| **COMPSCI 532: Design and Analysis of Algorithms** | August 26, 2015 |
|---|---|
| Lecture 1 | |
| *Lecturer: Debmalya Panigrahi* | *Scribe: Allen Xiao* |

# 1 Overview

In this lecture, we will define the maximum flow problem and discuss two algorithms for max flow based on augmenting paths. Along the way we will prove the flow decomposition lemma and the max-flow min-cut theorem, two useful tools for analyzing many network flow algorithms.

# 2 Maximum Flow

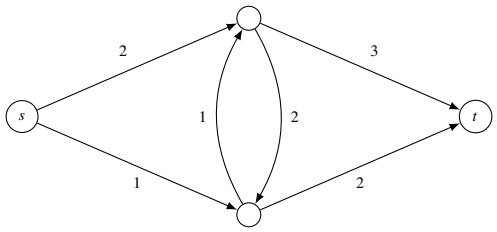


Figure 1: An example of a network. The edge labels are capacities.

**Definition 1.** *The input to the maximum flow problem comprises a directed graph (also called a network) $G = (V,E)$ with a source $s \in V$ and sink $t \in V$. Every edge $e = (v,w)$ has a nonnegative capacity $u(e) = u(v,w)$. If an edge $e$ is not in $E$, we give it capacity $u(e) = 0$.*

In network flow problems, the goal of the algorithm is to assign a *flow* to each edge, which we define in two ways:

**Definition 2.** *The **raw flow** $r(v,w)$ is a nonnegative function on directed edges.*

$$r : E \to \mathbb{R}_{\geq 0}$$

*Raw flow must satisfy the properties:*

1. *Conservation/Balance:*
$$\sum_{x \in V} r(v,x) = \sum_{x \in V} r(x,v) \qquad \forall v \neq s,t$$
*"For each vertex other than the source and the sink, incoming raw flow must equal the outgoing raw flow."*

2. *Capacity constraint:*
$$0 \leq r(v,w) \leq u(v,w) \qquad \forall (v,w) \in E$$
*"Raw flow on an edge cannot exceed its capacity."*

**Definition 3.** *For every directed edge $(v,w)$, the **net flow** $f(v,w)$, or simply flow, is defined as:*

$$f(v,w) = r(v,w) - r(w,v)$$

*Net flow satisfies the properties:*

1. *Conservation/Balance:*
$$\sum_{x \in V} f(v,x) = 0 \qquad \forall v \neq s,t$$
2. *Capacity constraint:*
$$f(v,w) \leq u(v,w) \qquad \forall (v,w) \in E$$
3. *Skew symmetry*
$$f(v,w) = -f(w,v) \qquad \forall (v,w) \in E$$

**Definition 4.** *The **total flow** $|f|$, or flow value, is the net flow out of $s$.*

$$|f| = \sum_{w \in V} f(s,w) = \sum_{y \in V} r(s,y) - \sum_{x \in V} r(x,s)$$

**Lemma 1.** *The total flow is also equal to the net flow into $t$, or:*

$$|f| = \sum_{w \in V} f(s,w) = \sum_{v \in V} f(v,t)$$

*Proof.*

$$\begin{aligned}
\sum_w f(s,w) &= \sum_w r(s,w) - \sum_v r(v,s) \\
&= \left( \sum_{v,w} r(v,w) - \sum_{v \neq s,t} r(v,w) - \sum_w r(t,w) \right) - \left( \sum_{v,w} r(v,w) - \sum_{w \neq s,t} r(v,w) - \sum_v r(v,t) \right) \\
&= \left( \sum_{v,w} r(v,w) - \sum_{v,w} r(v,w) \right) + \left( \sum_{w \neq s,t} r(v,w) - \sum_{v \neq s,t} r(v,w) \right) + \left( \sum_v r(v,t) - \sum_w r(t,w) \right) \\
&= 0 + \sum_{x \neq s,t} (r(v,x) - r(x,w)) + \sum_v f(v,t) \\
&= \sum_v f(v,t)
\end{aligned}$$

□

**Definition 5.** *The **maximum flow problem** (*MAXFLOW*): Given a network $G = (V,E)$ with capacities $u(e)$, find a feasible flow $f$ with maximum value.*

## 2.1 Flow Decomposition and Cuts

As we will prove, any flow can be decomposed into a set of flows along s-t paths and cycles. This will eventually give us a way to upper bound the value of a flow in terms of cuts.

**Lemma 2** (flow decomposition)**.** *Any feasible flow $f$ can be decomposed into at most $m$ cycles and s-t paths with non-zero flows. The value of $f$ is equal to the sum of flows over the paths.*

*Proof.* We prove the lemma by constructing such a decomposition. Given network with a feasible flow $f$:

1. Find $\Gamma$, an $s-t$ path or cycle with positive flow.
2. Let the flow on $\Gamma$ be the be the minimum flow on any edge of $\Gamma$. Reduce the flow on every edge of $\Gamma$ by that quantity. Count $\Gamma$ as one of the decomposed paths/cycles.
3. The previous step's reduction completely removed flow from at least one edge of $\Gamma$. Repeat until all edges of the network have zero flow.

As long as the flow value is non-zero, we should always be able to find a path in (1). On the other hand, when the flow value is zero but some edges have non-zero flow, the balance constraints require a cycle to exist. Each invocation of (2) removed flow completely from at least one edge, so the number of times it can be repeated (and the number of $\Gamma$ we can find) is no more than $m$.

We can change the algorithm slightly to remove only flow on cycles first, and remove paths once the subgraph of flows is acyclic. Each cycle removal cannot change the flow value, so flow value must decrease from $f$ to 0 during the path removals, proving our second statement. □

*Cuts* are another useful entity in graph analysis. In this case, we can use the capacity of cuts to upper bound the flow value.

**Definition 6.** *A **cut** of $G$ is a bipartition of the vertices into $S \subseteq V$ and $\overline{S} = V \setminus S$, or the edges which go between this bipartition. An **s-t cut** is a cut where $s \in S$ and $t \in \overline{S}$.*

We will often refer to cuts as $(S, \overline{S})$, or just $S$. The previous network flow definitions on edges generalize for flow across cuts:

**Definition 7.** *The **net flow** across cut $(S, \overline{S})$ is*

$$f(S) = \sum_{\substack{v \in S \\ w \in \overline{S}}} f(v, w)$$

**Definition 8.** *The **capacity** or value of a cut $(S, \overline{S})$ is*

$$u(S) = \sum_{\substack{v \in S \\ w \in \overline{S}}} u(v, w)$$

Note that we only count edges going in the forward direction (from $S$ to $\overline{S}$) when discussing cut capacity. We do not include the capacity of edges going from $\overline{S}$ to $S$. Clearly, $f(S) \leq u(S)$.

**Lemma 3.** *For any s-t cut $S$, $f(S) = |f|$.*

*Proof.* Flow decomposition allows us to break $f$ into a collection of s-t paths and cycles. Consider the edges of each path/cycle included in $(S, \overline{S})$. Each s-t path crosses the cut an odd number of times and has a net of one edge in the cut. Each cycle crosses an even number of times, and has a net of zero edges. Let $P$ be the set of s-t paths, $C$ be the set of cycles:

$$f(S) = \sum_{e \in S} f(e) = \sum_{\Gamma \in P} (1 \cdot f(\Gamma)) + \sum_{\Gamma \in C} (0 \cdot f(\Gamma))$$

From flow decomposition, we know that:

$$\sum_{\Gamma \in P} f(\Gamma) = |f|$$

□

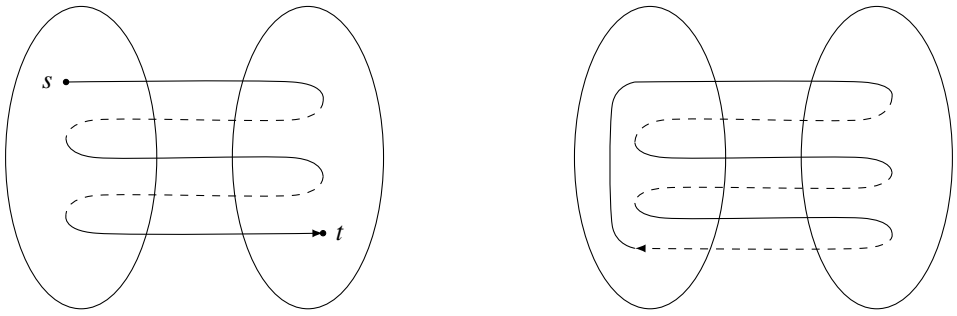


Figure 2: An s-t path has net flow across an s-t cut equal to its flow. A cycle has net flow 0 across any cut.

**Corollary 4.** *The value of the flow is bounded above by the capacity of the minimum s-t cut.*

$$|f| \leq \min_{\text{s-t cut } S} u(S)$$

## 2.2 Max-flow Min-cut Theorem

We will now prove a theorem which gives a strong relationship between the *maximum flow* and the value of the minimum s-t cut. First, we will introduce the notion of the *residual graph* of a flow.

**Definition 9.** *Let $f$ be a feasible flow on $G = (V, E)$. The **residual graph** is the graph $G_f = (V, E_f)$ with capacities $u_f(e)$. If edge $(v, w)$ has (net) flow $f(v, w)$:*

$$u_f(v, w) = u(v, w) - f(v, w)$$

*As usual, we omit edges with 0 residual capacity from $E_f$.*

One can show, using the feasibility constraints, that $u_f(e) \geq 0$.

**Definition 10.** *An **augmenting path** is a directed s-t path in the residual network.*

Let the flow along an augmenting path be the value of the smallest residual capacity of any edge in it. This flow is feasible in $G_f$. We can essentially add this flow as another path in the flow decomposition of $f$, increasing the flow value. More generally, we can prove that adding any feasible flow $f'$ from $G_f$ to $f$ creates a new feasible flow in $G$, i.e. that $f + f'$ is feasible.

*Proof.* We can prove this for each of the flow constraints.

1. For any vertex $v$, conservation constraints still hold in $f + f'$.

   By feasibility, conservation holds for $f$ and $f'$ individually.

$$\left(\sum_x f(v,x) - \sum_x f(x,v)\right) + \left(\sum_x f'(v,x) - \sum_x f'(x,v)\right) = 0 + 0 = 0$$

2. For any edge $(v,w)$, capacity constraints still hold in $f + f'$.

$$\begin{aligned} f'(v,w) &\leq u_f(v,w) \\ &= u(v,w) - f(v,w) \end{aligned}$$

   Then:

$$\begin{aligned} (f+f')(v,w) &= f(v,w) + f'(v,w) \\ &\leq f(v,w) + u(v,w) - f(v,w) \\ &= u(v,w) \end{aligned}$$

3. Similarly, antisymmetry is preserved by $f + f'$ when we add them.

□

**Theorem 5** (max-flow min-cut)**.** *The following are equivalent:*

1. *Flow from s to t (flow value) is maximized.*
2. *The residual network has no augmenting paths of positive flow.*
3. *There exists a cut $(S^*, \overline{S^*})$ where $f(S^*) = u(S^*, \overline{S^*})$.*

*Proof.* We will prove each direction in a cycle.

1. (1) $\implies$ (2)

   If an augmenting path of positive flow exists in $G_f$, we can augment along this path to find a new flow of greater value. Therefore no maximum flow can have an augmenting path in $G_f$.

2. (2) $\implies$ (3)

   Let $S^*$ be the set of all vertices reachable from $s$ in $G_f$. $t \notin S^*$ by the assumption, so both $S^*, \overline{S^*}$ are nonempty. Additionally, there are no residual edges from $S^*$ to $\overline{S^*}$ by our construction of $S^*$. It follows that all edges in $(S^*, \overline{S^*})$ are saturated and there is no flow on any edges in $(\overline{S^*}, S^*)$. Thus $u(S^*, \overline{S^*}) = f(S^*)$.

3. (3) $\implies$ (1)

   The bound in Corollary 4 applies here:

$$|f| \leq u(S^*)$$

   In (3) the bound is tight, so $|f|$ must be maximized.

□

# 3 Ford-Fulkerson

The following algorithm by Ford and Fulkerson [FF56] solves the maximum flow problem.

**Algorithm 1** (Ford-Fulkerson 1956)

1: $f(v,w) \leftarrow 0 \quad \forall (v,w) \in E$
2: **while** $G_f$ has an $s$-$t$ path **do**
3: Find an $s$-$t$ path $\Gamma$ in $G_f$.
4: $\Delta \leftarrow \min_{e \in \Gamma} u_f(v,w)$
5: Push $\Delta$ flow along $\Gamma$.
6: **end while**

## 3.1 Correctness and Running Time

Since the augmentations saturate at least one edge of the path, that path is no longer an $s$-$t$ path in the residual graph. The algorithm converges for integer capacities and runs in $O(m|f|)$ time. Each augmentation increases the value of the flow by at least 1, the residual graph can be computed in $O(m)$ time, and an augmenting path can be found in $O(m)$ time. The running time is linear in $|f|$ and performs better when $|f|$ is small. For example, unit capacities give $|f| \leq n$ and Ford-Fulkerson runs in $O(mn)$ time. In general, runtime could be linear in the unary representation of the input.

Ford-Fulkerson does always converge for rational capacities, but perhaps slower than $O(m|f|)$. For arbitrary real capacities, the algorithm may not converge, or may converge to a non-maximal flow. For the remainder of this discussion we will work with integral capacities.

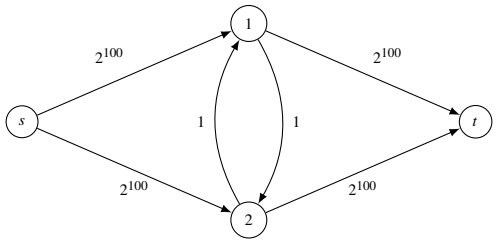


Figure 3: Ford-Fulkerson converges slowly if every augmenting path it chooses includes one of the middle vertical arcs. This will alternate between (s-1-2-t) and (s-2-1-t) and increase flow by at most 2 each augmentation.

## 3.2 Classifications of Polynomial Time

The integral Ford-Fulkerson algorithm runs in time polynomial in $|f|$, as opposed to only $m$ and $n$. We say that Ford-Fulkerson is a *pseudo-polynomial* time algorithm, and we formally distinguish this dependence in algorithm running times:

**Definition 11.** *An algorithm runs in **pseudo-polynomial** time if its runtime is polynomial in the size of the unary representation of the input.*

**Definition 12.** *An algorithm runs in **weakly-polynomial** time if its runtime is polynomial in the size of the binary representation of the input.*

**Definition 13.** *An algorithm runs in **strongly-polynomial** time if its runtime is polynomial in the number of combinatorial parameters and independent of their numerical values. In the case of max flow, these would be m and n.*

It's interesting to note that these are only intersting definitions under the *RAM model* of computation. On a Turing machine, it will take weakly-polynomial time to perform operations we typically think of as constant time, like comparisons and arithmetic.

# 4 Shortest Augmenting Path

A simple adjustment [EK72] to Ford-Fulkerson yields a strongly polynomial algorithm. In each step, instead of an arbitrary augmenting path, we choose one of the shortest (by number of edges).

**Algorithm 2** (Edmonds-Karp 1972)

1: $f(v,w) \leftarrow 0 \quad \forall (v,w) \in E$
2: **while** $G_f$ has an $s$-$t$ path **do**
3: Find the $s$-$t$ path $\Gamma$ in $G_f$ with the fewest number of edges.
4: $\Delta \leftarrow \min_{e \in \Gamma} u_f(v,w)$
5: Push $\Delta$ flow along $\Gamma$.
6: **end while**

## 4.1 Running Time

The idea is that when $s$ and $t$ are far apart in the residual graph, there are not many augmenting paths remaining, since long s-t paths make it more likely for there to be a small s-t cut. Augmenting along any path removes it in the residual graph, so we augment along the shortest paths first. We formalize this notion by analyzing a *distance function* over $G_f$.

**Lemma 6.** *For any $v,w \in V$ let $d(v,w)$ be the shortest path distance from v to w in $G_f$ by number of edges. In the Edmonds-Karp algorithm, $d(s,v)$ and $d(v,t)$ never decrease for any vertex v.*

*Proof.* Suppose, for contradiction, some vertices became closer to $s$ after an augmentation.

Let $x$ be the vertex closest to $s$ after that augmentation, $d(\cdot,\cdot)$ denote distances before that augmentation, and $d'(\cdot,\cdot)$ after augmentation (similarly, $f$ and $f'$). Consider any $y$, predecessor to $x$ on a shortest $s$-$x$ path after augmentation. Since $d(s,y)$ was not decreased (by minimality of $x$) but $x$ became became closer, the residual edge $(y,x)$ must be new in $G_{f'}$. This means that the augmentation sent flow along $(x,y)$ in $G_f$, which in turn means that $d(s,x) < d(s,y)$ in $G_f$ (by our choice of augmenting path). These together give:

$$d'(s,x) < d(s,x) < d(s,y) \leq d'(s,y)$$

But since we chose $y$ to be a predecessor of $x$ in $G_{f'}$:

$$d'(s,x) \geq d'(s,y)$$

And our two inequalities contradict. The proof for $d(v,t)$ is the same. □

**Theorem 7.** *Edmonds-Karp runs in $O(m^2 n)$ time.*

*Proof.* We will bound the number of times each edge can be saturated by an augmenting path. Think of this as labeling every augmentation/iteration with the edge it saturates. In the end we show that each edge labels at most $(n-1)/2$ iterations throughout the algorithm.

Let $(v,w)$ be an edge saturated in one iteration. Before $(v,w)$ can be saturated again, $(w,v)$ must be involved in an augmentation. During the first augmentation on $(v,w)$, $v$ is closer to $s$ than $w$. During the augmentation on $(w,v)$, $w$ is closer to $s$ than $v$. Finally, to augment on $(v,w)$ a second time, $v$ must once again be closer to $s$ than $w$. Lemma 6 says that the distances to $s$ are non-decreasing, so we have that:

$$d_1(s,v) < d_1(s,w) \leq d_2(s,w) < d_2(s,v) \leq d_3(s,v)$$

Distances are integral, so $d(s,v)$ must have increased by at least 2 by the second augmentation of $(v,w)$ (above as $d_3(s,v)$). No distance from $s$ is greater than $(n-1($, so $(v,w)$ labels at most $(n-1)/2$ augmentations. Summing over edges gives a $O(mn)$ bound on the number of iterations. We can perform the work in each iteration in $O(m)$ time (BFS plus augmentation), so the total running time is $O(m^2n)$. □

# 5 Summary

In this lecture we introduced the maximum flow problem and two algorithms based on augmenting paths. We also described the pseudo/weakly/strongly-polynomial classifications. Finally, we proved the flow decomposition lemma and max-flow min-cut theorem.

While Edmonds-Karp gave us a strongly polynomial algorithm, this notion of increasing distance leads to an even faster flow algorithm based on *blocking flows*, which will be discussed next lecture. Instead of repeatedly eliminating short paths from $G_f$, we send a flow which guarantees the distance increases.

# References


[EK72] Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[FF56] Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.