Algoritmi e Strutture Dati

Prof.ssa Rossella Petreschi

Reti di flusso, problema del flusso massimo e preflusso.

A cura di Lorenzo Bellincampi

Indice

Parte I
Flusso massimo

1 Introduzione

Una rete di flusso è uno strumento matematico che, esattamente come un grafo, consente di formalizzare alcuni problemi, e quindi di risolverli più agevolmente. In particolare, le reti di flusso permettono di formalizzare problemi in cui del materiale che è "prodotto in una sorgente", "confluisce in un pozzo", passando per delle vie intermedie. Problemi di questo tipo sono ad esempio lo studio delle tubature idriche oppure lo studio di percorsi commerciali per delle merci.

Nello studio del *flusso* di questo materiale si può essere interessati a diverse problematiche come ad esempio la ricerca del flusso di costo minimo oppure la ricerca del flusso di valore masimo. In questo elaborato si analizza quest'ultimo problema e si presentano alcune delle soluzioni più interessanti.

1.1 Definizioni preliminari

1.1.1 Rete di flusso

Una rete di flusso può essere agevolmente definita utilizzando la teoria dei grafi. In particolare:

Una rete di flusso $G = (V, E)$ è un grafo orientato in cui:

- Esiste un vertice s chiamato *sorgente* in cui non entrano archi.
- Esiste un vertice p chiamato *pozzo* da cui non escono archi
- Ogni coppia di vertici (u, v) ha una capacità $c(u, v)$. Se $(u, v) \in E$ allora $c(u, v) > 0$ altrimenti $c(u, v) = 0$.
- Per ogni vertice v esiste un cammino dalla sorgente al pozzo che passa per v.

Notare che formalmente la capacità è assegnata ad ogni arco da una funzione C.

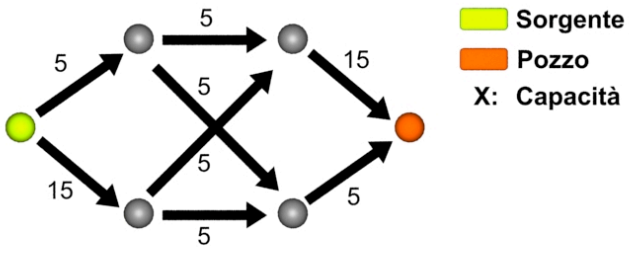

Figura 1: Una rete di flusso

1.1.2 Flusso

Data una rete di flusso è possibile definire un assegnamento di flusso sulla stessa:

Sia $G = (V, E)$ una rete di flusso con sorgente s e pozzo p. Un flusso in G è una funzione f tale che:

- $f : V \times V \to R^+ \cup \{0\}$
- Vale il *vincolo di capacità*: per tutti gli $u, v \in V$ si richiede che $f(u, v) \leq c(u, v)$.
- Vale il *vincolo di antisimmetria*: per tutti gli $u, v \in V$ si richiede che $f(u, v) = -f(v, u)$.
- Vale il *vincolo di conservazione del flusso*: per tutti gli $u \in V - \{s, t\}$ e per tutti i $v \in V$ si richiede che $\sum f(u, v) = 0$.

In parole povere, il vincolo di capacità impone che in ogni arco non possa esserci più flusso di quanto l'arco stesso ne supporti, il vincolo di antisimmetria impone che il flusso netto da un vertice u a un vertice v deve essere l'opposto del flusso netto dal vertice v al vertice u. Il vincolo di conservazione del flusso impone che il flusso netto uscente da un qualsiasi vertice v che non sia la sorgente o il pozzo sia 0.

Per concludere definiamo il *valore di flusso* come il flusso uscente dalla sorgente: $|f| = \sum f(s, v)$.

1.1.3 Flusso massimo

Come accennato in precedenza, quando si ha a che fare con una rete di flusso si può essere interessati ad una problematica anziché un'altra. In questo elaborato si analizza il problema del flusso massimo, ovvero si è interessati, data una rete di flusso, a massimizzare il valore del corrispondente flusso.

La risoluzione di questa problematica è chiaramente di grande interesse per le moltissime applicazioni nella realtà. Si pensi ad esempio a un produttore di merci che è interessato a conoscere la quantità massima di prodotti che può spedire attraverso la rete di trasporti cittadini.

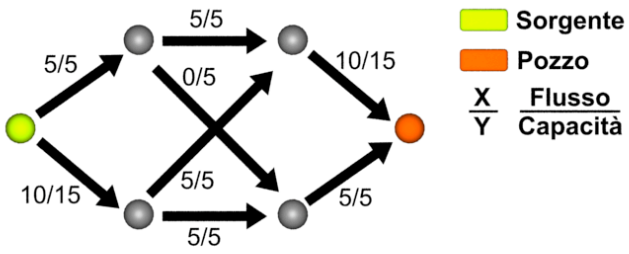

Figura 2: Il problema del flusso massimo

2 Il problema del flusso massimo

Il problema del flusso massimo esiste da molti anni e al giorno d'oggi sono noti molti algoritmi basati su intuizioni diverse per la sua risoluzione. In questo paragrafo vengono dapprima presentati alcuni metodi generici per la risoluzione del problema del flusso massimo e si passa poi a presentare il metodo di Ford-Fulkerson e Edmonds-Karp.

2.1 Il metodo delle reti residue

2.1.1 Definizione

Intuitivamente, data una rete di flusso ed un flusso, la rete residua è formata da tutti gli archi che non sono stati saturati dal flusso, ovvero che hanno ancora la possibilità di "far scorrere materiale".

Per la definizione formale di rete residua abbiamo bisogno di quella di capacità residua:

Dati una rete di flusso $G = (V, E)$ con sorgente s e pozzo p e un flusso in G, per ogni coppia di vertici u, v la quantità di flusso netto addizionale che è possibile inviare da u a v senza violare il vincolo di capacità è chiamata *capacità residua* di (u, v), ed è data da: $c_f(u, v) = c(u, v) - f(u, v)$.

Siamo ora in grado di dare la definizione formale di rete residua:

Dati una rete di flusso $G = (V, E)$ con sorgente s e pozzo p e un flusso in G, la rete residua è $G_f = (V, E_f)$, dove $E_f = \{(u, v) \in V : c_f(u, v) > 0\}$.

2.1.2 Algoritmo

L'idea è molto intuitiva.

Data una rete G con flusso f (non massimo, anche nullo) si prende in considerazione la rete residua G_f e vi si trova un flusso qualunque f^1per poi assegnare a G il flusso $f + f^1$. Il tutto si itera fino a che f non è massimo.

2.1.3 Correttezza

La verifica della correttezza di tale algoritmo si effettua dimostrando che il flusso $f + f^1$rispetti i tre vincoli del flusso. Più formalmente bisognerà dimostrare che se f è un flusso su G e f^1un flusso su G_f, allora $f + f^1$ è un flusso su G con valore $|f + f^1| = |f| + |f^1|$.

Dimostrazione.

1. Vincolo di capacità: $(f + f')(x, y) = f(x, y) + f'(x, y) \leq f(x, y) + c(x, y) - f(x, y) = c(x, y)$
2. Vincolo di antisimmetria: $(f + f')(x, y) = f(x, y) + f'(x, y) = -f(x, y) - f'(x, y) = -(f(x, y) + f'(x, y)) = -(f + f')(x, y)$
3. Vincolo di conservazione: $\sum(f + f')(x, y) = \sum(f(x, y) + f'(x, y)) = \sum f(x, y) + \sum f'(x, y) = 0$

2.2 Il metodo dei cammini aumentanti

2.2.1 Definizioni

- Data una rete G con flusso f un cammino aumentante π è un cammino dalla sorgente al pozzo nella rete residua G_f.
- La quantità massima di flusso netto che può essere inviato su un cammino aumentante π è chiamata capacità residua di π ed è data da $c_f(\pi) = min\{c_f(u,v) : (u,v) \text{ è un arco di } \pi\}$

2.2.2 Algoritmo

Anche in questo caso l'idea è piuttosto intuitiva.

Data una rete G e un flusso nullo f si cerchi un cammino aumentante π in G_f e si scelga f^1 come segue:

- $f^1(u,v) = c_f(\pi)$, se (u,v) è un arco di π.
- $f^1(u,v) = -c_f(\pi)$, se (v,u) è un arco di π.
- $f^1(u,v) = 0$, altrimenti.

Di conseguenza $|f^1| = c_f(\pi)$ e il valore del flusso $f + f^1$su G è pari a $|f| + c_f(\pi)$.

2.3 L'algoritmo di Ford e Fulkerson

Algorithm 1 Algoritmo di Ford e Fulkerson

```
for ogni arco(u,v) ∈ E
    do f(u,v) = 0
       f(v,u) = 0
while esiste un cammino π da s a t in G_f
    do c_f(π) = min{c_f(u,v) : (u,v) è in π}
    for ogni arco (u,v) in π
        do f(u,v) = f(u,v) + c_f(π)
           f(v,u) = −f(u,v)
```

L'algoritmo di Ford e Fulkerson ha un funzionamento molto semplice: dopo aver impostato a 0 tutti i flussi per ogni arco, finché trova un cammino aumentante π aumenta il flusso f della capacità residua $c_f(\pi)$.

La complessità di tale algoritmo, nel caso in cui le capacità della rete siano intere è $O(E|f^*|)$, con f^* = flusso massimo trovato dall'algoritmo.

2.4 L'algoritmo di Edmonds e Karp

Algorithm 2 Algoritmo di Edmonds e Karp

```
for ogni arco(u, v) ∈ E
    do f(u, v) = 0
       f(v, u) = 0
while esiste un cammino minimo π da s a t in G_f
    do c_f(π) = min{c_f(u, v) : (u, v) è in π}
    for ogni arco (u, v) in π
        do f(u, v) = f(u, v) + c_f(π)
           f(v, u) = −f(u, v)
```

Come è possibile osservare dal codice, l'algoritmo di Edmonds e Karp è pressoché identico a quello di Ford e Fulkerson. L'intuizione dei due studiosi è nel fatto che cercando di volta in volta il cammino minimo in G_f la complessità dell'algoritmo diminuisce. Infatti la complessità dell'algoritmo di Edmonds e Karp, nel caso i cammini minimi siano calcolati tramite visita in ampiezza, è $O(VE^2)$.

Parte II
Il preflusso

3 Un metodo generico basato sul preflusso

3.1 Introduzione al metodo del preflusso

Come già visto nella Parte I, le reti di flusso e più in particolare il problema del flusso massimo, possono essere affrontati utilizzando diversi approcci. Quello che verrà esaminato in questa parte è un approccio piuttosto innovativo il cui utilizzo ha portato a quelli che attualmente sono gli algoritmi più prestanti per la risoluzione del problema del flusso massimo.

Tale approccio, dovuto a Goldberg e Tarjan (1988) sulla base di un risultato di Karzanov (1974), è chiamato metodo del preflusso.

3.1.1 Il concetto di preflusso

Il concetto di preflusso, dovuto ad Alexander Karzanov, è estremamente simile a quello di flusso, con la sola differenza che la quantità di flusso totale entrante in un vertice può eccedere la quantità di flusso uscente.

Più formalmente il preflusso è una funzione $f : V \times V \to R^+ \cup \{0\}$ che soddisfa i vincoli di capacità, antisimmetria e un terzo vincolo simile a quello di conservazione del flusso:

- Per ogni vertice $u \in V - \{s\}$, $f(V, u) > 0$

In altre parole il flusso in entrata per un qualsiasi vertice esclusa la sorgente deve essere maggiore di zero.

Per capire meglio l'intuizione che porterà poi alla formalizzazione degli algoritmi immaginiamo il seguente scenario:

- Una rete idrica di tubi (archi) e snodi (vertici).
- Ogni snodo è posizionato su una piattaforma che è in grado di alzarsi e abbassarsi, ed ha a disposizione un serbatoio in cui accumulare il flusso entrante in eccesso.
- La sorgente ha un'altezza maggiore del pozzo e i due vertici speciali hanno altezza fissa.

Il funzionamento di questa peculiare rete idrica è piuttosto semplice: più il serbatoio di uno snodo si riempie e più la piattaforma relativa sale. Come suggerisce la legge di gravità uno snodo può far scorrere acqua verso un suo vicino solo se quest'ultimo ha altezza inferiore al primo.

L'idea quindi è di avviare il meccanismo saturando le tubature collegate direttamente alla sorgente e che quando tutti i serbatoi sono stati svuotati il preflusso sia massimo.

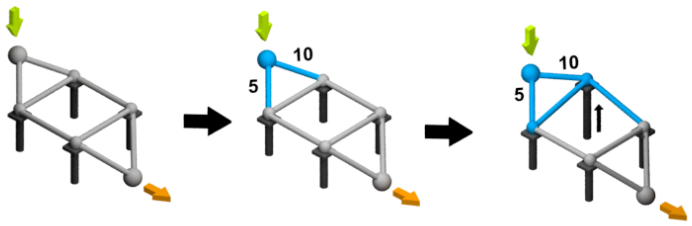

Figura 3: L'idea del preflusso

Più formalmente, l'algoritmo esamina i vertici "interni" alla rete (ovvero tutti tranne la sorgente e il pozzo) e per quelli che hanno un eccesso di flusso (serbatoio pieno) invia tale eccesso ai vertici che ritiene essere più vicini al pozzo (piattaforma più alta). Nel caso in cui il pozzo non sia raggiungibile dal vertice in esame, l'algoritmo invia il flusso in eccetto ai vertici che ritiene essere più vicini alla sorgente.

Questa fase di "smaltimento del preflusso" è necessaria per far si che al termine dell'esecuzione dell'algoritmo il flusso ottenuto sia "legale".

3.1.2 Le operazioni base

L'algoritmo generico di risoluzione del problema del flusso massimo basato sul preflusso, dovuto a Goldberg e Tarjan, fa uso di due operazioni base, Push e Lift, che simulano rispettivamente l'invio di flusso in eccesso e il movimento verticale della piattaforma.

Entrambe le operazioni dipendono direttamente dall'altezza di un vertice che è quindi necessario definire formalmente.

Data una rete G e un preflusso f, una funzione $h : V \to N$ è una **funzione altezza** se

- $h(s) = |V|$ e $h(p) = 0$
- $h(u) \leq h(v) + 1$ per ogni arco residuo(u, v)

Diciamo inoltre che un vertice u è *traboccante* se ha flusso in entrata in eccesso. In formule $e(u) > 0$.

3.1.3 Push

L'operazione Push è quella che invia il flusso in eccesso lungo un arco. Tale operazione, per essere applicata ad un arco (u, v), ha bisogno di tre condizioni:

- u deve essere traboccante.
- $c_f(u, v) > 0$, ovvero la capacità residua dell'arco in questione deve essere positiva.
- $h(u) = h(v) + 1$, ovvero u deve essere più alto di v di un'unità.

Algorithm 3 Operazione Push

AZIONE: Push(u,v) invia $d_f = min(e(u), c_f(u, v))$ unità di flusso da u a v.

$d_f(u, v) = min(e(u), c_f(u, v))$
$f(u, v) = f(u, v) + d_f(u, v)$
$f(v, u) = -f(v, u)$
$e(u) = e(u) - d_f(u, v)$
$e(v) = e(v) + d_f(u, v)$

Nella prima riga l'algoritmo calcola la quantità di preflusso da inviare, nelle righe 2 e 3 viene effettivamente inviato tale preflusso, e nelle ultime due righe vengono aggiornati i valori di flusso in eccesso dei due vertici in questione.

3.1.4 Lift

L'operazione Lift è quella che modifica l'altezza di un vertice, e anche tale operazione ha bisogno di alcune condizioni per poter essere eseguita:

- u deve essere traboccante
- Per ogni vertice v per il quale esiste una capacità residua da u a v non si può inviare flusso da u a v perché v non è più basso di u.

`Algorithm 4 Operazione Lift`

AZIONE: Lift (u) aumenta l'altezza di u

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

Come è possibile vedere dal codice, l'operazione Lift è molto semplice. L'altezza di u diventa 1 più la minima tra le altezze dei vicini di u facenti parte degli "archi traboccanti".

3.1.5 Algoritmo generico

Algorithm 5 Algoritmo di inizializzazione del preflusso (Initialize)

AZIONE: inizializza il preflusso.

```
for ogni vertice u ∈ V
    do h(u) = 0
       e(u) = 0
for ogni arco (u, v) ∈ E
    do f(u, v) = 0
       f(v, u) = 0
h(s) = |V|
for ogni vertice u ∈ Adj(s)
    do f(s, u) = c(s, u)
       f(u, s) = −c(s, u)
       e(u) = c(s, u)
```

Quella appena descritta è la procedura di inizializzazione del preflusso. I primi due cicli for impostano a 0 le altezze e i flussi in eccesso di tutti i vertici e i flussi in ogni arco. La riga "$h(s) = |V|$" imposta l'altezza della sorgente alla cardinalità dell'insieme dei vertici delle rete. L'ultimo for satura gli archi adiacenti alla sorgente inviandoci un flusso pari alla loro capacità.

A questo punto può essere eseguito l'algoritmo di preflusso.

Algorithm 6 Algoritmo generico di preflusso

AZIONE: calcola il flusso massimo tramite il metodo del preflusso.

```
Inizializza il preflusso con l'algoritmo precedente
while è possibile eseguire un'operazione di Push o Lift
    do esegui tale operazione
```

Come accennato nell'introduzione al metodo del preflusso, quindi, tutto ciò che è necessario fare è inizializzare correttamente il "meccanismo" (procedura Initialize) e lasciare che si stabilizzi (eseguire Push e Lift finché è possibile).

3.2 Correttezza del metodo del preflusso

Per dimostrare che questo procedimento è corretto si passerà per la dimostrazione di alcuni lemmi fino a giungere al Teorema di correttezza, alla fine di questa sezione.

Più dettagliatamente, la dimostrazione avverrà in due fasi

1. Si dimostrerà che se il procedimento termina allora il flusso ottenuto è massimo.
2. Si dimostrerà che il procedimento termina in tutti i casi.

3.2.1 Lemma 1

> Sia G una rete di flusso, f un preflusso e h la funzione di altezza.
> Se u è un vertice traboccante allora è possibile applicare un'operazione Push o Lift.

Dimostrazione.

Dato che h è una funzione di altezza, per tutti gli archi residui (u, v) abbiamo che $h(u) \leq h(v) + 1$. Se supponiamo per ipotesi che a u non possa essere applicata una operazione Push, per tutti gli archi residui (u, v) l'altezza di u è minore dell'altezza di v $+1$, il che implica che l'altezza di u è minore o al massimo uguale all'altezza di v. In formule $h(u) \leq h(v)$. Ma allora è possibile applicare l'operazione Lift a u.

3.2.2 Lemma 2

> Durante l'esecuzione dell'algoritmo generico di preflusso l'altezza di un vertice non decresce mai. Inoltre, nel caso venga applicata una operazione Lift, la sua altezza aumenta di almeno 1.

Dimostrazione.

Quando un vertice u viene innalzato dall'operazione Lift siamo sicuri (per i vincoli di Lift) che $h(u) < h(v)$ per tutti i v tali che $(u, v) \in E_f$; ma allora siamo sicuri che $h(u) \leq 1 + min\{h(v) : (u, v \in E_f)\}$. Quindi l'operazione innalzerà sicuramente u.

Nota: le altezze dei vertici variamo solo durante l'operazione Lift.

3.2.3 Lemma 3

> Sia G un rete di flusso.
> Durante l'esecuzione dell'algoritmo generico di preflusso, h è sempre una funzione altezza.

Dimostrazione.

La dimostrazione procede per induzione sul numero di operazioni Push e Lift eseguite.

Caso base: inizialmente la funzione altezza vale $|V|$ per la sorgente e 0 per tutti gli altri vertici. Gli unici archi (u, v) per i quali $h(u) > h(v)+1$ sono quindi quelli adiacenti alla sorgente ($u = s$); ma questi archi sono saturi e quindi non fanno parte della rete residua. Di conseguenza inizialmente h è una funzione altezza.

Passo induttivo: vediamo come si comporta la funzione altezza quando si effettuano le operazioni Push e Lift.

- Push: L'operazione può aggiungere un arco (v, u) ad E_f e può rimuovere (u, v) da E_f. Nel primo caso $h(v) = h(u) - 1$ e quindi h è ancora una funzione di altezza. Nel caso in cui (u, v) sia rimosso da E_f invece, tale rimozione elimina anche il vincolo corrispondente (ovvero che $h(u) \leq h(v) + 1$) e quindi non crea più problemi ad h che rimane così una funzione altezza.

- Lift: Consideriamo l'arco residuo (u, v) uscente da u. Dopo l'operazione Lift siamo sicuri che $h(u) \geq h(v) + 1$. Si consideri ora un arco residuo (w, u) uscente da w. Per il lemma 2 se prima della Lift $h(w) \leq h(u) + 1$, dopo la Lift avremo che $h(w) < h(u) + 1$. Di conseguenza h è ancora una funzione altezza.

3.2.4 Lemma 4

Sia G una rete di flusso, f un preflusso e h la funzione altezza.
Non esiste alcun cammino dalla sorgente al pozzo nella rete residua G_f.

3.2.5 Teorema di correttezza

Se l'algoritmo generico di preflusso termina su una rete di flusso G, il preflusso f che esso calcola è un flusso massimo per G.

Dimostrazione.
Grazie ai lemmi 1 e 3 e al fatto che f è sempre un preflusso, siamo sicuri che nessun vertice ha flusso in eccesso e che quindi f è un flusso. Inoltre per il lemma 4 non ci sono cammini da s a p nella rete residua, e quindi il flusso f è massimo.

3.3 Complessità del metodo del preflusso

La dimostrazione della complessità del metodo del preflusso coincide con quella del fatto che l'algoritmo generico di preflusso termina. Tale dimostrazione procede ponendo un limite superiore al numero di operazioni che l'algoritmo può eseguire.

Un'implementazione base dell'algoritmo generico di preflusso ha complessità $O|V^2E|$.

4 Miglioramento del metodo del preflusso

4.1 Implementazione

Goldberg e Tarjan presentano una semplice implementazione dell'algoritmo generico di preflusso la cui complessità, come accennato poco sopra, è $O|V^2E|$.

4.1.1 Strutture dati

Chiamiamo *arco non diretto* una coppia non ordinata $\{v, w\}$ tale che $(v, w) \in E$ oppure $(w, v) \in E$. Ad ogni arco non diretto associamo i tre valori $c_f(v, w)$, $c_f(w, v)$ e $f(v, w)$ $(= -f(w, v))$.

Ogni vertice v ha una lista degli archi non diretti incidenti, fissa ma in ordine arbitrario. Quindi ogni arco $\{v, w\}$ appare esattamente in due liste.

A ogni vertice v è inoltre assegnato un *arco corrente* $\{v, w\}$, che è candidato per una operazione di Push. Inizialmente l'arco corrente di v sarà il primo arco della sua lista.

Ricordiamo inoltre che un vertice v è traboccante se $e(v) > 0$, ovvero se ha flusso in eccesso.

4.1.2 La funzione push/lift

La funzione push/lift è il cuore dell'implementazione di Goldberg e Tarjan ed è applicabile al vertice v solo se tale vertice è traboccante.

Algorithm 7 Funzione push/lift

```
Sia {v, w} l'arco corrente di v.
if push(v, w) è applicabile then do push(v, w)
else
    if {v, w} non è l'ultimo arco
        della lista di v then
        sostituisci {v, w} con il prossimo ar-
        co nella lista di v
    else
        do rendi il primo arco del-
        la lista di v l'arco corrente di v
        do lift(v)
```

La funzione push/lift controlla se può eseguire una Push sull'arco corrente di v, e in caso la esegue. In caso contrario, se l'arco corrente non è l'ultimo rende arco corrente l'arco successivo nella lista (e quindi nell'iterazione successiva tenterà di effettuarvi una Push). Se invece l'arco corrente è l'ultimo, l'arco corrente diventa il primo arco della lista, e il vertice v viene innalzato tramite una Lift in modo che nel prossimo ciclo di iterazioni alcune Push saranno eseguibili.

4.1.3 Algoritmo

Algorithm 8 Implementazione base di Goldberg e Tarjan

```
while esistono vertici traboccanti
    do push/lift
```

L'algoritmo base non fa altro che eseguire la funzione push/lift fintantoché esistono dei vertici traboccanti o, in altre parole, finché il sistema non è stabile. Questa semplice implementazione porta ad una complessità polinomiale di $O(V^2E)$.

Goldberg e Tarjan riescono però a scendere fino all'$O(V^3)$ ottenuto da Karzanov tramite una semplice osservazione: esaminando la funzione push/lift si può osservare che non c'è alcun tipo di controllo sull'operazione Push, a parte i controlli di applicabilità. Questo porta ad avere alcune "Push non saturanti" che potrebbero essere evitate.

Per diminuire il numero di operazioni Push non saturanti si ricorre all'uso di una coda Q di tipologia First In, First Out e alla modifica del codice. L'algoritmo ripeterà ora la funzione discharge invece che la push/lift.

Algorithm 9 Funzione discharge

```
while e(v) = 0 oppure h(v) non aumenta
    push/lift(v)
    if w diventa traboccante then aggiungilo a Q
if v è ancora traboccante then aggiungilo a Q
```

In parole povere l'algoritmo applica la push/lift a v fino a che il suo flusso in eccesso non è zero oppure la sua altezza aumenta, aggiungendo alla coda, nel processo, tutti i vertici traboccanti che trova. Questa implementazione consente, come accennato poco sopra, di raggiungere una complessità di $O(V^3)$.

4.2 Miglioramento

Come abbiamo appena visto, per uguagliare gli altri metodi di risoluzione del problema del flusso massimo tramite il preflusso ci si è serviti di un'implementazione dell'algoritmo generico di preflusso abbastanza semplice. Di conseguenza è possibile migliorare ancora i tempi di esecuzione effettuando piccole modifiche.

Nell'implementazione vista precedentemente il costo di una Push non saturante è $O(1)$. I due autori decidono di migliorare il tempo di esecuzione totale abbassando ulteriormente tale costo su un certo numero di operazioni. Il metodo scelto è quello del raffinamento delle strutture dati utilizzate e la scelta, in questo caso, è ricaduta sugli alberi dinamici.

4.2.1 Gli alberi dinamici

Gli alberi dinamici sono una particolare struttura dati che consente di memorizzare un insieme di alberi in cui ad ogni vertice v è assegnato un valore reale $g(v)$, e che permette di eseguire su tali alberi diverse operazioni in maniera efficiente. In particolare, l'implementazione utilizzata è quella ideata da Sleator e Tarjan nel 1982 in cui il tempo totale richiesto per una sequenza di n operazioni è $O(nlogk)$ dove k è un limite superiore al massimo numero di nodi in un albero.

Nell'applicazione al problema del flusso massimo gli archi degli alberi dinamici rappresentano un sottoinsieme degli archi correnti dei vertici. In particolare, considerando ogni arco di un albero come diretto verso la radice e chiamando $p(v)$ il genitore di v, un arco corrente $\{v, w\}$ è eleggibile ad essere un arco di albero dinamico (con $p(v) = w$) se $h(v) = h(w) + 1$ e $c_f(u, w > 0)$.

Quelle che seguono sono le operazioni possibili sulla struttura dati utilizzata.

- find-root(v): ritorna la radice dell'albero contenente v.
- find-size(v): ritorna il numero di vertici nell'albero contenente v.
- find-value(v): ritorna $g(v)$.
- find-min(v): ritorna l'antenato w di v di minimo valore $g(w)$.
- change-value(v,x): aggiunge il valore reale x a tutti i $g(w)$ per ogni w antenato di v.
- link(v,w): combina gli alberi contenenti i vertici v e w rendendo w padre di v.
- cut(v): spezza l'albero contente v in due sotto alberi cancellando l'arco tra v e suo padre.

4.2.2 Send

L'algoritmo ad alto livello rimane lo stesso visto precedentemente, che aveva complessità $O(V^3)$, con la differenza che la funzione push/lift è sostituita con la tree-push/lift che vedremo tra poco. Questa procedura fa uso di una funzione ausiliaria che rappresenta il cuore dell'algoritmo, la funzione send.

Algorithm 10 Funzione send

```
while find-root(v)≠v and e(v) > 0
    invia d =
    min(e(v), find−value(find−min(v))) unità di flus-
    so lungo il cammino ad albero da v eseguen-
    do una change-value(v,-d)
    while find-value(find-min(v))=0
        u = find-min(v)
        esegui una cut(u) seguita da una change-
        value(u,∞)
```

La funzione send invia il flusso in eccesso da un vertice non radice v fino alla radice del suo albero, taglia gli archi saturati dall'invio e ripete il tutto finché v non trabocca più oppure è diventato radice.

4.2.3 Tree-push/lift

Algorithm 11 Funzione tree-push/lift

```
Sia v una radice traboccante e {v,w} il suo arco corrente
if h(v) = h(w) + 1 and c_f(v,w) > 0 then
    if find-size(v)+find-size(w)≤ k then
        rendi w padre di v eseguendo una change-value(v,-∞), change-value(v,c_f(v,w)), link(v,w)
        invia il flusso in eccesso da v a w eseguendo una send(v)
    else
        esegui una operazione di push da v a w
        esegui una send(w)
else //ovvero se h(v) ≤ h(w) oppure c_f(v,w) = 0
    if {v,w} non è l'ultimo arco della lista di v then
        sostituisci, come arco corrente, {v,w} con il prossimo arco nella lista
    else
        rendi il primo arco nella lista l'arco corrente
        esegui una cut(u) e una change-value(u,∞) per ogni figlio u di v
        esegui una operazione di lift su v
```

Come si può vedere dal codice la funzione tree-push/lift è sensibilmente più complessa della push/lift. All'interno di questa funzione, eseguibile solo su radici traboccanti, sono esaminati due grandi casi: il primo caso è quello in cui l'arco corrente $\{v, w\}$ di v è eleggibile per una operazione di Push, e il secondo caso è quello in cui tale arco non consente di eseguire la Push.

- Nel primo caso se gli alberi contenenti v e w hanno in totale meno di k vertici vengono collegati rendendo w padre di v e poi viene eseguita una send da v. Se invece gli alberi contengono più di k vertici viene eseguita un'operazione di Push ordinaria da v a w seguita da una send da w.
- Nel secondo caso viene aggiornato l'arco corrente di v e, se necessario, viene innalzato v. Se v è stato innalzato vengono tagliati tutti gli archi

d'albero entranti in v.

4.2.4 Complessità

Tramite l'uso degli alberi dinamici sfruttati come abbiamo visto poco sopra, la complessità totale dell'algoritmo diviene $O((VE + V^3/k)logk)$, che, scegliendo $k = V^2/E$, diventa $O(VElog(V^2/E))$.

5 Conclusioni

5.1 Sintesi

In questo elaborato abbiamo analizzato le reti di flusso con particolare attenzione alla problematica del flusso massimo. Abbiamo visto come è possibile risolvere tale problema utilizzando approcci anche molto diversi tra loro come ad esempio le reti residue e i cammini aumentanti utilizzati da Ford-Fulkerson e da Edmonds-Karp, e successivamente è stato realizzato un approfondimento sul metodo del preflusso ideato da Karzanov e formalizzato e migliorato da Goldberg e Tarjan.

5.2 Considerazioni Generali

Come è ben noto, il problema del flusso massimo e ancor di più quello delle reti di flusso in generale presenta molte questioni di vario tipo e in letteratura si trovano diversi studi per affrontare queste tematiche. Questo elaborato ha come scopo quello di fornire una panoramica sulle principali soluzioni a queste problematiche con particolare attenzione al metodo del preflusso che è tra i più studiati per la ricerca di nuove implementazioni che ne riducano ulteriormente la complessità.

6 Riferimenti

- T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduzione agli algoritmi e strutture dati, 2a ed., cap. 27.
- A. V. Goldberg , R. E. Tarjan, A new approach to the maximum-flow problem, Journal of the ACM (JACM), v.35 n.4, p.921-940, Oct. 1988.
- R. Petreschi, Lezioni e materiale didattico del corso di Algoritmi e Strutture Dati, Informatica, Sapienza, 2010-2011.