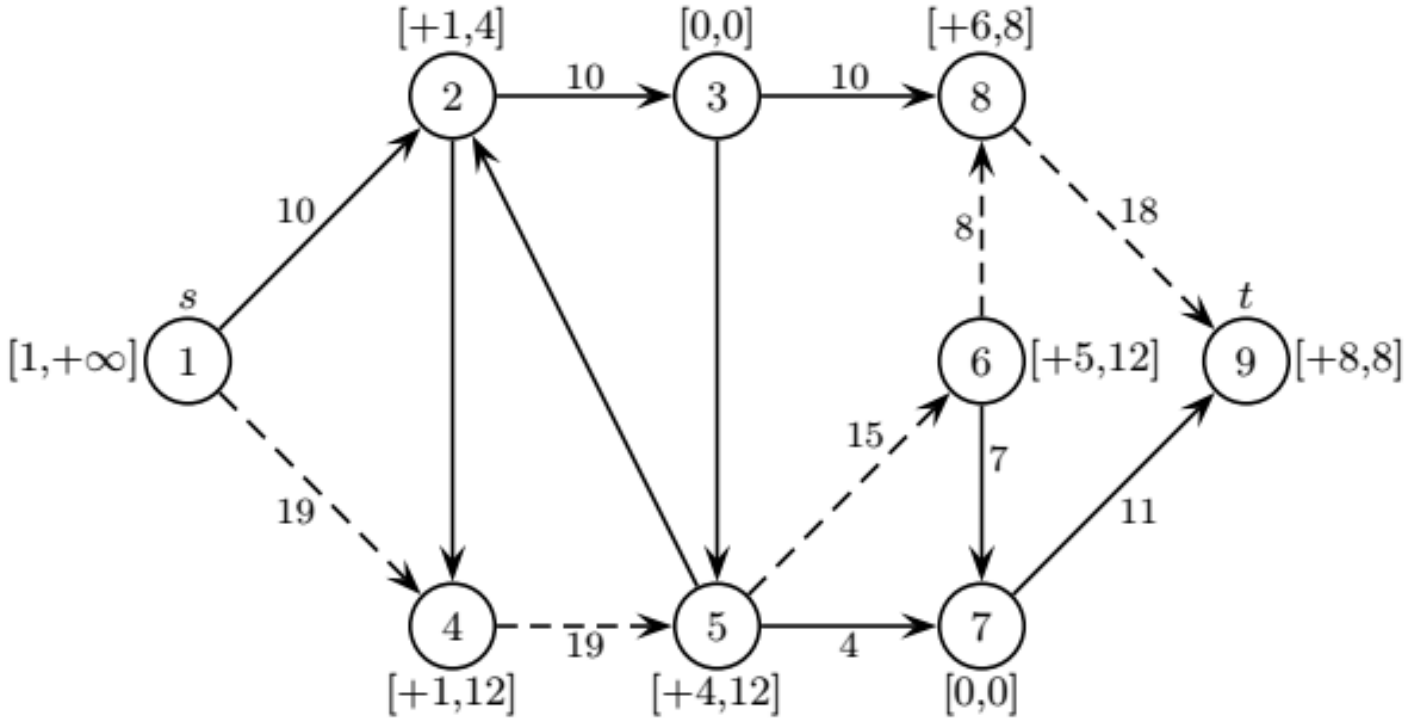
4^a iterazione:



Al termine dell'etichettamento possiamo aumentare il flusso di $\delta_t = 10$ unità lungo il cammino aumentante $P = (1, 4, 5, 3, 8, 9)$ e il nuovo flusso è $v = v + \delta_t = 11 + 10 = 21$.

Algoritmo di Ford-Fulkerson: Esempio 2 (13)

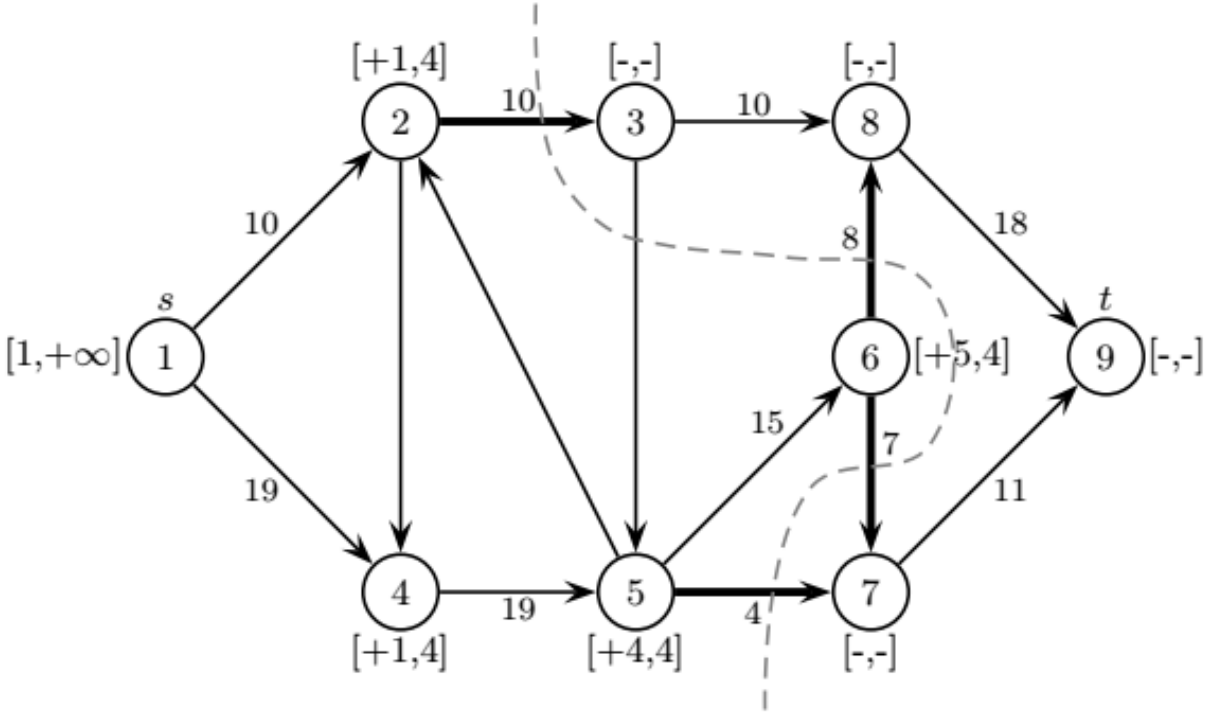
5^a iterazione:



Al termine dell'etichettamento possiamo aumentare il flusso di $\delta_t = 8$ unità lungo il cammino aumentante $P = (1, 4, 5, 6, 8, 9)$ e il nuovo flusso è $v = v + \delta_t = 21 + 8 = 29$.

Algoritmo di Ford-Fulkerson: Esempio 2 (14)

6^a iterazione:



Non esiste un cammino aumentante che raggiunge il nodo $t = 9$ per cui la soluzione corrente è ottima. Il flusso massimo è pari a 29.

Il taglio di capacità minima è determinato dall'insieme $S = \{1, 2, 4, 5, 6\}$ dei nodi etichettati e l'insieme $\bar{S} = \{3, 7, 8, 9\}$ dei nodi non etichettati.

Formulazione Matematica

- Sia dato un grafo direzionato $G = (N, A)$, in cui ad ogni arco $(i, j) \in A$ è associato un costo c_{ij} e una capacità u_{ij}.
- Ad ogni vertice $i \in N$ è associata una disponibilità $b_i > 0$ oppure una richiesta $b_i < 0$ oppure $b_i = 0$.
- Il problema del flusso di costo minimo può essere formulato come segue:

$$\begin{array}{rll} Min \;\; z(P) = & \sum_{(i,j)\in A} c_{ij} x_{ij} & \\ s.t. & \sum_{j\in\Gamma_i} x_{ij} - \sum_{j\in\Gamma_i^{-1}} x_{ji} = b_i, & i \in N \\ & 0 \le x_{ij} \le u_{ij}, & (i,j) \in A \end{array}$$

dove $\Gamma_i = \{j : (i, j) \in A\}$ e $\Gamma_i^{-1} = \{j : (j, i) \in A\}$.

Problema Duale

- Al vincolo di conservazione del flusso corrispondente al nodo $i \in N$ è associata una variabile duale π_i e al vincolo di capacità relativo all'arco $(i,j) \in A$ è associata la variabile duale α_{ij}.

- Il duale della formulazione P del problema del flusso di costo minimo è il seguente:

$$
\begin{array}{rll}
Max \;\; z(D) = & \sum_{i \in N} b_i \pi_i - \sum_{(i,j) \in A} u_{ij} \alpha_{ij} & \\
s.t. & \pi_i - \pi_j - \alpha_{ij} \leq c_{ij}, & (i,j) \in A \\
& \pi_i \;\; \text{qualsiasi}, & i \in N \\
& \alpha_{ij} \geq 0, & (i,j) \in A
\end{array}
$$

Assunzioni

- Tutti i dati (i.e., costi, richieste/disponibilità e capacità) sono valori interi.
- Il grafo è direzionato.
- Le richieste e le disponibilità dei diversi nodi soddisfano la condizione $\sum_{i \in N} b_i = 0$.
- Il problema di flusso di costo minimo ha una soluzione ammissibile.
- Il grafo contiene un cammino diretto di capacità infinita per ogni coppia di nodi.
- Tutti i costi c_{ij} sono non negativi e le capacità u_{ij} sono positive.

Definizioni

Parametri

- $n = |N|$ è il numero di nodi del grafo G;
- $m = |A|$ è il numero di archi del grafo G;
- $U = max\{u_{ij} : (i,j) \in A\}$;
- $C = max\{c_{ij} : (i,j) \in A\}$.

Grafo Residuo

- Il grafo residuo $G(x)$ corrispondente al flusso x è ottenuto sostituendo ogni arco $(i,j) \in A$ con due archi (i,j) e (j,i) tali che:
  - Il loro costo è pari a $c_{ij} = c_{ij}$ e $c_{ji} = -c_{ij}$.
  - La loro *capacità residua* è pari a $r_{ij} = u_{ij} - x_{ij}$ e $r_{ji} = x_{ij}$.

Potenziali e Proprietà

Potenziali

- Denotiamo con $\pi_i \in \mathbb{R}$ il *potenziale* associato ad ogni nodo $i \in N$.
- Definiamo il *costo ridotto* c_{ij}^{π} come segue:

$$c_{ij}^{\pi} = c_{ij} - \pi_i + \pi_j$$

- Si noti che il potenziale π_i corrisponde anche alla variabile duale associata al vincolo di conservazione del flusso relativo al nodo $i \in N$.

Proprietà dei Potenziali

- Per ogni cammino diretto P dal nodo k al nodo l si ha:

$$\sum_{(i,j)\in P} c_{ij}^{\pi} = \sum_{(i,j)\in P} c_{ij} - \pi_k + \pi_l$$

Potenziali e Proprietà (2)

- Per ogni ciclo diretto W si ha:

$$\sum_{(i,j)\in W} c_{ij}^{\pi} = \sum_{(i,j)\in W} c_{ij}$$

Condizioni di Ottimalità

Si consideri una soluzione ammissibile x^* del problema di flusso di costo minimo.

- **Assenza Cicli di Costo Negativo**

  La soluzione x^* è ottima se e solo se il grafo residuo $G(x^*)$ non contiene cicli di costo negativo.

- **Costi Ridotti Positivi**

  La soluzione x^* è ottima se e solo se è possibile trovare dei *potenziali* π tali che per ogni arco (i,j) di $G(x^*)$ soddisfano la seguente condizione:

$$c_{ij}^{\pi} = c_{ij} - \pi_i + \pi_j \geq 0$$

Condizioni di Ottimalità (2)

- **Relazione degli Scarti Complementari**

  La soluzione x^* è ottima se e solo se è possibile trovare dei *potenziali* π tali che per ogni arco $(i,j) \in A$ il *costo ridotto* c_{ij}^{π} soddisfa le seguenti condizioni degli scarti complementari:

$$\begin{aligned} &\text{If } c_{ij}^{\pi} > 0, \quad \text{then } x_{ij}^* = 0 \\ &\text{If } c_{ij}^{\pi} = 0, \quad \text{then } 0 \leq x_{ij}^* \leq u_{ij} \\ &\text{If } c_{ij}^{\pi} < 0, \quad \text{then } x_{ij}^* = u_{ij} \end{aligned}$$

Cycle Cancelling Algorithm

L'algoritmo si basa sulla seguente osservazione: "se nel grafo $G(x)$ identifico un ciclo W di costo negativo, allora posso aumentare il flusso in W diminuendo il costo complessivo della soluzione".

algorithm cycle cancelling
begin
    Stabilisci un flusso ammissibile x nel grafo G;
    while $G(x)$ contiene un ciclo di costo negativo **do**
    begin
        Applica un algoritmo per identificare un ciclo di costo negativo W;
        Calcola $\delta = min\{r_{ij} : (i,j) \in W\}$;
        Aumenta di δ unita' il flusso nel ciclo W;
        Aggiorna $G(x)$;
    end
end

Cycle Cancelling Algorithm (2)

L'algoritmo mantiene ad ogni passo l'ammissibilità della soluzione e cerca di ottenere il soddisfacimento delle condizioni di ottimalità.

Calcolo di un flusso ammissibile

Un flusso ammissibile può essere calcolato risolvendo un problema di flusso massimo sul grafo G' costruito come segue:

- Aggiugiamo a G due nuovi nodi s e t;
- Per ogni nodo $i \in N$ con $b_i > 0$ aggiungiamo un *arco sorgente* (s, i) con capacità $u_{si} = b_i$;
- Per ogni nodo $i \in N$ con $b_i < 0$ aggiungiamo un *arco destinazione* (i, t) con capacità $u_{it} = -b_i$.

Cycle Cancelling Algorithm (3)

Se il flusso massimo da s a t calcolato satura tutti gli archi sorgente e destinazione, allora il flusso trovato è ammissibile per il grafo G, altrimenti per G non può esistere un flusso ammissibile.

Identificazione in un ciclo di costo negativo

Per identificare un ciclo W di costo negativo in $G(x)$ si può usare un algoritmo di tipo "label-correcting" per il calcolo del cammino di costo minimo (per esempio l'algoritmo di Bellman-Ford).

Cycle Cancelling Algorithm (4)

Complessità computazionale

Nel caso peggiore il Cycle Cancelling Algorithm esegue $O(mCU)$ iterazioni.

Ad ogni iterazione l'algoritmo risolve il problema del cammino di costo minimo con costi qualsiasi per identificare un ciclo di costo negativo.

Nel caso in cui sia utilizzato Bellman-Ford, la ricerca di un ciclo di costo negativo ha complessità pari a $O(nm)$.

Quindi, nel caso sia impiegato Bellman-Ford, la complessità computazionale del Cycle Cancelling Algorithm è pari a $O(nm^2CU)$.

Cycle Cancelling Algorithm: Esempio

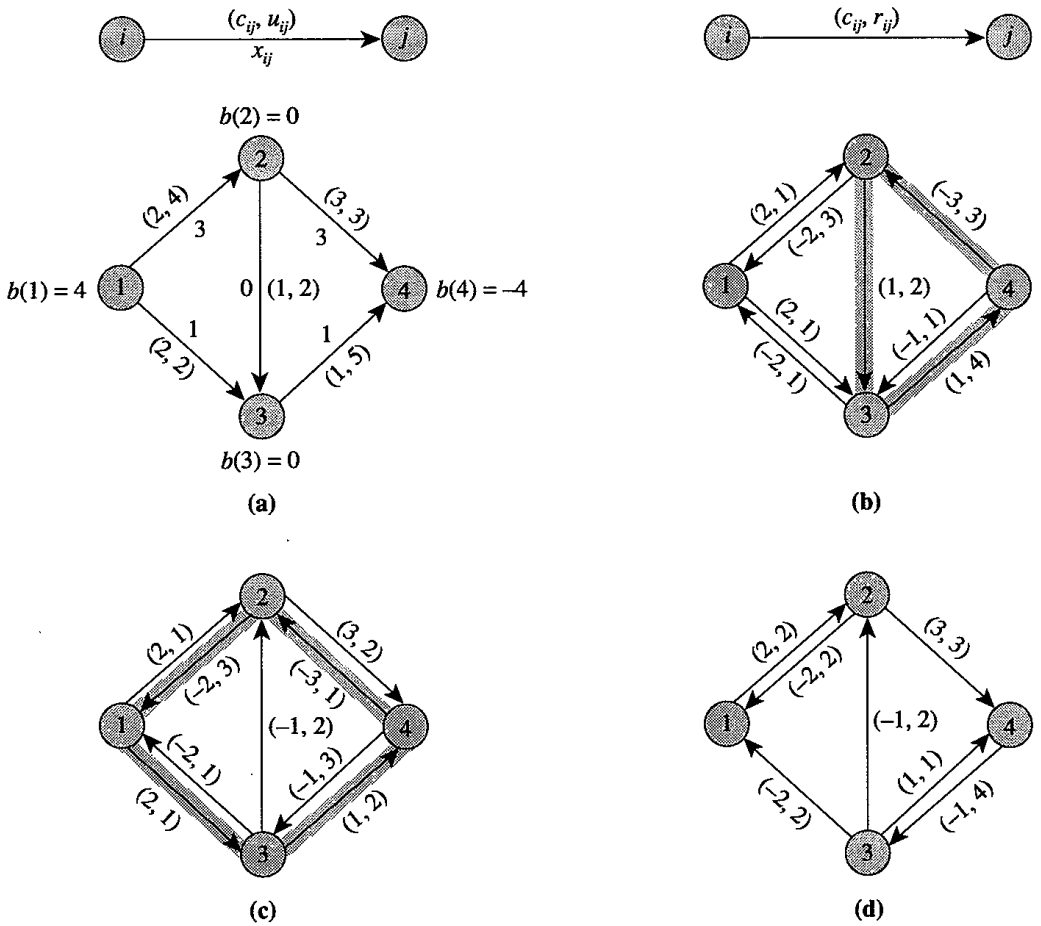


Figure 9.8 Illustrating the cycle canceling algorithm: (a) network example with a feasible flow x; (b) residual network $G(x)$; (c) residual network after augmenting 2 units along the cycle 4–2–3–4; (d) residual network after augmenting 1 unit along the cycle 4–2–1–3–4.

Successive Shortest Path Algorithm

L'algoritmo ad ogni passo mantiene soddisfatte le condizioni di non negatività dei costi ridotti e i vincoli di capacità, mentre cerca di soddisfare i vincoli di conservazione del flusso:

$$\sum_{j \in \Gamma_i} x_{ij} - \sum_{j \in \Gamma_i^{-1}} x_{ji} = b_i$$

Il flusso x sarà ammissibile solo al termine dell'algoritmo e quindi nelle fasi intermedie diremo che x è un *pseudoflusso*. Inoltre, per ogni nodo i definiamo:

$$e_i = b_i - \sum_{j \in \Gamma_i} x_{ij} + \sum_{j \in \Gamma_i^{-1}} x_{ji}$$

Successive Shortest Path Algorithm (2)

Lemma

Sia x un pseudoflusso che soddisfa le condizioni di non negatività dei costi ridotti per un qualche π, i.e. $c_{ij}^{\pi} = c_{ij} - \pi_i + \pi_j \geq 0$, $\forall (i,j)$ in $G(x)$.
Sia $d = (d_1, \ldots, d_n)$ la distanza minima in $G(x)$ da un nodo s a tutti gli altri nodi rispetto ai costi ridotti c_{ij}^{π}. Le seguenti proprietà sono valide:

- *Il pseudoflusso x soddisfa le condizioni di non negatività dei costi ridotti anche per i potenziali $\pi' = \pi - d$;*
- *I costi ridotti $c_{ij}^{\pi'}$ sono nulli per tutti gli archi (i,j) nel cammino minimo da s a ogni altro nodo.* □

Successive Shortest Path Algorithm (3)

Lemma

Sia x un pseudoflusso che soddisfa le condizioni di non negatività dei costi ridotti per un qualche potenziale π.
Se x' è il pseudoflusso ottenuto da x aumentando il flusso lungo il cammino di costo minimo da s a un altro nodo k, allora anche x' soddisfa le condizioni di non negatività dei costi ridotti per un qualche π', e.g. $\pi' = \pi - d$. $\square$

Il Lemma suggerisce la seguente strategia: "Ad ogni iterazione l'algoritmo deve selezionare un nodo s con un *eccesso* di flusso, i.e. $e_s > 0$, e un nodo t con un *deficit* di flusso, i.e. $e_t < 0$, e aumentare il flusso lungo il cammino di costo minimo da s a t nel grafo $G(x)$".

Si noti che nel calcolo del cammino di costo minimo si impiegano i costi ridotti c_{ij}^{π} che sono positivi. Pertanto, può essere utilizzato Dijkstra.

Successive Shortest Path Algorithm (4)

algorithm successive shortest path
begin
    $x = 0$, $\pi = 0$ e $e_i = b_i$ per ogni $i \in N$;
    $E = \{i \in N : e_i > 0\}$ e $D = \{i \in N : e_i < 0\}$;
    while $E \neq \emptyset$ **do**
    begin
        Seleziona un nodo $k \in E$ e un nodo $l \in D$;
        Calcola in $G(x)$ le distanze minime d da k
            a tutti gli altri nodi rispetto ai costi ridotti c_{ij}^{π};
        Sia P il cammino di costo minimo da k a l;
        Aggiorna $\pi = \pi - d$;
        Calcola $\delta = min\{e_k, -e_l, min\{r_{ij} : (i,j) \in P\}\}$;
        Aumenta di δ unita' il flusso nel cammino P;
        Aggiorna $G(x)$;
    end
end

Successive Shortest Path Algorithm (5)

Complessità computazionale

Nel caso peggiore il Successive Shortest Path Algorithm esegue $O(nU)$ iterazioni.

Ad ogni iterazione l'algoritmo risolve il problema del cammino di costo minimo con costi positivi o uguali a zero (i.e., non-negativi).

Nel caso sia utilizzato Dijkstra con Fibonacci Heap la complessità del calcolo dei cammini di costo minimo è pari a $O(m + n \log n)$.

Nel caso sia impiegato Dijkstra con Fibonacci Heap, la complessità computazionale complessiva del Successive Shortest Path Algorithm è pari a $O(nU(m + n \log n))$.

Successive Shortest Path Algorithm: Esempio

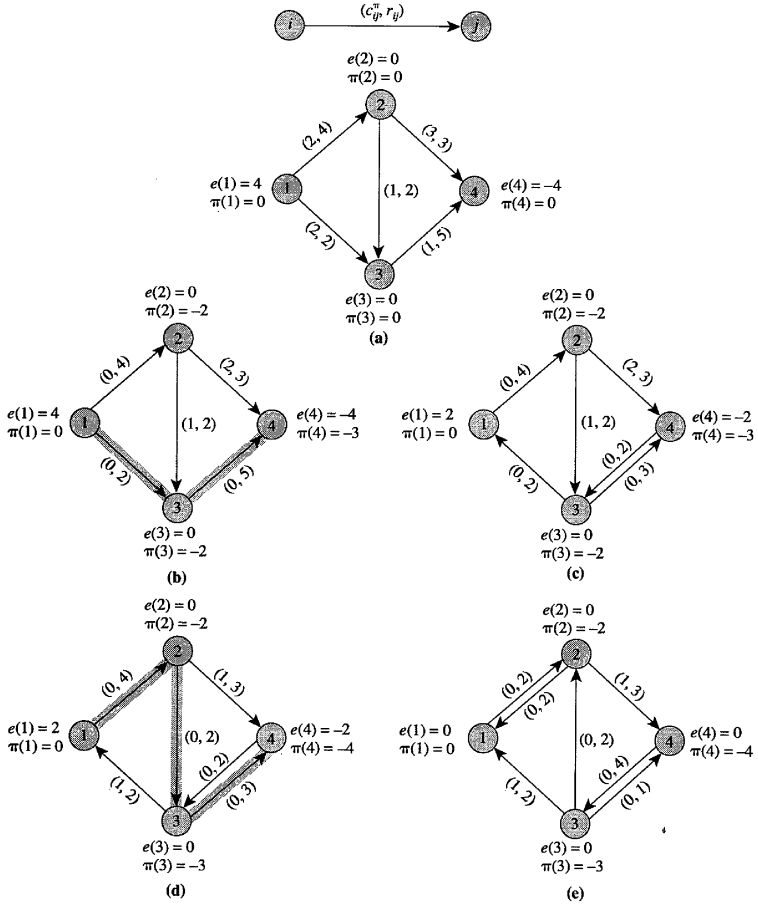
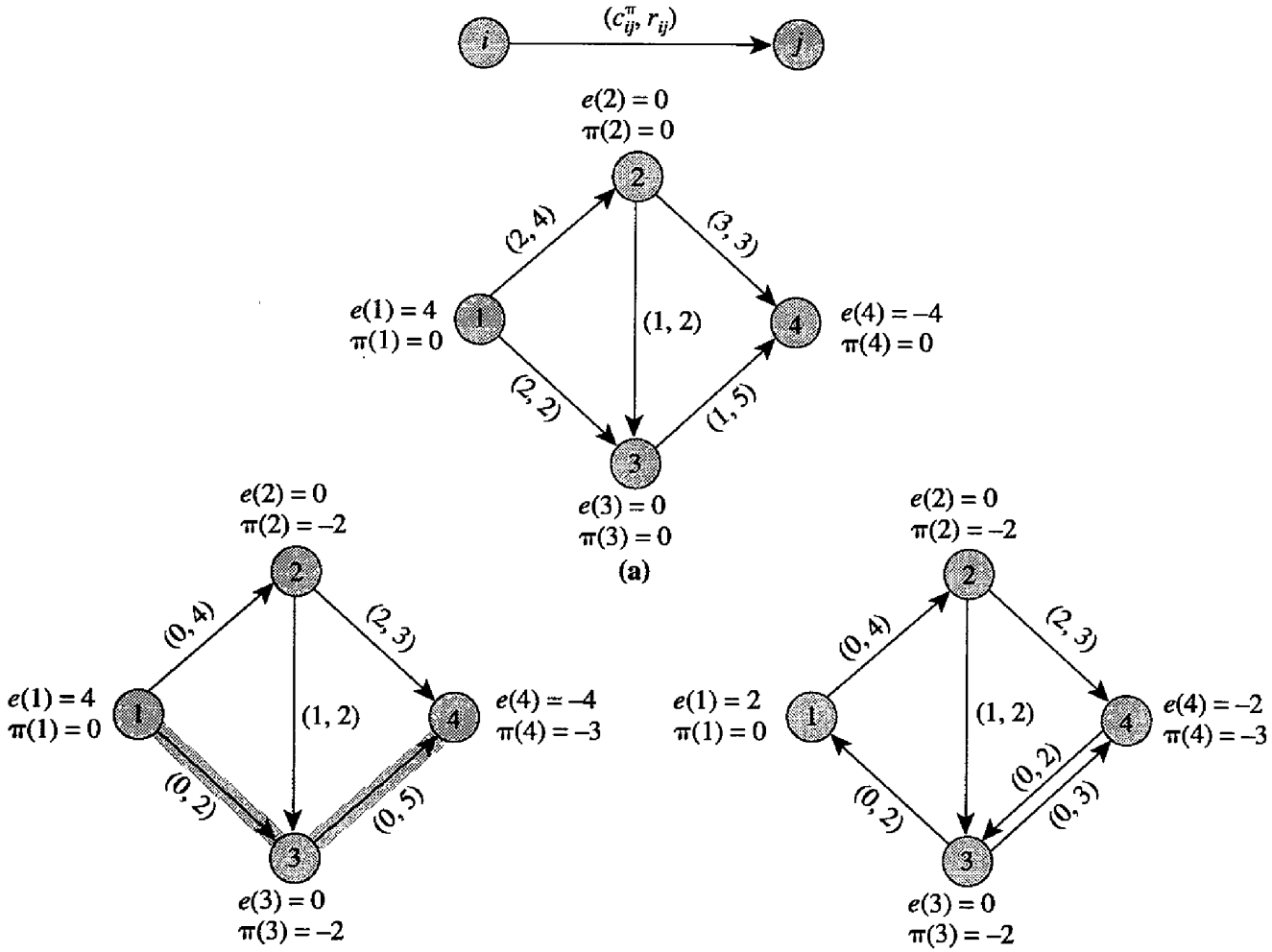
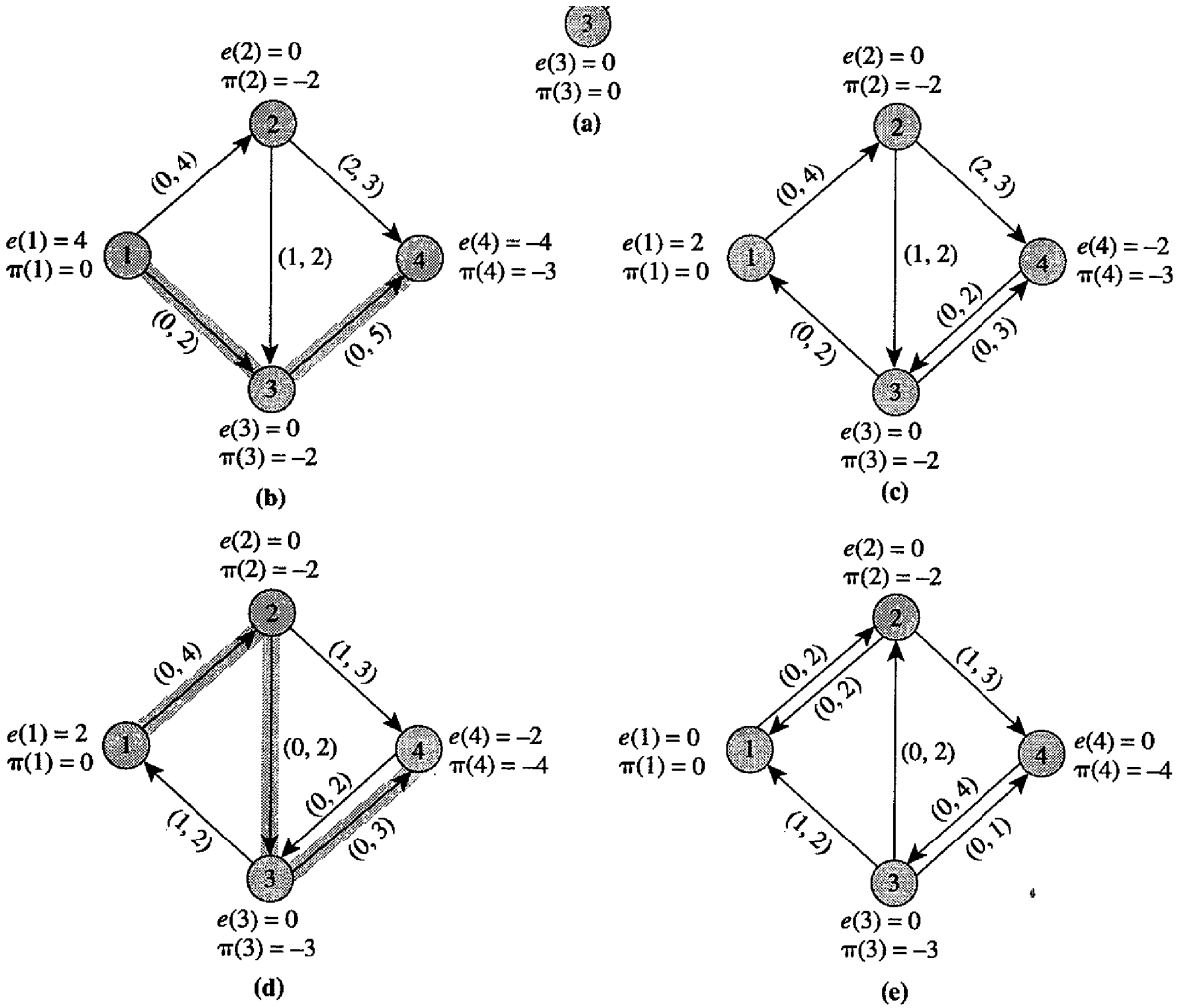


Figure 9.10 Illustrating the successive shortest path algorithm: (a) initial residual network for $x = 0$ and $\pi = 0$; (b) network after updating the potentials π; (c) network after augmenting 2 units along the path 1–3–4; (d) network after updating the potentials π; (e) network after augmenting 2 units along the path 1–2–3–4.

Successive Shortest Path Algorithm: Esempio



Successive Shortest Path Algorithm: Esempio



Primal-Dual Algorithm

L'algoritmo primale-duale per la soluzione del problema del flusso di costo minimo è simile al Successive Shortest Path Algorithm, in quanto anch'esso mantiene un pseudoflusso che soddisfa le condizioni di non negatività dei costi ridotti e i vincoli di capacità, mentre i vincoli di conservazione del flusso sono soddisfatti solo al raggiungimento della soluzione ottima.

L'algoritmo primale-duale prevede di trasformare il problema originale in modo tale da avere un solo nodo con un eccesso di flusso e un solo nodo con un deficit di flusso. Quindi il grafo viene modificato come segue:

- Aggiugiamo a G due nuovi nodi s e t;
- Per ogni nodo $i \in N$ con $b_i > 0$ aggiungiamo un *arco sorgente* (s, i) con capacità $u_{si} = b_i$;

Primal-Dual Algorithm (2)

- Per ogni nodo $i \in N$ con $b_i < 0$ aggiungiamo un *arco destinazione* (i,t) con capacità $u_{it} = -b_i$;
- Poniamo $b_s = \sum_{i \in N^+} b_i$, dove $N^+ = \{i \in N : b_i > 0\}$, $b_t = -b_s$ e $b_i = 0$ per ogni $i \in N$.

Grafo Ammissibile

Siano dati i potenziali π. Il *grafo ammissibile* $G^a(x)$ è un sottografo di $G(x)$ che contiene solo gli archi (i,j) di $G(x)$ di costo ridotto nullo, i.e. $c_{ij}^{\pi} = 0$. La capacità residua r_{ij} è la stessa del corrispondente arco in $G(x)$.

Primal-Dual Algorithm (3)

algorithm primal-dual
begin
    $x = 0$ e $\pi = 0$;
    $e_s = b_s$ e $e_t = b_t$;
    while $e_s > 0$ **do**
    begin
        Calcola in $G(x)$ le distanze minime d da s
            a tutti gli altri nodi rispetto ai costi ridotti c_{ij}^{π};
        Aggiorna $\pi = \pi - d$;
        Definisci il *grafo ammissibile* $G^a(x)$;
        Calcola in $G^a(x)$ il flusso massimo dal s a t;
        Aggiorna e_s, e_t e $G(x)$;
    end
end

Primal-Dual Algorithm (4)

Complessità computazionale

Nel caso peggiore l'algoritmo primale duale esegue un numero di iterazioni pari a $O(min\{nU, nC\})$.

Ad ogni iterazione l'algoritmo risolve un problema del cammino di costo minimo con costi non negativi e un problema del flusso massimo.

Nel caso sia utilizzato Dijkstra con Fibonacci Heap la complessità del calcolo dei cammini di costo minimo è pari a $O(m + n \log n)$.

Nel caso sia utilizzato l'Highest-Label Preflow-Push la complessità del calcolo del flusso massimo è pari a $O(n^2\sqrt{m})$.

Quindi, nel caso sia impiegato Dijkstra con Fibonacci Heap e l'Highest-Label Preflow-Push, la complessità computazionale dell'algoritmo primale duale è pari a $O((min\{nU, nC\})((m + n \log n) + (n^2\sqrt{m})))$.

Primal-Dual Algorithm: Esempio

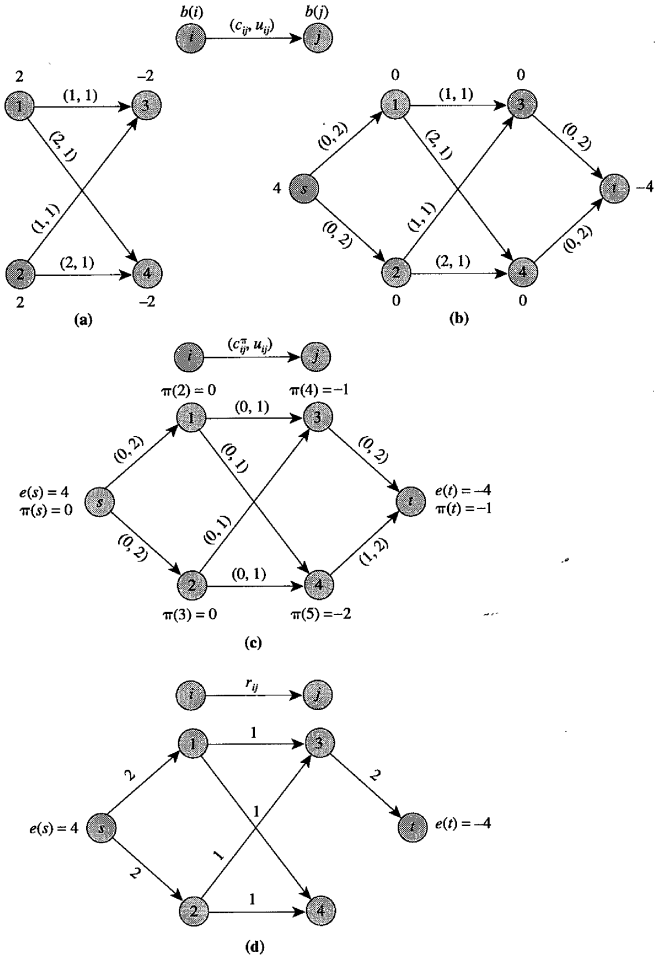
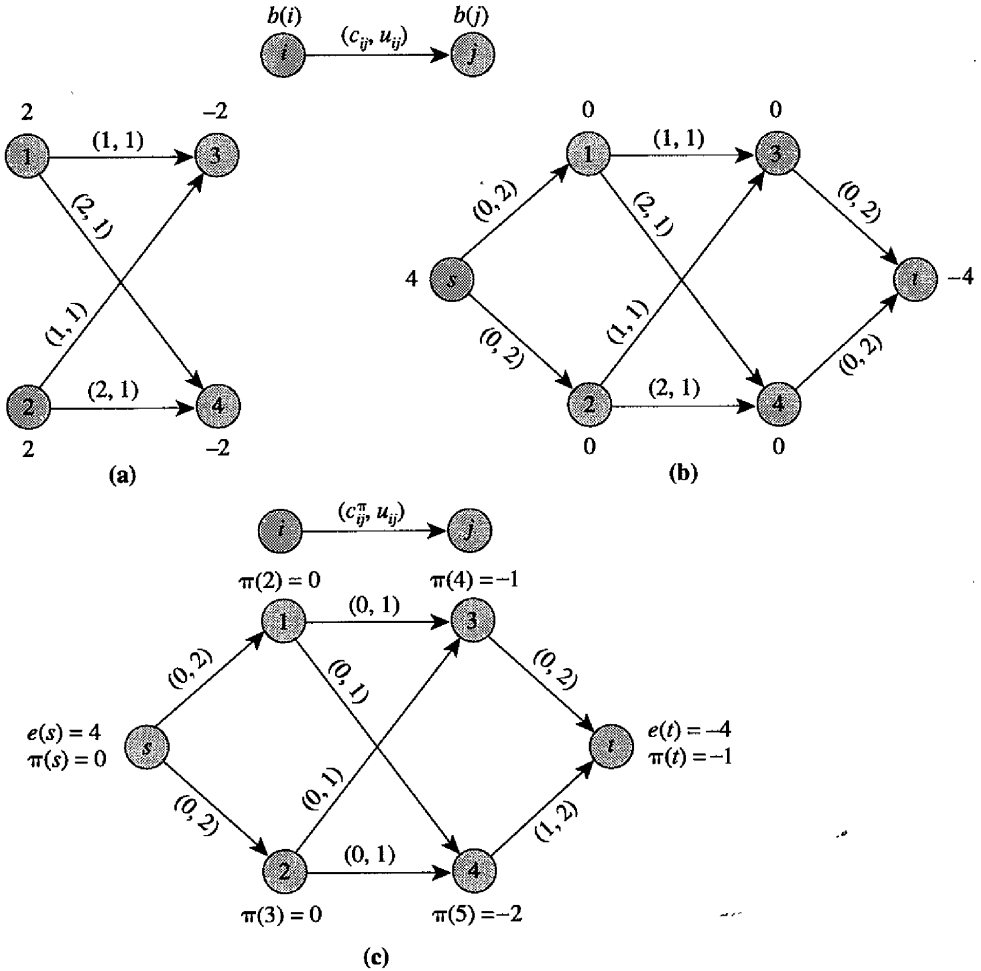
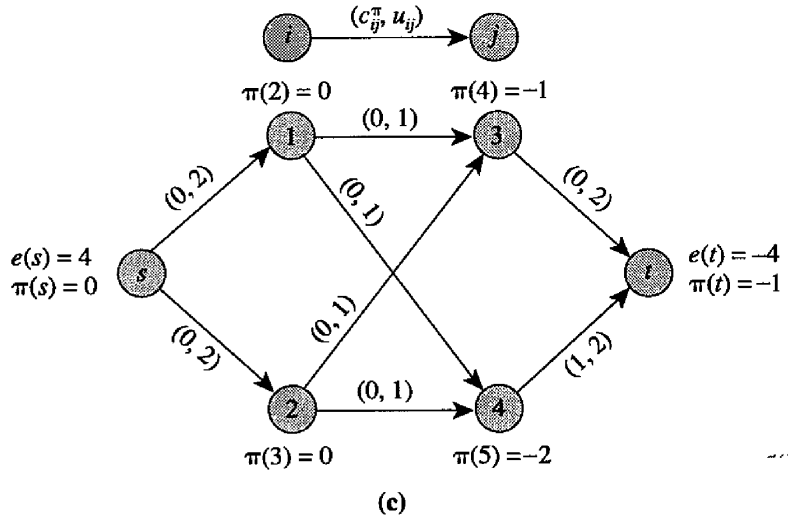


Figure 9.12 Illustrating the primal–dual algorithm: (a) example network; (b) transformed network; (c) residual network after updating the node potentials; (d) admissible network.

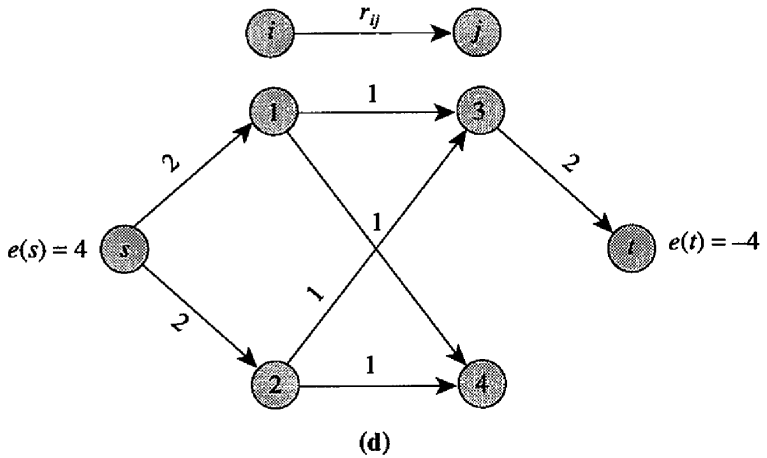
Primal-Dual Algorithm: Esempio



Primal-Dual Algorithm: Esempio



(c)



(d)

Out-of-Kilter Algorithm

L'algoritmo Out-of-Kilter mantiene soddisfatti i vincoli di conservazione del flusso ad ogni nodo $i \in N$, mentre consente la violazione sia delle condizioni di ottimalità che dei vincoli di capacità degli archi $(i,j) \in A$.

L'algoritmo iterativamente modifica il flusso x e i potenziali π, in modo da *diminuire* la non ammissibilità della soluzione e convergere alla soluzione ottima.

Ricordiamo che la soluzione x^* è ottima se e solo se è possibile trovare dei potenziali π tali che per ogni arco $(i,j) \in A$ il corrispondente costo ridotto c_{ij}^{π} soddisfa le seguenti condizioni degli scarti complementari:

$$\begin{aligned} &\text{If } c_{ij}^{\pi} > 0, \quad \text{then } x_{ij}^* = 0 \\ &\text{If } c_{ij}^{\pi} = 0, \quad \text{then } 0 \le x_{ij}^* \le u_{ij} \\ &\text{If } c_{ij}^{\pi} < 0, \quad \text{then } x_{ij}^* = u_{ij} \end{aligned}$$

Out-of-Kilter Algorithm (2)

Se un arco $(i,j) \in A$ soddisfa le condizioni di ottimalità diremo che è *in-kilter*, mentre se non le soddisfa diremo che è *out-of-kilter*.

Kilter Number

Il *Kilter Number* $k_{ij} \geq 0$ indica di quanto deve essere modificato il flusso x_{ij} per rendere *in-kilter* l'arco $(i,j) \in A$ rispetto al costo ridotto c^{π}_{ij}:

$$\begin{aligned}
&\text{If } c^{\pi}_{ij} > 0, && \text{then } k_{ij} = |x_{ij}| \\
&\text{If } c^{\pi}_{ij} = 0 \text{ and } x_{ij} > u_{ij}, && \text{then } k_{ij} = x_{ij} - u_{ij} \\
&\text{If } c^{\pi}_{ij} = 0 \text{ and } x_{ij} < 0, && \text{then } k_{ij} = -x_{ij} \\
&\text{If } c^{\pi}_{ij} < 0, && \text{then } k_{ij} = |u_{ij} - x_{ij}|
\end{aligned}$$

Il kilter number k_{ij} indica il livello di non ammissibilità dell'arco $(i,j) \in A$.

Out-of-Kilter Algorithm (3)

L'algoritmo Out-Of-Kilter ad ogni iterazione individua un arco $(i,j) \in A$ out-of-kilter e riduce di una quantità positiva il corrispondente kilter number k_{ij} senza dover aumentare il kilter number degli altri nodi.

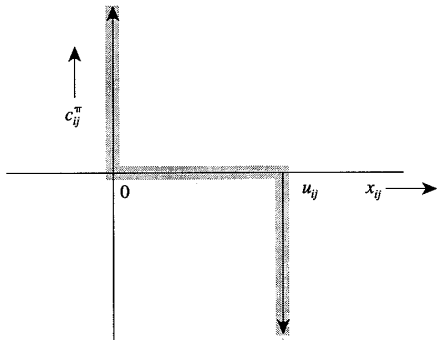


Figure 9.13 Kilter diagram for arc (i, j).

Se facciamo l'assunzione di avere un flusso iniziale ammissibile, allora la definizione del kilter number per ogni arco $(i,j) \in A$ si riduce a:

$$k_{ij} = \begin{cases} 0, & \text{If } c_{ij}^{\pi} \geq 0 \\ r_{ij}, & \text{If } c_{ij}^{\pi} < 0 \end{cases}$$

Out-of-Kilter Algorithm (4)

algorithm out-of-kilter
begin
Sia $\pi = 0$ e x un flusso ammissibile;
Calcola $G(x)$ e il *kilter number* k_{ij} per ogni arco (i,j);
while $G(x)$ contiene un arco (p,q) *out-of-kilter* **do**
begin
Definisci il costo di ogni arco (i,j) in $G(x)$ come $c'_{ij} = max\{0, c^{\pi}_{ij}\}$;
Calcola in $G(x) - \{(q,p)\}$ le distanze minime d
da q a tutti gli altri nodi rispetto ai costi c'_{ij};
Sia P il cammino di costo minimo da q a p;
Aggiorna $\pi' = \pi - d$;
if $c^{\pi'}_{pq} < 0$ **then**
$W = P \cup \{(p,q)\}$;
Calcola $\delta = min\{r_{ij} : (i,j) \in W\}$;
Aumenta di δ unita' il flusso nel ciclo W;
Aggiorna x, $G(x)$, i costi ridotti c^{π}_{ij} e i kilter number k_{ij};
end if
end
end

Out-of-Kilter Algorithm (5)

Complessità computazionale

Nel caso peggiore l'algoritmo Out-Of-Kilter esegue un numero di iterazioni pari a $O(mU)$.

Ad ogni iterazione l'algoritmo risolve il problema del cammino di costo minimo con costi non negativi.

Nel caso sia utilizzato Dijkstra con Fibonacci Heap la complessità del calcolo dei cammini di costo minimo è pari a $O(m + n \log n)$.

Nel caso sia impiegato Dijkstra con Fibonacci Heap, la complessità computazionale dell'algoritmo Out-Of-Kilter è pari a $O(mU(m + n \log n))$.

Algoritmi Polinomiali

Per ottenere degli algoritmi polinomiali per risolvere il problema del flusso di costo minimo possono essere utilizzate delle tecniche di *scaling*.

Capacity Scaling Algorithm

Il Capacity Scaling Algorithm è una variante del Successive Shortest Path Algorithm in cui ad ogni iterazione viene garantito che la capacità residua del *cammino aumentante* sia "*sufficientemente grande*".

Viene definito un parametro Δ e ad ogni iterazione:

- si selezionano i nodi s e t tali che $e_s \geq \Delta$ e $e_t \leq -\Delta$;
- si sostituisce il grafo residuo $G(x)$ con il suo sottografo $G(x, \Delta)$ contenente solo quegli archi di capacità residua pari almeno a Δ.

Algoritmi Polinomiali (2)

Quando non è più possibile determinare una coppia di vertici s e t tali che $e_s \geq \Delta$ e $e_t \leq -\Delta$, allora si diminuisce il parametro Δ, i.e. $\Delta = \Delta/2$. Se $\Delta = 1$ allora il flusso x corrente è ottimo.

Complessità computazionale

Se si inizializza $\Delta = 2^{\lfloor \log U \rfloor}$ allora il Capacity Scaling Algorithm nel caso peggiore eseguirà $O(m \log U)$ iterazioni.

Quindi, nel caso sia impiegato Dijkstra con Fibonacci Heap, la complessità computazionale complessiva del Capacity Scaling Algorithm è pari a $O((m \log U)(m + n \log n))$.

Algoritmi Fortemente Polinomiali

Algoritmi fortemente polinomiali per risolvere il problema del flusso di costo minimo possono essere ottenuti modificando il Cycle Cancelling Algorithm e il Capacity Scaling Algorithm.

Minimum Mean Cycle Cancelling Algorithm

Il Minimum Mean Cycle Cancelling Algorithm è un caso particolare del Cycle Cancelling Algorithm in cui a ogni iterazione il flusso è aumentato nel ciclo W di costo negativo del grafo $G(x)$ che ha *costo medio* minimo. Il costo medio di un ciclo W è dato da $(\sum_{(i,j)\in W} c_{ij})/|W|$.

Complessità computazionale

Per calcolare il ciclo W di costo medio minimo può essere impiegata una procedura di programmazione dinamica di complessità $O(nm)$.

Algoritmi Fortemente Polinomiali (2)

La complessità computazionale del Minimum Mean Cycle Cancelling Algorithm è pari a $O(n^2 m^3 \log n)$.

Enhanced Capacity Scaling Algorithm

L'Enhanced Capacity Scaling Algorithm è una variante del Capacity Scaling Algorithm ed è basato sul seguente lemma.

Lemma

Quando il Capacity Scaling Algorithm aggiorna il parametro Δ, i.e. $\Delta = \Delta/2$, se nell'arco (i,j) il flusso è tale che $x_{ij} \geq 8n\Delta$ allora:

- *in qualsiasi soluzione ottima $x_{ij} > 0$.*
- *il costo ridotto c_{ij}^{π} è sempre nullo comunque siano scelti i potenziali π ottimi.* $\square$

Algoritmi Fortemente Polinomiali (3)

Quando l'algoritmo identifica un arco (i,j) tale che $x_{ij} \geq 8n\Delta$, allora i potenziali dei nodi i e j possono essere fissati. Quindi, l'arco (i,j) viene "*contratto*" e i nodi i e j sono sostituiti da un nuovo nodo. L'operazione di contrazione da origine a un nuovo problema di flusso di costo minimo con un nodo in meno.

L'algoritmo non effettua le operazioni di contrazione esplicitamente ed è in grado di procedere senza dover ricalcolare i flussi e i potenziali per il nuovo problema ridotto.

Complessità computazionale

La complessità computazionale dell'Enhanced Capacity Scaling Algorithm è pari a $O((m \log n)(m + n \log n))$.

Network Simplex Method

- Il problema del flusso di costo minimo essendo un problema di programmazione lineare può essere risolto con il metodo del simplesso.
- Sfruttando delle caratteristiche strutturali del problema del flusso di costo minimo può essere implementato un metodo del simplesso "*ad hoc*" più efficiente del metodo standard.
- La formulazione del problema del flusso di costo minimo può essere scritta in forma matriciale come segue:

$$\begin{aligned} Min \quad & cx \\ s.t. \quad & Ax = b \\ & 0 \leq x \leq u \end{aligned}$$

- Si noti che ogni colonna di A corrisponde a una variabile x_{ij} e ogni sua riga corrisponde a un vincolo di conservazione del flusso.

Network Simplex Method (2)

- Il rango della matrice A è pari a $n-1$, quindi un vincolo è ridondante e $m-n+1$ variabili sono libere.
- Le colonne della matrice A possono essere partizionate in tre insiemi, i.e. $A = (B, L, U)$:

  B: variabili base, i.e. $0 \leq x_{ij} \leq u_{ij}$;
  L: variabili non base tali che $x_{ij} = 0$;
  U: variabili non base tali che $x_{ij} = u_{ij}$;

  Quindi, possiamo partizionare $x = (x_B, x_L, x_U)$ e $u = (u_B, u_L, u_U)$.
- La matrice B che rappresenta la base contiene $n-1$ colonne linearmente indipendenti.

Network Simplex Method (3)

- Due delle operazioni critiche, che ad ogni iterazione il metodo del simplesso deve svolgere, sono la determinazione della soluzione base ammissibile x_B e delle variabili duali π, risolvendo i seguenti sistemi lineari:

$$Bx_B = b - Uu_U$$

e

$$\pi B = c_B$$

- Mostreremo che la soluzione di questi sistemi lineari nel caso del problema del flusso di costo minimo è semplice.

Teorema

Le righe e le colonne della matrice B possono essere sempre riordinate in modo da ottenere una matrice triangolare inferiore. □

Network Simplex Method (4)

- Quindi i sistemi lineari in cui la matrice dei coefficienti è B possono essere risolti semplicemente con delle sostituzioni forward o backward.
- Inoltre, si può dimostrare che la matrice A è "*totalmente unimodulare*", quindi, se il problema di flusso di costo minimo ha capacità, domande e richieste intere, allora il flusso ottimo x sarà intero.

Strutture Spanning Tree

- Il Network Simplex Method si basa sul seguente teorema.

Teorema

Ogni spanning tree (albero di copertura) definito sul grafo G definisce una base B del corrispondente problema di flusso di costo minimo. Viceversa, ogni base B relativa a un problema di flusso di costo minimo definisce uno spanning tree sul grafo G. □

- Data una soluzione base ammissibile x, definiamo una "*Struttura Spanning Tree ammissibile*" (T, L, U) come segue:

  T: archi nello spanning tree;
  L: archi il cui flusso deve essere nullo;
  U: archi in cui il flusso deve essere pari alla capacita'.

Strutture Spanning Tree (2)

- Un arco (i,j) è "*libero*" se $0 < x_{ij} < u_{ij}$, mentre diremo che è "*vincolato*" se $x_{ij} = 0$ oppure $x_{ij} = u_{ij}$.
- Se tutti gli archi nello spanning tree sono liberi, allora diremo che lo spanning tree è *non degenere*, altrimenti diremo che è *degenere*.

Teorema

Una struttura spanning tree (T,L,U) è una struttura spanning tree ottima del problema di flusso di costo minimo se è ammissibile e per un qualche π i costi ridotti c_{ij}^{π} soddisfano le seguenti condizioni:

$$\begin{aligned} &\text{If } c_{ij}^{\pi} = 0, \quad \text{then } (i,j) \in T; \\ &\text{If } c_{ij}^{\pi} \geq 0, \quad \text{then } (i,j) \in L; \\ &\text{If } c_{ij}^{\pi} \leq 0, \quad \text{then } (i,j) \in U. \ \square \end{aligned}$$

Strutture Spanning Tree (3)

- Il metodo del simplesso verifica a ogni iterazione se esite un arco (i,j) che viola le condizioni di ottimalità. Se questo arco esiste inserisce (i,j) nello spanning tree T.
- Quando l'arco (i,j) viene inserito in T genera un ciclo W e l'insieme degli archi T non è più uno spanning tree.
- Per determinare la nuova base il metodo del simplesso seleziona un arco dal ciclo W e lo toglie da T.
- Per il momento supponiamo che la struttura spanning tree non sia degenere.

Algoritmo del Simplesso su Reti

algorithm network-simplex
begin
    Determina una struttura (T, L, U) ammissibile;
    Calcola il flusso x corrispondente a (T, L, U);
    Calcola i potenziali π corrispondenti a (T, L, U);
    while esiste un arco non ammissibile **do**
    begin
        Seleziona l'arco (k, l) non ammissibile;
        Aggiungi l'arco (k, l) a T e determina
            l'arco (p, q) uscente;
        Aggiorna la struttura (T, L, U);
        Aggiorna i potenziali π e il flusso x;
    end
end

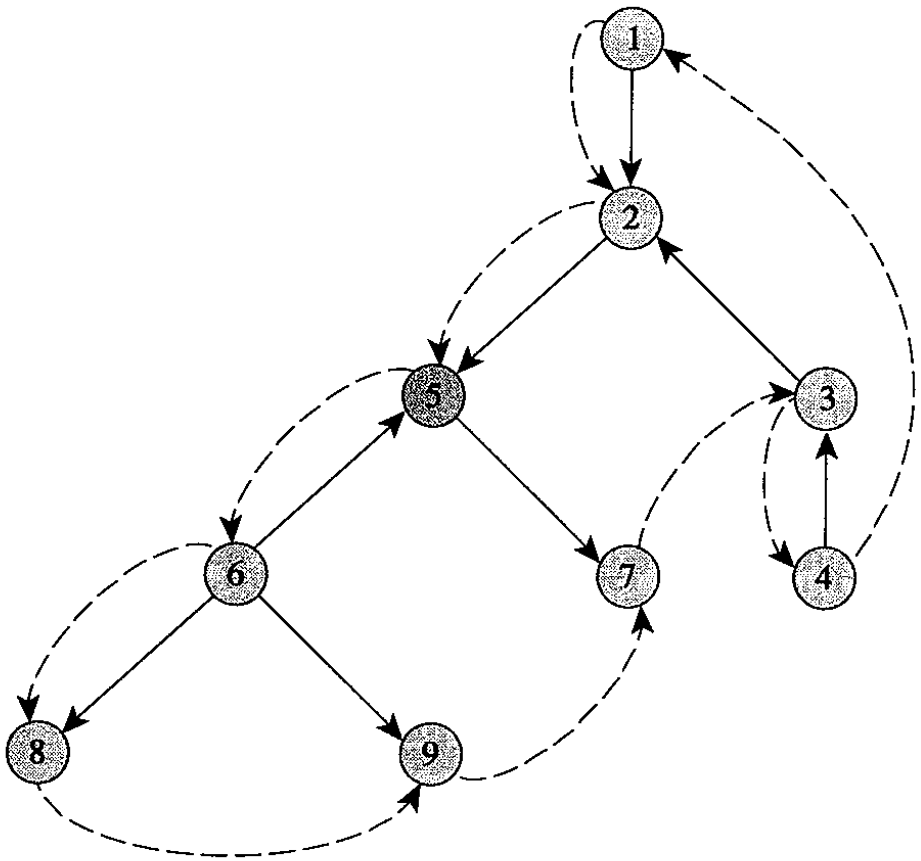
Implementazione della Struttura Spenning Tree

- Per implementare la struttura spanning tree definiamo in modo arbitrario un *nodo radice* e ad ogni nodo i possiamo associare le seguenti tre informazioni:
  - $pred(i)$: indica il nodo padre, ossia il nodo che succede a i nel cammino da i al nodo radice.
  - $depth(i)$: indica il numero di nodi presenti nel cammino da i al nodo radice.
  - $thread(i)$: indica il nodo che in una ricerca di tipo depth-first segue il nodo i.
- L'utilizzo dei vettori $pred(i)$, $depth(i)$ e $thread(i)$ consente di effettuare in modo efficiente le operazioni richieste dal simplesso.

Implementazione della Struttura Spenning Tree (2)

- Le operazioni richieste dal simplesso sono le seguenti:
  - Calcolo di un flusso x;
  - Calcolo dei potenziali π;
  - Inserimento di un nuovo arco in T, identificazione di un ciclo W e determinazione dell'arco uscente;
  - Aggiornamento del flusso x;
  - Aggiornamento dei potenziali π.

Implementazione della Struttura Spenning Tree (3)



(a)

i	1	2	3	4	5	6	7	8	9
pred (i)	0	1	2	3	2	5	5	6	6
depth (i)	0	1	2	3	2	3	3	4	4
thread (i)	2	5	4	1	6	8	3	9	7

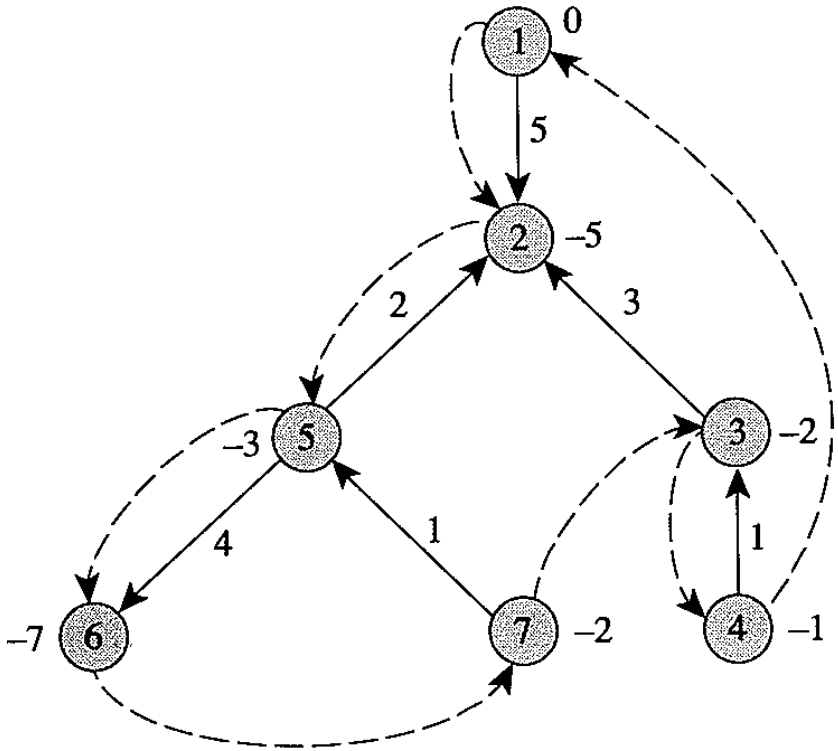
(b)

Figure 11.3 Example of a tree indices: (a) rooted tree; (b) corresponding tree indices.

Calcolo dei Potenziali

procedure compute-potentials
begin
    $\pi_1 = 0$;
    $j = thread(1)$;
    while $j \neq 1$ **do**
    begin
        $i = pred(j)$;
        if $(i,j) \in A$ **then** $\pi_j = \pi_i - c_{ij}$;
        if $(j,i) \in A$ **then** $\pi_j = \pi_i + c_{ji}$;
        $j = thread(j)$;
    end
end

Calcolo dei Potenziali (2)



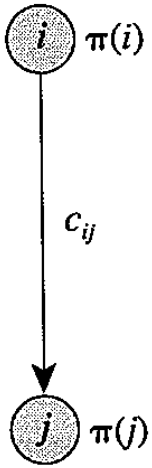


Figure 11.5 Computing node potentials for a spanning tree.

Calcolo del Flusso

procedure compute-flows
begin
  for each $i \in N$ **do** $b'_i = b_i$;
  for each $(i,j) \in L$ **do** $x_{ij} = 0$;
  for each $(i,j) \in U$ **do**
  begin
    $x_{ij} = u_{ij}$;
    $b'_i = b'_i - u_{ij}$;
    $b'_j = b'_j + u_{ij}$;
  end
  $T' = T$;
  while $T' \neq \emptyset$ **do**
  begin
    Seleziona una foglia j nel sottoalbero T';
    $i = pred(j)$;
    if $(i,j) \in T'$ **then** $x_{ij} = -b'_j$;
    else $x_{ji} = b'_j$;
    $b'_i = b'_i + b'_j$;
    Eliminare l'arco selezionato da T';
  end
end

Calcolo del Flusso (2)

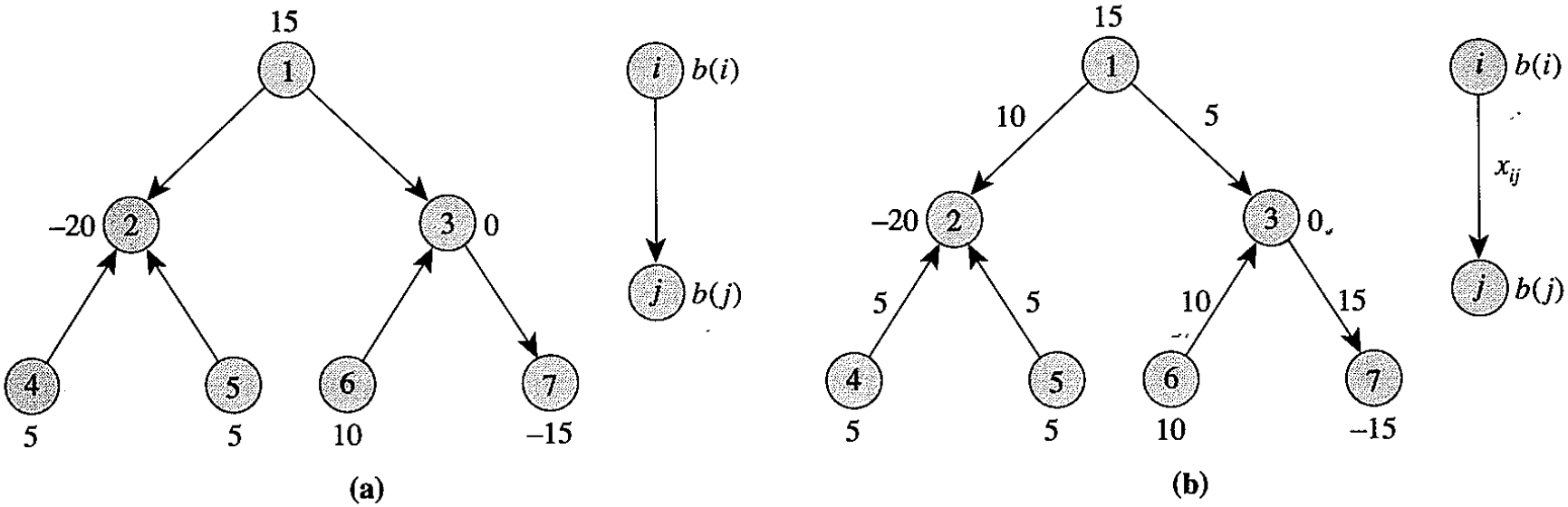


Figure 11.6 Computing flows for a spanning tree.

Identificazione del Ciclo

procedure identify-cycle(k,l,W)
begin
    $i = k$, $j = l$, $W = \{(k,l)\}$;
    while $i \neq j$ **do**
    begin
        if $depth(i) > depth(j)$ **then**
            Aggiungi in W l'arco di estremi i e $pred(i)$;
            $i = pred(i)$;
        else if $depth(j) > depth(i)$ **then**
            Aggiungi in W l'arco di estremi j e $pred(j)$;
            $j = pred(j)$;
        else
            Aggiungi in W l'arco di estremi i e $pred(i)$;
            $i = pred(i)$;
            Aggiungi in W l'arco di estremi j e $pred(j)$;
            $j = pred(j)$;
    end
end

Identificazione del Ciclo (2)

- Nella procedura *identify-cycle* potrà anche essere identificato l'arco (p, q) uscente e calcolato il quantitativo δ con cui aumentare il flusso nel ciclo W.

- Il ciclo W è detto "*pivot cycle*" e, dato l'arco entrante (k, l), è orientato nella direzione:
  - dell'arco (k, l) se $(k, l) \in L$;
  - opposta a quella dell'arco (k, l) se $(k, l) \in U$.

- Denotiamo con $\overline{W} \subseteq W$ l'insieme degli archi *forward* (quelli con lo stesso orientamento del ciclo W) e con $\underline{W} \subseteq W$ l'insieme degli archi *backward* (quelli con orientamento opposto rispetto a quello del ciclo W).

Identificazione del Ciclo (3)

- Il quantitativo δ con cui aumentare il flusso nel ciclo W è dato da:

$$\delta = min\{\delta_{ij} : (i,j) \in W\}$$

dove

$$\delta_{ij} = \begin{cases} u_{ij} - x_{ij}, & \text{if } (i,j) \in \overline{W} \\ x_{ij}, & \text{if } (i,j) \in \underline{W} \end{cases}$$

- Gli archi $(i,j) \in W$ tali che $\delta_{ij} = \delta$ sono detti "*blocking arc*".
- L'arco uscente $(p,q) \in W$ è scelto tra i blocking arc.

Identificazione del Ciclo (4)

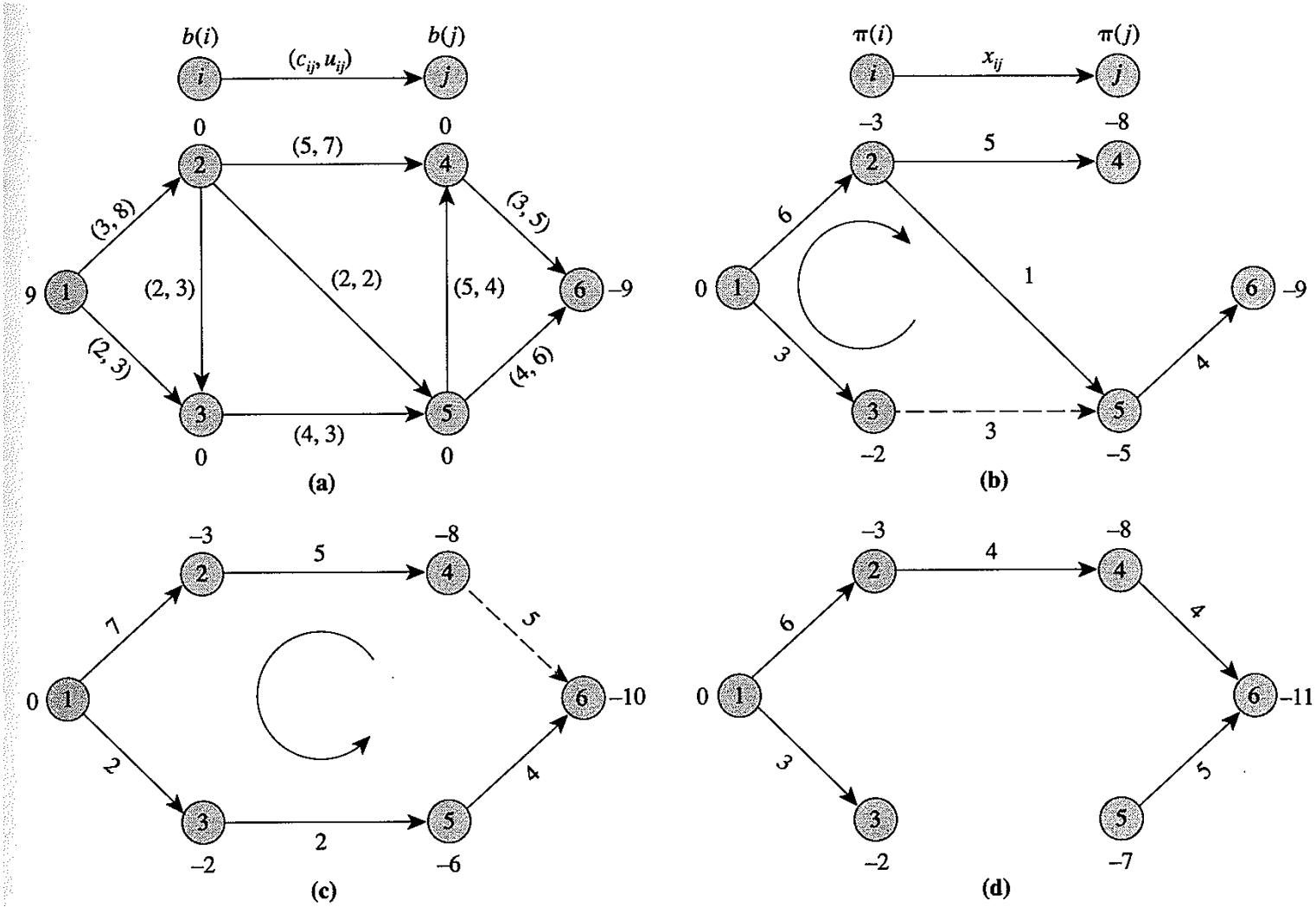


Figure 11.11 Numerical example for the network simplex algorithm.

Aggiornamento dei Potenziali

procedure update-potentials(k,l,p,q)
begin
    if $q \in T_2$ **then** $y = q$ **else** $y = p$;
    if $k \in T_1$ **then** $\gamma = -c^{\pi}_{kl}$ **else** $\gamma = c^{\pi}_{kl}$;
    $\pi_y = \pi_y + \gamma$;
    $z = thred(y)$;
    while $depth(z) > depth(y)$ **do**
    begin
        $\pi_z = \pi_z + \gamma$;
        $z = thred(z)$;
    end
end

Spanning Tree Fortemente Ammissibili

- Finora abbiamo fatto l'ipotesi che la struttura spanning tree fosse non degenere, ma studi computazionali hanno dimostrato che mediamente il 90% delle operazioni di pivot sono degeneri.
- La presenza di pivot degeneri potrebbe impedire la convergenza alla soluzione ottima (*degenerazione ciclante*).
- Per evitare la degenerazione ciclante è sufficiente che la struttura spanning tree sia "*fortemente ammissibile*".
- Una struttura spanning tree è fortemente ammissibile se ogni arco $(i,j) \in T$ è diretto:
  - verso la radice se $x_{ij} = 0$;
  - in direzione opposta rispetto la radice se $x_{ij} = u_{ij}$.

Spanning Tree Fortemente Ammissibili (2)

- Equivalentemente, diremo che una struttura spanning tree è fortemente ammissibile se possiamo sempre inviare una quantità positiva di flusso da un qualsiasi nodo alla radice attraverso i soli archi appartenenti allo spanning tree senza violare i vincoli di capacità.

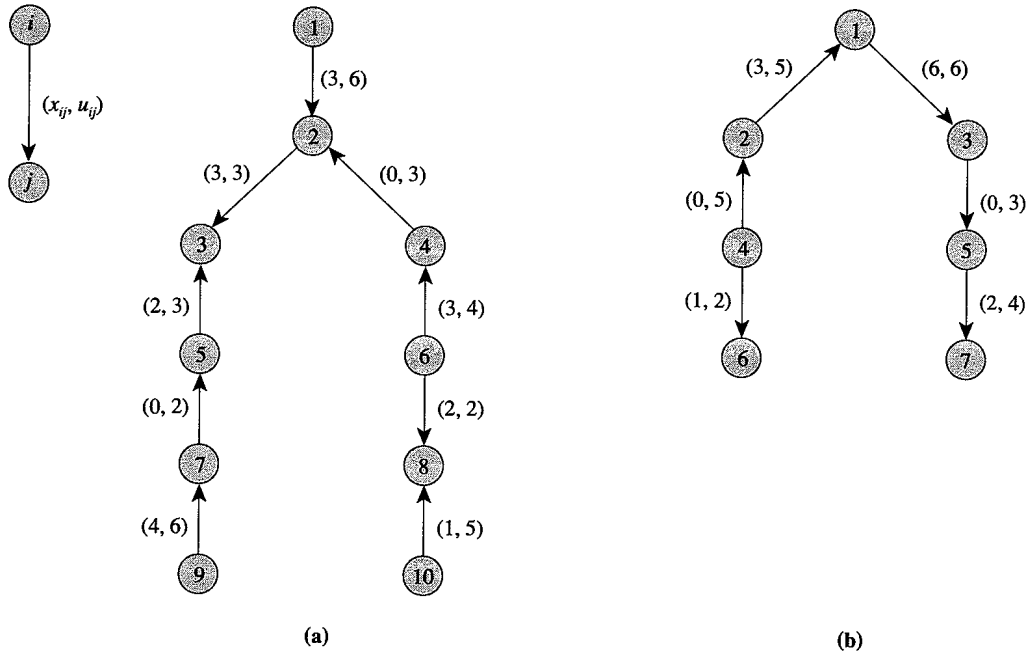


Figure 11.12 Feasible spanning trees: (a) strongly feasible; (b) nonstrongly feasible.

Regola di selezione dell'arco uscente

- Dato l'arco entrante (k, l) e il corrispondente pivot cycle W, definiamo "*apice*" w il vertice in W che è predecessore sia di k che di l.
- Regola: L'arco uscente è l'ultimo blocking arc incontrato attraversando il pivot cycle W partendo dall'apice w e seguendo il suo orientamento.
- Se lo spanning tree iniziale è fortemente ammissibile, applicando questa semplice regola si ha la garanzia di mantenere lo spanning tree fortemente ammissibile e di evitare la degenerazione ciclante.

Regola di selezione dell'arco uscente (2)

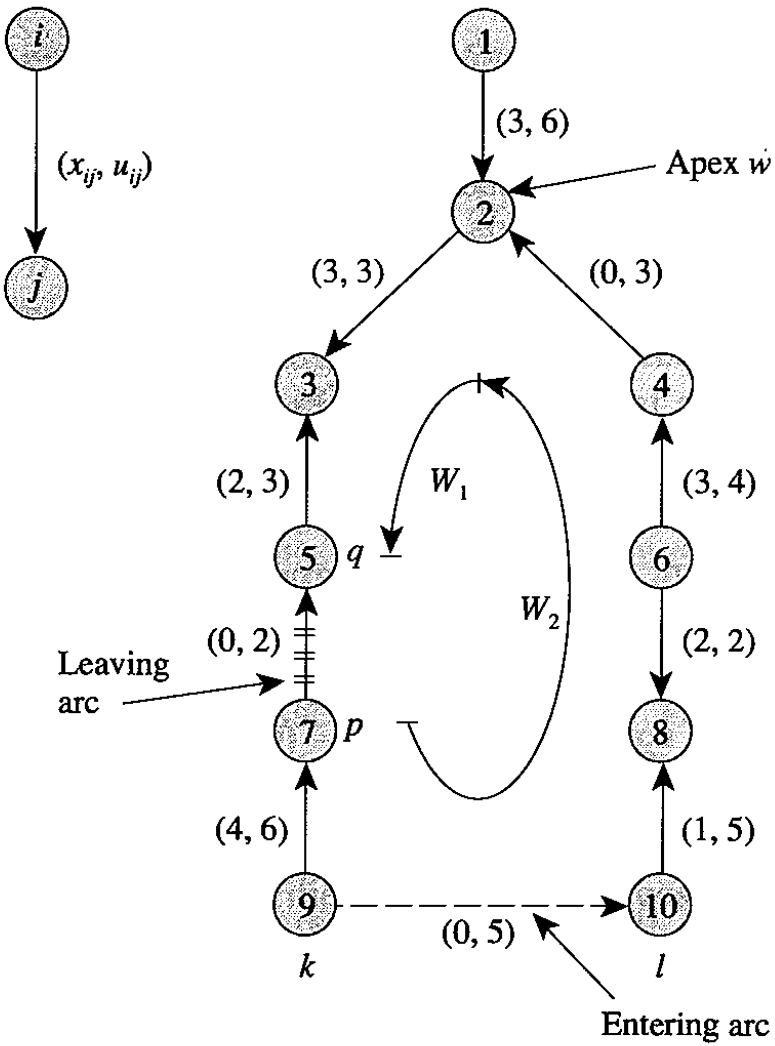


Figure 11.13 Selecting the leaving arc.