L’algoritmo di Edmonds-Karp

Luca Foschiani

Università degli studi di Udine

Advanced Algorithms

Overview

Definizioni: rete di flusso

Una **rete di flusso** $G = (V, E)$ è un grafo orientato nel quale ad ogni arco $(u, v) \in E$ è assegnata una capacità non negativa $c(u, v) \geq 0$.
Inoltre, assumiamo che se esiste un arco $(u, v) \in E$, allora $(v, u) \notin E$.
Individuiamo due nodi s (**sorgente**) e t (**pozzo**). Per questi due nodi, abbiamo che per ogni nodo in V esiste almeno un cammino che lo contiene e che va da s a t (ogni nodo è raggiungibile da s e raggiunge t).

Definizioni: flusso

Un **flusso** in G è una funzione $f : V \times V \to \mathbb{R}$ che soddisfa le seguenti proprietà:

- Per ogni $u, v \in V$ richiediamo $0 \leq f(u, v) \leq c(u, v)$ (vincolo di capacità)
- Per ogni $u \in V \setminus \{s, t\}$ richiediamo $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$ (conservazione del flusso)

La quantità $f(u, v)$ viene chiamata flusso dal nodo u al nodo v.
Il **valore** $|f|$ di un flusso f è definito nel seguente modo:

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$$

Obiettivo del **problema del massimo flusso** è, data una rete di flusso, trovare il flusso di valore massimo.

Concetti:

- Rete residua
- Cammino aumentante
- Taglio

Per poi arrivare al **Teorema del flusso massimo e taglio minimo** (permette di dimostrare che l'algoritmo trova sempre il flusso massimo).

L'algoritmo di Ford-Fulkerson
Rete residua

Dati una rete G e un flusso f, la rete residua $G_f = (V_f, E_f)$ permette di identificare i cammini lungo i quali è possibile aumentare il flusso.
Ponendo $G = (V, E)$, s sorgente e t pozzo, Possiamo definire la **capacità residua** c_f nel seguente modo:

$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v) & \text{se } (u, v) \in E \\ f(v, u) & \text{se } (v, u) \in E \\ 0 & \text{altrimenti} \end{cases}$$

Il primo caso della definizione corrisponde alla "capacità residua" degli archi presenti in G.
Il secondo caso permette all'algoritmo di "vedere" le quantità di flusso già assegnate agli archi, dando la possibilità di diminuire il flusso su un arco (assegnando al corrispondente arco in G_f un flusso non nullo).
In G_f avrò gli stessi nodi presenti in G, mentre gli archi saranno tutti gli (u, v) tali che $c_f(u, v) > 0$.

Il metodo di Ford-Fulkerson
Rete residua

Idea: un flusso individuato nella rete residua permette di aumentare il flusso nella rete originale.
Dati f flusso in G e f' flusso in G_f, definisco la funzione $(f \uparrow f') : V \times V \to \mathbb{R}$ nel seguente modo:

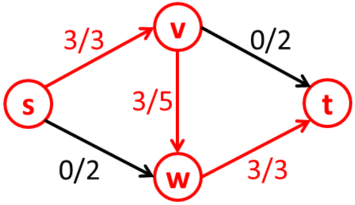
$$(f \uparrow f')(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{se } (u,v) \in E \\ 0 & \text{altrimenti} \end{cases}$$

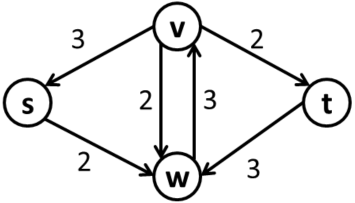
Lemma

La funzione $f \uparrow f'$ è un flusso in G avente valore $|f \uparrow f'| = |f| + |f'|$.

Il metodo di Ford-Fulkerson
Esempio rete residua

Rete di flusso (flusso/capacità) e corrispondente rete residua:





Data una rete di flusso G e un flusso f, un **cammino aumentante** è un cammino semplice che va da s a t nella rete residua G_f.
Dato un cammino aumentante p, chiamiamo **capacità residua** di p la massima quantità di flusso che è possibile mandare su questo cammino: $c_f(p) = min\{c_f(u, v) : (u, v) \text{ sta in } p\}$.
Definiamo inoltre una funzione $f_p : V \times V \to \mathbb{R}$ nel seguente modo:

$$f_p(u, v) = \begin{cases} c_f(p) & \text{se } (u, v) \text{ sta in } p \\ 0 & \text{altrimenti} \end{cases}$$

Lemma

f_p è un flusso in G_f di valore $|f_p| = c_f(p) > 0$.

Corollario

$f \uparrow f_p$ è un flusso in G di valore $|f \uparrow f'| = |f| + |f_p| > |f|$.

Un **taglio** (S, T) di una rete di flusso $G = (V, E)$ è una partizione di V in S e T tale che $s \in S$ e $t \in T$.
Dato f flusso, esprimiamo con $f(S, T)$ il **flusso che attraversa il taglio**:

$$f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u)$$

La **capacità di un taglio** (S, T) è:

$$c(S, T) = \sum_{u \in S} \sum_{v \in T} c(u, v)$$

Il **taglio minimo** è il taglio di capacità minima fra i tagli della rete.

Il metodo di Ford-Fulkerson
Taglio

Lemma

Dati f flusso in G con s sorgente e t pozzo, e (S,T) un qualsiasi taglio di G, il flusso attraverso (S, T) è $f(S, T) = |f|$.

Corollario

Il valore di qualsiasi flusso f su una rete G è limitato superiormente dalla capacità di un qualsiasi taglio di G.

Teorema del flusso massimo e taglio minimo

Dati f flusso in una rete $G = (V, E)$ con s sorgente e t pozzo, le seguenti condizioni sono equivalenti:

- f è un flusso massimo in G
- La rete residua G_f non ammette cammini aumentanti
- $|f| = c(S, T)$ per qualche taglio (S, T) di G

Il metodo di Ford-Fulkerson
Pseudocodice

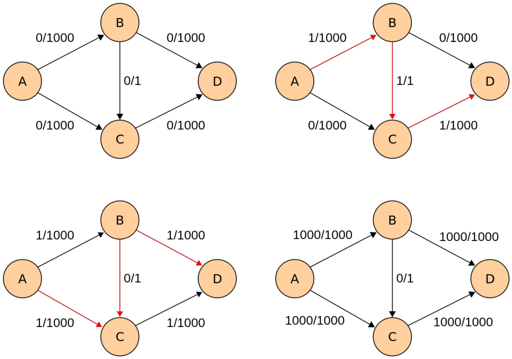
Algorithm 1 Ford-Fulkerson(G,s,t)

for each edge $(u, v) \in E$
    $(u, v).f = 0$
while esiste un cammino aumentante p in G_f
    $c_f(p) = min\ \{c_f(u, v) : (u, v) \text{ sta in } p\}$
    for each edge (u, v) in p
        if $(u, v) \in E$
            $(u, v).f = (u, v).f + c_f(p)$
        else
            $(v, u).f = (v, u).f - c_f(p)$

Il metodo di Ford-Fulkerson
Complessità

Una possibile implementazione del metodo consiste nello scegliere il cammino aumentante alla linea 3 in maniera arbitraria: l'algoritmo che si ottiene termina sempre a patto che le capacità degli archi siano valori razionali.
Per il caclolo della complessità, possiamo ridurci al caso in cui le capacità prendono valori interi.
Il ciclo **for** alle linee 1-2 ha complessità $O(|E|)$.
Se chiamiamo f^* il flusso massimo, abbiamo che il ciclo **while** alla linea 3 verrà eseguito al più $|f^*|$ volte (ad ogni iterazione il valore del flusso aumenterà almeno di una unità). Trovare un cammino nella rete residua costa $O(|V| + |E|) = O(|E|)$ con BFS o DFS.
La complessità dell'algoritmo è quindi $O(|E| \cdot |f^*|)$. L'algoritmo è pseudopolinomiale.

Le prime tre immagini mostrano il primo passo dell'algoritmo e la successiva selezione del cammino aumentante. L'ultima immagine mostra lo stato al quale si arriva dopo 2000 iterazioni.



L'algoritmo di Edmonds-Karp

L'algoritmo di Edmonds-Karp è un'implementazione del metodo di Ford-Fulkerson che ci permette di risolvere il problema del massimo flusso in tempo $O(|V| \cdot |E|^2)$. L'algoritmo utilizza una ricerca in ampiezza per trovare, ad ogni iterazione, il **cammino minimo** da s a t nella rete residua.

Lemma

Dato $G = (V, E)$ con s sorgente e t pozzo, per tutti i $v \in V \setminus \{s, t\}$ abbiamo che la lunghezza del cammino minimo $\delta_f(s, v)$ fra s e v nella rete residua G_f aumenta monotonicamente durante l'esecuzione dell'algoritmo (in particolare, ad ogni aumento del flusso).

L'algoritmo di Edmonds-Karp

Dimostrazione

Supponiamo, per assurdo, che esista un vertice $v \in V \setminus \{s, t\}$ per il quale, in corrispondenza di un aumento del flusso, la distanza da s si riduca.
Chiamiamo f il flusso prima di questo aumento, e f' il flusso risultante.
Sia v il nodo tale che $\delta_{f'}(s, v) < \delta_f(s, v)$ e per il quale $\delta_{f'}(s, v)$ è minima.
Sia inoltre $p = s \rightsquigarrow u \rightarrow v$ un cammino minimo da s a v in $G_{f'}$.
Abbiamo $\delta_{f'}(s, u) = \delta_{f'}(s, v) - 1$.
Per come abbiamo scelto v, sappiamo inoltre che $\delta_{f'}(s, u) \geq \delta_f(s, u)$.
Se $(u, v) \in E_f$ avremmo:

$$\begin{aligned}
\delta_f(s, v) &\leq \delta_f(s, u) + 1 \quad \text{(disuguaglianza triangolare)} \\
&\leq \delta_{f'}(s, u) + 1 \\
&= \delta_{f'}(s, v)
\end{aligned}$$

è una contraddizione, quindi $(u, v) \notin E_f$.

L'algoritmo di Edmonds-Karp

Dimostrazione

Abbiamo mostrato che $(u, v) \notin E_f$, sappiamo inoltre che $(u, v) \in E_{f'}$.
Questo significa che l'aumento di flusso deve aver aumentato il flusso da v a u.
L'algoritmo che stiamo analizzando aumenta sempre il flusso sul cammino minimo, quindi il cammino minimo da s a u in G_f ha (v, u) come ultimo arco. Abbiamo quindi:

$$\begin{aligned} \delta_f(s, v) &= \delta_f(s, u) - 1 \\ &\leq \delta_{f'}(s, u) - 1 \\ &= \delta_{f'}(s, v) - 2 \end{aligned}$$

Questo contraddice la nostra assunzione iniziale, cioè $\delta_{f'}(s, v) < \delta_f(s, v)$.
Non esiste quindi un nodo v con queste caratteristiche.

L'algoritmo di Edmonds-Karp

Teorema

Data una rete di flusso $G = (V, E)$, l'algoritmo di Edmonds-Karp aumenta il valore del flusso $O(|V| \cdot |E|)$ volte.

Dimostrazione

Un arco (u, v) appartenente ad un cammino aumentante è **critico** se $c_f(p) = c_f(u, v)$. Quando aumentiamo il flusso, tutti gli archi critici appartenenti al cammino aumentante scompaiono dalla rete residua (inoltre, ogni cammino aumentante ha almeno un arco critico).
Quello che mostriamo è che ogni arco può diventare critico al più $|V|/2$ volte.

Dimostrazione

Siano u, v nodi in V tali che $(u, v) \in E$. Quando (u, v) diventa critico per la prima volta, avremo $\delta_f(s, v) = \delta_f(s, u) + 1$.
Dopo aver aumentato il flusso, l'arco (u, v) scomparirà dalla rete residua.
Per far sì che (u, v) ricompaia su un altro cammino aumentante, il flusso da u a v deve diminuire, cosa che succede solamente quando (v, u) compare su un cammino aumentante.
Sia f' un flusso tale che (v, u) compare su un cammino aumentante.
Abbiamo:

$$\begin{aligned}
\delta_{f'}(s, u) &= \delta_{f'}(s, v) + 1 \\
&\geq \delta_f(s, v) + 1 \qquad \text{(Lemma)} \\
&= \delta_f(s, u) + 2
\end{aligned}$$

Questo dimostra che da quando (u, v) diventa critico al successivo momento in cui sarà critico, la distanza da s ad u aumenterà almeno di 2.

Dimostrazione

I nodi intermedi di un cammino minimo da s a u non possono mai contenere s, u o t. Quindi, la sua distanza (dopo ogni iterazione) sarà al massimo $|V| - 2$.
Dopo che (u, v) è diventato critico per la prima volta, può ridiventarlo al più altre $(|V| - 2)/2 = |V|/2 - 1$ volte, per un totale di $|V|/2$ volte.
Visto che il numero di archi che possono comparire in una rete residua è $O(|E|)$, il numero totale di archi critici durante l'esecuzione dell'algoritmo sarà $O(|V| \cdot |E|)$

Come visto precedentemente, possiamo individuare un cammino aumentante utilizzando BFS in tempo $O(|E|)$. La complessità dell'algoritmo è quindi $O(|V| \cdot |E|^2)$.

Algoritmi push-relabel
Algoritmo generico

L'approccio **push-relabel** consente di risolvere il problema del massimo flusso più velocemente rispetto all'approccio di Edmonds-Karp: arriveremo ad un algoritmo di complessità $O(|V|^3)$, uno degli algoritmi più veloci per questo tipo di problema.
Gli algoritmi push-relabel mantengono durante l'esecuzione il vincolo di capacità ma rilassano il vincolo di conservazione del flusso: richiediamo solamente che la quantità di flusso uscente da un nodo (fatta eccezione per sorgente e pozzo) non superi la quantità di flusso entrante:

$$\sum_{v \in V} f(v,u) - \sum_{v \in V} f(u,v) \geq 0$$

Chiamiamo inoltre **eccesso di flusso** la seguente quantità:

$$e(u) = \sum_{v \in V} f(v,u) - \sum_{v \in V} f(u,v)$$

L'idea alla base di questi algoritmi consiste nell'assegnare ad ogni nodo un'altezza, e fare in modo che un nodo u possa aumentare il flusso su un arco (u,v) solamente se v sta "più in basso".

Algoritmo generico

- Inizialmente, alla sorgente assegnamo altezza $|V|$ mentre al pozzo e a tutti gli altri nodi assegnamo 0.
- Assegnamo più flusso possibile agli archi uscenti dalla sorgente e avremo uno o più nodi del grafo aventi eccesso di flusso positivo.
- Scelto uno di questi nodi (u), aumentiamo la sua altezza in modo che superi di 1 l'altezza del "più basso" dei suoi vicini (v) scelto in modo che sia possibile aumentare il flusso su (u, v). A questo punto possiamo assegnare del flusso ad almeno un arco uscente dal nodo u.
- Iterando queste operazioni, arriveremo ad un punto nel quale il pozzo riceve la massima quantità di flusso possibile. Il flusso non è però legale visto che (in generale) può violare il vincolo di conservazione.
- Per renderlo legale permettiamo che l'altezza dei nodi possa superare $|V|$. In questo modo, il flusso in eccesso viene "spinto" di nuovo alla sorgente (cancellazione) e otteniamo un flusso vero e proprio (legale e massimo).

Algoritmi push-relabel
Algoritmo generico

Quando $u.e > 0$, $c_f(u,v) > 0$ e $u.h = v.h + 1$ possiamo eseguire un'operazione di Push:

Algorithm 2 Push(u,v)

1: $\Delta_f(u,v) = min(u.e, c_f(u,v))$
2: **if** $(u,v) \in E$
3: $\quad (u,v).f = (u,v).f + \Delta_f(u,v)$
4: **else**
5: $\quad (v,u).f = (v,u).f - \Delta_f(u,v)$
6: $u.e = u.e - \Delta_f(u,v)$
7: $v.e = v.e + \Delta_f(u,v)$

La linea 5 corrisponde ad una diminuzione del flusso sull'arco (v,u). Questo accade quando (u,v) fa parte della rete residua ma non della rete originale.

Quando $u.e > 0$ e per ogni $(u, v) \in E_f$ si ha $u.h \leq v.h$, possiamo eseguire un'operazione di Relabel:

Algorithm 3 Relabel(u)

1: $u.h = 1 + min\{v.h : (u, v) \in E_f\}$

La precondizione è sufficiente a garantire che esista almeno un arco uscente da u nella rete residua:
$u.e > 0$ implica l'esistenza di almeno un nodo v tale che $(v, u).f > 0$, quindi $c_f(u, v) > 0$ che a sua volta implica $(u, v) \in E_f$.

Algoritmi push-relabel
Algoritmo generico

Algorithm 4 Initialize(G,s)

1: **for** each vertex $v \in V$
2: $\quad v.h = 0$
3: $\quad v.e = 0$
4: $s.h = |V|$
5: **for** each $(u, v) \in E$
6: $\quad (u, v).f = 0$
7: **for** each vertex $v \in s.Adj$
8: $\quad (s, v).f = c(s, v)$
9: $\quad v.e = c(s, v)$
10: $\quad s.e = s.e - c(s, v)$

La linea 10 serve ad impedire che l'algoritmo inizi a ciclare scegliendo s come nodo dal quale effettuare operazioni di push.

Algoritmi push-relabel
Algoritmo generico

Algorithm 5 Generic-push-relabel(G)

```
1: Initialize(G,s)
2: while esiste un'operazione Push o Relabel applicabile
3:     seleziona un'operazione ed eseguila
```

E' possibile mostrare che questo algoritmo ha complessità asintotica pari a $O(|V|^2 \cdot |E|)$.

L'algoritmo **relabel-to-front** è un algoritmo push-relabel nel quale viene specificato l'ordine con il quale applicare le operazioni di base (Push e Relabel) tramite un'opportuna struttura dati.
Questo accorgimento permette di ottenere una complessità pari a $O(|V|^3)$ (l'algoritmo generico ha complessità $O(|V|^2 \cdot |E|)$).
Una **Neighbor List** $u.N$ per il nodo u è una linked list contenente (tutti) i vicini di u (tutti i nodi tali che $(u, v) \in E$ oppure $(v, u) \in E$).
Con l'attributo $u.N.head$ puntiamo al primo elemento della lista mentre $v.next\text{-}neighbor$ punta all'elemento che segue v all'interno della lista.
$u.current$ punta al nodo che sto attualmente analizzando in $u.N$.

Algoritmi push-relabel
Algoritmo relabel-to-front

Algorithm 6 Discharge(u)

```
while u.e > 0
    v = u.current
    if v == NIL
        Relabel(u)
        u.current = u.N.head
    elseif c_f(u, v) > 0 ∧ u.h == v.h + 1
        Push(u,v)
    else
        u.current = v.next-neighbor
```

Algoritmi push-relabel
Algoritmo relabel-to-front

Per definire l'algoritmo completo, utilizziamo un'ulteriore lista L contenente tutti i nodi eccetto sorgente e pozzo.

Algorithm 7 Relabel-to-front(G,s,t)

```
Initialize(G,s)
L = V \ {s, t}
for each vertex u ∈ V \ {s, t}
    u.current = u.N.head
u = L.head
while u ≠ NIL
    old-height = u.h
    Discharge(u)
    if u.h > old-height
        move u to the front of L
    u = u.next
```

Complessità

Lemma

Il numero totale di operazioni **relabel** che eseguiamo è al più $2|V| - 1$ per ogni vertice. In totale è minore di $2|V|^2$.

Lemma

Il tempo totale che spendiamo per le operazioni di **relabel** è limitato da $O(|V| \cdot |E|)$.

Lemma

Durante l'esecuzione dell'algoritmo, il numero totale di operazioni di **push** tali che l'arco viene **saturato** è limitato superiormente da $|V| \cdot |E|$.

Algoritmi push-relabel
Complessità

Teorema

L'algoritmo relabel-to-front ha complessità $O(|V|^3)$.

Dimostrazione