Caroselli Stefano

ALGORITMI & STRUTTURE DATI

A.A. 2002 – 2003

prof.ssa Morgana

TESINA Nº 4 :

"Problema del flusso massimo:
algoritmo Lift-to-Front"

I N D I C E

I . ESPOSIZIONE DEL PROBLEMA E DEFINIZIONI DI BASE

Una ***Rete di Flusso*** è una rete che descrive il comportamento e il percorso del flusso di un materiale (ad es. un liquido) da una ***sorgente*** ad un ***pozzo***: una quantità di materiale sarà immessa nella rete dalla sorgente, percorrerà la rete, e quando giungerà al pozzo sarà tolta dalla rete.

Ipotizziamo che la sorgente e il pozzo lavorino in tempo costante, in particolare che la quantità di materiale immessa dalla sorgente in un'unità di tempo sia uguale alla quantità di materiale tolta dal pozzo nella stessa unità di tempo; possiamo identificare il ***flusso*** del materiale in un punto della rete come la velocità istantanea del materiale nel punto: più aumenta il flusso, più aumenterà la quantità di materiale prodotto dalla sorgente e arrivata al pozzo nell'unità di tempo.

Si vuole studiare un algoritmo che calcoli il flusso massimo in una data rete, ossia la massima quantità di materiale producibile dalla sorgente, in modo da non creare però accumuli nella rete.

Possiamo rappresentare una rete di flusso con un grafo orientato G = (V, E), dove:

- un vertice s rappresenta la sorgente, un vertice t il pozzo;
- l'insieme degli archi rappresenta i collegamenti da s a t attraverso cui far fluire il materiale;
- i vertici rimanenti rappresentano eventuali incroci e biforcazioni dei collegamenti.

Quindi ∃ un arco (u,v) solo se ∃ un collegamento tramite cui è possibile far scorrere il materiale da u a v; supponiamo inoltre che ogni collegamento (u,v) abbia una propria prefissata ***capacità*** c(u,v), che rappresenti quanto materiale può passare da u a v in una data unità di tempo.

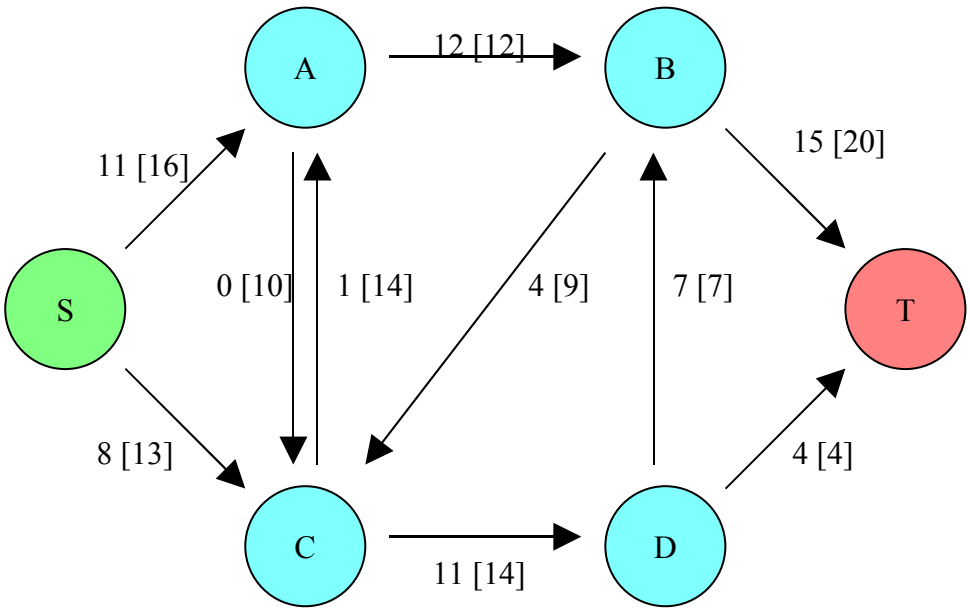

Figura 1. *Un esempio di rete di flusso : s è la sorgente, t il pozzo; per ogni arco (u,v) sono indicati il flusso netto da u a v, e tra parentesi quadre la capacità di (u,v).*

Possiamo a questo punto introdurre una definizione formale di flusso:

DEFINIZIONE (1)

Un ***Flusso*** in G è una funzione $f : V^2 \rightarrow \mathbf{R}$, che soddisfi le seguenti 3 proprietà :

1. **Vincolo di capacità** : $f(u,v) \leq c(u,v) \qquad \forall (u,v) \in E$;
2. **Asimmetria** : $f(u,v) = -f(v,u) \qquad \forall (u,v) \in E$;
3. **Conservazione del flusso** : $\sum_{v \in V} f(u,v) = 0 \qquad \forall u \in V \setminus \{s,t\}$.

dove la quantità $f(u,v)$ è detta ***flusso netto*** da u a v; il ***valore*** del flusso f è $|f| = \sum_{v \in V} f(s,v)$.

Quindi il nostro problema consiste nel trovare un flusso in G di valore massimo.

OSSERVAZIONE

In generale un problema di flusso massimo può avere più sorgenti e più pozzi; ma un problema di questo tipo può essere ricondotto al nostro problema, aggiungendo una ***supersorgente*** S che rifornisca tutte le sorgenti e un ***superpozzo*** T dove scarichino tutti i pozzi; se assumiamo che ognuno di questi nuovi collegamenti abbia capacità ∞, allora il flusso massimo del nuovo problema, da S a T, sarà anche il flusso massimo del problema iniziale, dalle sorgenti s_i ai pozzi t_j .

Sia (u,v) un arco del grafo G, e sia f un flusso di G; la ***capacità residua*** di (u,v) è la quantità $cf(u,v) = c(u,v) - f(u,v)$, che è non negativa per il vincolo di capacità; se $f(u,v)<0$ allora la capacità residua sarà maggiore della capacità c(u,v).

Introduciamo quindi un altro concetto base, che servirà nello studio dell'algoritmo:

DEFINIZIONE (2)

Data una rete di flusso G = (V, E), e un flusso f, la ***Rete Residua*** di G indotta da f è la rete $Gf = (V, Ef)$ dove $Ef = \{(u,v) \in V^2 : cf(u,v)>0\}$. Ogni arco di Ef è detto ***arco residuo***.

La rete residua Gf è essa stessa una rete di flusso, dove le nuove capacità sono date dalle capacità residue.

II . CENNI SUGLI ALGORITMI DI FORD-FULKERSON & EDMONDS-KARP

Un semplice algoritmo (anche se non molto efficiente) per la risoluzione di questo problema è rappresentato dal *metodo Ford-Fulkerson*, che di seguito è brevemente descritto.

Diamo ora alcune semplici definizioni: sia G una rete di flusso, e f un flusso; un ***cammino aumentante*** p è un cammino semplice da s a t in Gf; la ***capacità residua*** di p è la minima capacità tra quelle degli archi di p.

Un ***taglio*** (S, T) di G è una partizione di V in S e T = V \ S, t.c. $s \in S$ e $t \in T$; il ***flusso netto*** attraverso il taglio è $f(S, T) = \sum_{u \in S,\, v \in T} f(u,v)$, e la ***capacità del taglio*** è $c(S, T) = \sum_{u \in S,\, v \in T} c(u,v)$.

Da queste definizioni segue un importante teorema, su cui si basa l'algoritmo di Ford-Fulkerson:

TEOREMA (1) ***(flusso massimo taglio minimo)***

Sia f è un flusso della rete G; allora le seguenti affermazioni sono equivalenti:

- f è un flusso massimo in G.
- La rete residua Gf non contiene cammini aumentanti.
- Sia τ la minima tra le capacità di tutti i possibili tagli di G; allora $|f| = \tau$.

Metodo Ford-Fulkerson (in pseudo-codice)

FORD-FULKERSON METHOD (G, s, t)
- inizializza il flusso f a 0
- **while** $\exists$ un cammino aumentante p
- **do** aumenta il flusso f lungo p
- **return** f

Il metodo Ford-Fulkerson è iterativo: all'inizio $f(u,v)=0$ $\forall$arco; ad ogni passo si cerca un cammino aumentante, e si incrementa il flusso lungo questo cammino; l'algoritmo termina quando non ci sono più cammini aumentanti, ossia quando f è un flusso massimo.

Sia f^* il flusso massimo trovato da tale algoritmo; allora il costo totale è $O(|E| \cdot |f^*|)$

Questo algoritmo può essere migliorato se i cammini aumentanti vengono cercati solo tra i cammini minimi da s a t (con una V.I.A.); tale algoritmo così modificato prende il nome di *algoritmo di Edmonds-Karp*, ed ha un costo di $O(|V| \cdot |E|^2)$.

III . ALGORITMI DI PREFLUSSO – INTRODUZIONE E OPERAZIONI BASE

I più veloci algoritmi per il calcolo del flusso massimo si basano sul *metodo dei preflussi*. In questi algoritmi non si esamina l'intera rete residua per trovare un cammino aumentante, ma si considera un vertice alla volta, studiando i suoi vicini nella rete residua. Durante lo svolgimento dell'algoritmo si utilizza una funzione di ***preflusso*** $f : V^2 \rightarrow \mathbf{R}$ che ha le stesse proprietà dei flussi, con una eccezione: la condizione sulla conservazione del flusso viene "rilassata", e viene chiesto solamente che il flusso netto entrante in un vertice diverso dalla sorgente sia non negativo; tale quantità la chiameremo ***flusso in eccesso*** in u: $e(u) \geq 0$. Un vertice u è ***traboccante*** se $e(u) > 0$.

Iniziamo la spiegazione di questo metodo con un esempio.

Consideriamo una rete di tubature per liquidi, di prefissata capacità; il metodo Ford-Fulkerson visto prima aggiunge ripetutamente piccole quantità di flusso nella rete, finché è possibile farlo senza creare accumuli nella rete.

Nel metodo dei preflussi i vertici, che sono i punti di giunzione dei tubi, possiedono due importanti proprietà: (1) ogni vertice ha un tubo di scolo con un serbatoio dove accumulare temporaneamente il flusso in eccesso, e (2) ogni vertice giace ad una altezza prefissata, che non decresce durante l'algoritmo $h: V \rightarrow \mathbf{N}$; all'inizio la sorgente ha altezza |V|, e tutti gli altri vertici hanno altezza 0; durante tutto l'algoritmo deve inoltre non essere violata la condizione:

$h(u) \leq h(v) + 1$ $\forall$ arco residuo (u, v)

L'algoritmo per prima cosa invia quanto più flusso possibile a partire dalla sorgente, una parte del quale sarà accumulato nei serbatoi dei vertici intermedi; per poter inviare del flusso da un vertice intermedio u ad un suo vicino v occorre che $h(u) > h(v)$ (poiché il flusso scorre dal basso verso l'alto), sarà quindi necessario effettuare un'operazione di "innalzamento" di u.

Una volta raggiunto il pozzo, sarà arrivato tutto il flusso che potenzialmente può attraversare la rete rispettando i vincoli di capacità; rimane solo da rispedire alla sorgente il flusso accumulato nei serbatoi, alzando i vertici ad un'altezza superiore a |V|.

Le operazioni base di un algoritmo di preflusso sono le seguenti 2:

1. **Operazione base Push(u ,v) – invio di flusso traboccante.**

Possiamo applicare quest'operazione da un vertice traboccante u ad un suo vicino v nella rete residua, solo se h(u) = h(v) + 1, in quanto non ha senso considerare coppie di vertici con maggiori dislivelli di altezza; essa ha come scopo lo scaricare il più possibile il vertice u, riempiendo il serbatoio di v, ma sempre rispettando i vincolo della capacità residua $c_f(u, v)$.

Quindi verrà inviata una quantità $\delta = \min (e(u), c_f(u, v))$ da u a v, in questo modo:

$f(u, v) \leftarrow f(u, v) + \delta$ $\qquad$ $e(u) \leftarrow e(u) - \delta$

$f(v, u) \leftarrow -f(u, v)$ $\qquad$ $e(v) \leftarrow e(v) + \delta$

Questo invio sarà un ***invio saturante*** se $\delta = e(u)$, sarà ***non saturante*** se $\delta = c_f(u, v)$; se un arco è stato saturato, scompare dalla rete residua.

2. **Operazione base Lift(u) – innalzamento di un vertice.**

Possiamo applicare quest'operazione da un vertice traboccante u solo se $\forall$suo vicino v nella rete residua si ha $h(u) \leq h(v)$, ossia se non è possibile far defluire il flusso in eccesso di u; di conseguenza si ha come scopo alzare u in modo tale che esista almeno un suo vicino v più basso di u, e si opera in questo modo:

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

Diremo quindi che u è stato ***innalzato***; poiché u era traboccante, E_f contiene almeno un arco uscente da u, quindi ha senso calcolare il minimo.

Algoritmo generico dei preflussi (in pseudo-codice)

GENERIC-PREFLOW-PUSH (G)
- inizializza il preflusso f, le altezze e il flusso in eccesso
- **while** $\exists$ un'operazione di base applicabile
- **do** esegui tale operazione
- **return** f

La correttezza di questo algoritmo è giustificata dal seguente:

TEOREMA (2) ***(correttezza dell'algoritmo generico di preflusso)***
Quando l'algoritmo generico di preflusso su una rete di flusso G termina, allora il preflusso f che esso calcola è un flusso massimo per G.

Inoltre con uno studio più approfondito si può dimostrare che un algoritmo generico di preflusso ha un costo di $O(|V|^2 \cdot |E|)$.

IV . ALGORITMO LIFT-TO-FRONT

L'algoritmo *lift-to-front* è un algoritmo di preflusso che applica in un ordine particolare le operazioni base, e gestisce la struttura di dati in modo efficiente, in modo da migliorare il tempo di esecuzione asintotico dell'algoritmo: ha un costo di $O(|V|^3)$.

I vertici della rete vengono memorizzati in una lista, scandita più volte dall'algoritmo partendo dalla testa, esaminando quelli traboccanti: se è possibile scarica il vertice esaminato, altrimenti lo innalza; quando un vertice viene innalzato la procedura lo porta in testa alla lista, e ricomincia da capo la scansione (da qui il nome: Lift-to-Front).

Per semplificare la spiegazione di tale algoritmo diamo qualche altra definizione e qualche utile risultato:

DEFINIZIONE (3)

Siano dati una rete di flusso G = (V, E), un preflusso f, e un'altezza h; un arco (u, v) è un ***arco ammissibile*** se $c_f(u,v)>0$ e $h(u) = h(v)+1$. Altrimenti (u, v) è un arco ***non ammissibile***.

La ***rete ammissibile*** $G_{f_h} = (V, E_{f_h})$ è l'insieme di tutti gli archi ammissibili.

La rete ammissibile consiste quindi di tutti quegli archi per i quali è applicabile l'operazione **Push**; inoltre questa rete è un grafo orientato aciclico (d.a.g.).

LEMMA (1)

Siano dati una rete di flusso G = (V, E), un preflusso f, un'altezza h, e un vertice traboccante u; allora:

(1) se $\exists$ un arco ammissibile (u, v) allora è possibile applicare **Push** (u, v); tale operazione non crea nessun arco ammissibile nuovo, ma può far diventare (u, v) non ammissibile;

(2) se non $\exists$ alcun arco ammissibile uscente da u, è possibile applicare **Lift** (u); dopo tale operazione $\exists$ un arco ammissibile uscente da u, ma nessun non ci sono archi ammissibili entranti in u.

Nell'algoritmo *lift-to-front* gli archi sono organizzati in "liste di vicini" definite in questo modo:

DEFINIZIONE (4)

Data una rete di flusso G = (V, E), la ***lista dei vicini*** N(u) per un vertice $u \in V$ è una lista concatenata contenente tutti i vertici v tali che (u, v) o (v, u) $\in$ E.

Il primo vertice vicino di u è puntato da head(N(u)) = head(u), e i successivi vicini sono collegati per mezzo del campo "vic"; l'ultimo vicino avrà vic = nil. Il vettore cur(u) punta al vicino di u attualmente considerato, quindi inizialmente si ha head(u) = cur(u).

L'algoritmo opera nel seguente modo.

Dopo aver inizializzato il preflusso come in un qualunque algoritmo generico di preflusso, si crea una lista L contenente tutti i vertici della rete, eccetto la sorgente s e il pozzo t. Durante tutto l'algoritmo la lista L è un ordinamento topologico dei vertici nella rete ammissibile; poiché inizialmente $E_{f_h} = \varnothing$, un qualunque ordinamento iniziale dei vertici è accettabile. Si inizializza poi il vettore d'appoggio cur(v) $\leftarrow$ head(v), $\forall$ vertice interno v.

Si inizia quindi ad esaminare la lista L, entrando in un ciclo while, che parte con un valore iniziale u = testa della lista L, e termina solo se u = nil; terminato questo ciclo, il preflusso f è diventato un flusso massimo della rete G.

Ad ogni passo di questo ciclo si applica una procedura, chiamata **scarica(u)**, applicabile solo se u è traboccante, e come effetto rende u non più traboccante. Se questa procedura modifica l'altezza di u, allora tale vertice verrà posto in testa alla lista, e la scansione ricomincerà daccapo.

La procedura **scarica** consiste di un ciclo while, che si ripete finché il vertice in esame u è traboccante; ad ogni passo del ciclo esamina il vicino di u attualmente considerato v = cur(u); può verificarsi uno ed solo uno tra il seguenti casi:

1. v = nil: allora la procedura applica **lift(u)**, e riprende la scansione dall'inizio della lista;
2. (u, v) è un arco ammissibile: allora viene chiamata **push(u, v)**;
3. (u, v) non è ammissibile: si esamina il vicino successivo;

Possiamo osservare come quest'algoritmo sia un algoritmo di preflusso, in quanto applica le operazioni base **lift** e **push** solo quando esse possono essere applicate, in base al Lemma 1, e quando l'algoritmo termina, nessuna operazione di base è più applicabile, in quanto non esistono più vertici traboccanti.

V . COSTO DELL'ALGORITMO LIFT-TO-FRONT

TEOREMA (3)

Il tempo di esecuzione di LIFT-TO-FRONT su di una qualunque rete di flusso G è O ($|V|^3$).

Verrà ora data una dimostrazione del teorema 3, senza soffermarsi su alcuni risultati validi per un qualunque algoritmo di preflusso. Sia n = |V| , m = |E|; in particolare daremo per valido che:

1. Il numero massimo di operazioni di innalzamento per un singolo vertice è O(n); quindi il numero totale di innalzamenti nel corso dell'algoritmo è $O(n^2)$.
2. Le operazioni di innalzamento hanno un costo di O(m·n).
3. Il numero massimo di operazioni di invio saturante è O(m·n).
4. Lemma della "stretta di mano": $\sum_{u \in V} \delta(u) = 2|E|$ essendo δ(u) il *grado* di u in G.

Dimostrazione.

Chiameremo "fase" dell'algoritmo il tempo trascorso tra due **lift** consecutivi. Il numero totale di fasi sarà $O(n^2)$. Ogni fase non può contenere più di n chiamate alla procedura **scarica**: infatti se questa procedura chiama la procedura **lift**, la fase termina; altrimenti si continua a scorrere la lista L, lunga non più di n. Pertanto l'algoritmo invoca la procedura **scarica** $O(n^3)$ volte.

Studiamo quindi la procedura **scarica**. Ogni volta che è chiamata, la procedura compie una delle seguenti azioni:

- Innalzamento **lift(u)**.
  Per le ipotesi fatte, vi è un limite superiore O(m·n) al tempo necessario ad eseguire gli $O(n^2)$ innalzamenti.
- Aggiornamento di cur(u).
  Ogni volta che viene innalzato un vertice u viene chiamata O(δ(u)) volte la procedura di aggiornamento; per ogni vertice u si hanno (n·δ(u)) chiamate. Il numero totale di chiamate sarà: O(Σ n·δ(u)) = O(n · Σ δ(u)) = O(n · 2m) = O(m·n) per il lemma della "stretta di mano".
- Invio **push(u)**.
  Per le ipotesi fatte gli invii saturanti sono in numero di O(m·n). Ogni chiamate di **scarica** può contenere al più un invio non saturante: gli invii non saturanti sono al più $O(n^3)$. Il numero totale di invii è $O(m \cdot n + n^3) = O(n^3)$

Il tempo di esecuzione dell'algoritmo LIFT-TO-FRONT è : $O(m \cdot n) + O(m \cdot n) + O(n^3) = O(n^3)$. ♦