CSC 364S Notes **University of Toronto, Spring, 2003**

Flow Algorithms

The networks we will consider are directed graphs, where each edge has associated with it a nonnegative *capacity*. The intuition is that if edge (u, v) has capacity c, then this means that at most c amount of "stuff" (for example water, or electricity, or bits) can flow along the edge from u to v. We will also have two distinguished vertices s and t (for *source* and *terminus*), and the goal is to compute the maximum amount of "stuff" that can be made to flow through the network from s to t. Of course, we also have to obey the "flow conservation" constraint that for every vertex v other than s or t, the total amount of stuff flowing into v is equal to the total amount flowing out of v. Actually, instead of having the two notions of stuff flowing into a vertex and of stuff flowing out of a vertex, we will allow *negative* flows along an edge; if there is a flow of x on the edge (u, v), then there will be a flow of $-x$ on the edge (v, u). The flow conservation constraint then becomes the statement that the total amount flowing out of vertex v (other than s or t) is exactly 0; this is equivalent to saying that the total amount flowing *into* v is exactly 0. We now state all this formally.

A *Flow network* $\mathcal{F} = (G, c, s, t)$ consists of:

- a directed graph $G = (V, E)$; actually, G is *bidirected*, that is, if $(u, v) \in E$ then $(v, u) \in E$; we also assume there are no self loops;
- a nonnegative capacity function $c : E \rightarrow \mathbb{R}^{\geq 0}$;
- two distinguished vertices: s (the source) and t (the terminus or target).

Note: The terminology "flow network" is pretty standard, but in fact it should be called a "capacity network" (or perhaps to be more grammatical, a "capacitated network") to stress the fact that we start with capacities and not flows.

We can assume (without loss of generality) that for every $v \in V$, there is a path s to t, and we can also assume that for all $(u, v) \in E$, either $c(u, v) > 0$ or $c(v, u) > 0$ (or both).

We now define the notion of a *flow* in G. Note that unlike capacities, flow values are allowed to be negative. For a vertex u, we use the notation $N(u)$ to denote the *neighbors* of u: $N(u) = \{v \in V \mid (u, v) \in E\}$.

A *flow* in G is a function $f : E \to \mathbb{R}$ satisfying:

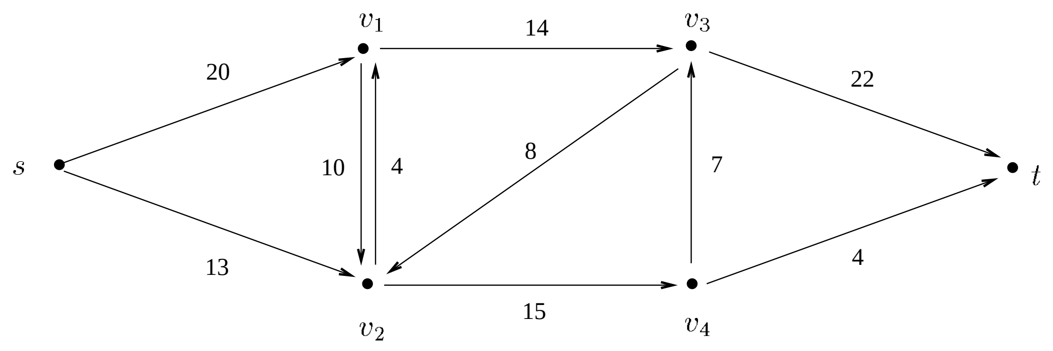
- capacity constraint: $f(u,v) \le c(u,v)$ for every $(u,v) \in E$
- skew symmetry: $f(u,v) = -f(v,u)$ for every $(u,v) \in E$
- flow conservation: For all $u \in V$ other than s or t, $\displaystyle\sum_{v \in N(u)} f(u,v) = 0$

The *value* of a flow f is $|f| = \displaystyle\sum_{v \in N(s)} f(s,v)$.
Note that although we defined $|f|$ to be the amount flowing out of s, we will see later that this is equal to the amount flowing into t.

Maximum flow problem:
Given a flow network (G, c, s, t), find a flow of maximum possible value from s to t.

Example: The following is an example of a flow network. Only edges with positive capacities are shown; edges with capacity 0 are omitted from the diagram. Each edge is labelled with its capacity. For each edge (u,v) that is shown, the edge (v,u) is also assumed to be present; if the edge (v,u) is not shown, then it has capacity 0.

FLOW NETWORK $\mathcal{F}$:

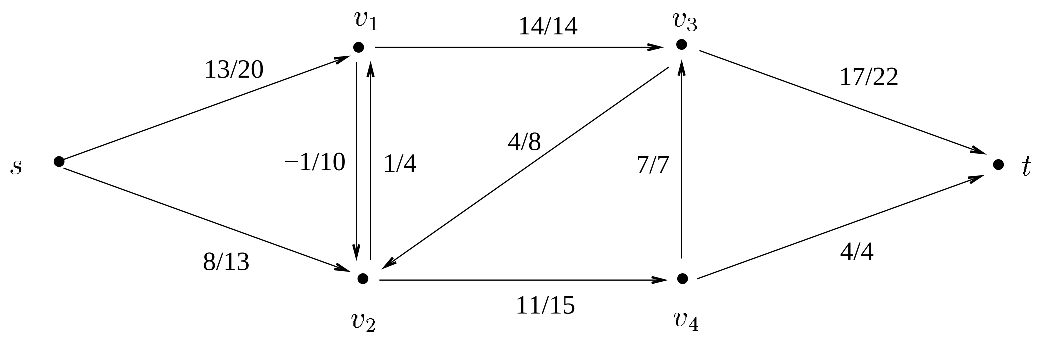


The following shows the above example of a flow network, together with a flow.
The notation x/y on an edge (u,v) means

x is the flow ($x = f(u,v)$)
y is the capacity ($y = c(u,v)$)

Only flows on edges of positive capacity are shown. Note that negative flows on edges of 0 capacity are not shown. For example, $f(v_2, s) = -8$, but this is not explicitly shown on the diagram. In this example we have:
$|f| = 13 + 8 = 21$.

FLOW NETWORK $\mathcal{F}$ WITH A FLOW f:



Residual Networks

Let $\mathcal{F}$ be a flow network, f a flow. For any $(u, v) \in E$, the *residual capacity* of (u, v) induced by f is

$$c_f(u, v) = c(u, v) - f(u, v) \geq 0.$$

The *residual graph* of G induced by f is

$$G_f = (V, E_f)$$

where

$$E_f = \{(u, v) \in E \mid c_f(u, v) > 0\}.$$

The flow f also gives rise to the residual flow network $\mathcal{F}_f = (G, c_f, s, t)$.

The residual network $\mathcal{F}_f$ is itself a flow network with capacities c_f, and any flow in $\mathcal{F}_f$ is also a flow in $\mathcal{F}$.

The next lemma shows that if we have a flow f in a network and a flow f_0 in the residual network, then f_0 can be added to f to obtain an improved flow in the original network. This will be a technique we will use to continuously improve flows until we have a maximum possible flow.

Lemma 1 *Let $\mathcal{F} = (G, c, s, t)$ be a flow network and let f be a flow in $\mathcal{F}$. Let $\mathcal{F}_f$ be the residual network induced by f, and let f_0 be a flow in $\mathcal{F}_f$. Then the flow sum defined by*

$$(f + f_0)(u, v) \overset{d}{=} f(u, v) + f_0(u, v)$$

is a flow in $\mathcal{F}$ with value

$$|f + f_0| = |f| + |f_0|.$$

Proof: In order to show that $f + f_0$ is a flow in $\mathcal{F}$, we have to check that the capacity constraint, skew symmetry, and flow conservation hold.

Capacity Constraint:
$(f+f_0)(u,v) = f(u,v)+f_0(u,v) \leq f(u,v)+c_f(u,v) = f(u,v)+(c(u,v)-f(u,v)) = c(u,v)$.

Skew Symmetry:
$(f+f_0)(u,v) = f(u,v)+f_0(u,v) = -f(v,u) - f_0(v,u) = -(f(v,u)+f_0(v,u)) =$
$-(f+f_0)(v,u)$.

Flow Conservation:
Let u be a vertex other than s or t. Then $\sum_{v\in N(u)}(f+f_0)(u,v) =$
$\sum_{v\in N(u)}(f(u,v)+f_0(u,v)) = \sum_{v\in N(u)} f(u,v) + \sum_{v\in N(u)} f_0(u,v) = 0+0 = 0$.

It remains to check that $|f+f_0| = |f| + |f_0|$. We have $|f+f_0| = \sum_{v\in N(s)}(f+f_0)(s,v) =$ $\sum_{v\in N(s)}(f(s,v)+f_0(s,v)) = \sum_{v\in N(s)} f(s,v) + \sum_{v\in N(s)} f_0(s,v) = |f|+|f_0|$. □

Augmenting Paths

Given a flow network $\mathcal{F} = (G,c,s,t)$ and a flow f, an *augmenting path* π is a simple path (that is, a path where no vertex repeats) from s to t in the residual graph, G_f; note that every edge in G_f has positive capacity. Equivalently, an augmenting path is a simple path from s to t in G consisting only of edges of positive residual capacity. We will use an augmenting path to create a flow f_0 of positive value in $\mathcal{F}_f$, and then add this to f as in the above lemma, in order to create the flow $f' = f + f_0$ of value bigger than f.

The maximum amount of net flow we can ship along the edges of an augmenting path π is called the *residual capacity* of π. We denote it by $c_f(\pi)$; because π is augmenting, $c_f(\pi)$ is guaranteed to be positive.

$$c_f(\pi) = \min\{c_f(u,v) \mid (u,v) \text{ is on } \pi\} > 0.$$

Lemma 2 *Fix flow network $\mathcal{F} = (G,c,s,t)$, flow f, augmenting path π, and define $f_\pi : E \rightarrow \mathbb{R}$:*

$$f_\pi(u,v) = \begin{cases} c_f(\pi) & \textit{if } (u,v) \textit{ is on } \pi \\ -c_f(\pi) & \textit{if } (v,u) \textit{ is on } \pi \\ 0 & \textit{otherwise} \end{cases}$$

Then f_π is a flow in $\mathcal{F}_f$, and $|f_\pi| = c_f(\pi) > 0$.

Proof: In order to show that f_π is a flow in $\mathcal{F}_f$, we have to check that the capacity constraint, skew symmetry, and flow conservation hold.

Capacity Constraint:
If (u,v) is on π, then $f_\pi(u,v) = c_f(\pi) \leq c_f(u,v)$; this last inequality holds since by definition,

$c_f(\pi)$ is the smallest residual capacity of the edges of π.
If (u,v) is not on π, then $f_\pi(u,v) \le 0 \le c_f(u,v)$.

Skew Symmetry:
If (u,v) is on π, then we have $f_\pi(u,v) = c_f(\pi)$, and $f_\pi(v,u) = -c_f(\pi)$, so
$f_\pi(u,v) = -f_\pi(v,u)$.
If neither (u,v) nor (v,u) is on π, then $f_\pi(u,v) = f_\pi(v,u) = 0$, so $f_\pi(u,v) = -f_\pi(v,u)$.

Flow Conservation:
Let u be a vertex other than s or t.
Let us first consider the case where u is on π. Say that edges (w,u) and (u,v) are on π. Then $f_\pi(u,v) = c_f(\pi)$ and $f_\pi(u,w) = -c_f(\pi)$; for every $x \in N(u)$ other than v or w, we have $f_\pi(u,x) = 0$. So $\sum_{x\in N(u)} f_\pi(u,x) = 0$.
Now consider the case where u is *not* on π. Then for every $x \in N(u)$, $f_\pi(u,x) = 0$. So $\sum_{x\in N(u)} f_\pi(u,x) = 0$.

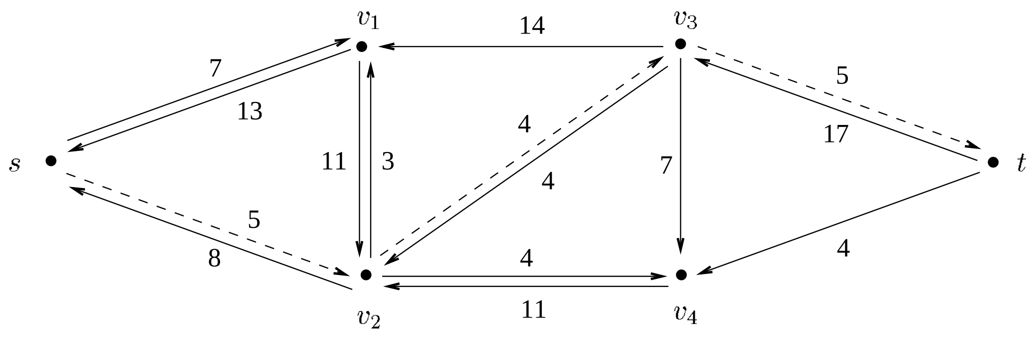
It remains to check that $|f_\pi| = c_f(\pi)$. Let (s,v) be the unique edge coming out of s that is on π. Then $f_\pi(s,v) = c_f(\pi)$. For every $x \in N(s)$ other than v, we have $f_\pi(s,x) = 0$. So $\sum_{x\in N(s)} f_\pi(s,x) = c_f(\pi)$. □

Corollary 1 *Fix flow network $\mathcal{F} = (G,c,s,t)$, flow f, augmenting path π, and let f_π be defined as above. Let $f' = f + f_\pi$. Then f' is a flow in $\mathcal{F}$, and*

$$|f'| = |f| + |f_\pi| > |f|.$$

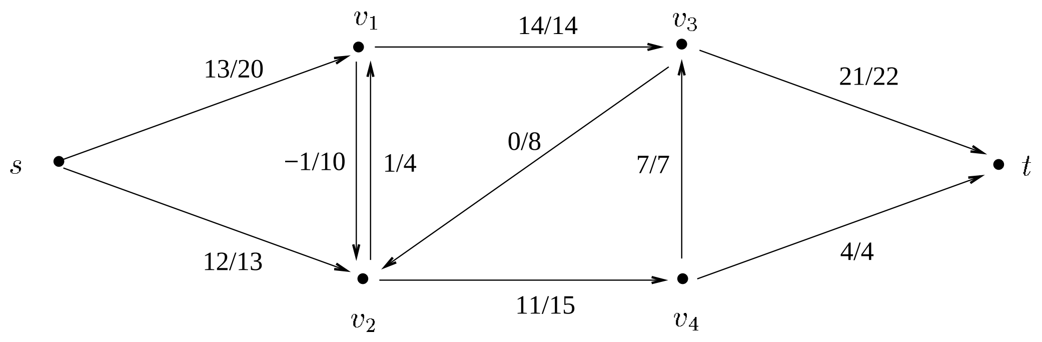
Example: Continuing the previous example, the following diagram shows the residual graph G_f consisting of edges with positive residual capacity. The residual capacity of each edge is also shown. An augmenting path π is indicated by $- - -$. We have
$c_f(\pi) = 4$.

THE RESIDUAL GRAPH G_f WITH AUGMENTING PATH π:



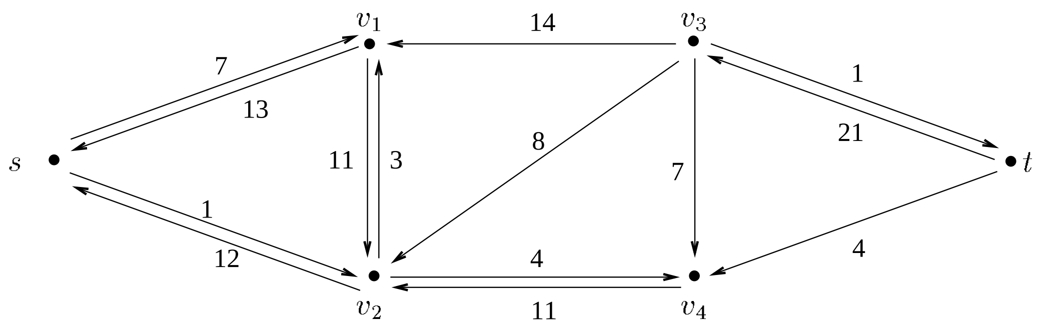
The following diagram shows the network $\mathcal{F}$ with flow $f' = f + f_\pi$. We have
$|f'| = |f| + |f_\pi| = 21 + c_f(\pi) = 21 + 4 = 25$.

FLOW NETWORK $\mathcal{F}$ WITH FLOW f':



After creating the improved flow f', it is natural to try the same trick again and look for an augmenting path with respect to f'. That is, we consider the new residual graph $G_{f'}$ and look for a path from s to t. We see however, that no such path exists.

THE RESIDUAL GRAPH $G_{f'}$:



All of the above suggests the famous *Ford-Fulkerson* algorithm for network flow. The algorithm begins by initializing the flow f to the all-0 flow, that is, the flow that is 0 along every edge. The algorithm then continually improves f by searching for an augmenting path π, and using this path to improve f, as in the previous lemma. The algorithm halts when there is no longer any augmenting path.

```
FORD-FULKERSON($G, c, s, t$)

Initialize flow $f$ to the all-0 flow
WHILE there exists an augmenting path in $G_f$ DO
    choose an augmenting path $\pi$
    $f \leftarrow f + f_\pi$
end WHILE
```

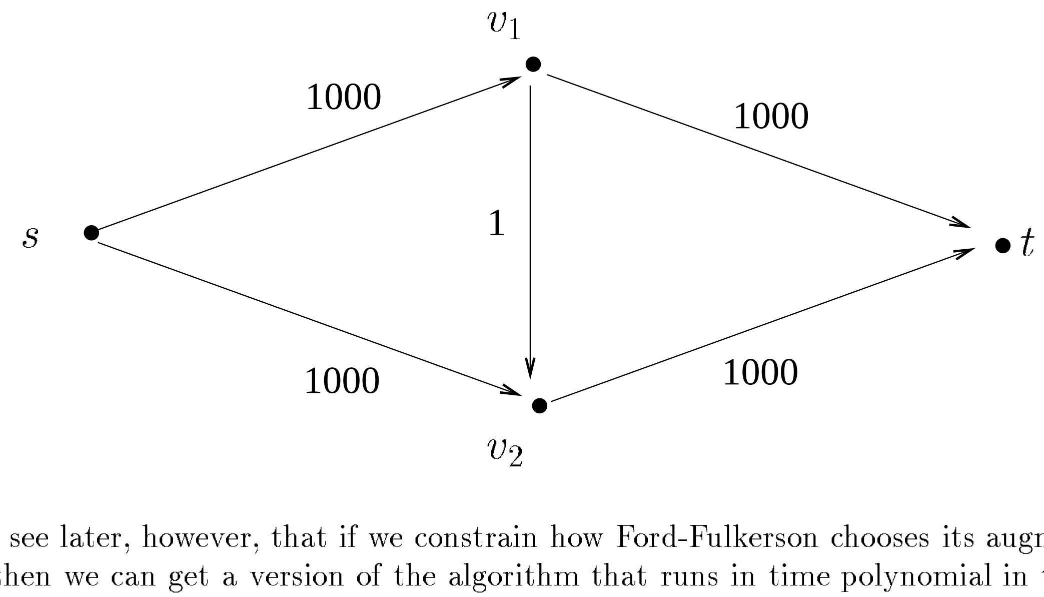
There are a number of obvious questions to ask about this algorithm. Firstly, how are we supposed to search for augmenting paths? That is, how do we look for a path from s to t in the graph G_f? There are many algorithms we could use. However, since all we want to do is find a path between two points in an unweighted, directed graph, two of the simplest and fastest algorithms we can use are "depth-first search" or "breadth-first" search. Each of these algorithms runs in time linear in the size of the graph, that is, linear in the number of edges in the graph.

The next question is, is the algorithm guaranteed to halt? The answer to this, remarkably, is *NO.* If we do not constrain how the algorithm searches for augmenting paths, then there are examples where it can run forever. These examples are complicated and use irrational capacities, and we will not show one here.

What if the capacities are all integers? Then it is clear that the algorithm increases the flow by at least 1 each time through the loop, and so it will eventually halt. (Exercise: fill in the details of this argument; give a similar argument in case the capacities are only guaranteed to be rational numbers.) However, this may still take a very long time.

As an example, consider the following flow network with only 4 vertices. If we are lucky (or careful), the algorithm will choose $[s, v_1, t]$ for the first augmenting path, creating a flow with value 1000; it will then have no choice but to choose $[s, v_2, t]$ for the next augmenting path, and it will be halt, having created a flow with value 2000. However, the algorithm *may* choose $[s, v_1, v_2, t]$ as its first augmenting path, creating a flow with value 1; it *may* then choose $[s, v_2, v_1, t]$ as the next augmenting path, creating a flow with value 2; continuing in this way, it *may* go 2000 times through the loop before it eventually halts.

EXAMPLE OF A BAD FLOW NETWORK FOR FORD-FULKERSON:



We will see later, however, that if we constrain how Ford-Fulkerson chooses its augmenting paths, then we can get a version of the algorithm that runs in time polynomial in the size of the network.

For the moment, we will concern ourselves with one last question about the algorithm. Let's assume it *does* halt; is it then the case that the flow it has found is as large as possible? The answer turns out to be *YES*! We know that if an augmenting path exists then the current flow is not optimal. We want to prove that if there is *no* augmenting path, then the current flow *is* optimal.

This is a very subtle proof. Let us fix flow network $\mathcal{F} = (G, c, s, t)$, $G = (V, E)$, and flow f. We are going to introduce the new notion of a *cut* of $\mathcal{F}$. We will see that if there is no augmenting path, then there will exist a special cut that shows that f is optimal.

Cuts of Flow Networks

A *cut* (S, T) of $\mathcal{F}$ is a partition of V into S and $T = V - S$ such that $s \in S$ and $t \in T$. We define the *capacity* of (S, T) to be the sum of the capacities over all edges going from S to T; note that this is a sum of nonnegative numbers. We define the *flow* across (S, T) to be the sum of the flows over all edges going from S to T; note that this sum may consist of negative numbers. More formally:

The *capacity* of the cut (S, T) is defined by

$$c(S,T) = \sum_{(x,y)\in(S\times T)\cap E} c(x,y)$$

The *flow* across (S, T) is

$$f(S,T) = \sum_{(x,y)\in(S\times T)\cap E} f(x,y)$$

Note that we are slightly abusing notation by extending the definition of flows and capacities to sets of nodes.

Example: Consider our earlier example of the flow network $\mathcal{F}$ with flow f'. Consider the cut $(S, T) = (\{s, v_3\}, \{t, v_1, v_2, v_4\})$. We have $c(S, T) = 20 + 13 + 8 + 22 = 63$ and $f'(S, T) = 13 + 12 + (-14) + (-7) + 21 = 25$.

We see that $f(S, T)$ in the above example is exactly equal to $|f|$, and this is no coincidence. Intuitively it makes sense that the amount flowing out of s should be exactly the same as the amount flowing across any cut, and this is proven in the next lemma. In particular, by considering the cut $(V - \{t\}, \{t\})$, we see that $|f|$ is exactly equal to the amount flowing into t.

Lemma 3 *Fix flow network $\mathcal{F} = (G, c, s, t)$ and flow f. Then for every cut (S, T), $f(S, T) = |f|$.*

Proof: Let (S,T) be a cut. Consider the following five sets of edges:
$E_1 = (S \times T) \cap E$;
$E_2 = (S \times V) \cap E$;
$E_3 = (S \times S) \cap E$;
$E_4 = (\{s\} \times V) \cap E$;
$E_5 = ((S - \{s\}) \times V) \cap E$;

First, we claim that $\sum_{e \in E_3} f(e) = 0$. This is because for every edge (u,v) in E_3, (v,u) is also in E_3, and by skew symmetry we have $f(u,v) + f(v,u)$=0.

We next claim that $\sum_{e \in E_5} f(e) = 0$. This is because flow conservation for every node u other than s or t allows us to write
$\sum_{e \in E_5} f(e) = \sum_{u \in S-\{s\}} \sum_{v \in N(u)} f(u,v) = \sum_{u \in S-\{s\}} 0 = 0$.

Since E_2 is the disjoint union of E_1 and E_3, we have
$\sum_{e \in E_2} f(e) = \sum_{e \in E_1} f(e) + \sum_{e \in E_3} f(e) = \sum_{e \in E_1} f(e)$.

Since E_2 is the disjoint union of E_4 and E_5, we have
$\sum_{e \in E_2} f(e) = \sum_{e \in E_4} f(e) + \sum_{e \in E_5} f(e) = \sum_{e \in E_4} f(e)$.

Putting all this together, we have
$f(S,T) = \sum_{e \in E_1} f(e) = \sum_{e \in E_2} f(e) = \sum_{e \in E_4} f(e) = |f|$. □

Lemma 4 *Fix flow network $\mathcal{F} = (G,c,s,t)$ and flow f. Then for every cut (S,T), $f(S,T) \le c(S,T)$.*

Proof: $f(S,T) = \sum_{e \in (S \times T) \cap E} f(e) \le \sum_{e \in (S \times T) \cap E} c(e) = c(S,T)$. □

Corollary 2 *The value of every flow in $\mathcal{F}$ is less than or equal to the capacity of every cut of $\mathcal{F}$.*

We now state and prove the famous "max-flow, min cut" theorem. This theorem says that the maximum value over all flows in $\mathcal{F}$ is exactly equal to the minimum capacity over all cuts. It also tells us that if $\mathcal{F}$ has no augmenting paths with respect to a flow f, then $|f|$ is the maximum possible.

Theorem 1 *(MAX-FLOW, MIN-CUT THEOREM)*
Fix flow network $\mathcal{F} = (G,c,s,t)$, $G = (V,E)$, and flow f. Then the following are equivalent

1. *f is a max flow (that is, a flow of maximum possible value) in $\mathcal{F}$.*
2. *There are no augmenting paths with respect to f.*
3. *$|f| = c(S,T)$ for some cut (S,T) of $\mathcal{F}$.*

Proof:
(1) $\Rightarrow$ (2)
Suppose (1) holds. We have already seen that if there were an augmenting path with respect to f, then a flow with value larger than $|f|$ could be constructed. Since f is a max flow, there must be no augmenting paths.

(2) $\Rightarrow$ (3)
Suppose (2) holds. Then there is no path from s to t in G_f.
Let $S = \{v \in V \,|\, \text{there exists a path from } s \text{ to } v \text{ in } G_f\}$, and let $T = V - S$. Clearly (S, T) is a cut. We claim that $|f| = c(S, T)$. From the above Lemma 3, it suffices to show that $f(S, T) = c(S, T)$. For this, it suffices to show that for every edge $(u, v) \in (S \times T) \cap E$, $f(u, v) = c(u, v)$. So consider such an edge (u, v). If we had $f(u, v) < c(u, v)$, then (u, v) would be an edge with positive residual capacity, and hence (u, v) would be an edge of G_f, and hence (since $u \in S$), there would be a path in G_f from s to v, and hence $v \in S$ – a contradiction.

(3) $\Rightarrow$ (1)
Suppose (3) holds. Let (S, T) be a cut of $\mathcal{F}$ such that $|f| = c(S, T)$. From the above corollary, we know that every flow has value less than or equal to $c(S, T)$, and hence every flow has value less than or equal to $|f|$. So f is a max flow. $\square$

This theorem tells us that if Ford-Fulkerson halts, then the resulting flow is optimal. Let us now return to the question of how to modify Ford-Fulkerson so that it is guaranteed to not only halt, but to halt reasonably quickly.

The Edmonds-Karp version of this algorithm looks for a path in G_f using *breadth-first search.* This finds a path that contains as few edges as possible. We call this algorithm FF-EK:

```
FF-EK($G, c, s, t$)

Initialize flow $f$ to the all-0 flow
WHILE there exists an augmenting path in $G_f$ DO
    choose an augmenting path $\pi$ using breadth-first search in $G_f$
    $f \leftarrow f + f_\pi$
end WHILE
```

Let us assume (without loss of generality) that $|E| \geq |V|$. Then breadth-first search finds an augmenting path (if there is one) in time $O(|E|)$. Using an augmenting path to improve the flow takes time $O(|E|)$, so each execution of the main loop runs in time $O(|E|)$.

It is a difficult theorem (that we will not prove here) that the main loop of FF-EK will be executed at most $O(|V||E|)$ times. Thus, FF-EK halts in time $O(|V||E|^2)$. Hence, this is a polynomial time algorithm. A huge amount of research has been done in this area, and even better algorithms have been found. One of the fastest has running time $O(|V|^3)$.

Example: Consider the previous example of flow network $\mathcal{F}$ with flow f'. We know that $|f'| = 25$, and so the flow across every cut will be 25, and the capacity of every cut will be greater than or equal to 25. We have seen that there is no augmenting path, so the above theorem tells us that there must be a cut of capacity 25. It even tells us how to find such a cut: let S be the set of nodes reachable from s in $G_{f'}$. In fact, it is easy to check that if we choose $S = \{s, v_1, v_2, v_4\}$ and $T = \{v_3, t\}$, then $c(S,T) = 14 + 7 + 4 = 25$.

An Application to Finding Maximum Matchings in Bipartite Graphs

There are many applications of max-flow algorithms to areas which appear very different. As an example, we shall show how to use a max flow algorithm to find a maximum matching in a bipartite graph.

A bipartite graph is a directed graph $G = (V, E)$ where V is partitioned into two parts L and R (for left and right) such that all edges go from L to R. That is, $V = L \cup R$, $L \cap R = \emptyset$, and $E \subseteq L \times R$. A *matching* is defined to be a set of edges $M \subseteq E$ such that no two (distinct) edges of M have a vertex in common. The size of M is defined to be $|M|$, the number of edges in M. We wish to find a maximum matching, that is, a matching of biggest possible size.

It turns out that there is a way to convert such a matching problem into a flow problem. First we add two vertices to create $V' = V \cup \{s, t\}$. We add edges from s to each vertex in L, and edges from each vertex in R to t; all of these edges (including the original edges in E) are assigned capacity 1. Lastly, we add the the reverse of all these edges, with capacity 0. In this way we form the flow network $\mathcal{F} = (G', c, s, t)$, $G' = (V', E')$. Consider integer flows in $\mathcal{F}$, that is, flows that take on integer values on every edge; for each edge of capacity 1, the flow on it must be either 1 or 0 (since its reverse edge has capacity 0). It is easy to see that Ford-Fulkerson, when applied to a network with only integer capacities, will always yield an integer (maximum) flow. The following two lemmas show that an integer flow f can be used to construct a matching of size $|f|$, and that a matching M can be used to construct a flow of value $|M|$. This will allow us to use Ford-Fulkerson to compute a maximum flow in polynomial time.

Lemma 5 *Let $G = (V, E)$ and $\mathcal{F} = (G', c, s, t)$ be as above, and let f be an integer flow in $\mathcal{F}$. Then there is a matching M in G such that $|M| = |f|$.*

Proof:
Let f be an integer flow. Let $M = \{(u, v) \in L \times R \,|\, f(u, v) = 1.\}$.

To see why M is a matching, imagine that $(u, v_1), (u, v_2) \in M$ for some $v_1 \neq v_2$; since u has only one edge of positive capacity (namely 1) coming into it, we would have $\sum_{v \in N(u)} f(u, v) \geq 1$, contradicting flow conservation. (A similar argument shows that no two edges in M can share a right endpoint.)

We now show that $|M| = |f|$. Recall that $|f|$ is equal to the total flow coming out of s. So we must have $|f|$ distinct vertices $u_1, u_2, \ldots, u_{|f|}$ such that
$f(s,u_1) = 1, f(s,u_2) = 1, \ldots, f(s,u_{|f|}) = 1$. So in order for flow conservation to hold, for each u_i we must have some $v_i \in R$ such that $f(u_i, v_i) = 1$. So $(u_i, v_i) \in M$ for each i, and we have $|M| = |f|$. $\square$

Lemma 6 *Let $G = (V, E)$ and $\mathcal{F} = (G', c, s, t)$ be as above, and let M be a matching in G. Then there exists a flow f in $\mathcal{F}$ with $|f| = |M|$.*

Proof:
Let $M = \{(u_1, v_1), (u_2, v_2), \ldots, (u_{|M|}, v_{|M|}) \subseteq L \times R\}$ be a matching. Define f by
$f(s, u_i) = 1$, $f(u_i, v_i) = 1$, $f(v_i, t) = 1$ (and $f(u_i, s) = -1$, $f(v_i, u_i) = -1$, $f(t, v_i) = -1$) for each i, and $f(e) = 0$ for every other edge $e \in E'$. It is easy to check that f is a flow, and that $|f| = |M|$ (exercise). $\square$

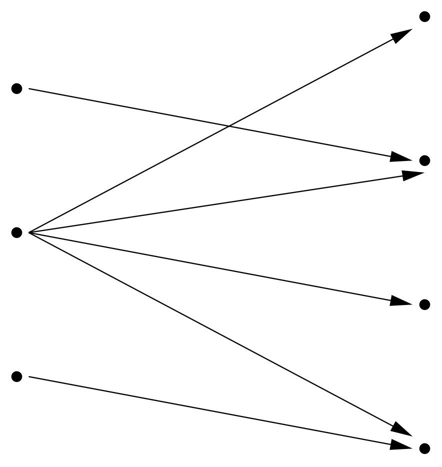
We now see how to use Ford-Fulkerson to find a maximum matching in $G = (V, E)$. Assume, without loss of generality, that $|V| \leq |E|$. Note that $|V'| \in O(|V|)$ and $|E'| \in O(|E|)$.

We first construct $\mathcal{F} = (G', c, s, t)$ as above; this takes time $O(|E|)$. We then perform the Ford-Fulkerson algorithm to create a maximum flow f. We observe that this algorithm, no matter how we find augmenting paths, will increase the flow value by exactly 1 each time, and hence will execute its main loop at most $|V|$ times. If we use an $O(|E|)$ time algorithm to search for augmenting paths, then each execution of the loop will take time $O(|E|)$. So the total time of the Ford-Fulkerson algorithm here is $O(|V||E|)$. Lastly, we use the integer flow f to create a matching M in G such that $|M| = |f|$; this takes time $O(|E|)$. The last lemma above tells us that since f is a maximum flow, M must be a maximum matching.

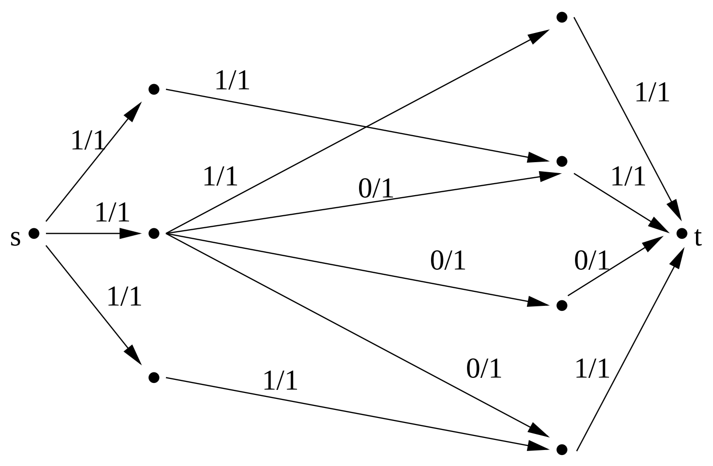
So the entire maximum matching algorithm runs in time $O(|V||E|)$. This is a polynomial time algorithm. Faster algorithms have also been found. For example, there is an algorithm for this problem that runs in time $O(\sqrt{|V|}|E|)$.

Example: The following is an example of a (directed) bipartite graph G. The next figure shows the network $\mathcal{F}$ derived from G, together with a maximum flow f in $\mathcal{F}$. The last figure shows the maximum matching M obtained from f.

A BIPARTITE GRAPH G:



FLOW NETWORK $\mathcal{F}$ DERIVED FROM G, WITH A MAXIMUM FLOW f:



A MAXIMUM MATCHING IN G DERIVED FROM FLOW f: