# CSCI 270 Lecture 20: Capacity-Scaling

We will now analyze the runtime of the Capacity-Scaling algorithm.

We'll refer to each iteration as a phase. We need to figure out how many phases there will be, we need to figure out how long it takes to find a path on $G_f(\Delta)$, and we need to figure out how many paths we might find in each phase.

- How many iterations will our algorithm have?
- How long does it take to find a path on $G_f(\Delta)$? Why?

So now we have to cap the number of paths we find in a phase.

Let $f$ be the flow at the end of an iteration with $\Delta$-scaling.

Let $S$ be the set of nodes reachable from $s$ in $G_f(\Delta)$.

Now consider $G_f$, which has additional edges of capacity $< \Delta$.

- Consider an arbitrary edge passing from S to V-S on $G_f$. What is the max amount of flow we could push along this edge?
- How many edges could pass from $S$ to $V - S$?
- How much flow could we conceivably be missing at this point in time?

Remember, we just finished the $\Delta$-scaling phase. So in the next phase, we will consider $G_f(\frac{\Delta}{2})$.

- Suppose we find a new $s - t$ path when considering $G_f(\frac{\Delta}{2})$. What is the minimum amount of flow we will be able to push along this path?
- How much flow did we say we might be missing at this point in time?
- How many paths can we find, maximum, during this phase?

Runtime for Ford-Fulkerson with Capacity-Scaling is $\theta(m^2 \log C)$. Is this a polynomial runtime?

Other network flow algorithms:

- Preflow-Push = $O(n^3)$
- Goldberg-Rao = $O(min(n^{\frac{2}{3}}, m^{\frac{1}{2}})m \log n \log C)$

## Poly-Time Reductions

**Minimum Cut(Graph $G$)**
**1:** Find the max-flow $f$ and the residual graph $G_f$
**2:** Find the set of nodes $A$ reachable from $s$ on $G_f$
**3:** Return $A$

What is the runtime of Minimum Cut?

The runtime of Minimum Cut depends on the runtime of Maximum Flow! If someone finds a better Max-Flow algorithm, they will simultaneously improve the best known Min-Cut algorithm!

Min-Cut is not the only problem like this. There are a **lot** of problems for whom the runtime bottleneck is Max-Flow. This is why the research community has spent so much time trying to improve the existing Max-Flow algorithms.

This is called a **reduction**. In a reduction you take a problem you do not know how to solve, and turn it into a problem you do know how to solve.

In a **poly-time reduction**, the reduction takes no more than polynomial-time. This is also written: $MinimumCut \leq_p NetworkFlow$. Or: Minimum Cut is poly-time reducible to Network Flow.

- If $A \leq_p B$, and $B$ is poly-time, what can we state about the running time for $A$?
- If $A \leq_p B$, and $A$ is poly-time, what can we state about the running time for $B$?

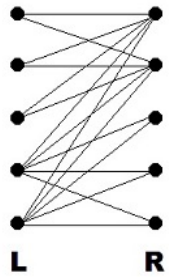
So, by reducing Minimum Cut to Network Flow in polynomial time, we have proven that Minimum Cut is also solvable in polynomial time!

Now that we know Minimum Cut has a polynomial time algorithm, if we can give a poly-time reduction from some problem X to Min-Cut, we can determine that X has a polynomial-time algorithm as well.

Let $P$ be the set of problems with polynomial time solutions. If we want to show a problem is in $P$, we either write a poly-time algorithm for it, or we reduce it to a problem we already know is in $P$.

## Bipartite Matching

We are given an undirected bipartite graph $G = (L \cup R, E)$.




$M \subseteq E$ is a matching if each node appears at most once in M. Find the max-size matching.

- What is the largest-sized matching you can find?
- Why can't there be a larger matching?

A perfect matching includes every node. This instance cannot have a perfect matching.

We could try to come up with our own algorithm to solve this problem, but that sounds like too much work. Instead we're going to be **lazy**, and use an existing algorithm. Namely, we'll use Ford-Fulkerson.

To use Ford-Fulkerson, we have to transform the current graph into a network flow graph. In addition, it must be done in such a way that the solution to the network flow graph helps us find the max-sized matching.

- What components are missing in this graph, which are needed in a Network Flow graph?
- How should we transform our graph into a Network Flow graph?
- How do we extract the answer to Bipartite Matching once we've solved Network Flow?

So, by reducing Bipartite Matching to Network Flow in polynomial time, we have proven that Bipartite Matching is also solvable in polynomial time!

Now that we've reduced Bipartite Matching to Network Flow, Bipartite Matching is a valid problem to reduce to as well. You could reduce some new problem $C \leq_p$ Bipartite Matching to show $C$ is poly-time solvable.

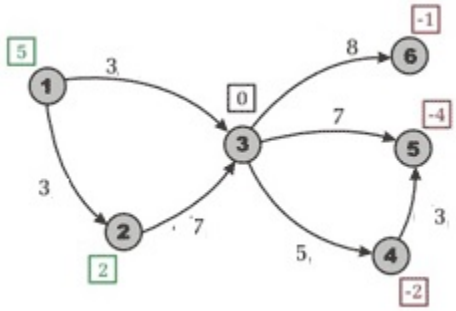
## Circulations

We are given a directed graph $G$, with edge capacities $c(e)$, for all $c \in E$.

Each node has an integer **demand** $d(v)$. If $d(v) < 0$, then we say that this node has a **supply**.

A circulation is a function $f$ which satisfies:

1. $0 \leq f(e) \leq c(e)$, for all $e$
2. $\sum_{(x,v)} f(x,v) - \sum_{(v,y)} f(v,y) = d(v)$




- Is there a circulation that satisfies these constraints?
- What components are missing in this graph, which would be needed in Network Flow?
- What components are in this graph, which must be removed in Network Flow?
- How should we transform our graph into a Network Flow graph?
- How can we extract the answer for Circulations, once we've solved Network Flow?

Extra Problems:

- Chapter 7, exercises 8, 11, 12, 14, 24, 27, 29
- Challenge problems: Chapter 7, exercises 22, 41