
Munich Personal RePEc Archive

Maximum flow - minimum cost algorithms

Parrini, Alessandro

Università di Firenze - Dipartimento di Matematica per le Decisioni

8 October 2009

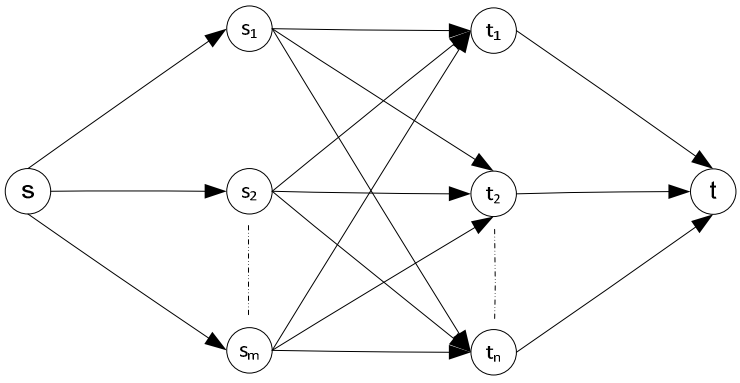
Online at https://mpra.ub.uni-muenchen.de/39759/
MPRA Paper No. 39759, posted 03 Jul 2012 06:26 UTC

Indice

1. Introduzione

Il problema di flusso massimo al minimo costo, racchiudendo una classe molto ampia di applicazioni, ha assunto una posizione centrale tra i modelli di ottimizzazione su rete. Per capire bene di cosa si tratta, ne presentiamo subito una.

Si consideri una azienda con m stabilimenti indipendenti in grado di produrre lo stesso tipo di prodotto ma a costi c_i differenti; tale prodotto viene richiesto da n clienti. Ogni mese la i-esima fabbrica può produrre a_i unità di prodotto a fronte di una richiesta di b_j unità del j-esimo cliente. Dalla fabbrica i al cliente j si possono trasportare ogni mese al più d_{ij} unità, e il costo da sostenere per ogni unità di prodotto trasportata è pari a c_{ij}. Si assume che $\sum_j b_j \leq \sum_i a_i$ ossia che la domanda dei clienti non superi l'offerta dell'azienda. È possibile soddisfare la richiesta dei clienti? E se sì, si può minimizzare il costo complessivo di produzione e trasporto?
Un modo per rispondere a questa domanda in modo razionale è quello di modellare il problema costruendo una *rete* come la seguente:



dove s_i è lo stabilimento i-esimo e t_j il j-esimo cliente; s e t sono invece vertici fittizi: s è un "superstabilimento" pensato come in grado di distribuire le materie prime agli stabilimenti $s_1, \dots, s_m$ che poi le trasformeranno nel prodotto con un certo costo c_i; t è un "supercliente" che raccoglie la domanda dei clienti $t_1, \dots, t_n$.

Questa rappresentazione delle reti fu utilizzata da Kirchhoff tra il 1845 e il 1850 per sviluppare le leggi che governano le reti elettriche. Nella seconda metà del secolo scorso esse sono poi state utilizzate come modello matematico in numerosi contesti: dai flussi di petrolio negli oleodotti all'ottimizzazione del traffico stradale, dalle strategie di investimento ai problemi di cammino minimo.

Assegniamo ora una funzione *capacità* $u(a)$, $u : A \to \mathbb{N}$ sugli archi della rete. Ad ogni *arco* uscente dal superstabilimento s si assegna la capacità a_i, ossia la quantità di output che ogni mese la fabbrica s_i può produrre. Le capacità degli archi (s_i, t_j) sono invece pari alla quantità di prodotto d_{ij} che lo stabilimento i è in grado di far arrivare al cliente j. Infine, le capacità degli archi che collegano i *nodi* t_j al supercliente t sono definite come la richiesta b_j da soddisfare. La richiesta dei clienti è soddisfatta se esiste una assegnazione x_{ij}, $x : A \to \mathbb{R}$, detta *flusso* (che rappresenta la quantità di bene inviato sugli archi) tale che sugli archi (t_j, t) il flusso sia pari alla loro capacità.

Quello appena descritta è un tipico contesto del problema di ***flusso massimo***, detto "problema di trasporto ottimo": per risolverlo è possibile ricorrere ad algoritmi di ottimizzazione su rete come l'algoritmo dei "*cammini aumentanti*", proposto da Ford e Fulkerson nel 1956. Esso prende in considerazione un flusso lungo una rete con capacità limitate e permette di ottenere un flusso di *valore massimo*. Con l'algoritmo dei cammini aumentanti siamo in grado di determinare se il valore massimo raggiungibile dal flusso nella rete soddisfa la richiesta dei clienti e quanta produzione è effettivamente necessaria.

Tuttavia, poiché i costi di trasporto e di produzione costituiscono una spesa rilevante per una azienda, è importante ridurli quanto più possibile. Pertanto non sarà sufficiente massimizzare il flusso ma occorrerà contemporaneamente minimizzare i costi (per esempio, i kilometri percorsi per trasportare la merce). Un algoritmo che permette di trovare un ***cammino diretto*** **di** ***costo minimo*** su una rete è, ad esempio, l'algoritmo dei "*cammini minimi*" di Dijkstra (1959).

Se su una rete sono assegnate le funzioni di capacità e di costo, possiamo chiederci in generale quale flusso di valore massimo realizzi il costo minimo cercando il cosiddetto "***flusso massimo al minimo costo***".

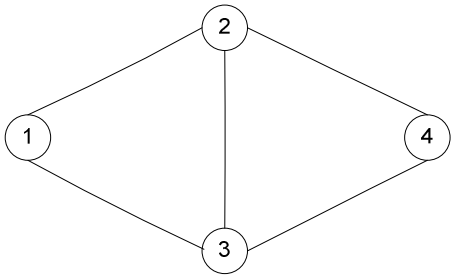
La ricerca di questo particolare flusso può essere ottenuta interfacciando l'algoritmo di flusso massimo con uno di cammino minimo: ciò condurrà alla nascita di due nuovi algoritmi il "***cycle-cancelling algorithm"*** e il "***successive shortest path algorithm***".

Un confronto tra questi risulterà interessante, in particolare relativamente al diverso modo di interfacciare gli algoritmi di Ford-Fulkerson e Dijkstra, in un caso "in serie" e nell'altro "in parallelo".

2. Definizioni

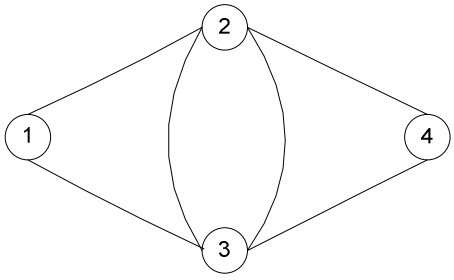
2.1 Grafi

Un ***grafo semplice*** è una coppia di insiemi $G = (V, E)$ dove $\mathrm{V} \neq \emptyset$ è l'insieme dei *vertici* (o *nodi*) ed $E \subseteq \{\{i, j\} \mid i, j \in V, i \neq j\}$ è l'insieme dei *lati*. Si noti che non sono ammessi lati che iniziano e terminano nello stesso vertice ("cappi" o "loops") giacché $i \neq j$ e che è ammesso al più un lato per ogni coppia di vertici. Tratteremo sempre con *grafi finiti* ossia supporremo $|V| < +\infty$.
Geometricamente i grafi si rappresentano mediante cerchi e linee:



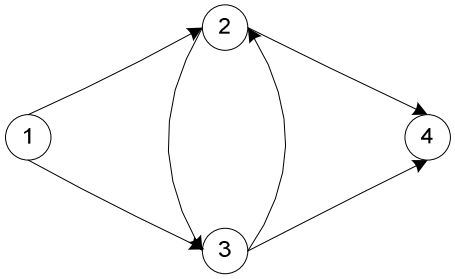
(a) Grafo semplice

Quando si ammettono più lati tra la stessa coppia di vertici si parla di *multigrafi* (la definizione formale risulta più complessa, ma ai nostri scopi è sufficiente questa idea intuitiva).



(b) Multigrafo

Assegnando ai lati una direzione, ossia definendo coppie ordinate di vertici, ci poniamo nell'ambito dei ***d-grafi*** (*grafi orientati* o *diretti*) e invece che di lati si parla di *archi*. Un d-grafo è quindi una coppia $G = (V, A)$ i cui $A \subseteq \{(i,j) \mid i,j \in \mathrm{V},\ i \neq j\}$.



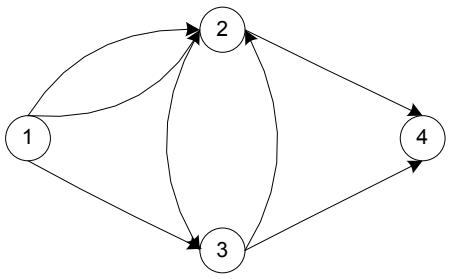
(c) Grafo orientato

Se $e = \{i,j\}$ è un lato di G, si dice che i e j sono gli *estremi* di e.
Un vertice $i \in V$ e un lato $e \in E$ si dicono *incidenti* se i è un estremo di e.
i e $j \in V$ si dicono *adiacenti* se $\{i,j\} \in E$.
Due lati si dicono *consecutivi* se hanno esattamente un vertice in comune.

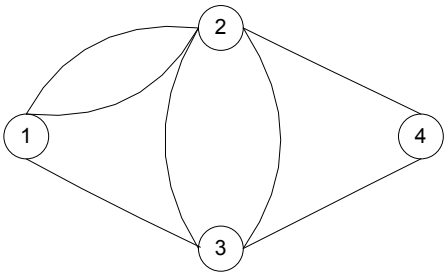
In un contesto orientato, due vertici i e j si dicono adiacenti se $(i,j) \in A$ o $(j,i) \in A$. Inoltre se $a = (i,j) \in A$ allora chiamiamo i il "primo estremo" di a e j il "secondo estremo" di a: in figura (c) il vertice 2 è il primo estremo dell'arco $(2,3)$ ed è il secondo estremo dell'arco $(3,2)$.

Se i_0 è un vertice fissato, indichiamo con $A(i_0)$ l'insieme di tutti gli archi di cui i_0 è il primo estremo (archi "uscenti" da i_0), ossia $A(i_0) = \{(i_0,j) \mid (i_0,j) \in A\}$.

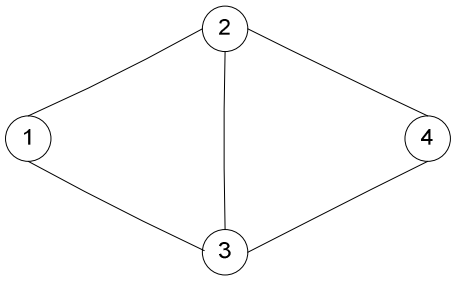
Osservazione. Un multigrafo diretto G definisce in modo naturale un multigrafo non orientato (cancellando i sensi di percorrenza sugli archi) ed anche un grafo semplice, ottenuto passando dalla coppia (i,j) al sottoinsieme $\{i,j\}$.



(d) Multigrafo diretto G



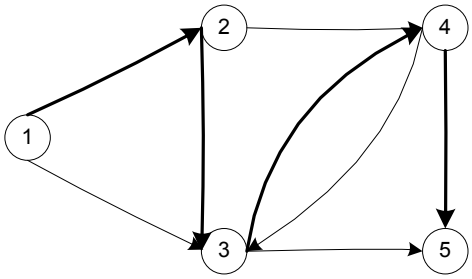
(e) Multigrafo non diretto associato a G



(f) Grafo semplice associato a G

2.2 Cammini

Sia $G = (V, A)$ un grafo diretto. Un ***cammino diretto*** P (da "path") dal vertice i_0 al vertice i_n è una sequenza di vertici e archi $[i_0, a_1, i_1, \dots, a_n, i_n]$ con i_k distinti e $a_k = (i_{k-1}, i_k)$: in altre parole è un percorso compiuto sugli archi di G seguendo l'orientamento indicato dalle frecce; pertanto un cammino diretto è sempre identificato senza ambiguità dal solo elenco dei vertici $[i_0, i_1, \dots, i_n]$ (si noti che tale affermazione non è vera se G è un multigrafo).



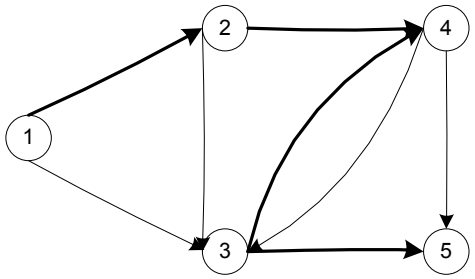
(g) Cammino diretto $P = [1, 2, 3, 4, 5]$

Se si consente $i_0 = i_n$ si parla di ***ciclo diretto***.

Un ***cammino*** P da i_0 a $i_n \in V$ è una sequenza di vertici e archi $[i_0, a_1, i_1, \dots, a_n, i_n]$ con i_k distinti e $a_k = (i_{k-1}, i_k) \in A$ oppure $a_k = (i_k, i_{k-1}) \in A$. Nel primo caso

diremo che il cammino P percorre l'arco a_k in modo "concorde" e nel secondo in modo "discorde".

Ad esempio il cammino in figura (h) da $i_0 = 1$ a $i_n = 5$ ha un unico arco discorde $(3,4)$.



(h) Cammino $P = [1, 2, 4, (3,4), 3, 5]$

Un grafo G si dice *connesso* se $\forall\ i \neq j \in V$ esiste un cammino da i a j.

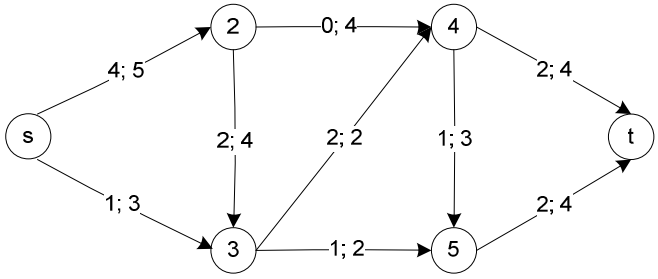
2.3 Reti elementari

Una ***rete elementare*** $\mathcal{R} = (V, A, s, t)$ è un grafo diretto e connesso in cui sono evidenziati due vertici distinti: una sorgente s e un termine t; sia $n = |V|$ e $m = |A|$.

Assumiamo che sugli archi di $\mathcal{R}$ siano assegnate le seguenti funzioni:

- la funzione ***capacità***, $u : A \to \mathbb{Z}_+$ che associa ad ogni arco $a = (i, j) \in A$ una capacità positiva o nulla u_{ij}; gli archi con capacità nulla vengono usualmente estromessi dalla rete.
- la funzione ***costo***, $c : A \to \mathbb{R}$ che associa ad ogni arco $a = (i, j) \in A$ un costo (o una lunghezza) c_{ij} che si deve sostenere per percorrerlo. Essendo la funzione a valori reali, è possibile che alcuni archi abbiano costo negativo, ossia che percorrendoli si ottenga un "risparmio".

In genere vengono indicati sugli archi i costi separati dalle capacità da ";" come in figura (i).



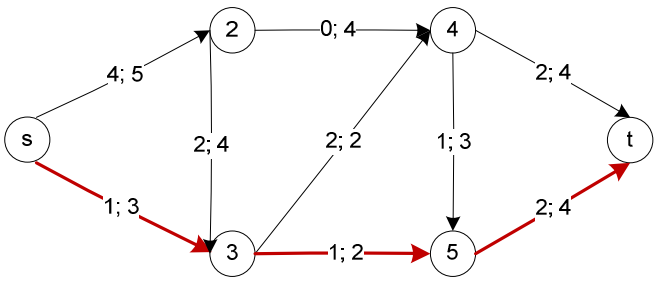
(i) Rete $\mathcal{R} = (V, A, s, t)$ con costi e capacità

2.4 Cammini minimi

Data una rete elementare $\mathcal{R} = (V, A, s, t)$, per ogni cammino diretto P da s a t si dice *costo* (o *lunghezza*) di P il numero reale $c(P) \coloneqq \sum_{(i,j)\in P} c_{ij}$.

Un ciclo diretto W per cui $c(W) \coloneqq \sum_{(i,j)\in W} c_{ij} < 0$ è chiamato ***ciclo negativo***.

Si dice ***distanza*** da s a t il minimo della funzione $c(P)$ al variare di P fra tutti i possibili cammini diretti da s a t. Si noti che l'esistenza di tale minimo è garantita dal fatto che essendo in un contesto di grafi finiti, è finito anche il numero dei possibili cammini da s a t. Un cammino P che realizza la distanza è detto ***cammino minimo***.



(j) $P = [s, 3, 5, t]$ è un cammino minimo di lunghezza 4

Se non esiste un cammino diretto da s a t si dice che la distanza tra s e t è $+\infty$.

Chiamiamo ***potenziale*** di un vertice i la distanza tra la sorgente s e il vertice i e lo indichiamo con $p(i)$. Per esempio il vertice 3 in figura (j) ha potenziale $p(3) = 1$.

In generale un cammino diretto $P = [i_0, i_1, \dots, i_n]$ da i_0 a i_n può essere rappresentato usando gli *indici predecessori* $pred(i_1) = i_0, pred(i_2) = i_1, \dots, pred\ (i_n) = i_{n-1}$. Ad esempio, il cammino minimo in figura (j) si può scrivere come: $pred(3) = s, pred(5) = 3, pred\ (t) = 5$.

Osservazione. Se $P = [s, i_1, \dots, i_n]$ è un cammino minimo da s a i_n, allora $Q = [s, i_1, \dots, i_r]$ è un cammino minimo da s a i_r , $\forall\ 1 \leq r \leq n$.

Illustriamo ora un algoritmo per la ricerca di cammini minimi su una rete elementare. L'algoritmo di Dijkstra citato nella introduzione, non è adeguato a reti con assegnati sugli archi costi arbitrari: esso si fonda infatti sull' idea che i costi sono crescenti rispetto al numero di archi utilizzati e pertanto funziona solo se se la funzione di costo è del tipo $c : A \to \mathbb{R}_+$. In questo caso l'algoritmo dei cammini minimi esamina un vertice una volta sola, assegnandogli il potenziale che rappresenta la distanza tra la sorgente e il vertice stesso. Poiché sulla rete si ammettono anche costi negativi, dobbiamo servirci di un algoritmo più "tecnico" come quello riportato a pagina 11, che denomineremo "***generalized shortest path algorithm***".

GENERALIZED SHORTEST PATH ALGORITHM:

input: rete elementare $\mathcal{R} = (V, A)$ con una sorgente s
e una funzione di costo (o lunghezza) a valori in $\mathbb{R}$

output: cammino minimo P da s a un qualunque vertice $j \in V$ e la sua lunghezza

start:
$p(s) \coloneqq 0$; $pred(s) \coloneqq \emptyset$
$p(j) \coloneqq +\infty \ \forall\, j \in V - \{s\}$

while qualche arco $(i,j) \in A$ soddisfa $p(j) > p(i) + c_{ij}$ **do**
for ogni $(i,j) \in A(i)$ e ogni $i \in V$
if $p(j) > p(i) + c_{ij}$ **then**
$p(j) \coloneqq p(i) + c_{ij}$
$pred(j) \coloneqq i$
endwhile

Come nell'algoritmo di Dijkstra si inizia settando il potenziale della sorgente a 0 e tutti gli altri a $+\infty$. I primi archi considerati saranno quindi gli $(s,j) \in A$ in quanto $+\infty = p(j) > p(s) + c_{sj} = c_{sj}$ e quindi si aggiornano i potenziali ponendo $p(j) = c_{sj}$. Si aggiornerà poi l'indice predecessore di j ponendo $pred(j) = s$. Per ogni vertice $i \in V$, l'output fornirà quindi anche il potenziale e il relativo indice predecessore.

Una volta terminato l'algoritmo, dalla lista degli indici predecessori si può facilmente risalire ad un cammino minimo P dalla sorgente ad un qualunque altro vertice.

La differenza con l'algoritmo di Dijkstra, come detto, sta nel fatto che un vertice $i \in V$ non viene esaminato una sola volta, giacchè il potenziale $p(i)$ non è più crescente rispetto al numero degli archi utilizzati.

2.5 Flussi

Un ***flusso*** è una funzione definita sugli archi di una rete elementare $\mathcal{R} = (V, A, s, t)$, $x : A \to \mathbb{R}$, $x = (x_{ij})$, ossia una funzione che associa ad ogni arco un numero reale: $x : (i, j) \mapsto x_{ij}$.

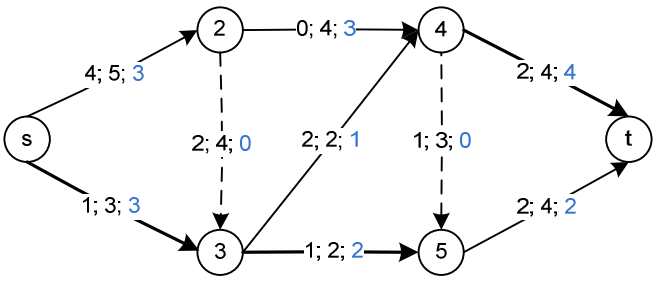
Un flusso si dice ***ammissibile*** se soddisfa i seguenti vincoli:

- *Legge di conservazione*:
  $\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = 0 \qquad \forall\, i \in V - \{s, t\}$

- *Condizione di compatibilità*:
  $0 \leq x_{ij} \leq u_{ij} \qquad \forall\, (i, j) \in A$

La legge di conservazione del flusso, indica che il flusso totale entrante in ogni nodo deve essere uguale a quello totale uscente dallo stesso nodo, fatta eccezione per la sorgente e per il termine. La condizione di compatibilità impone per un flusso ammissibile x_{ij} zero come limite inferiore e la capacità dell'arco come limite superiore.

Si pensi al caso del passaggio di petrolio da un oleodotto: le capacità di un arco sono il massimo ammontare di petrolio per unità di tempo che può passare dalla conduttura; la legge di conservazione indica che in nessun nodo si può aggiungere o prelevare petrolio.



(k) In blu, un flusso ammissibile x

Un arco $(i,j) \in A$ si dice *saturo* se $x_{ij} = u_{ij}$. Gli archi $(s,3),(3,5)$ e $(4,t)$ in figura (l) sono saturi; in genere si evidenziano in grassetto.

Un arco $(i,j) \in A$ si dice *scarico* se $x_{ij} = 0$. In (k) sono scarichi gli archi $(2,3)$ e $(4,5)$; essi vengono usualmente tratteggiati.

Osservazione. Un flusso ammissibile esiste sempre: è la funzione identicamente nulla $x = 0$.

Definiamo ora il ***valore del flusso*** come

$$v(x) := \sum_{\{j:(s,j)\in A\}} x_{sj} - \sum_{\{i:(i,s)\in A\}} x_{is}$$

ossia la somma del flusso sugli archi uscenti da s meno la somma del flusso su quelli entranti in s.
Si dimostra che se x è ammissibile allora $v(x) = \sum_{\{i:(i,t)\in A\}} x_{it} - \sum_{\{j:(t,j)\in A\}} x_{tj}$ ossia il "flusso netto" entrante in t è uguale al "flusso netto" uscente da s.

Il flusso x in figura (k) ha valore $v(x) = 6$.

Nel seguito considereremo la situazione più frequente, in cui
$\sum_{\{i:(i,s)\in A\}} x_{is} = \sum_{\{j:(t,j)\in A\}} x_{tj} = 0$ ossia nulla refluisce in s e nulla si disperde da t.

Un flusso x si dice ***massimo*** se realizza il massimo valore per $v(x)$.
La realizzazione di tale massimo ha interesse in numerosi contesti applicativi, come ad esempio il problema di trasporto ottimo accennato nella introduzione, che ha soluzione se e solo se esiste un flusso massimo che satura gli archi che collegano i clienti al supercliente.

Un ***taglio*** S che separa la sorgente dal termine, è un sottoinsieme di V tale che $s \in S, t \in S^c = V - S$. La capacità di un taglio è data dal naturale

$$u(S) := \sum_{\{(i,j):i\in S, j\in S^c\}} u_{ij}$$

ossia la somma delle capacità degli archi che hanno come primo estremo un vertice in S e come secondo estremo un vertice in S^c.

Lemma. Sia S un taglio su $\mathcal{R}$; allora per ogni flusso x risulta $v(x) = \sum_{\{(i,j):i\in S, j\in S^c\}} x_{ij} - \sum_{\{(j,i):j\in S^c, i\in S\}} x_{ji}$ (valore del flusso ottenuto lavorando solo sugli archi entranti ed uscenti dal taglio).

Si noti che per $S = \{s\}$ si ritrova la definizione di $v(x)$.

Lemma. Il valore di un flusso non supera mai la capacità di un taglio: $v(x) \leq u(S)$, per ogni flusso x e per ogni taglio S.

L'algoritmo di Ford e Fulkerson citato nella introduzione, permette di trovare uno specifico flusso il cui valore è esattamente uguale alla capacità di uno specifico taglio: per il lemma precedente tale flusso è massimo e la capacità del taglio corrispondente è necessariamente minima. Inoltre, se le capacità sono naturali, allora tale valore massimo è anch'esso naturale. La costruzione del flusso massimo tramite l'algoritmo è ottenibile mediante l'utilizzo dei "cammini aumentanti" per definire i quali ci serviamo della cosiddetta rete residuale.

Dato un flusso x per $\mathcal{R} = (G = (V, A), s, t)$, una ***rete residuale*** (o *grafo incrementale*) $G(x)$ è una rete che ha come insieme dei vertici V e come archi:

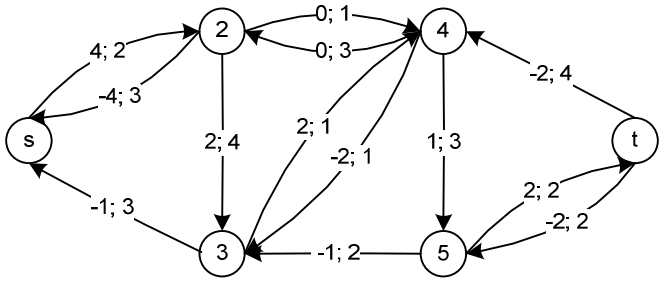
- $\forall\, a = (i,j) \in A$ in $\mathcal{R}$ con $x_{ij} < u_{ij}$ si ha in $G(x)$ un arco concorde $a \coloneqq (i,j)$ con ***capacità residua*** $r_{ij} \coloneqq u_{ij} - x_{ij}$ e costo $c^r_{ji} \coloneqq c_{ij}$

- $\forall\, a = (i,j) \in A$ in $\mathcal{R}$ con $x_{ij} > 0$ si ha in $G(x)$ un arco discorde $a^{-1} \coloneqq (j,i)$ con capacità residua $r_{ji} \coloneqq x_{ij}$ e costo $c^r_{ij} = -c_{ij}$

La capacità residua r_{ij} per l'arco $(i,j) \in A$ rappresenta il massimo flusso addizionale che si può inviare da i e j attraverso l'arco (i,j). Infatti se il flusso sull'arco (i,j) è tale che $x_{ij} < u_{ij}$ allora è ancora possibile aumentare il flusso fino al più a raggiungere la capacità dell'arco (i,j); d'altra parte se $x_{ij} > 0$ sarà possibile diminuire il flusso al più di x_{ij} unità, pertanto si genera in $G(x)$ un arco di ritorno/diminuzione orientato in senso opposto.

Si noti che il costo cambiato di segno allude al fatto seguente: se far passare una unità di flusso in un certo verso costa c, cancellare questo flusso fa risparmiare c, ossia costa – c.

Si noti inoltre che nella rete residuale gli archi hanno tutti capacità residua positiva: in particolare (i,j) non compare in $G(x)$ se è saturo in $\mathcal{R}$, (j,i) non vi compare se (i,j) è scarico. In figura (l) è riportata la rete residuale $G(x)$ per il flusso ammissibile x prima considerato.

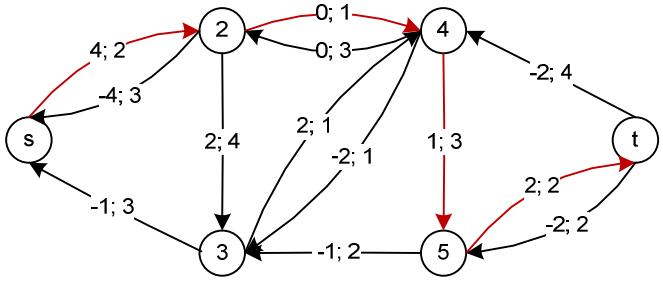


(l) Rete residuale per il flusso in figura (k)

Notiamo che se $0 < x_{ij} < u_{ij}$ e $a \in A$, allora $G(x)$ contiene entrambi gli archi a e a^{-1}: se nella rete di partenza $\mathcal{R}$ si hanno sia (i,j) che (j,i), il numero di archi in $G(x)$ tra i e j sarà quindi compreso tra un minimo di due e un massimo di quattro (due coppie di archi dello stesso verso).

Un cammino diretto da s a t in $G(x)$ è detto ***cammino aumentante***.

Se P è un ciclo in $G(x)$, ossia è tale che il vertice di partenza e di arrivo coincidono, esso è chiamato *ciclo aumentante*.



(m) Cammino aumentante $P = [s, 2, 4, 5, t]$

Indichiamo con δ la ***capacità di un cammino aumentante*** P data da $\delta = \min \{r_{ij} \mid (i,j) \in P\}$. Si noti che $\delta > 0$, in accordo con la nostra costruzione di $G(x)$. In figura (m) la capacità del cammino aumentante $P = [s, 2, 4, 5, t]$ evidenziato in rosso sarà $\delta = \min \{2; 1; 3; 2\} = 1$.

Sulla rete G si può ora assegnare un nuovo flusso x_1 ponendo $x_1 = x + \delta$ sugli archi di P concordi, e $x_1 = x - \delta$ sugli archi discordi. Dal flusso x_1 così ottenuto si potrà costruire una nuova rete residuale $G(x_1)$ sulla quale cercare un nuovo cammino aumentante.

Proposizione (corollario del *maxflow-mincut theorem*, Ford & Fulkerson). Un flusso x è massimo se e solo se il grafo incrementale $G(x)$ non contiene cammini diretti da s a t.

Si noti che il grafo incrementale è dinamico: via via che incremento il flusso lungo i cammini aumentanti, $G(x)$ si modifica; in particolare aumentando un flusso lungo un cammino diretto P di δ, le capacità residue r_{ij} degli archi di P decrescono di δ e le capacità r_{ji} degli archi di verso opposto aumentano sempre di δ unità.

Per risalire dall'ultima rete residuale (quella priva di cammini diretti) al flusso (ottimo) sulla rete originaria $\mathcal{R}$ si può seguire il seguente procedimento:

- Se tra i e j vi è solo l'arco (i,j) di costo $c_{ij}^r \geq 0$ allora il flusso x_{ij} in $\mathcal{R}$ è nullo
- Se tra i e j vi è solo l'arco (j,i) di costo $c_{ji}^r = -c_{ij} \leq 0$ allora l'arco (i,j) in $\mathcal{R}$ è saturo ($x_{ij} = u_{ij}$)
- Se tra i e j vi sono entrambi gli archi con costi $c_{ij}^r \geq 0$ e $c_{ji}^r = -c_{ij} \leq 0$ allora si guarda la capacità residua dell'arco (j,i), e il corrispondente arco (i,j) in $\mathcal{R}$ avrà $x_{ij} = r_{ji}$.

Un problema sorge qualora $c_{ij} = 0$ poiché non è più possibile riconoscere se l'arco è concorde o discorde, o meglio è possibile ma solo facendo riferimento alla rete di partenza $\mathcal{R}$: si sceglierà l'arco di verso opposto a quello in $\mathcal{R}$.

Osservazione. Se su una rete $\mathcal{R}^*$ le capacità u_{ij} sono espresse da valori in $\mathbb{Q}_+$ si può passare ad $\mathcal{R}$ elementare semplicemente moltiplicando ogni capacità per un opportuno identico fattore d (minimo comune denominatore tra quelli presenti): una volta trovato il flusso massimo x, si risalirà a $x^* = x/d$ flusso massimo per $\mathcal{R}^*$.

2.6 Flussi massimi al minimo costo

Sia $\mathcal{R} = (V, A, u_{ij}, c_{ij})$ una rete elementare con assegnate sugli archi le funzioni di costo e di capacità; sia al solito $n = |V|$ e $m = |A|$.
Il costo c_{ij} va interpretato come il prezzo da sostenere per inviare una unità di flusso sull'arco (i, j). In molte applicazioni tale assunzione non è realistica: ad esempio i costi di spedizione di una merce lungo una rete stradale (come nel problema di trasporto ottimo) non sono direttamente proporzionali alla quantità di merce trasportata. Tuttavia è una semplificazione che può servire a dare una idea per molti casi concreti.

A partire da $\mathcal{R}$ rete elementare, costruiamo adesso una ***rete di domanda/offerta*** $\mathcal{R}^* = (V, A, u_{ij}, c_{ij}, b(i))$ associando ad ogni vertice i un intero $b(i)$ di "domanda" o di "offerta": se $b(i) > 0$ allora i è detto *vertice di offerta*, se $b(i) < 0$ i è detto *vertice di domanda*, se infine $b(i) = 0$, i è un *vertice di trasferimento*. Chiediamo poi che

$$\sum_{i \in V} b(i) = 0$$

ossia che il sistema sia "autosufficiente" : in altre parole il flusso totale generato dai nodi di offerta deve essere uguale al flusso totale richiesto dai nodi di domanda.

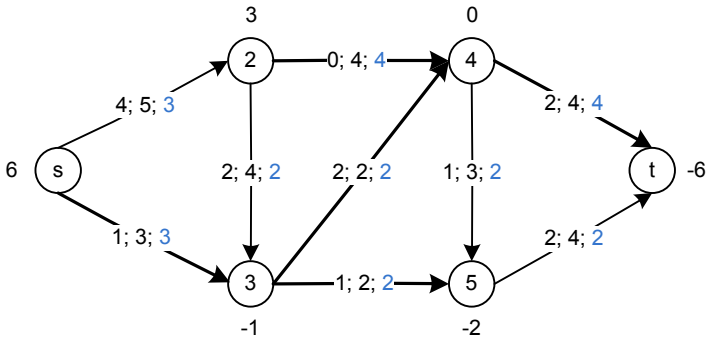
L'assegnazione di vertici di domanda e offerta può risultare utile in alcune applicazioni; si pensi ad esempio al problema di massimizzazione del numero di passeggeri trasportati da un aereo che visita n città: in ognuna di queste

scenderanno e/o saliranno un certo numero di passeggeri, creando offerte e domande sugli n nodi che rappresentano le città.

Un ***flusso ammissibile*** in una rete di domanda/offerta è una funzione $x : A \to \mathbb{R}, x = (x_{ij})$ che soddisfa, oltre alla solita *condizione di compatibilità*, la seguente *legge di conservazione* del flusso:

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i)\ ,\ \forall\ i \in V$$

Il vincolo assicura che per $b(i) > 0$ il flusso che esce dal nodo i è uguale a quello che entra più l'offerta in i; analogamente per $b(i) < 0$ il flusso che entra in i è uguale a ciò che esce più la domanda in i; se invece $b(i) = 0$ la quantità di flusso entrante è identica a quella uscente.



(n) Flusso ammissibile per una rete G con vertici di domanda/offerta

È immediato controllare che la condizione $\sum_{i\in V} b(i) = 0$ è necessaria per l'esistenza di una soluzione ammissibile.

Si noti che una rete $\mathcal{R} = (V, A, s, t)$ elementare in cui si cerchi un flusso di valore v può essere interpretata come una rete $\mathcal{R}^* = (V, A, b(i))$ di domanda/offerta relativa alla funzione $b : V \to \mathbb{R}$, data da

$$b(i) = \begin{cases} 0 & \forall\ i \neq s, t \\ v & per\ i = s \\ -v & per\ i = t \end{cases}$$

Osserviamo inoltre che da una rete $\mathcal{R}^* = (V, A, b(i))$ è sempre possibile passare ad una rete elementare $\mathcal{R} = (V, A, s, t)$ aggiungendo una sorgente s e un termine t:

per ogni vertice j di offerta viene aggiunto un arco (s,j) di capacità $u_{sj} = b(j) > 0$; per ogni vertice i di domanda viene aggiunto un arco (i,t) di capacità $u_{it} = -b(i) > 0$.
Si noti che $\sum_{i \in V} b(i) = 0$ comporta $\sum_{\{j:(s,j)\in A\}} u_{sj} = \sum_{\{i:(i,t)\in A\}} u_{it}$.

Un particolare flusso con $\sum_{\{i:(i,j)\in A\}} x_{ij} = \sum_{\{j:(j,i)\in A\}} x_{ji} \Leftrightarrow b(i) = 0 \ \forall\, i \in V$, è detto *circolazione*.

Il ***costo*** di un flusso x è, per definizione, la quantità $c(x) \coloneqq \sum_{(i,j)\in A} c_{ij}x_{ij}$. Il flusso x in figura (n) ha costo $c(x) = 39$.

Un flusso x si dice ***estremo*** se ha costo $c(x)$ minimo tra tutti i flussi che hanno lo stesso valore $v(x)$ di x.

Osservazione. Il flusso nullo è estremo perché è l'unico di valore 0.

Il *problema di flusso al minimo costo* consiste nella ricerca di un flusso x tale che:

- tutte le offerte/domande sui vertici siano soddisfatte
- tutte le capacità u_{ij} siano rispettate
- il flusso sia estremo

Esso può essere scritto anche come problema di programmazione lineare:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

$$\text{s.t.} \quad \begin{cases} \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i) & \forall\, i \in V \\ 0 \le x_{ij} \le u_{ij} & \forall\, (i,j) \in A \end{cases}$$

dove $\sum_{(i,j)\in A} c_{ij}x_{ij}$ è la funzione obiettivo ed il flusso la variabile decisionale; la matrice dei coefficienti $M \in \mathbb{M}(n,m)$ del sistema lineare ha tante righe quanti sono gli n vertici e tante colonne quanti gli m archi di $G = (V,A)$: la colonna corrispondente all'arco (i,j) ha $+1$ alla riga i-esima, -1 alla riga j-esima e 0 altrove (l'arco (i,j) "esce" da i ed "entra" in j). Una matrice siffatta è detta la *matrice di incidenza* (vertice-lato) del grafo G.

Se il valore $v(x)$ del flusso estremo trovato è massimo, diremo che x è un ***flusso massimo al minimo costo***.

Noi siamo interessati alla forma che questo problema assume nel caso delle reti elementari che, come detto, sono particolari reti di domanda/offerta.

Proposizione. Il problema di flusso massimo al minimo costo ammette soluzione ottima se e solo se esiste una soluzione ammissibile.

Dimostrazione. Per il "*teorema fondamentale della programmazione lineare*", un problema di p.l. che abbia una soluzione ammissibile e che non sia illimitato ammette soluzione ottima. Poiché la condizione di compatibilità garantisce che il problema sia limitato, se esiste una soluzione ammissibile il problema ammette soluzione ottima. ■

Osservazione. Nel caso di reti elementari $\mathcal{R} = (V, A, s, t)$, il problema ammette sempre soluzione ottima, giacché abbiamo sempre almeno la soluzione nulla.

Sottolineiamo da subito che il problema di flusso massimo al minimo costo è una generalizzazione delle problematiche di flusso massimo e cammino minimo dalle quali siamo partiti.
Per il flusso basta ricordare che un flusso massimo in una rete può vedersi come quello di minimo costo se sono assegnati $c_{ij} = 0,\ \forall\ (i,j) \in A$.
Se invece volessimo trovare un cammino minimo da s a t in $\mathcal{R} = (V, A, s, t)$, si può cercare una soluzione di un corrispondente problema di flusso al minimo costo ponendo $b(s) \coloneqq 1,\ b(t) \coloneqq -1,\ b(i) \coloneqq 0\ \forall\, i \neq s, t$ e $u_{ij} = 1 \quad \forall\ (i,j) \in A$. I flussi sugli archi avranno valore 0 o 1, e il cammino minimo cercato sarà dato dagli archi con flusso 1.

3. Cycle-cancelling algorithm

Sia $\mathcal{R} = (V, A, s, t)$ una rete elementare e via Ford e Fulkerson determiniamo un flusso massimo x^* di valore $v(x^*)$. Possiamo rivedere ora $\mathcal{R}$ come rete di domanda/offerta $\mathcal{R}^*$ con assegnati $b(i)$ nel seguente modo:

$$b(i) = \begin{cases} 0 & \forall\, i \neq s, t \\ v(x^*) & per\ i = s \\ -v(x^*) & per\ i = t \end{cases}$$

Per la ricerca del flusso massimo al minimo costo si può procedere ora con l'applicazione del ***cycle-cancelling algorithm***:

CYCLE-CANCELLING ALGORITHM:

input: rete elementare $\mathcal{R} = (V, A, s, t)$ con costi, capacità e flusso massimo x^*

output: flusso massimo al minimo costo

while $G(x)$ contiene cicli negativi **do**
identifica un ciclo negativo W
poni $\delta = \min \{r_{ij} |\ (i,j) \in W\} > 0$
aumenta il flusso x di δ unità lungo W,
aggiorna $G(x)$
endwhile

L'algoritmo presentato parte quindi da un flusso di valore massimo, successivamente ne modifica la configurazione diminuendone via via il costo e mantenendo fermo il suo valore allo scopo di trovare quello estremo. Ciò avviene inviando flusso sui cicli negativi individuati sulla rete residuale. L'aggiornamento può essere compiuto sulla rete originaria oppure su quella residuale, purché ci si ricordi di aggiornare non solo gli archi dove si incrementa il flusso, ma anche i loro (eventuali) inversi. Lo stop dell'algoritmo si ha quando $G(x)$ non contiene più cicli di

costo negativo. Si noti che questa situazione prima o poi si verifica: inviando flussi sugli archi del ciclo infatti, ad ogni iterazione almeno uno di questi (in particolare quelli con capacità residua pari a δ) o si saturano/scaricano in $\mathcal{R}$ sparendo in $G(x)$, oppure si crea l'arco di verso opposto che non consente di chiudere il ciclo.
Intuitivamente sembra naturale che tale condizione conduca al flusso cercato: infatti, per come si definisce il costo di un flusso e per come è costruita $G(x)$, questa non conterrà cicli negativi quando su $\mathcal{R}$ gli archi con costo alto portano flusso nullo mentre gli archi con costo basso si avvicinano il più possibile ad essere saturi. Per ogni arco scarico infatti, in $G(x)$ si avrà solo l'arco di costo positivo e più questo costo è alto, più sarà difficile che si creino cicli negativi. Questa idea è avvalorata dalla seguente condizione di ottimalità, su cui è basato l'algoritmo:

Condizione di ottimalità dei cicli negativi. Un flusso ammissibile x è un flusso estremo $\Leftrightarrow$ $G(x)$ non contiene cicli negativi.

Dimostrazione.
Necessità ($\Rightarrow$). Supponiamo, per assurdo, che x sia un flusso ammissibile ed estremo e $G(x)$ contenga un ciclo negativo. Allora aumentando il flusso lungo il ciclo negativo, si otterrebbe un flusso di eguale valore ma di costo più basso.

Vediamo ora due teoremi, utili per dimostrare la sufficienza.

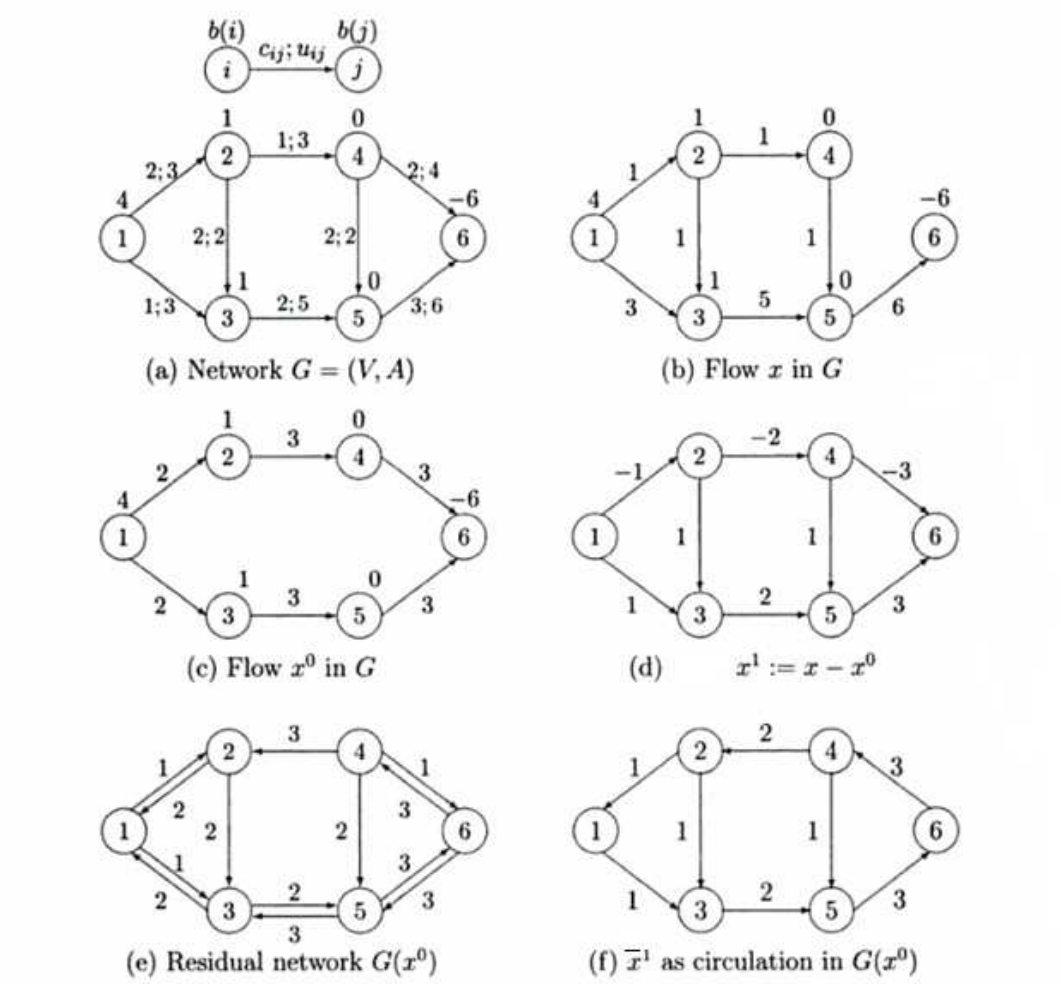
Introduciamo prima la *funzione caratteristica di un ciclo*. Sia $\mathcal{C}$ l'insieme di tutti i cicli diretti in una rete $\mathcal{R}$. Per ogni ciclo diretto $C \in \mathcal{C}$ sia $\chi^c : A \to \{0,1\}$,

$$\chi^c(a) = \begin{cases} 1 \ \ se\ a \in C \\ 0 \ \ se\ a \notin C \end{cases}$$

Teorema di decomposizione di una circolazione. Una circolazione x si può scomporre in un flusso inviato lungo cicli diretti, ossia $x = \sum_{C \in \mathcal{C}} \lambda\,(C)\chi^C$, con opportuni $\lambda(C) \geq 0$.

Si osservi il prospetto (o), riportato a pagina 23. Notiamo come la circolazione in (f) si possa esprimere come somma dei seguenti 3 cicli, ciascuno con una unità di flusso: $[2,1,3,5,6,4,2]$, $[2,3,5,6,4,2]$, $[4,5,6,4]$.

Teorema del ciclo aumentante. Siano x e x^0 due soluzioni ammissibili arbitrarie per una rete $\mathcal{R}^* = (V, A, b(i))$ con $v(x) = v(x^0)$. Allora $c(x)$ in G è pari a $c(x^0)$ in G più $\sum_{C \in \mathcal{C}} \lambda(C) c(C)$ in $G(x^0)$ per opportuni $\lambda(C) \geq 0$.



(o) Illustrazione del teorema dei cicli aumentanti

In figura è riportata, a titolo di esempio, una illustrazione del teorema appena descritto che mostra il legame tra i costi di due flussi ammissibili per la rete $G = (V, A)$ in (a). Sopra i vertici sono al solito riportati i rispettivi $b(i)$ e, di fianco agli archi, i costi seguiti dalle capacità. Notare che i dati che non appaiono valgono 0: ad esempio sono nulli i flussi in $(2, 3)$ e $(4, 5)$ in (c) e i $b(i)$ in (f).

In (b) e (c) si hanno i due flussi x e x^0 di eguale valore. Controlliamo ora che $c(x) - c(x^0) = c(x - x^0)$ in G sia uguale a $\sum_{C \in \mathcal{C}} \lambda(C) c(C)$ in $G(x^0)$:

$c(x - x^0) = -1 * 2 + (-2) * 1 + \cdots = 8$; la somma dei cicli $[2, 1, 3, 5, 6, 4, 2], [2, 3, 5, 6, 4, 2], [4, 5, 6, 4]$ ha anch'essa costo 8 (lasciamo il calcolo al lettore: si tenga presente che i costi sugli archi inversi in $G(x^0)$ sono ovviamente cambiati di segno rispetto ai costi di G riportati in (a)).

Possiamo ora dimostrare l'altro verso della doppia implicazione della condizione di ottimalità dei cicli negativi.

Sufficienza (⇐). Sia x un flusso ammissibile tale che $G(x)$ sia priva di cicli negativi. Poiché esiste un flusso ammissibile allora ne esiste anche uno estremo x^0. Per il teorema dei cicli aumentanti esiste $C = \sum_{C \in \mathcal{C}} \lambda(C)\chi^C$ con $\lambda(C) \geq 0$ tale che $c(x^0) \coloneqq c(x) + c(C)$; ma $c(C) \geq 0 \ \forall\, C \in \mathcal{C}$ per ipotesi, pertanto $c(x) \leq c(x^0)$ e, poiché x^0 è estremo, ne segue $c(x) = c(x^0)$ ossia anche x è estremo. ■

In generale un flusso estremo non ha valore massimo, ma è il flusso con costo minimo tra i flussi di un certo valore. Tuttavia, giacché l'algoritmo parte dal flusso massimo e ad ogni passo ne tiene fermo il valore, quando $G(x)$ non conterrà cicli negativi, il flusso che da questa verrà ricostruito sarà estremo e di valore massimo: abbiamo trovato la soluzione ottima del *max-flow min-cost problem*.

Proposizione. Se tutte le capacità u_{ij} e i $b(i)$ (eventuali) di domanda o di offerta sono interi, il problema di flusso massimo al minimo costo ha soluzione intera.

Dimostrazione. Se tutte le capacità sono intere, partendo con un flusso nullo e applicando l'algoritmo dei cammini aumentanti si hanno δ interi e pertanto un flusso massimo anch'esso intero. Da questo, poiché sono intere anche le capacità residue in $G(x)$, il flusso inviato sui suoi cicli negativi porta a un flusso estremo che è, a sua volta, intero. ■

Da chiarire, a questo punto, lo step dell'algoritmo che prevede l'individuazione dei cicli negativi su $G(x)$. Dobbiamo servirci di un algoritmo di cammino minimo che ammetta costi negativi sugli archi, come ad esempio il generalized shortest path algorithm, illustrato nel paragrafo 2.4. Tuttavia, occorre modificare opportunamente l'algoritmo per renderlo funzionale al nostro scopo, che è quello di

individuazione dei cicli negativi. Dobbiamo cioè aggiungere uno stop non appena l'algoritmo riconosce un ciclo di costo negativo che parte da s, altrimenti si continuerebbe a girare all'infinito sul ciclo diminuendo ad ogni giro i potenziali dei vertici che formano il ciclo stesso. Un modo per evitare che l'algoritmo dei cammini minimi generalizzato riconosca i cicli negativi senza andare in loop potrebbe essere:

NEGATIVE CYCLE ALGORITHM:

input: rete elementare $\mathcal{R} = (V, A)$ con una sorgente s
e una funzione di costo (o lunghezza) a valori in $\mathbb{R}$

output: ciclo negativo W che parte da s

start:
$p(s) \coloneqq 0$; $pred(s) \coloneqq \emptyset$
$p(j) \coloneqq +\infty \ \forall\, j \in V - \{s\}$

while qualche arco $(i,j) \in A$ soddisfa $p(j) > p(i) + c_{ij}$ **do**
for ogni $(i,j) \in A(i)$ e ogni $i \in V$
if $p(j) > p(i) + c_{ij}$ **then**
$p(j) \coloneqq p(i) + c_{ij}$
$pred(j) \coloneqq i$
if $j = s$ **and** $p(s) < 0$ **then**
stop
endwhile

L'idea è quella di utilizzare l'algoritmo modificato settando via via come sorgente tutti gli $i \in V$: si può così controllare facilmente se una rete contiene o meno cicli negativi. In particolare, applicandolo ad una rete residuale $G(x)$ si riesce a valutare se la condizione di ottimalità dei cicli negativi è verificata, e quindi se il flusso ottenuto col cycle-cancelling algorithm è un flusso massimo al minimo costo.

4. Successive shortest path algorithm

Prima di illustrare l'algoritmo occorre introdurre le seguenti due definizioni:

Si dice ***pseudoflusso*** una funzione $x : A \to \mathbb{R}, x = (x_{ij})$ che soddisfa la condizione di compatibilità $0 \leq x_{ij} \leq u_{ij}$, ma non necessariamente la legge di conservazione.

Si definisce lo *squilibrio* di un vertice $i \in V$ per un dato pseudoflusso x come $e(i) = b(i) + \sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij}$.
Si dice inoltre vertice in *eccesso* un vertice con $e(i) > 0$, vertice in *difetto* un vertice con $e(i) < 0$.
Quindi, dato uno psuedoflusso, se i è di eccesso, in esso si offre di meno rispetto a $b(i)$ ed è quindi necessario inviare flusso aggiuntivo sugli archi uscenti da i per ottenere un flusso ammissibile; se invece i è di difetto, in esso la domanda è inferiore $b(i)$, e pertanto sarà necessario inviare flusso sugli archi entranti.

Partiamo adesso da una rete elementare con sorgente e termine e dotiamola di una struttura di domanda/offerta definendo i seguenti $b(i)$:

$$b(i) := \begin{cases} 0 & per\ i \neq s, t \\ M & per\ i = s \\ -M & per\ i = t \end{cases}$$

dove $M = \min \left\{ \sum_{\{j:(s,j)\in A\}} u_{sj} ; \sum_{\{i:(i,t)\in A\}} u_{it} \right\}$, il minimo tra la somma delle capacità degli archi uscenti dalla sorgente e la somma delle capacità di quelli entranti nel termine. M rappresenta cioè il massimo valore potenzialmente raggiungibile da un flusso su una rete: pertanto $v(x) \leq M$, e inoltre $M > 0$ giacché abbiamo estromesso dalla rete gli archi con capacità nulla e la rete è connessa.

Di seguito presentiamo il "***successive shortest path algorithm***", proposto da Jewell nel 1958.

SUCCESSIVE SHORTEST PATH ALGORITHM:

input: rete diretta $\mathcal{R} = (V, A, s, t)$ con costi e capacità,
$x \coloneqq 0$, $e(i)\ \forall i \in V$

output: flusso massimo al minimo costo

while esiste un cammino da s a t in $G(x)$ **do**
individua un cammino minimo P tra s e t in $G(x)$
poni $\delta = \min\{r_{ij} \mid (i,j) \in P\}$
aumenta x di δ unità lungo P
aggiorna $G(x)$, $e(i)$
endwhile

L'algoritmo parte quindi dallo pseudoflusso nullo: al primo passo gli sbilanciamenti saranno pertanto tutti uguali ai $b(i)$ e dunque nulli tranne in s e t. Si individua poi un cammino aumentante P da s a t in $G(x)$ di costo minimo e si aumenta di δ unità lungo P: così oltre a decrescere le capacità residue degli archi di P in $G(x)$ di δ (e quindi accrescere quelle degli archi inversi), anche $e(s)$ diminuirà mentre $e(t)$ aumenterà, entrambi di δ unità.

Infatti, sia $\hat{x}$ il nuovo pseudoflusso; P essendo un cammino da s a t contiene un solo arco uscente da s: su di esso il flusso è cresciuto di δ per cui $\sum_{\{i:(i,s)\in A\}} \hat{x}_{is} - \sum_{\{j:(s,j)\in A\}} \hat{x}_{sj} = \delta + \sum_{\{i:(i,s)\in A\}} x_{is} - \sum_{\{j:(s,j)\in A\}} x_{sj} =$
$= \delta + b(i) - e(i) = b(i) - (e(i) - \delta)$ ossia il nuovo sbilanciamento è $b(i) - \delta$.

Si noti che il cammino P utilizzato all' r-esimo passo, non è più presente in $G(x)$ all' $r+1$-esimo passo in quanto almeno un arco di P o ha cambiato verso o è stato estromesso dalla rete perché ha capacità residua nulla. Dunque ad ogni passo diminuisce il numero di cammini presenti da s a t.

Lo stop dell'algoritmo si ha quando non ci sono più cammini diretti P tra la sorgente e il termine in $G(x)$, ossia quando non esistono cammini aumentanti in $\mathcal{R}$. Per il corollario del *maxflow-mincut theorem* questa condizione implica che il valore del flusso sia massimo.

Osservazione. $e(s) = -e(t) = 0$, per come abbiamo definito i $b(i)$, si raggiunge solo se $v(x) = M$ ossia il massimo valore che il flusso può assumere sulla rete: ciò si verifica a patto che la rete non contenga "colli di bottiglia", cioè archi di capacità ristretta da cui il flusso è obbligato a passare riducendone di conseguenza il valore.
Qualora l'algoritmo conduca ad un flusso $\bar{x}$ di valore $v(\bar{x}) < M$, ossia con $e(i) = b(i) - v(\bar{x}) > 0$, siamo di fronte ad uno pseudoflusso per una rete $\mathcal{R}^*$ di domanda/offerta: per avere un flusso ammissibile su $\mathcal{R}^*$ dovremo quindi ridefinire i $b(i)$ ponendo $b(s) = v(\bar{x})$, $b(t) = -v(\bar{x})$ e 0 altrove.

Siamo partiti quindi da uno pseudoflusso di costo minimo ($x = 0$) e mediante le iterazioni dell'algoritmo siamo arrivati ad un flusso di valore massimo, individuando cammini di costo minimo e inviando flusso sui suoi archi. Bisogna ora chiedersi se il flusso ottenuto è ancora estremo.

Teorema. Sia x un flusso estremo per $\mathcal{R}$ e sia P un cammino minimo in $G(x)$; il flusso x' ottenuto inviando su P il massimo incremento di flusso possibile (δ) è ancora estremo.

Dimostrazione. Prima di aumentare x su P, $G(x)$ è priva di cicli negativi poiché x è estremo per ipotesi. Supponiamo, per assurdo, che dopo aver aumentato x su P ottenendo x', $G(x')$ contenga un ciclo negativo W. Allora esso deve contenere almeno un arco b di P poiché solo lì il flusso e quindi la rete residuale si è modificata. Inoltre b deve essere di verso opposto su P e su W altrimenti avremmo avuto W in $G(x)$ essendo $W - \{b\}$ tutti archi che appartenevano anche a $G(x)$. In altre parole esiste un lato $b \in P$ tale che il suo inverso b^{-1} chiude un ciclo negativo W in $G(x')$. Ne segue che
$c(b) + \sum_{a \in W - \{b\}} c(a) < 0$ e $c(P) = \sum_{a \in P - \{b\}} c(a) - c(b)$ poiché b in P cambia di verso (e con lui anche il segno del costo); quindi
$c(P) > \sum_{a \in P - \{b\}} c(a) + \sum_{a \in W - \{b\}} c(a) = c(T)$ dove
$T = \{a \in P - \{b\} \cap W - \{b\}\}$ è un cammino di costo inferiore a P contro l'ipotesi P cammino minimo. ■

Si desume pertanto che quando in $G(x)$ non ci saranno cammini aumentanti disponibili avremo davanti un flusso ottimo per il problema di flusso massimo al minimo costo: il flusso è infatti massimo perché non ha cammini aumentanti, ed è estremo perché lo garantisce, ad ogni passo e dunque fino all'ultimo passo, il teorema precedente.

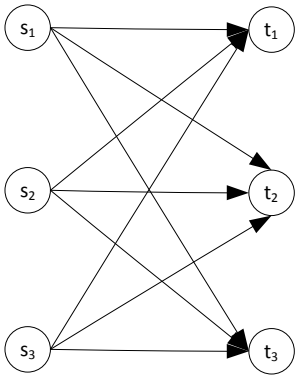
Analogamente al cycle-cancelling algorithm, per l'individuazione del cammino minimo nella rete residuale occorre utilizzare un algoritmo di minima distanza con costi arbitrari sugli archi.

La soluzione ottima è intera se, oltre alle capacità, si hanno $b(i)$ (e quindi squilibri) interi.

5. Il problema del trasporto ottimo

Torniamo ora all'applicazione cui abbiamo accennato nella introduzione, presentando un esempio numerico e risolvendolo facendo ricorso ai due algoritmi illustrati nei capitoli 3 e 4.

Si supponga, per semplicità, che le fabbriche s_i dell'azienda suddetta siano 3 e che 3 siano anche i clienti che richiedono il suo prodotto. Ciò dà luogo al seguente grafo:



(p)

Riportiamo di seguito i dati per questo problema.

- Capacità (in unità di prodotto mensili):

  Capacità produttive a_i degli stabilimenti: $a_1 = 9, a_2 = 13, a_3 = 4$

  Richieste b_j dei clienti: $b_1 = 3, b_2 = 7, b_3 = 12$

  Capacità di trasporto d_{ij} dalla fabbrica i al cliente j:

d_{ij}	$j = 1$	$j = 2$	$j = 3$
$i = 1$	5	0	6
$i = 2$	3	8	4
$i = 3$	0	1	2

- Costi (per unità di prodotto):

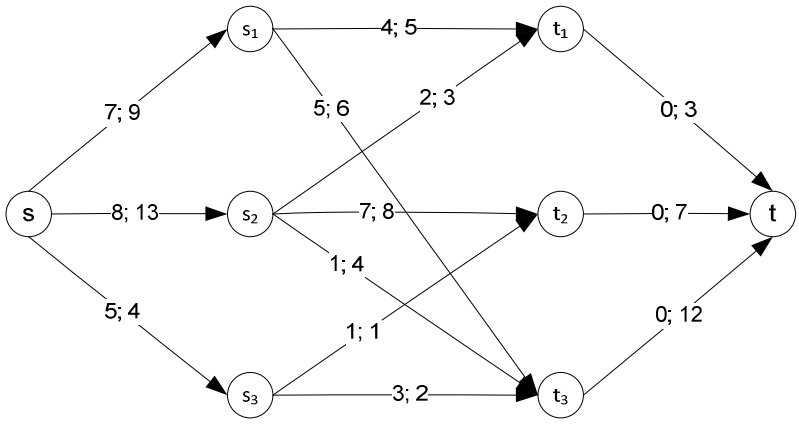
  Costi c_i di produzione: $c_1 = 7, c_2 = 8, c_3 = 5$.

  Costi di trasporto c_{ij} dalla fabbrica i al cliente j:

c_{ij}	$j = 1$	$j = 2$	$j = 3$
$i = 1$	4	-	5
$i = 2$	2	7	1
$i = 3$	-	1	3

Si noti che poiché è nulla la capacità degli archi (s_1, t_2) e (s_3, t_1) non ha senso definirvi un costo in quanto tali archi saranno estromessi dalla rete.

Aggiungendo ora un "superstabilimento" s e un "supercliente" t, si può modellare il problema creando una rete e assegnandovi capacità e costi secondo i dati riportati: i costi c_i e le capacità a_i di produzione sugli archi (s, s_i), i costi c_{ij} e le capacità d_{ij} di trasporto su (s_i, t_j) e infine costi nulli e le richieste b_j come capacità degli archi (t_j, t).

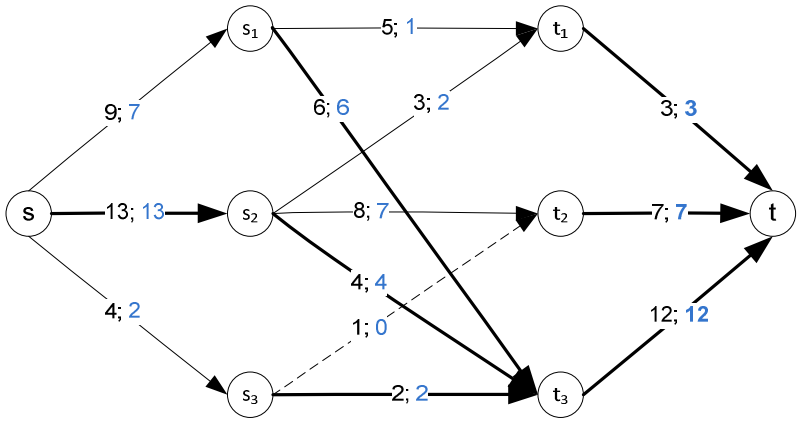


(q)

La richiesta dei clienti può essere soddisfatta ⇔ esiste un flusso (massimo) che satura gli archi (t_j, t), ossia tale che il valore del flusso in (t_j, t) coincide con le capacità assegnate su quell'arco. Se questo esiste i valori (s, s_i) corrispondono al livello di produzione che ogni mese la fabbrica i deve garantire e i valori (s_i, t_j) rappresentano le unità di prodotto che la i-esima fabbrica deve spedire al j-esimo cliente.

Trovando poi il flusso massimo al minimo costo, oltre a soddisfare la richiesta dei clienti, si determina come ripartire la produzione di ogni stabilimento e come trasportare il prodotto, in maniera da minimizzare i costi che le fabbriche devono sostenere, massimizzandone così il profitto.

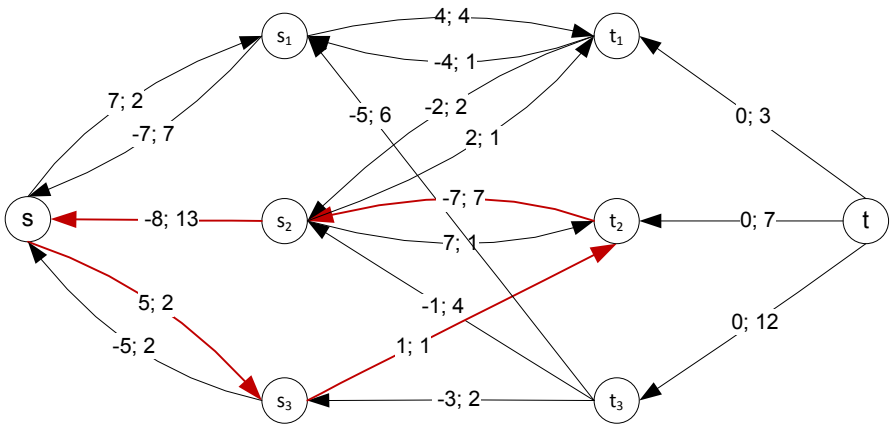
Vogliamo utilizzare il cycle-cancelling algorithm per la ricerca del flusso massimo al minimo costo. Innanzitutto occorre trovare il flusso massimo (senza preoccuparci dei costi) con l'algoritmo di Ford e Fulkerson che saturi gli archi (t_j, t). L'applicazione dell'algoritmo dei cammini aumentanti porta ad un flusso x di valore $v(x) = 22$ come quello in figura (r).



(r)

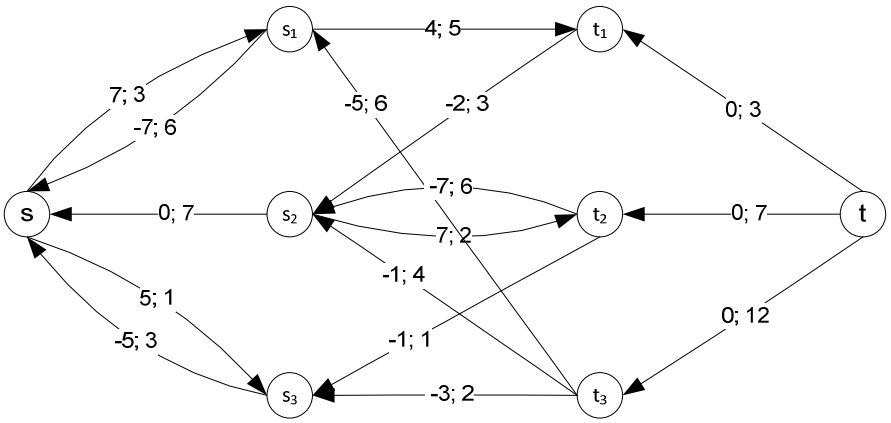
Il flusso ottenuto permette di affermare che la richiesta dei clienti può essere soddisfatta, ad esempio, con un piano produttivo e di trasporto come quello mostrato dagli archi (s, s_i) e (s_i, t_j). Ma questo piano permette di minimizzare i costi dell'impresa? In altre parole il flusso massimo trovato è di minimo costo? Se si osserva attentamente il flusso in figura, si nota come archi di costo basso come

(s_3, t_2) o (s_2, t_1) non siano saturi. A prima vista sembrerebbe quindi che il costo totale del flusso (ora pari a $c(x) = 260$) non sia minimo, ossia che il flusso non sia estremo. Per esserne certi applichiamo ora il cycle-cancelling algorithm per individuare eventuali cicli negativi sui quali poter inviare flusso e diminuirne così il costo. Costruiamo a tale fine la rete residuale $G(x)$, raffigurata in (s).



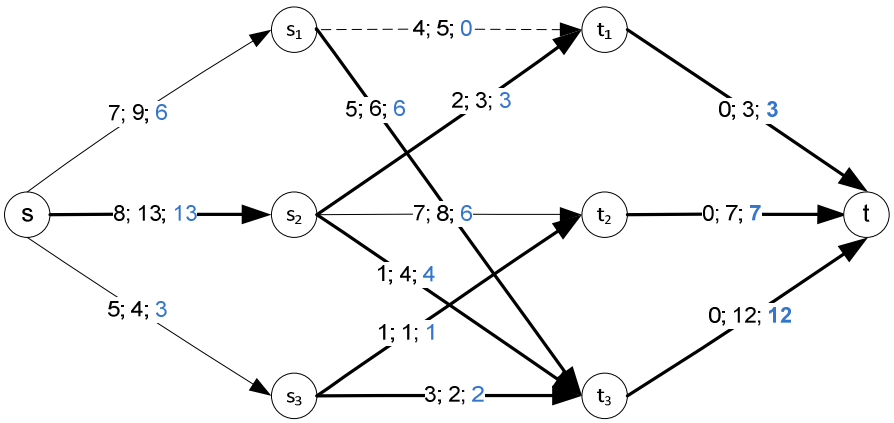
(s)

In $G(x)$ vi è un ciclo negativo $W^1 = [s, s_3, t_2, s_2, s]$, evidenziato in rosso, che ha come massimo incremento $\delta = \min\{2, 1, 7, 13\} = 1$. Si aumenta quindi di una unità il flusso lungo W^1 e si aggiorna la rete. Fatto questo si individua un altro ciclo negativo $W^2 = [s, s_2, t_1, s_1, s]$ sempre con $\delta = 1$ che, dopo l'incremento, dà luogo al seguente grafo incrementale:



(t)

$G(x)$ è ora privo di cicli negativi dunque l'algoritmo termina e per la condizione di ottimalità si ha che x è estremo.



(u) Flusso massimo al minimo costo

L'algoritmo di cancellazione dei cicli negativi ha condotto ad un flusso ottimo di valore $v(x) = 22$ e costo $c(x) = 250$. Come si nota l'algoritmo ha spedito flusso sugli archi (s_3, t_2) e (s_2, t_1) saturandoli e ottenendo un risparmio di 10 dal flusso raggiunto con l'algoritmo di Ford e Fulkerson. I livelli di produzione e il piano di trasporto ottimi si desumono dalla rete in figura (u).

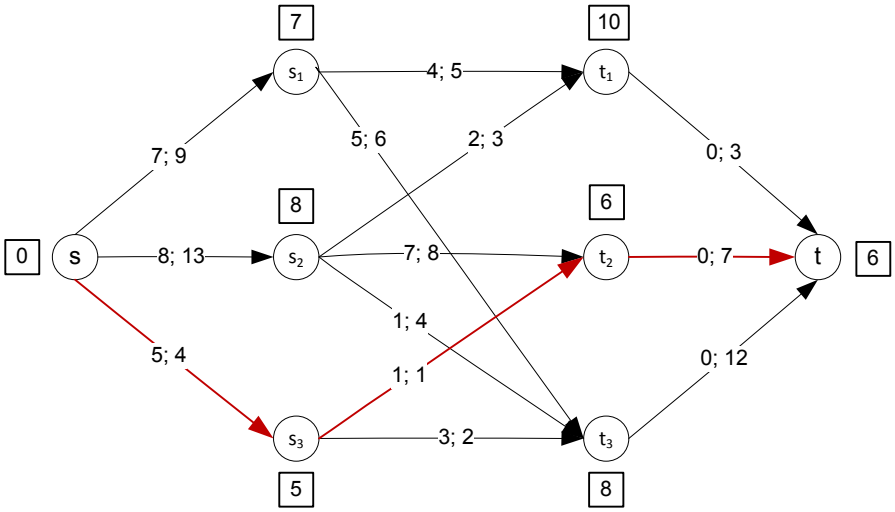
Vediamo ora se ad un risultato analogo si perviene mediante l'algoritmo dei cammini minimi successivi.

Dobbiamo dotare la rete iniziale di una struttura di domanda/offerta definendo

$$b(i) := \begin{cases} 0 & per\ i \neq s, t \\ 22 & per\ i = s \\ -22 & per\ i = t \end{cases}$$

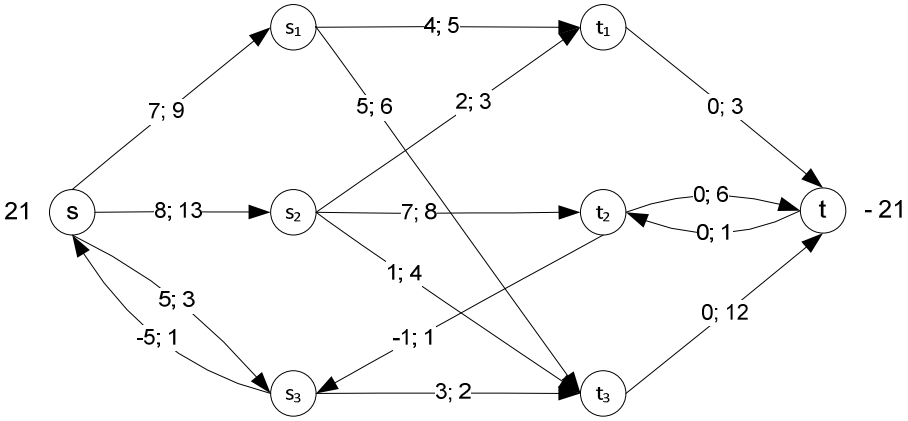
poiché 22 è il massimo valore che il flusso può raggiungere sulla rete e in particolare quello che soddisfa la domanda dei clienti.

L'algoritmo parte dallo psedoflusso $x = 0$, per cui la rete residuale è identica a quella di partenza in figura (q) con $b(i) = 22$ sulla sorgente e -22 sul termine. Individuiamo su questa un cammino di costo minimo, ad esempio con l'algoritmo di Dijkstra (poiché la rete ha costi ancora tutti positivi) assegnando ad ogni vertice il suo potenziale, che in genere viene indicato dentro un riquadro come in figura (v).



(v)

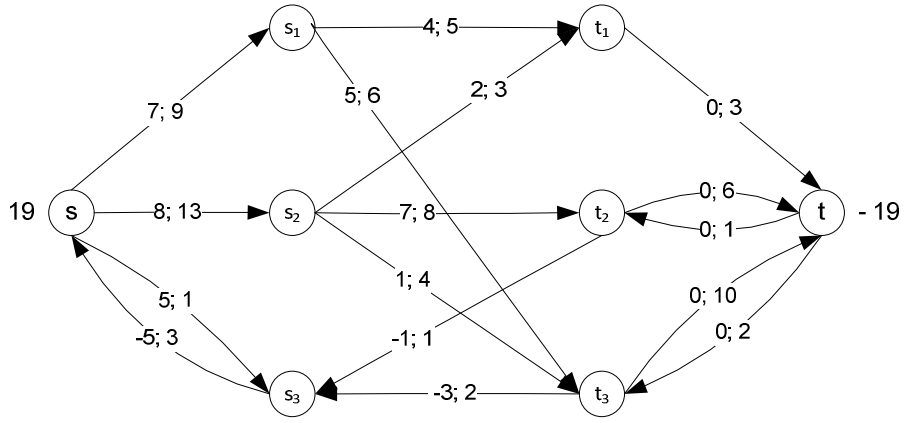
Il cammino $P^1 = [s, s_3, t_2, t]$ evidenziato in rosso è minimo di lunghezza 6; lo step successivo del successive shortest path algorithm prevede che si invii su P^1 una quantità di flusso pari a $\delta = \min\{4; 1; 7\} = 1$ che dà luogo alla seguente rete residuale:



(w)

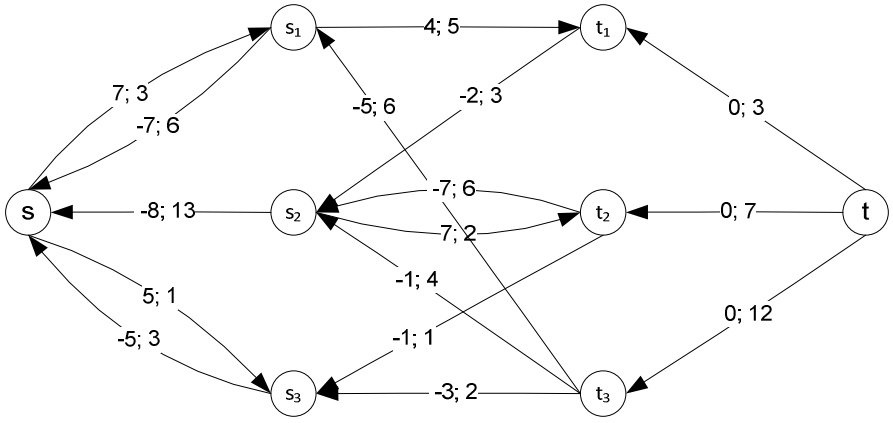
Come si può notare da (w), le capacità r_{ij} sugli archi concordi di P^1 in $G(x)$ sono diminuite di 1 unità e si sono generati archi discordi di capacità $r_{ji} = 1$. Inoltre, poiché lo pseudoflusso ha valore 1, lo squilibrio $e(s) = b(s) - \delta$ è diminuito e lo squilibrio $e(t)$ è aumentato, entrambi di 1 unità; gli squilibri verranno annotati in figura ad ogni passo.

Come si vede, dopo l'aggiornamento di $G(x)$, $P^1 = [s, s_3, t_2, t]$ non esiste più, pertanto dovremo cercare un altro cammino di costo minimo sulla rete residuale in (w). Lo individuiamo in $P^2 = [s, s_3, t_3, t]$, cammino di distanza pari a 8. Si invierà pertanto flusso su tale cammino ($\delta = 2$) aggiornando poi $G(x), e(s), e(t)$, (figura (x)).



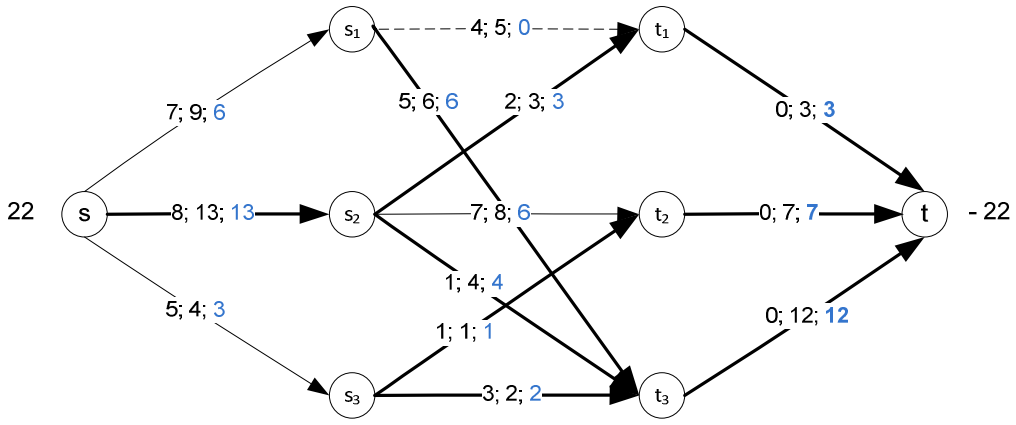
(x)

Continuando a far girare l'algoritmo, alla sesta iterazione, si trova un ultimo cammino minimo da s a t $P^6 = [s, s_2, t_2, t]$ di costo $c(P) = 15$ cui corrisponde il grafo incrementale aggiornato in (y).



(y)

Adesso la rete è priva di cammini aumentanti, per cui l'algoritmo si ferma (si noti che $e(s) = e(t) = 0$) e dall'ultimo grafo incrementale si risale al flusso ottimo in figura (z).



(z)

Da un confronto delle figure (z) e (u) si nota che i due algoritmi hanno condotto ad un risultato equivalente: un flusso massimo che soddisfa le richieste dei clienti (il primo ottiene 3 unità, il secondo 7 e il terzo 12) e di costo totale minimo per l'azienda (pari a 250).

6. Altre applicazioni

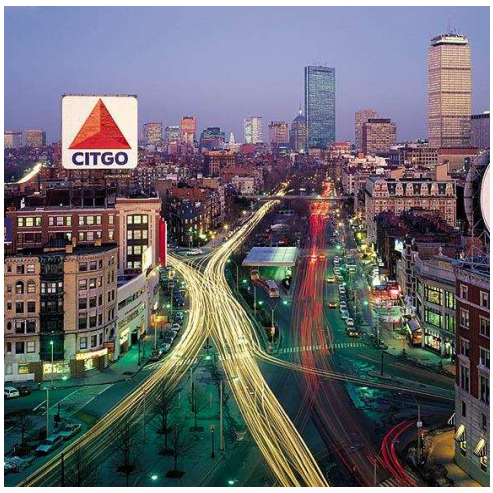
Una grande varietà di applicazioni può essere modellata come problema di cammino minimo con costi arbitrari. Di seguito ne riportiamo un tipico esempio, conosciuto come "problema del rappresentante".

Un rappresentante sta per intraprendere una serie di viaggi (ad esempio aerei) per vendere i suoi prodotti, partendo dalla città A e terminando il suo tour in B. In base a viaggi fatti in passato, è possibile stimare quanti prodotti riuscirà a vendere (e quindi il guadagno ottenuto) in ogni città visitata lungo il percorso intrapreso. L'*itinerario ottimo* può essere trovato risolvendo un problema di cammino minimo su una rete elementare con A e B come sorgente e termine, le città intermedie come vertici e le tratte aeree che le collegano come archi. Ogni arco (i, j) ha un dato costo $c_{ij} = f_{ij} - l_j$ dove f_{ij} è il costo del volo dalla città i alla città j e l_j è il guadagno ottenuto vendendo i prodotti in j. Un cammino minimo tra A e B identifica l'itinerario ottimo per il rappresentante.

I problemi di flusso massimo al minimo costo riguardano numerosi ambiti industriali e scientifici, ma probabilmente il tipo di applicazione più importante in questo contesto è quella che abbiamo chiamato "trasporto ottimo" che riguarda la distribuzione di una azienda ai suoi clienti e che abbiamo presentato in generale nella introduzione per poi mostrarne un esempio con dati fittizi nel capitolo 5.
Questa applicazione implica sempre la determinazione di un piano per la produzione ed il trasporto di merci dai nodi sorgente (nel nostro caso fabbriche) ai nodi termine (clienti) passando eventualmente da nodi di trasferimento (ad esempio magazzini). Il seguente prospetto riporta altre particolari tipologie di questa applicazione:

Tipo di applicazione	Nodi sorgente	Nodi di trasporto	Nodi i destinazione
Operazione su una rete di distribuzione	Produttori di beni	Servizi intermedi di immagazzinamento	Clienti
Smaltimento di rifiuti solidi	Produttori di rifiuti solidi	Impianti di smaltimento	Discariche
Operazione su una rete di servizi	Venditori	Magazzini intermedi	Impianti di produzione
Determinaz. del mix ottimo di produzione nelle fabbriche	Fabbriche	Produzione di uno specifico prodotto	Mercato per uno specifico prodotto
Gestione del flusso di fabbrica	Sorgente di contante a un dato istante	Opzioni di investimento a breve termine	Bisogni di contante a uno specifico istante

Per alcune applicazioni del problema di flusso massimo al minimo costo, le fabbriche non sono nodi sorgente ma nodi intermedi nei quali avviene una qualche operazione sulle merci. Questo è il caso dello *smaltimento di rifiuti solidi* (seconda riga del prospetto). In questo caso, il flusso dei materiali attraverso la rete parte dai punti che generano rifiuti solidi, poi si dirige verso le fabbriche per il trattamento di tali rifiuti e la loro trasformazione, e quindi viene spedito verso i vari siti dove sono dislocate le discariche. L'obiettivo è comunque ancora quello di determinare un flusso che minimizzi il costo totale.



Un caso degno di nota è quello della *Citgo Petroleum Corporation* che per migliorare l'operazione di distribuzione dei suoi prodotti si servì di modelli simili a quelli che abbiamo descritto. La Citgo Petroleum Corporation (in figura uno stabilimento a Boston) è una industria specializzata nella raffinazione e nella vendita di prodotti petroliferi. A metà degli anni '80 ha avuto vendite annuali per

diversi miliardi di dollari, posizionandosi tra le prime 150 degli Stati Uniti. Dopo un periodo di pesanti perdite finanziarie, nel 1983 fu acquisita dalla *Southland Corporation* proprietaria di una catena di grandi magazzini. Per circoscrivere le perdite finanziarie della Citgo, la Southland creò una task force, a dirigere la quale fu nominato un eminente consulente di ricerca operativa chiamato a riferire direttamente al presidente della Southland.

Durante il 1984 e il 1985 questa task force ha applicato varie tecniche di ricerca operativa, le più importanti delle quali basate sulla programmazione lineare, tra cui appunto modelli di ottimizzazione su reti di distribuzione. Queste hanno portato a grandi miglioramenti nel rendimento delle raffinerie grazie a riduzioni sostanziali del costo del lavoro, contribuendo nel 1985 a un incremento del profitto di circa 50 milioni di dollari.

7. Conclusioni

Il problema di flusso massimo al minimo costo occupa una posizione centrale tra i modelli di ottimizzazione su rete, sia perché, come abbiamo visto, comprende una classe ampia di applicazioni e sia perché molte altre problematiche possono essere trattate come suoi casi particolari.

Tale problema inoltre può essere risolto in maniera molto efficiente mediante diversi algoritmi; in questa sede ne abbiamo presentati due.
Abbiamo notato che questi, nel raggiungere il flusso ottimo, trattano ammissibilità e ottimalità in maniera opposta: il cycle-cancelling algorithm avendo come presupposto l'ottenimento di un flusso di valore massimo, parte da una soluzione ammissibile e, ai passi successivi, enumera flussi di costo via via inferiore di cui solo l'ultimo soddisfa la condizione di ottimalità. Per contro il successive shortest path algorithm ha allo start, uno pseudoflusso estremo che soddisfa quindi la condizione di ottimalità, ma non è ammissibile (è uno pseudoflusso) salvo diventarlo dopo l'ultima iterazione dell'algoritmo.

Vi è inoltre un'altra sostanziale differenza, già sottolineata nella introduzione, ovvero il modo di interfacciare gli algoritmi di flusso massimo e di cammino minimo. L'algoritmo basato sulla cancellazione dei cicli negativi utilizza infatti in un primo momento l'algoritmo dei cammini aumentanti al fine di trovare un flusso massimo e successivamente, ad ogni passo dell'algoritmo, si serve di un algoritmo di cammino minimo (che ammetta sugli archi costi arbitrari) per controllare se sul grafo incrementale vi sono cicli negativi. Viceversa l'algoritmo dei cammini minimi successivi, ad ogni step utilizza i cammini aumentanti inviando il massimo incremento possibile sulla rete residuale, ma tale incremento è inviato su cammini di minima distanza. In conclusione, il cycle-cancelling tiene separati i due algoritmi di base mentre il successive shortest path li interfaccia ad ogni passo.

8. Riferimenti bibliografici

- Kennet H. Roses, John G. Micheals, Jonathan L. Gross, Jerrold W. Grossman, Douglas R. Shier, "*Handbook of Discrete and Combinatorial Mathematics"*, CRC Press, Boca Raton, Florida, 2000.

- Alexander Schrijver, "*A course in Combinatorial Optimization*", (Lecture notes), Amsterdam, 2003.

- Friederick S. Hillier, Gerard J. Lieberman, "*Ricerca Operativa*", McGraw-Hill, Milano, 2005.

- Peter Brucker, Sigrid Knust, "*Complex Scheduling*", Springer, Berlino, 2006.