

# 1

## NETWORK FLOWS

This book deals with discrete and continuous optimization problems that arise in the context of networks (also called *graphs*). There are several important problems in this area. Two well known examples are: **flow** problems dealing with sending one or more commodities from one specified set of points (called sources) to another set of points (called sinks); *path* or route problems dealing with finding the shortest way of traversing from a specified point to another possibly through a specified set of points. We will come back to these later on. There are several text books in this area as well as journal articles of importance. Much of this is listed in the bibliography.



## 2

# Single Commodity Maximum Flow Problem

In this chapter, we consider single commodity flow problems. The first of these is called the *maximum flow problem* and its description is given below:

### PROBLEM I:

Given a directed network  $G = [N; A]$ , a special node  $s$ , called the *source* or *origin*, a node  $t$ , called the *sink* or *destination* and positive numbers  $u_{i,j}$  representing the capacity of arc  $(i, j) \in A$ , we want to maximize the total shipment from  $s$  to  $t$ . This problem is called the *maximum flow problem* in the literature and is one starting point in this area.

### HISTORY

The special case when all  $u_{i,j}$  are 0/1 was treated by **Menger**. The general case gives rise to the now famous *max-flow-min-cut theorem* which seems to have been proved independently by three sets of authors: **Elias, Shannon and Feinstein [EFS]**; **Ford and Fulkerson [?]**; and **Robacker [R]**. **Ford and Fulkerson** also gave an algorithm called the *labeling algorithm* to solve the problem. This algorithm can be shown to differ from the simplex algorithm of linear programming in some of its versions. It also preserves the intuitive appeal that is common to this area. While practical implementations of this algorithm worked astoundingly well, **Ford and Fulkerson** showed that it is possible to create pathological examples for which certain implementations of this algorithm not only did not terminate in a finite number of steps but did not even converge to the right answer. Independently, **Dinic [D]** and **Edmonds & Karp [EK]**

showed that there were implementations that not only terminated in a finite number of steps but were strongly polynomial. These were improved upon by **Karzanov [K]**, as well as **Malhotra, Kumar, and Maheshwari [MKM]** and recently computer scientists have reduced the order of complexity even further by clever use of data structures. At the same time there have been attempts to fully understand the structure of the problem – for example, the work of **Picard and Queyranne [PQ]** on the structure of minimal cuts. There have also been attempts to generalize the problem to multi-terminal flows, multi-commodity flows, and to flows in matroids other than graphic matroids etc.

## 2.1 FORMULATION:

Let  $f_{i,j}$  be the flow on arc  $(i,j) \in A$  and  $F$  be the total flow across the network. Then:

$$\sum_j (f_{i,j} - f_{j,i}) = \begin{cases} F & i = s \\ 0 & i \neq s, t \\ -F & i = t \end{cases} \quad (2.1)$$

$$0 \leq f_{i,j} \leq u_{i,j} \quad \forall (i,j) \in A \quad (2.2)$$

$$\max F \quad (2.3)$$

This is, of course, a linear program and can, in principle, be solved by the simplex method. However, the problem could be highly degenerate and unless special precautions are taken (as described in the section on *threaded index method*) it could be very inefficient to use the simplex method. Before describing methods to solve the problem we remark that it is easy to find a feasible solution for this problem as it has been stated. For example  $[F = 0; f_{i,j} = 0 \quad \forall (i,j) \in A]$  will suffice.

## 2.2 Max-flow Min-cut Theorem:

First some definitions and a few preliminary results. The notion of a *cut* or a *cut set* separating  $s$  and  $t$  can be defined in several slightly different ways. The first is to think of it as a partition of the node set  $N$  into two disjoint sets  $S$  and  $\bar{S}$  with  $s \in S$  and  $t \in \bar{S}$ . The next is to think of a cut as the set of arcs that connect these two sets mentioned above. A slightly weaker version is a minimal set of arcs whose removal disconnects the nodes  $s$  and  $t$ . Unless otherwise stated, we will use the first form.

**Lemma 1** :Let  $[f, F]$  be any feasible flow and let  $(S, \bar{S})$  be any cut separating  $s$  and  $t$ . Then  $F \leq u(S, \bar{S})$  where  $u(S, \bar{S}) = \sum_{i \in S} \sum_{j \in \bar{S}} u_{i,j}$ .  $u(S, \bar{S})$  is called the value or capacity of the cut  $(S, \bar{S})$ . If equality holds then  $[f, F]$  is optimal to the maximum flow problem and  $(S, \bar{S})$  is a cut whose value is minimum among all cuts separating  $s$  and  $t$ .

**Proof:** The first part of the lemma clearly implies the second. To prove the first consider the flow conservation equations. Adding the equations in (2.1) corresponding to the set  $S$  we get:

$$\begin{aligned} F &= \sum_{i \in S} \sum_{j \in N} f_{i,j} - \sum_{i \in S} \sum_{j \in N} f_{j,i} \\ &= f(S, N) - f(N, S) \\ &= f(S, \bar{S}) - f(\bar{S}, S) \leq u(S, \bar{S}) \end{aligned} \quad (2.4)$$

The last of these inequalities follows from the fact that  $0 \leq f_{i,j} \leq u_{i,j}$  for all arcs  $(i, j)$  in the network.  $\square$

The following algorithm, known in the literature as the (*flow*) *labeling algorithm*, is one way to show that there exist solutions that achieve equality in the relation above. It is usually attributed to Ford and Fulkerson.

## 2.3 Labeling Algorithm:

**Input:** A feasible solution  $[f, F]$ .

**Step 0:** Label  $s : (-, \infty)$  and let  $S \in S$ , the set of labeled nodes.

**Step 1:** If  $t \in S$  stop; a flow augmenting path has been found. (This is a path along which additional flow can be sent thereby increasing the total flow  $F$ ). If not look for a pair  $(i, j)$  with  $i \in S$  and  $j \in \bar{S}$  and either (i)  $[(i, j) \in A; f_{i,j} < u_{i,j}]$  or (ii)  $[(j, i) \in A; f_{j,i} > 0]$ . If no such  $(i, j)$  exists, then stop;  $[f, F]$  is the optimal solution to the maximal flow problem and  $(S, \bar{S})$  is a minimal cut separating  $s$  and  $t$ . If  $(i, j)$  of type (i) exists label  $j : (i^+, \epsilon_j)$  where  $\epsilon_j = \min[\epsilon_i, u_{i,j} - f_{i,j}]$  and include  $j$  in  $S$ . If  $(i, j)$  is of type (ii) then label  $j : (i^-, \epsilon_j)$  where  $\epsilon_j = \min[\epsilon_i, f_{j,i}]$  and include  $j$  in  $S$ . Return to step 1. Type (i) arcs are called *forward* arcs and those of type (ii) are called *reverse* arcs. If we succeed in labeling  $t$ , we get an augmenting path as well as the nature of these arcs from the labels themselves. This is done as follows: If the label of  $t$  is  $(j^+, \epsilon_t)$  then the previous node to  $t$  in the flow augmenting path is  $j$  and the last arc is a forward arc; if it is  $(j^-, \epsilon_t)$  then the previous node is still  $j$  but the arc is a reverse arc. Now we look at the label of the node  $j$  and the process is repeated until we reach  $s$  and this identifies the entire path. We now augment the flow by  $\epsilon_t$  along the path — by which we mean that flows on forward arcs along the path are increased by  $\epsilon_t$  and those on reverse arcs are decreased by the

same amount. This gives us a new feasible flow and the process is repeated. If the starting solution is optimal then at some point before labeling  $t$  we will find no arc of type (i) or (ii). At this point the set  $S$  of labeled nodes gives a minimum cut separating  $s$  and  $t$ . Further, all arcs across the cut  $(S, \bar{S})$  with the initial end in  $S$  will be saturated ( $f = u$ ) and all arcs with the terminal end in  $S$  will be flowless ( $f = 0$ ). In other words, all forward arcs across the cut will be *saturated* and all reverse arcs will be *flowless*. Of course, if this happens with any feasible flow and any cut separating  $s$  and  $t$  then this flow is optimal and the corresponding cut is minimal.

**Lemma 2** *Let  $(X, \bar{X})$  and  $(Y, \bar{Y})$  be minimal cuts. Then  $(X \cup Y, \overline{X \cup Y})$  and  $(X \cap Y, \overline{X \cap Y})$  are also minimal cuts. This lemma shows that the set of minimal cuts form a lattice.*

**Proof:** If  $(i, j) \in A$  and  $i \in (X \cup Y)$  and  $j \notin (X \cup Y)$  then either  $i \in X$  and  $j \notin X$  or  $i \in Y$  and  $j \notin Y$ ; in either case  $f_{i,j} = u_{i,j}$  since both  $(X, \bar{X})$  and  $(Y, \bar{Y})$  are minimal cuts separating  $s$  and  $t$ . Thus, all forward arcs across the cut  $(X \cup Y, \overline{X \cup Y})$  are saturated. In a similar manner we can show that all reverse arcs are flowless. Thus, this is also a minimal separating  $s$  and  $t$ . Similar proof applies to other part of the lemma.  $\square$

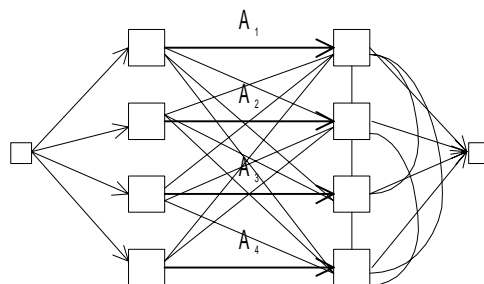
Note that the labeling algorithm gives the optimal solution starting from any integer feasible flows on all arcs (one such is  $f \equiv 0$ ) in a finite number of steps if the capacities are all integral (or rational) since the augmentations are integral at each step and the final solution is also integral. This is true regardless of the augmenting path chosen at each step. However, the number of such augmentations may be very large if the augmentations are chosen cleverly to make the algorithm look poor. Also, if the capacities are irrational the algorithm may not terminate in a finite number of steps nor converge to the optimal answer. This is shown by the following example found in [FF2].

## 2.4 Finite Termination of Maximum Flow Algorithms

### 2.4.1 Ford-Fulkerson Example:

$G = [N, A]$  where  $N = \{s; t; x_i, y_i; i = 1, 2, 3, 4\}$  and  $A = \{(s, x_i); i = 1, 2, 3, 4\} \cup \{(y_i, t) i = 1, 2, 3, 4\} \cup \{(x_i, y_j) i \neq j\} \cup$

$$\{(y_i, y_j) \mid i \neq j\} \cup \{(y_i, x_j) \mid i \neq j\} \cup \{(x_i, y_i) \mid i = 1, 2, 3, 4\}$$



The last four arcs are special arcs and are denoted by  $A_i$ ,  $i = 1, 2, 3, 4$ . The capacities of the special arcs are respectively  $a_0, a_1, a_2,$  and  $a_2$  where  $a_n$  satisfies the recursion  $a_{n+1} + a_{n+2} = a_n$  and is given by  $a_n = r^n$  where  $r = \frac{-1+\sqrt{5}}{2} < 1$ . The capacities of the remaining arcs equals  $S$  where  $S = \sum_n a_n < \infty$ . Clearly the optimal solution is  $F = 4S$ . However, the following augmenting scheme which is infinite has a limit of only  $S$ . In order to state the scheme we need to introduce the notion of *residual capacity* of the arcs.

Residual capacity

$$r_{i,j} = \max[u_{i,j} - f_{i,j}, 0] + \max[f_{j,i}, 0]$$

The residual capacity of arc  $(j, i)$  (even if this is not an arc in the original network) is also defined in the same manner. Now let us consider the following flow augmentations.

**Step 0:** Starting from zero flows on all arcs, increase the total flow by  $a_0$  along the path  $s - x_1 - y_1 - t$ . The residual capacities of the special arcs are  $0, a_1, a_2, a_2$  respectively. We will not have to bother about the residual capacities of any of the other arcs since these will always be large in comparison. We will start every one of the remaining cycles with the residual capacities of the special arcs in the form  $0, a_n, a_{n+1}, a_{n+1}$  although the arcs themselves may occur in different orders.

**Step 1:** Let us suppose that some rearrangement of the special arcs, say  $A'_1, A'_2, A'_3, A'_4$  have residual capacities  $0, a_n, a_{n+1}, a_{n+1}$  respectively. Consider the path  $s - x'_2 - y'_2 - x'_3 - y'_3 - t$ . The arc with the lowest residual capacity is  $A'_3$  whose residual capacity is  $a_{n+1}$ . (Recall that the series  $\{a_n\}$  is a decreasing series). So we send an amount  $a_{n+1}$  along this path to make the residual capacities on the special arcs  $[0, a_n - a_{n+1}, 0, a_{n+1}] = [0, a_{n+2}, 0, a_{n+1}]$ .

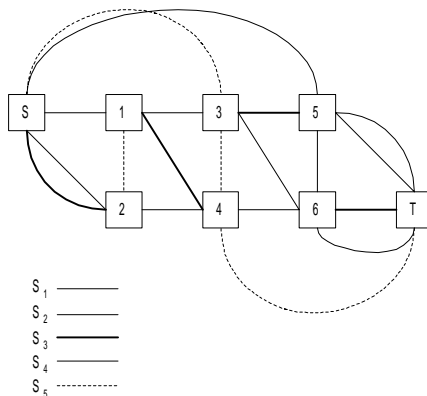
**Step 2:** Now consider the path  $s - x'_2 - y'_2 - y'_1 - x'_1 - y'_3 - x'_3 - y'_4 - t$ . The forward arc with the lowest residual capacity in this path is  $A'_2$  whose residual capacity is  $a_{n+2}$ . The reverse arcs are  $A'_1$  and  $A'_3$ . Both have residual capacity equal to 0 and hence are capacitated. Hence in the reverse direction they can each take more flow than  $a_{n+2}$ . Hence we can send only  $a_{n+2}$

along this path. The new residual capacities will be  $[a_{n+2}, 0, a_{n+2}, a_{n+1}]$ . We now have exactly the same pattern of residual capacities as we started with in step 1. The cycle repeats indefinitely. The flow increase in steps 1 and 2 equals  $a_{n+1} + a_{n+2} = a_n$ . Thus the total flow (taking limits) equals  $\sum_0^\infty a_n = S$ . This example further illustrates why this algorithm is not an implementation of the simplex method. It is an interior path method.

In the above example, a particular implementation of the flow labeling algorithm does not have a finite termination nor a convergence to the optimal value. One way of ensuring finite termination is due to **T.C. Hu**. In his algorithm, during each *phase* we allow an arc to become *critical* (saturated or flowless) at most once *by disallowing it after it becomes critical* as long as we are in the same phase. At the end of a phase, we put all arcs back into consideration for flow labeling and repeat the process. It is clear that each phase can not have more than  $|A|$  flow augmentations. *Also, if the network is undirected, we only have saturations for criticality.* At the end of a phase, the cut defined by the set of labeled and unlabeled nodes has each arc either saturated or flowless (in the undirected case they must be saturated). Since there are only a finite number of cuts separating  $s$  and  $t$  and for each such cut there are only finitely many ways of having arcs saturated or flowless, the algorithm is finite. This combines the result in [P] as well.

#### 2.4.2 Queyranne's Example:

**M. Queyranne [Q]** discovered an example for which the algorithm "Capacity", mentioned by **Edmonds and Karp [EK]**, is not finite. This algorithm augments flow at each step along the path which allows the maximum increase in flow at that step. *It is very easy to modify an algorithm (for example Dijkstra's algorithm) for shortest route to determine such a path efficiently; details of such a discussion will be found in the chapter on shortest routes.* Contrast Queyranne's result with that in [EK].



Nodes:  $N = \{s, t, 1, 2, \dots, 6\}$ ; origin  $s$  and destination  $t$ .

Arcs and their capacities:

$(s, 1) : S_1 = \alpha$ ;  $(s, 2)^1 : S_2 = \beta$ ;  $(s, 2)^2 : S_3 = \gamma$ ;  $(s, 3) : r$ ;  $(s, 5) : 1$

$(1, 2) : r$ ;  $(1, 3) : \beta$ ;  $(1, 4) : \gamma$

$(2, 4) : \alpha$

$(3, 4) : r$ ;  $(3, 5) : \gamma$ ;  $(3, 6) : \alpha$

$(4, 6) : \beta$ ;  $(4, t) : r$

$(5, 6) : 1$ ;  $(5, t)^1 : \alpha$ ;  $(5, t)^2 : \beta$

$(6, t)^1 : 1$ ;  $(6, t)^2 : \gamma$

where  $r = (\sqrt{5} - 1)/2 < 1$ , and satisfies the relation:  $r^2 + r - 1 = 0$  and hence  $r^{n+2} + r^{n+1} = r^n$ . Also  $\alpha$ ,  $\beta$ , and  $\gamma$  are given by:

$$\beta = \sum_{k=0}^{\infty} r^{3k+2} = \frac{r^2}{1-r^3} = 1/2 \text{ since } 1-r^3 = 1-r+r^2 = 2r^2.$$

$$\gamma = \sum_{k=0}^{\infty} r^{3k+3} = \frac{r^3}{1-r^3} = r/2 \text{ by the same argument.}$$

$$\alpha = \sum_{k=0}^{\infty} r^{3k+1} = \beta + \gamma = \frac{r+1}{2}.$$

**Lemma 3**  $\alpha \geq r \geq \beta \geq r^2 \geq \gamma \geq r^3 \geq \alpha - r \geq r^4 \geq \beta - r^2$ .

**Proof:**  $\alpha \geq r$ ,  $\beta \geq r^2$ , and  $\gamma \geq r^3$  are clear because these are the first terms in these respective sums. Since  $7/3 > \sqrt{5} > 2$ , it follows that  $r \geq \beta$  and  $r^2 \geq \gamma$ .  $r \geq \beta = 1/2$  implies  $r^3 = r(1-r) \geq (1-r)/2 = \alpha - r$ .  $r^4 = r^2 - r^3 = (1-r)^2 \leq (1-r)/2 = \alpha - r$  since  $(1-r) < 1/2$ .  $\beta - r^2 = r(\alpha - r) \leq r^4$  since  $(\alpha - r) \leq r^3$ . Hence the lemma.  $\square$

*Flow Augmentations:*

Since no capacity is larger than one the first augmentation must be on path  $s - 5 - 6 - t$  and the flow sent along this path is one unit. Now there are three possible paths with capacity equal to  $r$ . Note that  $\alpha$ ,  $\beta$ , and  $\gamma$  are all less than  $r$ . These three paths are:

$s - 3 - 4 - t$ ;  $s - 1 - 2 - 4 - t$ ;  $s - 3 - 6 \leftarrow -5 - t^1$ .

$\leftarrow -$  refers to reverse arc. For now, we assume that we select the first since it is easy to see that this selection does not affect the result. At this time arcs  $(s, 3)$ ,  $(s, 5)$ ,  $(4, t)$  and  $(6, t)$  are saturated and their flow will not be modified from now on. The remaining arcs and their flows are :

Arc	Flow
1, 2	0
3, 4	$r$
5, 6	1
$\alpha$ - type	0
$\beta$ - type	0
$\gamma$ - type	0

At this point on the following cyclic pattern sets in:

<i>Arc</i>	<i>Flow 1</i>	<i>Flow 2</i>	<i>Flow 3</i>
1, 2	0	$r^{3k-2}$	$r^{3k}$
3, 4	$r^{3k-2}$	0	$r^{3k-1}$
5, 6	$r^{3k-3}$	$r^{3k-1}$	0
$\alpha$ -type	$\alpha(1 - r^{3k-3})$	$\alpha(1 - r^{3k})$	$\alpha(1 - r^{3k})$
$\beta$ -type	$\beta(1 - r^{3k-3})$	$\beta(1 - r^{3k-3})$	$\beta(1 - r^{3k})$
$\gamma$ -type	$\gamma(1 - r^{3k-3})$	$\gamma(1 - r^{3k-3})$	$\gamma(1 - r^{3k-3})$

The corresponding unique paths and the flow increases on them are:  
 $s - 1 - 2 - 4 - 3 - 6 - 5 - t$  : all  $\alpha$ -type arcs  $-- r^{3k-2}$ ;  
 $s - 2 - 1 - 3 - 4 - 6 - 5 - t$  : all  $\beta$ -type arcs  $-- r^{3k-1}$ ;  
 $s - 2 - 1 - 4 - 3 - 5 - 6 - t$  : all  $\gamma$ -type arcs  $-- r^{3k}$ .

*Convergence:*

**Theorem 4** *The above algorithm converges to the optimum value.*

**Proof:** We have seen that any feasible flow can be obtained in no more than  $|A|$  augmentations by the “peeling” process. Also, these augmentations can be made in any order since we use only forward arcs in the whole process. Hence there exists such a sequence of augmentations starting from any feasible flow  $F$  to reach the optimal flow  $F^*$ . Hence, at the step when we have a flow  $F$ ,  $\exists$  an augmentation whose value is at least  $(F^* - F)/|A|$ .  $\square$

Now suppose we have an infinite sequence  $F^i, i = 1, 2, \dots$ , of flows. Since this is monotone increasing sequence bounded above by the value of any cut, it converges to a limit, say  $F^0$ . If  $F^* - F^0 > 0$ , then  $\exists h \ni F^{h+1} - F^h < (F^* - F^0)/|A|$ . But this impossible by using the above argument using the peeling process with  $F = F^h$  and  $F^*$ . Hence the theorem.

**Consistent Labeling:**

In this type of labeling (with broad interpretation) first the origin is labeled as before. Next we select a pair of nodes  $i$  and  $j$  where  $i$  is labeled and  $j$  is unlabeled as before; however, the selection of this pair follows the following rules: it is determined by the set of presently labeled nodes and by the set of *unsuccessful pairs* that have been attempted so far at this step. In the narrow interpretation, the next node to be scanned depends on the set of currently labeled nodes that are unscanned. All unlabeled neighbors of this selected node that can be labeled from this node are so labeled. It turns out that the broad interpretation includes the narrow one. Tucker showed that consistent labeling in the narrow sense leads to a finite algorithm and Megiddo and Galil showed that such an algorithm need not be polynomially bounded. We leave these as exercises.

## 2.5 Strongly Polynomial Algorithms:

Although the fact that the algorithm is infinite with irrational data is of little practical concern, the fact that even with integral data the number of flow augmentations can be very large is of practical concern. It turns out that the algorithm which at each step augments flow along the path that will allow the maximum possible increase is practically efficient unless the capacities are very large. There are, however, algorithms that serve both purposes of being theoretically and practically efficient. We turn our attention to these now. Indeed one of these is a natural way in which **Ford and Fulkerson** flow labeling is programmed.

**J.Edmonds and R.Karp [EK]** showed that the algorithm which at each step does the flow augmentation along a path with the fewest number of arcs is not only finite but strongly polynomial. This of course corresponds to what is known as “breadth-first-search”. We now give a proof of this fact and estimate the complexity of such an algorithm.

Let

$\sigma_i^k$  = the minimum number of arcs in a flow augmenting path from  $s$  to  $i$  after  $k$  flow augmentations in this algorithm and

$\tau_i^k$  = the minimum number of arcs in a flow augmenting path from  $i$  to  $t$  after  $k$  such augmentations.

**Lemma 5**  $\sigma_i^{k+1} \geq \sigma_i^k; \tau_i^{k+1} \geq \tau_i^k$  for all  $i$  and  $k$ .

**Proof:** We will only prove the first of these. Suppose it is false. Let

$$\sigma_i^{k+1} = \min[\sigma_j^{k+1} : \sigma_j^{k+1} < \sigma_j^k]$$

Clearly  $\sigma_i^{k+1} \geq 1$  (since  $i \neq s$  and only  $\sigma_s^{k+1} = 0$ ). Let  $j$  be the last point on this path before  $i$ .  $\sigma_i^{k+1} = \sigma_j^{k+1} + 1$  and by the definition of  $i$ ,  $\sigma_j^{k+1} \geq \sigma_j^k$ . If the last arc is a forward arc  $(j, i)$  with  $f_{j,i} < u_{j,i}$  after  $k+1$  augmentations, then we claim that it must have been saturated after  $k$  augmentation. For, otherwise,  $\sigma_i^k \leq \sigma_j^k + 1 \leq \sigma_i^{k+1}$  which is a contradiction to the definition of  $i$ . But if arc  $(j, i)$  was saturated after  $k$  augmentations and unsaturated after  $k+1$  augmentations then it was used in the reverse direction during the  $k+1^{st}$  augmentation. Hence  $\sigma_j^k = \sigma_i^k + 1$  and combining this with the fact that  $\sigma_j^k + 1 \leq \sigma_i^{k+1}$ , we get  $\sigma_i^k + 2 \leq \sigma_i^{k+1}$  and this is a contradiction to the definition of  $i$ . If the arc is a reverse arc  $(i, j)$  the proof is along similar lines with slight modifications.  $\square$

**Theorem 6** *The number of augmentations in this algorithm is no more than  $mn/2 \leq (n^3 - n^2)/2$ .*

**Proof:** Each time an augmentation is made one arc in the augmenting path is *critical* in the sense that it is the limiting arc. The flow on this arc is either increased to capacity or decreased to 0. If  $(i, j)$  is a critical arc on

the  $(k+1)^{st}$  augmentation then the number of arcs in this path =  $\sigma_i^k + \tau_i^k = \sigma_j^k + \tau_j^k$ . The next time the arc is used in a flow augmenting path, say the  $(l+1)^{st}$  augmentation, it will be used in the opposite direction. Thus, if  $(i, j)$  was a forward arc in the  $(k+1)^{st}$  augmentation then,  $\sigma_j^k = \sigma_i^k + 1$  and  $\sigma_i^l = \sigma_j^l + 1$ . Because of the lemma  $\sigma_j^l \geq \sigma_j^k$ , and  $\tau_j^l \geq \tau_j^k$ , and hence  $\sigma_i^l + \tau_i^l \geq \sigma_i^k + \tau_i^k + 2$ . Thus, it follows that each succeeding path in which an arc is critical is longer by at least two arcs. Thus, an arc can be critical at most  $(n-1)/2$  times and hence the theorem.  $\square$

**N.Zadeh [Z]** has shown that this bound is achieved in some class of networks. Now we describe the algorithms of **Dinic [D]** and **Malhotra, Kumar and Maheshwari [MKM]**.

### Layered Network:

This is best described in a paper by **S. Even [E]**. A *layered network* is an acyclic network constructed for a given feasible flow  $[f, F]$ . It consists of sets  $V_0, V_1, \dots, V_p$  where  $V_0 = \{s\}$  and  $V_{i+1}$  is defined as the set of nodes that can be reached from the nodes of  $V_i$  by one step of the labeling process and have not so far been included in the previous sets  $V_j$  for  $j \leq i$ . If at some point  $p$ , we can not label any other nodes and  $t$  is not yet labeled then we stop with  $S = \cup_{i=1}^p V_i$ . In this case, we have the maximal flow and the minimal cut. If  $t \in V_p$  then  $V_p$  is redefined to be  $V_p = \{t\}$ , and the layered network definition is complete. The arcs in this network go from nodes in  $V_i$  to nodes in  $V_{i+1}$  and correspond to those arcs (or their reverse in the original network) along which labeling was possible. The capacities are the residual capacities ( $= u_{i,j} - f_{i,j}$ ) if the arc is used in its original direction and  $f_{i,j}$  if it is used in the direction  $(j, i)$ .

**Saturation flows:** In a network  $[f, F]$  is a *saturation flow* if there is no augmenting path containing only forward arcs. Thus, if  $[f, F]$  is a saturation flow then any chain will have an arc that is capacitated.

### Dinic's Algorithm:

Start with  $[f = 0, F = 0]$ . At any stage of the algorithm we will have a feasible flow  $[f, F]$ . Construct a layered network with respect to this flow; saturate this network. Superimpose this additional flow on to the existing flow to get a new solution  $[f', F']$  and repeat this cycle until the flow is optimal. To show that such an algorithm works, we prove that the number of layers in the networks so constructed increases at each step and there can not be more than  $n$  layers. Thus, the number of major cycles of this algorithm can not exceed  $n$  and each cycle can not exceed the number of arcs since at each augmentation in a specific network one arc is saturated. [MKM] do the saturation process more cleverly.

**Lemma 7** *Let  $l_k$  denote the length of the layered network in the  $k^{th}$  step. Then  $l_{k+1} > l_k$ .*

**Proof:** There is a path between  $s$  and  $t$  in the  $(k+1)^{st}$  layered network of length  $l_{k+1}$  (indeed all paths in this network between  $s$  and  $t$  are of this length). This is depicted in the following diagram:

**Case 1:** Suppose all nodes in this path are also part of the  $k^{th}$  layered network. Let  $V_j$  denote the  $j^{th}$  layer in the  $k^{th}$  layered network. If  $i_a \in V_b$  then we claim that  $a \geq b$ . (This will show that  $l_{k+1} \geq l_k$ ). This is proved by induction on  $a$ . Since  $s$  is always in the  $0^{th}$  layer the claim is clearly true for  $a = 0$ . Suppose that the result is true up to  $a$  and let  $i_{a+1} \in V_c$ . If  $c \leq b+1$  then we are done. If  $c > b+1$ , then edge  $e_{a+1}$  does not belong to the  $k^{th}$  layered network and hence not used in the augmentations there in. Thus, if it is useful in the  $k+1^{st}$  layered network then it was useful in the  $k^{th}$  phase. This is a contradiction to  $i_{a+1} \in V_c$ . Hence the claim.

Now to show that  $l_{k+1} > l_k$ . This is done by showing that if  $l_{k+1} = l_k$  then  $i_{l_{k+1}-1} \in V_c$  implies  $c \leq l_k - 1$ ; however,  $c < l_k - 1$  would mean that edge  $e_{l_{k+1}}$  was not part of the  $k^{th}$  layered network and if it is useful in the  $k+1^{st}$  layered network then it must have been useful in  $k^{th}$  layered network as well and the definition of  $l_k$  is wrong. Thus  $c = l_k - 1$  as well. Hence this previous edge  $e_{l_{k+1}}$  is also in the  $k^{th}$  layered network. By repeating this argument we can show that the entire path is in the  $k^{th}$  layered network. This is a contradiction to the fact that we saturated the  $k^{th}$  layered network before proceeding to  $k+1^{st}$ . Thus  $l_{k+1} > l_k$ .

**Case 2:** Let  $i_{a+1}$  be the first node in the above path not in the  $k^{th}$  layered network. Then by using the same argument as above, we get if  $i_a \in V_q$  then  $a \geq q$ . Since  $i_{a+1}$  does not belong to  $k^{th}$  network the edge  $e_{a+1}$  does not either. This implies that flows on this edge did not change in  $k^{th}$  phase and hence this was an eligible edge at that time. Therefore, the only reason why  $i_{a+1}$  is not in the  $k^{th}$  network is because  $l_k = q+1$ . Clearly  $l_{k+1} > a+1$  (since  $t$  is in the  $k^{th}$  network) and hence  $l_{k+1} > l_k$  in this case as well.  $\square$

**Corollary 8** *The number of major cycles  $\leq n - 1$ .*

Dinic used a straight forward algorithm to saturate the layered network problem and hence got a bound of  $O(nm)$  per cycle and hence a  $O(n^2m)$  algorithm overall. **Karzanov** [K] improved the bound to  $O(n^2)$  per cycle and hence a  $O(n^3)$  algorithm. [MKM] also obtained the same results as Karzanov; their algorithm is much easier to explain and this is what we do now.

### **Malhotra, Kumar, and Maheshwari Algorithm:**

**Input:** An acyclic graph with feasible flow  $[f, F]$ .

**Output:** A saturation flow  $[f', F']$ .

Since the layered network is acyclic we can use this as a subroutine to solve each cycle's problem in Dinic's algorithm. We actually assume that the nodes of the acyclic network are numbered so that all arcs start from a lower numbered node. But this can easily be done in linear time.

Define a node flow potential  $\rho_f(i)$  for each node as follows:

$$\rho_f(s) = \sum_{j \in N} (u_{s,j} - f_{s,j}) \quad (2.5)$$

$$\rho_f(t) = \sum_{j \in N} (u_{j,t} - f_{j,t}) \quad (2.6)$$

$$\rho_f(i) = \min \left[ \sum_{j \in N} (u_{i,j} - f_{i,j}), \sum_{j \in N} (u_{j,i} - f_{j,i}) \right] \quad (2.7)$$

Thus, the flow potential measures the maximum possible flow that can pass through this node. Let  $\rho_f(r) = \min_{i \in N} (\rho_f(i))$ . We will call  $r$  the *reference node* and  $\rho_f(r)$  the *reference potential*.

**Lemma 9** *Let  $r$  be a reference node in an acyclic graph  $G$  with flow  $f$ . Then  $f$  can be augmented by  $\rho_f(r)$  to result in a flow  $f'$  with  $\rho_{f'}(r) = 0$ . This can be achieved by sending flow along paths with forward arcs only.*

**Proof:** Case (i):  $r = s$

Since  $\rho_f(s)$  is the minimum potential, an additional amount of flow of  $\rho_f(s)$  can be distributed among the outgoing arcs of the node  $s$  (Indeed this will saturate all of them and hence the network. The extra flow reaching the next node (in the numbering system that we mentioned before for the acyclic graph) can be distributed among its outgoing nodes because  $\rho_f(i) \geq \rho_f(s)$ . The same argument can be repeatedly applied. Hence the flow  $[f, F]$  can be augmented to  $[f', F']$  with  $F' - F = \rho_f(s)$ .

Case (ii):  $r = t$

This case is treated in a manner similar to that of (i).

Case (iii):  $r \neq s, t$

Delete all arcs  $(i, j)$  with  $i < r$  and  $j > r$  as shown in the diagram below. Now treat the graphs  $G_1$  and  $G_2$  separately and use cases (i) and (ii) on these graphs respectively. Note that the deleted arcs do not play any role in being able to send the flow of an amount  $\rho_f(r)$  forward and backwards.  $\square$

This saturates  $r$  in the original acyclic network. Saturated nodes and arcs are dropped from further consideration and new potentials are calculated along the way. By using proper data structures this is repeated at most  $n$  times and requires an effort of  $O(n^2)$ . Thus one cycle in Dinic's algorithm implemented this way takes  $O(n^2)$  effort and hence the entire algorithm is of  $O(n^3)$  for a general network. For sparse networks computer scientists have reduced this to slightly lower orders.

## 2.6 (Undirected) Edges, Parallel Arcs, Multiple Origins/Destinations, Node Capacities

An (undirected) edge with  $i$  and  $j$  as its terminals and a capacity of  $u_{i,j} > 0$  is replaced by two directed arcs  $(i, j)$  and  $(j, i)$  both with capacity equal to  $u_{i,j}$ . Parallel arcs are replaced by one arc whose capacity equals the sum of their capacities. In case there are multiple origins/destinations we create a *super source/super sink*; the super source is connected to each of the original sources and these arcs have infinite capacity. The super sink is connected to all destinations in the original network and these arcs also have infinite capacity. In the case of node capacity, first we make the network a directed one by the above construction if necessary. Now suppose node  $i$  has a capacity  $\alpha_i$ . Split this node into two – say  $i'$  and  $i''$  with an arc between them. Let all arcs of the type  $(j, i)$  be replaced by arcs of the type  $(j, i')$  and arcs of the type  $(i, j)$  by  $(i'', j)$  with the same capacity. Let the arc  $(i', i'')$  have capacity equal to  $\alpha_i$ . *Please note that if we started with an undirected network with node capacities this operation renders it a directed network; we know of no construction that will preserve the undirectedness. However, we may modify the algorithms directly if we are interested in retaining the undirected network.* We can also convert the case of arc/edge capacities to node capacities by inserting a node in the *middle* of every arc/edge and endow this node with the capacity of the arc/edge. *Thus, the conversion between node and arc capacities goes both ways in the case of directed network but the node capacity case seems more general in the case of undirected network.* This does cause some problems in the case multi terminal networks and this case is discussed there in detail.

## 2.7 Arc – Chain Formulation:

In the above *node-arc* formulation, the variables were *arc flows*. There is another formulation in which variables correspond to *s – t chain flows* (*directed s – t path flows*). Suppose  $A$  is an incidence matrix whose rows  $(i, j)$  correspond to arcs and columns to  $s – t$  chains;

$$a_{i,j}^k = \begin{cases} 1 & (i, j) \in C_k \\ 0 & \text{else} \end{cases}$$

Let  $x_k$  represent the flow along chain  $C_k$ . Then, we have the following formulation for the maximum flow problem:

$$[\max \sum_k x_k : x_k \geq 0; \sum_k a_{i,j}^k x_k \leq c_{i,j}]$$

This is called the *arc-chain formulation* of the maximum flow problem. Sometimes, instead of  $s – t$  chains we add an arc  $(t, s)$  whose capacity is

$\infty$  (and if there are lower bounds then the lower bound of this arc is  $-\infty$ ). Now the above chains correspond to directed circuits passing through the arc  $(t, s)$ . This formulation then easily extends to matroidal flows as in the works of Seymour et al.

### 2.7.1 Chain Decomposition of a feasible flow:

Suppose we have a feasible solution  $[f, F]$  to the flow problem in node arc form. To produce a set of chain flows (*not necessarily unique*) corresponding to this feasible solution we use the following process that we call *peeling process*.

#### Algorithm P

**Step 1:** Label  $s : (-, F)$ ; let  $S = \{s\}$ .

**Step 2:** For  $i \in S$ , and  $j \notin S$ , with  $f_{i,j} > 0$ , label  $j : (i^+, \epsilon_j)$  where  $\epsilon_j = \min(\epsilon_i, f_{i,j})$ ; let  $S^{new} = S^{old} \cup \{j\}$ . If  $t \in S$  go to step 3; else go to step 2.

**Step 3:** The label on  $t$  identifies a directed path with positive flow  $\epsilon_t$  on it. (This path is identified as in flow labeling algorithm before). Reduce  $F$  and all arc flows on this path by  $\epsilon_t$  and increase  $x$  of this path by  $\epsilon_t$ . If new value of  $F$  is zero then stop; we have the required decomposition. If some of the remaining flows are still positive, then these are cycle flows (also called *circulations*) and can be ignored without affecting the total flow  $F$ . If not, remove all labels and go to step 1 with new  $[f, F]$ .

This process shows that corresponding to any feasible set of arc flows there is at least one set of chain flows with the same total value. The reverse process is very easy; given an  $x$  to find  $f$ , let  $f = Ax$  and  $F = \sum_k x^k$ . *Actually this process works even in the undirected case; you have to watch out that opposite flows do not make a difference.* Please note that at every step one arc flow becomes zero in this algorithm; hence the number of steps in this algorithm can never exceed the number of arcs. *This shows that if we were clever, the number of flow augmentations in the flow labeling algorithm will not exceed the number of arcs and the algorithm would also never use reverse labels. The difficulty is that we do not know how to find such augmentations in general. There is a case where we do; it concerns the planar graph case.*

The first proof by Ford and Fulkerson [?] was using this formulation and not the node arc formulation. Their proof is used by Hoffman [H] in some other contexts that generalize maximum flow on networks.

## 2.8 Positive Lower Bounds and Exogenous Flows

Suppose, in the maximum flow problem, we have lower bounds, then we get the following problem:

$$\begin{aligned} & \max F \\ \text{s.t. } & \sum_j f_{i,j} - \sum_j f_{j,i} = \begin{cases} F & i = s \\ 0 & i \neq s, t \\ -F & i = t \end{cases} \\ & l_{i,j} \leq f_{i,j} \leq u_{i,j} \text{ for } (i,j) \in A \end{aligned}$$

Without loss, we may assume that  $u \geq l \geq 0$ . If we have a feasible flow, then minor modifications to the labeling algorithm will enable us to solve this problem. These are given below:

*Algorithm:*

Same as the algorithm before except the step “if  $i \in S, j \notin S, (j,i) \in A$  and  $f_{j,i} > 0$ , then label  $j : (i^-, \epsilon_j)$ , where  $\epsilon_j = \min(\epsilon_i, f_{j,i})$  and let  $j \in S$ ” is to be changed to read: “if  $i \in S, j \notin S, (j,i) \in A$  and  $f_{j,i} > l_{j,i}$ , then label  $j : (i^-, \epsilon_j)$ , where  $\epsilon_j = \min(\epsilon_i, f_{j,i} - l_{j,i})$  and let  $j \in S$ ”.

**Theorem 10**  $\max F = \min[u(S, \bar{S}) - l(\bar{S}, S)]$  where the minimum is over all cuts separating the origin and the destination.

**Theorem 11**  $\min F = \max[l(S, \bar{S}) - c(\bar{S}, S)]$ .

The main difficulty is to check whether or not there is a feasible solution and to produce one if it exists – somewhat like Phase I of the simplex method. The trick is to use this theory itself to do that. Before we do that let us describe a closely related area of exogenous flows. In this case we allow the flow at nodes other than the origin and destination to be not conserved – there may flow to or from the “outside”. The resulting problem is of the following form:

$$\begin{aligned} & \max F \\ \text{s.t. } & \sum_j f_{i,j} - \sum_j f_{j,i} = \begin{cases} F + q_s & i = s \\ 0 + q_i & i \neq s, t \\ -F + q_t & i = t \end{cases} \\ & l_{i,j} \leq f_{i,j} \leq u_{i,j} \text{ for } (i,j) \in A \end{aligned}$$

where  $q$  is a specified vector. In this problem we may assume that  $l = 0$  without loss of generality.

**Exercise:** Show that this can be converted to the problem with lower bounds but no exogenous flows.

**Feasible Flows:**

Consider the problem with lower bounds that are nonnegative. Let  $G = [N; A]$  be the original graph. Define a new graph  $G' = [N'; A']$  as follows:

$N' = N \cup \{\alpha, \beta\}$ , and  $(i, j) \in A \implies (i, j), (\alpha, j), (i, \beta)$  all belong to  $A'$ ;  $(t, s) \in A'$ . The capacities are given as follows: for  $(i, j)$  with bounds  $l_{i,j}, u_{i,j}$  the corresponding new arcs have capacities:  $u_{i,j} - l_{i,j}, l_{i,j}$ , and  $l_{i,j}$  respectively. Of course, their lower bounds are all zero. The arc  $(t, s)$  has as its bounds  $(-\infty, \infty)$ . Please note that this does not pose any problems in the starting solution of 0. We maximize the flow between  $\alpha$  and  $\beta$ .

Note that in the network  $G'$  there are two cuts separating  $\alpha$  and  $\beta$  whose capacity is  $L = \sum_{(i,j) \in A} l_{i,j}$ . Hence the max flow  $F^*$ , in this network satisfies the relation  $F^* \leq L$ .

**Theorem 12** *A feasible solution to the original problem exists iff  $F^* = L$ .*

**Exercise:** Read chapter 2 of F & F.

## References

- [FF1] L.R. Ford and D.R. Fulkerson: Maximal Flow Through a Network, *Canad. J. Math* , 8. (1956), pp. 399-404.
- [EFS] P. Elias, A. Feinstein. and C.E. Shannon: Note on Maximal flow Through a Network, *IRE Trans. on Information Theory*, IT-2, (1956), pp. 117-119.
- [R] J.T.Robacker: On Network Theory, Rand Research Memorandum, RM-1498, May 1955.
- [FF2] L.R. Ford and D.R. Fulkerson: *Flows in Networks*, Princeton University Press, 1962.
- [D] E.A. Dinic: Algorithm for a Solution of a Problem of Maximal Flow in a Network with Power Estimation, *Sov. Math. Dokl.*, 11, (1970), pp. 1277-1280. (see [E] for a nicer explanation)
- [E] S. Even: *Graph Algorithms*, Computer Science press, (1979).
- [EK] J. Edmonds and R. M. Karp: Theoretical improvements in Algorithmic Efficiency for Network Flow problems, *JACM*, 19, (1972), pp. 248-264.
- [H] A.J. Hoffman: A Generalization of Max Flow-Min Cut, *Math. Prog.* 6, (1974), pp. 352-359.
- [Z] N. Zadeh: A Bad Network Problem for the Simplex method and other minimum Cost Flow Algorithms, *Math. Prog.*, 5, (1973), pp. 255-266.

- [K] A.V.Karzanov: Determining the Maximal Flow in a Network with the Pre flows, *Sov. Math. Dokl.*, 15, (1974), pp., 434-437.
- [C] B.V.Cherkaski: Algorithm for Construction of Maximal Flows in Networks with Complexity  $O(|V| \sqrt{|E|})$  Operations, *Math. methods of Solutions of Economic Problems*, 7, (1977), pp. 117-125.
- [G] Z. Galil: A New Algorithm for the Maximal Flow problem, *Proc. 19<sup>th</sup> Symposium on Foundations of Computer Science*, (1978), pp. 231-245.
- [GN] Z. Galil and A. Noamad: Network Flow and Generalized Path Compression, *11<sup>th</sup> Annals ACM Symp. on Theory of Computing*, (1979), pp. 13-26.
- [MKM] V.M. Malhotra, M.P. Kumar, and S.N. Maheshwari: An  $O(|V|^3)$  Algorithm for Finding Maximum Flows in Networks, *Information Processing Letters*, 6, (1978) pp. 277-278.
- [IS] A. Itai and Y. Shiloach: Maximum Flow in Planar Networks, *SIAM Jour. on Computing*, 8, (1979), pp. 135-150.
- [ST] D.D. Sleator and R.E Tarjan: A data Structure for Dynamic Trees, *J. Comp. Sys. Sci.*, 24, (1983), pp. 362-391.
- [T] A.C.Tucker: A Note on Convergence of the Ford-Fulkerson Flow Algorithm, *Math. of O.R.*, 2, (1977), pp. 143-144.
- [PQ] J.C. Picard and M.Queyranne: On the Structure of all Minimal Cuts in a Network and Applications, *Math. Prog. Stud.* 13, pp. 8-16.
- [Q] M.Queyranne: Theoretical Efficiency of the Algorithm "Capacity" for the maximum Flow problem, *Math. of O.R.*, 5, (1980), pp. 258-266.
- [AO] R.K. Ahuja and J.B. Orlin: A Fast and Simple Algorithm for the Maximum Flow Problem, *MIT Report*, (1987).
- [AOM] R.K. Ahuja, J.B. Orlin, and T. Magnanti: *Network Flows*, Aug 1988.
- [P] Ponstein, J.: On Maximal Flow Problems with Real Arc Capacities, *Math. Prog.* **3**, (1972), pp 254-256.
- [Z2] Zadeh, N.: More Pathological Examples for Network Flow Problems, *Math. Prog.* **5**, (1973), pp 217-224.
- [MG] Megiddo, N., Galil, Z: On Fulkerson's Conjecture about Consistent Labeling Processes, *Math. of Oper. Res.*, 4, (1979), pp 265-267.