SCHOOL OF OPERATIONS RESEARCH
AND INFORMATION ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14853-3801

TECHNICAL REPORT NO. 1460

July 2007

LECTURE NOTES ON NETWORK FLOW SPRING 2004

by

David P. Williamson

Lecture Notes on Network Flow Algorithms

David P. Williamson

Spring 2004

Contents

Preface

The contents of this book are lecture notes from a class taught in the School of Operations Research and Industrial Engineering of Cornell University during the Spring 2004 semester (ORIE 633, Network Flow). The notes were created via the "scribe" system: each lecture one student was appointed as the scribe for that lecture, and was responsible for turning their notes into a LaTeX document. I then edited the notes, and made copies for the entire class. The students in the class who served as scribes were Dhruv Bhargava, Alice Cheng, Ed Hua, Patrick Kongsilp, Sumit Kunnumkal, Retsef Levi, Ivan Lysiuk, Chandrashekar Nagarajan, Christina Peraki, Xin Qi, Mateo Restrepo, Yun Shi, Sam Steckley, Christina Tavoularis, Yankun Wang, Stefan Wild, and Anke van Zuylen. Any errors which remain (or were there to begin with!) are, of course, entirely my responsibility.

David P. Williamson
Ithaca, NY

ORIE 633 Network Flows January 26, 2004

Lecture 1

Lecturer: David P. Williamson *Scribe: Patrick Kongsilp*

1.1 Course overview

The course will cover algorithms for network flow problems. Networks flows is an important subfield of combinatorial optimization, and combinatorial optimization is all about how to make decisions that have discrete choices. It helps us answer natural questions involving networks of one sort or another, such as roads, railway lines, and computer networks. We can answer questions such as: how much stuff can be routed from point A to point B? What paths should it take? How can it be done most cost effectively? How quickly can it be done? What if some fraction of the stuff gets lost along the way?

Less obviously, network flow problems turn out to be useful in modelling problems that don't seem to have anything to do with networks. It can be used to answer questions such as: is my favorite baseball/basketball/hockey team eliminated from winning its division? What types of amino acids should be combined to give a protein that folds into a specified shape? How should prices of goods be set so that a marketplace operates most efficiently? We will see some of these applications during the semester.

In this class we will focus on several things:

- **Efficient algorithms in combinatorial optimization**. That is, algorithms that run in polynomial time. Fortunately for us, many network problems have efficient algorithms. We may look at some NP-hard network problems towards the end of the semester and there we will consider *approximation algorithms*; that is, efficient algorithms that return near-optimal solutions.
- **Central role of linear programming and duality.** LP is one of the fundamental tools in combinatorial algorithms and in thinking about these problems – both in modelling the problems and algorithms for them. Often for flow problems one can think about the problem in purely combinatorial terms, but we will sometimes step back and show LP-based explanations.
- **Problems/algorithms that are either a fundamental piece of background, a useful technique for solving other problems, or an interesting open research direction.**

1.2 The maximum s-t flow problem

We begin with the grandaddy of all network flow problems.

Maximum s-t Flow Problem

- **Input:**
  - Directed graph $G = (V, A)$
  - Capacities $u_{i,j} \geq 0, \forall (i,j) \in A, u_{i,j}$ integer
  - Source node $s \in V$, sink node $t \in V, s \neq t$
- **Goal:** Find an s-t flow that maximizes the net flow out of the source node.

We need to start by defining what we mean by a flow.

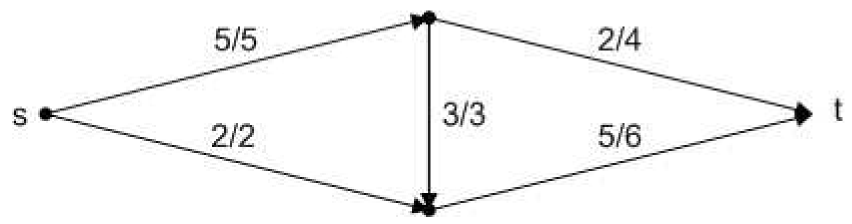
Definition 1.1 An *s-t flow* is a function $f : A \rightarrow \mathbb{R}^{\geq 0}$ s.t.

1. $f_{ij} \leq u_{ij} \qquad \forall (i,j) \in A$ *(capacity constraints)*
2. $\sum_{k:(i,k)\in A} f_{ik} = \sum_{k:(k,i)\in A} f_{ki}, \qquad \forall i \in V, i \neq s, t$ *(flow conservation constraints)*

The second constraint can be alternatively described as "flow in = flow out" for non-source, non-sink nodes.

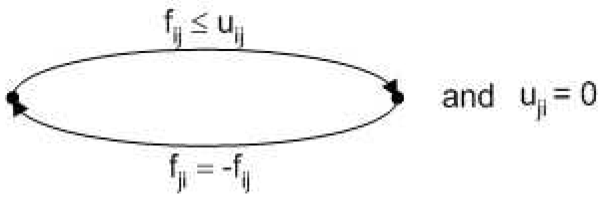
Definition 1.2 The value of a flow f is $|f| \equiv \sum_{k:(s,k)} f_{s,k} - \sum_{k:(k,s)} f_{k,s}$

For the following network below, observe that both flow conditions are met and $|f| = 7$. (The first number above an arc is the flow along the arc, and the second number is the capacity of the arc.)



For notational simplicity, we will use an alternative definition of flow. For this alternate definition, we will assume that if $(i,j) \in A$, then $(j,i) \in A$. In the alternate definition, if there is a flow f_{ij} on arc (i,j) then the flow $f_{ji} = -f_{ij}$ on the reverse arc (j,i). If (j,i) not in the original instance, then we set $u_{ji} = 0$; note that since $f_{ji} = -f_{ij} \leq u_{ji} = 0$, this enforces that the flow on the "original" arc is nonnegative. Pictorially, for each "original" arc $(i,j) \in A$, we have

$$0 \leq f_{ij} \leq u_{ij}$$
$$\Leftrightarrow$$



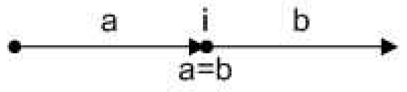
The original constraint for flow conservation,

$$\sum_{k:(i,k)\in A} f_{ik} = \sum_{k:(k,i)\in A} f_{ki}$$

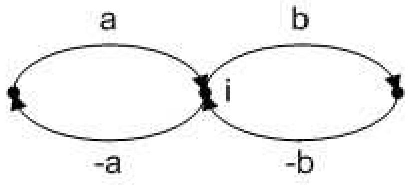
now becomes

$$\sum_{k:(i,j)\in A} f_{ik} = 0.$$

Visually,



$\Leftrightarrow$



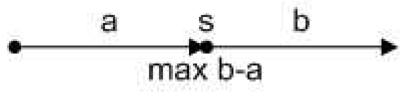
The original expression for the value of the flow of the network,

$$|f| = \sum_k f_{sk} - \sum_k f_{ks}$$

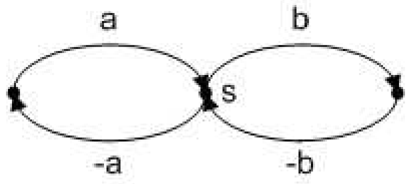
now becomes

$$|f| = \sum_k f_{sk}$$

Visually,



$\Leftrightarrow$



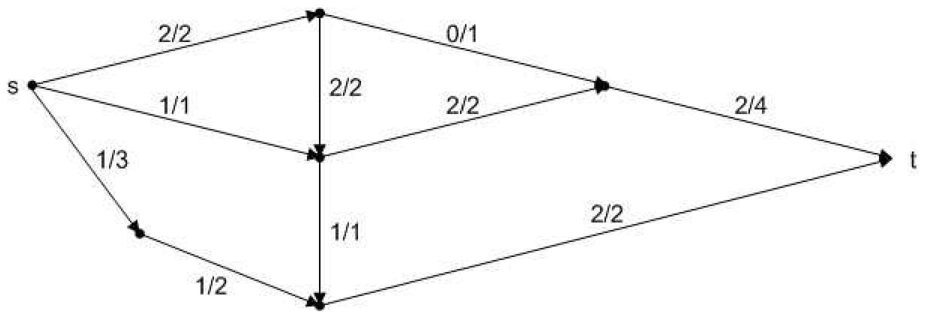
To summarize, the alternate definition of network flow is as follows.

Definition 1.3 An *s-t flow* $f : A \to \mathbb{R}$ s.t.

1. $f_{ij} \le u_{ij}, \qquad \forall (i,j) \in A$ *(capacity constraints)*
2. $f_{ij} = -f_{ji}, \qquad \forall (i,j) \in A$ *(anti-symmetry)*
3. $\sum_{k:(i,k)\in A} f_{ik} = 0, \qquad \forall i \in V, i \ne s, t$ *(flow conservation constraints)*

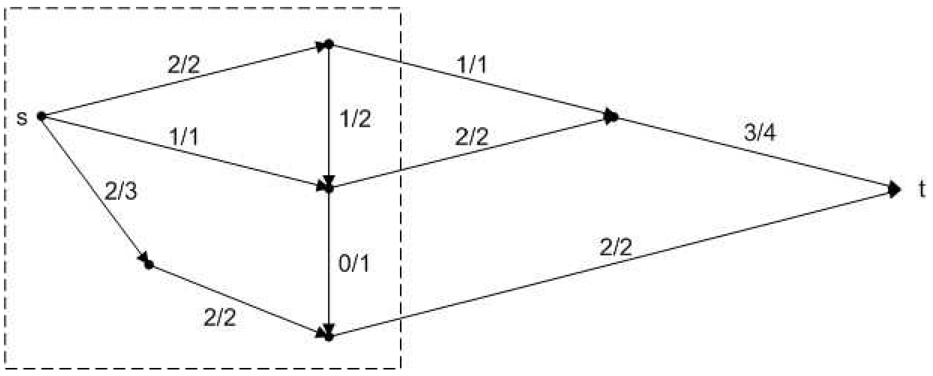
Definition 1.4 The value of a flow f is $|f| \equiv \sum_{k:(s,k)} f_{sk}$

Example:



Question: Observe that $|f| = 4$. Is this a maximum flow? One reason for thinking that perhaps it is a maximum flow is because on every s-t path there is some arc that has reached its capacity, so we cannot augment flow along some s-t path.

However, it is not a maximum flow, because we can give a flow f where $|f| = 5$, as below.



Is this a maximum flow? In this case we can show that this is a maximum flow be demonstrating an appropriate *s-t cut*. Intuitively, we can cut the network into one set of nodes containing the source node and another set of nodes containing the sink node. We want to send as much flow across this cut. Obviously, the amount of flow we can send is bounded by the total capacities of arcs that cross some cut. Specifically, the cut in the example has value 5, so the value of the maximum s-t flow can't be more than 5. Since we found a flow of value 5, we've found a maximum flow.

First, we need to formalize this intuition of cut and the capacity of a cut bounding the value of a flow.

Definition 1.5 An *s-t cut* is a set $S \subseteq V$ s.t. $s \in S, t \notin S$.

Definition 1.6 $\delta^+(S) = \{(i,j) \in A : i \in S, j \notin S\}$

Definition 1.7 The *capacity* of an s-t cut S is $u(\delta^+(S)) \equiv \sum_{(i,j)\in\delta^+(S)} u_{ij}$.

Lemma 1.1 For any s-t cut S and any flow f, $|f| \leq u(\delta^+(S))$.

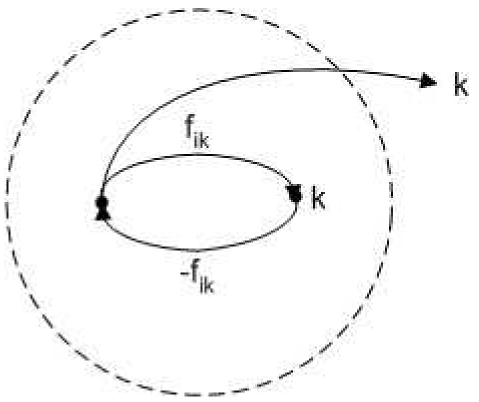
Proof:

$$\begin{aligned}
|f| &= \sum_{k:(s,k)\in A} f_{sk} + 0 \\
&= \sum_{k:(s,k)\in A} f_{sk} + \sum_{i\in S, i\neq s} \sum_{k:(i,k)\in A} f_{ik} \qquad (1.1) \\
&= \sum_{i\in S} \sum_{k:(i,k)\in A} f_{ik} \\
&= \sum_{i\in S} \left(\sum_{k:(i,k)\in A, k\in S} f_{ik} + \sum_{k:(i,k)\in A, k\notin S} f_{ik} \right) \\
&= \sum_{k:(i,k)\in A, k\notin S, i\in S} f_{ik} \qquad (1.2) \\
&\leq \sum_{k:(i,k)\in A, k\notin S, i\in S} u_{ik} = u(\delta^+(S))
\end{aligned}$$

Equality (1.1) follows from the flow conservation constraints. Equality (1.2) follows from the fact that

$$\sum_{i\in S} \sum_{k:(i,k)\in A, k\in S} f_{ik} = \sum_{k:(i,k)\in A, i,k\in S} f_{ik} = 0.$$

This follows from the flow conservation constraint and the fact that $i, k \in S$; by anti-symmetry, f_{ik} will be cancelled out by $f_{ki} = -f_{ik}$.



□

Now, back to the question of how to decide whether or not we have a maximum flow. Did we just get lucky in the example above in which we had a cut whose capacity was equal to the value of the flow? A famous theorem says no; there is always a cut whose value is equal to the value of a maximum flow.

Definition 1.8 A *minimum s-t cut* S^* is an s-t cut S^* s.t.

$$u(\delta^+(S^*)) = \min_{S \subseteq V, s \in S, t \notin S} u(\delta^+(S))$$

Theorem 1.2 (Ford, Fulkerson 1955) The value of a maximum s-t flow equals the capacity of a minimum s-t cut.

Lex Schrijver has written a history of combinatorial optimization that shows that Ford and Fulkerson's initial interest in flows was motivated by looking at the total capacity of railway lines to ship goods between two points. After some digging, Lex discovered that their work arose from an Air Force application that was in fact interested in finding a minimum cut: an "interdiction" of railway lines in Eastern Europe that would cut shipments between the Soviet Union and its satellite states.

ORIE 633 Network Flows January 28, 2004

Lecture 2

Lecturer: David P. Williamson *Scribe: Sumit Kunnumkal*

2.1 The maximum s-t flow problem

We continue our discussion of the max flow problem. Recall that the input is a directed graph $G = (V, A)$. Each arc $(i, j) \in A$ has a capacity $u_{ij} \geq 0$, which is assumed to be integer. There is a designated source s and sink t. We assume that $\forall (i, j) \in A \; \exists (j, i) \in A$. With this assumption, a flow has the following properties:

(i) $f_{ij} \leq u_{ij}$

(ii) $f_{ij} = -f_{ji}$

(iii) $\sum_{k:(i,k)\in A} f_{ik} = 0 \; \forall i \in \; V - \{s, t\}$

Our goal is to maximize the net flow out of the source $|f| = \sum_{k:(s,k)\in A} f_{sk}$. Also recall that an s-t cut is a set $S \subset V$ such that $s \in S, t \notin S$, and the capacity of the cut is denoted $u(\delta^+(S))$. We had stated the following theorem in the last lecture:

Theorem 2.1 (Ford, Fulkerson '55) The value of a maximum s-t flow equals the capacity of a minimum s-t cut.

We now turn to proving the theorem.

First, we introduce the concept of a residual graph, which will be useful in proving the theorem.

Definition 2.1 Given a flow f, the *residual graph* G_f is the graph (V, A_f, u^f), where $A_f = \{(i, j) \in A : f_{ij} < u_{ij}\}$ and $u^f_{ij} = u_{ij} - f_{ij}$.

We call an arc $(i, j) \in A_f$ a *residual arc* and call u^f_{ij} the *residual capacity* of arc (i, j). A_f is therefore the subset of arcs with positive residual capacity.

Figure 2.1 shows the flow on a network and the associated residual graph. By our convention that $f_{ji} = -f_{ij}$, if a flow uses an arc (i, j) at less than its full capacity, the residual graph has a residual arc (i, j) of residual capacity $u_{ij} - f_{ij}$ and a residual arc (j, i) of residual capacity $u_{ji} - f_{ji} = 0 - f_{ji} = f_{ij}$. Intuitively, the residual capacity of (i, j) is the extra flow we can send forward on arc (i, j), while the residual capacity of (j, i) corresponds to decreasing the flow on (i, j); one can thinking of "sucking back" the flow on (i, j).

We showed in the last lecture that the flow in Figure 2.1 was not maximum even though every s-t path in the original graph had an arc with flow at its capacity. However, notice that there is an s-t path in the residual graph with positive residual capacity. This leads us to the definition of an augmenting path.

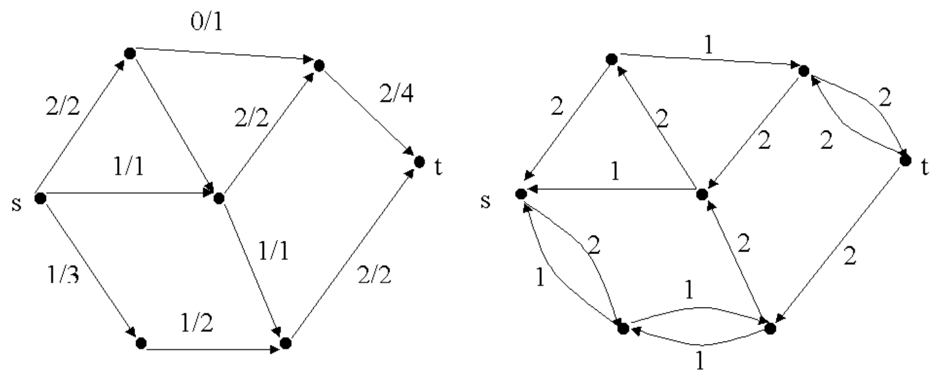


Figure 2.1: A flow and its associated residual graph.

Definition 2.2 A directed path from s to t in the residual graph G_f is called an *augmenting path*.

Given a flow f, residual graph G_f and an augmenting path P, let $\delta = \min_{(ij)\in P} u_{ij}^f$ – the smallest residual capacity of arcs in the augmenting path P. In the above example, $\delta = 1$. Define a new flow f' such that:

$$f'_{ij} \longleftarrow \begin{cases} f_{ij} + \delta & \forall (i,j) \in P \\ f_{ij} - \delta & \forall (j,i) \in P \\ f_{ij} & \text{o.w.} \end{cases}$$

We now claim the following.

Claim 2.2 f' is a valid flow with $|f'| = |f| + \delta$.

This is easy to see. f' respects the arc capacity constraints since we never increased the flow beyond the residual capacity of any arc. Flow conservation constraints are still satisfied since flow is augmented along a path from s to t, and for any node other than s or t along the path, the flow entering the node, and the flow leaving the node both increase by δ. Finally, the s-t flow increases by δ since the net flow out of s increases by δ. So, we see that while there exists an augmenting path in the residual network, it is possible to increase the s-t flow. Thus, we have the following theorem:

Theorem 2.3 The following are equivalent:

(i) f is a max-flow;

(ii) There is no augmenting path in G_f;

(iii) $|f| = u(\delta^+(S))$ for some s-t cut S.

Proof:

(i) ⇒ (ii) We showed that $\neg(ii) \Rightarrow \neg(i)$, since if there is an augmenting path in G_f, then we can increase the current flow and so it is not maximum.

(ii) ⇒ (iii) Let S be the set of all vertices reachable from the source s in G_f. We know $t \notin S$ since otherwise there would be an augmenting path from s to t in G_f. Note that for any $(i,j) \in A$ s.t $i \in S$ and $j \notin S$, $f_{ij} = u_{ij}$. This follows since if $f_{ij} < u_{ij}$, then arc (i,j) would be present in the residual graph as it has positive residual capacity. This implies that j can be reached from the source, which is a contradiction.

Writing $|f|$ as:

$$\begin{aligned}
|f| &= \sum_{k:(s,k)\in A} f_{sk} + \sum_{i\in S, i\neq s} \sum_{k:(i,k)\in A} f_{ik} \\
&= \sum_{i\in S} \sum_{k:(i,k)\in A} f_{ik} \\
&= \sum_{i\in S} \sum_{(i,j)\in A, i\in S, j\notin S} f_{ij} \\
&= \sum_{i\in S} \sum_{(i,j)\in A, i\in S, j\notin S} u_{ij} \overset{def}{=} u(\delta^+(S)).
\end{aligned}$$

The first equality follows by flow conservation ($\sum_{k:(ik)} f_{ik} = 0$). We get the third equality by noting that $\forall (i,j) \in A$ s.t $i, j \in S$ $\exists (j,i)$ with $f_{ij} = -f_{ji}$ (flow property (ii)). So, such terms cancel out in the summation and we are left with summing flows over $(i,j) \in A, i \in S, j \notin S$. The last equality is based on the earlier observation that the flow on each arc in the cut is at its capacity.

(iii) ⇒ (i) Since we showed last time that for any s-t cut S, $|f| \leq u(\delta^+(S))$, the fact that $|f| = u(\delta^+(S))$ implies the flow is maximum.

□

We note that the number of s-t cuts is very large ($= 2^{|V|-2}$). So, it is not a good idea to find the max-flow value by calculating all possible cut capacities. The theorem above motivates the following algorithm for finding the max-flow.

Augmenting path algorithm

```
$f \leftarrow 0$
while $\exists$ an augmenting path $P$ in $G_f$
    Push flow along $P$
    Update $f$
```

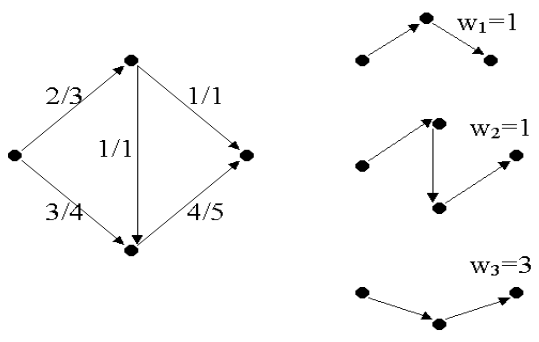


Figure 2.2: Example of Lemma 2.4.

The correctness of the algorithm immediately follows from Theorem 2.3. Under the assumption that all capacities are integer, we push an integral amount of flow at each step of the algorithm. This yields the following result:

Integrality property: If all capacities u_{ij} are integers, there is a max-flow f such that all f_{ij} are integers.

Although the above algorithm correctly finds the max-flow, it is not a polynomial-time algorithm. Later, we will look at ways to obtain a polynomial-time algorithm.

Note that from here on out we will denote $|A|$, the number of arcs in the graph, by m, and $|V|$, the number of vertices in the graph by n.

In order to give our first polynomial-time algorithm for the maximum flow problem, we first need some lemmas. Here's one.

Lemma 2.4 Given an s-t flow f, $\exists$ a set $\mathcal{P}$ of s-t paths and a set $\mathcal{C}$ of cycles, with weights $w : \mathcal{P} \cup \mathcal{C} \rightarrow \Re^+$ s.t.

(i) $f_{ij} = \sum_{P \in \mathcal{P} \cup \mathcal{C} : (i,j) \in P} w(P) \qquad \forall (i,j) \in A,\ f_{ij} > 0.$

(ii) $|f| = \sum_{P \in \mathcal{P}} w(P)$.

(iii) $|\mathcal{P}| + |\mathcal{C}| \leq m$.

In words, the lemma states that any flow can be decomposed into weighted s-t paths and cycles so that the flow on an arc is the sum of weights on paths and cycles using the arc, and the value of the s-t flow is the sum of weights along all s-t paths. The example in Figure 2.2 further illustrates this.

Proof: By induction on the number of $(i,j) \in A$ s.t $f_{ij} > 0$.

Base case: $f_{ij} = 0 \forall (i,j) \in A$. Then $\mathcal{P}, \mathcal{C} = \emptyset$ and the lemma is trivially true.

Inductive step: Consider an arc (i, j) with $f_{ij} > 0$. If $j \neq t$, by flow conservation at node j, there is a k s.t. $f_{jk} > 0$. Similarly, if $i \neq s$, there is an h s.t. $f_{hi} > 0$. Proceeding in this manner, we either find a cycle or an s-t path. Denote it by P. Let $w(P) = \min_{(i,j)\in P} f_{ij}$. Update the flow:

$$f'_{ij} \longleftarrow \begin{cases} f_{ij} - w(P) & \forall (i,j) \in P \\ f_{ij} + w(P) & \forall (j,i) \in P \\ f_{ij} & \text{o.w.} \end{cases}$$

Now, the number of paths/cycles with positive weight increases by 1, while the number of arcs with positive flow decreases by at least 1 and the proof follows. □

ORIE 633 Network Flows	January 30, 2004
Lecture 3	
Lecturer: David P. Williamson	*Scribe: Stefan Wild*

3.1 Applications of the maximum flow problem

3.1.1 Carpool fairness

Description: n people are sharing a carpool for m days. Each announces their schedule in advance.

Ex.- (4 People, 5 Days)

Person	**Days:** 1	2	3	4	5
1	X	X	X		
2	X		X		
3	X	X	X	X	X
4		X	X	X	X

Problem: Every day someone has to drive... We want to allocate driving responsibilities 'fairly.'

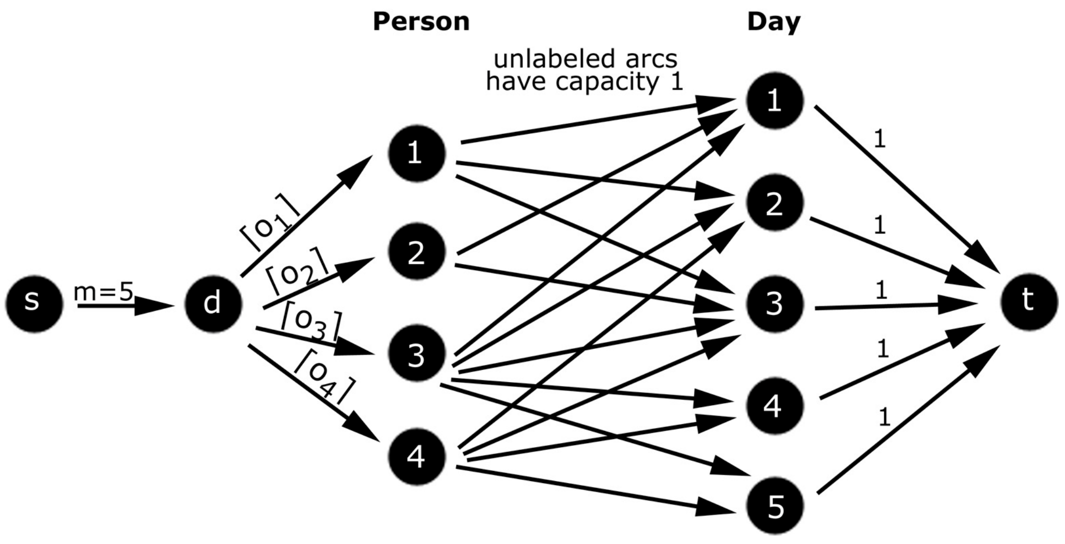
A possible approach/objective is to split the responsibilities equally among the people using the car on a given day. Thus on a day with k people using the carpool, each driver is responsible for a share of $\frac{1}{k}$.

Ex.- (Responsibilities)

Person	**Days:** 1	2	3	4	5
1	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{4}$		
2	$\frac{1}{3}$		$\frac{1}{4}$		
3	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{2}$
4		$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{2}$
$\sum$	1	1	1	1	1

We use these shares to calculate a driving obligation, o_i for person i. In the example, we have $o_1 = \frac{1}{3} + \frac{1}{3} + \frac{1}{4} = \frac{11}{12}$. We can then require that person i drives no more than $\lceil o_i \rceil$ times every m days. To see if this can even be done, we formulate the problem as a network in Figure 3.1.1.

Claim 3.1 If flow of value $|f| = 5$ exists, then a fair driving schedule exists.



Proof: Observe that because the flow is integer:

- All arcs from the sink t have flow 1 which implies that exactly one arc from some person to each day has flow value 1. That person is assigned to drive on that day.
- No one will have to drive more than their obligation (by flow conservation at the person's node and the capacity on their incoming arcs).

□

Claim 3.2 Such a flow always exists and a fair driving allocation always exists.

Proof: We can give a fractional flow of value using the o_i's. Therefore, because the capacities are all integers, there exists an integer flow of value 5. □

3.1.2 Baseball elimination

Ex.- (4 team division)

Team	**Wins**	**Remaining Games**	**Games Against**			
			NYY	Bos	Tor	Bal
NY Yankees	93	8	-	1	6	1
Boston Red Sox	89	4	1	-	0	3
Toronto Blue Jays	88	7	6	0	-	1
Baltimore Orioles	86	5	1	3	1	-

Definition 3.1 A team *wins their division* if it wins more games than the other teams in the division.

Definition 3.2 A team is *eliminated* if they can't finish first given any outcome of the remaining games.

Ex.- Baltimore is clearly eliminated because they will end the season with at most 91 wins while the Yankees already have 93 wins.

Claim 3.3 Boston is also eliminated.

Proof: Boston can still win 93 games but: either the Yankees win one more game and have 94 wins, or Toronto wins all 6 of their games versus the Yankees giving them 94 wins. □

Letting T denote the set of teams in the division, we adopt the following notation for each $i \in T$:

w_i = number of wins for team i

g_i = number of games left to play for team i

g_{ij} = number of games left for team i to play team j.

For subsets $R \subseteq T$ and $S \subseteq T$, we also define:

$$\begin{aligned} w(R) &= \sum_{i \in R} w_i \\ g(R,S) &= \sum_{i \in R} \sum_{j \in S} g_{ij} \\ a(R) &= \frac{w(R) + g(R,R)}{|R|}. \end{aligned}$$

Claim 3.4 Some team $i \in R$ wins at least $a(R)$ games.

Proof: $w(R)$ is the number of wins of the teams in R and $g(R,R)$ represents the number of games in which some team in R must win. Therefore, the average number of wins by teams in R is $a(R)$, which some team surely obtains. □

Corollary 3.5 If $i \in T$, $R \subseteq T - \{i\}$, $a(R) > w_i + g_i$, then team i is eliminated.

Ex.- Let $R = \{\text{Yankees, Toronto}\}$ and $i = \text{Boston}$. Then $a(R) = \frac{(93+88)+6}{2} = 93.5 > 93$. So Boston is eliminated.

Now let x_{ij} be the number of times team i defeats team j (in the remaining games). Then team k is <u>not</u> eliminated if:

$$\begin{aligned}
&\exists\, x_{ij} \\
\text{such that:}\quad & \\
& x_{ij} + x_{ji} = g_{ij} && \forall i, j \in T \\
& w_k + \sum_{j \in T} x_{kj} \geq w_i + \sum_{j \in T} x_{ij} && \forall i \in T \\
& x_{ij} \geq 0, \quad x_{ij} \text{ integer}
\end{aligned}$$

If such x_{ij} exist, then there exist x'_{ij} such that team k wins all its remaining games. Therefore we only need to check that:

$$w_k + g_k \geq w_i + \sum_{j \in T - \{k\}} x_{ij} \qquad \forall i \in T - \{k\}.$$

ORIE 633 Network Flows February 2, 2004

Lecture 4

Lecturer: David P. Williamson *Scribe: Anke van Zuylen*

4.1 Applications of the maximum flow problem

4.1.1 Baseball elimination (cont.)

Recall the problem from last time. We want to decide, based on the outcomes of the games played so far and the games that are still to be played, whether or not a specific team has been eliminated. Team i is eliminated if no possible outcome of the remaining games results in it winning the most games.

Recall the notation we set up last time.

$$\begin{array}{llll} T & & := & \text{teams} \\ i \in T: & w_i & := & \text{\# of wins} \\ & g_i & := & \text{\# of games team } i \text{ has left to play} \\ & g_{ij} & := & \text{\# of games team } i \text{ has left to play against team } j \end{array}$$

Let x_{ij} denote the number of times team i defeats team j. Then, team k is not eliminated if there exist x_{ij} such that

$$\begin{array}{ll} x_{ij} + x_{ji} = g_{ij} & \forall i, j \in T \\ w_k + g_k \geq w_i + \sum_{j \in T - \{k\}} x_{ij} & \forall i \in T - \{k\} \\ x_{ij} \geq 0, x_{ij} \text{ integer} & \forall i, j \in T \end{array}$$

Recall that we also defined the following notation.

$$\begin{aligned} w(R) &= \sum_{i \in R} w_i \\ g(R) &= \sum_{i,j \in R, i<j} g_{ij} \\ a(R) &= \frac{w(R) + g(R)}{|R|} \end{aligned}$$

Remark by DPW: $g(R)$ is what was intended by $g(R,R)$ as used in class. As Anke has pointed out, in the original notation $g(R,R)$ counted every game played twice.

As was shown in the previous lecture, $a(R)$ is a lower bound on the average number of games won by teams in R. Therefore some team in R will win at least $a(R)$ games, and the lemma below follows.

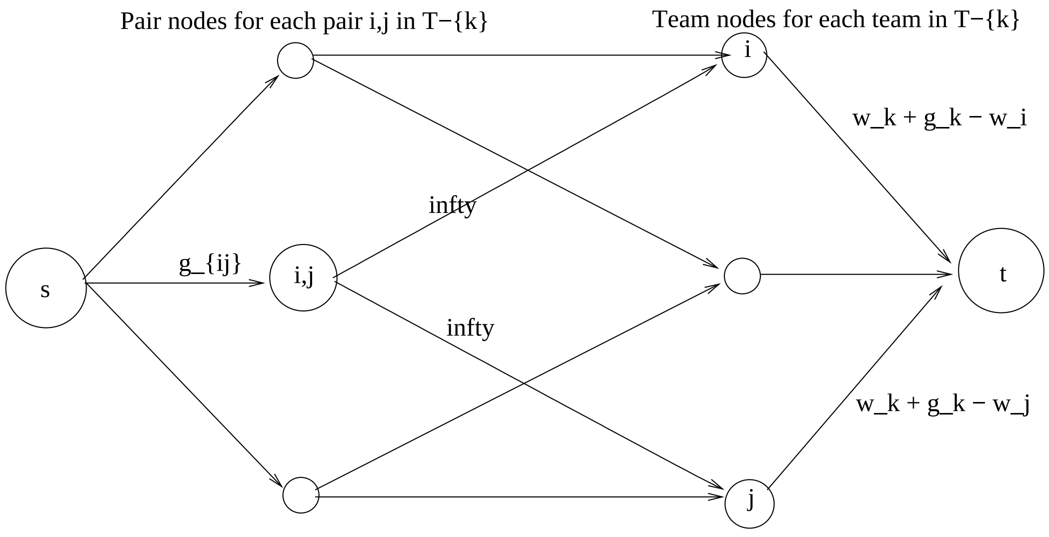

Pair nodes for each pair i,j in T−{k}
Team nodes for each team in T−{k}
i
w_k + g_k − w_i
infty
g_{ij}
s
i,j
t
infty
w_k + g_k − w_j
j

Figure 4.1: Flow instance for deciding if team k is not eliminated.

Lemma 4.1 If there exists $R \subseteq T - \{k\}$ such that $a(R) > w_k + g_k$, then team k is eliminated.

We can use the maximum flow instance shown in Figure 4.1to decide if team k has *not* been eliminated. Note that we can assume that the capacities on the arcs going from the team nodes to the sink t are non-negative, since if $w_k + g_k - w_j < 0$, then $w_j > g_k + w_k$, and we know that team k is eliminated.

Lemma 4.2 If a flow of value $g(T - \{k\}) = \sum_{i,j \in T-\{k\}, i<j} g_{ij}$ exists, then team k is not eliminated.

Proof: If a flow of value $g(T - \{k\}) = \sum_{i,j \in T-\{k\}, i<j} g_{ij}$ exists, then the arcs from s to the pair nodes are at capacity. Let x_{ij} denote the flow from pair node $\{i, j\}$ to team node i. Then $x_{ij} + x_{ji} = g_{ij}$ by flow conservation at the pair node $\{i, j\}$.

By the integrality property of flow, we know that the x_{ij} are integer.

Flow conservation and capacity constraints for team node i give:

$$\sum_{j \in T-\{k\}} x_{ij} \leq w_k + g_k - w_i \Rightarrow w_k + g_k \geq w_i + \sum_{j \in T-\{k\}} x_{ij}.$$

So x_{ij} satisfies the conditions given above, and k is not eliminated. □

Now we can show the opposite direction; if a flow of this value does not exist, then we can prove that the team is eliminated.

Lemma 4.3 If a flow of value $g(T - \{k\})$ does not exist, then team k is eliminated.

Proof: Let S be a minimum $s-t$ cut, and let R be the set of team nodes in S. We can give the following expression for the capacity of S:

$$\begin{aligned} u(\delta^+(S)) &= \sum_{\text{Pairs } \{i,j\} \notin S} g_{ij} + \sum_{i \in R} (w_k + g_k - w_i) \\ &= \sum_{\text{Pairs } \{i,j\} \notin S} g_{ij} + |R|(w_k + g_k) - w(R). \end{aligned}$$

Since there exists no flow of value $g(T - \{k\})$, we know that

$$g(T - \{k\}) > u(\delta^+(S)) = \sum_{\text{Pairs } \{i,j\} \notin S} g_{ij} + |R|(w_k + g_k) - w(R).$$

Since $g(T - \{k\}) = \sum_{\text{All pairs } \{i,j\}} g_{ij}$, we can rewrite the inequality as

$$\sum_{\text{All pairs } \{i,j\}} g_{ij} - \sum_{\text{Pairs } \{i,j\} \notin S} g_{ij} > |R|(w_k + g_k) - w(R),$$

or as

$$\sum_{\text{Pairs } \{i,j\} \in S} g_{ij} > |R|(w_k + g_k) - w(R).$$

If pair node $\{i, j\}$ is in S, then both team nodes i and j are in R, otherwise the cut has infinite capacity. So the sum of g_{ij} for pair nodes $\{i, j\}$ in S cannot be more than $g(R)$; in other words,

$$\sum_{\text{Pairs } \{i,j\} \in S} g_{ij} \leq g(R).$$

Thus we have that

$$g(R) > |R|(w_k + g_k) - w(R),$$

or, rearranging terms once again,

$$\frac{w(R) + g(R)}{|R|} > w_k + g_k.$$

By Lemma 4.1, team k is eliminated. □

In the problem set, the class is asked to show the following.

Lemma 4.4 If team k is eliminated, then for any team ℓ such that

$$w_k + g_k \geq w_\ell + g_\ell$$

team ℓ is also eliminated.

Proof: See problem set 1, problem 2(a). □

This will lead to the following corollary.

Corollary 4.5 $O(\log |T|)$ flow computations determine all eliminated teams.

Proof: See problem set 1, problem 2(b). □

4.2 Preliminaries for a polynomial-time algorithm for the maximum flow problem

We've now shown some applications of the maximum flow problem. Let's turn back now to considering polynomial-time algorithms for computing a maximum flow. We'll need the following lemma for our first algorithm.

Lemma 4.6 Let f be a flow, f^* be a maximum flow in G. Then the maximum flow in the residual graph G_f has value $|f^*| - |f|$.

Proof: Given a flow f' in G_f, let

$$\tilde{f}_{ij} = f_{ij} + f'_{ij} \ \forall (i,j) \in A.$$

Then $\tilde{f}$ is a flow on G, and $|\tilde{f}| = |f| + |f'| \leq |f^*| \Rightarrow |f'| \leq |f^*| - |f|$. Thus the value of any flow in G_f is bounded above by $|f^*| - |f|$.

Also define

$$\hat{f}_{ij} = f^*_{ij} - f_{ij} \ \forall (i,j) \in A.$$

Then $\hat{f}$ is a flow on G_f, since $\hat{f}_{ij} = f^*_{ij} - f_{ij} \leq u_{ij} - f_{ij} \equiv u^f_{ij}$, and $\hat{f}$ is a maximum flow since $|\hat{f}| = |f^*| - |f|$. $\square$

ORIE 633 Network Flows February 4, 2004

Lecture 5

Lecturer: David P. Williamson *Scribe: Yun Shi*

5.1 Polynomial-time algorithms for the maximum flow problem

5.1.1 Augmenting path algorithms

We now turn to giving polynomial-time algorithms for the maximum flow problem. Recall the augmenting path algorithm:

Augmenting path

```
$f \leftarrow 0$
While $\exists$ $s$-$t$ path in $G_f$
    Pick some augmenting path $P$
    Augment flow on $P$
    Update $f$.
```

As we will see in the problem set, this does not necessarily lead to a polynomial-time algorithm because it is possible that there will be too little progress made for each augmentation. However, if one picks a path to make substantial progress in each augmentation, then we can give a polynomial-time algorithm. One natural choice is to pick P with largest possible capacity; i.e. $\max_P \min_{(i,j)\in P}\{u^f_{ij}\}$. Then the algorithm above becomes:

Maximum capacity augmenting path

```
$f \leftarrow 0$
While $\exists$ $s$-$t$ path in $G_f$
    Pick augmenting path $P$ that attains $\max_P \min_{(i,j)\in P}\{u^f_{ij}\}$
    Augment flow on $P$
    Update $f$.
```

To analyze the algorithm, we will need to recall two lemmas from earlier classes:

Lemma 5.1 Any flow can be decomposed into at most m s-t paths.

Lemma 5.2 The value of the max flow in the residual graph G_f is $|f^*| - |f|$, where f^* is a max flow in G.

These two lemmas indicate that some augmenting path P will have capacity at least $\frac{|f^*|-|f|}{m}$. Let's consider $2m$ iterations of loop in the above algorithm; either

(i) all $2m$ iterations augment flow value by $\geq \frac{|f^*|-|f|}{2m}$. Or

(ii) at least one iteration augments flow by $< \frac{|f^*|-|f|}{2m}$.

If (i) happens, we are done, since we have a flow of value at least $|f^*|$. If (ii) happens, then the capacity of the maximum capacity augmenting path has dropped by factor of 2. The upper bound of the capacity of P is $U \equiv \max_{(i,j)\in A} u_{ij}$. The lower bound of the capacity of P is 1. Therefore, there can be at most $O(\log U)$ decreases of the capacity of the maximum capacity augmenting path by factor of 2. Since every $2m$ iterations either the algorithm terminates with a maximum flow or the capacity drops by a factor of 2, there are at most $O(m \log U)$ iterations of the main loop overall. This gives a polynomial-time algorithm.

To get the exact running time, we would have to determine the time needed to find the maximum capacity augmenting path. Rather than get into this, we will consider a variation of the algorithm above in which we only need to find a path in a network. The idea of the this algorithm is to look for paths in which each edge is 'big'. If such a path exists, then we can increase the flow by a significant amount. If there is no such path, then we will show that we must be closer to the maximum flow value. We first need the following definition.

Definition 5.1 A δ-capacity augmenting path P is an augmenting path s.t. $\forall (i,j) \in P$, $u^f_{ij} \geq \delta$.

The algorithm is as follows:

Capacity scaling

```
$f \leftarrow 0$,
$\delta = 2^{\lfloor \log_2 U \rfloor}$
While $\exists$ $s$-$t$ path in $G_f$
    If $\exists$ $\delta$-capacity augmenting path $P$
        Augment flow on $P$, update $f$.
    Else
        $\delta \leftarrow \delta/2$
```

Theorem 5.3 The capacity scaling algorithm runs in $O(m^2 \log U)$ time.

Proof: Clearly if a δ-augmenting path exists, we can increase the value of the flow by at least δ. Now we need to see why the non-existence of δ-augmenting paths is helpful. Suppose there does not exist a δ-capacity path in G_f. Let $\delta' \leftarrow \delta/2$. We know the max flow in G_f can be decomposed into at most m paths. The non-existence of δ-augmenting paths implies that each such path has capacity $< \delta$. Thus the flow in G_f has value $< m\delta = 2m\delta'$.

Thus at the beginning of the while loop in the algorithm we can find at most $2m$ δ-augmenting paths until either we have found a maximum flow or the value of δ is halved.

We know δ can be halved at most $O(\log U)$ times, which implies that there are at most $O(m \log U)$ augmentations overall. Each augmentation requires finding a path in the graph G_f in which only edges with capacity at least δ are retained. Thus finding a δ-augmenting path takes at most $O(m)$ time. Therefore capacity scaling is an $O(m^2 \log U)$ time algorithm. $\square$

If the graph is dense ($m = O(n^2)$), this is a $O(n^4 \log U)$ algorithm, which is not very good. We now turn to an algorithm for finding a maximum flow that takes $O(n^3)$ time, and with fancy data structures takes $O(mn \log \frac{n^2}{m})$ time. The best known running time for finding a maximum flow is $O(\min(m^{1/2}, n^{2/3}) m \log \frac{n^2}{m} \log U)$ (Goldberg, Rao '98).

5.1.2 The push-relabel algorithm

So far we have considering augmenting path algorithms. These algorithms are primal feasible, because capacity constraints are obeyed and flow conservation constraints are obeyed. We maintain a feasible flow and work towards finding a maximum flow. But the next algorithm we will consider, Push-Relabel, is primal infeasible, because it does not obey flow conservation constraints. Here we will maintain a flow that has value at least that of the maximum, and work towards finding a feasible flow. The algorithm will maintain a *preflow.*

Definition 5.2 A preflow is a function $f : A \to \Re$ that obeys capacity constraints, antisymmetry constraints (i.e. $f_{ij} = -f_{ji}$) and

$$\sum_{j:(j,i)\in A} f_{ji} \geq 0$$

for all $i \in V - \{s, t\}$.

That is, in a preflow, instead of flow in equalling flow out for every vertex other than the source and the sink, we have that total flow in is at least total flow out. We define the *excess* to be the difference between the flow in and flow out.

Definition 5.3 We define the *excess* at node i to be $e_i \equiv \sum_{j:(j,i)\in A} f_{ji}$.

If every node (aside from the source and sink) have zero excess, then the preflow is a flow. Given a preflow, we try to reach a feasible flow by pushing excess e_i to sink t and the remainder to source s along shortest paths.

Maintaining shortest path lengths is expensive, so instead we maintain a distance labelling d_i which gives us estimates on the shortest path to the sink.

Definition 5.4 A *distance labelling* is a set of d_i for all $i \in V$ such that:

- d_i is a non-negative integer;

- $d_t = 0$
- $d_s = n$
- $d_i \leq d_j + 1 \quad \forall\, (i, j) \in A_f$

The intuition is that $d_i < n$ gives a lower bound on distance to t, and $d_i \geq n$ gives a lower bound on distance to s.

Claim 5.4 d_i is a lower bound on the distance from i to t.

To see this, consider the shortest path P from i to t. Any arc (i, j) on this path has the relation $d_i \leq d_j + 1$. Thus, $d_i \leq |P|$, and is the lower bound on distance of i to t.

ORIE 633 Network Flows February 6, 2004

Lecture 6

Lecturer: David P. Williamson *Scribe: Yun Shi*

6.1 Polynomial-time algorithms for the maximum flow problem

6.1.1 The push/relabel algorithm (cont.)

Recall from the previous lecture that the push/relabel algorithm will maintain both a preflow and a distance labelling, as defined below.

Definition 6.1 A preflow is a function $f : A \to \Re$ that obeys capacity constraints, antisymmetry constraints (i.e. $f_{ij} = -f_{ji}$) and

$$\sum_{j:(j,i)\in A} f_{ji} \geq 0$$

for all $i \in V - \{s, t\}$.

Definition 6.2 A *distance labelling* is a set of d_i for all $i \in V$ such that:

- d_i is a non-negative integer;
- $d_t = 0$
- $d_s = n$
- $d_i \leq d_j + 1 \quad \forall\ (i, j) \in A_f$

We also need the concept of the excess at a node, which tells us by how much the flow conservation constraint is violated at that node.

Definition 6.3 We define the *excess* at node i to be $e_i \equiv \sum_{j:(j,i)\in A} f_{ji}$.

Recall that in the previous lecture we showed that d_i is a lower bound on the distance from i to the sink t (the source s with $d_s = n$ is an anomaly that we will come back to later). If we want to push flow along shortest paths, then an edge (i, j) is in the shortest path if $d_i = d_j + 1$. So we will only modify flow on edges where this condition holds. What if we have excess at a node i, and the condition does not hold for any edge $(i, j) \in A_f$? Then our distance estimates for i must not be correct, since $d_i \leq d_j$ for all $(i, j) \in A_f$. So we will update the label of i to maintain a distance labelling by setting $d_i \leftarrow \min(d_j + 1, (i, j) \in A_f)$.

After one more definition, we can give the push/relabel algorithm.

Definition 6.4 If $e_i > 0$ for $i \in V - \{s, t\}$, call i active.

Push/Relabel(Goldberg, Tarjan '88)

```
$f \leftarrow 0$, $e \leftarrow 0$
$f_{sj} \leftarrow u_{sj}$;   $f_{js} \leftarrow -f_{sj}$;   $e_j = u_{sj}$
$d_s \leftarrow n$;   $d_t \leftarrow 0$;   $d_i \leftarrow 0 \, \forall \, i \in V - \{s, t\}$
While $\exists$ active $i$
    If $\exists j$, s.t. $u^f_{ij} > 0$ and $d_i = d_j + 1$
        Push $\delta \leftarrow \min(e_i, u^f_{ij})$
        $f_{ij} \leftarrow f_{ij} + \delta$;   $f_{ji} \leftarrow f_{ji} - \delta$
        $e_i \leftarrow e_i - \delta$;   $e_j \leftarrow e_j + \delta$
    Else Relabel $d_i \leftarrow \min(d_j + 1, (i, j) \in A_f)$.
```

Let's first worry about whether the algorithm is correct, and then turn to determining the running time.

Lemma 6.1 The algorithm maintains a valid distance labeling d.

Proof: By induction on algorithm.

Base case: This is trivial for $i \neq s$. For s, there is no condition on d_s, because there is no edge out of s in A_f.

Inductive step: Note that relabelling does not invalidate it the distance labels. What about pushes? If we push on arc (i, j), then two things might happen to cause the distance labelling to be invalid. First, (j, i) might enter A_f. In that case, $d_j = d_i - 1 \leq d_i + 1$, so the distance labelling is valid. Second, (i, j) might be deleted from A_f. This is fine since then there is one less condition to worry about. □

Now we show that if the algorithm terminates, it will find a maximum flow.

Theorem 6.2 If algorithm terminates and all d_i are finite, then f is a maximum flow.

Proof: If algorithm terminates, then f is a flow, since there will not be any excess at any node other than the source and sink. Suppose f is not a max flow. Then there exists an augmenting path P in G_f. By the properties of a distance labelling, this implies that $d_s \leq n - 1$, which contradicts $d_s = n$. □

Now we prove the following lemma, which will be useful in showing that the distance labels stay finite.

Lemma 6.3 If f is a preflow and i is active, then s is reachable form i in G_f.

Proof: Let S be vertices reachable from i in G_f. Suppose $s \notin S$.
Clearly, for $j \in S, k \notin S, \quad f_{kj} \leq 0$, because (j,k) reaches its capacity.
Thus,

$$\begin{aligned} \sum_{j\in S} e_j &= \sum_{j\in S}\sum_{k:(k,j)\in A_f} f_{kj} = \sum_{\substack{j\in S, k\notin S \\ (k,j)\in A_f}} f_{kj} \leq 0 \\ \Rightarrow e_j &= 0 \quad \forall j \in S \\ \Rightarrow e_i &= 0 \quad \Rightarrow i \text{ is not active,} \end{aligned}$$

which causes a contradiction. □

Lemma 6.4 At any point in algorithm, $d_i \leq 2n-1 \quad \forall i \in V$

Proof: d_s, d_t never change. d_i increases only when i is active.
i is active implies there exists a path P in G_f from i to s by last lemma. The path in G_f has the length of at most n-1. So $d_i \leq d_s + n - 1 = 2n-1$. □

Lemma 6.5 At most $2n^2$ executions of relabel.

Proof: $0 \leq d_i \leq 2n-1$, d_i is integer, d_i never decreases, and relabel increases it by at least 1. So each vertice need at most $2n-1$ executions and there are n vertices. Thus there are at most $n(2n-1) \leq 2n^2$ executions of relabel. □

In the algorithm, there are two types of pushes:

(i) push is saturating if $\delta = u_{ij}^f$

(ii) push is nonsaturating if $\delta < u_{ij}^f$, i.e $\delta = e_i$

Lemma 6.6 At most mn saturating pushes.

Proof: Pick an edge $(i,j) \in A$, need $d_i = d_j + 1$ to push from i to j; to do it again, need to push back from j to i and $d_j = d_i + 1 \quad \Rightarrow$ need d_j to increase at least by 2
$\Rightarrow$ at most $n-1$ saturating pushes from i to j by lemma 4.
There are at most m edges $\Rightarrow$ at most $m(n-1)$ saturating pushes. □

Lemma 6.7 At most $4n^2m$ nonsaturating pushes.

Proof: Let

$$\Phi \equiv \sum_{\text{active } i} d_i$$

At the start of algorithm $\Phi = 0$. At the end of algorithm $\Phi = 0$ too, since there is no active vertex then. So Φ must decrease by the amount Φ increases.

What makes Φ increase? Relabel will increase it by at most $2n^2$. One saturating push may create a new active vertex and increase it by at most $2n$. So Φ can increase by at most $2n^2 + 2n(mn)$
What makes Φ decrease? Only nonsaturating push has Φ decrease, because it make i inactive.
$\Rightarrow$ no more than $4n^2m$ nonsaturating pushes. $\square$

Theorem 6.8 Push/Relabel takes $O(n^2m)$ push/relabel operations.

The proof of this theorem follows from the above 3 lemmas.

ORIE 633 Network Flows February 9, 2004

Lecture 7

Lecturer: David P. Williamson *Scribe: Christina Tavoularis*

7.1 Polynomial-time algorithms for the maximum flow problem

7.1.1 The push/relabel algorithm (cont.)

Recall from the previous lecture the Push/Relabel algorithm:

Push/Relabel

```
$f \leftarrow 0$
Saturate edges out of source
$d_s \leftarrow n, d_i \leftarrow 0, \forall i \in V - \{s\}$
While $\exists$ active $i$ $(e_i > 0)$
If $\exists j : u^f_{ij} > 0$ and $d_i = d_j + 1$
    Push $\delta = \min\{e_i, u^f_{ij}\}$ flow on $(i, j)$
else Relabel $d_i \leftarrow \min(d_j + 1 : (i, j) \in A_f)$.
```

Notice that this algorithm is typically able to find the min s-t cut before the max flow, since the excess flow must be pushed back to the sink before the maximum flow is revealed. We claim that to find the min s-t cut only, one can simply change the definition of *active*: Let i be active if $e_i > 0$ and $d_i < n$. When the algorithm terminates, any node with excess will have $d_i \geq n$. By arguments from the previous class, this implies that such a node has no path to the sink. Let S be the set of all nodes that cannot reach the sink. Then it must be the case that all arcs in $\delta^+(S)$ are at capacity. We claim that S must then be a min s-t cut; if the algorithm were to continue to run, it would push all remaining excesses back to the source, and all arcs in $\delta^+(S)$ continue to be at capacity.

Recall the following lemmas, which we proved last time about the execution of the Push/Relabel algorithm.

Lemma 7.1 If f is a preflow and $e_i > 0$, then i can reach s in G_f.

Lemma 7.2 $d_i \leq 2n - 1, \forall i \in V$.

Lemma 7.3 There can be at most $2n^2$ relabels.

Lemma 7.4 There can be at most nm saturating pushes.

Lemma 7.5 There can be at most $4n^2m$ non-saturating pushes.

From these we derived the following theorem.

Theorem 7.6 Push/Relabel takes $O(n^2m)$ push/relabel operations.

It can be shown that this leads to an $O(n^2m)$ time algorithm.

We now turn to improving the running time of Push/Relabel. From the analysis above, it is the bound on the number of non-saturating pushes that determines the running time of the algorithm. In the algorithm above, we did not specify how pushes and relabels should be executed. We now show that if we are a little more careful, we can obtain a better bound on the non-saturating pushes.

FIFO Push/Relabel

```
$f \leftarrow 0$
Saturate edges out of source
$d_s \leftarrow n, d_i \leftarrow 0, \forall i \in V - \{s\}$
Put all active vertices in queue $Q$
While $Q \neq \emptyset$
    Let $i$ be vertex at front of $Q$
    While $e_i > 0$ and $\exists j : u^f_{ij} > 0$ and $d_j = d_i + 1$
        Push$(i, j)$
        If $j$ becomes active
            Add $j$ to end of $Q$
    If $e_i > 0$
        Relabel $i$ and add it to end of $Q$.
```

Lemma 7.7 The number of passes over the queue in FIFO push-relabel is at most $4n^2$.

Corollary 7.8 The number of non-saturating pushes in FIFO push-relabel is $O(n^3)$. Thus, the algorithm runtime is $O(n^3)$.

Proof: This follows since there is at most one non-saturating push per vertex per pass. □

Proof of Lemma 7.7: Use the potential function $\Phi = \max\{d_i : i \text{ active}\}$. Divide the passes into two types: passes in which some distance label increases, and passes in which no distance label changes.

By our previous argument that each $d_i \leq 2n - 1$, the total number of passes in which some distance label increases is at most $2n^2$.

If no distance label changes during the pass, then each vertex has excess moved to lower-labeled vertices, and Φ decreases by at least 1 during the pass.

Let $\Delta\Phi_\ell$ be the change in Φ from the beginning of pass ℓ until the end of pass ℓ. When $\Delta\Phi_\ell > 0$, we know that some distance label has increased by at least $\Delta\Phi_\ell$. Thus $\sum_{\ell:\Delta\Phi_\ell>0} \Delta\Phi_\ell \le 2n^2$.

Because Φ is initially zero, stays non-negative, and is zero at the end of the algorithm, the total number of passes in which Φ decreases cannot be more than $\sum_{\ell:\Delta\Phi_\ell>0} \Delta\Phi_\ell \le 2n^2$; that is, since we know that the total increase in Φ over all passes in which it increases is at most $2n^2$, the total number number of passes in which it decreases is also at most $2n^2$.

Thus the total number of passes can be bounded by $4n^2$: $2n^2$ for the passes in which no distance label changes (and thus Φ decreases), and $2n^2$ for the passes in which some distance label increases. $\square$

By using fancy data structures, Push/Relabel can be implemented even more efficiently.

Theorem 7.9 (Goldberg, Tarjan, 1988) Push/Relabel implemented in $O(nm\log(n^2/m))$ time.

7.2 Polynomial-time algorithms for the global min-cut problem

We now turn to the following problem.

Global Min-cut

- **Input:**
  - directed graph G=(V,A)
  - capacities $u_{ij} \ge 0\ \forall (i,j) \in A$, *integer*
- **Goal:** Find $S \subset V, S \ne \emptyset$ that minimizes $u(\delta^+(S))$

We will make use of the *minimum s-cut* when solving for the global min-cut.

Definition 7.1 The minimum s-cut for a specified $s \in V$ is a cut $S \subset V$ that minimizes $u(\delta^+(S))$ such that $s \in S$.

Lemma 7.10 We can find the global min-cut by running a min s-cut algorithm twice.

Proof: Pick any $v \in V$ and set $s = v$. Obviously if we find a min s-cut, we find the minimum cut among all those such that $s \in S$. We now need to find the minimum cut among all those such that $s \notin S$. To do this, we construct G' from G by reversing all its arcs; that is, for each (i,j) in G with capacity u_{ij} , add arc (j,i) to G' with the same capacity. Now find a minimum s-cut in G'. Note that any cut S in G' with $s \in S$ has the same capacity as the cut $V - S$ in G. Thus the min s-cut in G' has the same value as the minimum cut in G that does not contain s. We take the smaller of the two cuts to obtain the global min-cut. $\square$

Lemma 7.11 We can find min s-cut in $n-1$ max flows.

Proof: Observe that there must exist some $i \notin S$. Thus if we find minimum s-i cuts for all possible $i \neq s$, one of these must also be the minimum s-cut; we just take the s-i cut that has the smallest value. □

Next time we'll see an algorithm that finds a min s-cut in the time needed to run a single Push/Relabel computation.

ORIE 633 Network Flows February 11, 2004

Lecture 8

Lecturer: David P. Williamson *Scribe: Christina Tavoularis*

8.1 Polynomial-time algorithms for the global min-cut problem

8.1.1 The Hao-Orlin algorithm

Recall from the previous lecture the global min-cut problem and the claim:

Global Min-cut

- **Input:**
  - directed graph G=(V,A)
  - capacities $\nu_{ij} \geq 0 \ \forall (i,j) \in A$, *integer*
- **Goal:** Find $S \subset V, S \neq \emptyset$ that minimizes $\nu(\delta^+(S))$

Definition 8.1 Min s-t cut: Input $s, t \in V$. Find $S : s \in S, t \notin S$ that minimizes $u(\delta^+(S))$.

Definition 8.2 Min s-cut: Input $s, t \in V$. Find $S : s \in S$ that minimizes $u(\delta^+(S))$.

We showed last time the following two lemmas.

Lemma 8.1 We can find the min global cut by running a min s-cut algorithm twice.

Lemma 8.2 We can find the min s-cut with $n-1$ max flows.

In fact, we can also find the min s-cut by finding something more exotic, called a minimum X-t cut.

Definition 8.3 The min X-t cut: Input $X \subset V, X \neq \emptyset, t \in V$. Find $S : X \subseteq S, t \notin S$ that minimizes $u(\delta^+(S))$.

Claim 8.3 If we can find the min X-t cut for any X and t, we can find the min s-cut.

Proof: Number the nodes $S \leftarrow 1, 2, 3, \ldots, n$. For $i \leftarrow 2, \ldots, n$ let $X = \{1, \ldots, i-1\}$ and find min X-i cut. We claim that one of these X-i cuts is the min s-cut. To see this, let S be a min s-cut. Let j be the smallest vertex not in S. So $\{1, 2, \ldots, j-1\} \subseteq S$. Therefore, S is a min X-j cut for $X = 1, \ldots, j-1$ and the algorithm will find it. □

We will now show how we can implement the algorithm given in the proof above to find a minimum s-cut. First we need a few definitions.

Definition 8.4 A *distance level* k, D_k, is the set $\{i \in V : d_i = k\}$.

Definition 8.5 Distance level k is *empty* if $D_k = \emptyset$.

Definition 8.6 Distance level k is called a *cut level* if $|D_k| = 1$ and for $i \in D_k, \forall (i,j) \in A_f, d_i < d_j$.

We will now establish why cut levels are useful when finding min cuts.

Lemma 8.4 If the distance level k is a cut level, then for $S = \{i : d_i \geq k\}$, all arcs in $\delta^+(S)$ are saturated.

Proof: Pick any $(i,j) \in \delta^+(S)$. By definition of S, $d_i \geq k, d_j < k$. If $d_i = k$, then by definition of cut level $(i,j) \notin A_f$ which implies that (i,j) is saturated. If $d_i > k$, then $d_i > d_j + 1$, and thus $(i,j) \notin A_f$, which again implies that (i,j) is saturated. □

The intuition is that the min cut can be found when there are no active nodes strictly below the cut level. Here is the implementation of the push/relabel algorithm to find the min s-cut.

Push/relabel min s-cut (Hao and Orlin, 1993)

```
$X \leftarrow \{s\}$; Pick any vertex in $V - X$ as $t$
$d_s \leftarrow n, d_t \leftarrow 0, d_i \leftarrow 0, \forall i \in V - \{s\}$
$cutval \leftarrow \infty$, $cut \leftarrow \emptyset$
While $X \neq V$
    Run Push/Relabel which selects only active nodes $i$ with $d_i < k$
        for lowest cut level $k$ (or $d_i < n-1$ if no cut level)
    Let $k$ be lowest cut level ($n-1$ if no cut level)
        Note that there are no active nodes $i$ with $d_i < k$.
    $S \leftarrow \{i : d_i \geq k\}$
    If $u(\delta^+(S)) < cutval$
        $cutval \leftarrow u(\delta^+(S))$ and $cut \leftarrow S$
    Pick $t' \neq t : d_t \leq d_i, \forall i \in V - X - \{t\}$
    Let $X \leftarrow X \bigcup \{t\}$
    Let $d_t \leftarrow n$ and saturate all arcs out of $t$
    Set $t \leftarrow t'$
Return cutval, cut.
```

Lemma 8.5 The non-empty distance levels k for $k < n$ are consecutive.

Proof: This is clearly true at the beginning of the algorithm. If some distance level D_l with $l < n$ becomes empty, let i be the last node in D_l. We will consider two cases for i to leave D_l.

- Case (1): i is relabelled. This happens if $e_i > 0$ and $d_i < k$ for lowest cut level k. Consequently, $d_i \leq d_j \forall (i,j) \in A_f$. This is a contradiction since $|D_l| = 1$, implying that l is the lowest cut level, not k.
- Case (2): i is sink t and $d_i \leftarrow n$ at the end of the execution of that loop. Then, in the previous iteration i had the minimum distance d_i. Also, since i is a sink, its distance has not increased. Therefore, i is still the minimum d_i and setting d_i to n does not contradict the lemma.

□

The following lemmas will be proved in the next lecture:

Lemma 8.6 If $i \notin X, d_i \leq n - 2$.

Lemma 8.7 Each time through the while loop, S is the min X-t cut.

Lemma 8.8 There are at most $O(n^2)$ relabels.

Lemma 8.9 There are at most $O(nm)$ saturating pushes.

Lemma 8.10 There are at most $O(n^3)$ non-saturating pushes.

The lemmas above go through as before, although we need to modify the definition of Φ slightly for the last one.

Theorem 8.11 (Hao, Orlin 1993) The push/relabel min s-cut finds the min s-cut in $O(n^3)$ time.

ORIE 633 Network Flows February 13, 2004

Lecture 9

Lecturer: David P. Williamson *Scribe: Sumit Kunnumkal*

9.1 Polynomial-time algorithms for the global min-cut problem

9.1.1 The Hao-Orlin algorithm

In this lecture, we will complete the analysis of the algorithm by Hao and Orlin, which can be used to find a global min-cut in directed graphs. Then, we will look at the problem of finding a global min-cut in undirected graphs. Recall from last time:

Definition 9.1 Distance level k, $D_k = \{i \in V : d_i = k\}$. Distance level k is said to be empty if $D_k = \emptyset$. It is said to be a cut-level if $|D_k| = 1$ and for $i \in D_k$ and all $(i,j) \in A_f, d_i \leq d_j$.

Push/Relabel min s-cut (Hao and Orlin, 1993)

```
$X \leftarrow \{s\}$; Pick any vertex in $V - X$ as $t$
$d_s \leftarrow n, d_t \leftarrow 0, d_i \leftarrow 0, \forall i \in V - \{s\}$
$cutval \leftarrow \infty$, $cut \leftarrow \emptyset$
While $X \neq V$
    Run Push/Relabel which selects only active nodes $i$ with $d_i < k$
        for lowest cut level $k$ (or $d_i < n-1$ if no cut level)
    Let $k$ be lowest cut level ($n-1$ if no cut level)
        Note that there are no active nodes $i$ with $d_i < k$.
    $S \leftarrow \{i : d_i \geq k\}$
    If $u(\delta^+(S)) < cutval$
        $cutval \leftarrow u(\delta^+(S))$ and $cut \leftarrow S$
    Pick $t' \neq t : d_t \leq d_i, \forall i \in V - X - \{t\}$
    Let $X \leftarrow X \bigcup \{t\}$
    Let $d_t \leftarrow n$ and saturate all arcs out of $t$
    Set $t \leftarrow t'$
Return cutval, cut.
```

We showed the following lemma last time

Lemma 9.1 The non-empty distance levels k for $k < n$ are consecutive.

We stated the following lemmas last time:

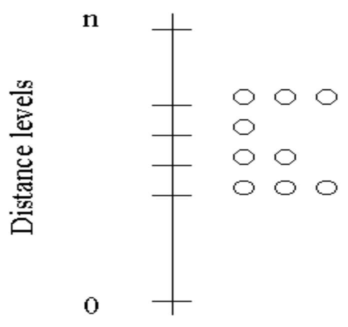


Figure 9.1: Consecutive non-empty distance levels.

Lemma 9.2 If $i \notin X, d_i \leq n-2$.

Lemma 9.3 Each time through the while loop (While $X \neq V$), the cut S that the algorithm finds is a min X-t cut.

Before we prove the two lemmas let us first take a look at their implications.

Lemma 9.4 At most $O(n^2)$ relabels.

Lemma 9.5 At most $O(mn)$ saturating pushes.

Lemma 9.6 At most $O(n^3)$ non-saturating pushes.

Lemma 9.4 is true since each time a node other than the sink is relabelled, its distance label increases by at least 1 and the total increase is bounded by $n-2$ (by Lemma 9.2). The distance label of the sink is set to n at the end of the iteration and it is not relabelled further. Lemma 9.5 holds since between 2 saturating pushes on an arc, the distance labels of its end nodes must have increased by 2. Again, as the distance labels are bounded, the number of saturating pushes is $O(n)$ for any arc and $O(mn)$ overall. Lemma 9.6 can be shown using the FIFO implementation of the push/relabel algorithm and a modified potential function.

Lemma 9.7 Cut returned at the end of the algorithm is a min s-cut.

Proof: Let S^* be a min s-cut with capacity $u(\delta^+(S^*))$. Consider the first iteration of the while loop for which the current sink is not in the min s-cut, i.e., $t \notin S^*$. Then, in this iteration $X \subseteq S^*$. The min X-t cut found in this execution (by Lemma 9.3) can have capacity at most $u(\delta^+(S^*))$. This is because S^* is also an X-t cut. Also since any X-t cut is an s-cut, its capacity is at least $u(\delta^+(S^*))$. The above statements imply that the min X-t cut found at the end of this iteration is a min s-cut. □

Recall that we also showed last time that two executions of a min s-cut algorithm can be used to find a global min cut. Then the above lemmas lead to the following theorem:

Theorem 9.8 (Hao, Orlin '94) The algorithm finds a min s-cut and also a global min-cut in $O(n^3)$ time.

In fact, it can be shown that algorithm can run in $O(mn \log n)$ time. We contrast this running time with the earlier "crude" estimates of $(n-1)$ and $n(n-1)$ max flow computations required to find a min s-cut and global min-cut, respectively.

Now, we return to the proofs of Lemmas 9.2 and 9.3.

Proof of Lemma 9.2: By induction on $|X|$. Let $i \notin X$ have the max distance label. Noting that at each iteration the current sink t has the lowest distance label, Lemma 9.1 implies that the distance levels between d_t and d_i are all non-empty.

Initially, $X = \{s\}$, $d_t = 0$, and since each distance level between d_i and d_t must contain at least one vertex from the remaining $n - |X| - 1$ vertices,

$$d_i \leq d_t + (n - |X| - 1) \leq n - 2$$

Now assume that $d_i \leq d_t + (n - |X| - 1) \leq n - 2$ at the start of an iteration. At the end of this iteration, $|X|$ increases by 1 as the current sink t is added to X. The distance label of the new sink $t^{'}$ increases by at most 1. This is because even if the lowest distance level becomes empty after t has been added to X, there must be a node in the next higher distance level (by the property that the non-empty distance levels are consecutive). Letting $d^{'}$ denote the distance labels in the next iteration and $X^{'} = X \cup \{t\}$,

$$d_i^{'} \leq d_t^{'} + (n - |X^{'}| - 1) \leq d_t + 1 + (n - (|X| + 1) - 1) \leq n - 2$$

The first inequality follows from Lemma 9.1, while the last one comes from the inductive hypothesis. □

Proof of Lemma 9.3: We know that $d_i = n \ \forall i \in X$, $d_i \leq n - 2 \ \forall i \notin X$ and the sink t has the minimum distance label. Now, the way in which S is chosen implies that $X \subseteq S$ while $t \notin S$. Also, since any node with an excess is inside S and all arcs in $\delta^+(S)$ are saturated (proved in last lecture), S is a min X-t cut. □

So, in directed graphs an algorithm for finding a global min-cut is based on a max-flow computation. Next, we look at an algorithm for finding a global min-cut in undirected graphs which has almost nothing to do with flows.

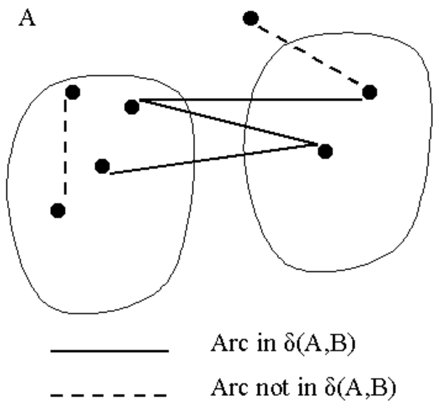


Figure 9.2: Example of $\delta(A, B)$.

9.1.2 Global min-cut in undirected graphs

Global min-cut in an undirected graph

- **Input:**
  - Undirected graph $G = (V, E)$
  - Arc capacities $u_{ij} > 0 \ \forall (i, j) \in E$
- **Goal:** Find $S \subset V, S \neq \emptyset$ that minimizes $u(\delta(S)) = \sum_{(i,j)\in\delta(S)} u_{ij}$.

We define $\delta(S)$ for $S \subset V$ as

Definition 9.2 $\delta(S) = \{(i, j) \in E : i \in S, j \notin S \text{ or } i \notin S, j \in S\}$

We also define $\delta(A, B)$ for two vertex sets $A, B \subset V$, $A \cap B = \emptyset$ as

Definition 9.3 $\delta(A, B) = \{(i, j) \in E : i \in A, j \in B \text{ or } i \in B, j \in A\}$

Let us now consider the following greedy algorithm:

MA (max adjacency) ordering

```
$S \leftarrow \{v_1\}$
For $i \leftarrow 2$ to $n$
    Choose $v_i$ to maximize $u(\delta(S, \{v\})) \ \forall v \in V - S$
    $S \leftarrow S \cup \{v_i\}$
```

Given some arbitrarily chosen vertex v_1, the algorithm returns an ordering of the vertices. In each iteration, the algorithm looks at all vertices not in the set S and picks the one which maximizes the capacity of arcs connecting it to nodes in S. We will prove the following claim in the next lecture, which at first glance looks very surprising.

Claim 9.9 For MA ordering $v_1, \ldots, v_n$, $\{v_n\}$ is a minimum v_{n-1}-v_n cut (or v_n-v_{n-1} cut, the order doesn't matter in an undirected graph).

Remarks

1. We don't know in advance what the nodes v_{n-1} and v_n will be.
2. The minimum v_{n-1}-v_n cut is special in that one side of the cut just consists of a single vertex.
3. The MA ordering algorithm can be used as a subroutine to find a global min-cut. To see this, let S be a global min-cut. Consider 2 cases:

Case 1: $v_n \in S, v_{n-1} \notin S$. Then, since $(S, V\text{-}S)$ is a v_n-v_{n-1} cut, $u(\delta(S)) \geq u(\delta(\{v_n\}))$. Also, since S is a global min-cut, $u(\delta(S)) \leq u(\delta(\{v_n\}))$. So, the given v_n-v_{n-1} cut is a global min-cut and we are done.

Case 2: v_n, v_{n-1} are on the same of side of the global min-cut. Then we "contract" v_n and v_{n-1} to a single vertex and repeat MA ordering on a graph with one fewer vertex.

We end with the following claim:

Claim 9.10 After $n-1$ runs of MA ordering, we will have found a global min-cut.

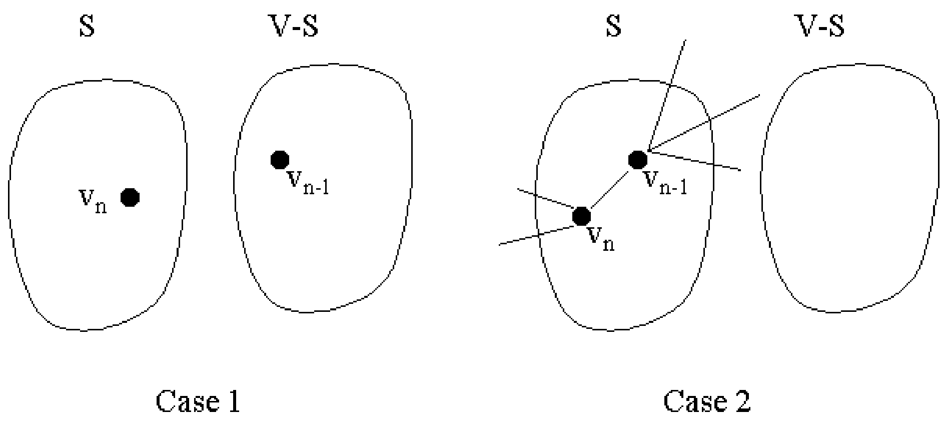


Figure 9.3: Illustration of cases 1 and 2.

ORIE 633 Network Flows February 16, 2004

Lecture 10

Lecturer: David P. Williamson *Scribe: Retsef Levi*

10.1 Polynomial-time algorithms for the global min-cut problem

10.1.1 MA Orderings

Last class, we returned to discuss undirected graphs. Given an undirected graph $G = (V, E)$, let $\delta(A, B) = \{(i, j) \in E : i \in A, j \in B \text{ or } j \in A, i \in B\}$, we have defined the following notion of what we called MA ordering on the nodes of G.

Recall that an MA ordering is any ordering on the nodes that is computed in the following way.

MA (max adjacency) ordering

```
$S \leftarrow \{v_1\}$
For $i \leftarrow 2$ to $n$
    Choose $v_i$ to maximize $u(\delta(S, \{v\}))$ $\forall v \in V - S$
    $S \leftarrow S \cup \{v_i\}$
```

Last time we also presented the following claim about one of the properties of any MA ordering.

Claim 10.1 For MA Ordering $v_1, v_2, \ldots, v_n$, the cut around the node $\{v_n\}$ is a min v_{n-1}-v_n cut.

We then showed how this property can be used to construct an algorithm for computing the global min cut on an undirected graph. The proof of the claim will follow the algorithm.

Finding MinCut using MA ordering

```
$MC \leftarrow \infty, S \leftarrow \emptyset$
While $|V| > 1$
    Compute MA ordering $v_1, v_2, ..., v_n$
    If $u(\delta(v_n)) < MC$
        $MC \leftarrow u(\delta(v_n)), S \leftarrow \{v_n\}$
        Contract $v_{n-1}$ and $v_n$ into a single node
Return $S$.
```

An example of the above algorithm is presented at the end.

To prove the claim made above, we need the following lemma.

Lemma 10.2 Let $\lambda(G,s,t)$ denote value of the min s-t cut in G. Then for any three vertices $p,q,r \in V$, $\lambda(G,p,q) \geq \min(\lambda(G,r,q), \lambda(G,p,r))$.

Proof: Let S be the min p-q cut of the graph and $p \in S$. Now suppose $r \in S$. Then, $\lambda(G,p,q) \geq \lambda(G,r,q)$, since S is also an r-q cut. If $r \notin S$, then $\lambda(G,p,q) \geq \lambda(G,p,r)$ since S is also a p-r cut. In either case, the result holds. □

Proof of Claim 10.1: We know that by the definition of the min cut $\lambda(G,v_{n-1},v_n) \leq u(\delta(v_n))$. We need to show that $\lambda(G,v_{n-1},v_n) \geq u(\delta(v_n))$. We do this through an induction on the number of nodes and edges, $|E|+|V|$.

- The base case, i.e. when either $|E| = 0$ or $|V| = 2$, holds trivially.
- For the inductive case, there are two possibilities
  (i) $(v_{n-1},v_n) \in E$:
  Let $(v_{n-1},v_n) = e$, $G' \leftarrow G - e$, $\delta' \leftarrow \delta$. Now, observe that $v_1, v_2, ..., v_n$ is still an MA ordering of G', and

$$\begin{aligned} u(\delta(v_n)) &= u(\delta'(v_n)) + u_e \\ &= \lambda(G', v_{n-1}, v_n) + u_e \\ &= \lambda(G, v_{n-1}, v_n). \end{aligned}$$

  The second equality is by induction and the final equality is because for each any v_{n-1}-v_n cut in G' has the same value in G (adding in edge e) and vice versa.

  (ii) $(v_{n-1},v_n) \notin E$:
  In this case, we need to apply the inductive hypothesis twice. First, let $G' \leftarrow G - v_{n-1}$. Note that $v_1, v_2, \ldots, v_{n-2}, v_n$ is an MA ordering in G', and by the inductive hypothesis,

$$\begin{aligned} u(\delta(v_n)) &= u(\delta'(v_n)) \\ &= \lambda(G', v_{n-2}, v_n) \\ &\leq \lambda(G, v_{n-2}, v_n). \end{aligned}$$

  The last inequality follows since the cut in G separating v_{n-2} and v_n has no greater value in G'.
  Now, let $G' \leftarrow G - v_n$. Again, $v_1, v_2, \ldots, v_{n-1}$ is an MA ordering in G', and by the construction of the ordering, and the inductive hypothesis,

$$\begin{aligned} u(\delta(v_n)) &\leq u(\delta(v_{n-1})) \\ &= u(\delta'(v_{n-1})) \\ &= \lambda(G', v_{n-2}, v_{n-1}) \\ &\leq \lambda(G, v_{n-2}, v_{n-1}). \end{aligned}$$

Again, the last inequality follows since the cut in G separating v_{n-2} and v_{n-1} has no greater value in G' (we could just delete edges touching v_{n-1}).

Now using Lemma 10.2,

$$\lambda(G, v_{n-1}, v_n) \geq \min(\lambda(G, v_{n-2}, v_{n-1}), \lambda(G, v_{n-2}, v_n)) \geq u(\delta(v_n)).$$

Therefore by the principle of mathematical induction, $\lambda(G, v_{n-1}, v_n) \geq u(\delta(v_n))$ holds for any number of vertices and edges. This proves the claim.

$\square$

A note: Using Fibonacci heaps, an MA ordering of a graph can be computed in $O(m + n \log n)$ time. Thus, the algorithm to compute a global min-cut in an undirected graph using MA orderings presented above has a time complexity of $O(n(m+n\log n))$. The fastest known algorithm for finding a global min-cut in an undirected graph runs in $O(m \log^3 n)$ randomized time.

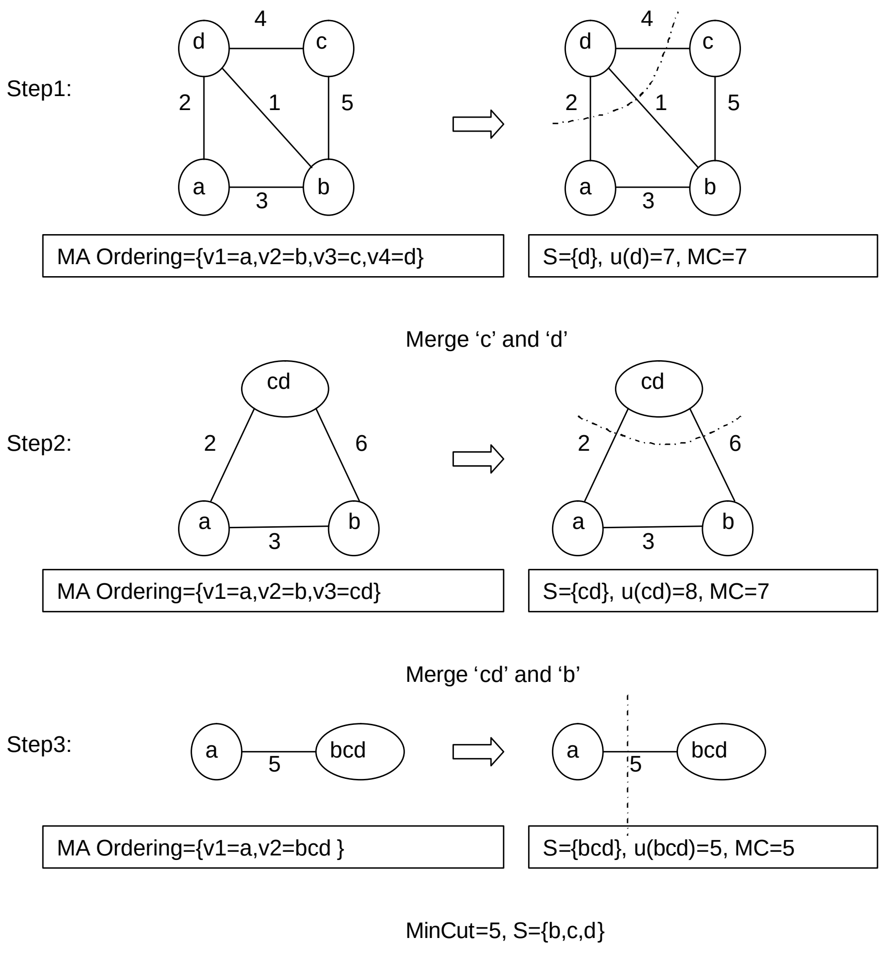


Figure 10.1: An example of the min-cut algorithm via MA orderings.

ORIE 633 Network Flows February 18, 2004

Lecture 11

Lecturer: David P. Williamson *Scribe: Christina Peraki*

11.1 More polynomial-time algorithms for the maximum flow problem

11.1.1 Blocking flows

Starting today, we'll work toward yet one more algorithm for computing a maximum flow; it's based on the concept of a *blocking flow.* Although we've covered several different types of flow algorithms, the algorithm we will eventually present is the theoretically fastest algorithm out there, and it seems negligent to leave it out.

Definition 11.1 A flow f in G is **blocking** if every s-t path in G, the original graph, has some arc saturated.

Every maximum flow is obviously also a blocking flow. Is every blocking flow a maximum flow? No, we've seen a counterexample in Lecture 1. However, it is useful in order to compute maximum flows. We give an algorithm below.

Dinic's Algorithm (Dinic 1970)

```
$f \leftarrow 0$
while $\exists s$-$t$ path in $G_f$
    Compute distances $d_i$ to sink $t$ in $G_f$ $\forall i \in V$
    Find blocking flow $\tilde{f}$ in graph $\tilde{G}$ with arcs $\tilde{A} = \{(i,j) \in A_f : d_i = d_j + 1\}$
        capacity $u^f_{ij}$
    $f \leftarrow f + \tilde{f}$.
```

In the problem set, we considered an augmenting path algorithm in which we sent flow down the shortest path in the residual graph each time. In Dinic's algorithm we effectively saturate all the shortest paths at the same time.

Definition 11.2 An arc in $\tilde{A} = \{(i,j) \in A_f : d_i = d_j + 1\}$ is called **admissible**.

The easier part of the following theorem is a bonus problem on the current problem set.

Theorem 11.1 Blocking flows in acyclic graphs can be found in $O(mn)$ time, but if fancy data structures are used, then they can be found in $O(m \log n)$ time.

Note that the efficient algorithms for computing blocking flows are for *acyclic* graphs. The set of admissible arcs is acyclic, otherwise we would have an inconsistency of the $d_i = d_j + 1$ equation as is shown in Figure 11.1.

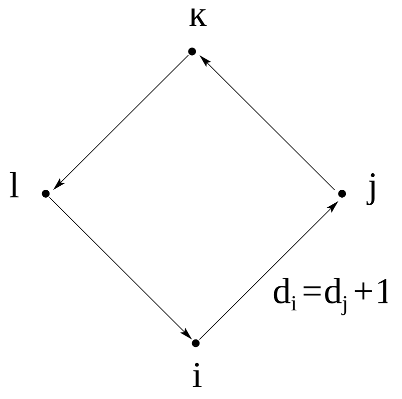


Figure 11.1: Inconsistency of distances equation if the graph has a cycle.

The following lemma is the key to proving a bound on the running time of Dinic's algorithm.

Lemma 11.2 The distance to the sink d_s strictly increases in each iteration of the algorithm.

Clearly this implies that the algorithm takes at most n iterations. Given the two blocking flow algorithms mentioned above, we get the following results.

Theorem 11.3 (Dinic 1970): Maximum flow via blocking flows can be computed in $O(mn^2)$ time.
(Sleator, Tarjan 1980): Maximum flow via blocking flows can be computed in $O(mn \log n)$ time.

Now we turn to the proof of the lemma.

Proof of Lemma 11.2: Let d_i be distance labels in one iteration, d'_i in the next. Let f be the flow in one iteration, f' in the next one.

To begin, we claim that the d_i is a valid distance labelling for the flow in the next iteration; that is, $d_i \le d_j + 1$ for all $(i,j) \in A_{f'}$. To see this, first consider how an arc (i,j) can be in the residual graph of the flow in the next iteration. It could be because it was in the residual graph of the previous iteration; that is $(i,j) \in A_{f'}$, because $(i,j) \in A_f$, in which case the statement holds. Or it could be that the arc is in the residual graph of the next iteration because we pushed flow along the reverse of the arc; that is, $(i,j) \in A_{f'}$ since $(j,i) \in A_f$. In this case, we know that $d_j = d_i + 1$, which implies that $d_i = d_j - 1 \le d_j + 1$. Thus the claim holds.

We want to show that $d'_s > d_s$. Look at any s-t path P in $A_{f'}$. By the properties of a blocking flow, there exists an arc $(i,j) \in P$ that was not admissible in the previous iteration;

$(i,j) \notin \tilde{A}$. In other words, $d_i \neq d_j + 1$, which implies that $d_i \leq d_j$ since $d_i \leq d_j + 1$. Since d_i is a distance labelling for the arcs in $A_{f'}$, this implies that $|P| > d_s$, which implies that $d'_s > d_s$. The reason for this can also be seen in Figure 11.2. Recall our definition of a *distance level* $D_k = \{i \in V : d_i = k\}$. After one iteration, any path you take will have to use

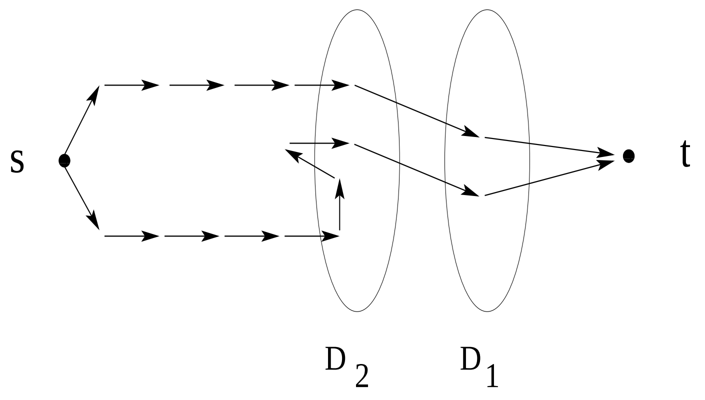


Figure 11.2: s-t paths with respect to distances d_i before and after a blocking flow.

an arc that stays at the same distance level or goes backwards with respect to the distances d_i; this implies that the distance from the source to the sink must be larger than d_s. □

11.1.2 Blocking flows in unit capacity graphs

In some cases, we can show that blocking flow algorithms give a much better result. We consider the special case of *unit capacity* graphs.

Definition 11.3 A graph has unit capacity if $u_{ij} \in \{0,1\}$ for all arcs $(i,j) \in A$.

In this case, we can give the following result.

Lemma 11.4 For unit capacity graphs, Dinic's algorithm takes $O(\min(m^{\frac{1}{2}}, n^{\frac{2}{3}}))$ iterations.

Proof: We define a number of s-t cuts $S_k = \{i : d_i \geq k\}$; note that for $k > 0$, $s \in S_k$ and $t \notin S_k$.

Suppose first that $d_s \geq m^{\frac{1}{2}}$. Then there exists a distance level D_k such that there are at most $m^{\frac{1}{2}}$ arcs in S_k. As can be seen in Figure 11.3, arcs from D_k to D_{k-1} are disjoint for all available distance levels and there are at most m arcs. By the Pigeonhole Principle, this implies that if there are at least $m^{\frac{1}{2}}$ distance levels, then there exists a D_k such that there are at most $m^{\frac{1}{2}}$ arcs from D_k to D_{k-1}. Therefore, the residual capacity of the cut S_k is at most $m^{\frac{1}{2}}$ since the graph is unit capacity (that is, $u^f(\delta^+(S_k)) \leq m^{\frac{1}{2}}$). Thus we know that only $m^{\frac{1}{2}}$ more augmentations will be required until the algorithm finds a maximum flow. The algorithm takes $\sqrt{m}$ iterations until the distance from the source is $d_s \geq m^{\frac{1}{2}}$, and $\sqrt{m}$ more iterations until the flow is maximum, for a total of $O(\sqrt{m})$ iterations.

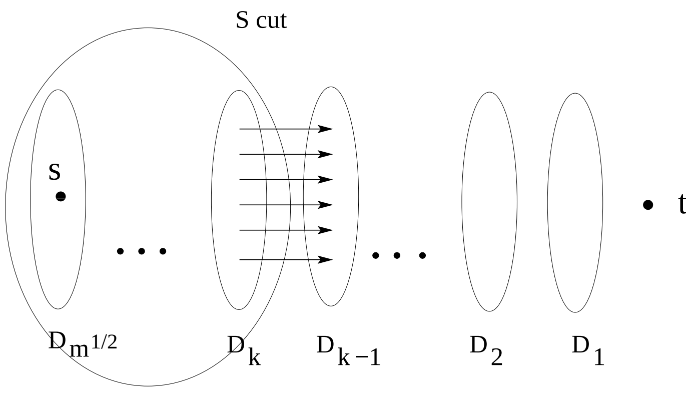


Figure 11.3: Apply pigeonhole principle by considering all arcs from D_k to D_{k-1}.

Let us now suppose that $d_s \geq 2n^{\frac{2}{3}}$. Then, there exists D_k, D_{k-1} such that $|D_k| \leq n^{\frac{1}{3}}$ and $|D_{k-1}| \leq n^{\frac{1}{3}}$, again by the pigeonhole principle. Since there are n vertices that are partitioned into the distinct distance levels, we can't have $2n^{\frac{2}{3}}$ distance levels such that every D_k with $|D_k| \leq n^{\frac{1}{3}}$ is followed by $|D_{k-1}| > n^{\frac{1}{3}}$. If we now consider all possible arcs

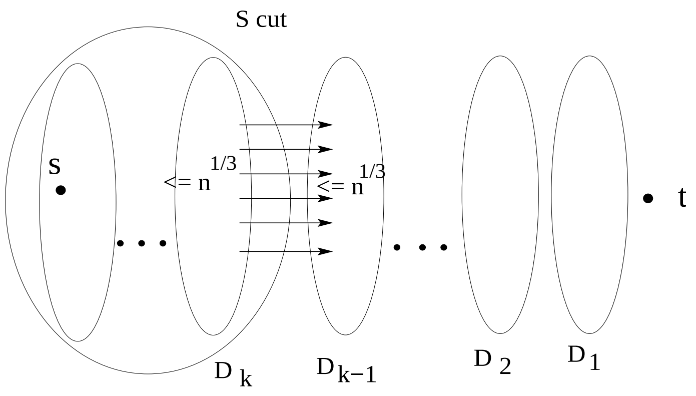


Figure 11.4: Second case of Theorem.

from D_k to D_{k-1} (as in Figure 11.4), there can be at most $n^{\frac{2}{3}}$ arcs. Thus the residual capacity of the cut S_k is $u^f(\delta^+(S_k)) \leq n^{\frac{2}{3}}$, since all arcs have unit capacity. This proves that only $n^{\frac{2}{3}}$ more iterations are needed to find the maximum flow. Thus as in the previous case, after $2n^{\frac{2}{3}}$ iterations, $d_s \geq 2n^{\frac{2}{3}}$ and after $n^{\frac{2}{3}}$ iterations, the maximum flow is achieved, for a total of $O(n^{\frac{2}{3}})$ iterations. □

We claim without proof that in unit capacity graphs, it is easy to find a blocking flow.

Claim 11.5 In unit capacity graphs, the blocking flow can be found in $O(m)$ time.

Because the quantities in the proof above will come up so frequently in following lectures, let's set aside a special symbol for them.

Definition 11.4

$$\Lambda = \min(m^{\frac{1}{2}}, 2n^{\frac{2}{3}}).$$

Thus combining the above, we obtain the following.

Theorem 11.6 In unit capacity graphs, the maximum flow can be found in $O(\Lambda m)$ time.

In the next lecture, we will consider how to apply the ideas of this algorithm to graphs with general capacities. Here's an idea to start with. Suppose we can somehow make sure that the arcs from D_k to D_{k-1} have residual capacity at most Δ. Then we know that after Λ iterations, we know (by the proof above) that the remaining residual capacity is $\Lambda\Delta$. This somehow seems useful. How can we obtain such a property? The basic idea we will consider is that of altering the distance function. Up until now, the distance of a vertex to the sink has been the number of arcs on the shortest path. But of course we could have general lengths on the arcs. If we change the length of each arc to be the following:

$$l_{ij} \leftarrow \begin{cases} 1 & \text{if } u^f_{ij} < \Delta \\ 0 & \text{otherwise} \end{cases}$$

then we will get the property that we want, namely, the arcs from D_k to D_{k-1} will have residual capacity at most Δ. In the next lecture we will see how this idea plays out.

ORIE 633 Network Flows February 20, 2004

Lecture 12

Lecturer: David P. Williamson *Scribe: Stefan Wild*

12.1 More polynomial-time algorithms for the maximum flow problem

12.1.1 Blocking flows (cont.)

Recall the algorithm from last time:

Dinic's blocking flow algorithm

```
$f \leftarrow 0$
while $\exists s - t$ path in $G_f$
    Compute distances $d_i$ to $t$ in $A_f$.
    Compute a blocking flow $\tilde{f}$ on admissible arcs $(i,j) \in A_f$ (i.e. $d_i = d_j + 1$).
    $f \leftarrow f + \tilde{f}$.
```

Definition 12.1 Define $\Lambda = \min\{m^{\frac{1}{2}}, 2n^{\frac{2}{3}}\}$.

We then had the following theorem:

Theorem 12.1 If $u_{ij} \in \{0,1\}$ for all $(i,j) \in A$, then we can find a max flow in $\mathcal{O}(\Lambda)$ blocking flows.

Proof: (Sketch of proof from last time:) Define the kth *distance level* $D_k = \{i \in V : d_i = k\}$; let the s-t cut be $S_k = \{i \in V : d_i \geq k\}$. Then observe that for any $(i,j) \in A_f$, there is at most one k such that $(i,j) \in \delta^+(S_k) \cap A_f$; namely, the value of k such that $d_i = k$, $d_j = k - 1$. Now:

(i) If $d_s \geq m^{\frac{1}{2}}$, then by the Pigeonhole Principle we showed that there exists a D_k such that there are at most $m^{\frac{1}{2}}$ arcs in A_f going from D_k to D_{k-1}. This implies that $|\delta^+(S_k) \cap A_f| \leq m^{\frac{1}{2}}$ and hence $u^f(\delta^+(S_k)) \leq m^{\frac{1}{2}}$ because the arcs have unit capacity. Thus we only need $m^{\frac{1}{2}}$ more augmentations to get a max flow. Note that it takes only $m^{\frac{1}{2}}$ iterations until $d_s \geq m^{\frac{1}{2}}$, so that it takes $O(m^{\frac{1}{2}})$ iterations overall to find a maximum flow.

(ii) If $d_s \geq 2n^{\frac{2}{3}}$, then by the Pigeonhole Principle we also showed that there exist D_k, D_{k-1} such that $|D_k| \leq n^{\frac{1}{3}}$ and $|D_{k-1}| \leq n^{\frac{1}{3}}$. Therefore there are at most $n^{\frac{2}{3}}$ arcs in A_f going from D_k to D_{k-1}. This implies that $u^f(\delta^+(S_k)) \leq n^{\frac{2}{3}}$ and so we only need $n^{\frac{2}{3}}$ more augmentations to get a max flow. Note that it takes only $2n^{\frac{2}{3}}$ iterations until $d_s \geq 2n^{\frac{2}{3}}$, so that it takes $O(n^{\frac{2}{3}})$ iterations overall to find a maximum flow.

□

12.1.2 The Goldberg-Rao algorithm

How can we use this to help us in the case of general capacities?

<u>Idea:</u> Suppose that the arcs from D_k to D_{k-1} all have residual capacity no more than Δ for all k. Then after Λ blocking flows we'll have a cut with residual capacity no more than $\Delta\Lambda$. This seems like it is useful; the amount of remaining flow is reduced significantly with a relatively few blocking flow computations. Then we will reduce Δ and repeat.

<u>Problem 1:</u> How do we get manage to get all arcs from D_k to D_{k-1} to have residual capacity at most Δ?

<u>Solution 1:</u> Change our notion of distance. For all $(i,j) \in A_f$, set:

$$l'_{ij} \leftarrow \begin{cases} 1 & \text{if } u^f_{ij} < \Delta \\ 0 & \text{otherwise,} \end{cases}$$

and revise our definitions:

$$\begin{aligned} d'_i &= \text{distance to } t \text{ using edge lengths } l'_{ij} \\ D'_k &= \{i : d'_i = k\} \\ \text{Arc } (i,j) \in A_f \ \underline{\text{Admissible}} &\Leftrightarrow d'_i = d'_j + l'_{ij}. \end{aligned}$$

Note that this does exactly what we want: now any arc from distance level k to distance level $k-1$ must in fact have length 1, and therefore it must have residual capacity no more than Δ.

This new idea of lengths causes its own set of problems, however. In particular we have:

<u>Problem 2:</u> The graph of admissible arcs might have cycles (since some l'_{ij} may equal 0). This is an issue since the efficient algorithms we know for blocking flows only run on acyclic graphs.

Problem 3: In order for the blocking flow style proof to work, we need to have d'_s increase in each iteration, and it's not obvious that it will under the new definitions.

We will table Problem 3 for the time being, and attempt to address Problem 2.

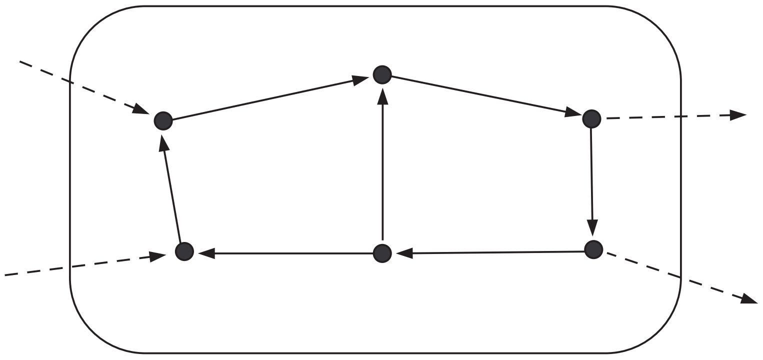


Figure 12.1: A set of strongly-connected components of admissible arcs to be contracted.

Solution 2: Suppose the graph of admissible arcs has cycles. Then we will "shrink" (contract) each strongly-connected component of admissible arcs to a single node and then run the blocking flow algorithm. See Figure 12.1.

But then we have a new problem:

Problem 4: How do we route flow in the "unshrunken" arcs? The arcs inside the strongly connected components have capacity at least Δ (since they had length zero), but it is possible that the total flow going in and coming out of a strongly connected component overwhelms the capacity of those arcs.

Solution 4: Limit the flow so that flow in/ flow out (the dotted arcs in Figure 12.1) is less than $\frac{\Delta}{4}$. We can do this by changing the basic step in each iteration from "find a blocking flow" to "either find a flow of value $\frac{\Delta}{4}$ or find a blocking flow of value at most $\frac{\Delta}{4}$." Then given a flow on the graph with shrunken components, we can easily route the flow on the graph with unshrunken components as follows:

(i) For each of the unshrunken strongly-connected components, pick some root node, r;

(ii) Build 2 trees: an *intree* to r, and an *outtree* from r (see Figure 12.2);

(iii) Use the intree to route incoming flow to r and the outtree to route outgoing flow from r.

Each edge is then used at most twice (at most once in the intree and at most once in the outtree) so by routing at most $\frac{\Delta}{4}$ flow on each of these trees, we're only using $\frac{\Delta}{2}$ capacity of the the arcs, each of which by definition has capacity at least Δ. We are now ready to present the algorithm:

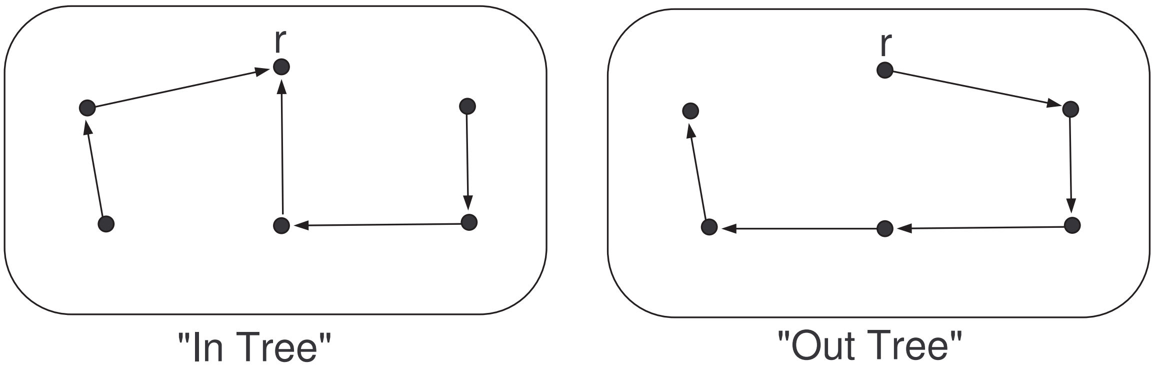


Figure 12.2: In and out trees.

[Almost] Goldberg-Rao (1998)

```
$f \leftarrow 0$
$F \leftarrow mU$ where $U = \max_{(i,j)\in A} u_{ij}$
While $F \geq 1$
    $\Delta \leftarrow \frac{F}{2\Lambda}$
    Repeat 5 $\Lambda$ times:
        $l'_{ij} \leftarrow \begin{cases} 1 & \text{if } u^f_{ij} < \Delta \\ 0 & \text{otherwise,} \end{cases} \forall (i,j) \in A_f$
        Compute distances $d'_i$ to sink $t$;
        Shrink strongly-connected components of admissible arcs;
        Find $\tilde{f}$ in the shrunken graph:
            either a flow of value $\frac{\Delta}{4}$
            or a blocking flow of value $\leq \frac{\Delta}{4}$;
        $\hat{f} \leftarrow \tilde{f}$ with flows routed in the shrunken components;
        $f \leftarrow f + \hat{f}$
    $F \leftarrow \frac{F}{2}$
```

We now prove a lemma that we will need to bound the running time.

Lemma 12.2 F is an upper bound on the flow value in G_f.

Proof: We prove the statement by induction on the algorithm. First note that it's true initially; $F = mU$ is an upper bound on the total amount of flow.

Now consider the repeat loop. After 5Λ times, either:

- $d'_s \geq \Lambda$ (i.e. we compute a blocking flow at least Λ times). By the blocking flow arguments used in unit capacity graphs, this implies that there is a cut in G_f of residual capacity at most $\Lambda\Delta = \frac{F}{2}$ and hence the remaining flow in G_f is less than or equal to $\frac{F}{2}$.

- Flow has increased by $\Lambda\Delta = \frac{F}{2}$ (i.e. we found a $\frac{\Delta}{4}$ flow at least 4Λ times). This implies that there will be no more than $\frac{F}{2}$ units of flow remaining in G_f, since there were at most F initially.

In either case, we can legitimately reduce F by a factor of 2 after we have repeated the main step of the algorithm 5Λ times. $\square$

Next time we will come back to Problem 3, and show that in fact d'_s does strictly increase each time we compute a blocking flow; however, we will need to make a slight change to the algorithm to get this to work out.

ORIE 633 Network Flows February 23, 2004

Lecture 13

Lecturer: David P. Williamson *Scribe: Patrick Kongsilp*

13.1 More polynomial-time algorithms for the maximum flow problem

13.1.1 The Goldberg-Rao algorithm (cont.)

Recall from last time the Goldberg-Rao maximum flow algorithm.

[Almost] Goldberg-Rao (1998)

```
$f \leftarrow 0$
$F \leftarrow mU$ where $U = \max_{(i,j)\in A} u_{ij}$
While $F \geq 1$
    $\Delta \leftarrow \frac{F}{2\Lambda}$
    Repeat 5 $\Lambda$ times:
        $l'_{ij} \leftarrow \begin{cases} 1 & \text{if } u^f_{ij} < \Delta \\ 0 & \text{otherwise,} \end{cases} \quad \forall (i,j) \in A_f$
        Compute distances $d'_i$ to sink $t$;
        Shrink strongly-connected components of admissible arcs;
        Find $\tilde{f}$ in the shrunken graph:
            either a flow of value $\frac{\Delta}{4}$
            or a blocking flow of value $\leq \frac{\Delta}{4}$;
        $\hat{f} \leftarrow \tilde{f}$ with flows routed in the shrunken components;
        $f \leftarrow f + \hat{f}$
    $F \leftarrow \frac{F}{2}$
```

The last bit of analysis left over from last time is to show that under the new definitions of distances d' given the lengths l', the blocking flow analysis goes through as before; namely, if we compute a blocking flow, then the distance from the source to the sink has strictly increased.

Lemma 13.1 If we compute a blocking flow, then d'_s strictly increases.

Proof: Let $\tilde{A}$ be the set of admissible arcs. ($\tilde{A} = \{(i,j) \in A_f : d'_i = d'_j + l'_{ij}\}$) Let d'_i be distance labels in one iteration, d''_i the next iteration. Let l'_{ij} be length labels in one iteration, l''_{ij} the next iteration. Let f' be the flow in one iteration, f'' the next iteration.

The proof structure we want to follow is the same that we used for Dinic's algorithm: we first show that the distances d'_i are a valid distance labelling for arcs in the residual graph of the flow in the next iteration: that is, we show that for all $(i,j) \in A_{f''}$ it is the case that $d'_i \leq d'_j + l''_{ij}$. Then we observe that by the properties of a blocking flow, in any s-t path in the residual graph of f'', there must be some arc that wasn't admissible in the previous iteration; that is, there is some arc (i,j) in any s-t path of $A_{f''}$ such that $d'_i < d'_j + l'_{ij}$. From this we hope to infer that the length of the path must in fact be greater than d'_s.

First, we show that d'_i is a valid distance labelling for $A_{f''}$. If (i,j) is in the residual graph for f'', then it must be the case that either (i,j) was in the residual graph for f', or that (j,i) was in the residual graph for f' and (j,i) was admissible. In the latter case, if (j,i) was admissible, then $d'_j = d'_i + l'_{ji}$, which implies that $d'_i = d'_j - l'_{ji} \leq d'_j + l''_{ij}$ and we're done. If on the other hand $(i,j) \in A_{f'}$ and $d'_i \leq d'_j + l'_{ij}$, the only bad case is if $d'_i = d'_j + l'_{ij}$, $l'_{ij} = 1$, and $l''_{ij} = 0$. Suppose we run into this case. But, $d'_i = d'_j + l'_{ij} = d'_j + 1$ implies that (j,i) is not admissible. Thus we can't have pushed any flow on (j,i), so the residual capacity of (i,j) in f' must be no less than that of (i,j) in f''. Therefore, $u^{f'}_{ij} \geq u^{f''}_{ij} \geq \Delta$, since $l''_{ij} = 0$. But this contradicts our assumption that $l'_{ij} = 1$.

Now by the properties of a blocking flow, we know that for any s-t path P in $A_{f''}$ there exists $(i,j) \in P$ such that (i,j) was not admissible; that is, $d'_i < d'_j + l'_{ij}$. We want to show that $d'_i < d'_j + l''_{ij}$. This will imply that the length of P under lengths l''_{ij} must be strictly greater than d'_s.

Suppose it is not the case that $d'_i < d'_j + l''_{ij}$, and thus $d'_i = d'_j + l''_{ij}$. What could happen so that this occurs? This can happen if $l'_{ij} = 1$, $l''_{ij} = 0$, and $d'_i = d'_j$. However, the case that $l'_{ij} = 1$ and $l''_{ij} = 0$ can only happen if flow was sent from j to i. This implies that (j,i) is admissible, which further implies that $l'_{ji} = 0$, since $d'_i = d'_j$.

Unfortunately, nothing in the algorithm so far prevents this bad case from happening. So we make one final change to the algorithm to ensure that this case cannot happen. We change the definition of edge lengths as follows:

$$l'_{ij} = \begin{cases} 0, & \text{if } u^f_{ij} \geq \Delta \\ 1, & \text{if } u^f_{ij} < 0 \end{cases}$$

as before. However, if $d'_i = d'_j$ and $\Delta/2 \leq u^f_{ij} < \Delta$ and $u^f_{ji} \geq \Delta$, then (i,j) is considered to be a "special arc", and we set $l'_{ij} = 0$.

Before continuing with the proof, we first quickly observe that this change does not break the rest of the algorithm. What changes?

- Note that special arcs are admissible; since $d'_i = d'_j$ for special arc (i,j), and $l'_{ij} = 0$, we have that $d'_i = d'_j + l'_{ij}$.
- Note also that distances do not change, since for special arcs it is already the case that $d'_i = d'_j$.
- Finally, since $l'_{ij} = 0$, special arcs could be in shrunken strongly connected components. But we will still be able to route flow through them in the way that we mentioned

earlier since the total amount of flow routed through an arc in a shrunken component was at most $\Delta/2$, and the capacity of any special arc is at least $\Delta/2$.

Now let's finish the proof, taking into account the "fix". We have a bad case when (i, j) is not admissible, $l'_{ij} = 1$, $l''_{ij} = 0$, and $d'_i = d'_j$. In order to have $l'_{ij} = 1$ and $l''_{ij} = 0$ (i.e. the capacity increasing from one iteration to the next), it must have been the case that we pushed flow across (j, i) and that (j, i) is admissible. Since $d'_i = d'_j$, (j, i) admissible implies that $l'_{ji} = 0$. Since (i, j) is not admissible, it cannot be a special arc. Thus, $u^f_{ij} < \Delta/2$.

To make $l''_{ij} = 0$ (i.e. $u^{f''}_{ij} \geq \Delta$), more than $\Delta/2$ units of flow must have been pushed across (j, i). But this cannot happen, since in one iteration flow is never increased by more than $\Delta/2$ on any arc. □

We can now give the running time of the algorithm. Observe that the main loop is executed $\log mU$ times; in each execution of the loop we execute a blocking flow algorithm $O(\Lambda)$ times; and we can run a blocking flow algorithm in $O(m \log n)$ time.

Theorem 13.2 (Goldberg, Rao 1998) Max flow can be computed in $O(\Lambda m \log n \log(mU))$ time.

Note that for reasonable values of U this is $o(mn)$. It is a big open question if this algorithm can be made to have a strongly polynomial running time that is also $o(mn)$.

ORIE 633 Network Flows February 25, 2004

Lecture 14

Lecturer: David P. Williamson *Scribe: Christina Peraki*

14.1 Types of polynomial time

There are different flavors of polynomial-time algorithms that we have mentioned in passing; now we will formalize the definitions.

Definition 14.1 An algorithm runs in **polynomial time** if the number of basic operations (arithmetic operations, compares, branches, etc) can be bounded above by a polynomial in the size of the input with data items encoded in binary (e.g. capacities, costs, etc). This is also known as **weakly polynomial time**.

If data items such as capacities u are coded in binary, then to run in time bounded by a polynomial in the input size, we must run in time bounded by a polynomial in $\log u$. As an example, the capacity scaling algorithm for the maximum flow problem runs in $O(m^2 \log U)$ time, and is a weakly polynomial-time algorithm.

Definition 14.2 An algorithm runs in **pseudopolynomial time** if the number of basic operations (arithmetic operations, compares, branches, etc) can be bounded above by a polynomial in the size of the input with data items encoded in unary.

By "unary", we mean that we write down u bits for the data item u. The augmenting path algorithm for maximum flow is a pseudopolynomial-time algorithm since it runs in $O(m^2 U)$ time.

Definition 14.3 An algorithm runs in **strongly polynomial time** if the number of basic operations (arithmetic operations, compares, branches, etc) can be bounded above by a polynomial in the number of data items that were input and is not dependent on the size of the input.

For example, shortest augmenting path algorithm for the maximum flow problem runs in $O(m^2 n)$ time, and the FIFO push-relabel algorithm runs in $O(n^3)$ time.

14.2 Minimum-cost flows

We now turn to flow problems that include costs.

Minimum-cost circulation problem

- **Input:**
  - A directed graph $G = (V, A)$.
  - Integer costs $c_{ij} \geq 0$, $\forall (i,j) \in A$.
  - Integer capacities $u_{ij} \geq 0$, $\forall (i,j) \in A$.
  - Integer demands $0 \leq l_{ij} \leq u_{ij}$, $\forall (i,j) \in A$.
- **Goal:** Find a circulation f that minimizes $\sum_{(i,j)\in A} c_{ij} f_{ij}$.

Now we define a circulation.

Definition 14.4 A circulation $f : A \to R \geq 0$ such that

$$\begin{array}{rl} l_{ij} \leq f_{ij} \leq u_{ij}, & \forall (i,j) \in A \\ \sum_{k:(i,k)\in A} f_{ik} - \sum_{k:(k,i)\in A} f_{ki} = 0, & \forall i \in V. \end{array}$$

We will show below that this is equivalent to the more commonly studied *minimum-cost flow problem.* In the minimum-cost flow problem the input is the same (a directed graph $G = (V, A)$ with integer costs $c_{ij} \geq 0$ and integer capacities $u_{ij} \geq 0$ for each edge $(i,j) \in A$). The difference is that there are no demands l_{ij} but instead, there are integer demands b_i $\forall i \in V$, such that the sum of demands over all the vertices is zero: $\sum_{i\in V} b_i = 0$. The goal of the minimum-cost flow problem is to find a flow that minimizes the cost $\sum_{(i,j)\in A} c_{ij} f_{ij}$ such that

$$\begin{array}{rl} 0 \leq f_{ij} \leq u_{ij}, & \forall (i,j) \in A, \\ \sum_{k:(k,i)\in A} f_{ki} - \sum_{k:(i,k)\in A} f_{ik} = b_i, & \forall i \in V. \end{array}$$

Theorem 14.1 The minimum-cost flow problem and the minimum-cost circulation problem are equivalent.

Proof: (*flow $\Rightarrow$ circulation*) Given an instance of the minimum-cost flow problem, add a node s to the graph. For $i \in V$ such that $b_i > 0$ then attach an arc (i, s) with cost 0, and $l_{is} = u_{is} = b_i$. For $i \in V$ such that $b_i < 0$ we attach an arc (s, i) of cost 0 such that $l_{si} = u_{si} = |b_i|$ (See Figure 14.1). Note that given a feasible flow in the original problem we can get a circulation of the same cost in the modified instance since the flow coming into each node is equal to the flow going out of each node (including the node s, since $\sum_{i:b_i>0} b_i = \sum_{i:b_i<0} |b_i|$). The reverse is also true – given a circulation in the modified instance, the flow on the arcs of the original problem is a feasible flow of the same cost. So by finding a minimum-cost circulation in the modified instance we can find a minimum-cost flow in the original instance.

(*circulation $\Rightarrow$ flow*) For this part, we change variables. Set $f'_{ij} = f_{ij} - l_{ij}$, and $u'_{ij} = u_{ij} - l_{ij}$. Set $b_i = \sum_{k:(i,k)\in A} l_{ik} - \sum_{k:(k,i)\in A} l_{ki}$. This provides a direct transformation

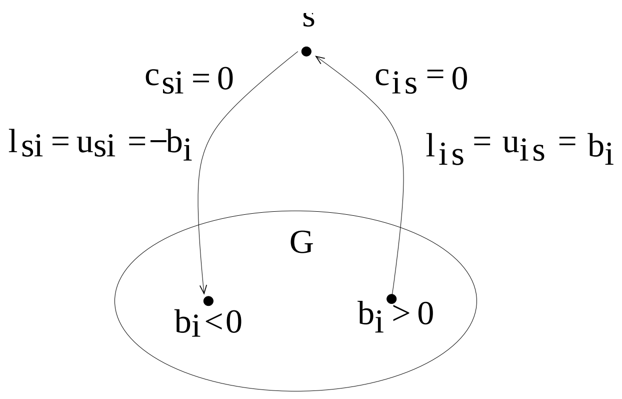


Figure 14.1: Transformation of minimum-cost flow instance to minimum-cost circulation instance.

between the two problems. Given a feasible circulation f in the original problem, we have a feasible flow f' in the modified problem of the same cost, and vice versa. Thus by finding a minimum-cost flow in the modified instance we can find a minimum-cost circulation in the original instance. □

From here on we will consider only the minimum-cost circulation problem.

We will now change our notation slightly for the problem, as we did for the maximum flow problem, since it will make our algorithms and proofs simpler. Replace each arc by two arcs of opposite orientations. If f_{ij} is the flow in (i, j), then force $f_{ji} = -f_{ij}$. This is called antisymmetry. Also set $u_{ji} = -l_{ij}$. This removes the lower bound constraints, since $f_{ji} \leq u_{ji} \Rightarrow -f_{ij} \leq -l_{ij} \Rightarrow f_{ij} \geq l_{ij}$. We make the costs antisymmetric, too: $c_{ji} = -c_{ij}$. Thus the total cost for the two edges with flow f is $c_{ji}f_{ji} + c_{ij}f_{ij} = 2c_{ij}f_{ij}$. Hence optimizing for the total cost for this new graph is the same as optimizing for the total cost for the original graph. Thus our definition of a feasible circulation becomes the following.

Definition 14.5 A circulation $f : A \to R$ such that

$$\begin{aligned} f_{ij} &\leq u_{ij}, & \forall (i,j) \in A \\ f_{ij} &= -f_{ji}, & \forall (i,j) \in A \\ \textstyle\sum_{k:(i,k)} f_{ik} &= 0, & \forall i \in V \end{aligned}$$

We will use the following claim frequently.

Claim 14.2 Via one max flow computation, we can tell if the circulation problem is feasible and find a feasible circulation if one exists.

Proof: See Problem Set 1 solutions. □

In the case of the maximum flow problem, we had conditions that told us when a flow was optimal; i.e. we knew a flow was maximum if and only if there was no augmenting path. We would like to give similar conditions for the minimum-cost circulation problem, but we need a few definitions first.

Definition 14.6 A residual graph for a circulation f is $G_f = (V, A_f)$ where $A_f = \{(i,j) \in A : f_{ij} < u_{ij}\}$ with residual capacity $u_{ij}^f = u_{ij} - f_{ij}$.

Definition 14.7 Let $p : V \to R$. Then p are called *node potentials* (or sometimes *node prices*). The *reduced cost* of (i,j) with respect to potentials p is $c_{ij}^p = c_{ij} + p_i - p_j$. If Γ is a cycle, let $c(\Gamma) = \sum_{(i,j)\in\Gamma} c_{ij}$.

Observe that the cost of a cycle Γ and the reduced cost of a cycle Γ is the same for any set of potentials p; that is, $c(\Gamma) = c^p(\Gamma)$, since the potentials cancel out (see Figure 14.2).

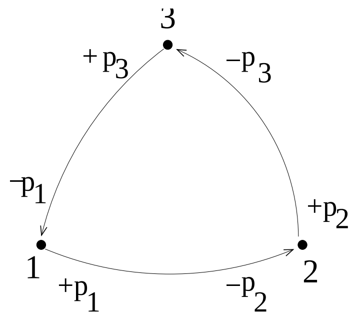


Figure 14.2: Example showing cost of cycle is same as the reduced cost of the cycle.

We can now state the theorem giving us conditions under which a circulation is optimal. Next time we will prove the theorem.

Theorem 14.3 The following are equivalent:

1. f is a minimal cost flow,
2. there are no negative cost cycles in G_f, and,
3. there exist potentials p such that $c_{ij}^p \geq 0$ for all $(i,j) \in A_f$.

ORIE 633 Network Flows February 27, 2004

Lecture 15

Lecturer: David P. Williamson *Scribe: Alice Cheng*

15.1 Minimum-cost circulations

Recall the minimum-cost circulation problem, introduced in the previous lecture:

Minimum-cost circulation problem

- **Input:**
  - A directed graph $G = (V, A)$.
  - Integer costs $c_{ij} \geq 0$, $\forall (i,j) \in A$.
  - Integer capacities $u_{ij} \geq 0$, $\forall (i,j) \in A$.
  - Integer demands $0 \leq l_{ij} \leq u_{ij}$, $\forall (i,j) \in A$.
- **Goal:** Find a minimum-cost circulation.

The goal is to find a flow $f : A \to \mathbb{R}^{\geq 0}$ that minimizes $\sum_{(i,j)\in A} c_{ij} f_{ij}$ such that

$$\begin{aligned} l_{ij} \leq f_{ij} \leq u_{ij}, \qquad & \forall (i,j) \in A \\ \textstyle\sum_{k:(i,k)\in A} f_{ik} - \sum_{k:(k,i)\in A} f_{ki} = 0, \qquad & \forall i \in V \end{aligned}$$

In the previous lecture, we defined a notation change for circulations similar to the one we defined for s-t flows.

Definition 15.1 A *circulation* f satisfies the following:

1. $f_{ij} \leq u_{ij} \ \forall \ (i,j) \in A$
2. $f_{ij} = -f_{ji}, \ \forall \ (i,j) \in A$
3. $\sum_{k:(k,i)\in A} f_{ki} = 0$

In the new definition, flow in the original arc f_{ij} satisfies the constraints $l_{ij} \leq f_{ij} \leq u_{ij}$, and each unit of flow incurs cost c_{ij}. Flow on the reverse arc f_{ji} satisfies $f_{ji} \leq u_{ji} = -l_{ij}$ and incurs cost $c_{ji} = -c_{ij}$ per unit of flow. The total cost for the two edges with flow f is $c_{ji} f_{ji} + c_{ij} f_{ij} = 2c_{ij} f_{ij}$. Hence optimizing for the total cost for this new graph is the same as optimizing for the total cost for the original graph.

15.1.1 Residual graph

Given a flow f on G, define the *residual graph* $G_f = (V, A_f)$ where the new arc set

$$A_f := \{(i,j) \in A : f_{ij} < u_{ij}\}.$$

Note that we are using the new notation here. Impose the upper bounds $u^f_{ij} = u_{ij} - f_{ij}$. Then clearly $u^f_{ij} > 0$ for all $(i,j) \in A_f$.

15.1.2 Potentials

Definition 15.2 A *potential* is a function $p : V \to \mathbb{R}$.

Definition 15.3 Given a potential p, define the *reduced cost* $c^p_{ij} := c_{ij} + p_i - p_j$. Then $c^p_{ji} = -c^p_{ij}$.

The potential plays the role of the dual variable. We shall show this formally in another lecture.

Definition 15.4 The cost of a cycle Γ is $c(\Gamma) = \sum_{(i,j)\in\Gamma} c_{ij}$

Observe that if Γ is a cycle and $c(\Gamma) := \sum_{(i,j)\in\Gamma} c_{ij}$, and $c^p(\Gamma)$ is defined similarly, then $c^p(\Gamma) = c(\Gamma)$.

Definition 15.5 If f is a circulation, let $c \cdot f = \sum_{(i,j)\in A} c_{ij} f_{ij}$

We can then prove the following.

Theorem 15.1 $c \cdot f = c^p \cdot f$.

Proof:

$$\begin{aligned}
c^p \cdot f &= c \cdot f + \sum_{(i,j)\in A} (p_i - p_j) f_{ij} \\
&= c \cdot f + \sum_{i\in V} p_i \Big(\sum_{k:(i,k)\in A} f_{ik} - \sum_{k:(k,i)\in A} f_{ki} \Big) \\
&= c \cdot f.
\end{aligned}$$

This follows since the term in parentheses is zero because of flow conservation. □

15.1.3 Optimality conditions

We now characterize the minimum-cost circulation.

Theorem 15.2 The following are equivalent:

1. f is a minimal cost circulation,
2. There are no negative cost cycles in G_f, and,
3. There exists a potential p such that $c^p_{ij} \geq 0$ for all $(i,j) \in A_f$.

Proof:

$[\neg(2) \Rightarrow \neg(1)]$ Let Γ be a negative cost cycle in A_f. Define

$$\delta = \min_{(i,j)\in\Gamma} u^f_{ij}.$$

Then $\delta > 0$. Let

$$f'_{ij} = \begin{cases} f_{ij} + \delta, & (i,j) \in \Gamma, \\ f_{ij} - \delta, & (j,i) \in \Gamma, \\ f_{ij}, & \text{otherwise.} \end{cases}$$

Thus, $f'_{ij} = -f'_{ji}$ and f' is a feasible circulation if f is. Also, $f'_{ij} \leq u_{ij}$. Furthermore,

$$c \cdot f' = c \cdot f + 2\delta c(\Gamma) < c \cdot f,$$

since Γ is a negative cost cycle. Therefore, f is not of minimum cost.

Note: In $G_{f'}$, Γ does not exist. This is so because $f'_{ij} = u_{ij}$ for some $(i,j) \in \Gamma$. Then $(i,j) \notin A_{f'}$, and so $\Gamma \not\subseteq A_{f'}$. We say that Γ has been *cancelled*.

$[(2) \Rightarrow (3)]$ Add a node s to G_f, and add arcs of cost 0 from s to each $i \in V$. Then let p_i be the length of the shortest path from s to i using costs c_{ij} as the edge lengths. These paths are well defined since there are no negative-cost cycles, by assumption. Moreover, by properties of shortest paths, for any $(i,j) \in A_f$, $p_j \leq p_i + c_{ij}$, so that $c^p_{ij} = c_{ij} + p_i - p_j \geq 0$.

$[(3) \Rightarrow (1)]$ Suppose f^* is any other valid circulation. We want to show that $c \cdot f \leq c \cdot f^*$. Consider the circulation f', where $f'_{ij} = f^*_{ij} - f_{ij}$. f' is a feasible circulation. Let p be a potential such that $c^p_{ij} \geq 0$ for all $(i,j) \in A_f$. Note that if $f'_{ij} > 0$ then $f_{ij} < f^*_{ij} \leq u_{ij}$. This implies $(i,j) \in A_f$ and $c^p_{ij} \geq 0$. Consider the following.

$$\begin{aligned} c \cdot f' = c^p \cdot f' = \sum_{(i,j)\in A} c^p_{ij} f'_{ij} &= \sum_{(i,j)\in A, f'_{ij}>0} c^p_{ij} f'_{ij} + \sum_{(i,j)\in A, f'_{ij}<0} (-c^p_{ji})(-f'_{ji}) \\ &= 2\left(\sum_{(i,j)\in A, f'_{ij}>0} c^p_{ij} f'_{ij} \right) \geq 0. \end{aligned}$$

Thus, $c \cdot f^* = c \cdot (f' + f) \geq c \cdot f$. Therefore f is a min-cost circulation.

□

15.1.4 A cycle-cancelling algorithm

This theorem yields a natural algorithm for computing a min-cost circulation:

Cycle-Cancelling Algorithm (Klein '67)

```
Let $f$ be a feasible circulation.
While $A_f$ contains a negative cycle $\Gamma$
    Cancel $\Gamma$, update $f$.
```

The correctness of the algorithm follows immediately from the above theorem. Note that we can always find a feasible circulation, if one exists, by running one max flow computation (see Problem Set 1, # 3). Furthermore, we can find a negative cycle, if one exists, in $O(mn)$ time (Problem Set 3).

Also, notice that the algorithm implies that min-cost circulations, like max-flows, satisfies an **integrality property**: If u_{ij} and c_{ij} are integer for all $(i, j) \in A$, then if a feasible circulation exists, there is always integer-valued minimum-cost circulation. This is true, since we can always cancel a cycle with integer flow during each iteration of the cycle-cancelling algorithm.

To get a bound on the running time of the algorithm, define

$$U = \max_{(i,j)\in A} u_{ij} \qquad C = \max_{(i,j)\in A} |c_{ij}|.$$

Then any feasible circulation can cost at most mCU and must cost at least $-mCU$. Therefore, since a cycle cancellation improves the cost of a circulation by at least 1, at most $O(mCU)$ cancellations are needed in order to find an optimal circulation. This gives us a pseudopolynomial running time of $O(m^2nCU)$.

As with the augmenting path algorithm for the maximum flow problem, we can obtain a polynomial-time algorithm by a better choice of cycle at each iteration. Consider the following.

Definition 15.6 Let the *mean cost* of a cycle Γ be $\frac{c(\Gamma)}{|\Gamma|}$ where $c(\Gamma)$ is the cost of the cycle.

Definition 15.7 Given a circulation f, let $\mu(f)$ be the minimum mean-cost cycle in G_f:

$$\mu(f) = \min_{\text{cycle } \Gamma\subseteq A_f} \frac{c(\Gamma)}{|\Gamma|}$$

We will show next time that cancelling the minimum mean-cost cycle in each iteration gives a polynomial-time algorithm.

ORIE 633 Network Flows March 1, 2004

Lecture 16

Lecturer: David P. Williamson *Scribe: Dhruv Bhargava*

16.1 Polynomial-time algorithms for minimum-cost circulations

16.1.1 Minimum mean-cost cycle cancelling

Recall that last class we proved the following theorem:

Theorem 16.1 The following are equivalent:

1. f is a min cost circulation
2. There are no negative cost cycles in the residual graph G_f
3. $\exists$ potentials p such that $c^p_{ij} \geq 0$ for all $(i,j) \in A_f$.

Then we suggested an algorithm for cancelling negative-cost cycles. As in the case of the maximum flow algorithm, the naive algorithm prompted by the optimality theorem does not lead immediately to a polynomial-time algorithm. To obtain a polynomial-time algorithm, we must carefully select which negative cost cycle to cancel. It turns out we can get a polynomial-time algorithm by cancelling the *minimum mean-cost cycle*, defined below.

Definition 16.1 The **mean cost** of a cycle Γ is

$$\frac{c(\Gamma)}{|\Gamma|}$$

Definition 16.2 The **minimum mean cost cycle** in A_f is given by

$$\mu(f) = \min_{\text{cycles } \Gamma \text{ in } A_f} \frac{c(\Gamma)}{|\Gamma|}$$

We can now give the following algorithm.

Minimum mean-cost cycle cancelling algorithm (Goldberg-Tarjan '89)

```
Let $f$ be any circulation
While $\mu(f) < 0$
    Cancel min-mean cycle $\Gamma$, update $f$
```

Observe that the condition $\mu(f) < 0$ is equivalent to having a negative-cost cycle in A_f.

To have a polynomial-time algorithm, we need to be able to find the minimum mean-cost cycle in polynomial-time.

Claim 16.2 We can compute $\mu(f)$ and find the corresponding cycle in $O(mn)$ time.

Proof: See Problem Set 3. $\square$

To begin our analysis, we need to introduce a few terms.

Definition 16.3 A circulation f is ϵ-optimal if there exist potentials p s.t. $c^p_{ij} \geq -\epsilon$ for all $(i, j) \in A_f$.

Clearly f is 0-optimal if and only if f is a min cost circulation, by the third equivalence in Theorem 16.1. Further, if we have

$$C = \max_{(i,j)\in A} |c_{ij}|$$

then for any circulation, f is $C-$optimal, since if we assign $p_i = 0$ for all $i \in V$, $c^p_{ij} \geq -C$ for all $(i, j) \in A_f$.

Definition 16.4 Define $\epsilon(f)$ to be the minimum ϵ such that f is ϵ-optimal.

Interestingly, the two values of $\epsilon(f)$ and $\mu(f)$ are closely related.

Theorem 16.3 For a circulation f, $\mu(f) = -\epsilon(f)$.

Proof: We first show that $\mu(f) \geq -\epsilon(f)$. Since $c^p_{ij} \geq -\epsilon(f)$ for all $(i, j) \in A_f$, by summing over all arcs in cycle Γ we obtain that $c^p(\Gamma) \geq -\epsilon(f)|\Gamma|$. Thus

$$\mu(f) = \frac{c(\Gamma)}{|\Gamma|} = \frac{c^p(\Gamma)}{|\Gamma|} \geq -\epsilon(f).$$

for a minimum mean-cost cycle Γ.

We now show that $\mu(f) \leq -\epsilon(f)$. Set $\overline{c}_{ij}= c_{ij} - \mu(f)$. Then for any cycle Γ in A_f, $\overline{c}(\Gamma) = c(\Gamma) - |\Gamma|\mu(f)$. As $\mu(f) \leq \frac{c(\Gamma)}{|\Gamma|}$, we have $\overline{c}(\Gamma) \geq 0$. We introduce a source vertex s, connected to all vertices i with arcs of cost $\overline{c}_{si} = 0$, and define the potential p_i of node i to be the length of shortest path from s to i using costs $\overline{c}_{ij}$. By the definition of shortest path, for all $(i, j) \in A_f$, $p_j \leq p_i + \overline{c}_{ij} = p_i + c_{ij} - \mu(f)$ which implies $c^p_{ij} = c_{ij} + p_i - p_j \geq \mu(f)$ for all $(i, j) \in A_f$, which implies that $\epsilon(f) \leq -\mu(f)$. $\square$

Given circulation f, let $f^{(i)}$ denote the circulation i iterations later. The following theorems, which we will prove later, will show that the Goldberg-Tarjan algorithm runs in polynomial time.

Theorem 16.4 $\epsilon(f^{(1)}) \leq \epsilon(f)$

Theorem 16.5 $\epsilon(f^{(m)}) \leq (1 - 1/n)\epsilon(f)$

where m, n are the number of arcs and nodes in the graph, respectively.

We will also need the following.

Theorem 16.6 When $\epsilon(f) < 1/n$ then circulation f is optimal.

Proof: Since $\epsilon(f) < 1/n$, this implies that there exist potentials p such that $c^p_{ij} > -1/n$ for all $(i,j) \in A_f$. Thus for all cycles $\Gamma \in A_f$, $c^p(\Gamma) > -1$, which implies $c(\Gamma) > -1$. By the integrality of costs, this gives $c(\Gamma) \geq 0$. □

We shall now prove using the previous three results that the Goldberg-Tarjan algorithm terminates in time bounded by a polynomial in the input size.

Theorem 16.7 (Goldberg-Tarjan '89) The Goldberg-Tarjan minimum mean-cost cycle cancelling algorithm requires at most $O(mn\log(nC))$ iterations.

Proof: Any initial circulation is C-optimal. After $k = mn\log(nC)$ iterations, we have that

$$\epsilon(f^{(k)}) \leq (1 - 1/n)^{n\log(nC)}C < e^{-\log(nC)}C = 1/n,$$

using the fact that $(1 - 1/n)^n < e^{-1}$. This proves the optimality of $f^{(k)}$ by Theorem 16.6. □

The running of the Goldberg-Tarjan algorithm is $O(m^2n^2\log(nC))$ time as min-mean cycle computations take $O(mn)$ time. Note that this algorithm is not strongly polynomial. A strongly polynomial algorithm will be presented in the next lecture along with the proof of Theorem 16.5. For now, we return and prove Theorem 16.4.

Proof of Theorem 16.4: We know there exist potentials p such that

$$c^p_{ij} \geq -\epsilon(f) \text{ for all } (i,j) \in A_f$$

For the minimum-mean cost cycle Γ, $\mu(f) = -\epsilon(f)$. Since $\mu(f) = c^p(\Gamma)/|\Gamma|$, it follows that for all $(i,j) \in \Gamma$, $c^p_{ij} = -\epsilon(f)$. We now claim that $c^p_{ij} \geq -\epsilon(f)$ for all $(i,j) \in A_{f^{(1)}}$. We have $(i,j) \in A_{f^{(1)}}$ if either (i,j) was in A_f, or if $(j,i) \in \Gamma$. In the first case, $c^p_{ij} \geq -\epsilon(f)$. In the latter case, $c^p_{ij} = -c^p_{ij} = \epsilon(f) \geq 0$. In both cases, it follows that $f^{(1)}$ is $\epsilon(f)$-optimal, so the theorem statement follows. □

ORIE 633 Network Flows March 3, 2004

Lecture 17

Lecturer: David P. Williamson *Scribe: Dhruv Bhargava*

17.1 Polynomial-time algorithms for minimum-cost circulations

17.1.1 Minimum mean-cost cycle cancelling algorithm (cont.)

Recall that,

Theorem 17.1 For a circulation f the following are equivalent:

1. f is of minimum cost
2. There are no negative cost cycles in G_f
3. There exist potentials p such that $c^p_{ij} \geq 0 \; \forall (i,j) \in A_f$.

Recall also that we defined the minimum-mean cost cycle of the residual graph as

Definition 17.1 $\mu(f) = \min\limits_{\text{cycles } \Gamma \in G_f} \dfrac{c(\Gamma)}{|\Gamma|}$

We presented an algorithm for finding a minimum cost circulation.

Min cost circulation

```
Find initial circulation $f$
While $\mu(f) < 0$
    Cancel mean-min cycle $\Gamma$, update $f$
```

Definition 17.2 A circulation f is ϵ-optimal if $\exists$ potentials p such that $c^p_{ij} \geq -\epsilon \; \forall (i,j) \in A_f$. Further, we define $\epsilon(f)$ as the minimum ϵ such that f is an ϵ-optimal circulation.

Given a circulation f we denote by $f^{(i)}$ the circulation after i cancellations. Last time we stated the following theorems:

Theorem 17.2 $\epsilon(f^{(1)}) \leq \epsilon(f)$

Theorem 17.3 $\epsilon(f^{(m)}) \leq \left(1 - \frac{1}{n}\right) \epsilon(f)$

Theorem 17.4 If $\epsilon(f) < \frac{1}{n}$ then f is a minimum cost circulation.

Theorem 17.5 Let $C = \max_{i,j} |c_{ij}|$. Then the above algorithm terminates after at most $\mathcal{O}(mn \log nC)$ iterations. This gives the overall running time of $\mathcal{O}(m^2 n^2 \log nC)$.

Last time we showed that Theorem 17.5 follows from Theorems 17.2, 17.3, and 17.4, and gave a proof for Theorems 17.4 and 17.2. We now complete the proof of Theorem 17.5 by proving Theorem 17.3. Recall that last time we also proved the following.

Theorem 17.6 $\mu(f) = -\epsilon(f)$.

Proof of Theorem 17.3: We know there exist potentials p such that $c^p_{ij} \geq -\epsilon(f)$ for all $(i,j) \in A_f$. Suppose that in some iteration k we cancel cycle Γ such that $\exists(i,j) \in \Gamma$ with $c^p_{ij} \geq 0$ Then:

$$\begin{aligned} -\epsilon(f^{(k)}) = \mu(f^{(k)}) &= \frac{c^p(\Gamma)}{|\Gamma|} \\ &\geq \frac{|\Gamma|-1}{|\Gamma|}(-\epsilon(f)) \\ &\geq \left(1 - \frac{1}{n}\right)(-\epsilon(f)). \end{aligned}$$

Thus

$$\epsilon(f^{(k)}) \leq \left(1 - \frac{1}{n}\right)\epsilon(f).$$

How many consecutive iterations can there be the case that cycle Γ that is cancelled has $c^p_{ij} < 0$ for all $(i,j) \in \Gamma$? Cancelling the cycle removes one edge with $c^p_{ij} < 0$ from the residual graph and creates only edges with $c^p_{ij} \geq 0$. So we need no more than m iterations before we cancel such a cycle Γ. $\square$

17.1.2 Strongly polynomial time analysis

Definition 17.3 An algorithm runs in strongly polynomial time if the number of basic operations (e.g. additions, subtractions, multiplications, comparisons, etc.) can be bounded by a polynomial in the number of data items that were input and is not dependent on the size of data inputs (e.g. bits to encode cost, lower bounds, etc).

If an algorithm is strongly polynomial for minimum-cost circulations, its running time depends only on m and n. The first such algorithm is due to Professor Éva Tardos of Cornell in 1985. She won the Fulkerson Prize for it in 1988.

Definition 17.4 An arc $(i,j) \in A$ is ϵ-fixed if the flow on it is the same for all ϵ-optimal circulations f.

Before we begin discussing conditions under which an arc becomes ϵ-fixed, we give a lemma that we will need.

Lemma 17.7 For any circulation f, any $\emptyset \neq S \subseteq V$, we have that

$$\sum_{i \in S, j \notin S, (i,j) \in A} f_{kl} = 0$$

Proof: For any set S we know:

$$\sum_{i:(i,j) \in A} f_{ij} = 0 \Longrightarrow \sum_{j \in S} \sum_{i:(i,j) \in A} f_{ij} = 0$$

We also have the antisymmetry conditions:

$$f_{ij} + f_{ji} = 0, \forall (i,j) \in S$$

Combining the two, we conclude:

$$\sum_{\substack{i \in S \\ j \notin S \\ (i,j) \in A}} f_{ij} = 0$$

□

Theorem 17.8 Let $\epsilon > 0$, let f be a circulation, and let p be potentials such that f is ϵ-optimal with respect to p. If $|c^p_{ij}| \geq 2n\epsilon$ then (i,j) is ϵ-fixed.

Proof: Suppose that f' is an ϵ-optimal circulation such that $f'_{ij} \neq f_{ij}$. Assume that $c^p_{ij} \leq -2n\epsilon$; this is without loss of generality since costs are antisymmetric. The idea is that $f'_{ij} \neq f_{ij}$ will imply that there exists a cycle $\Gamma \in A_{f'}$ containing (i,j). The cost of (i,j) is so negative that

$$\frac{c(\Gamma)}{|\Gamma|} < -\epsilon,$$

contradicting the ϵ-optimality of f'.

First, we show that there exists a cycle Γ in $A_{f'}$ such that $(i,j) \in \Gamma$. Since $c^p_{ij} \leq -2n\epsilon$ we know that $(i,j) \notin A_f$ because of ϵ-optimality of f. Therefore $f_{ij} = u_{ij}$. Thus we must have $f'_{ij} < f_{ij} = u_{ij}$.

Let $E_< = \{(k,l) \in A : f'_{kl} < f_{kl}\}$. Observe that $E_< \subseteq A_{f'}$ since $f'_{kl} < f_{kl} \leq u_{kl}$. Let S be the set of nodes reachable from j in $E_<$. We will show that $i \in S$ therefore a cycle Γ exists as claimed. Note that $E_< \subseteq A_{f'}$. Suppose by contradiction that $i \notin S$. Lemma 17.7 tells us that

$$\sum_{\substack{k \in S \\ l \notin S}} f_{kl} = 0 \text{ and } \sum_{\substack{k \in S \\ l \notin S}} f'_{kl} = 0$$

These together imply that

$$\sum_{\substack{k \in S \\ l \notin S}} (f_{kl} - f'_{kl}) = 0$$

But $f'_{ij} < f_{ij} \Rightarrow f'_{ji} > f_{ji}$. Therefore there is a term in the sum that is negative. Then there must be a term that is positive. So $\exists (k,l), k \in S, l \notin S$ such that $f'_{kl} < f_{kl}$. By then $(k,l) \in E_<$ and since $k \in S$, it must be that $l \in S$, which is a contradiction.

Therefore we know that if $|c^p_{ij}| \geq 2n\epsilon$ then (i,j) is part of a cycle Γ in the set of edges (k,l) for which $f'_{kl} < f_{kl}$. Note that this implies that the reverse cycle $\Lambda = \{(l,k) : (k,l) \in \Gamma\}$ exists in the set of arcs (l,k) for which $f'_{lk} > f_{lk}$, which implies that Λ exists in A_f since flow on the edges in this cycle cannot be at their upper bounds. Since f is ϵ-optimal we know that for $(l,k) \in A_f, c^p_{lk} \geq -\epsilon$. Therefore for any $(k,l) \in \Gamma$ we know that $c^p_{kl} \leq \epsilon$.

We know that $\mu(f') = -\epsilon(f') \geq -\epsilon$. Thus:

$$\begin{aligned}
\frac{c(\Gamma)}{|\Gamma|} &= \frac{c^p(\Gamma)}{|\Gamma|} \\
&= \frac{1}{|\Gamma|} \left(c^p_{ij} + \sum_{(k,l) \in \Gamma : (k,l) \neq (i,j)} c^p_{kl} \right) \\
&\leq \frac{1}{|\Gamma|} (-2n\epsilon + (|\Gamma| - 1)\epsilon) \\
&< \frac{1}{|\Gamma|} (-|\Gamma|\epsilon) \\
&= -\epsilon
\end{aligned}$$

Therefore there exists a cycle in $A_{f'}$ whose mean cost is less than $-\epsilon$, which is a contradiction. Therefore the flow on the arc (i,j) must be fixed. □

Next time we will show that this analysis gives a strongly polynomial-time algorithm.

ORIE 633 Network Flows March 5, 2004

Lecture 18

Lecturer: David P. Williamson *Scribe: David P. Williamson*

18.1 Polynomial-time algorithms for minimum-cost circulations

18.1.1 Minimum mean-cost cycle cancelling algorithm (cont.)

Recall that last time we defined the notion of an ϵ-fixed arc.

Definition 18.1 A circulation f is ϵ-optimal if $\exists$ potentials p such that $c^p_{ij} \geq -\epsilon \; \forall (i,j) \in A_f$. Further, we define $\epsilon(f)$ as the minimum ϵ such that f is an ϵ-optimal circulation.

Definition 18.2 An arc $(i,j) \in A$ is ϵ-fixed if the flow on it is the same for all ϵ-optimal circulations f.

We then proved the following theorem.

Theorem 18.1 Let $\epsilon > 0$, let f be a circulation, and let p be potentials such that f is ϵ-optimal with respect to p. If $|c^p_{ij}| \geq 2n\epsilon$ then (i,j) is ϵ-fixed.

We now show that this leads to a strongly polynomial-time algorithm.

Theorem 18.2 The minimum mean-cost cycle cancelling algorithm terminates after $\mathcal{O}(m^2 n \log n)$ iterations.

Proof: Once an arc is fixed, it will always remain fixed since $\epsilon(f)$ is non-increasing. We now claim that a new arc will be fixed after at most $k = mn\log(2n)$ iterations. Let f be the current circulation and Γ be the cycle cancelled in this iteration. Then

$$\begin{aligned} \epsilon(f^{(k)}) &\leq \left(1 - \frac{1}{n}\right)^{n \log 2n} \epsilon(f) \\ &< e^{-\log 2n} \epsilon(f) \\ &= \frac{\epsilon(f)}{2n} \end{aligned}$$

Let p^k be the potentials associated with the flow $f^{(k)}$ such that the flow is $\epsilon(f^{(k)})$-optimal. Then

$$-\epsilon(f) = \frac{c^{p^k}(\Gamma)}{|\Gamma|} < -2n\epsilon(f^{(k)})$$

Therefore, $\exists (i,j) \in \Gamma$ such that $c_{ij}^{p^k} < -2n\epsilon(f^{(k)})$. Therefore (i,j) is fixed.

Further, note that (i,j) was not $\epsilon(f)$-fixed since $(i,j) \in \Gamma$ and the flow on it changed when we cancelled Γ. But if it was $\epsilon(f)$-fixed, the flow on it would not have changed. Therefore we fixed a new edge. □

18.1.2 A primal-dual algorithm

So far, the algorithm for the minimum-cost circulation problem that we have studied has been a primal algorithm. The algorithm starts with some feasible circulation and moves towards optimality. One could also consider a dual algorithm, which maintains a dual feasible solution, and moves towards optimality. Today we will start discussions of a special case of dual algorithms known (in combinatorial optimization) as primal-dual algorithms. They start with some dual feasible solution and a primal infeasible solution. The algorithm moves to reduce the infeasibility of the primal and increase the value of the dual while maintaining complimentary slackness.

To have a primal-dual method, we need first a primal and a dual. Let's go back to the original notation for the circulation problem in which we had lower bounds on the flows for each arc and didn't have the antisymmetry condition. A primal LP for the min-cost circulation problem is as follows.

$$\text{Min} \quad \sum_{(i,j)\in A} c_{ij} f_{ij}$$

subject to:

$$\sum_{k:(k,i)\in A} f_{ki} - \sum_{k:(i,k)\in A} f_{ik} = 0 \qquad \forall i \in V$$
$$l_{ij} \le f_{ij} \le u_{ij}.$$

We then take the dual of this LP to obtain the following:

$$\text{Max} \quad \sum_{(i,j)\in A} l_{ij} w_{ij} - \sum_{(i,j)\in A} u_{ij} z_{ij}$$

subject to:

$$p_j - p_i + w_{ij} - z_{ij} = c_{ij} \qquad \forall (i,j) \in A$$
$$w_{ij} \ge 0$$
$$z_{ij} \ge 0.$$

Now, suppose that the node potentials p are given. The reduced cost $c_{ij}^p = c_{ij} + p_i - p_j$, and $c_{ij}+p_i-p_j = w_{ij}-z_{ij}$ in the dual LP. If we know the potentials, then we can compute the best possible setting of the dual variables w and z (that is, the ones that will maximize the objective function) by setting $w_{ij} = \max(c_{ij}^p, 0) \equiv (c_{ij}^p)^+$ and $-z_{ij} = min(c_{ij}^p, 0) \equiv (c_{ij}^p)^-$.

Therefore, finding potentials p yields a solution to the dual and the following LP is equivalent to the dual LP:

$$\text{Max} \quad \sum_{(i,j)\in A} l_{ij}(c^p_{ij})^+ + \sum_{(i,j)\in A} u_{ij}(c^p_{ij})^-$$

subject to:

$$c_{ij} + p_i - p_j = c^p_{ij} \qquad \forall (i,j) \in A$$

Then by complementary slackness we have

$$\begin{aligned} c^p_{ij} > 0 \Leftrightarrow w_{ij} > 0 \Rightarrow f_{ij} = l_{ij} \\ c^p_{ij} < 0 \Leftrightarrow z_{ij} > 0 \Rightarrow f_{ij} = u_{ij} \end{aligned}$$

In general, the primal-dual method works as follows. We start with some dual feasible solution. We then check whether or not we can find a feasible primal solution that obeys the complementary slackness conditions with respect to the current dual. If so, then we have a feasible primal and feasible dual that obey complementary slackness with respect to each other, and thus must be optimal. If not, then we claim that we can find some way to modify the dual so that the dual objective function increases, and we repeat the step of checking for a feasible primal solution that obeys complementary slackness with respect to the current dual.

Our primal-dual algorithm for the minimum-cost circulation problem will start with a dual feasible solution by setting all potentials equal to 0. We will then determine whether there exists a primal feasible solution that obeys complimentary slackness by defining a new circulation problem with modified upper and lower bounds $\tilde{u}$ and $\tilde{l}$.

$$\begin{aligned} c^p_{ij} > 0 &\Rightarrow \tilde{l}_{ij} = \tilde{u}_{ij} = l_{ij} \\ c^p_{ij} < 0 &\Rightarrow \tilde{l}_{ij} = \tilde{u}_{ij} = u_{ij} \\ c^p_{ij} = 0 &\Rightarrow \tilde{l}_{ij} = l_{ij}, \tilde{u}_{ij} = u_{ij} \end{aligned}$$

As with most primal dual approaches, we have reduced a problem with cost to a problem without cost where we only need to check for feasibility. If we can find a feasible circulation in the problem with bounds $\tilde{l}_{ij}$ and $\tilde{u}_{ij}$, we are finished, since then we will have a primal feasible solution and a dual feasible solution that obey the complementary slackness conditions, and thus are optimal.

If not, then by Hoffman's circulation theorem (on Problem Set 1) we can find a cut S such that $\tilde{l}(\delta^+(S)) > \tilde{u}(\delta^-(S))$. We also showed on Problem Set 1 that we could check for feasibility and find such an S with one maximum flow computation. We will modify the dual to increase the dual objective function. To do this, we will increase the potentials of nodes in the cut S by a value β. We will show next time that this will lead to an increase in the dual objective function.

ORIE 633 Network Flows March 8, 2004

Lecture 19

Lecturer: David P. Williamson *Scribe: Stefan Wild*

19.1 Polynomial-time algorithms for minimum-cost circulations

19.1.1 A primal-dual algorithm (cont.)

From last lecture, recall the new form of the dual for a min-cost circulation:

$$\text{Max} \quad \sum_{(i,j)\in A} l_{ij}(c^p_{ij})^+ + \sum_{(i,j)\in A} u_{ij}(c^p_{ij})^-$$

subject to:

$$c_{ij} + p_i - p_j = c^p_{ij} \qquad \forall (ij) \in A,$$

where $(c^p_{ij})^+ \equiv \max(c^p_{ij}, 0)$ and $(c^p_{ij})^- \equiv min(c^p_{ij}, 0)$. The corresponding complementary slackness conditions are:

$$c^p_{ij} > 0 \Leftrightarrow w_{ij} > 0 \Rightarrow x_{ij} = l_{ij},$$
$$c^p_{ij} < 0 \Leftrightarrow z_{ij} > 0 \Rightarrow x_{ij} = u_{ij}.$$

We then had the following algorithm:

Primal-Dual Algorithm

```
Find feasible dual ($p \leftarrow 0$)
While current primal doesn't obey complementary slackness conditions
with respect to the current dual,
    Get a direction of dual increase and update.
```

Our primal-dual algorithm then starts with a dual feasible solution and we must determine whether there exists a primal feasible solution that obeys complementary slackness by defining a new circulation problem with modified upper and lower bounds $\tilde{u}$ and $\tilde{l}$.

$$c^p_{ij} > 0 \Rightarrow \tilde{l}_{ij} = \tilde{u}_{ij} = l_{ij}$$
$$c^p_{ij} < 0 \Rightarrow \tilde{l}_{ij} = \tilde{u}_{ij} = u_{ij}$$
$$c^p_{ij} = 0 \Rightarrow \tilde{l}_{ij} = l_{ij}, \tilde{u}_{ij} = u_{ij}$$

If we can find a feasible circulation in the problem with bounds $\tilde{l}_{ij}$ and $\tilde{u}_{ij}$, we are finished, since then we will have a primal feasible solution and a dual feasible solution that obey the complementary slackness conditions, and thus are optimal.

If not, by Problem Set 1, we can find a cut $S \subset V$ such that $\tilde{l}(\delta^+(S)) > \tilde{u}(\delta^-(S))$. Note that by Problem Set 1, we could either find a feasible solution to the circulation problem or find such a cut S in a single max flow computation.

We will now use this set S to prove the following lemma.

Lemma 19.1 If there is no feasible circulation with respect to bounds $\tilde{l}, \tilde{u}$, then we can increase the dual objective function.

Proof: We want to adjust the reduced cost c^p_{ij} so that the c^p_{ij} don't flip sign for all $(i,j) \in A$. We do this by increasing p_i by β for all $i \in S$, where S is the cut that has $\tilde{l}(\delta^+(S)) > \tilde{u}(\delta^-(S))$. We then have:

$$c^p_{ij} = \begin{cases} c^p_{ij} + \beta & \text{if } (i,j) \in \delta^+(S) \\ c^p_{ij} - \beta & \text{if } (i,j) \in \delta^-(S) \\ c^p_{ij} & \text{otherwise.} \end{cases}$$

So in order to preserve the signs of c^p_{ij}, we set:

$$\beta = \min\left(\min\left\{|c^p_{ij}| : (i,j) \in \delta^+(S),\, c^p_{ij} < 0\right\}, \min\left\{c^p_{ij} : (i,j) \in \delta^-(S),\, c^p_{ij} > 0\right\}\right),$$

where this definition implies that $\beta > 0$.

We now consider the change, Δ, in the dual objective function:

$$\Delta = \beta \left(\sum_{\substack{(i,j)\in\delta^+(S) \\ c^p_{ij} \geq 0}} l_{ij} - \sum_{\substack{(i,j)\in\delta^-(S) \\ c^p_{ij} > 0}} l_{ij} + \sum_{\substack{(i,j)\in\delta^+(S) \\ c^p_{ij} < 0}} u_{ij} - \sum_{\substack{(i,j)\in\delta^-(S) \\ c^p_{ij} \leq 0}} u_{ij} \right).$$

Now observe that

$$\sum_{\substack{(i,j)\in\delta^+(S) \\ c^p_{ij} \geq 0}} l_{ij} + \sum_{\substack{(i,j)\in\delta^+(S) \\ c^p_{ij} < 0}} u_{ij} = \sum_{(i,j)\in\delta^+(S)} \tilde{l}_{ij} = \tilde{l}(\delta^+(S)),$$

and

$$\sum_{\substack{(i,j)\in\delta^-(S) \\ c^p_{ij} > 0}} l_{ij} + \sum_{\substack{(i,j)\in\delta^-(S) \\ c^p_{ij} \leq 0}} u_{ij} = \sum_{(i,j)\in\delta^-(S)} \tilde{u}_{ij} = \tilde{u}(\delta^-(S)).$$

Therefore $\Delta = \beta\left(\tilde{l}(\delta^+(S)) - \tilde{u}_{ij}\right) > 0$ and hence we have a dual objective function increase.

□

Since the costs and initial potentials are integral, we have that β is integral and so the next potentials will remain integral. Furthermore, if the bounds l and u are integral, then the dual objective function increase will also be integral. This will give a pseudo-polynomial time algorithm that requires $\mathcal{O}(mCU)$ max-flow computations, where m is the number of arcs, C is the value of the largest (in magnitude) edge cost, and U is the value of the largest capacity. In fact, clever analysis will give an algorithm that requires $\mathcal{O}(\min(nC, nU))$ max-flow computations.

19.1.2 A cost scaling algorithm

We now turn to another non-primal algorithm for the minimum-cost circulation problem.

Cost Scaling (Goldberg, Targan '90)

```
Let $f$ be any feasible circulation
Initialize $\epsilon \leftarrow C$, $p_i \leftarrow 0 \ \forall i \in V$
while $\epsilon \geq \frac{1}{n}$
    $(\star)$
    $\epsilon \leftarrow \frac{\epsilon}{2}$
    $(f, p) \leftarrow$ Run Subroutine: find $\epsilon$-optimal circulation given input $(f, \epsilon, p)$
```

The idea is that given a 2ϵ-optimal circulation f with respect to potentials p, the subroutine will find an ϵ-optimal circulation f' with respect to potentials p'. Since the initial circulation is C-optimal and the final f is $< \frac{1}{n}$-optimal (and hence optimal by the proof in the previous lecture), we will require $\log(nC)$ iterations of the while loop.

We can also show that the number of iterations is strongly polynomial by tweaking one of our previous theorems. Recall the following definition and result.

Definition 19.1 An arc (i, j) is ϵ-*fixed* if the flow on (i, j) is the same for all ϵ-optimal circulations.

Theorem 19.2 For $\epsilon > 0$ and circulation f with respect to potentials p, if $|c_{ij}^p| \geq 2n\epsilon$, then (i, j) is ϵ-fixed.

We then have the following theorem:

Theorem 19.3 For circulation f and $\epsilon' < \frac{\epsilon(f)}{2n}$, the set of ϵ'-fixed arcs strictly contains the set of $\epsilon(f)$-fixed arcs.

Proof: Clearly if an arc is ϵ'-fixed, then it is also $\epsilon(f)$-fixed. We now want to show that there exists an arc that is ϵ'-fixed, but not $\epsilon(f)$-fixed. Let p be the potentials such that f is $\epsilon(f)$-optimal. Then there exists a cycle $\Gamma \in A_f$ such that $-\epsilon(f) = \frac{c^p(\Gamma)}{|\Gamma|}$ by a previous theorem. We also know that $c_{ij}^p \geq -\epsilon(f) \, \forall (i, j) \in A_f$ by definition. Hence $c_{ij}^p = -\epsilon(f) \, \forall (i, j) \in \Gamma$.

If we cancel cycle Γ, the resulting circulation, $\hat{f}$, is still ϵ-optimal. Thus no arc in Γ is ϵ-fixed. Now let f' be any ϵ'-optimal circulation with respect to potentials p'. Then $-\epsilon(f) = \frac{c^{p'}(\Gamma)}{|\Gamma|} < -2n\epsilon'$ and thus $\exists (i, j) \in \Gamma$ such that $c_{ij}^{p'} \leq -2n\epsilon'$. Therefore (i, j) is ϵ'-fixed (but not $\epsilon(f)$-fixed). □

We now want to claim the following corollary.

Corollary 19.4 Every $\log(2n)$ iterations of the while loop, a new arc is fixed.

But note that the lemma states that ϵ' must be a factor of $2n$ less than $\epsilon(f)$, not just any ϵ such that f is ϵ-optimal. In order to make this true, at step $(\star)$ in the Cost Scaling algorithm, we must now add a subroutine to find potentials p such that f is $\epsilon(f)$-optimal and then set $\epsilon \leftarrow \epsilon(f)$. This will only decrease ϵ as the procedure continues. Then we can claim the corollary above.

Since we can fix at most m arcs, we have the following theorem.

Theorem 19.5 After $\min(m \log(2n), \log(nC))$ iterations, Cost Scaling finds a min-cost circulation.

ORIE 633 Network Flows March 10, 2004

Lecture 20

Lecturer: David P. Williamson *Scribe: Patrick Kongsilp*

20.1 Polynomial-time algorithms for minimum-cost circulations

20.1.1 A cost-scaling algorithm (cont.)

In the last lecture, the cost scaling algorithm was introduced but the subroutine find-ϵ-opt-circ(f, ϵ, p) wasn't given.

Cost Scaling (Goldberg, Tarjan '90)

```
Let $f$ be any feasible arc
    $\epsilon \leftarrow C$
    $p_i \leftarrow 0, \forall i \in V$
    While $\epsilon \geq 1/n$
        $\epsilon \leftarrow \epsilon/2$
        $(f, p) \leftarrow$ find-$\epsilon$-opt-circ$(f, \epsilon, p)$
```

Recall the following theorem from last lecture.

Theorem 20.1 After $\min(\log(nC), m\log(2n))$ iterations, cost scaling finds an optimal circulation.

Today, we'll given an algorithm for the subroutine find-ϵ-opt-circ(f, ϵ, p) based on the ideas from the push/relabel algorithm that we saw for the maximum flow problem.

find-ϵ-opt-circ

- **Input:** 2ϵ-opt circulation f, potentials p s.t. $c^p_{ij} \geq -2\epsilon, \forall (i,j) \in A_f$
- **Goal:** 2ϵ-opt circulation f', potentials p' s.t. $c^p_{ij} \geq -\epsilon, \forall (i,j) \in A_{f'}$

The basic idea of the algorithm is that we will first convert the 2ϵ-optimal circulation to an ϵ-optimal *pseudoflow*, and then convert the ϵ-optimal pseudoflow to an ϵ-optimal circulation.

Definition 20.1 A *pseudoflow* $f : A \to \mathbb{R}$ satisfies the following:

- $f_{ij} = -f_{ji}$, for all $(i,j) \in A$

- $f_{ij} \leq u_{ij}$, for all $(i, j) \in A$.

Note that a pseudoflow obeys antisymmetry and capacity constraints but not flow conservation.

Definition 20.2 For pseudoflow f, the *excess* at node $i \in V$ is

$$e_i^f = \sum_{k:(k,i) \in A} f_{ki}$$

Note that this quality may be negative. If so, then negative excess is sometimes called a *deficit.*

How can we convert a 2ϵ-optimal circulation to an ϵ-optimal pseudoflow? It's easy; we just saturate every edge with negative cost. That is, for $(i, j) \in A$ such that $c_{ij}^p < 0$, set f_{ij} to u_{ij}. Then f is a 0-optimal pseudoflow.

To use a push/relabel scheme, we need to specify the conditions needed (and actions taken) for doing a push operation and a relabel operation. Obviously, in order to get from a pseudoflow to a circulation, we'd like to get rid of all excesses; following the idea of the push/relabel algorithm for maximum flow, we'll do a push on nodes with positive excess. Recall that in the maximum flow case, we only pushed along *admissible* arcs that met some criterion with their distance label. What should be the concept of an admissible arc in this case? Here we say an arc (i, j) is admissible if $c_{ij}^p < 0$. Thus we push from node i with $e_i^f > 0$ if there exists j such that $u_{ij}^f > 0$ and $c_{ij}^p < 0$. As in the maximum flow case, we will push $\delta = \min(e_i^f, u_{ij}^f)$ units of flow along (i, j).

Observe that ϵ-optimality is maintained during a push operation on (i, j) since if (j, i) is created in the residual graph, it will have reduced cost $c_{ji}^p = -c_{ij}^p > 0$.

What happens during a relabel operation? We need to relabel if there is excess at a node i, but there are no admissible arcs leaving i. In this case, all arcs with residual capacity must have non-negative reduced cost. To create some admissible arc, we will simply alter the potential p_i at node i. In particular, we set

$$p_i \leftarrow \max_{(i,j) \in A_f} (p_j - c_{ij} - \epsilon).$$

Note that after a relabel operation, we have

- $c_{ij} + p_i - p_j \geq -\epsilon, \forall (i, j) \in A_f$
- $c_{ij} + p_i - p_j = -\epsilon$ for some $(i, j) \in A_f$

Therefore, p_i is decreased by at least ϵ, and f maintains ϵ-optimality.

Putting these together, we obtain the following algorithm.

```
**Push/relabel find-$\epsilon$-opt-circ$(f, \epsilon, p)$**

    $\forall (i,j) \in A_f$ if $c^p_{ij} < 0, f_{ij} \leftarrow u_{ij}$
    While $\exists$ active $i \in V$ $(e^f_i > 0)$
        If $\exists j$ s.t. $u^f_{ij} > 0$ and $c^p_{ij} < 0$
            Push $\delta = \min(e^f_i, u^f_{ij})$ flow on $(i,j)$
        Else
            Relabel $p_i \leftarrow \max_{(i,j) \in A_f} (p_j - c_{ij} - \epsilon)$
    Return $(f, p)$
```

We now want to show that the algorithm is correct and bound its running time. Recall the following lemma from previous lectures.

Lemma 20.2 For any circulation f, any $S \subseteq V, S \neq \emptyset$,

$$\sum_{i \in S, j \notin S, (i,j) \in A} f_{ij} = 0.$$

We will need the following lemma for our proof.

Lemma 20.3 Let f be a pseudoflow, f' a circulation. For any i such that $e^f_i > 0$, there exists j such that $e^f_j < 0$ and there exists a path P from i to j with $(k,l) \in A_f, (l,k) \in A_{f'}$ for all $(k,l) \in P$.

Proof: We first claim that we can find P in set of arcs

$$A_< = \{(i,j) : f_{ij} < f'_{ij}\}$$

Note $A_< \subseteq A_f$ since $f_{ij} < f'_{ij}$ implies $f_{ij} < u_{ij}$. Further note that if $(i,j) \in A_<$, then $(j,i) \in A_{f'}$ since then $f'_{ji} < f_{ji} \leq u_{ji}$. Thus given a vertex i such that $e^f_i > 0$, it will be sufficient to find a path in $A_<$ to some j such that $e^f_j < 0$.

To do this, let S be all vertices reachable from i using arcs in $A_<$. Then,

$$\begin{aligned}
\sum_{k \in S} e^f_k &= \sum_{k \in S} \sum_{j:(k,j) \in A} f_{kj} \\
&= \sum_{k \in S, j \notin S, (k,j) \in A} f_{kj} \\
&\geq \sum_{k \in S, j \notin S, (k,j) \in A} f'_{kj} = 0.
\end{aligned}$$

The inequality holds because $(k,j) \notin A_<$. The last equality holds because f' is a circulation.

Since $e^f_i > 0$, then there must be $j \in S$ such that $e^f_j < 0$. Furthermore, j is reachable from i using arcs of $A_<$. □

Using the lemma above, we can now bound the amount that the potential of any node changes during the course of algorithm.

Lemma 20.4 For any i, p_i decreases by at most $3n\epsilon$ during the algorithm.

Proof: Let f' be the initial 2ϵ-optimal circulation, and p' initial potentials. We consider the last point in the algorithm during which p_i is relabelled. Note that if p_i is relabelled, then $e_i^f > 0$. By Lemma 20.3, we know there is $j \in V$ such that $e_j^f < 0$ and there is a path P from i to j in A_f, with the reverse of the path in $A_{f'}$.

First, observe that f being ϵ-optimal implies

$$-|P|\epsilon \leq \sum_{(k,l)\in P} c_{kl}^p = \sum_{(k,l)\in P} (c_{kl} + p_k - p_l) = \left(\sum_{(k,l)\in P} c_{kl}\right) + p_i - p_j$$

Next, observe that since f' is 2ϵ-optimal and the reverse of P from j to i is in $A_{f'}$ implies that

$$-2\epsilon|P| \leq \sum_{(k,l)\in P} c_{lk}^{p'} = \sum_{(k,l)\in P} c_{lk} + p'_j - p'_i$$

Finally, observe that by our definition of costs $\sum_{(k,l)\in P} c_{kl} = \sum_{(k,l)\in P} c_{lk}$. Thus by adding the previous inequalities, we get

$$-3\epsilon|P| \leq (p_i - p'_i) + (p'_j - p_j).$$

Because $e_j^f < 0$, the node j must not have been relabelled to this point in the algorithm, and thus $p_j = p'_j$. Therefore we have that

$$-3n\epsilon \leq p_i - p'_i.$$

Since we assumed that this was the last point in the algorithm during which i was relabelled, the lemma statement follows. $\square$

The corollary below follows immediately from the fact that every relabelling decreases the potential at a node by at least ϵ.

Corollary 20.5 The number of relabels per vertex is at most $3n$, and there are at most $3n^2$ relabels total.

As with the push/relabel algorithm for maximum flows, the bounds on the remaining push operations follow almost immediately from this; we will discuss this next time.

ORIE 633 Network Flows March 12, 2004

Lecture 21

Lecturer: David P. Williamson *Scribe: Ed Hua*

21.1 Polynomial-time algorithms for minimum-cost circulations

21.1.1 A cost-scaling algorithm (cont.)

Recall from the previous lectures that we had shown that $\min(m \log(2n), \log(nC))$ iterations of a subroutine `find-ε-opt-circ` was sufficient for computing a minimum-cost circulation. This subroutine takes a 2ϵ-optimal circulation and computes from it an ϵ-optimal circulation. Last time we gave an implementation of the subroutine based on ideas from the push/relabel algorithm for the maximum flow problem. We repeat the algorithm below.

Push/Relabel find-ε-opt-circ(f,ε,p)

```
Input: $2\varepsilon$–optimal circulation $f$, pricess $p$
Output: $\varepsilon$–optimal circulation $f'$, prices $p'$
$\forall (i,j) \in A$ If $C^p_{ij} < 0, f_{ij} \leftarrow u_{ij}$
while $\exists$ active $i \in V$, $(e^f_i > 0)$
    If $\exists j$ such that $u^f_{ij} > 0$ and $c^p_{ij} < 0$
        Push $\delta = \min\left(e^f_i, u^f_{ij}\right)$ flow on $(i,j)$
    Else
        Relabel $p_i \leftarrow \max_{(i,j)\in A_f}(p_j - c_{ij} - \varepsilon)$
Return $(f,p)$
```

In dealing with maximum flow problems, we said that an arc (i,j) is admissible if $d_i = d_j + 1$. For minimum-cost circulation problems, we gave the following definition.

Definition 21.1 In a minimum-cost circulation problem, for some set of prices $p_i, \forall i \in V$, an arc (i,j) is admissible if $c^p_{ij} < 0$.

Last time we also proved the following lemma.

Lemma 21.1 For all $i \in V$, p_i decreases by at most $3n\varepsilon$ during the algorithm.

We now give the following corollary.

Corollary 21.2 The total number of relabels is at most $3n^2$.

Proof: Since p_i decreases by at least ε in each relabel operations, there can be at most $3n$ relabels of i. This implies that there are at most $3n^2$ relabel operations in total. □

Recall that a *Push* operation is said to be *saturating* if $\delta = u^f_{ij}$, or *non-saturating* otherwise (in which case $\delta = e^f_i$). As in the case of the push/relabel algorithm for the maximum flow problem, we now bound the number of push operations by considering the two types of pushes separately.

Lemma 21.3 The number of saturating pushes in the above algorithm is at most $3nm$.

Proof: Pick any arc (i,j). Initially, $c^p_{ij} \geq 0$ if $u^f_{ij} > 0$. Therefore, we have to relabel i before we can push on (i,j), since for (i,j) to be admissible, we need $c^p_{ij} < 0$. Having had a saturating push on (i,j), in order to push flow again on it, we must first push flow back on (j,i), which implies $c^p_{ji} < 0$, which in turn implies $c^p_{ij} \geq 0$. Therefore, we need to relabel i once more to push flow on (i,j) again. This leads directly to a bound of at most $3n$ saturating pushes on (i,j). Thus for all m arcs in the graph, there can be at most $3nm$ saturating pushes. □

Now we wish to find an upper bound for the total number of non-saturating pushes in this algorithm. We need the following lemma to help us with this bound.

Lemma 21.4 The set of admissible arcs is acyclic.

Proof: We prove this lemma by induction on the algorithm. The base case of the algorithm is simple since initially no admissible arcs exist. Now suppose that the claim holds in the middle of the algorithm. Each time a *push* is executed, it can only remove admissible arcs from the residual graph, but cannot add them, so the claim holds. Each time a *relabel* is executed, it adds admissible outgoing arcs of vertex i, but removes all of i's admissible incoming arcs because all of the reduced costs of the arcs entering i are increased by at least ϵ. Thus no cycles can be created by the new admissible arcs coming out of vertex i. □

Now we can bound the number of non-saturating pushes.

Lemma 21.5 The number of non-saturating pushes in the algorithm is $O(n^2 m)$.

Proof: Define Φ_i to be the number of vertices reachable from i via the admissible arcs, and let $\Phi = \sum_{active\ i} \Phi_i$. Initially $\Phi \leq n$ (since every vertex can reach only itself); when the algorithm terminates, $\Phi = 0$, since there are no active vertices i.

What makes Φ increase? A *saturating push* on the arc (i,j) could result in a new active node j, and therefore Φ can increase by at most n. In addition, a *relabel* can increase Φ_i by at most n, but for a vertex j such that $j \neq i$, the relabel does not increase Φ_j, since all arcs entering i are no longer admissible. So, the amount that Φ increases is at most $n\left(3nm + 3n^2\right)$.

What then makes Φ decrease? From the algorithm above, we see that a *non-saturating push* decreases Φ by at least 1: after such a push, i has turned inactive, and even if some other vertex j became active as a result of the non-saturating push, it would still reach fewer vertices than i by the acyclicity of the admissible arcs.

So, the total number of non–saturating pushes in this algorithm has to be at most $n\left(3nm + 3n^2\right) = 3n^2m + 3n^3 = O\left(n^2m\right)$. $\square$

From the above lemmas, we see that the total number of push/relabel operations of the algorithm is at most $O(n^2m)$. Given an implementation with $O(1)$ time per operation (which we will not discuss), we may obtain the overall computational time of the *Push/Relabel find-ε-opt-circ* subroutine:

Theorem 21.6 The *Push/Relabel find-ε-opt-circ* subroutine takes $O(n^2m)$ time. Furthermore, with a FIFO implementation of $Push/Relabel$, the subroutine runs in $O\left(n^3\right)$ time.

Combining this with the bound on the number of iterations of the cost-scaling algorithm, we obtain the following.

Theorem 21.7 (Goldberg, Tarjan '90) The cost-scaling algorithm for the minimum-cost circulation problem can be implemented in $O(n^3 \min(\log(nC), m\log n))$ time.

Note that if we replace *Push/Relabel find-ε-opt-circ* with a subroutine based on blocking flows, the cost-scaling algorithm can be shown to run in $O(mn\log n \cdot \min\left(m\log n, \log(nC)\right))$ time.

We close our performance analysis of the cost-scaling algorithm with two open questions. First, is a minimum-cost circulation problem solvable with $O(\min(m\log n, \log(nC)))$ iterations of any maximum flow algorithm? It looked like the push/relabel algorithm could be used as the *find-ε-opt-circ* subroutine with only minor modifications; this is also the case for the blocking flow variant of this subroutine.

More to the point, can the Goldberg-Rao maximum flow algorithm be used for this subroutine? This would then give us a minimum-cost circulation algorithm that runs in $O(\Lambda m\log n(\log(mU))(\log(nC)))$ time, which would be the fastest known algorithm.

To these questions we have no definitive answers. The current best strongly polynomial time bound is due to Orlin, whose minimum-cost circulation algorithm runs in $O(m\log n \cdot (m + n\log n))$ time.

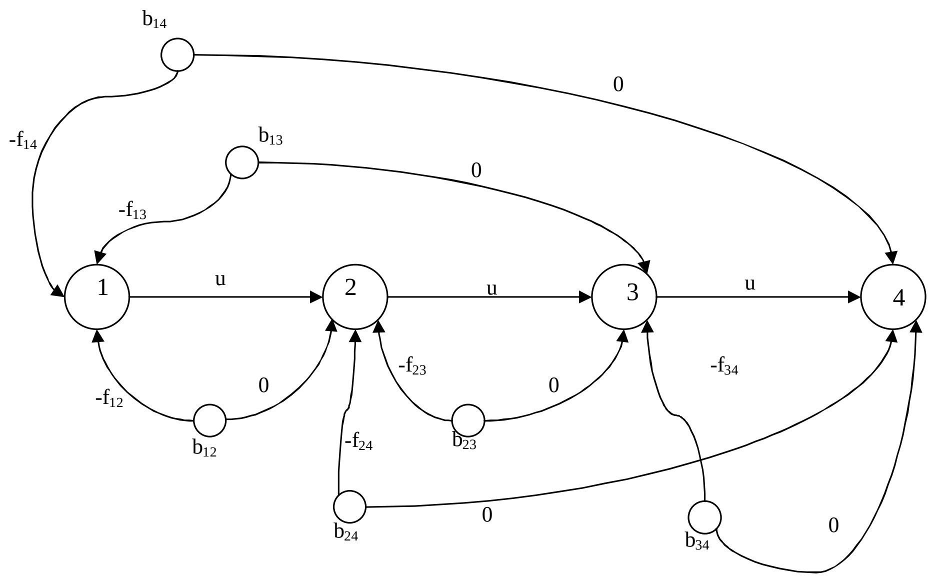
21.2 An application of minimum-cost flows

We look at a problem on the optimal loading of a hopping aircraft. Consider an airplane that makes stops at LaGuardia, Elmira, Ithaca, and Rochester, at each stop picking up some passengers. Our goal is to maximize the revenue while obeying the airplane's seating capacity constraints.

To solve this problem we first make some definitions as follows:

- b_{ij} = number of passengers who want to travel from i to j;
- f_{ij} = the fare for passengers travelling from i to j
- u = the capacity of the airplane

The figure below shows how we may transform this problem into a minimum-cost flow problem. In the figure, Vertex 1 is LaGuardia, Vertex 2 Elmira, Vertex 3 Ithaca, and Vertex 4 Rochester. We create arcs from vertex i to vertex $i+1$ of capacity u for each i; this corresponds to the capacity of the aircraft. For each value b_{ij}, we create a node with a supply b_{ij} at the node; each node has two arcs, one pointing to node i and one pointing to node j. The cost of the arc to node i is $-f_{ij}$; this corresponds to the revenue we get for each passenger traveling from i to j. The cost of the arc to node j is zero; we get no revenue from passengers we do not transport to node j in the aircraft. We put demands at each of the four city nodes corresponding to the number of passengers who want to end up there; that is $b_1 = 0$; $b_2 = -b_{12}$; $b_3 = -b_{13} - b_{23}$; and $b_4 = -b_{14} - b_{24} - b_{34}$.



b14
0
-f14
b13
0
-f13
1
u
2
u
3
u
4
-f23
-f34
0
0
-f12
b12
-f24
b23
0
b24
b34
0

ORIE 633 Network Flows March 15, 2004

Lecture 22

Lecturer: David P. Williamson *Scribe: Ed Hua*

22.1 The market-clearing pricing problem

In this lecture, we consider a problem from economics: that of finding prices that will cause a market to clear. We refer to this problem as the Market-Clearing Pricing Problem. This turns out to be a nice application of maximum flow techniques.

Market-Clearing Pricing Problem

- **Input:**
  - Set B of buyers
  - Set A of unit amounts of divisible goods ($|A| = n$)
  - Integer amount of money m_i $\forall i \in B$
  - Integer utilities u_{ij} $\forall i \in B$, $\forall j \in A$
    (utility u_{ij} specifies the happiness buyer i derives from one unit of good j)
- **Goal:** Find prices p_j $\forall j \in A$ such that the *market clears*:
  - All buyers buy only goods that maximize happiness
  - All money is spent
  - No goods remain unpurchased

It has been long known that prices exist that clear the market. A result of Arrow and Debreu from 1954 implies the existence of market-clearing prices, though this may not be earliest work that establishes the existence of such prices. The previous proofs that market-clearing prices exist, however, were non-constructive.

The Market-Clearing Pricing Problem was defined in 1891 by Fisher, who invented a hydraulic machine to solve it (in the case of three goods). Recently, in 2002, a polynomial-time algorithm was given for the problem, demonstrating that there still exist nice problems, which are solvable in polynomial time, for which no polynomial-time algorithm was previously known. We present this algorithm for computing market-clearing prices, which was developed by Devanur, Papadimitriou, Saberi, and Vazirani.

22.1.1 Characterizing market clearance using maximum flow

First, we formalize the notion that all the buyers must buy only goods that maximize their happiness in order for the market to clear. Given prices p_j, the "bang per buck" that a buyer i derives from a good j is the ratio of the utility u_{ij} to the price p_j. Figure 22.1(a) depicts sample data and the corresponding bang per buck ratios. Buyers try to maximize the bang per buck they get for the goods that they buy, and so we define α_i as follows to represent the best bang per buck that a buyer i can obtain.

$$\alpha_i = \max_{j \in A} \frac{u_{ij}}{p_j}$$

A buyer i will only buy goods j such that $\frac{u_{ij}}{p_j} = \alpha_i$. We define a graph that represents the goods that each buyer may purchase.

Definition 22.1 The *equality subgraph* $G = (A, B, E)$ is a bipartite graph (with vertex sets A and B) where $(i, j) \in E$ if and only if $\alpha_i = \frac{u_{ij}}{p_j}$.

Given a particular set of prices $p_j\ \forall j \in A$, we can determine whether the prices clear the market by performing a maximum flow computation. We add a source vertex s and a sink vertex t to the equality subgraph. For each good $j \in A$, we add an arc (s, j) with capacity p_j. For each buyer $i \in B$, we add an arc (i, t) with capacity m_i. We orient each edge (i, j) corresponding to a buyer $i \in B$ and a good $j \in A$ in the equality subgraph as a directed arc (j, i) with capacity ∞. Figure 22.1(b) shows an example of this graph for a particular collection of buyers and goods.

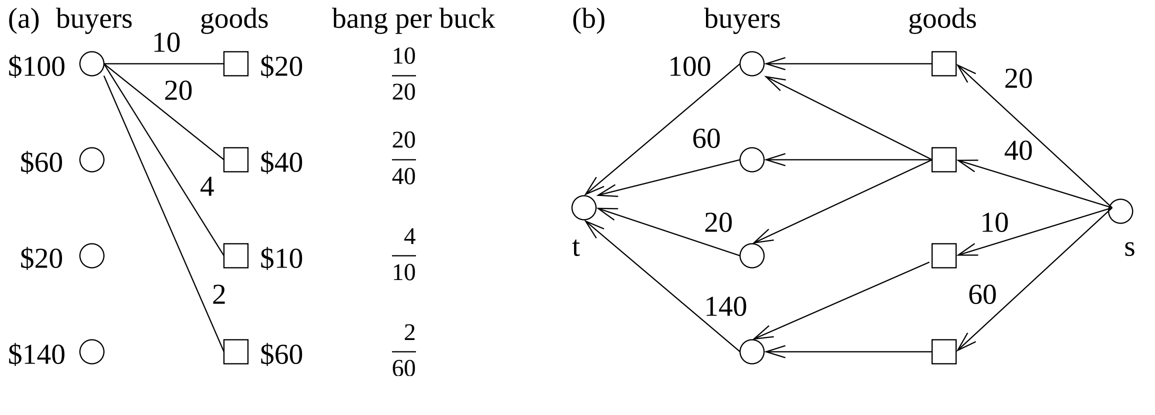


Figure 22.1: (a) An example of the computation of the bang per buck that a buyer obtains from different goods. The amounts of money the buyers have are shown on the left, the prices of the goods are shown on the right, and the label for an edge (i, j) indicates the utility u_{ij}. (b) A graph in which we can compute a maximum flow to determine whether a set of prices clears a market. The arcs from goods to buyers have infinite capacity.

In this graph, flow from the source to the sink represents the transfer of money in the market. A unit of flow on an arc (j, i) from a good j to a buyer i represents a dollar spent by buyer i on good j. The total amount of flow from the source to the sink is the total amount of money spent by the buyers on goods. Therefore, the market clears (the buyers spend all their money) if and only if the maximum flow value is $\sum_{i \in B} m_i$.

22.1.2 A polynomial-time algorithm

The idea behind this algorithm for the Market-Clearing Pricing Problem is to start with small prices, and to raise the prices over the course of the execution of the algorithm. We will keep the prices sufficiently low to ensure that all the goods are sold, but the buyers have left-over money (a surplus). We will maintain the invariant that the singleton set $\{s\}$ is a minimum s-t cut; this corresponds to all goods being sold. The goal will be to find prices such that $V - \{t\}$ is also a minimum s-t cut, because the capacity of the arcs crossing this cut is the total amount of money the buyers have. When this cut becomes a minimum s-t cut, the value of the maximum flow is $\sum_{i \in B} m_i$, and the market clears. We raise the prices gradually, decreasing the surplus of the buyers until it reaches zero.

Initialization of prices

We want to assign small initial values to the prices to ensure that $\{s\}$ is a minimum s-t cut. To initialize the prices, we set $p_j = \frac{1}{n} \ \forall j \in A$. Under these prices, $\{s\}$ is a minimum s-t cut with value 1. We also need at least one buyer for each good. If there are no buyers for good j, we compute $\alpha_i = \max_{j \in A} \frac{u_{ij}}{p_j}$ for all buyers i. Then, we reduce the price p_j to the value $\max_{i \in B} \frac{u_{ij}}{\alpha_i}$.

Raising prices

When we raise the prices to decrease the surplus of the buyers, we would like to ensure that all edges remain in the equality subgraph. Consider a buyer i for which the edges (i, j) and (i, k) are both in the equality subgraph. By the definition of the equality subgraph, we have $\frac{u_{ij}}{p_j} = \frac{u_{ik}}{p_k}$, which implies that $\frac{p_k}{p_j} = \frac{u_{ik}}{u_{ij}}$. Multiplying both p_j and p_k by the same factor will leave this ratio unchanged. As such, we increase the prices from p_j to p'_j by setting $p'_j = p_j x \ \ \forall j \in A$ for some factor x.

To determine the factor x that we will use to raise the prices, we consider the different ways in which the equality subgraph may change when we raise the prices.

- **Event type (1)**: By increasing x, the invariant that $\{s\}$ is a minimum s-t cut becomes violated.

  In the previous example, multiplying the prices by the factor $x = 2$ causes another minimum s-t cut to emerge, as shown in Figure 22.2(a). If we multiply the prices by a factor $x > 2$, then we violate the invariant, because $\{s\}$ is no longer a minimum s-t cut.

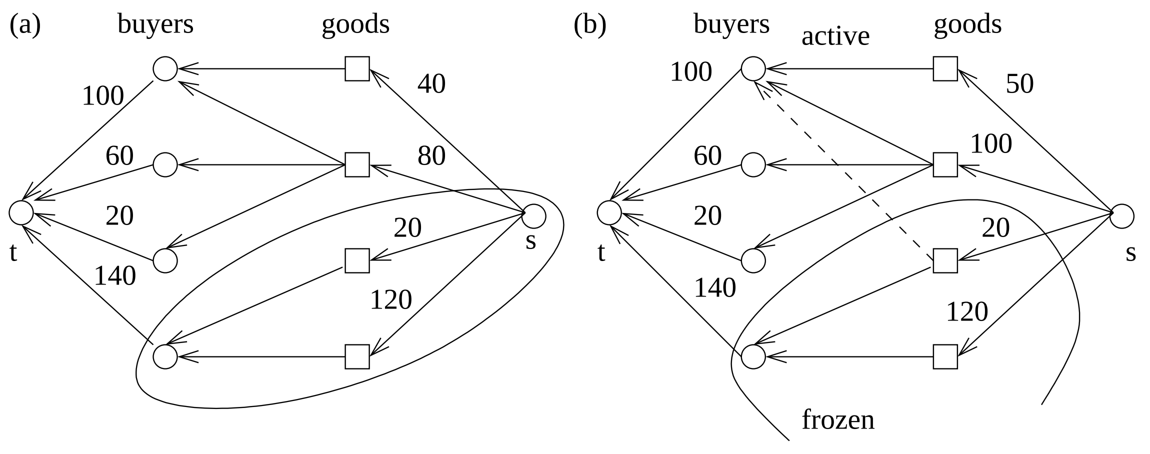


Figure 22.2: (a) An example of event type (1). If the prices are multiplied by a factor $x > 2$, then the cut shown becomes the minimum *s*-*t* cut. (b) An example of event type (2). Multiplying the prices of the active goods in (a) by a factor $x = 1.25$ causes the dashed edge shown between an active buyer and a frozen good to enter the equality subgraph.

Note that in the example, the emergence of the new minimum *s*-*t* cut when the prices are raised creates a desirable scenario for the last buyer, because all the money available to that buyer can be spent on goods. In general, the market clears in the subgraph involved in the new minimum *s*-*t* cut. As a result, we can "freeze" the subgraph involved in the cut, and consider only the remaining graph when we raise the prices again. At any point in the algorithm, we refer to the subgraph in which we are increasing the prices as *active*, and to the rest of the graph as *frozen*.

- **Event type (2)**: A new edge from an active buyer to a frozen good enters the equality subgraph.

  Continuing the example from above, if we take the prices that caused the event of type (1) to occur and multiply the prices for the active goods by $x = 1.25$, then an edge between an active buyer and a frozen good is created in the equality subgraph, as shown in Figure 22.2(b). To address this type of event, we unfreeze the good incident on the new edge, and the connected component containing the good.

Analysis and description of algorithm

We now state the algorithm for the Market-Clearing Pricing Problem.

Market-Clearing Prices

```
$p_j \leftarrow \frac{1}{n} \ \forall j \in A$
Compute $\alpha_i = \max_{j \in A} \frac{u_{ij}}{p_j} \ \forall i \in B$
For each $j \in A$ such that $\nexists i \in B : (i,j) \in E$, $p_j \leftarrow \max_{i \in B} \frac{u_{ij}}{\alpha_i}$
$(F, F') \leftarrow (\emptyset, \emptyset)$ (frozen graph)
$(H, H') \leftarrow (A, B)$ (active graph)
While $H \neq \emptyset$
    Raise prices $p_j \leftarrow p_j x \ \forall j \in H$ until either:
    (1) $S \subseteq H$ becomes tight
        Move $(S, \Gamma(S))$ from $(H, H')$ to $(F, F')$
        Remove edges from $F'$ to $H$
    (2) For $i \in H'$, $j \in F$   $\alpha_i = \frac{u_{ij}}{p_j}$
        Add $(i, j)$ to $E$
        Move connected component containing $j$ from $(F, F')$ to $(H, H')$
Return $p_j \ \forall j \in A$.
```

There are several outstanding issues that we must address. First, can we implement the steps of the algorithm? Second, how long does the algorithm take? More details of the algorithm are presented in the next lecture.

ORIE 633 Network Flows March 17, 2004

Lecture 23

Lecturer: David P. Williamson *Scribe: Mateo Restrepo*

23.1 The market-clearing pricing problem

Recall from the previous lecture that the Market-Clearing pricing problem receives as input a set B of buyers, a set A of n divisible goods (in unit amounts), integer amounts of money m_i for each buyer $i \in B$, and integer utilities $u_{ij}\,\forall i \in B, \forall j \in A$. Each u_{ij} specifies the happiness that buyer i derives from one unit of good j. For a given set of prices and each user i we defined the best bang-per-buck ratio of user i to be $\alpha_i = \max_{j \in A} \frac{u_{ij}}{p_j}$. The goal of the problem is to find prices p_j such that the market clears, that is, such that all money is spent, no goods remain unpurchased and every buyer i buys items j that maximize his happiness, that is, goods j with $\frac{u_{ij}}{p_j} = \alpha_i$.

In this setting, it was natural to define the equality subgraph $G = (A, B, E)$, a bipartite (directed) graph. An edge (j, i), going from a good $j \in A$ to a buyer $i \in B$, is included in the graph if and only if item j maximizes i's happiness.

We showed how, given a set of prices, we can determine whether or not the market will clear by means of a simple flow computation. This max-flow computation is carried out in a flow network G' obtained from G by adding a source and sink, denoted by s and t, edges (s, j) of capacity p_j, $\forall j \in B$, and edges (i, t) of capacity m_i, $\forall i \in A$. The edges previously in G are assigned capacity ∞. The basic result was that the market clears iff both $\{s\}$ and $V - \{t\}$ are min s-t cuts (here $V = A \cup B \cup \{s, t\}$); that is, all goods are sold, and all money is spent.

Last time we started to discuss an (exponential time) algorithm for finding market-clearing prices. The algorithm raises prices on subsets of the goods while maintaining the invariant that $\{s\}$ is a min s-t cut in G'. Thus along the algorithm all goods can be completely allocated while buyers have money left over (surplus).

The algorithm begins by setting prices for all goods in such a way that every single buyer can buy the totality of the goods and every single good is bought by at least one buyer. In each iteration the algorithm proportionally raises prices of all goods in a certain set H (the active set), $p'_j \leftarrow p_j \cdot x$, for some $x > 1$, until one of the following two events occurs:

- **Event (1):** For some $x > 1$ there is another min cut besides $\{s\}$. In this event we *freeze* this part of the graph.
- **Event (2):** An edge from some active buyer i to a frozen good j enters the equality subgraph. In this event we unfreeze the portion of the frozen graph connected to j.

23.1.1 Formal statement of the algorithm and analysis

Definition 23.1 For a subset $S \subset A$ of the goods we define $\Gamma(S)$ to be the set of all buyers that are interested in some good in S, formally $\Gamma(S) := \{i \in B : j \in S, (i,j) \in E\}$.

Definition 23.2 The total price $p(S)$ of a set of goods $S \subset A$ is naturally defined as $p(S) := \sum_{j \in S} p_j$. In the same way, for a subset of buyers $T \subset B$ the total money they have is denoted by $m(T) = \sum_{i \in T} m_i$.

The notation being set, we state and prove the following crucial result.

Lemma 23.1 The invariant ($\{s\}$ is a minimum cut in G') holds if and only if $p(S) \leq m(\Gamma(S))$ for every $S \subset A$.

Proof: $\Rightarrow$. Suppose the invariant holds. Then the value of the min cut is $p(A)$ and every edge (s,j) of capacity p_j carries flow at full capacity. Thus, given any $S \subset A$, $p(S)$ units of flow are shipped from the nodes of S to the nodes of B connected to them, that is, to $\Gamma(S)$. Hence, there must be enough capacity among the buyers in $\Gamma(S)$ to ship this flow to the sink. Thus $P(S) \subset m(\Gamma(S))$, as desired.

$\Leftarrow$. Suppose $p(S) \leq m(\Gamma(S))$ for every $S \subset A$. Let $\{s\} \cup A_1 \cup B_1$ be any cut, with $A_1 \subset A$ and $B_1 \subset B$. We will prove that its capacity is at least that of $\{s\}$. For this let $A_2 = A \setminus A_1$ and $B_2 = B \setminus B_1$. The edges coming out of this cut can be classified into three groups: edges going from s to A_2, edges going from A_1 to B_2, edges going from B_1 to t. Notice that if there are any edges of the second type then the capacity of the cut is infinite and there is nothing to prove. So we may assume there are no edges of this type. This also implies that $B_1 \supset \Gamma(A_1)$, and correspondingly, $m(B_1) \geq m(\Gamma(A_1))$. The capacity of the remaining edges of the first and third types is clearly $p(A_2) + m(B_1)$. The inequality just deduced and the hypothesis then give

$$p(A_2) + m(B_1) \geq p(A_2) + m(\Gamma(A_1)) \geq p(A_2) + p(A_1) = p(A),$$

as desired. This finishes the proof. $\square$

The last lemma implies that the algorithm's invariant is *near* violation if for some factor x and some set S we have $x \cdot p(S) = m(\Gamma(S))$. This motivates the following definition.

Definition 23.3 We call a set S *tight* (with respect to a set of prices) if $p(S) = m(\Gamma(S))$.

It is easy to see that S is tight if the market clears in the part of the graph determined by $(S, m(\Gamma(S)))$.

With this terminology we then present (a high level description of) the algorithm:

Market-Clearing Prices (Devanur, Papadimitriou, Saberi, Vazirani 2002)

```
Price initialization:
    $p_j \leftarrow \frac{1}{n} \quad \forall j \in A$
    Compute $\alpha_i = \max_{j \in A} \frac{u_{ij}}{p_j} \quad \forall i \in B$
    For each $j \in A$ such that $\nexists i \in B : (i,j) \in E, p_j \leftarrow \max_i u_{ij}/\alpha_i$
    Recompute $G$
$(F, F') \leftarrow (\emptyset, \emptyset)$ (frozen graph)
$(H, H') \leftarrow (A, B)$ (active graph)
While $H \neq \emptyset$
    Raise prices $p_j \leftarrow p_j x$ for $j \in H$ for the minimal $x > 1$ such that either:
    (1) $S \subset H$ becomes tight
        Move $(S, \Gamma(S))$ from $(H, H')$ to $(F, F')$
        Remove edges from $F'$ to $H$.
    (2) For $i \in H'$, $j \in F$, $\alpha_i = \frac{u_{ij}}{p_j}$
        Add $(i,j) \in E$
        Move connected com of $j$ from $(F, F')$ to $(H, H')$. Return $p_j$, $\forall j \in A$.
```

We have not yet specified how to determine the minimal value of x such that either event (1) or event (2) occurs. Determining the minimum x such that event (2) occurs is not very difficult. For this, we just have to consider all pairs (i,j) of a buyer in H' and a product in F and determine $x_{ij} = \frac{p_j \alpha_i}{u_{ij}}$ is the minimal factor x for which the bang-per-buck factor of a good $j^* \notin H'$ maximizing the happiness of i equals the bang-per-buck factor of the frozen good j:

$$\frac{u_{ij^*}}{x p_{j^*}} = \frac{\alpha_i}{x} = \frac{u_{ij}}{p_j}$$

The minimum of these x_{ij} values is clearly the minimum x for which event (2) occurs. The discussion above implies that it can be calculated in $O(N^2)$ time where $N = |A| + |B|$.

The following lemma, whose proof we defer until next lecture, assures us that we can efficiently determine the minimum x for which event (1) occurs.

Lemma 23.2 The minimum x for which event (1) occurs can be determined by means of n max-flow computations.

Proof: COMING SOON (next lecture). □

But, how can we be sure that the algorithm finishes at all? The fact that the prices never decrease gives us a hint. Nevertheless, in order to guarantee that every time a price is raised we are making some non-negligible amount of progress is made, we need something like the following lemma:

Lemma 23.3 For any item j in a tight set S, p_j has denominator no greater than $\Delta \equiv nU^n$ (where $U \equiv \max_{ij} u_{ij}$).

Proof: We begin with the observation that if S is a tight set, then every connected component[1] of S is also a tight set.

For this, suppose $K_1, K_2, \dots K_c$ are the connected components of S. Then $\Gamma(S) = \bigcup_p \Gamma(K_p)$ where the union is disjoint (by the definition of connected components). This implies $m(\Gamma(S)) = \sum_p m(\Gamma(K_p)) \geq \sum_p p(K_p) = p(S)$, since $m(\Gamma(K_p)) \geq p(K_p)$. If it where the case that $m(\Gamma(K_p)) > p(K_p)$ for any connected component then the latter inequality would be strict and S wouldn't be tight.

Now, going back to our problem, consider any $j \in S$ and let S' be the minimal connected component of S containing j. Then for every other $k \in K$ there is path going from j to k. This path has the form $\langle j, i_1, j_1, i_2, j_2, \dots, k\rangle$, that is, it goes back and forth between A and B. For each time the path touches a buyer i between two goods j' and j^* we can write $\frac{u_{ij'}}{p_{j'}} = \frac{u_{ij^*}}{p_{j^*}}$, or $p_{j^*} = \frac{u_{ij^*}}{u_{ij'}} p_{j'}$. Iterating this relation along the path we can show that $p_k = p_j \frac{a_k}{b_k}$ where each a_k and b_k is a product of at most n utilities.

Since we can do the same for at every k in S', we get.

$$m(\Gamma(S')) = p(S') = \sum_{k \in S'} p_k = p_j \sum_{k \in S'} \frac{a_k}{b_k}$$

which implies that

$$p_j = \frac{m(\Gamma(S'))}{\sum_{k \in S'} \frac{a_k}{b_k}}.$$

This shows that p_j can be written in the form of a fraction where the denominator is a sum of at most n products of n utilities, and therefore is bounded by $\Delta = nU^n$. □

Before we proceed to the main result of this section, we note that if the price p_j in iteration $i + k$ is *strictly greater* than the price p_j in iteration i, then the difference must be at least $1/\Delta^2$. This result follows from the fact that for any positive integers a, b, c, d, if $a/b > c/d$ and $b, d \leq \Delta$, then $(a/b) - (c/d) \geq 1/\Delta^2$.

Now we are ready to bound the overall running time of the algorithm

Theorem 23.4 The algorithm runs in $O(m(B)n^2\Delta^2 MF)$ time, where MF is the time required by a max flow computation.

Proof: First, we observe that, by Lemma 23.2, the time per iteration is no more than that of n max flows, or $O(n \cdot MF)$. We proceed to bound the number of iterations.

By the Lemma and observation above, each time good j is frozen because of event (1), its price p_j has increased by $1/\Delta^2$. Each time event (1) happens, some good's price has

[1]We here abuse of the terminology a bit and call $S \subset A$ a *connected component* if S results from the intersection of a connected component of the bipartite graph G with A

increased, so we assign it to this freezing. Thus after k executions of event (1), the total surplus is at most $m(B) - (k/\Delta^2)$. Thus event (1) can occur at most $m(B)\Delta^2$ times. On the other hand, there can be at most n consecutive iterations of the main loop in which event (2) occurs instead of event (1), simply because there are at most n goods, and each time event (2) occurs one good gets unfrozen. We conclude that the total number of iterations is at most $(n+1)m(B)\Delta^2 = O(n \cdot m(B)\Delta^2)$ and the total time is $O(n \cdot m(B)\Delta^2 \cdot nMF)$, as desired. $\square$

ORIE 633 Network Flows March 19, 2004

Lecture 24

Lecturer: David P. Williamson *Scribe: Ivan Lysiuk*

24.1 The market-clearing pricing problem

24.1.1 Analysis of the algorithm (cont.)

Recall the market-clearing pricing problem we introduced last time:

Market-Clearing Pricing Problem

- **Input:**
  - Set B of buyers
  - Set A of unit amounts of divisible goods ($|A| = n$)
  - Integer amount of money m_i, $\forall i \in B$
  - Integer utilities u_{ij}, $\forall i \in B$, $\forall j \in A$
    (u_{ij} = happiness for buyer i from one unit of good j)
- **Goal:** Find prices p_j, $\forall j \in A$ such that the market clears:
  - All buyers buy only goods that maximize happiness
  - All money is spent
  - No good remains unpurchased

Let α_i denote the maximum "bang-per-buck" that the buyer i can receive, i.e.

$$\alpha_i = \max_{j \in A} \frac{u_{ij}}{p_j}.$$

The buyer i will purchase only goods j such that $\alpha_i = u_{ij}/p_j$. Given the prices of the goods, we can define an equality subgraph that represents the goods that each buyer may purchase.

Definition 24.1 The equality subgraph $G = (A, B, E)$ is a bipartite graph (with vertex set A and B) where $(i, j) \in E$ if and only if $\alpha_i = u_{ij}/p_j$.

Given a particular set of prices $p_j, j \in A$, we can determine whether the prices clear the market by performing a maximum flow computation. We add a source vertex s and a sink vertex t to the equality subgraph. For each good $j \in A$, we add an arc (s, j) with capacity p_j. For each buyer $i \in B$, we add an arc (i, t) with capacity m_i. We orient each edge (i, j) corresponding to a buyer $i \in B$ and a good $j \in A$ in the equality subgraph as a directed arc (j, i) with capacity ∞. In the previous lecture, we showed that the market clears if and only if the maximum flow value is $m(B) \equiv \sum_{i \in B} m_i$.

Our algorithm maintains the invariant that $\{s\}$ is a minimum s-t cut. We showed that the following.

Lemma 24.1 The invariant that $\{s\}$ is a minimum s-t cut holds if and only if for all $S \subseteq A$, $p(S) \leq m(\Gamma(S))$, where $\Gamma(S)$ is the neighborhood of S.

In the Market Clearing algorithm we needed to compute x^* such that for $x \leq x^*$ the invariant is maintained, and for $x > x^*$ the invariant is violated. For x^* there exists a set S such that $x^* \cdot p(S) = \Gamma(m(S))$.

Lemma 24.2 We can determine x^* and S using n max-flow computations.

Proof: Without loss of generality, we may assume that (A, B) is active. The same argument applies to arbitrary active subgraph. To determine such x, we need to determine

$$x^* \equiv \min_{\emptyset \neq S \subseteq A} \frac{m(\Gamma(S))}{p(S)}.$$

Let S^* denote the set that minimizes the above ratio.

We will start with $x \equiv m(B)/p(A) \geq x^*$, and compute max-flow for prices $x \cdot p_j$. If $\{s\}$ turns out to be a min s-t cut, then by Lemma 24.1 we know that $x = x^*$ and we're done. Furthermore, we can determine S^* by taking the maximum minimum cut (i.e., the largest set S such that S is an s-t min cut). The maximum minimum cut can easily be determined from the residual graph produced by maximum s-t flow algorithms.

If $x > x^*$ and $\{s\}$ is not a min s-t cut, let $\{s\} \cup A_1 \cup B_1$ be the min s-t cut. If we can show that $S^* \subseteq A_1 \subset A$, then the lemma is proven because we can recurse on $(A_1, \Gamma(A_1))$.

Claim 1: $A_1 \subset A$. If $A_1 = A$, then we must have $B_1 = B$ because the edges between A and B have infinite capacity. But, the cut $\{s\} \cup A \cup B$ has value $m(B)$ while the cut $\{s\}$ has value $x \cdot p(A)$, and we have $x \cdot p(A) \leq m(B)$. This implies that $\{s\}$ is a min s-t cut, contradicting our assumption. Therefore, $A_1 \subset A$.

Claim 2: $S^* \subseteq A_1$. Let $S_1 = S^* \cap A_1$ and $S_2 = S^* \cap A_2$. Note that we must have $\Gamma(S_1) \subseteq B_1$ since otherwise the cut will have infinite capacity. Note that the value of the cut $\{s\} \cup A_1 \cup B_1$ is $x \cdot p(A_2) + m(B_1)$.

First observe that it cannot be the case that $m(\Gamma(S_2) \cap B_2) < x \cdot p(S_2)$. Otherwise consider the cut $\{s\} \cup A_1 \cup S_2 \cup B_1 \cup (\Gamma(S_2) \cap B_2)$. It has value $x(p(A_2) - p(S_2)) + m(B_1) + m(\Gamma(S_2) \cap B_2) < x \cdot p(A_2) + m(B_1)$, which contradicts the fact that $\{s\} \cup A_1 \cup B_1$ is a minimum cut.

Note that this observation implies that it cannot be the case that $S^* = S_2$ since then $x^* < x$ implies that $m(\Gamma(S^*) \cap B_2) \leq m(\Gamma(S^*)) < x \cdot p(S^*)$.

Thus $S_1 \neq \emptyset$. Furthermore, we have that

$$m(\Gamma(S_2) \cap B_2) \geq x \cdot p(S_2) > x^* \cdot p(S_2).$$

By the definition of x^*,

$$m(\Gamma(S_2) \cap B_2) + m(\Gamma(S_1)) \leq m(S^*) = x^*(p(S_1) + p(S_2)).$$

Subtracting the first inequality from the second we obtain that

$$m(\Gamma(S_1)) < x^* \cdot p(S_1),$$

which contradicts the definition of x^*.
Thus, it must be the case that $S_2 = \emptyset \Rightarrow S^* \subseteq A_1$. $\square$

ORIE 633 Network Flows March 29, 2004

Lecture 25

Lecturer: David P. Williamson *Scribe: Ivan Lysiuk*

25.1 Generalized flows

In this lecture we return to discussing algorithms on (generalized) flows. We already introduced a generalization of flows when we considered adding costs to the edges. Today we will consider a model in which the edges are also "lossy", so the flow is no longer conserved, but transformed along edges. This models leaks, theft, taxes, etc.

$$\begin{tikzcd}
80\bullet \arrow[r, bend left=10, "\gamma=3/4"] & \bullet \arrow[r, bend left=10, "\gamma=1/2"] & \bullet 30
\end{tikzcd}$$

In the above graph, if we start with 80 units of flow, we obtain 60 units after following the first arc and 30 units after the second arc. We call the parameter γ the "gain" of the edge.

Another application for this model would be converting currency. Consider, for instance, the graph below in which we want to convert, say, \$1000 into Hungarian forints. Besides the "gain" factor we can also add, as before, capacity constraints to (some) edges, for example we can convert at most \$800 directly into forints. Note that some paths lead better rates than others; for example, the $\$ \rightarrow euro \rightarrow forint$ path gives an exchange rate of 6 $forints/\$$ as opposed to the direct path for which the rate is just 5.

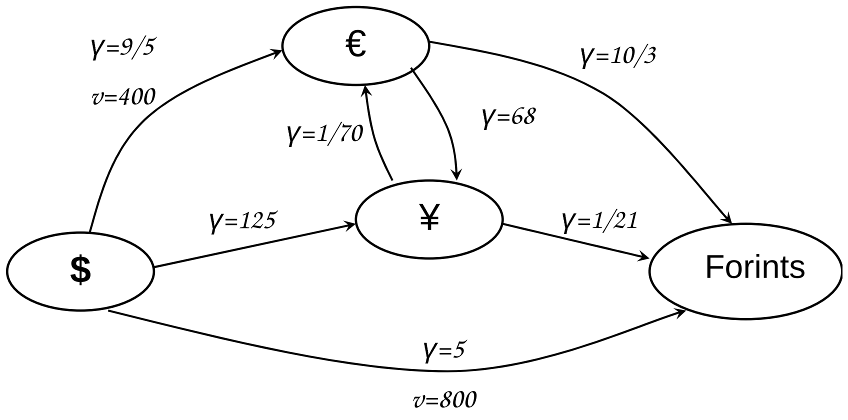


Figure 25.1: Currency conversion

25.1.1 Definitions

In this section we will define the generalized circulation problem. We state the problem first, then we give additional definitions to clarify the notation/meaning of our goal.

Generalized Circulation Problem

- **Input:**
  - A *symmetric* directed graph $G = (V, A)$, i.e. $(i, j) \in A \Rightarrow (j, i) \in A$
  - Designated sink $t \in V$
  - Integer capacities $u_{ij}\ \forall (i, j) \in A$
  - Gains $\gamma_{ij} : \gamma_{ji} = 1/\gamma_{ij}$ for all $(i, j) \in A$
  - All γ's are ratios of integers
  - All input integers are bounded by B.
- **Goal:** Find a circulation g that maximizes the excess e_t^g, denoted by $|g|$, and also called the value of the flow.

The following definitions will help us clarify what we mean by excess of a flow in the context of the generalized circulation problem.

Definition 25.1 A flow $g : A \to \Re$ is a *generalized pseudoflow* if:

- $g_{ij} \leq u_{ij}$ for all $(i, j) \in A$ (capacity constraints)
- $g_{ij} = -\gamma_{ji} g_{ji}$ for all $(i, j) \in A$ (anti-symmetry condition)

Definition 25.2 The *residual excess* of a flow g at a node i is given by

$$e_i^g = - \sum_{j:(i,j)\in A} g_{ij}.$$

If $e_i^g > 0$ we say we have an *excess* at node i. If $e_i^g < 0$ we say we have a *deficit* at node i.

For example, if the flow on the upper edge of the figure below is 200 units, then the flow on the lower reverse edge is -40 by antisymmetry. Note that the definition of excess, although somewhat unintuitive, is capturing the notion of the total amount of flow entering a node minus that leaving the node.

$$\begin{tikzcd} 200\bullet \arrow[r, bend left, "\gamma=1/5"] & \bullet 40 \arrow[l, bend left, "\gamma=5"] \end{tikzcd}$$

Definition 25.3 A *flow* g is a pseudoflow such that $e_i^g \geq 0 \; \forall i \in V$.

Definition 25.4 A *circulation* is a flow such that $e_i^g = 0 \; \forall i \in V, i \neq t$.

Thus our goal is to find a circulation that maximizes the excess at the sink vertex t.

We now start defining some concepts we will need to give our optimality conditions for the generalized circulation problem.

Definition 25.5 Given a pseudoflow g in G, we define the *residual graph* $G_g = (V, A_g)$:

$$\begin{aligned} A_g &= \{(i,j) \in A : g_{ij} < u_{ij}\} \\ u_{ij}^g &= u_{ij} - g_{ij} \end{aligned}$$

Definition 25.6 A *labelling function* $\mu : V \to \Re^{\geq 0} \cup \{\infty\}$ such that $\mu_t = 1$, represents the change in units of measurement of a node. Namely

$$\mu_i = \frac{\text{new units}}{\text{old units}}$$

For example if we wanted to perform the currency conversion (from Figure 1) in cents instead of dollars, we would need $\mu_\$ = 100$. The conversion rates involving the relabelled node would be affected (5 forints/\$ becomes .05 forints/cent), and also the capacity of the edges incident to the node (800 would become 80000 on the lowest edge, for instance).

In general we would have to perform the following changes for the gains, capacities, and excess at each relabelled node:

$$\begin{aligned} u_{ij}^\mu &= u_{ij}\mu_i \\ \gamma_{ij}^\mu &= \gamma_{ij} \times \mu_j / \mu_i \\ e_i^\mu &= e_i \mu_i \end{aligned}$$

If we already have some pseudoflow g, note that we also have to relabel it: $g_{ij}^\mu = g_{ij}\mu_i$. Note that relabelling does not change the value of $|g|$ since $\mu_t = 1$ by definition. Thus $|g^\mu| = |g|$.

Definition 25.7 For a path P, we define the *gain of the path* as follows:

$$\gamma(P) = \prod_{(i,j) \in P} \gamma_{ij}$$

Similarly for a cycle C, the gain of the cycle is:

$$\gamma(C) = \prod_{(i,j) \in C} \gamma_{ij}.$$

We use the following terminology for a cycle C. If $\gamma(C) > 1$, then C is a *flow-generating cycle.* If $\gamma(C) < 1$, then C is a *flow-absorbing cycle.*

Definition 25.8 We call μ a *canonical labelling* if

$$\mu_i = \max_{\text{path P from i to t}} \gamma(P)$$

We can find the canonical labels by setting $c_{ij} = -\log(\gamma_{ij})$, and finding the shortest path in G using lengths c_{ij}. If we set $c_{ij} = -\log(\gamma_{ij})$, then

$$c_{ij} = -\log(\gamma_{ij}) = -\log(\frac{1}{\gamma_{ji}}) = -\log(\gamma_{ji}) = -c_{ji}$$

Then for path P

$$\sum_{(i,j)\in P} c_{ij} = -\sum_{(i,j)\in P} \log(\gamma_{ij}) = -\log \prod_{(i,j)\in P} \gamma_{ij} = -\log(\gamma(P)).$$

Therefore, finding the maximum gain path from i to t is equivalent to finding the shortest path from i to t using costs c_{ij}. However, shortest paths are not well-defined if we have any negative-cost cycles. Here negative-cost cycles are equivalent to having flow generating cycles, since

$$\sum_{(i,j)\in C} c_{ij} < 0 \Leftrightarrow \sum_{(i,j)\in C} \log(\gamma_{ij}) > 0 \Leftrightarrow \log(\gamma(C)) > 0 \Leftrightarrow \gamma(C) > 1.$$

We will use the convention that if we cannot reach t from i then $\mu_i = 0$.

Definition 25.9 An *augmenting path P* in G_g is a path from a node with excess to the sink t.

Definition 25.10 A *generalized augmenting path* (GAP) is a flow generating cycle in the *residual graph G_g* with a (possibly trivial) path from a node on cycle to the sink t.

25.1.2 Optimality conditions

We are now ready to state the optimality conditions for the generalized circulation problem.

Theorem 25.1 The following are equivalent for a generalized circulation g:

1. g is optimal
2. G_g has no generalized augmenting paths (no GAPs).
3. There exist labelling μ such that the relabelled gains satisfy

$$\gamma^{\mu}_{ij} \leq 1, \forall (i,j) \in A_g$$

We will prove this theorem in the next lecture.

ORIE 633 Network Flows March 31, 2004

Lecture 26

Lecturer: David P. Williamson *Scribe: Chandrashekhar Nagarajan*

26.1 Generalized flows

We have already seen about generalized flows in the last lecture. Let us recollect some of the definitions required to come up with an algorithm to solve generalized flow problems.

26.1.1 Definitions

In this section we will define the generalized circulation problem. We will state the problem first, then give additional definition to clarify the notation/meaning of our goal.

Generalized Circulation Problem

- **Input:**
  - A *symmetric* directed graph $G = (V, A)$, i.e. $(i, j) \in A \Rightarrow (j, i) \in A$
  - A sink $t \in V$
  - Integer capacities u_{ij} $\forall (i, j) \in A$
  - Gains $\gamma_{ij} : \gamma_{ji} = 1/\gamma_{ij}$ for all $(i, j) \in A$
  - All γ's are ratios of integers
  - All input integers are bounded by B.
- **Goal:** Find a circulation g that maximizes the excess e^g_t, denoted by $|g|$, and also called the value of the flow.

The following definitions will help us clarify what we mean by excess of a flow in the context of the generalized circulation problem.

Definition 26.1 A flow $g : A \rightarrow \Re$ is a *generalized pseudoflow* if:

- $g_{ij} \leq u_{ij}$ for all $(i, j) \in A$ (capacity constraints)
- $g_{ij} = -\gamma_{ji} g_{ji}$ for all $(i, j) \in A$ (anti-symmetry condition)

Definition 26.2 The *residual excess* of a flow g at a node i is given by

$$e^g_i = - \sum_{j:(i,j)\in A)} g_{ij}.$$

If $e_i^g > 0$ we say we have an *excess* at node i. If $e_i^g < 0$ we say we have a *deficit* at node i.

Definition 26.3 A *flow* g is a pseudoflow such that $e_i^g \geq 0 \ \forall i \in V$.

Definition 26.4 A *circulation* is a flow such that $e_i^g = 0 \ \forall i \in V, i \neq t$.

Definition 26.5 Given a pseudoflow g in a graph $G = (V, A, u)$, we define the *residual graph* $G_g = (V, A_g, u^g)$ (where the u's denote the capacities) as follows:

$$A_g = \{h(i,j) \in A : g_{ij} < u_{ij}\}$$
$$u_{ij}^g = u_{ij} - g_{ij}$$

Definition 26.6 A *labeling function* $\mu : V \to \Re^{\geq 0} \cup \{\infty\}$ such that $\mu_t = 1$, represents the change in units of measurement of a node. Namely

$$\mu_i = \frac{\text{new units}}{\text{old units}}$$

In general we would have to perform the following changes for the gains, capacities, and excess at each relabeled node:

$$u_{ij}^\mu = u_{ij}\mu_i$$
$$\gamma_{ij}^\mu = \gamma_{ij} \times \mu_j/\mu_i$$
$$e^{g,\mu} = e_i^g \mu_i$$
$$g_{ij}^\mu = g_{ij}\mu_i$$

Note that the definitions above preserve antisymmetry: namely, $g_{ij}^\mu = -\gamma_{ji}^\mu g_{ji}^\mu$ if and only if $g_{ij} = -\gamma g_{ji}$. Also notice that relabelling does not change the value of $|g|$ since $\mu_t = 1$ by definition; thus $|g^\mu| = |g|$.

Definition 26.7 For a path P, we define the *gain of the path* as follows:

$$\gamma(P) = \prod_{(i,j)\in P} \gamma_{ij}$$

Similarly for a cycle C, the gain of the cycle is:

$$\gamma(C) = \prod_{(i,j)\in C} \gamma_{ij}.$$

We use the following terminology for a cycle C. If $\gamma(C) > 1$, then C is a *flow-generating cycle.* If $\gamma(C) < 1$, then C is a *flow-absorbing cycle.* If $\gamma(C) = 1$, then C is a *unit-gain cycle.*

Definition 26.8 We call μ a *canonical labeling* if

$$\mu_i = \max_{\text{path P from i to t}} \gamma(P)$$

We can find the maximum $\gamma(P)$ by setting $c_{ij} = -\log(\gamma_{ij})$, and finding the shortest path in G using lengths c_{ij}. This is true because

$$\sum_{(i,j)\in P} c_{ij} = -\sum_{(i,j)\in P} -\log(\gamma_{ij}) = -\log \prod_{(i,j)\in P} \gamma_{ij} = -\log(\gamma(P)).$$

So finding the shortest path using lengths c_{ij} is equivalent to maximizing the gain from i to t. However, shortest paths are not well-defined if we have any negative-cost cycles. Here negative-cost cycles are equivalent to having flow generating cycles, since

$$\sum_{(i,j)\in C} c_{ij} < 0 \Leftrightarrow \log(\gamma(C)) > 0 \Leftrightarrow \gamma(C) > 1.$$

We will use the convention that if we cannot reach t from i then $\mu_i = 0$.

Finally we want to define what we would like to detect if we have not yet discovered the optimal solution (our circulation does not yet produce the maximal excess).

Definition 26.9 A *generalized augmenting path* (GAP) is a flow generating cycle in the *residual graph* G_g with a (possibly trivial) path from a node on cycle to the sink t.

26.1.2 Optimality conditions

We are now ready to state the optimality conditions for the generalized circulation problem.

Theorem 26.1 The following are equivalent for a generalized circulation g:

1. g is optimal
2. G_g has no generalized augmenting paths (no GAPs).
3. There exist labeling μ such that the relabeled gains satisfy

$$\gamma^{\mu}_{ij} \le 1, \forall (i,j) \in A_g$$

Proof:

- ($\neg 2 \Rightarrow \neg 1$) Assume that a GAP exists in G_g. Let C be the flow generating cycle, and P be the path from a node i on the cycle to the sink t. Now consider a flow of δ coming into i (ignore for now the source of this flow). If we push this flow around the cycle C we end up back at i with a flow of $\delta\gamma(C)$. Since $\gamma(C) > 1$ we can pay back the original δ flow, and still remain with $\delta(\gamma(C)-1) > 0$ amount of flow at i. Pushing forward this flow from i to t on the path P, we add an extra $\delta(\gamma(C)-1)\gamma(P)$ flow at t. Set δ such that residual capacities (along C and P) are obeyed, and we get a circulation g' such that $|g'| > |g|$. Thus g was not optimal. Note that flow constraints are satisfied for every node and hence no nodes with excess are created.

- ($2 \Rightarrow 3$) Let S be the set of nodes that can reach t in G_g. We have no GAPs (by assumption) in S, thus there are no negative cost cycles in S for costs $c_{ij} = -\log \gamma_{ij}$. Set C_i to be the shortest path from i to t with costs c_{ij}, and $\mu_i = e^{-C_i}$. If $(i,j) \in A_g$ then, by definition of the c_i's we have that $C_i \leq c_{ij} + C_j$. This implies that

  $$\mu_i = e^{-C_i} \geq e^{-c_{ij}-C_j} = \mu_j e^{-c_{ij}} = \gamma_{ij}\mu_j.$$

  Thus $\gamma_{ij}\mu_j/\mu_i \leq 1$. By setting $\mu_i = 0$ for all $i \in V - S$ we ensure that our labeling satisfies the conditions of (3). To see this, note that by the definition of S, there are no arcs in A_g with $i \notin S$ and $j \in S$. If $i \in S$ and $j \notin S$ then $\gamma^{\mu}_{ij} = 0 \leq 1$. If $i, j \notin S$, then using the convention that $0/0 = 0$, we have that $\gamma^{\mu}_{ij} = 0 \leq 1$.

- ($3 \Rightarrow 1$) Given labelling μ and circulation g, consider any other circulation $\tilde{g}$. Although we will consider the relabelled circulations g^{μ} and $\tilde{g}^{\mu}$, we drop the superscript of μ for the rest of this proof. Let us focus on an edge $(i,j) \in A$.

  - If $g_{ij} < \tilde{g}_{ij}$ then $g_{ij} < u_{ij}$ as $\tilde{g}_{ij} \leq u_{ij}$. So $(i,j) \in A_g \Rightarrow \gamma^{\mu}_{ij} \leq 1$ (By assumption of (3)).
  - If $g_{ij} > \tilde{g}_{ij}$ then $-\gamma^{\mu}_{ji} g_{ji} > -\gamma^{\mu}_{ji}\tilde{g}_{ji}$ (by anti-symmetry) $\Rightarrow g_{ji} < \tilde{g}_{ji} \Rightarrow (j,i) \in A_g \Rightarrow \gamma^{\mu}_{ji} \leq 1 \Rightarrow \gamma^{\mu}_{ij} \geq 1$.

  So for any arc $(i,j) \in A$

  $$(\gamma^{\mu}_{ij} - 1)(g_{ij} - \tilde{g}_{ij}) \geq 0.$$

  Summing over all arcs in A, we obtain

  $$\sum_{(i,j)\in A} (\gamma^{\mu}_{ij} - 1)(g_{ij} - \tilde{g}_{ij}) \geq 0.$$

  We can rewrite this as

  $$\sum_{(i,j)\in A} \gamma^{\mu}_{ij}(g_{ij} - \tilde{g}_{ij}) - \sum_{(i,j)\in A} (g_{ij} - \tilde{g}_{ij}) \geq 0.$$

  By antisymmetry $-\gamma^{\mu}_{ij} g_{ij} = g_{ji}$; note here we are really using the relabelled flows so that antisymmetry holds. We again rewrite the above as

  $$\sum_{(i,j)\in A} (\tilde{g}_{ji} - g_{ji}) - \sum_{(i,j)\in A} (g_{ij} - \tilde{g}_{ij}) \geq 0.$$

  Since in a circulation $e^i_g = -\sum_{j:(i,j)\in A} g_{ij} = 0$, for all $i \neq t$, we can reduce in the previous expression, for both g and $\tilde{g}$, all arcs that are not leaving t. We obtain, finally, that

  $$\sum_{i:(t,i)\in A} \tilde{g}_{ti} - \sum_{i:(t,i)\in A} g_{ti} \geq 0 \Leftrightarrow -\sum_{i:(t,i)\in A} g_{ti} \geq -\sum_{i:(t,i)\in A} \tilde{g}_{ti}.$$

  The last expression is, by definition, $|g| \geq |\tilde{g}|$, and it is true for any arbitrary circulation $\tilde{g}$. Thus we can conclude that g is optimal, so (1) holds.

$\square$

ORIE 633 Network Flows April 2, 2004

Lecture 27

Lecturer: David P. Williamson *Scribe: Anke van Zuylen*

27.1 Generalized flows

Recall the generalized flow problem that we talked about in the last two classes:

Generalized Flow Problem

- **Input:**
  - A *symmetric* directed graph $G = (V, A)$, i.e. $(i, j) \in A \Rightarrow (j, i) \in A$
  - Source s and sink t, $s, t \in V$
  - Integer capacities u_{ij} $\forall (i, j) \in A$
  - Gains $\gamma_{ij} : \gamma_{ji} = 1/\gamma_{ij}$ for all $(i, j) \in A$
  - All γ's are ratios of integers
  - All input integers are bounded by B.
- **Goal:** Find a circulation g that maximizes $|g| \equiv e_t^g$.

In the previous lecture we proved the following theorem.

Theorem 27.1 The following are equivalent for a circulation g

(1) g is optimal,

(2) there are no generalized augmenting paths (GAPs) in G_g,

(3) there exists a labelling μ such that $\gamma_{ij}^{\mu} \leq 1$ for all $(i, j) \in A_g$.

Recall that a Generalized Augmenting Path (GAP) is a flow generating cycle with a path (possibly trivial) from a node on the cycle to the sink t. A labelling function $\mu : V \to R^{>0}$, $\mu_t = 1$, changes units of measurement in the graph in the following manner:

$$\begin{aligned} u_{ij}^{\mu} &= u_{ij}\mu_i \\ e_i^{g,\mu} &= e_i^g \mu_i \\ \gamma_{ij}^{\mu} &= \frac{\gamma_{ij}\mu_j}{\mu_i} \end{aligned}$$

We also made the implicit assumption that $g^{\mu}_{ij} = g_{ij}\mu_i$, so a feasible circulation is still feasible after relabelling.

27.1.1 Truemper's algorithm

We'll look at a primal-dual style algorithm which decouples GAPs by (1) pushing flows along flow-generating cycles to create excesses at nodes, and (2) pushing these excesses to the sink.

Idea: Look at costs $c_{ij} = -\log \gamma_{ij}$. Then flow generating cycles are equivalent to negative cost cycles with respect to c.

Claim 27.2 By using min-mean cost cycle cancelling, we can cancel all flow generating cycles in $O(m^2n^3 \log(nB))$ time, where B is the max integer involved in the gain ratios.

Proof: See problem set 4, problem 3. □

To identify maximum gain paths, recall the definition of canonical labels.

Definition 27.1 μ is a canonical labelling if

$$\mu_i = \max_{\text{paths } P \text{ from } i \text{ to } t} \gamma(P)$$

where $\gamma(P) = \prod_{(i,j)\in P} \gamma_{ij}$.

Note that we can compute the canonical labels by finding the shortest path from each node to the sink t using costs $c_{ij} = -\log \gamma_{ij}$, since all negative cycles with respect to c have been cancelled.

Let μ be a canonical labelling. Then

$$\gamma^{\mu}_{ij} = \frac{\gamma_{ij}\mu_j}{\mu_i} \leq 1 \;\; \forall (i,j) \in A_g.$$

This follows from the following: Let C_i denote the cost of the shortest path from i to t using costs c. Note that if P is the maximum gain path from i to t, then $C_i = \sum_{(i,j)\in P} c_{ij} \Rightarrow e^{C_i} = e^{\sum_{(i,j)\in P} c_{ij}} = e^{-\log \gamma(P)} = \frac{1}{\gamma(P)} = \frac{1}{\mu_i}$.

$$\begin{aligned} & C_i \leq c_{ij} + C_j \\ \Rightarrow \quad & e^{C_i} \leq e^{c_{ij}} e^{C_j} \\ \Rightarrow \quad & \frac{1}{\mu_i} \leq e^{-\log \gamma_{ij}} \frac{1}{\mu_j} \\ \Rightarrow \quad & \frac{\gamma_{ij}\mu_j}{\mu_i} \leq 1 \end{aligned} \tag{27.1}$$

Note that if an edge (i,j) is on the shortest path from i to t, then we have $C_i = c_{ij} + C_j$, and its relabelled gain $\gamma^{\mu}_{ij} = 1$.

The above suggests the following algorithm:

Truemper's algorithm (1977)

```
Cancel all flow generating cycles
While $\exists e^g_i > 0$ that can reach $t$ in $G_g$
    Compute canonical labels $\mu$
    Compute a max flow $f$ pushing flow from $\{i : e^g_i > 0\}$ to $t$
        in graph $(V, \{(i,j) \in A_g : \gamma^{\mu}_{ij} = 1\})$, capacities $u_{ij} = u^{g,\mu}_{ij}$
    $g^{\mu}_{ij} \leftarrow g^{\mu}_{ij} + f_{ij}$ .
```

By the discussion above, the algorithm finds maximum flows from the nodes with excess along the highest-gain paths to the sink. Note that we will not go into how the algorithm handles situations where excesses cannot reach the sink; we assume that we can "undo" the creation of any excess by pushing flow back along the flow-generating cycle that created it.

Lemma 27.3 No flow-generating cycles are created by augmenting g^{μ} by the maximum flow f.

Proof: All arcs initially have $\gamma^{\mu}_{ij} \leq 1$. The maximum flow creates only arcs with $\gamma^{\mu}_{ij} = 1$, since it only pushes flow along arcs with $\gamma^{\mu}_{ij} = 1$, so reverse arcs that appear in the residual graph have $\gamma^{\mu}_{ji} = \frac{1}{\gamma^{\mu}_{ij}} = 1$. □

Lemma 27.4 The number of iterations of the main loop is no more than the number of different possible gains of paths.

Proof: After augmentation, there exists no augmenting path P from a node with excess to the sink with $\gamma^{\mu}(P) = 1$. So $\gamma^{\mu}(P) < 1$ for any path in the new residual graph. Let μ_i be the old canonical label for some node with excess i, and let its new canonical label be $\mu'_i = \gamma(P)$ for some path P. Then

$$\frac{\mu'_i}{\mu_i} = \frac{\gamma(P)}{\mu_i} = \frac{1}{\mu_i} \prod_{(k,l)\in P} \gamma_{kl} = \frac{\mu_t}{\mu_i} \prod_{(k,l)\in P} \frac{\gamma_{kl}\mu_k}{\mu_k} = \prod_{(k,l)\in P} \frac{\gamma_{kl}\mu_k}{\mu_l} = \gamma^{\mu}(P) < 1 \Rightarrow \mu'_i < \mu_i.$$

Now, since the canonical label of node i is equal to the gain of some path, the fact that the canonical label of i is strictly decreasing in each iteration implies that there can be no more iterations than the number of different gains of paths. □

27.1.2 A gain-scaling algorithm

How can we make Truemper's algorithm into a polynomial time algorithm?

Idea Modify the gains so that there are only a polynomial number of different gains of paths.

Let $b = (1+\varepsilon)^{\frac{1}{n}}$. For $\gamma_{ij} \leq 1$, round γ_{ij} down to the nearest power of b.

$$\begin{aligned} \bar{\gamma}_{ij} &= b^{\lfloor \log_b \gamma_{ij} \rfloor} \\ \bar{\gamma}_{ji} &= \frac{1}{\bar{\gamma}_{ij}} \end{aligned}$$

How many different gains of paths are there with respect to the scaled gains?

- gain of any path is no more than B^n
- gain of any path is not less than B^{-n}

So at most $\log_b B^{2n} = O(\frac{n \log B}{\log b}) = O(\frac{n^2 \log B}{\log(1+\varepsilon)})$ different gain paths. For a reasonable choice of ε, this will be polynomial.

ORIE 633 Network Flows April 5, 2004

Lecture 28

Lecturer: David P. Williamson *Scribe: Yankun Wang*

28.1 Generalized flows

28.1.1 Truemper's algorithm

In previous lectures, we considered a generalized circulation problem in which arcs (i,j) are associated with gains $\gamma_{ij} > 0$ which serve as multiplicative transformations of flows along those edges. We defined *pseudoflows* $g : A \to \mathbb{R}$ which met capacity ($g_{ij} \leq u_{ij}$) and antisymmetry ($g_{ji} = -\gamma_{ij} g_{ij}$) constraints, and defined the residual excess of a node i in a pseudoflow g as $e_i^g = -\sum_{j:(i,j)\in A} g_{ij}$. A *flow* is a pseudoflow that has only non-negative residual excesses. The generalized circulation problem then was to find a pseudoflow that maximized the residual excess at some sink node t, $e_t^g \equiv |g|$, subject to constraints that $e_i^g = 0, \forall i \in V, i \neq t$.

Truemper's algorithm (1977)

```
Cancel flow-generating cycles
While $\exists e_i^g > 0$ that can reach $t$ in $G_g$
    Compute canonical labels $\mu$
    Compute max flow $f$ that pushes flow from $\{i \in V : e_i^g > 0\}$ in graph
        $(V, \{(i,j) \in A_g : \gamma_{ij}^\mu = 1\}, u_{ij}^g)$
    $g_{ij}^\mu \leftarrow g_{ij}^\mu + f_{ij}$
```

We have shown the following lemma.

Lemma 28.1 The number of iterations of the while loop is no more than the possible number of different gains of paths.

Given this lemma, we just need to make sure that the number of different possible gains is polynomially bounded. In general, though, this is not the case.

28.1.2 Gain scaling

However, we can force the desired condition by modifying the gains so that there are only a polynomial number of different gains of paths by rounding the reduced gains as follows.

Let $b = (1+\epsilon)^{1/n}$. Then for $\gamma_{ij} \leq 1$, define

$$\begin{aligned}\overline{\gamma}_{ij} &= b^{\lfloor \log_b \gamma^{\mu}_{ij} \rfloor} \\ \overline{\gamma}_{ji} &= 1/\overline{\gamma}_{ij},\end{aligned}$$

Note that rounding down is consistent for both $\overline{\gamma}_{ij}$ and $\overline{\gamma}_{ji}$ since either $\gamma_{ij} = 1$ (which then implies that $\overline{\gamma}_{ij} = \overline{\gamma}_{ji} = b^0 = 1$) or only one of γ_{ij} and γ_{ji} is greater than 1.

How many different gain values of paths are now possible? We can bound the gain of a path P by

$$B^{-n} \leq \quad \overline{\gamma}(P) \quad \leq B^n,$$

and given that all gains are powers of b, then only

$$O(\log_b B^{2n}) = O(n \log_b B) = O(n^2 \log_{(1+\epsilon)} B)$$

paths with different gains are possible. Thus, if we let H denote a network with gains $\overline{\gamma}$, then we may use Truemper's algorithm to find an optimal flow h in H is polynomial time. To obtain an approximate solution to the original generalized flow problem, we *interpret h in G* as follows:

$$g_{ij} = \begin{cases} h_{ij} & \text{if } h_{ij} \geq 0 \\ -\gamma_{ji} h_{ji} & \text{if } h_{ij} < 0. \end{cases}$$

Finally, we obtain the following bounds on the amount of flow found.

Definition 28.1 A flow g is ϵ-*optimal* if for an optimal flow g^*, $|g| \geq (1-\epsilon)|g^*|$.

Theorem 28.2 For an optimal flow h in H, its interpretation in G is ϵ-optimal.

Proof: Let g^* be the optimal flow in G. What is its value in H? For each path P pushing δ units of excess to the sink t gives $\gamma(P)\delta$ units at the sink. In H, the same path gives

$$\begin{aligned}\overline{\gamma}(P) &\geq \frac{\gamma(P)\delta}{b^{|P|}} \\ &\geq \frac{\gamma(P)\delta}{b^n} \\ &\geq \frac{\gamma(P)\delta}{1+\epsilon} \\ &\geq \gamma(P)(1-\epsilon)\delta\end{aligned}$$

units of flow at the sink. Thus, the total flow pushed to the sink in H by g is

$$\begin{aligned}\sum_P \overline{\gamma}(P)\delta_P &\geq \sum_P \gamma(P)\delta_P(1-\epsilon) \\ &= (1-\epsilon)|g^*|\end{aligned}$$

so the optimal flow h must have value greater than $(1-\epsilon)|g^*|$ in the network H. Since the gains in G are only larger than those in H, the interpretation of h in G will only have larger value, and thus is at least $(1-\epsilon)|g^*|$. □

This gives a polynomial-time ϵ-optimal approximation algorithm for the generalized flow problem.

28.1.3 Error scaling

Now we will present the following lemma. Its proof will be given later.

Lemma 28.3 Given a B^{-4m} optimal flow with no flow-generating cycles, we can compute an optimal flow with one max-flow computation.

Setting $\epsilon = B^{-4m}$, we can obtain a B^{-4m}-approximation using the Truemper algorithm with gain scaling. Unfortunately, this method is not polynomial in B, since $\log_{1+\epsilon} B$ for $\epsilon = B^{-4m}$ is $O(B^{4m}\log B)$. This is exponential in the size of the input. It is possible, however, to modify the Truemper gain scaling approach to derive an actual polynomial time algorithm for computing exact generalized flows. The basic idea is that we will invoke the Truemper gain scaling algorithm to iteratively obtain half of the remaining flow in the residual graph by setting $\epsilon = 1/2$. Then only $log_2 B^{4m}$ iterations of the Truemper gain scaling algorithm are needed to get a B^{-4m}-optimal flow. We introduce the algorithms below.

Iterated Rounded Truemper (Tardos and Wayne, 1998)

```
$g \leftarrow 0$
For $i \leftarrow 1$ to $\log_2 B^{4m}$
    $g \leftarrow$Cancel cycles in $G_g$
    $g \leftarrow$Rounded Truemper $(G_g, \frac{1}{2})$
```

Rounded Truemper(G, ϵ)

```
Round down gains to $(1+\epsilon)^{1/n}$ to get graph $H$
$h \leftarrow$Truemper$(H)$
Return interpretation of $h$ in $G$
```

Theorem 28.4 For $\epsilon = \frac{1}{2}$, Rounded Truemper runs in $O(CC + (n^2 \log B)MF)$ time.

Proof: Trivial. □

Theorem 28.5 Iterated Rounded Truemper computes a B^{-4m}-optimal flow in $O((m \log B)(CC + (n^2 \log B)MF))$ time.

Proof: The initial flow is 1-optimal. Each iteration finds a $\frac{1}{2}$-optimal flow in G_g, so the ith iteration is 2^{-i}-optimal. In $\log_2 B^{4m}$ iterations, the flow is B^{-4m}-optimal. □

We now turn to the proof of Lemma 28.3. We start with the following lemma.

Lemma 28.6 Suppose we have a flow g and labels μ such that $\gamma^{\mu}_{ij} \leq 1$ for all $(i,j) \in A_g$. If for an optimal flow g^*, $|g^*| - |g| < B^{-2m}$, we can compute the optimal flow in one max-flow computation.

Proof: Given g, μ, let

$$h^{\mu}_{ij} \leftarrow \begin{cases} 0 & \text{if } \gamma^{\mu}_{ij} = 1 \\ g^{\mu}_{ij} & o.w. \end{cases}$$

We claim that B^{-2m} is the least common denominator for gains of paths.

If the claim is true, then μ_i must be an integral multiple of B^{-2m}. This in turn implies that u^{μ}_{ij} is an integral multiple of B^{-2m}. Furthermore,

$$e^{\mu,h}_i = -\sum_{j:(i,j)\in A} h^{\mu}_{ij} = -\sum_{j:(i,j)\in A, \gamma^{\mu}_{ij}\neq 1} h^{\mu}_{ij} = -\sum_{j:(i,j)\in A, \gamma^{\mu}_{ij}>1} u^{\mu}_{ij} + \sum_{j:(i,j)\in A, \gamma^{\mu}_{ij}<1} \gamma^{\mu}_{ji} u^{\mu}_{ji}$$

is an integral multiple of B^{-2m}. This implies that $|h^{\mu}| = e^{\mu,h}_t$ is also an integral multiple of B^{-2m}.

Now we set up the network K as is shown in Figure 28.1. Add a dummy source node s and a sink node t', and all the edges with $\gamma^{\mu}_{ij} = 1$. We compute flow f from s, which satisfies all the deficits ($e^{h,\mu}_i < 0$) and maximizes the flow into t. We know that a flow satisfying all deficits exists, since g^{μ} satisfies them. By the integrality property of the maximum flow problem, we know that the flow f^{μ} has value that must be an integral multiple of B^{-2m}, since all the capacities, supplies, and demands are multiples of this factor.

By a similar argument, we can show that $|g^*|$ must be an integral multiple of B^{-2m}. Now we have the following inequalities

$$|g^*| \geq |h^{\mu}| + |f^{\mu}| \geq |g^{\mu}| > |g^*| - B^{-2m},$$

where $|h^{\mu}|, |f^{\mu}|$ and $|g^*|$ are all integral multiples of B^{-2m}. Therefore $|h^{\mu}| + |f^{\mu}|$ is optimal. □

We can now prove Lemma 28.3.

Proof of Lemma 28.3: If we have no flow-generating cycles, we can compute the canonical labels μ. For optimal flow g^*

$$|g| > |g^*| - B^{-4m}|g^*|,$$

but $|g^*| \leq mU \leq mB \leq B^m$. Thus we can apply the previous lemma. □

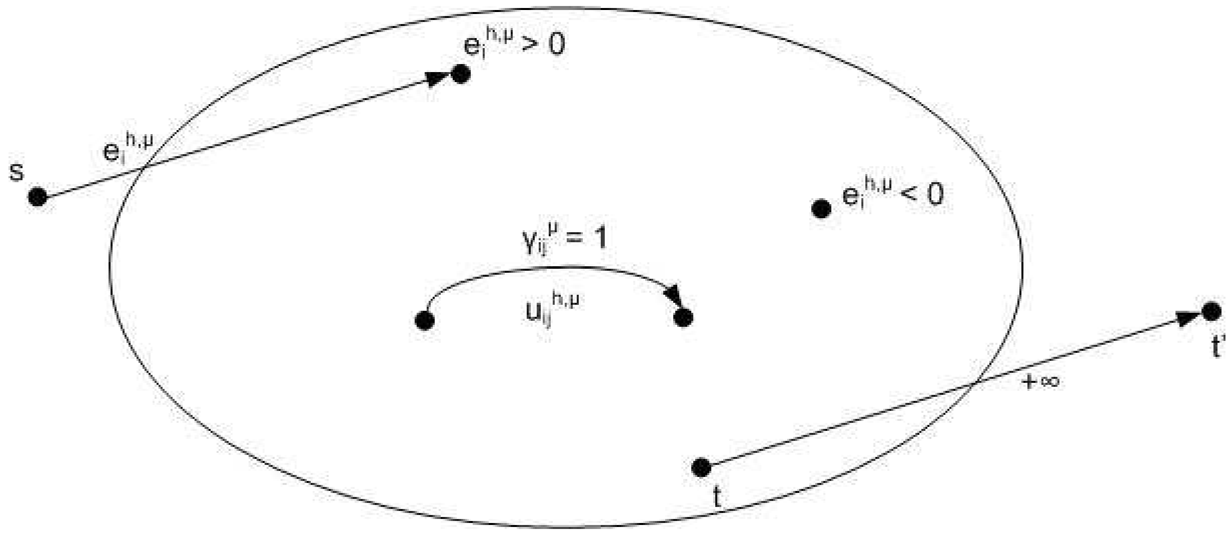


Figure 28.1: Network K

ORIE 633 Network Flows April 7, 2004

Lecture 29

Lecturer: David P. Williamson *Scribe: Yankun Wang*

29.1 Network design problems

29.1.1 The survivable network design problem

So far in the class, we have taken the network as a given. There has been some fixed network, and we try to find some flow in it. However, finding a network that can support certain kinds of flows is also an interesting problem, and one that comes up in practice (in the telecommunications industry, for example). For the next few lectures we will consider a simple type of network design problem known as the Survivable Network Design Problem (SNDP).

Survivable Network Design Problem (SNDP)

- **Input:**
  - An undirected graph $G = (V, E)$
  - Costs $c_e \geq 0$ for each edge $e \in E$
  - Requirement $r_{ij} \in N \; \forall i, j \in N, i \neq j$
- **Goal:** Find a minimum-cost set of edges F such that for $\forall i, j$, $\exists$ at least r_{ij} edge disjoint paths from i to j in (V, E).

In other words, we are trying to find a network such that for each i, j, we can send a flow of r_{ij} units from i to j, treating each edge as having unit capacity. In typical industrial applications, r_{ij} is small; for example, $r_{ij} \in \{0, 1, 2\}$. SNDP is NP-hard even when $r_{ij} \in \{0, 1\}$. Because it is NP-hard, we do not think we can find an efficient algorithm to solve it, so instead we will consider approximation algorithms for the problem.

Definition 29.1 An algorithm is an *α-approximation algorithm* for SNDP if

1. it runs in polynomial time
2. it produces a solution whose value $\leq \alpha$ times the value of the optimal solution (OPT).

29.1.2 The generalized Steiner tree problem

We consider a simple case first: $r_{ij} \in \{0, 1\}$. This is called the Generalized Steiner tree problem. We are going to present a primal-dual 2-approximation algorithm for the problem.

To have a primal-dual algorithm, we'll need both a primal and a dual. Consider the integer program

$$\begin{aligned} \text{Min} \quad & \sum c_e x_e \\ \text{subject to:} \quad & \\ & \sum_{e \in \delta(S)} x_e \geq \max_{i \in S, j \notin S} r_{ij} \qquad && \forall S \\ & x_e \in \{0,1\}, \end{aligned}$$

where we have defined $\delta(S) = \{(i,j) \in E : i \in S, j \notin S\}$. We claim that this integer program models the SNDP. This follows from the max-flow/min-cut theorem: the constraints ensure that for any i-j cut, there are at least r_{ij} edges in any cut. Hence we will be able to send a flow of value r_{ij} from i to j. By the integrality property of flows, we can decompose such a flow into r_{ij} edge-disjoint paths.

The LP relaxation of this problem replaces the integer constraints $x_e \in \{0,1\}$ with $0 \leq x_e \leq 1$. Note that we can solve the relaxation in polynomial time via the ellipsoid method. The ellipsoid method says that given a solution x, if we can tell in polynomial time whether x is a feasible solution for the LP, and, if not, produce a constraint violated for the LP in polynomial time, then we can find an optimal solution to the LP in polynomial time. In this case, we can find a violated constraint as follows. Given a solution x, treat each edge as having capacity x_e. For each i and j, compute a maximum i-j flow. If some flow has value less than r_{ij}, then the minimum i-j cut for this flow gives a constraint that is violated for the LP above.

It will be useful to restate the right-hand side of the primal LP in terms of a function f. Define $f(S) = \max_{i \in S, j \notin S} r_{ij}$. Then our primal LP becomes

$$\begin{aligned} \text{Min} \quad & \sum c_e x_e \\ \text{subject to:} \quad & \\ & \sum_{e \in \delta(S)} x_e \geq f(S) \qquad && \forall S \\ & x_e \geq 0. \end{aligned}$$

Note that we have dropped $x_e \leq 1$. In the case that $r_{ij} \in \{0,1\}$, then $f(S) \leq 1$ for all S and a minimum-cost solution will have no variable $x_e > 1$ anyway. Taking the dual of this LP, we obtain

$$\begin{aligned} \text{Max} \quad & \sum_S f(S) y_S \\ \text{subject to:} \quad & \\ & \sum_{S : e \in \delta(S)} y_S \leq c_e \qquad && \forall e \in E \\ & y_S \geq 0. \end{aligned}$$

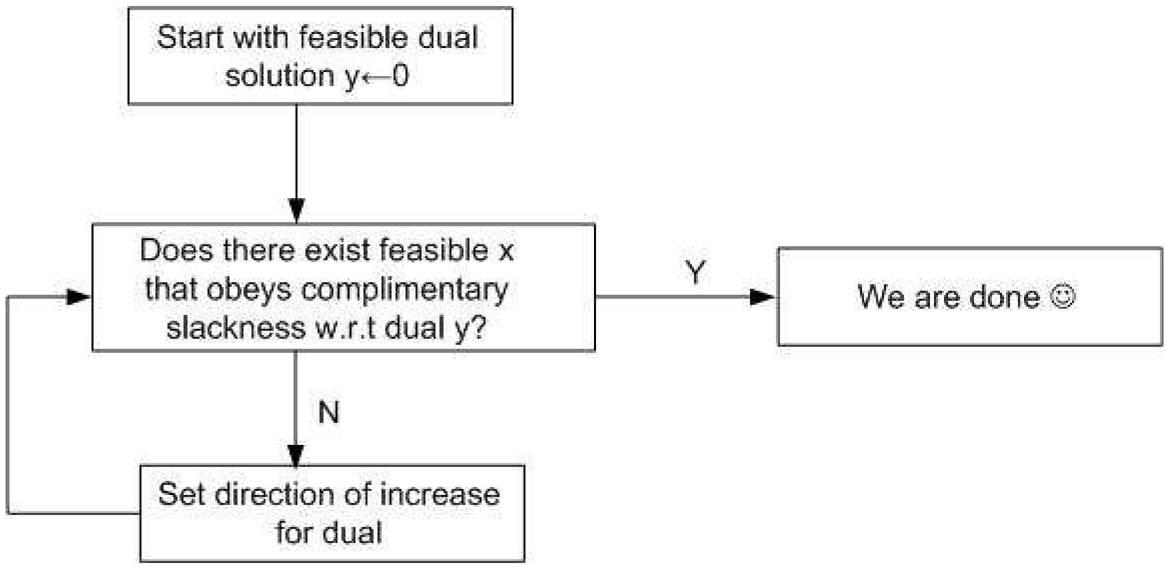

Start with feasible dual solution y←0
Does there exist feasible x that obeys complimentary slackness w.r.t dual y?
Y
We are done ☺
N
Set direction of increase for dual

Figure 29.1: The General Process of Primal-Dual Method

Recall the basic format of the primal-dual method, shown in Figure 29.1. In the case of these LPs, the complimentary slackness conditions are

$$\begin{aligned} \text{(primal) } x_e > 0 &\Rightarrow \sum_{S:e\in\delta(S)} y_S = c_e \\ \text{(dual) } y_S > 0 &\Rightarrow \sum_{e\in\delta(S)} x_e = f(S) \end{aligned}$$

Because we are dealing with a problem in which we would like to find an integer optimum solution, and the linear programming relaxation does not necessarily have integer vertices, we will have to modify the primal-dual method slightly. We cannot hope to have a solution such that there is an integer primal solution x that will obey the complementary slackness conditions with respect to our current dual y. So we will have to give up on something. In this case, we give up on the second type of complementary slackness conditions. We will only hope to find an integer solution x obeying the primal complementary slackness conditions.

Now we need to worry about whether the primal-dual method will still work in such circumstances. Is it the case that if we do not have a feasible integer solution x obeying the primal complementary slackness with respect to our current dual y, can we get a direction of increase for the dual? To see that we can, given a feasible dual y, let $A = \{e \in E : \sum_{S:e\in\delta(S)} y_S = c_e\}$. A is the set of all edges e such that we could set $x_e = 1$ and obey the primal complementary slackness conditions. Suppose A is not a feasible solution. This implies that there exists i, j, such that $r_{ij} = 1$, but i, j are not connected in (V, A). Pick some connected component C such that $i \in C$ and $j \notin C$. Since C is a connected component of (V, A), we know that for any $e \in \delta(C)$ it must be the case that $\sum_{S:e\in\delta(S)} y_S < c_e$. Let

$$\epsilon = \min_{e\in\delta(C)} (c_e - \sum_{S:e\in\delta(S)} y(S));$$

by the previous reasoning $\epsilon > 0$. We can increase $y(C)$ by $\epsilon > 0$; since $f(C) = 1$, the dual objective function increases.

29.1.3 A primal-dual algorithm for the generalized Steiner tree problem

This motivates the following primal-dual algorithm for the Generalized Steiner tree problem. We will analyze it in the next lecture.

Primal-DualGST

```
$y \leftarrow 0$
$A_0 \leftarrow \emptyset$
$l \leftarrow 0$ ($l$ is a counter)
While $A_l$ is not feasible
    $l \leftarrow l + 1$
    $\mathcal{C}_l \leftarrow$ {set of all connected components $C : f(C) = 1$}
    Increase $y_C$ for all $C \in \mathcal{C}_l$ uniformly until $\exists\ e_l \notin A_{l-1} : \sum_{S:e_i\in\delta(S)} y_S = c_{e_l}$
    $A_l \leftarrow A_{l-1} \cup \{e_l\}$
$A' \leftarrow A_l$
For $j \leftarrow l$ down to 1
    If $A' - \{e_j\}$ is still feasible
        $A' \leftarrow A' - \{e_j\}$
Return $A'$
```

ORIE 633 Network Flows April 9, 2004

Lecture 30

Lecturer: David P. Williamson *Scribe: Chandrashekhar Nagarajan*

30.1 Network design problems

30.1.1 A primal-dual algorithm for the generalized Steiner tree problem (cont.)

Last class we introduced the Survivable Network Design Problem (SNDP) and looked at a simple case with $r_{ij} \in \{0,1\}$ which is called the generalized Steiner tree (GST) problem. We also gave a primal-dual algorithm which we will show is a 2-approximation algorithm for the GST problem. Let us recall the problem and the algorithm.

GST problem

- **Input:**
  - Undirected graph $G = (V, E)$
  - Costs $c_e \geq 0 \;\forall\; e \in E$
  - $r_{ij} \in \{0,1\} \;\forall\; i,j \in V$
- **Goal:** Find a minimum-cost set $F \subseteq E$ such that $\forall\; i,j\;\; s\;\; t\;\; r_{ij} = 1, i$ and j are connected in (V, F).

Primal Dual GST

```
$y \leftarrow 0$
$A_0 \leftarrow \emptyset$
$l \leftarrow 0$
while $A_l$ not feasible
    $l \leftarrow l + 1$
    $\mathcal{C}_l \leftarrow \{$ connected components $C$ of $(V, A)$ s.t. $f(C) = 1\}$
    Increase $y_c$ for all $c \in \mathcal{C}_l$ uniformly until $\exists e_l \in A_l$ such that
        $\sum_{s:e_l \in \delta(S)} y_s = c_{e_l}$
    $A_l \leftarrow A_{l-1} \cup \{e_l\}$
$A' \leftarrow A_l$
for $j \leftarrow l$ down to 1
    if $A' - \{e_j\}$ is feasible
        $A' \leftarrow A' - \{e_j\}$
```

Definition 30.1 Let $f(S) = \max_{i \in S, j \notin S} r_{ij}$

The LP relaxation of the above GST problem is as follows.

$$\begin{aligned}
\text{Min} \quad & \sum c_e x_e \\
\text{subject to:} \quad & \\
& \sum_{e \in \delta(S)} x_e \geq f(S) \qquad \forall S \\
& x_e \geq 0.
\end{aligned}$$

The dual of the above LP is

$$\begin{aligned}
\text{Max} \quad & \sum_S f(S) y_S \\
\text{subject to:} \quad & \\
& \sum_{S: e \in \delta(S)} y_S \leq c_e \qquad \forall e \in E \\
& y_S \geq 0.
\end{aligned}$$

Note that the algorithm outputs a feasible primal and a feasible dual solution. The for loop at the end of the algorithm is to remove extra edges while maintaining feasibility of the set A'.

We now turn to showing that this algorithm is a 2-approximation algorithm for GST. We make the following claim, which we will prove later on.

Claim 30.1 For any step l, final solution A'

$$\sum_{C \in \mathcal{C}_l} \left| A' \cap \delta(C) \right| \leq 2 \cdot |\mathcal{C}_l|$$

Given the claim, we can prove the following theorem, showing that the algorithm is a 2-approximation algorithm.

Theorem 30.2 For solution A' and final dual solution y

$$\sum_{e \in A'} c_e \leq 2 \cdot \sum_{S \subseteq V} f(S) y_S \leq 2 \cdot OPT$$

where OPT is the optimal solution for the GST problem.

Proof: The second inequality follows since any feasible dual solution, by weak duality, has value no more than the optimal solution to the primal LP, which in turn has value no more than the optimal solution to the corresponding integer program, which is OPT.

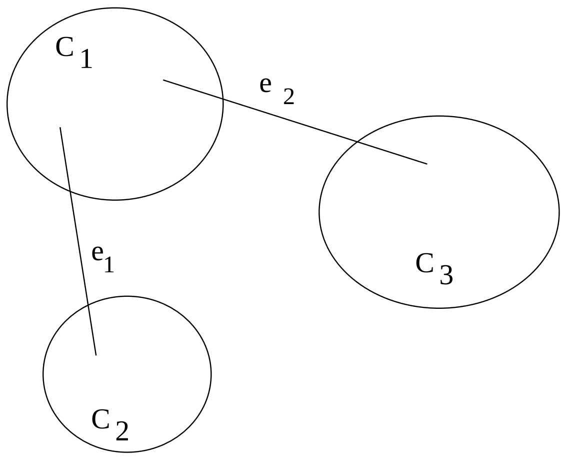


Figure 30.1: Components at l^{th} iteration. $C_1, C_2, C_3 \in \mathcal{C}_l, e_1, e_2 \in A'$

Since we are maintaining primal complementary slackness conditions, we know that for all $e \in A'$

$$c_e = \sum_{S: e \in \delta(S)} y_S.$$

Therefore

$$\sum_{e \in A'} c_e = \sum_{e \in A'} \sum_{S: e \in \delta(S)} y_s = \sum_{S \subseteq V} y_S \left| A' \cap \delta(S) \right|.$$

So we want to show that

$$\sum_{S \subseteq V} y_S \left| A' \cap \delta(S) \right| \leq 2 \cdot \sum_{S \subseteq V} f(S) y_S.$$

We will prove this by induction on the construction of the dual solution y.
Base case: Since $y = 0$ at the beginning of the algorithm, the inequality follows.
Inductive Step : Assume the inequality holds for dual variables at iteration $l - 1$ and we will show that it holds for iteration l.

To do this, we will compare the change in LHS and RHS of the inequality. In step l, each y_C for $C \in \mathcal{C}_l$ is increased by some ϵ. Then the change of the RHS is $2\epsilon|\mathcal{C}_l|$, while the change in LHS is $\epsilon \sum_{C \in \mathcal{C}_l} |A' \cap \delta(S)|$. So by Claim 30.1 the change in the LHS is no greater than the change in the RHS. So by induction the theorem is proved. $\square$

Now we must prove Claim 30.1.

Proof of Claim 30.1: Let

$$B_{l-1} = A' - A_{l-1}$$

Then $A_{l-1} \cup B_{l-1}$ is feasible since $A' \subseteq A_{l-1} \cup B_{l-1}$. We claim that for any $e \in B_{l-1}$, $A_{l-1} \cup B_{l-1} - \{e\}$ is not feasible. Consider the order in which edges are added to our set of edges.

$$\{e_1, e_2, \cdots, e_{l-1}\}, e_l, \cdots, e_{k-1}, e_k.$$

The reverse order is the order in which we delete the extra edges in the set while maintaining feasibility. So we absolutely need $e \in B_{l-1}$, where $e = e_j$ for some index $j \geq l$, since we considered it for deletion at a point when all the edges in $(e_1, e_2, \cdots, e_{l-1})$ were in A' during the deletion step. If we didn't remove e, then it must be needed for feasibility.

Contract each connected component of (V, A_{l-1}) to a single vertex. Let the new vertex set formed be V'. Consider $G' = (V', B_{l-1})$ where each edge $e = (u, v) \in B_{l-1}$ is connected to the vertices $u', v' \in V'$ corresponding to components that contain u and v respectively. We claim that G' is a forest. To see this, suppose there is some cycle $\Gamma \in G'$. We can delete any edge in Γ while still maintaining feasibility since any pair of vertices i and j that are connected are still connected after the edge is removed.

We now say that a vertex $v \in V'$ is labeled "red" if the corresponding component $C \subseteq V$ has $f(C) = 1$. Other vertices are labeled "blue". Let *Red* be the set of red vertices and *Blue* be the set of blue vertices. See Figure 30.2.

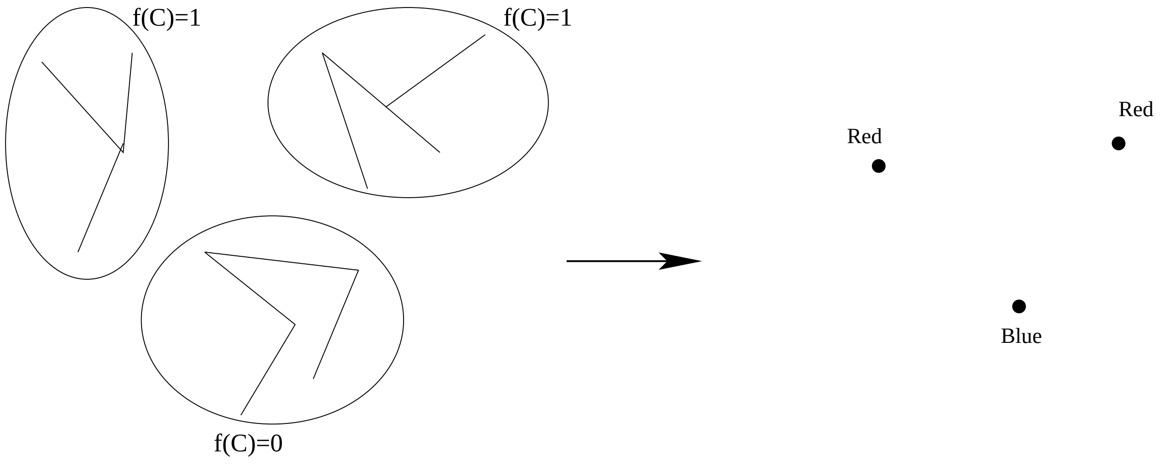


Figure 30.2: Components to vertices transformation

By the definition of $\mathcal{C}_l$, $|\mathcal{C}_l| = |Red|$ and $|A' \cup \delta(C)|$ is the degree of vertex v' corresponding to the component C. Thus the inequality we are trying to prove is equivalent to

$$\sum_{v \in Red} deg(v) \leq 2 \cdot |Red| \,.$$

To prove this, we need to establish one more claim; namely, that $v \in Blue \Rightarrow deg(v) \neq 1$. Consider the edge $e \in B_{l-1}$ that connects a blue vertex v to its parent (See Figure 30.3). Let S be the corresponding component of v. We know that $A_{l-1} \cup B_{l-1} - \{e\}$ is not feasible. This implies that there exists some $i \in S$ and $j \notin S$ with $r_{ij} = 1$. But then $f(S) = 1$, which implies that v is a red vertex. This is a contradiction. So any blue vertex v must have degree other than 1.

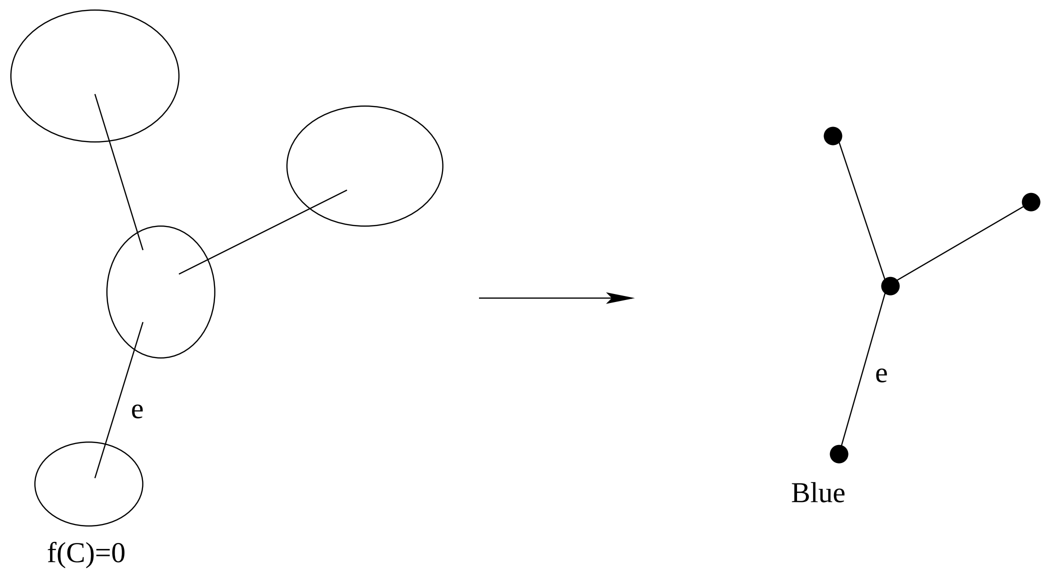


Figure 30.3: No blue vertex can be a leaf.

We now discard blue vertices of degree zero. Then

$$\begin{aligned}\sum_{v\in Red} deg(v) &= \sum_{v\in Red\cup Blue} deg(v) - \sum_{v\in Blue} deg(v)\\ &\leq 2\cdot(|Red|+|Blue|) - 2\cdot|Blue|\\ &\leq 2\cdot|Red|.\end{aligned}$$

The first inequality follows since G' is a forest and since all blue vertices have degree at least 2. □

ORIE 633 Network Flows April 12, 2004

Lecture 31

Lecturer: David P. Williamson *Scribe: Xin Qi*

31.1 Network design problems

31.1.1 The survivable network design problem

In this lecture, we will start talking about approximation algorithm based on LP rounding techniques for more general case of SNDP.

First, let us repeat the definition of SNDP:

Survivable Network Design Problem (SNDP)

- **Input:**
  - Undirected graph $G = (V, E)$
  - Costs $c_e \geq 0$ ($\forall e \in E$)
  - Requirements $r_{ij} \in \mathbb{N}$ ($\forall i, j \in V, i \neq j$)
- **Goal:** Find a min-cost $F \subseteq E$, s.t. $\forall i, j \in V, i \neq j$, there are at least r_{ij} edge-disjoint paths between i and j.

We also gave an integer LP formulation for SNDP.

Let $f(S) = \max_{i \in S, j \notin S} r_{ij}$. Then we have the following integer program:

$$
\begin{aligned}
\text{Min} \quad & \sum_{e \in E} c_e x_e \\
\text{subject to:} \quad & \\
& \sum_{e \in \delta(S)} x_e \geq f(S) \qquad \forall S \subset V \\
& x_e \in \{0, 1\}
\end{aligned}
$$

The above cut-based formulation completely characterizes SNDP, according to the maxflow-mincut theorem.

Now we can relax the last constraint as $0 \leq x_e \leq 1$, and thereby obtain a LP relaxation of SNDP. Obviously, the optimal solution to the LP sets a lower bound of the optimal solution of SNDP, denoted by OPT.

By the ellipsoid algorithm, we can solve the LP in polynomial time, provided a separation oracle that decides in polynomial time whether a solution is feasible or not, and if not, gives the violated constraint. In our case, the separation oracle will be a routine that computes maximum flow between all pairs of nodes (i, j), and checks if the flow value is less than r_{ij}. If not, the corresponding minimum i-j cut must give a violated constraint. Thus the above LP relaxation is polynomial-time solvable.

31.1.2 An LP rounding algorithm for the survivable network design problem

Before diving into the details of the algorithm, let us first take a look at the following useful terminology.

Definition 31.1 A function $f : 2^V \to \mathbb{Z}$ is *weakly supermodular*, if $f(V) = 0$, and for any $A, B \subseteq V$, one of the following holds:

- either $f(A) + f(B) \leq f(A \cap B) + f(A \cup B)$
- or $f(A) + f(B) \leq f(A - B) + f(B - A)$

Note that supermodular condition is only the first inequality, and the opposite inequality $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ is called submodular condition. They are sort of the discrete analogies of concavity and convexity.

Claim 31.1 The function

$$f(S) = \max_{i \in S, j \notin S} r_{ij}$$

is weakly supermodular.

Now we are going to give the central theorem underlying the whole algorithm. We will not prove it at this time, but just use it to build our approximation algorithm first.

Theorem 31.2 (Jain '98) For any weakly supermodular function f, any basic solution to the LP has at least one variable $x_e \geq \frac{1}{2}$.

This is really important in the development of the algorithm: we can round up x_e to 1, if it is no less than $\frac{1}{2}$, and we will not increase the value of the objective function by too much. Of course, we have to notice that rounding up the variables will never make the solution infeasible, according to the constraints of the LP.

Iterated Rounding (Jain '98)

```
$F \leftarrow \emptyset$
$f' \leftarrow f$
While $F$ not feasible solution
    find a basic solution to the LP with the function $f'$
    $F \leftarrow F \cup \{e \in E : x_e \geq \frac{1}{2}\}$
    $f'(S) \leftarrow f(S) - |F \cap \delta(S)|$
Return $F$
```

Note that we are unable to do the third statement in the while loop literally, since that would be exponential time. We would rather do it "conceptually"; the function f' will be defined implicitly.

Here comes the first question: how can we solve the LP with f'? We need to come up with a separation oracle that runs in polynomial time, and actually, it will not be that different than the one for the case of original function f. We will still compute maximum flow for each pair of nodes, setting the capacity of any edge $e \in F$ to be 1. (We can alternatively view it as explicitly adding the constraint $x_e = 1$ ($\forall e \in F$), since we round them up to 1.) We then find a maximum flow between every pair of vertices i and j; if the value of the flow is less than r_{ij}, then the minimum i-j cut S on edges in $E - F$ must have value less than $r_{ij} - |\delta(S) \cap F|$, and this gives a violated constraint.

Another important question we need to ask is: why is f' weakly supermodular?

Lemma 31.3 For any given subset $F \subseteq E$, if f is weakly supermodular, then so is $f'(S) = f(S) - |F \cap \delta(S)|$.

Proof: We will first show that for general $z \in \mathbb{R}^{|E|}_{\geq 0}$, if we define

$$z(H) = \sum_{e \in H} z_e$$

then

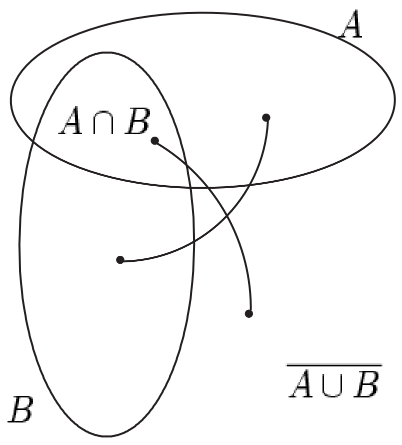
$$z(\delta(A)) + z(\delta(B)) \geq z(\delta(A \cap B)) + z(\delta(A \cup B))$$

and

$$z(\delta(A)) + z(\delta(B)) \geq z(\delta(A - B)) + z(\delta(B - A))$$

Do the proof by picture:

We can check all types of edges, and see that any edge in RHS appears in LHS. For example,



- Edge from $A \cap B$ to $\overline{A \cup B}$

  It shows up in both A and B. It also appears in both items of the RHS of the first inequality, but neither of the RHS of the second one.

- Edge from $A - B$ to $B - A$

  It shows up in both A and B. It also appears in both items of the RHS of the first inequality, but neither of the RHS of the second one.

In fact, these two types of edges are the only cases to make the inequalities not tight.

Now, we will use the above fact to prove the lemma.

Let

$$z_e = \begin{cases} 1 & e \in F \\ 0 & o.w. \end{cases}$$

then $f'(S) = f(S) - z(\delta(S))$.

Given any A and B, one the two statements of the weakly supermodularity holds for f:

- If the first holds,

$$\begin{aligned} f'(A) + f'(B) &= f(A) - z(\delta(A)) + f(B) - z(\delta(B)) \\ &\leq f(A \cap B) - z(\delta(A \cap B)) + f(A \cup B) - z(\delta(A \cup B)) \\ &= f'(A \cap B) + f'(A \cup B) \end{aligned}$$

- If the second holds, the proof is similar.

□

With the above discussion, we know that the algorithm is well defined, and yields a feasible solution to SNDP. Now we have to prove the performance guarantee of the algorithm.

Theorem 31.4 Iterated rounding is a 2-approximation algorithm for SNDP.

Proof: First, we need to argue that it runs in polynomial time. This is easy: every iteration we solve a LP instance, which can be done in polynomial time by ellipsoid algorithm, and the number of iterations is bounded by m, since $|F|$ increases by at least one per iteration.

Second, we want to show that

$$\sum_{e \in F} c_e \leq 2 \sum_{e \in E} c_e x_e$$

where x is a solution to the original LP, and F is the final set given by our algorithm.

The above result will lead to that the approximation ratio is 2, since the integral optimal solution to SNDP is clearly a feasible solution to the LP, and therefore $\sum_{e \in E} c_e x_e \leq OPT$.

The proof will be done by induction on the number of iterations of the main loop. We will do the proof in a "backward" way, that is, the induction hypothesis will be applied to the execution after the first iteration.

Let

$$\hat{x}_e = \begin{cases} x_e & x_e \geq \frac{1}{2} \\ 0 & o.w. \end{cases}$$

For the set of edges F_1 added to F in the first iteration, the cost is clearly no more than $2 \sum_{e \in E} c_e \hat{x}_e$, according to the way of rounding.

Base case: If the algorithm terminates in one iteration and $F_1 = F$, then

$$\sum_{e \in F} c_e \leq 2 \sum_{e \in E} c_e \hat{x}_e \leq 2 \sum_{e \in E} c_e x_e.$$

Inductive case: Observe that

$$\sum_{e \in \delta(S)} x_e - \sum_{e \in \delta(S)} \hat{x}_e \geq f(S) - |F_1 \cap \delta(S)| = f'(S)$$

which implies that $x - \hat{x}$ is a feasible solution to the LP given the new function f' at the end of the first iteration.

Now apply the induction hypothesis: we know that for the set of edges F' added in future iterations (after the first one), the cost of F' will be no larger than the optimal of the new LP, hence no larger than any feasible solution of the new LP:

$$\sum_{e \in F'} c_e \leq 2 \sum_{e \in E} c_e (x_e - \hat{x}_e)$$

$\Rightarrow$

$$\sum_{e \in F' \cup F_1} c_e \leq 2 \sum_{e \in E} c_e(x_e - \hat{x}_e) + 2 \sum_{e \in E} c_e \hat{x}_e = 2 \sum_{e \in E} c_e x_e$$

$\Rightarrow$

$$\sum_{e \in F} c_e \leq 2 \cdot OPT$$

□

ORIE 633 Network Flows April 14, 2004

Lecture 32

Lecturer: David P. Williamson *Scribe: Xin Qi*

32.1 Network design problems

32.1.1 An LP rounding algorithm for the survivable network design problem

In this lecture, we will dive into the proof of the Jain's Theorem.

Let us first repeat the definition of weakly supermodular functions, and the content of Jain's Theorem.

Definition 32.1 A function $f : 2^V \to \mathbb{Z}$ is *weakly supermodular*, if $f(V) = 0$, and for any $A, B \subseteq V$, one of the following holds:

- either $f(A) + f(B) \leq f(A \cap B) + f(A \cup B)$
- or $f(A) + f(B) \leq f(A - B) + f(B - A)$

Theorem 32.1 (Jain '98) For any weakly supermodular function f, any basic solution to the LP has at least one $x_e \geq \frac{1}{2}$.

32.1.2 Proof of Jain's theorem

Actually, we are going to cheat a little bit. We will prove a weaker version of Jain's Theorem, and show that any basic solution to the LP has at least one $x_e \geq \frac{1}{3}$. The proof of the stronger version is significantly more work, but doesn't introduce many more ideas than what we are going to do.

Take a basic solution of LP, we can assume that $0 < x_e < 1$ $(\forall e \in E)$. We are safe to make this assumption, since

- if $\exists e$ such that $x_e = 1$, then we are done;
- if $\exists e$ such that $x_e = 0$, then we can remove that edge from the graph, and that will not affect the proof.

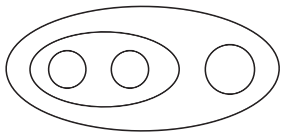
We will need the following definitions to facilitate our proof.

Definition 32.2 $A, B \subset V$ *cross* if $A - B$, $B - A$, and $A \cap B$ are non-empty.

Definition 32.3 $S \subset V$ is *tight* if $\sum_{e \in \delta(S)} x_e = f(S)$

Definition 32.4 A collection $\mathcal{L}$ of sets is called *laminar* if no two sets in L cross.

For a laminar set $\mathcal{L}$, all the sets in $\mathcal{L}$ either contain each other, or are disjoint from each other, as shown in the following figure.



Definition 32.5 For any cut $S \subset V$, $\chi_{\delta(S)} \in \{0,1\}^{|E|}$ is defined as

$$\chi_{\delta(S)}(e) = \begin{cases} 1 & e \in \delta(S) \\ 0 & o.w. \end{cases}$$

Then the constraints of the LP can be rewritten as

$$\chi_{\delta(S)} \cdot x \geq f(S) \quad \forall S \subset V.$$

Let m denote the number of fractional x_e, which is equal to the number of edges in the graph, by our assumption.

Theorem 32.2 For a basic solution x to the LP, there exists a collection of $\mathcal{L}$ of m sets such that

(1) S is tight for all $S \in \mathcal{L}$;

(2) The set of vectors $\{\chi_{\delta(S)} : S \in \mathcal{L}\}$ is linear independent;

(3) $\mathcal{L}$ is laminar.

Claim 32.3 (1) and (2) follow by properties of a basic solution.

So we only need to prove (3). Before actually doing the proof, we will first show the following lemma.

Lemma 32.4 If A and B cross, and are both tight, then either

- either $A \cap B$ and $A \cup B$ are tight, and $\chi_{\delta(A)} + \chi_{\delta(B)} = \chi_{\delta(A\cup B)} + \chi_{\delta(A\cap B)}$;
- or $A - B$ and $B - A$ are tight, and $\chi_{\delta(A)} + \chi_{\delta(B)} = \chi_{\delta(A-B)} + \chi_{\delta(B-A)}$.

Proof: Since f is weakly supermodular, one of the following two cases holds:

- either $f(A) + f(B) \le f(A \cap B) + f(A \cup B)$
- or $f(A) + f(B) \le f(A - B) + f(B - A)$

We will assume the first one holds, and the proof for the other case is similar.

Since A and B are both tight, we have

$$f(A) + f(B) = x(\delta(A)) + x(\delta(B))$$

Recall that last time we proved that for any $z \in \mathbb{R}^{|E|}_{\ge 0}$, the following two inequalities both hold:

$$z(\delta(A)) + z(\delta(B)) \ge z(\delta(A \cap B)) + z(\delta(A \cup B)) \tag{32.1}$$

$$z(\delta(A)) + z(\delta(B)) \ge z(\delta(A - B)) + z(\delta(B - A)). \tag{32.2}$$

We can apply this result to x:

$$f(A) + f(B) = x(\delta(A)) + x(\delta(B)) \ge x(\delta(A \cap B)) + x(\delta(A \cup B)) \ge f(A \cup B) + f(A \cap B)$$

in which the last inequality holds according to the constraints of LP.

Combining the above result with our assumption of f, we know that all the inequalities are tight. Especially for the the last inequality, it is not only tight for the sums, but also tight for those summands, i.e. $x(\delta(A\cap B)) = f(A\cap B)$ and $x(\delta(A\cup B)) = f(A\cup B)$, which means that $A \cap B$ and $A \cup B$ are both tight.

Another thing we have known from last lecture is that the only thing that can make inequality (32.1) not tight is an edge from $A-B$ to $B-A$. Thus because $x(\delta(A))+x(\delta(B)) = x(\delta(A \cap B)) + x(\delta(A \cup B))$ and no edge e has $x_e = 0$, we know that there is no edge from $A - B$ to $B - A$. This implies

$$\chi_{\delta(A)} + \chi_{\delta(B)} = \chi_{\delta(A\cap B)} + \chi_{\delta(A\cup B)}.$$

□

Now let us come back to the proof of theorem 32.2.

Let $\mathcal{T}$ be a basic solution meeting properties (1) and (2), and let $span(\mathcal{T})$ be the span of vectors $\{\chi_{\delta(S):S\in\mathcal{T}}\}$.

Let $\mathcal{L}$ be maximal collection of sets obeying properties (1) (2) and (3).

If $|\mathcal{L}| = m$, then we are done. So let us suppose $|\mathcal{L}| < m$, then we can choose a tight set S, such that $\chi_{\delta(S)} \in span(\mathcal{T})$, $\chi_{\delta(S)} \notin span(\mathcal{L})$, and there is no other such set crossing fewer sets in $\mathcal{L}$; that is, S crosses the fewest number of sets in $\mathcal{L}$.

Now we pick $T \in \mathcal{L}$ such that S and T cross. By Lemma 32.4, one of the two cases holds. Suppose it is the case that $S-T$ and $T-S$ are tight, and $\chi_{\delta(S)} + \chi_{\delta(T)} = \chi_{\delta(S-T)} + \chi_{\delta(T-S)}$.

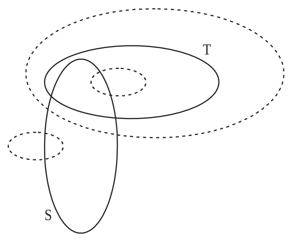
Claim 32.5 We can not have both $S-T$ and $T-S$ in $\mathcal{L}$.

We know that $\chi_{\delta(T)} \in span(\mathcal{L})$, and $\chi_{\delta(S)} \notin span(\mathcal{L})$.

If both $S-T$ and $T-S$ are in $\mathcal{L}$, then $\chi_{\delta(S-T)} + \chi_{\delta(T-S)} \in span(\mathcal{L})$, which implies $\chi_{\delta(S)} + \chi_{\delta(T)} \in span(\mathcal{L})$. Then we have a contradiction. $\diamond$

Claim 32.6 $S-T$, $T-S$, $S \cup T$, and $S \cap T$ all cross fewer sets in $\mathcal{L}$ than S.

Proof by picture:



We can observe that any set crossing one of $S-T$, $T-S$, $S \cup T$, $S \cap T$, but not T must also cross S. However, these four sets do not cross T, but S does cross T.

$\diamond$

From claim 32.5, we know that either $\chi_{\delta(S-T)} \notin span(\mathcal{L})$, or $\chi_{\delta(S-T)} \notin span(\mathcal{L})$. No matter which is the case, we have another tight set (either $S-T$ or $T-S$), outside $span(\mathcal{L})$, but crossing fewer sets in $\mathcal{L}$, which contradicts the choice of S.

Now we have finished the proof of theorem 32.2.

To be continued...

ORIE 633 Network Flows April 16, 2004

Lecture 33

Lecturer: David P. Williamson *Scribe: Alice Cheng*

33.1 Network design problems

33.1.1 Proof of Jain's theorem (cont.)

Recall the Survivable Network Design Problem from previous lectures:

Survivable Network Design Problem (SNDP)

- **Input:**
  - Undirected graph $G = (V, E)$
  - Costs $c_e \geq 0$ for all $e \in E$.
  - Requirements $r_{ij} \in \mathbb{N}$ for all $i, j \in V, i \neq j$.
- **Goal:** Find a min-cost $F \subseteq E$ such that $\forall i, j \in V$, $i \neq j$, there are at least r_{ij} edge-disjoint paths in F between i and j.

Our goal is to prove the following theorem by Jain:

Theorem 33.1 (Jain 1998) For any weakly supermodular function f, and any basic solution to the LP (stated below), x, there is at least one variable x_e such that $x_e \geq \frac{1}{2}$

The SNDP can be modelled by an integer program which relaxes to the following linear program:

$$\begin{aligned} \text{Min} \quad & \sum_{e \in E} c_e x_e \\ \text{subject to:} \quad & \\ & \sum_{e \in \delta(S)} x_e \geq f(S) && \text{for each } S \subseteq V, \\ & 0 \leq x_e \leq 1 && \text{for each } e \in E. \end{aligned}$$

Last time, we used this theorem to show that Jain's Iterated Rounding algorithm is a 2-approximation algorithm. For simplicity, we will show that there always exists a variable x_e such that $x_e \geq \frac{1}{3}$, rather than $x_e \geq \frac{1}{2}$ as stated in Theorem 1. Then the proof that we gave last lecture can be easily modified to show that the corresponding Iterated Rounding algorithm is a 3-approximation algorithm.

Recall from last lecture some definitions:

Definition 33.1 $A, B \subset V$, A, B *cross* if $A - B$, $B - A$, and $A \cap B$ are nonempty.

Definition 33.2 A collection $\mathcal{L}$ of sets is *laminar* if no pair $A, B \in \mathcal{L}$ cross.

Definition 33.3 For a solution x to the LP, S is *tight* if $\sum_{e \in \delta(S)} x_e = x(\delta(S)) = f(S)$.

Definition 33.4 The edge incidence vector $\chi_F \in \{0,1\}^m$ for $F \subseteq E$ is defined componentwise as $\chi_F(e) = \begin{cases} 1 & \text{if } e \in F, \\ 0 & \text{otherwise.} \end{cases}$

We proved last lecture the following theorem:

Theorem 33.2 For a basic solution x to the LP, there exists a collection $\mathcal{L}$ of m sets (where m is the number of fractional variables of x) such that

- S is tight for all $S \in \mathcal{L}$
- The set $\{\chi_{\delta(S)} : S \in \mathcal{L}\}$ is linearly independent.
- $\mathcal{L}$ is laminar.

Throughout the lecture, we will assume that all of the edges $e \in E$ have fractional x_e ($0 < x_e < 1$), since if $x_e = 0$, we can remove e from E without loss of generality, and in any case where $x_e = 1$, the theorem is trivially true.

Theorem 33.3 If $\mathcal{L}$ is a collection of sets with the properties stated in Theorem 2, there exists an $S \in \mathcal{L}$ such that there are no more than 3 edges in $\delta(S)$.

With the above result, we can prove Jain's theorem (Theorem 33.1).

Proof of Theorem 33.1: Let x be the basic solution for the LP. Let $\mathcal{L}$ be the collection given by Theorem 33.2. For all $S \in \mathcal{L}$, S is tight implies that $f(S) = x(\delta(S)) > 0$ if there are any edges in $\delta(S)$. Furthermore, since the set $\{\chi_{\delta(S)} : S \in \mathcal{L}\}$ is linearly independent, it cannot be the case that $\delta(S) =$. We know that $f(S)$ integer, so $f(S) \geq 1$ for all $S \in \mathcal{L}$. By Theorem 33.3, there is an $S \in \mathcal{L}$ such that $\delta(S) \leq 3$. This immediately implies that there is some $e \in \delta(S)$ such that $x_e \geq \frac{1}{3}$. □

Consider the tree defined by $\mathcal{L}$: Represent each set in $\mathcal{L}$ by a node in the tree. If $S, T \subset \mathcal{L}$, $S \subset T$ but S is not contained in any superset of T in $\mathcal{L}$, then the node representing S is a child of the node representing T. $\mathcal{L}$ is laminar, so this representation is well-defined. For an example, see Figure 1.

Definition 33.5 A *socket* is an edge-vertex pair (e, v) such that v is one of the endpoints of e, where $e \in E, v \in V$.

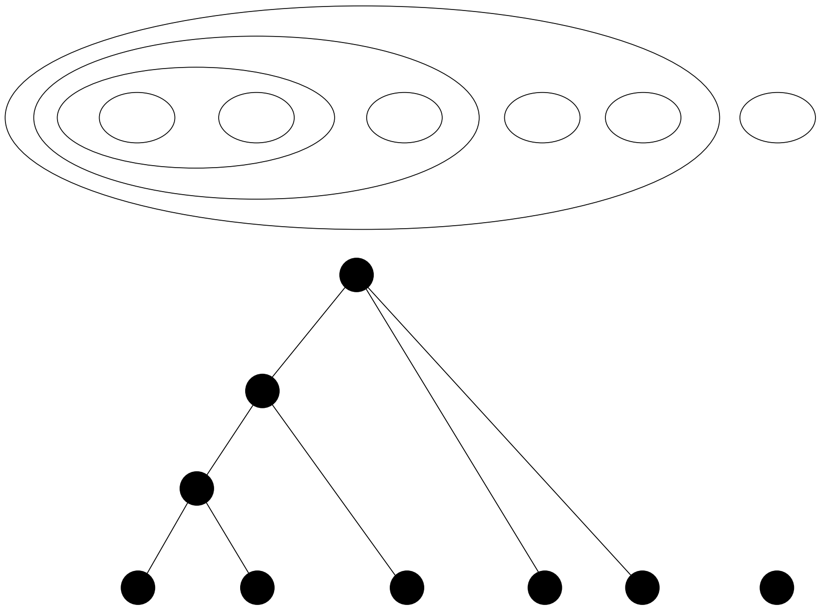


Figure 33.1: Example tree structure for a collection of laminar sets.

We can now prove the theorem.

Proof of Theorem 33.3: Suppose, for contradiction, that every $S \in \mathcal{L}$ has at least 4 edges in $\delta(S)$. There can be no more than $2m$ distinct sockets in the graph G, (where m is the number of edges). By induction, we will exhibit a charging scheme that charges 2 sockets for every node in the tree and 4 for the root node. This yields a total charge of $2m + 2 > 2m$, since $\mathcal{L}$ has m elements (Theorem 33.2), which will lead to a contradiction.

Base Case: We can charge 4 sockets to each leaf in the tree: For a leaf node on the tree corresponding to a set S, $\delta(S) \geq 4$, so we can take 4 edges out of $\delta(S)$ and their corresponding endpoints in S as sockets, and charge these sockets to S.

Inductive Step: For any parent, assume inductively that in each child's subtree, 2 sockets are charged per non-root node, and 4 sockets are charged to the root.

If there are 2 or more children, then each child can pass a charge of 2 to the parent, and the induction holds.

If there is exactly one child, the we have a few cases. If the parent has at least two additional sockets, then the child can pass a charge of 2 to the parent, and the induction holds. If the parent has no additional sockets, then $\delta(P) = \delta(C)$, where P is the parent set and C is the single child set. Therefore, $\chi_{\delta(P)} = \chi_{\delta(C)}$, which implies that $\chi_{\delta(P)}, \chi_{\delta(C)}$ cannot be linearly independent (violating a condition of $\mathcal{L}$), so this case cannot occur.

The only other case is if the parent P has exactly one child C, and exactly one additional socket, associated with some edge, e. In that case, we have either $f(P) = f(C) - x_e$ (if e is an edge between P and C) or $f(P) = f(C) + x_e$ (if e is an edge out of P), since P and C

are both tight. The function f is integer valued and x_e's are strictly fractional, so this case cannot occur.

Therefore, we can charge the sockets to elements of the tree satisfying the condition that 2 sockets are charged to each non-root node and 4 sockets are charged to the root. This gives a total charge of $2m + 2$, yielding the contradiction. □

An interesting open question is that of finding a combinatorial 2-approximation algorithm for SNDP.

33.2 The multicommodity flow problem

After studying max-flow, min-cost circulations and generalized flows, we now move on to an even more complex type of network problem: multicommodity flow. As suggested by the name, in this problem we wish to move multiple commodities between different source-sink pairs in the graph.

Multicommodity flow

- **Input:**
  - A directed graph $G = \{V, A\}$
  - A set of k source-sink pairs: s^a-t^a for $a = 1, \ldots, k$
  - integer capacities $u_{ij} \geq 0$ for all $(i, j) \in A$.
  - (optional) A set of k demands d^a for $a = 1, \ldots, k$.

For each $a = 1, \ldots, k$, let f^a be a valid s^a-t^a flow in G (f^a satisfies capacity constraints and flow conservation at vertices other than s_a and t_a). Then the f^a are a *multicommodity flow* if for all $(i, j) \in A$, $\sum_{a=1}^{k} f_{ij}^a \leq u_{ij}$.

We define the value of flows in the usual way:

$$|f^a| = \sum_{(s^a,i)\in A} f^a_{s^a i} - \sum_{(i,s^a)\in A} f^a_{is^a}$$

There are several potential goals for this problem:

1. Feasibility: Determine if there exist flows f^a such that $|f^a| = d_a$ for all a.
2. Maximum multicommodity flow: Maximize the total flow value $\sum_{a=1}^{k} |f^a|$ (ignoring the demands d_a).
3. Maximum concurrent flow: Find the maximum λ such that $|f^a| \geq \lambda d_a$ for all a.

Next lecture, we will consider approximation algorithms for goal 2: finding a maximum multicommodity flow.

ORIE 633 Network Flows April 19, 2004

Lecture 34

Lecturer: David P. Williamson *Scribe: Mateo Restrepo*

34.1 The multicommodity flow problem

34.1.1 Definition

In the last lecture we introduced the multicommodity flow problem. A multicommodity flow is a flow consisting of several independent components, the commodities. Each commodity has its own source and sink pair, but all of them share the same set of arcs for its transportation. In this lecture we are going to focus on the version of the problem summarized below.

Multicommodity flow

- **Input:**
  - A directed graph $G = \{V, A\}$
  - Integer capacities u_{ij} for all $(i,j) \in A$
  - k source-sink pairs: s^a-t^a for $a = 1, \dots, k$
- **Goal:** Find a set of functions $\{f^a : A \to \mathbb{R}, a = 1, \dots, k\}$ such that:
  - $\forall a = 1, \dots, k$, f^a is a valid flow from s^a to t^a, i.e it satisfies conservation constraints at all vertices except s_a and t_a.
  - The total flow respects capacities i.e. $\forall (i,j) \in A, \sum_{a=1,\dots,k} f^a_{i,j} \leq u_{i,j}$
  - The total flow is maximized. This flow is equal to $\sum_{a=1,\dots,k} |f^a|$ where $|f^a| = \sum_{(s^a,j)\in A} f^a_{s^a,j} - \sum_{(j,s^a)\in A} f^a_{j,s^a}$

Another version of the problem is to decide whether or not there exists a flow which satisfies a set of given demands d_a, $a = 1, \dots, k$, that is, for which $|f^a| = d_a$, for all a.

In still another one, known as the max-concurrent flow problem, the goal is to maximize a parameter λ such that $|f^a| \geq \lambda d_a$.

34.1.2 Linear programming formulation

The multicommodity flow problem has a very simple formulation as a linear program if we use the path formulation for flows. The resulting LP will actually contain exponentially many variables and we will not worry about solving it directly. Its usefulness comes from the fact that the dual problem has only m variables and its feasibility is easily checked. The dual will also provide us some intuition about the algorithm we are going to present.

Before proceeding we introduce some notation. Let the variable X_P represent the total flow along path P. We let $\mathcal{P}^a$ be the set of all paths P from s^a to t^a, and $\mathcal{P} = \bigcup_{a=1\ldots k} \mathcal{P}^a$.

The LP formulation of the max commodity flow problem is the following:

$$\max \sum_a \left(\sum_{P \in \mathcal{P}^a} X_P \right)$$

$$\forall (i,j) \in A, \quad \sum_a \sum_{P \in \mathcal{P}^a, (i,j) \in P} X_P \leq u_{ij}$$

$$\forall P \in \mathcal{P}, X_P \geq 0.$$

The dual version of this LP is:

$$\min \sum_{(i,j) \in A} u_{ij} l_{ij}$$

$$\forall P \in \mathcal{P}, \sum_{(i,j) \in P} l_{ij} \geq 1$$

$$\forall (i,j) \in A, l_{ij} \geq 0.$$

In this program l might be viewed as an arc length function, in which case $\sum_{(i,j)\in P} l_{ij}$ is the length of path P. Checking feasibility of the dual is equivalent to checking, for every a, that the length of the shortest path between s_a and t_a is at least 1. This can be easily done in polynomial time.

The comments above imply also that the dual problem is solvable in polynomial time using the ellipsoid method. Nevertheless, those methods are computationally expensive and the approximation algorithm that we are about to present might perform better in practice.

34.1.3 The Garg-Könemann approximation algorithm

The preceding discussion motivates the following algorithm for finding an approximate solution to our problem.

ε-approximate maximum multicommodity flow (Garg & Könemann, 1998)

```
$X_P \leftarrow 0 \; \forall P \in \mathcal{P}$
$l_{i,j} \leftarrow \delta \; \forall (i,j) \in A$      $(*)$
      // below we will discuss what an appropriate value of $\delta$ might be.
while $\exists P \in \mathcal{P}$ s.t. $l(P) = \sum_{(i,j)\in P} l_{i,j} < 1$
      Let $P$ a path such that $l(P) < 1$      $(*)$
      $u \leftarrow \min_{(i,j)\in P} u_{ij}$
      $X_P \leftarrow X_P + u$
      $\forall (i,j) \in P, l_{ij} \leftarrow l_{ij}(1 + \varepsilon \frac{u}{u_{ij}})$
Pick $M$ such that $\frac{X}{M}$ is feasible      $(*)$
      // below we will discuss how to calculate $M$ appropiately.
return $\frac{X}{M}$
```

To simplify the initial exposition we have left three steps of the algorithm – marked with an $(*)$ – unspecified. Later we will describe in detail how to determine optimal values for δ and M, and how to pick the path P so as to guarantee both polynomial running time of the algorithm and $(1 - 2\varepsilon)$ optimality of the resulting multicommodity flow.

Note that this algorithm is quite different from previous flow algorithms that we have considered. We are not using the notion of a residual graph. Our solution X while running the main loop is not even necessarily feasible; it is quite possible that the flow on an edge exceeds its capacity. Thus we scale down the flow at the end of the algorithm to ensure that the solution we return is a feasible flow.

We also notice that our problem has no integrality property. Thus we have to use some other argument to can use this fact to guarantee that the algorithm will end. This is what we do next.

Lemma 34.1 The algorithm terminates after at most $m \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}$ iterations.

Proof: Initially, for all $(i,j) \in A, l_{ij} = \delta$. At any point in the algorithm $l_{ij} \leq 1 + \varepsilon$. Indeed, l_{ij} only changes if it is in a path P of length $l(P) < 1$. Since all edges have positive length, this means that $l_{ij} < 1$. Furthermore, l_{ij} is increased by a factor that is not above $1 + \varepsilon$ (since by definition $u \leq u_{ij}$) so it can't become greater than $1 + \varepsilon$.

Also, at each iteration at least one edge has its length augmented by a factor of $1 + \varepsilon$. Call this edge a **tight edge** for that iteration. If a given edge e is the tight edge for i_e iterations then its length after the i_e-th such iteration is $\delta\,(1+\varepsilon)^{i_e}$ and since this quantity is no bigger than $(1+\varepsilon)$ we conclude that $i_e \leq \log_{1+\varepsilon} \frac{(1+\varepsilon)}{\delta}$, which imposes a bound on the total number of iterations for which e can be the tight edge. Since this bound is the same for all edges we conclude that the total number of iterations is no more than $m \log_{(1+\varepsilon)} \frac{(1+\varepsilon)}{\delta}$
□

We now show that if we scale the flow by a fixed quantity, the flow becomes feasible.

Lemma 34.2 If we scale flows f^a by $M = \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}$ then the total flow becomes feasible.

Proof: Fix an edge (i, j). At each iteration k, if $(i, j) \in P_k$ where P_k is the selected path, the flow on this edge (i, j) is increased by u_k. If we set $a_k = \frac{u_k}{u_{ij}} \leq 1$, the length l_{ij} is increased by a factor of $1 + a_k\varepsilon$. At the end, l_{ij} is increased by a factor of $\prod_{k:(i,j)\in P_k}(1 + a_k\varepsilon)$. The flow on these edges, on the other hand, is increased by $\sum_{k:(i,j)\in P_k} u_k = u_{ij} \sum_{k:(i,j)\in P_k} a_k$, starting from 0. Since initially $l_{ij} = \delta$, and at the end $l_{ij} < 1 + \varepsilon$, we have

$$\delta \prod_{k:(i,j)\in P_k} (1 + a_k\varepsilon) < 1 + \varepsilon.$$

Since $a_k \leq 1, 1 + a_k\varepsilon \geq (1 + \varepsilon)^{a_k}$ so that

$$\delta(1 + \varepsilon)^{\sum_{k:(i,j)\in P_k} a_k} < 1 + \varepsilon$$

$$\sum_{k,(i,j)\in P_k} a_k < \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta} = M.$$

Thus since the total amount of flow on edge (i, j) is $u_{ij} \sum_{k:(i,j)\in P_k} a_k$, if we divide the flows by M, the total amount of flow on edge (i, j) will be no more than u_{ij}, and the flow will be feasible. $\square$

Next time we will prove

Theorem 34.3 The algorithm computes a $1 - 2\varepsilon$ approximate flow.

ORIE 633 Network Flows April 21, 2004

Lecture 35

Lecturer: David P. Williamson *Scribe: Sam Steckley*

35.1 The multicommodity flow problem

35.1.1 The Garg-Könemann approximation algorithm (cont.)

Recall the multicommodity flow problem discussed last class.

Multicommodity flow

- **Input:**
  - A directed graph $G = \{V, A\}$
  - Integer capacities u_{ij} for all $(i, j) \in A$
  - k source-sink pairs: s^a-t^a for $a = 1, \dots, k$
- **Goal:** Find a set of functions $\{f^a : A \to \mathbb{R}, a = 1, \dots, k\}$ such that:
  - $\forall a = 1, \dots, k$, f^a is a valid flow from s^a to t^a, i.e it satisfies conservation constraints at all vertices except s_a and t_a.
  - The total flow respects capacities i.e. $\forall (i,j) \in A, \sum_{a=1,\dots,k} f^a_{i,j} \le u_{i,j}$
  - The total flow is maximized. This flow is equal to $\sum_{a=1,\dots,k} |f^a|$ where $|f^a| = \sum_{(s^a,j)\in A} f^a_{s^a,j} - \sum_{(j,s^a)\in A} f^a_{j,s^a}$

We defined the following notation. Let the variable X_P represent the total flow along path P. We let $\mathcal{P}^a$ be the set of all paths P from s^a to t^a, and $\mathcal{P} = \bigcup_{a=1\dots k} \mathcal{P}^a$.

The LP formulation of the max commodity flow problem is the following:

$$\max \sum_a \left(\sum_{P \in \mathcal{P}^a} X_P \right)$$

$$\forall (i,j) \in A, \quad \sum_a \sum_{P \in \mathcal{P}^a, (i,j) \in P} X_P \le u_{ij}$$

$$\forall P \in \mathcal{P}, X_P \ge 0.$$

The dual version of this LP is:

$$\min \sum_{(i,j)\in A} u_{ij} l_{ij}$$

$$\forall P \in \mathcal{P}, \sum_{(i,j)\in P} l_{ij} \geq 1$$
$$\forall (i,j) \in A, l_{ij} \geq 0.$$

We gave the following algorithm for the maximum multicommodity flow problem.

ε-approximate maximum multicommodity flow (Garg & Könemann, 1998)

```
$X_P \leftarrow 0 \;\forall P \in \mathcal{P}$
$l_{i,j} \leftarrow \delta \;\forall (i,j) \in A$
while $\exists P \in \mathcal{P}$ s.t. $l(P) = \sum_{(i,j)\in P} l_{i,j} < 1$
    Let $P$ a path such that $l(P) < 1$      $(*)$
    $u \leftarrow \min_{(i,j)\in P} u_{ij}$
    $X_P \leftarrow X_P + u$
    $\forall (i,j) \in P, l_{ij} \leftarrow l_{ij}(1 + \varepsilon \frac{u}{u_{ij}})$
Pick $M$ such that $\frac{X}{M}$ is feasible
return $\frac{X}{M}$
```

We proved the following lemmas:

Lemma 35.1 The algorithm terminates after at most $m \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}$ iterations.

Lemma 35.2 If we scale flows f^a by $M = \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}$ then the total flow becomes feasible.

Note that in the algorithm we choose an arbitrary path $P \in \mathcal{P}$ s.t. $l(P) = \sum_{(i,j)\in P} l_{i,j} < 1$ on which to augment. Now suppose we choose P to be the shortest path. That is to say, choose $P \in \mathcal{P}$ s.t. $P = \operatorname{argmin} l(P)$.

Given the slightly modified algorithm, we now can show the algorithm gives a $1 - 2\varepsilon$ approximate flow.

Theorem 35.3 The algorithm computes a $1 - 2\varepsilon$ approximate flow.

Proof:

A few definitions:

- For length function l, we'll set $D(l) = \sum_{(i,j)\in A} u_{i,j} l_{i,j}$ (dual objective function) and $\alpha(l) = \min_{P\in\mathcal{P}} l(P)$.
- we'll note l^s the length function at the end of iteration s.
- we'll also note $D(s) = D(l^s)$ and $\alpha(s) = \alpha(l^s)$.
- we'll set $\beta = \min_{l \text{ feasible}} D(l) = \min_{l\geq 0, \alpha(l)\neq 0} \frac{D(l)}{\alpha(l)}$. This equality comes from the fact that if you divide a positive length function by its corresponding shortest path, the new shortest path becomes 1 so the length function becomes feasible.

- we'll set $X^s = \sum_{P \in \mathcal{P}} X_P^s$, primal value at the end of iteration s.
- t is the index of the last iteration.

By definition of t, $1 \le \alpha(t)$. We'll assume for now (we'll prove it later) that

$$\alpha(t) \le \delta n e^{\varepsilon \frac{X^t}{\beta}}.$$

We then have:

$$\frac{X^t}{\beta} \ge \frac{\ln(\frac{1}{\delta n})}{\varepsilon}.$$

To get the result we must show that:

$$\frac{X^t}{M} \ge (1-2\varepsilon)\beta.$$

We will set $\delta = (1+\varepsilon)((1+\varepsilon)n)^{-\frac{1}{\varepsilon}}$. This value is chosen so that $\frac{\ln(\frac{1}{\delta n})}{M} = (1-\varepsilon)\ln(1+\varepsilon)$. By substitution,

$$\begin{aligned}\frac{X^t}{M\beta} &\ge \frac{\ln(\frac{1}{\delta n})}{M\varepsilon} \\ &\ge \frac{(1-\varepsilon)\ln(1+\varepsilon)}{\varepsilon}.\end{aligned}$$

By a Taylor series argument,

$$\begin{aligned}\frac{(1-\varepsilon)\ln(1+\varepsilon)}{\varepsilon} &\ge \frac{(1-\varepsilon)(\varepsilon-\varepsilon^2/2)}{\varepsilon} \\ &\ge (1-2\varepsilon).\end{aligned}$$

Now we return to the inequality we assumed above:

$$\alpha(t) \le \delta n e^{\varepsilon \frac{X^t}{\beta}}.$$

To show this is true, we consider how the dual objective function changes from iteration to iteration. Let P_s be the shortest path on which we augment in iteration s. For an arbitrary iteration s,

$$\begin{aligned}D(s) &= \sum u_{i,j} l_{i,j}^s \\ &= \sum_{(i,j)\in P_s} u_{i,j} l_{i,j}^{s-1}(1+\varepsilon\frac{u}{u_{i,j}}) \\ &= D(s-1) + \varepsilon u \sum_{(i,j)\in P_s} l_{i,j}^{s-1} \\ &= D(s-1) + \varepsilon(X^s - X^{s-1})\alpha(s-1).\end{aligned}$$

Thus we have in iteration s that

$$D(s) = D(0) + \varepsilon \sum_{h=1}^{s} (X^h - X^{h-1})\alpha(h-1).$$

We are looking for a bound on β. If we consider length function $l^s - l^0$, since β is minimum we have

$$\beta \leq \frac{D(l^s - l^0)}{\alpha(l^s - l^0)}$$

D is linear so $D(l^s - l^0) = D(s) - D(0)$. Now, $\alpha(l^s - l^0)$ is the length of some path P. Then $\alpha(l^s - l^0) = l^s(P) - l^0(P) \geq \alpha(s) - \delta n$ since for any P, $\alpha(s) \leq l^s(P)$ and $\delta n \geq l^0(P)$ (which follows from the observations that l^0 is δ on every edge, and P has less than n edges). Then we have that

$$\begin{aligned}
\beta &\leq \frac{D(s) - D(0)}{\alpha(s) - \delta n} \\
&\leq \frac{\varepsilon \sum_{h=1}^{s} (X^h - X^{h-1})\alpha(h-1)}{\alpha(s) - \delta n}.
\end{aligned}$$

Rearranging terms, we have that

$$\beta(\alpha(s) - \delta n) \leq \varepsilon \sum_{h=1}^{s} (X^h - X^{h-1})\alpha(h-1),$$

or that

$$\alpha(t) \leq \delta n + \frac{\varepsilon}{\beta} \sum_{h=1}^{s} (X^h - X^{h-1})\alpha(h-1).$$

Let $\alpha'(s)$ be the maximum possible value of $\alpha(s)$ given the above equation, for $1 \leq s \leq t$. Let $\alpha'(0) = \delta n$. Then we have that

$$\begin{aligned}
\alpha'(0) &= \delta n \\
\alpha'(1) &= \delta n + \frac{\varepsilon}{\beta}(X^1 - X^0)\alpha'(0) = (1 + \frac{\varepsilon}{\beta}(X^1 - X^0))\alpha'(0) \\
\alpha'(2) &= \delta n + \frac{\varepsilon}{\beta}((X^2 - X^1)\alpha'(1) + (X^1 - X^0)\alpha'(0)) \\
&= (1 + \frac{\varepsilon}{\beta}(X^1 - X^0))\alpha'(0) + \frac{\varepsilon}{\beta}(X^2 - X^1)\alpha'(1) \\
&= (1 + \frac{\varepsilon}{\beta}(X^2 - X^1))\alpha'(1).
\end{aligned}$$

In general we obtain that

$$\begin{aligned}
\alpha'(s) &= (1 + \frac{\varepsilon}{\beta}(X^s - X^{s-1}))\alpha'(s-1) \\
&\leq e^{\frac{\varepsilon}{\beta}(X^s - X^{s-1})}\alpha'(s-1).
\end{aligned}$$

Then applying the bound repeatedly, we get that

$$\alpha'(s) \leq \alpha'(0)e^{\frac{\varepsilon}{\beta}(X^s - X^0)}.$$

So then

$$\alpha(t) \leq \alpha'(t) \leq \alpha'(0)e^{\frac{\varepsilon}{\beta}(X^t - X^0)}.$$

Since $X^0 = 0$ and $\alpha'(0) = \delta n$,

$$\alpha(t) \leq \delta n e^{\frac{\varepsilon}{\beta}X^t}.$$

□

ORIE 633 Network Flows April 23, 2004

Lecture 36

Lecturer: David P. Williamson *Scribe: Sam Steckley*

36.1 Multicommodity flow

36.1.1 A dynamic, local control algorithm

In designing algorithms thus far, we have been assuming both global knowledge of the given network and that the given network is static. These assumptions may fail to hold. Consider a large computer network. In this case the network may be dynamic. Nodes or communication arcs may fail. New nodes or arcs may be added. In addition, the network may be so large that a complete, global description of the network is unwieldy. We now consider an algorithm which assumes only local knowledge of a possibly dynamic network.

The goal of the previous multicommodity flow algorithm was to obtain a maximum multicommodity flow. In introducing multicommodity flows, we discussed other possible objectives. One possible objective is finding a feasible multicommodity flow such that $|f^a| = d_a \, \forall a$ where d_a is the demand for commodity a. The following theorem states that there is a local control algorithm that achieves this goal provided the given network supports a flow that can do a bit better. We state the theorem here without proof.

Theorem 36.1 (Awerbuch, Leighton '94) There exists a local control algorithm to compute a multicommodity flow f s.t. $|f^a| = d_a \, \forall a$ if there exists a flow g s.t. $|g^a| \geq (1 + 3\varepsilon)d_a \, \forall a$

So although we don't assume need to assume global knowledge of the network or assume that the network is static, we do need to assume that the network can push through a flow of a certain value.

36.1.2 Definitions and assumptions

In this algorithm, we maintain a queue at each vertex i for each arc $(i, j) \in A$ and each commodity a. For vertex i, arc $(i, j) \in A$, and commodity a, let q^a_{ij} denote the length of this queue. Ignoring capacity constraints, q^a_{ij} is the amount of commodity a at vertex i that could be pushed along arc (i, j) to vertex j.

For each source s_a we assume that there is only one arc out of s_a. Similarly, for each sink t_a we assume that there is only one arc into t_a. If this were not true, we could easily alter our network so that it was. With only one edge out of each of the sources and with only one edge into each of the sinks we can denote the length of the queues at the source and sink for any commodity a as q_{s_a} and q_{t_a}, respectively.

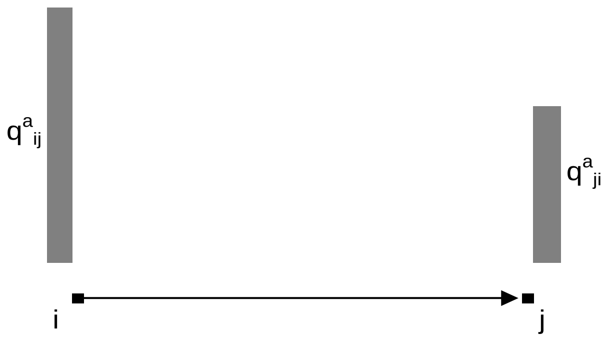


Figure 36.1: Queues associated with commodity a and arc (i, j)

At source s_a, we bound the queue height by Q_a, which will be determined later. The remaining flow at the source s_a is held in an overflow buffer. Let b_a be the amount of flow in the buffer for the commodity a.

The algorithm will use the following potential functions:

- potential of a queue is $\phi_a(x) = e^{\alpha_a x}$ for x units of flow of commodity a
- potential of overflow buffer is $\sigma_a(x) = \phi'_a(Q_a)x = \alpha_a x e^{\alpha_a Q_a}$ for x units of flow of commodity a

The constant α_a will be given later.

36.1.3 Algorithm

We now present the algorithm.

Dynamic, local control algorithm (Awerbuch & Leighton, 1994)

```
Repeat forever
    Phase 1: Add flow to sources:
        $b_a \leftarrow b_a + (1+\epsilon)d_a \quad \forall a$
        Move up to $Q_a$ flow from buffer to source queue
    Phase 2: Push flow on edges:
        For each arc $(i,j) \in A$
            Compute $f_{ij}^a$ to minimize
                $\sum_a \phi_a(q_{ij}^a - f_{ij}^a) + \phi_a(q_{ji}^a + f_{ij}^a)$
                s.t. $\sum_a f_{ij}^a \leq u_{ij}$
            Move $f_{ij}^a$ from $q_{ij}^a$ to $q_{ji}^a$
    Phase 3: Zero out flows at sinks: $q_{t_a} \leftarrow 0 \, \forall a$
    Phase 4: Balance the queues at the nodes:
        $q_{ij}^a \leftarrow \frac{1}{deg(i)} \sum_{j:(i,j) \in A} q_{ij}^a \quad \forall a, (i,j) \in A$
```

In Phase 2, flow is pushed along arc (i,j) to minimize the total potential of the queues at i and j. By the convexity of the node potentials this minimization tends to push flow from nodes with high potentials to nodes with low potentials. So the flow moves downhill. In Phase 1, flow is added to the sources, increasing potentials at the sources. In Phase 3, if any flow has reached the sinks, we empty it, so the sink potentials stay small ($\phi_a(0) = e^{\alpha_a * 0} = 1$). We note that the potential function is convex, so the balancing of the queues at the nodes only decreases the overall potential function value for each commodity at each node. Overall, the algorithm maintains high source potentials and low sink potentials so that the flow will run downhill from the sources to the sinks.

The outline of the analysis of this algorithm is as follows. First, we show the increase in potentials in Phase 1 is not too big. Then we show the decrease in potentials resulting from Phases 2 and 3 cancels the increase from Phase 1. This implies overall potential is bounded, which in turn implies the total flow in the queues and buffer is bounded. Therefore, flow must be reaching the sink.

The first lemma presented gives an upper bound on the potential increase in Phase 1. The second gives a lower bound on the potential decrease from Phases 2 and 3. We will prove these lemmas later.

Lemma 36.2 The potential increase in Phase 1 is at most $(1+\varepsilon)d_a\phi_a'(q_{s_a})$ for commodity a, where q_{s_a} is the height of source for commodity a after Phase 1.

Lemma 36.3 The potential decrease in Phase 2 and 3 is at least

$$(1 + \frac{3}{2}\varepsilon - \varepsilon^2)\sum_a d_a\phi_a'(q_{s_a}) - \frac{\varepsilon k}{8n}(1 + 2\varepsilon).$$

Observation 36.1 Phase 4 can only decrease the total potential.

Using the two lemmas and the observation above, we can bound the total potentials.

Lemma 36.4 The total potential is at most $2m\sum_a \phi_a(Q_a)$.

Proof: Suppose for some a', $q_{s'_a} = Q'_a$. By previous lemmas, the total decrease in potentials is

$$\begin{aligned}
&= (1+\frac{3}{2}\varepsilon-\varepsilon^2)\sum_a d_a\phi'_a(q_{s_a}) - \frac{\varepsilon k}{8n}(1+2\varepsilon) - (1+\varepsilon)\sum_a d_a\phi'_a(q_{s_a}) \\
&= (1+\frac{3}{2}\varepsilon-\varepsilon^2-(1+\varepsilon))d_{a'}\phi'_{a'}(Q_{a'}) - \frac{\varepsilon k}{8n}(1+2\varepsilon) + (1+\frac{3}{2}\varepsilon-\varepsilon^2-(1+\varepsilon))\sum_{a:a\neq a'} d_a\phi'_a(q_{s_a}) \\
&= (\frac{1}{2}\varepsilon-\varepsilon^2)d_{a'}\phi'_{a'}(Q_{a'}) - \frac{\varepsilon k}{8n}(1+2\varepsilon) + (\frac{1}{2}\varepsilon-\varepsilon^2)\sum_{a:a\neq a'} d_a\phi'_a(q_{s_a}) \\
&\geq (\frac{1}{2}\varepsilon-\varepsilon^2)d_{a'}\phi'_{a'}(Q_{a'}) - \frac{\varepsilon k}{8n}(1+2\varepsilon) \\
&= \frac{\varepsilon}{2}(1-2\varepsilon)d_{a'}\alpha_{a'}e^{\alpha_{a'}Q_{a'}} - \frac{\varepsilon k}{8n}(1+2\varepsilon)
\end{aligned}$$

Set $\alpha_a = \frac{\varepsilon}{8nd_a}$ and $Q_a = \frac{1}{\alpha_a}\ln(\frac{2k(1+2\varepsilon)}{\varepsilon(1-2\varepsilon)})$. Substituting into the above gives

$$\begin{aligned}
&= \frac{\varepsilon}{2}(1-2\varepsilon)d_{a'}\frac{\varepsilon}{8nd_a}\frac{2k(1+2\varepsilon)}{\varepsilon(1-2\varepsilon)} - \frac{\varepsilon k}{8n}(1+2\varepsilon) \\
&= 0
\end{aligned}$$

Otherwise, $q_{s_a} < Q_a\,\forall a$. Then all overflow buffers are empty and overflow buffer potentials are zero. Therefore total potential for each commodity a is at most $2m\phi_a(Q_a)$ since the queue heights are at most Q_a which follows from Phase 2 and the convexity of the potential function.

The lemma then follows by induction on the algorithm. The statement of the lemma holds initially since $(1+\varepsilon)d_a \leq Q_a$. Assume the lemma statement holds for the previous iteration. Either for some a, $q_{s_a} = Q_a$ or $\forall a$, $q_{s_a} < Q_a$. In the first case we showed that potentials do not increase so statement is true in the current iteration by the inductive hypothesis. In the second case we showed the statement of lemma is true for the current iteration.

□

ORIE 633 Network Flows April 26, 2004

Lecture 37

Lecturer: David P. Williamson *Scribe: Sumit Kunnumkal*

37.1 Multicommodity flow

37.1.1 A dynamic, local control algorithm (cont.)

In this lecture, we complete the analysis of the dynamic local control algorithm.

```
Dynamic local control algorithm

    Phase 1: Add $(1+\epsilon)d_a$ units of flow to buffers $b_a$. Move as much flow as possible
             to the source queue $q_a$.
    Phase 2: For each edge, move flow across edge to minimize queue potentials.
    Phase 3: Zero out flow at sinks.
    Phase 4: Balance queues at nodes.
```

Recall that the queue height at the source is bounded by Q_a. We defined the potential of a queue, $\Phi_a(x) = e^{\alpha_a x}$ and the potential of the overflow buffer, $\sigma_a(x) = x\Phi'_a(Q_a)$. In the above expressions, x is the units of flow of the commodity and α_a is a predefined constant. We also assumed that there was only one edge coming out of source s_a and one edge going into sink t_a. Last time we proved the following lemma:

Lemma 37.1 The total potential $\leq 2m\sum_a \Phi_a(Q_a)$.

Using this lemma, we obtain a bound on the potential of the overflow buffer and hence the bound the height of the overflow buffer. We then bound the total amount of a commodity present in the system. Finally, we use this bound along with the fact that at every iteration $(1+\epsilon)d_a$ units is put into the system to prove the theorem of Awerbuch and Leighton '94.

The potential of the overflow buffer is lower than the total potential of the system. So,

$$\sigma(b_a) \leq 2m\sum_{a'} \Phi_{a'}(Q_{a'}),$$

i.e.,

$$b_a\alpha_a e^{\alpha_a Q_a} \leq 2m\sum_{a'} e^{\alpha_{a'}Q_{a'}}.$$

Noting that $\alpha_a = \frac{\epsilon}{8nd_a}$ and $Q_a = \frac{1}{\alpha_a}\ln(\frac{2k(1+2\epsilon)}{\epsilon(1-2\epsilon)})$, we have that $e^{\alpha_{a'}Q_{a'}}$ is the same for all a'. Thus the height of the overflow buffer $b_a \leq \frac{2mk}{\alpha_a}$. Since all intermediate queues for commodity q are at most Q_a, we can now bound the total amount of commodity a in the system at any given time.

$$\begin{aligned}\text{Total amount of commodity } a \text{ in system} &\leq b_a + 2mQ_a \\ &\leq \frac{2mk}{\alpha_a} + \frac{2m}{\alpha_a}\ln\left(\frac{2k(1+2\epsilon)}{\epsilon(1-2\epsilon)}\right) \\ &= O\left(\frac{d_a nm(k+\ln(k/\epsilon))}{\epsilon}\right).\end{aligned}$$

Note that in the first inequality, the term $2mQ_a$ comes from there being two queues for a commodity at each edge and the amount in each being bounded by Q_a.

Since $(1+\epsilon)d_a$ units of commodity is put into the system at every iteration, the above bound implies that after $R = O(\frac{nm(k+\ln(k/\epsilon))}{\epsilon^2})$ iterations, only $\epsilon/(1+\epsilon)$ of total flow put into the network is still in the network. Therefore, over R rounds, $d_a R$ units of flow made it from s_a to t_a. Averaging over R rounds, we get a flow f^a such that $|f^a| = d_a$. This completes the proof of the theorem of Awerbuch and Leighton. We return to the proofs of two lemmas we had stated last time.

Lemma 37.2 The potential increase in Phase 1 $\leq (1+\epsilon)d_a\Phi_a^{'}(q_{s_a})$ per commodity a.

Proof: Let q be the initial height of the source queue, b the initial height of the overflow buffer, and q_{s_a} the final height of the source queue for commodity a.

We will use the fact that for all x and $\delta \geq 0$,

$$\phi(x+\delta) \leq \phi(x) + \delta\phi'(x+\delta).$$

If $q + b + (1+\epsilon)d_a \leq Q_a$, then the potential increase is

$$\begin{aligned}&\phi_a(q+b+(1+\epsilon)d_a) - \phi_a(q) - \sigma_a(b) \\ &\leq (b+(1+\epsilon)d_a)\phi_a'(q+b+(1+\epsilon)d_a) - b\phi_a'(Q_a) \\ &\leq (1+\epsilon)d_a\phi_a'(q_{s_a}),\end{aligned}$$

where the first inequality follows by the fact above and the definition of σ_a.

If $q + b + (1+\epsilon)d_a > Q_a$, then the potential increase is

$$\begin{aligned}&\phi_a(Q_a) + \sigma_a(q+b+(1+\epsilon)d_a - Q_a) - \phi_a(q) - \sigma_a(b) \\ &\leq (Q_a - q)\phi_a'(Q_a) + (q+(1+\epsilon)d_a - Q_a)\phi_a'(Q_a) \\ &= (1+\epsilon)d_a\phi_a'(Q_a) \\ &= (1+\epsilon)d_a\phi_a'(q_{s_a}),\end{aligned}$$

where again the first inequality follows by the fact above and the definition of σ_a. □

Lemma 37.3 The potential decrease in Phases 2 and 3 is at least $(1+\frac{3\epsilon}{2}-\epsilon^2)\sum_a d_a\Phi_a'(q_{s_a}) - \frac{\epsilon k(1+2\epsilon)}{8n}$.

Proof: We use the fact that there exists a flow g with $|g^a| = (1+3\epsilon)d_a$ to get a potential decrease of at least X. Since we minimize the potentials, the decrease will be at least X.

Consider moving δ units of flow across (i,j) with t units at i and h units at j. The potential decrease $= \Phi_a(t) - \Phi_a(t-\delta) + \Phi_a(h) - \Phi_a(h+\delta)$. Using the fact that $\forall x, \delta \geq 0$

$$\Phi(x+\delta) - \Phi(x) \geq \delta\Phi'(x+\delta) - \delta\Phi''(x+\delta)$$

and

$$\Phi(x+\delta) - \Phi(x) \leq \delta\Phi'(x) + \delta^2\Phi''(x+\delta),$$

the decrease in potential on moving δ units along an arc is at least $\delta\Phi'_a(t) - \delta^2\Phi''_a(t) - \delta\Phi'_a(h) - \delta^2\Phi''_a(h+\delta)$.

What if we move δ units on a path from s_a to t_a? The $\delta\phi'_a$ terms drop out, and the sum telescopes, since we rebalance the queues and the queue height of the head of an arc is equal to the queue height of the tail of the next arc on the path. If $\hat{q_a}$ is the maximum height along the path, the decrease is at least $\delta\Phi'_a(q_{s_a}) - \delta\Phi'_a(0) - 2n\delta^2\Phi''_a(\hat{q_a}+\delta)$. Note that the second term in the above expression comes from the sink node where the flow zeroed out.

We know that there exists a flow g such that $|g^a| = (1+3\epsilon)d_a$. Let δ_i^a be flow of commodity a along the i^{th} path of g. Let $\hat{q_i^a}$ be the max queue height on a path i for commodity a. Then potential decrease is at least

$$\begin{aligned}
&\sum_a\sum_i \delta_i^a \left(\Phi'_a(q_{s_a}) - \Phi'(0) - 2n\delta_i^a\Phi''_a(\hat{q_i^a} + \delta_i^a)\right) \\
&\geq \sum_a\sum_i \delta_i^a \left(\Phi'_a(q_{s_a}) - \Phi'(0) - 2n(1+2\epsilon)d_a\Phi''_a(\hat{q_i^a} + (1+2\epsilon)d_a)\right),
\end{aligned}$$

by substitution. For $\alpha_a = \frac{\epsilon}{8nd_a}$, we have that

$$\begin{aligned}
2n(1+2\epsilon)d_a\Phi''_a(\hat{q_i^a} + (1+2\epsilon)d_a) &= 2n(1+2\epsilon)d_a\alpha_a^2 e^{\alpha_a\hat{q_1^a}} \cdot e^{\alpha_a(1+2\epsilon)d_a} \\
&= \frac{(1+2\epsilon)\epsilon}{4}\Phi'(\hat{q_i^a}) \cdot e^{\alpha_a(1+2\epsilon)d_a} \\
&\leq \frac{\epsilon}{2}\Phi'_a(\hat{q_i^a}).
\end{aligned}$$

Plugging this into the inequality above, we have that the potential decrease is at least

$$\begin{aligned}
&\sum_a\sum_i \delta_i^a \left(\left(1-\frac{\epsilon}{2}\right)\Phi'_a(q_{s_a}) - \Phi'(0)\right) \\
&\geq \sum_a\sum_i \delta_i^a \left(\left(1-\frac{\epsilon}{2}\right)\Phi'_a(q_{s_a}) - \alpha_a\right) \\
&= \sum_a (1+2\epsilon)d_a \left(\left(1-\frac{\epsilon}{2}\right)\Phi'_a(q_{s_a}) - \alpha_a\right) \\
&= (1+\frac{3}{2}\epsilon - \epsilon^2)\sum_a d_a\phi'_a(q_{s_a}) - \frac{\epsilon k(1+2\epsilon)}{8n}.
\end{aligned}$$

Since we can show that the potential can decrease by this much given the flow g, and we maximize the potential decrease on each arc, the potential must decrease by at least this much. $\square$

Note that this proof only uses that a flow *exists* at each iteration! It doesn't even have to be the *same* flow in each iteration. So if arcs disappear, reappear, change capacity, or whatever, the algorithm will still work - as long as the required flow exists in each iteration.

ORIE 633 Network Flows April 28, 2004

Lecture 38

Lecturer: David P. Williamson *Scribe: Retsef Levi*

38.1 Unsplittable flow

From this class till the end of the semester we will start discussing several more advanced and up to date topics regarding flows. In particular, we will discuss in the this lecture the *unsplittable flow problem.* We start with a rigorous definition:

Unsplittable Flow Problem

- **Input:**
  - Directed graph $G = (V, A)$
  - Integer capacity $u_{ij} \geq 0 \; \forall (i, j) \in A$
  - Specified source-sink demand pairs in V, s_1-t_1, …,s_k-t_k.
  - Integer non-negative demands $d_1, \ldots, d_k$ each corresponding to the respective source-sink pair.
- **Goal:** For each s_a-t_a $(a = 1, \ldots, k)$ find an s_a-t_a path P_a, such that for each $(i, j) \in A$, $\sum_{a:\ (i,j)\in P_a} d_a \leq u_{ij}$

In this problem we need to send the demand of each source-sink pair on a single path in a way that will respect the capacities on all arcs. This problem is related to another well known problem, namely the *the edge-disjoint paths problem on a graph.* The edge-disjoint paths problem is a special case of the unsplittable flow problem where all demands and all arc capacities are equal 1.

One of the optimization variants of the problem is where $s_a = s$ for all $a = 1, \ldots, k$ and in addition we have a cost $c_{ij} \geq 0$ for each arc $(i, j) \in A$. The goal is as defined above, but with objective of minimizing the cost.

38.1.1 NP-hardness

We now wish to show that the latter problem is NP-hard. We do that by a reduction from the *knapsack problem*, which is known to be NP-hard.

Knapsack Problem:

- **Input:**
  - A set of items I
  - Size S_i and a value $V_i \geq 0$ for each item $i \in I$
  - Knapsack size S
- **Goal:** Find a subset $I' \subseteq I$ such that $S(I') \leq S$ and $V(I')$ is maximized

Given an input for the knapsack problem, we construct the following instance of the unsplittable flow problem with costs. We construct the following graph:

- A source s, an intermediate node s', and a node i for each item $i \in I$, each with demand $d_i = S_i$.
- An arc (s, s') with capacity S and cost 0. An arc (s, i) for each $i \in I$ with infinite capacity and cost V_i/S_i. Finally have arc (s', i) for each $i \in I$ with cost 0 and infinite capacity.

We now solve the constructed instance of the unsplittable flow problem with costs. Clearly if we minimize the value of all items not taken in the knapsack, this is equivalent to the knapsack problem defined above. It is readily seen that any solution to the unsplittable flow problem corresponds to a solution to the knapsack problem with the same cost. All demands satisfied through the arc (s, s') ($s - s' - i$ path) correspond to items taken in the knapsack (hence they incur no cost). On the other hand each demand i satisfied through an (s, i) arc, corresponds to an item not taken, and incurs a cost of $S_i V_i / S_i = V_i$. Conversely, any solution to the knapsack problem induces a solution to the flow problem with the same cost (again use $s - s' - i$ to satisfy all demands that correspond to items taken in knapsack, and satisfy all other demands using the (s, i) arcs). Thus the two problems are equivalent, and so if there exists a poly-time algorithm to solve the unsplittable flow problem with costs, there exists a poly-time algorithm to solve the knapsack problem, which is unlikely unless $P = NP$.

38.1.2 Decision version

We now consider a decision variant of the former problem in which we wish to answer the question of whether there exists an unsplittable flow of cost $\leq B$. Naturally, this is an NP-complete problem. So it is natural to talk on approximation algorithms of some sort.

Observe that we already know that the splittable flow problem is solvable in poly-time, and even more important, the optimal solution has an *integrality property*. Here we use the term *splittable* flow to refer to the problem, when we do not force the flow of any source-sink pair to be shipped on a single path. This is equivalent to the term *fractional* that was used in class.

Having this in mind we can think about the following relaxed procedure. First we try to find an optimal splittable flow. Now clearly if the optimal splittable flow is of cost $> B$, we know that the answer to the above decision problem is no. Suppose now that the optimal splittable flow is of cost $\leq B$. We then wish to find an approximated unsplittable flow. Here the notion of approximation can refer to two aspects:

- Finding an unsplittable flow, where the capacity constraint of each arc (i, j) is not violated by more that a factor of α ($\alpha > 1$). Naturally, our goal would be to make α as small as possible.
- Instead of sending the flow in one time, we send it in rounds, where in each round we respect the capacities on all arcs (e.g, in first round we satisfy demands 2,5,9, then in second round demands 1,3,4 and then in third round demands 6,7,8, etc). Here, we wish to minimize the number of rounds.

38.1.3 An algorithm

Along the lines described above, we will present an algorithm due to Skutella. The following theorem describes the performance guarantees of the algorithm (proof will be shown in next lecture).

From now on we will assume that $\max_a d_a \leq \min_{(i,j)\in A} u_{ij}$.

Theorem 38.1 *(Skutella 2000).* Assume there exists a splittable flow of cost $\leq B$. Then in poly-time one can find an unsplittable flow such that one of the following holds:

(i) The flow over any arc $(i, j) \in A$ is $\leq 3u_{ij}$.

(ii) The flow can be sent in at most 8 rounds.

The key idea underlying the theorem is to increase capacities and re-route the splittable flows, which can again assumed to have the integrality property. Moreover, observe that if $d_a = d$ and all capacities are multiples of d, then the unsplittable flow problem is reduced to the regular min-cost flow problem (we can scale capacities and measure them as multiple of d and scale all demands to be 1).

Having this in mind, we will first describe an algorithm for a special case. Let $d_{min} := \min_a d_a$, $d_{max} := \max_a d_a$ and $u_{min} := \min_{(i,j)\in A} u_{ij}$. Now assume that for each $a = 1, \ldots, k$ $d_a := d_{min} 2^q$ for some $q \in N$.

Algorithm

```
$q \leftarrow 0$
Compute splittable flow $f^0$
while $d_{min}2^q \leq d_{max}$
    $q \leftarrow q+1$, $\delta_q \leftarrow 2^q d_{min}$
    Set $u_{ij}^q$ to $f_{ij}^q$ rounded up to a multiple of $\delta_q$
    Compute $\delta_q$ integral flow $f^q$ with $cost(f^q) \leq cost(f^{q-1})$
    $A \leftarrow A \setminus \{(i,j) : f_{ij}^q = 0\}$
    For all $a :\ d_a = \delta_q$
        Find any $s - t_a$ path $P_a$
        $f_{ij}^q \leftarrow f_{ij}^q - d_a$ for each $(i,j) \in P_a$
    $A \leftarrow A \setminus\ \{(i,j) : f_{ij}^q = 0\}$
```

Observe that in each iteration δ_q is the new unit of measure. Also observe that in each iteration k the previous flow f^{k-1} is still feasible.

In next class we will show that the above algorithm finds an unsplittable flow such that the flow on each arc is $\leq u_{ij} + d_{max}$. We will also show how to use rounding to exploit this algorithm in solving the general problem.

ORIE 633 Network Flows April 30, 2004

Lecture 39

Lecturer: David P. Williamson *Scribe: Yankun Wang*

39.1 Unsplittable flow

Let's recall the algorithm given in last class:

Unsplittable Flows, Skutella 2002

```
$q \leftarrow 0$
Compute min-cost fractional flow $f^\circ$
while $d_{min} \cdot 2^q \leq d_{max}$
    $q \leftarrow q+1$;    $\delta_q \leftarrow d_{min} \cdot 2^q$
    Set $u_{ij}^q$ to $f_{ij}^{q-1}$ rounded up to the nearest multiple of $\delta_q$
    Compute $\delta_q$ -integral flow $f^q$ s.t. $cost(f^q) \leq cost(f^{q-1})$
    $A \leftarrow A - \{(i,j) : f_{ij} = 0\}$
    For all $a : d_a = \delta_q$
        Find any path $P_a$ from $s$ to $t_a$
        $f_{ij}^q = f_{ij}^q - \delta_q \quad \forall (i,j) \in P_a$
        $A \leftarrow A - \{(i,j) : f_{ij}^q = 0\}$
    Return $P_1, \cdots, P_k$
```

Last class we talked about a single source case: There is one source node, say s, and k sink nodes, say $t_1, \cdots, t_k$, with respective demands, $d_1, \cdots, d_k$.

We observed that if there is no fractional flow, then there is no way that we can find an unsplittable flow.

Theorem 39.1 (Skutella 2002) Assume fractional flow exists, and that $d_{max} \leq u_{min}$. Then we can find an unsplittable flow of cost no more than the fractional flow using capacity $3u_{ij}, \forall (i,j) \in A$

In order to prove this theorem, let's start from an easy case. Assume $d_a = d_{min}2^q, q \in N, \forall a$, then the above algorithm exactly does this for us.

Theorem 39.2 The algorithm finds an unsplittable flow using capacity $u_{ij} + d_{max}$ of cost no more than the fractional flow f^0.

Proof: We prove this by induction on the algorithm. Let's consider total flow using $arc(i,j)$ at the end of the q^{th} iteration, which is given by the current flow value f_{ij}^q plus the demands d_a of commodities a that have been routed across edge (i,j). The total flow is thus bounded by:

$$f^q_{ij} + \sum_{\substack{a:d_a \le \delta_a \\ (i,j)\in P_a}} d_a = f^q_{ij} + \sum_{\substack{a:d_a = \delta_a \\ (i,j)\in P_a}} d_a + \sum_{\substack{a:d_a < \delta_a \\ (i,j)\in P_a}} d_a \le u^q_{ij} + \sum_{\substack{a:d_a < \delta_a \\ (i,j)\in P_a}} d_a$$

Notice that u^q_{ij} is f^{q-1}_{ij} rounded up to a multiple of $d_{min}2^q$ and f^{q-1}_{ij} is a multiple of $d_{min}2^{q-1}$. As a result,

$$u^q_{ij} \le f^{q-1}_{ij} + d_{min}2^{q-1} = f^{q-1}_{ij} + d_{min}2^q - d_{min}2^{q-1}$$

So flow on (i,j) at the end of q^{th} iteration is

$$f^q_{ij} + \sum_{\substack{a:d_a \le \delta_a \\ (i,j)\in P_a}} d_a \le f^{q-1}_{ij} + d_{min}2^q - d_{min}2^{q-1} + \sum_{\substack{a:d_a < \delta_a \\ (i,j)\in P_a}} d_a$$

Applying this inequality iteratively, we get that the flow on (i,j) at the end of the algorithm is bounded by $f^0_{ij} + d_{max} - d_{min}$, thus also bounded by $u_{ij} + d_{max} - d_{min}$.

How about the cost? It's easy to see that the total cost is bounded by the cost of the initial flow f^0_{ij} by the construction of the algorithm. □

Corollary 39.3 For any (i,j), the sum of all demands but one using $arc(i,j)$ is bounded by $f^0_{ij} - d_{min}$.

Proof: Let's look at the above proof of the theorem 2. If the biggest demand on (i,j) is routed in q^{th} iteration, then the flow on $(i,j) \le f^0_{ij} + d_{min}2^q - d_{min}$.

□

So we have shown that in this special case, we can get an unsplittable flow from a fractional flow. But how about the general case?

One idea: Round demands up to have the form $d_{min}2^q$, thus no demand increases by more than a factor of 2. We can prove that in this case the capacity needed on each edge is bounded by $3u_{ij}$. But the problem with this is that costs might increase by a factor of 2.

Next idea: Round demands down to have the form: $d_{min}2^q$. Suppose $\hat{d_a} = (d_a$ rounded down$)$.

General Algorithm

```
Find fractional flow $f^0$ for demands $d_a$
Get fractional flow $\hat{f}^0$ for demands $\hat{d_a}$ by removing
    $d_a - \hat{d_a}$ flow from the most expensive $s$-$t_a$ paths in $f^0$ (via flow decomposition)
Apply previous algorithm to flow $\hat{f}^0$
Send $d_a$ units of flow on path $P_a$ found by the algorithm
```

Theorem 39.4 The above algorithm returns an unsplittable flow of cost no more than the fractional flow using capacity $\leq 3u_{ij}$.

Proof: By previous theorem we know that

$$\sum_a \hat{d}_a c(P_a) \leq c(\hat{f}^0) \tag{39.1}$$

Since there was flow on P_a in $\hat{f}^0$ after the most expensive s-t_a path removed, we get that

$$\sum_a (d_a - \hat{d}_a) c(P_a) \leq c(f^0) - c(\hat{f}^0) \tag{39.2}$$

Add (39.1) and (39.2) together, we get

$$\sum_a d_a c(P_a) \leq c(f^0) \tag{39.3}$$

Let a_0 be the largest demand commodity using (i,j). Then by the Corollary 39.3, we know that

$$\sum_{a:(i,j)\in P_a} d_a \leq d_{a_0} + 2 \sum_{\substack{a\neq a_0 \\ (i,j)\in P_a}} \hat{d}_a \leq d_{a_0} + 2\hat{f}^0_{ij} \leq d_{a_0} + 2u_{ij} \leq 3u_{ij}$$

□

Research Question: Can we use capacity bounded by $2u_{ij}$ to have un unsplittable flow without incurring greater cost? A special case of the generalized assignment problem considered by Shmoys and Tardos in 1993 gives us a positive answer. Also, for the version without costs, Dinitz, Garg, Goemans ('99) have designed an algorithm which uses capacities no more than $2u_{ij}$.

ORIE 633 Network Flows May 3, 2004

Lecture 40

Lecturer: David P. Williamson *Scribe: Xin Qi*

In this lecture, we will start discussing the problem "Flows over Time", aka "Dynamic Flows". However, we think the former name is a more favorable one. Typically, "dynamic" means "network changing with time", but here, we are considering the case that flow changes over time, and network stays the same.

40.1 Flows over time

40.1.1 Maximum s-t flow problem over time

A typical problem to consider is Maximum s-t Flow Problem over Time (aka Maximum Dynamic Flow Problem).

Maximum s-t Flow Problem over Time

- **Input:**
  - Directed graph $G = (V, A)$
  - Integer capacities $u_{ij} \geq 0$ $(\forall (i,j) \in A)$
  - Source s and sink t
  - Integer transit time τ_{ij} $(\forall (i,j) \in A)$
  - Integer time bound T
- **Goal:** Find maximum amount of flow sent from s arriving at t by time T.

Compared to the ordinary maxflow problem, this problem is sort of more realistic. We are trying to model highways or networks or something that requires some time to traverse an arc.

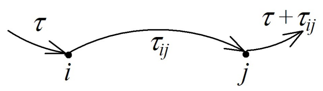
Let us clarify some ideas in this model:

- Transit times τ_{ij}:

  We can imagine that when flow enters arc (i,j) at node i at time τ, it will arrive at j at time $\tau + \tau_{ij}$.

- Capacities c_{ij}:

  There are two possible explanation of the capacity of an arc (i,j):

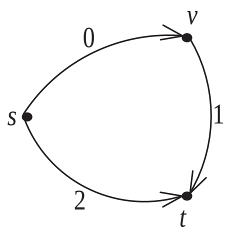


- Total flow using arc (i, j) at any time is bounded by u_{ij}.
- This is what we are going to use: u_{ij} bounds the rate of flow entering (i, j), i.e. no more than u_{ij} units per time unit $[\tau, \tau + 1)$.

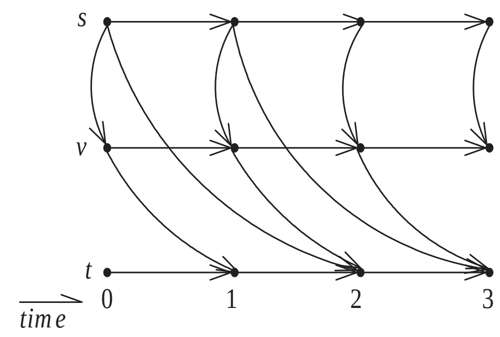
Another problem that can be considered is "Quickest Flow", i.e. given flow value V, find the smallest time T, s.t. we can ship V flow in time T. In fact, with either one solved, we can solve the other problem by binary search.

40.1.2 Time-expanded network

There is one idea to solve the flow problem, however, it is not polynomial time. First, let us take a look at a simple example as shown below. There are three nodes s, v, t, and three arcs. The transit times are marked beside the corresponding arcs.



We will make $T + 1$ copies $v(0), v(1), \ldots, v(T)$ of every vertex v. For any original arc (i, j), we have arcs $(i(\tau), j(\tau + \tau_{ij}))$ in the time-expanded network. We will also allow flow to stay at nodes, by adding "hold over" arcs (marked as straight lines) between consecutive copies of a node with capacities ∞.



We can solve the Maximum s-t Flow Problem over Time by computing maximum $s(0)$-$t(T)$ flow in time-expanded network. Since the constructed network has size proportional to T, it will be only a pseudo-polynomial time algorithm.

We will need some other idea to get a polynomial-time algorithm.

40.1.3 Temporally repeated flow

Suppose we have an s-t path P s.t. $\tau(P) \equiv \sum_{(i,j)\in P} \tau_{ij} \leq T$. If $u = \min_{(i,j)\in P} u_{ij}$, then we can send u units of flow along path P at time $0, 1, \ldots, T - \tau(P)$.

Definition 40.1 Such a flow over time is called a *temporally repeated flow.*

Lemma 40.1 Given standard s-t flow f, decomposition of f into $s-t$ paths $P_1, P_2, \ldots, P_l$, with P_k sending δ_k units of flow s.t. $\tau(P_k) \leq T \ \forall k$, then the value of flow over time by temporally repeating these paths is

$$(T+1)|f| - \sum_{(i,j)\in A} \tau_{ij} f_{ij}$$

Proof: First, we need to show that the temporally repeated flow is a valid flow over time. We check the capacity constraint: at any time τ, flow entering (i,j) is no more than

$$\sum_{k:(i,j)\in P_k} \delta_k = f_{ij} \leq u_{ij}$$

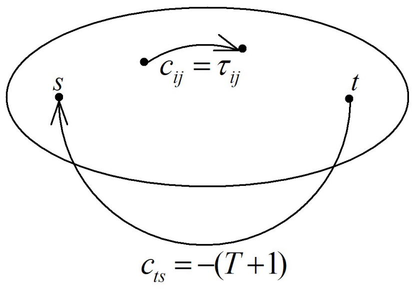
Hence, the capacity constraints are respected.

Second, we are going to compute the total flow. For each path P_k, it sends δ_k units of flow for $T + 1 - \tau(P_k)$ time units. Therefore the total amount of flow sent is

$$\begin{aligned}
\sum_k \delta_k(T+1-\tau(P_k)) &= \sum_k (T+1)\delta_k - \sum_k \delta_k \tau(P_k) \\
&= (T+1)|f| - \sum_k \delta_k \sum_{(i,j)\in P_k} \tau_{ij} \\
&= (T+1)|f| - \sum_{(i,j)\in A} \tau_{ij} \sum_{k:(i,j)\in P_k} \delta_k \\
&= (T+1)|f| - \sum_{(i,j)\in A} \tau_{ij} f_{ij}
\end{aligned}$$

□

Now we want to maximize the temporally repeated flow, which can be solved by a min-cost circulation problem. We will treat τ_{ij} as the cost of arc (i,j), and add an arc from t to s with cost $-(T+1)$, as shown in the above figure. Then we can find the min-cost circulation in that graph by canceling negative cost cycles, and this will exactly satisfy all the requirements:



- Any negative cost cycle uses arc (t, s).
- The $s - t$ part of any negative cost cycle has cost $\leq T$.

So we have a polynomial time algorithm to compute the optimal temporally repeated flow; however, it is not clear until now why this will be useful. Next time, we will prove the following theorem:

Theorem 40.2 (Ford, Fulkerson '62) The value of the maximum s-t flow over time equals the value of the maximum temporally repeated flow.

ORIE 633 Network Flows May 5, 2004

Lecture 41

Lecturer: David P. Williamson *Scribe: David P. Williamson*

41.1 Flows over time

41.1.1 Maximum s-t flow problem over time (cont.)

In this lecture, we will complete the proof of the theorem we stated last time.

Theorem 41.1 (Ford, Fulkerson '62) The value of the maximum s-t flow over time equals the value of the maximum temporally repeated flow.

Proof: Although we argued last time that we could not get a polynomial-time algorithm by using the time-expanded network, we can use it in the proof to show our result. We will argue that the minimum $s(0)$-$t(T)$ cut in the time-expanded network has the same value as the maximum temporally repeated flow. This will prove the theorem.

We found the maximum temporally repeated flow by finding a minimum-cost circulation in the network in which the cost of each arc (i,j) was set to the transit time τ_{ij}, and we added an arc from t to s of cost $-(T+1)$. This found an s-t flow f minimizing $\sum_{(i,j)\in A} f_{ij}\tau_{ij} - (T+1)|f|$, which maximized the value of the associated temporally repeated flow. Recall from our discussion of minimum-cost circulations that f is a min-cost circulation iff there exist potentials p such that $c^p_{ij} \equiv \tau_{ij} + p_i - p_j \geq 0$ for all $(i,j) \in A_f$. Also if $c^p_{ij} > 0$ then $f_{ij} = \ell_{ij} = 0$, and if $c^p_{ij} < 0$ then $f_{ij} = u_{ij}$. So we know that the cost of the circulation is

$$\sum_{(i,j)\in A} c_{ij} f_{ij} = \sum_{(i,j)\in A} c^p_{ij} f_{ij} = \sum_{c^p_{ij}<0} c^p_{ij} u_{ij},$$

which is equal to the value $\sum_{(i,j)\in A} f_{ij}\tau_{ij} - (T+1)|f|$.

We assume $|f| \neq 0$. We will consider a cut S in the time-expanded network with $S = \{i(\theta) : p_i - p_s \leq \theta\}$. Note that $s(0) \in S$ since $p_s - p_s = 0$. Furthermore, if $|f| \neq 0$, then $f_{ts} > 0$, which implies that $c^p_{ts} = 0$, which implies that $p_t - p_s = -c_{ts} = T+1$. Therefore, $t(T) \notin S$, since $p_t - p_s > T$.

Note that any holdover arc $(i(\theta), i(\theta+1))$ is not in the cut, since if $i(\theta) \in S$ then $i(\theta+1) \in S$.

Now we compute the capacity of the cut. An arc $(i(\theta), j(\theta + \tau_{ij}))$ is in the cut for all θ such that $p_i - p_s \leq theta$ and $p_j - p_s > \theta + \tau_{ij}$. Then the capacity of the cut is

$$\begin{aligned}
\sum_{(i,j)\in A} u_{ij} \cdot \max(0, p_j - p_s - (p_i - p_s) - \tau_{ij}) &= \sum_{(i,j)\in A} u_{ij} \cdot \max(0, p_j - p_i - \tau_{ij}) \\
&= \sum_{(i,j)\in A} u_{ij} \cdot \max(0, -c^p_{ij}) \\
&= - \sum_{(i,j)\in A : c^p_{ij} < 0} u_{ij} c^p_{ij} \\
&= (T+1)|f| - \sum_{(i,j)\in A} \tau_{ij} f_{ij},
\end{aligned}$$

where the last inequality follows by previous discussion, and is equal to the value of the temporally repeated flow. □

In the area of flows over time, the quickest transshipment problem is known to be in polynomial time, but needs fairly complicated subroutines. The quickest minimum-cost flow problem is NP-hard, as is the quickest multicommodity flow problem. Work on approximation algorithms for these problems has been done. A natural variant of this problem is to suppose that the transit time τ_{ij} is a function of the flow on the arc f_{ij}, and there has been some preliminary work on this.