Appunti lezione – Capitolo 15
Ricerca locale

Alberto Montresor

03 Giugno, 2016

1 Introduzione alla ricerca locale

Un approccio miope, ma talvolta efficace è quello della *ricerca locale*. L'idea è la seguente: se si conosce una soluzione ammissibile (ma non necessariamente ottima) ad un problema di ottimizzazione, si può cercare di trovare una soluzione migliore nelle "vicinanze" di quella precedente (nell'intorno $I(Sol)$ della soluzione *Sol*). Si continua in questo modo fino a quando non si è più in grado di trovare soluzioni migliori.

```
ricercaLocale()
    Sol ← una soluzione ammissibile del problema
    while ∃S ∈ I(Sol) migliore di Sol do
        Sol ← S
    return Sol
```

E' sempre possibile che un approccio del genere porti a "cadere" in un minimo locale, da dove non è più possibile uscire. Ma ci sono invece casi in cui è possibile dimostrare che il minimo globale viene sempre raggiunto. Uno di questi casi riguarda i problemi di flusso.

2 Problema di flusso

Nei problemi di cammino minimo, si cercava di minimizzare il costo di attraversamento di una rete. Attraversare un arco (u, v) aveva un costo $w(u, v)$, indipendentemente dalla quantità di "materiale" che attraversa l'arco.
Ma per molti altri problemi, il problema non è minimizzare il cammino, ma piuttosto massimizzare la quantità di materiale che attraversa la rete. Questa visione è complementare a quella dei cammini minimi, e si basa sul concetto di capacità.
Esempi classici: rete di comunicazione, la capacità di un collegamento è detta banda. Rete idraulica, la capacità di una conduttura è la quantità di liquidi che può attraversarla in unità di tempo.

3 Definizioni

Definizione 1. Una *rete di flusso* $G = (V, E, s, p, c)$ è data da un grafo orientato $G = (V, E)$, da una coppia di vertici detti *sorgente* s e *pozzo* p, e da una funzione di capacità $c : V \times V \to \mathbb{R}^{\geq 0}$, tale per cui $c(u, v) = 0$ se $(u, v) \notin E$.

Sebbene le soluzioni che vedremo possono essere applicate in modo generale, possiamo assumere che per ogni vertice $v \in V$, esista un cammino da s a p che passa per v: $s \rightsquigarrow v \rightsquigarrow p$.

Definizione 2 (flusso)**.** Un *flusso* in G è una funzione a valori reali $f : V \times V \to \mathbb{R}$ che soddisfa le seguenti proprietà:

- **Vincolo sulla capacità**: per ogni $u, v \in V$, si richiede che $f(u, v) \leq c(u, v)$. Ovvero, il flusso non deve eccedere la capacità sull'arco.
- **Antisimmetria**: per $u, v \in V$, si richiede che $f(u, v) = -f(v, u)$. Il flusso viene definito in questo modo per semplificare la proprietà successiva e altre regole.
- **Conservazione del flusso**: per ogni $u \in V - \{s, p\}$, si richiede che $\sum_{v \in V} f(u, v) = 0$. In altre parole, per ogni nodo, la somma dei flussi entranti deve essere uguale alla somma dei flussi uscenti.

Definizione 3 (Valore di flusso)**.** Il *valore di un flusso* f è definito come:

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

ovvero come la quantità di flusso uscente da s.

Definizione 4. Nel problema di flusso massimo, è data una rete G e una funzione di capacità c, e si vuole trovare un flusso f^* di valore massimo:

$$|f^*| = \max\{|f|\}$$

Definizione 5 (Flusso nullo)**.** Definiamo *flusso nullo* la funzione $f_0 : V \times V \to \mathbf{R}^+ \cup \{0\}$ tale che $f(u, v) = 0$ per ogni $u, v \in V$.

Definizione 6 (Capacità residua)**.** Definiamo *capacità residua* di un flusso f in una rete $G = (V, E, s, p, c)$ una funzione $c_f : V \times V \to \mathbf{R}^+ \cup \{0\}$ tale che $c_f(u, v) = c(u, v) - f(u, v)$.

Si noti che in un flusso non nullo, la capacità residua in una coppia di nodi ha valore positivo o nullo sia sull'arco in avanti, che sull'arco di ritorno (per la definizione di flusso con valori negativi sull'arco di ritorno).

Definizione 7 (Somma di flussi)**.** Per ogni coppia di flussi f_1 e f_2 in G, definiamo il flusso somma $g = f_1 + f_2$ come un flusso tale per cui $g(u, v) = f_1(u, v) + f_2(u, v)$.

4 Reti con sorgenti e pozzi multipli

Nel seguito tratteremo solo problemi con sorgenti uniche e pozzi unici. Ma è semplice estendere questi algoritmi ai casi in cui ci siano più sorgenti e pozzi. E' sufficiente (i) aggiungere una *supersorgente*, collegata con un arco uscente a tutte le sorgenti; (ii) aggiungere un *superpozzo*, collegato con un arco entrante a tutti i pozzi. Tutti gli archi aggiunti hanno capacità infinita.

5 Metodo delle reti residue

Questo metodo viene descritto in maniera informale, senza entrare nei dettagli. E' alla base dei passi successivi.
L'idea è la seguente: si parte da un flusso nullo e lo si aumenta progressivamente fino a quando non diventa un flusso massimo. Si lavora tramite reti residue: dato un flusso f per G, si ottiene una nuova rete residua G_f "sottraendo" il flusso alla rete originale G, e si cerca un nuovo flusso g nella rete residua. E' possibile dimostrare che $f + g$ è un flusso per G, con valore più alto di f. A questo punto si continua, in modo iterativo o ricorsivo.
Definiamo meglio il concetto di rete residua:

Definizione 8 (Rete residua)**.** Data una rete di flusso $G = (V, E, s, p, c)$ e un flusso f su G, possiamo costruire una rete residua $G_f = (V, E_f, s, p, c_f)$, tale per cui $(u, v) \in E_f$ se e solo se $c_f(u, v) > 0$.

Alcune osservazioni: La rete residua è una rete sugli stessi vertici del grafo originale. Per quanto riguarda gli archi, se un arco (u, v) ha un flusso associato $f(u, v)$ tale che $0 < f(u, v) < c(u, v)$, allora questo arco si sdoppierà in G_f in:

- un arco (u, v) con capacità $c_f(u, v) = c(u, v) - f(u, v)$;
- un arco (v, u) con capacità $c_f(v, u) = c(v, u) - f(v, u) = c(v, u) + f(u, v)$;

Il grafo G_f avrà al più il doppio degli archi di G. Si noti che se $f = f_0$, allora $G_f = G$.

integer[ ][ ] maxFlow(GRAPH G, NODE s, NODE p, **integer**[ ][ ] c)

$f \leftarrow f_0$ % Inizializza un flusso nullo
$r \leftarrow c$
repeat
  $g \leftarrow$ trova un flusso in r tale che $|g| > 0$, altrimenti f_0
  $f \leftarrow f + g$
  $r \leftarrow$ Rete di flusso residua del flusso f in G
until $g = f_0$
return f

Lemma 1. Se f è un flusso in G e g è un flusso in G_f, allora $f + g$ è un flusso in G.

Dimostrazione.

- **Vincolo sulla capacità**: sappiamo che

  $$g(u, v) \leq c_f(u, v) : \forall u, v \in V$$

  sommando ad entrambi i lati $f(u, v)$ e sostituendo $c_f(u, v)$ con $c(u, v) - f(u, v)$ otteniamo:

  $$(f + g)(u, v) = f(u, v) + g(u, v) \leq f(u, v) + c_f(u, v) = f(u, v) + c(u, v) - f(u, v) = c(u, v)$$

- **Antisimmetria**: Applicando le condizioni di antisimmetria di f e g, otteniamo:

  $$(f + g)(u, v) = f(u, v) + g(u, v) = -f(v, u) - g(v, u) = -(f(v, u) + g(v, u)) = -(f + g)(v, u)$$

- **Conservazione**: Applicando le condizioni di conservazione di f e g, otteniamo:

  $$\begin{aligned} \sum_{v \in V} (f + g)(u, v) &= \sum_{v \in V} (f(u, v) + g(u, v)) \\ &= \sum_{v \in V} f(u, v) + \sum_{v \in V} g(u, v) \\ &= 0 \end{aligned}$$

  per ogni $u \in V - \{s, p\}$

□

6 Metodo dei cammini aumentanti

Il problema del metodo precedente è il seguente: come trovare un flusso aggiuntivo? Vediamo il metodo suggerito da Ford e Fulkerson fin dal 1956.
L'idea è la seguente:

- Si trova un cammino $C = v_0, v_1, \ldots, v_n$, con $s = v_0$ e $p = v_n$ nella rete residua G_f;

- Si identifica la *capacità del cammino*, corrispondente alla minore capacità degli archi incontrati:

$$c_f(C) = \min_{i=2\ldots n} c_f(v_{i-1}, v_i)$$

  questo ovviamente perché l'arco con capacità minore è il "collo di bottiglia" del cammino. Si noti che se esiste un cammino, $c_f(C) > 0$ perché archi con capacità residua 0 sono eliminati.

- si crea un flusso addizionale g tale che $g(v_{i-1}, v_i) = c_f(C)$; $g(v_i, v_{i-1}) = -c_f(C)$ (per antisimmetria); tutte le altre coppie (x, y) hanno $g(x, y) = 0$.

Alcune annotazioni:

- La scelta del cammino è importante; scelte diverse portano ad algoritmi con prestazioni diverse.
- Il valore negativo assegnato al cammino inverso può potenzialmente "invertire" un flusso già assegnato; questo permette di migliorare una soluzione già trovata, e correggerla (ricerca locale).

6.1 Algoritmo di Ford-Fulkerson

```
integer[][] maxFlow(GRAPH G, NODE s, NODE p, integer[][] c)
NODE u, v                                                    % Indici nodi
integer[][] f ← new integer[][]                              % Flusso parziale
integer[][] g ← new integer[][]                              % Flusso da cammino aumentante
integer[][] r ← new integer[][]                              % Rete residua
foreach u, v ∈ G.V() do
    f[u, v] = 0                                              % Inizializza un flusso nullo
    r[u, v] = c[u, v]                                        % Copia c in r
repeat
    g ← flusso associato ad un cammino aumentante in r, f_0 altrimenti
    foreach u, v ∈ G.V() do
    f[u, v] = f[u, v] + g[u, v]                              % f = f + g
    r ← Rete di flusso residua del flusso f in G
until g = f_0
return f
```

Complessità: Questo algoritmo assume una scelta arbitraria del cammino. Qual è un limite superiore alla complessità?

Lemma 2. Se le capacità della rete sono intere, l'algoritmo di Ford e Fulkerson richiede tempo $O((m + n)|f^*|)$ nel caso peggiore.

Dimostrazione. L'algoritmo parte dal flusso nullo e termina quando il valore totale del flusso raggiunge $|f^*|$. Ogni incremento è positivo, nel senso che il valore del flusso viene incrementato di almeno uno. Ogni ricerca di un cammino richiede una visita del grafo, con costo $O(m + n)$; lo stesso avviene per la somma dei flussi e per il calcolo della rete residua. □

Si noti che esistono casi pessimi che effettivamente hanno questo tempo di esecuzione.

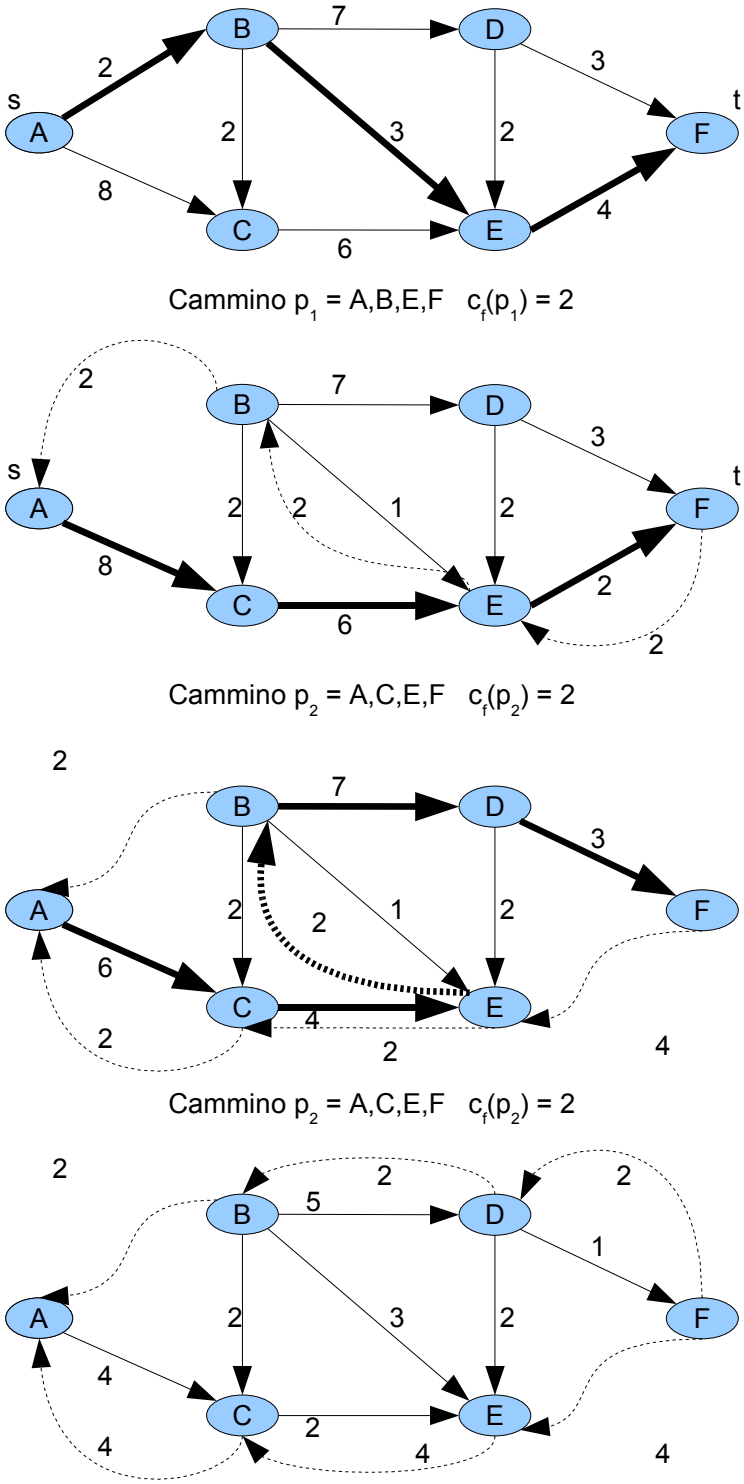

Figura 1: Esempio di funzionamento di Ford-Fulkerson

7 Taglio minimo / flusso massimo

Per dimostrare che il nostro algoritmo termina correttamente, dobbiamo dimostrare che il flusso è massimo se e solo se non esiste un cammino da s a p nella rete residua. Per farlo, abbiamo bisogno ancora di alcune definizioni.

Definizione 9 (Taglio)**.** Un *taglio* (S, P) della rete di flusso $G = (V, E, s, p, c)$ è una partizione di V in S e $P = V - S$ tale che $s \in S$ e $p \in P$.

Definizione 10 (Flusso netto)**.** Se f è un flusso in G, il *flusso netto* $f(S, P)$ attraverso (S, P) è pari a:

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

Definizione 11 (Capacità)**.** La *capacità* $c(S, P)$ attraverso (S, P) è pari a:

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

Si noti che il flusso include valori positivi e negativi, mentre la capacità include solo valori positivi.

Lemma 3. Dato un flusso f e un taglio S, P, la quantità di flusso $f(S, P)$ che attraversa il taglio è uguale a $|f|$.

Dimostrazione.

$$\begin{aligned} f(S,P) &= \sum_{u \in S, v \in P} f(u,v) \\ &= \sum_{u \in S, v \in V} f(u,v) - \sum_{u \in S, v \in S} f(u,v) \\ &= \sum_{u \in S - \{s\}, v \in V} f(u,v) + \sum_{v \in V} f(s,v) - \sum_{u \in S, v \in S} f(u,v) \\ &= 0 + |f| + 0 \end{aligned}$$

dove la prima uguaglianza a zero è vera per la conservazione del flusso, mentre la seconda è vera per antisimmetria. □

Un taglio definisce un limite superiore al flusso massimo che può essere presente nella rete. Nessun flusso attraverso un taglio può superare la capacità del taglio; e come abbiamo visto, il flusso che attraversa un taglio è uguale al valore del flusso. Quindi, il valore del flusso è limitato superiormente dalla capacità di tutti i possibili tagli. In realtà, quello che avviene è che al taglio minimo corrisponde il flusso massimo.
A questo punto, non ci resta che dimostrare questo risultato:

Lemma 4. Le seguenti tre affermazioni sono equivalenti:

1. f è un flusso massimo;
2. non c'è nessun cammino aumentante per G;
3. esiste un taglio S, P tale che $c(S, P) = |f|$.

Dimostrazione. Dimostriamo circolarmente che $1 \Rightarrow 2$, $2 \Rightarrow 3$ e $3 \Rightarrow 1$.

- $1 \Rightarrow 2$: se esistesse un cammino aumentante, il flusso potrebbe essere aumentato e quindi non sarebbe massimo (assurdo);

- $2 \Rightarrow 3$: consideriamo la rete residua G_f. Poiché non c'è nessun cammino aumentante per f, non c'è nessun cammino da s a p in G_f. Sia S l'insieme dei vertici raggiungibili da s e $P = V - S$; ovviamente $s \in S$ e $p \in P$, quindi (S, P) è un taglio. Per il lemma precedente,

$$|f| = \sum_{u \in S, v \in P} f(u, v)$$

  Poiché P non è raggiungibile da s in G_f, tutti gli archi (u, v) con $u \in S$ e $v \in P$ sono saturati; ovvero, $f(u, v) = c(u, v)$. Ne segue che:

$$|f| = \sum_{u \in S, v \in P} f(u, v) = \sum_{u \in S, v \in P} c(u, v)$$

- $3 \Rightarrow 1$: poiché un qualsiasi flusso f e un qualsiasi taglio (S, P) soddisfa la relazione $|f| \leq c(S, P)$, il flusso che soddisfa $|f| = c(S, P)$ deve essere massimo.

□

8 Algoritmo di Edmonds e Karp

Edmonds e Karp hanno suggerito di implementare la ricerca di un cammino con una visita in ampiezza, che corrisponde alla ricerca di un cammino minimo (nel senso del numero di archi, non dei pesi) fra s e p.
Utilizziamo la notazione $\delta_f(s, v)$ per denotare la distanza minima da s a v in una rete residua G_f.

Lemma 5. Si consideri un'esecuzione dell'algoritmo di Edmonds e Karp in una rete $G = (V, E, s, p, c)$; la distanza $\delta_f(s, v)$ aumenta monotonicamente per ogni aumento di flusso

Dimostrazione. In altre parole, non è possibile che aumentando il flusso la distanza diminuisca. La dimostrazione si trova sul libro, qui vediamo solo un'intuizione - parziale. Quando viene aggiunto un cammino, alcuni archi si "spengono": hanno una capacità residua di 0. La sparizione di un'arco (che prima era utilizzata come cammino minimo) non può rendere più corto un cammino; al limite, un cammino di uguale lunghezza si rende disponibile.
Nuovi archi si aggiungono, ma questi sono archi "all'indietro", che permettono semplicemente di tornare sui propri passi. □

Lemma 6. Il numero totale di aumenti di flusso eseguiti dall'algoritmo di Edmonds e Karp è $O(mn)$.

Dimostrazione. Sia G_f una rete residua. Sia C un cammino aumentante di G_f. Diciamo che (u, v) è un arco critico in C se $c_f(u, v) = c_f(C)$. In ogni cammino esiste almeno un arco critico.
Poiché i cammini aumentanti sono cammini minimi, abbiamo che

$$\delta_f(s, v) = \delta_f(s, u) + 1$$

Una volta aggiunto il flusso associato al cammino aumentante, l'arco critico scompare dalla rete residua. Potrà ricomparire se e solo se il flusso lungo l'arco diminuirà, e questo accade se e solo se (v, u) appare in un cammino aumentante.
Sia g il flusso quando questo accade; come sopra, abbiamo:

$$\delta_g(s, u) = \delta_g(s, v) + 1$$

Ma per il lemma precedente, abbiamo anche che $\delta_f(s, v) \leq \delta_g(s, v)$; quindi:

$$\begin{aligned} \delta_g(s, u) &= \delta_g(s, v) + 1 \\ &\geq \delta_f(s, v) + 1 \\ &\geq \delta_f(s, u) + 2 \end{aligned}$$

In altre parole, dal momento in cui un nodo è critico al momento in cui può tornare critico un'altra volta il cammino minimo si è allungato almeno di due passi. La lunghezza massima del cammino fino a u, tenuto conto che poi si deve ancora seguire l'arco (u, v), è $n - 2$. Quindi un arco può diventare critico al massimo $n/2 - 1$ volte. Poiché ci sono $O(m)$ archi che possono diventare critici $O(n)$ volte, abbiamo che il numero massimo di flussi aumentanti è $O(mn)$. Poiché ogni visita in ampiezza costa $O(m)$, il costo totale è $O(m^2n)$. □

9 Abbinamento (matching) massimo nei grafi bipartiti

Alcuni problemi combinatori possono essere trattati come problemi di flusso, se opportunamente "adattati". Dato un grafo non orientato $G = (V, E)$, un abbinamento (matching) è un sottoinsieme M di archi tale che, per ogni vertice $u \in V$, al massimo un arco di M sia incidente su v. Un abbinamento massimo è un abbinamento con cardinalità massima.
Limitiamoci a considerare qui i grafi bipartiti, grafi in cui sia l'insieme V è diviso in due sottoinsiemi disgiunti L e R, e ogni arco in E collega un nodo di L a un nodo di R.
Il problema di trovare un abbinamento massimo in un grafo bipartito ha molte applicazioni pratiche. Ad esempio, abbinare un insieme di L macchine a un insieme R di processi da eseguire contemporaneamente. La presenza di un arco significa che una particolare macchina $u \in L$ è in grado di eseguire un particolare processo. Un abbinamento massimo mette al lavoro il maggior numero di macchine.

9.1 Soluzione

Si costruisce una rete di flusso associata $G' = (V', E', s, p, c)$, dove:

- s e p sono nodi nuovi, che rappresentano la sorgente e il pozzo;
- $V' = \{s\} \cup L \cup R \cup \{t\}$
- $E' = \{(s, u)|u \in L\} \cup \{(u, v)|u \in L, v \in R, (u, v) \in E\} \cup \{(v, p)|v \in R\}$; ovvero tutti gli archi in E sono ora direzionati da L a R, e sono stati aggiunti archi che collegano s a L e R a p.
- $c(u, v) = 1$ se $(u, v) \in E'$, 0 altrimenti;

9.2 Complessità

Possiamo utilizzare per risolvere il problema in G'; è facile vedere che $|E'| = \Theta(E)$, e che il flusso massimo è $O(n)$; quindi, Ford-Fulkerson viene eseguito in tempo $O(mn)$.

10 Altri algoritmi

- Algoritmi push-relabel: $O(n^2m)$
- Algoritmi relabel-to-front: $O(n^3)$
- Algoritmo di Goldberg e Rao: $O(\min(n^{2/3}, m^{1/2})m \log(n^2/m+2) \log C)$, dove $C = \max_{(u,v)\in m)} c(u, v)$.
- Algoritmo di Orlin (2013!): $O(mn)$

11 Varianti

- Minimum-cost flow problem
- Minimum-cost maximum-flow problem