Advanced Graph Algorithms and Optimization Spring 2020

Classical Algorithms for Maximum Flow

Rasmus Kyng, Scribe: Meher Chaitanya *Lecture 10 — Wednesday, April 29th*

1 Overview

In this lecture, we will study the *Maximum Flow Problem* and discuss some classical algorithms for finding solutions to it.

2 Maximum Flow

Setup. Consider a directed graph $G = (V, E, \boldsymbol{c})$, where V denotes vertices, E denotes edges and $\boldsymbol{c} \in \mathbb{R}^E, \boldsymbol{c} \geq 0$ denotes edge capacities. In constast to earlier lectures, the direction of each edge will be important, and not just as a booking keeping tool (which is how we previously used it in undirected graphs). We consider an edge $(u, v) \in E$ to be *from* u and *to* v. Edge capacities are associated with directed edges, and we allow both edge (u, v) and (v, u) to exist, and they may have different capacities.

A *flow* is any vector $\boldsymbol{f} \in \mathbb{R}^E$.

We say that a flow is feasible when $\mathbf{0} \leq \boldsymbol{f} \leq \boldsymbol{c}$. The constraint $\mathbf{0} \leq \boldsymbol{f}$ ensures that the flow respects edge directions, while the constraint $\boldsymbol{f} \leq \boldsymbol{c}$ ensures that the flow does not exceed capacity on any edge.

We still define the edge-vertex incidence matrix $\boldsymbol{B} \in \mathbb{R}^{V \times E}$ of G by

$$\boldsymbol{B}(v, e) = \begin{cases} 1 & \text{if } e = (u, v) \\ -1 & \text{if } e = (v, u) \\ 0 & \text{o.w.} \end{cases}$$

As in the undirected case, a *demand vector* is a vector $\boldsymbol{d} \in \mathbb{R}^V$. And as in the undirected case, we can express the net flow constraint that $\boldsymbol{f}$ routes the demand $\boldsymbol{d}$ by

$$\boldsymbol{B}\boldsymbol{f} = \boldsymbol{d}.$$

We will focus on the case:

$$\boldsymbol{B}\boldsymbol{f} = F(-\boldsymbol{e}_s + \boldsymbol{e}_t) = F\boldsymbol{b}_{st}$$

Flows that satisfy the above equation for some scalar F are called s-t flows where $s, t \in V$. The vertex$s$ is called the source, t is called the sink and $\boldsymbol{e}_s$, $\boldsymbol{e}_t$ are indicator vectors for source and sink nodes respectively. The vector $\boldsymbol{b}_{st}$ has -1 at source and 1 at sink. The maximum flow can be

expressed as a linear program as follows

$$\begin{aligned} \max_{\boldsymbol{f}\in\mathbb{R}^E, F} \quad & F \\ \text{s.t.} \quad & \boldsymbol{B}\boldsymbol{f} = F\boldsymbol{b}_{st} \\ & 0 \le \boldsymbol{f} \le \boldsymbol{c} \end{aligned} \tag{1}$$

We use $\text{val}(\boldsymbol{f})$ to denote F when $B\boldsymbol{f} = F\boldsymbol{b}_{st}$.

2.1 Flow decomposition

We now look at a way of simplifying flows

Theorem 2.1. *An s-t path flow is a flow $\boldsymbol{f} \ge \boldsymbol{0}$ that can be written as*

$$\boldsymbol{f} = \alpha \sum_{e\in p} \boldsymbol{e}_e$$

where p is a simple path from s to t.

Theorem 2.2. *An cycle flow is a flow $\boldsymbol{f} \ge \boldsymbol{0}$ that can be written as a cycle i.e.*

$$\boldsymbol{f} = \alpha \sum_{e\in c} \boldsymbol{e}_e$$

where c is a simple cycle.

Lemma 2.3 (The path-cycle decomposition lemma)**.** *Any s-t flow $\boldsymbol{f}$ can be decomposed into a sum of s-t path flows and cycle flows such that the sum contains at most* $\text{nnz}(\boldsymbol{f})$ *terms. Note* $\text{nnz}(\boldsymbol{f}) \le |E|$.

Proof. We perform induction on the number of edges with non-zero flow, which we denote by $\text{nnz}(\boldsymbol{f})$. Note that by "the support of $\boldsymbol{f}$", we mean the set $\{e \in E : \boldsymbol{f}(e) \neq 0\}$.

Base case: $\boldsymbol{f} = 0$: nothing to do.

Inductive step: Try to find a path from s to t OR a cycle in the support of $\boldsymbol{f}$.
"Path" case. If there exists such a an s-t path, let α be the minimum flow value along the edges of the path. i.e.

$$\alpha = \min_{(a,b)\in p} \boldsymbol{f}(a,b)$$

$$\boldsymbol{f}' = \alpha \sum_{e\in p} \boldsymbol{e}_e$$

Update the flow $\boldsymbol{f}$ by

$$\boldsymbol{f} \leftarrow \boldsymbol{f} - \boldsymbol{f}'$$

The value of the flow will still be non-negative after this update as we subtracted the minimum entry along any positive edge on the path. The number of non-zeros, $\text{nnz}(\boldsymbol{f})$, went down by at least one. Note that the updated $\boldsymbol{f}$ must again be an s-t, as it is the difference of two s-t flows.

"Cycle" case. Suppose we find a c cycle in the support of $\boldsymbol{f}$. Let α be the minimum flow value along the edges of the cycle. i.e.

$$\alpha = \min_{(a,b)\in c} \boldsymbol{f}(a,b)$$

$$\boldsymbol{f}' = \alpha \sum_{e\in c} \boldsymbol{e}_e$$

Update the flow $\boldsymbol{f}$ by

$$\boldsymbol{f} \leftarrow \boldsymbol{f} - \boldsymbol{f}'$$

As in the path case, $\boldsymbol{f}$ stays non-negative, and number of non-zeros, $\text{nnz}(\boldsymbol{f})$, goes down by at least one. Note that the updated $\boldsymbol{f}$ must again be an s-t, as it is the difference of two s-t flows.

"No path or cycle" case. Suppose we can find neither a path nor a cycle, and $\boldsymbol{f} \neq \mathbf{0}$. Then there must be an edge (u, v) with non-zero flow leading into a vertex $v \neq s, t$ and with no outgoing edge from v in the support of $\boldsymbol{f}$. In that case, we must have $(\boldsymbol{B}\boldsymbol{f})(v) \geq 0$. But since $v \neq t$, this contradicts $\boldsymbol{B}\boldsymbol{f} = F\boldsymbol{b}_{st}$. So this case cannot occur. □

Lemma 2.4. *In any s-t max flow problem instance, there is an optimal flow $\boldsymbol{f}^*$ with a path-cycle decomposition that has only paths and no cycles.*

Proof. Let $\tilde{\boldsymbol{f}}$ be an optimal flow. Let $\boldsymbol{f}^*$ be the sum of path flows in the path cycle decomposition of $\tilde{\boldsymbol{f}}$. They route the same flow (as cycles contribute to no net flow from s to t). Thus

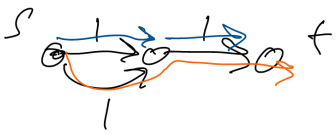
$$\boldsymbol{B}\boldsymbol{f}^* = \boldsymbol{B}\tilde{\boldsymbol{f}}$$

and hence $\text{val}(\boldsymbol{f}^*) = \text{val}(\tilde{\boldsymbol{f}})$. Furthermore

$$\mathbf{0} \leq \boldsymbol{f}^* \leq \tilde{\boldsymbol{f}} \leq \boldsymbol{c}$$

The first inequality follows from $\boldsymbol{f}^*$ being a sum of positive path flow. The second inequality holds as $\boldsymbol{f}^*$ is upper bounded in every single entry by $\tilde{\boldsymbol{f}}$, because we reduced it by positive entry cycles. The third inequality holds because $\tilde{\boldsymbol{f}}$ is a feasible flow, so it is upper bounded by the capacities. □

An optimal flow solving the maximum flow problem may not be unique. For example, consider the graph below with source s and sink t:



There are two optimal paths in this example. Maximum flow is a convex optimization problem but not a strongly convex problem as the solutions are not unique, and this is part of what makes it hard to solve.

2.2 Cuts and Minimum Cuts

The decomposition shown earlier provides a way to show that the maximum flow in a graph is upper bounded by constructing graph cuts.

Given a vertex subset $S \subseteq V$, we say that $(S, V \setminus S)$ is a *cut* in G and that the value of the cut is

$$c_G(S) = \sum_{e \in E \cap (S \times V \setminus S)} \boldsymbol{w}(e).$$

Note that in a directed graph, only edges crossing the cut going *from* S and *to* $V \setminus S$ count toward the cut.

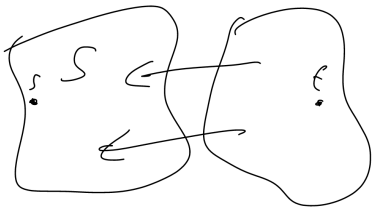


Figure 1: Example of a cut: No edges go from S to $(S, V \setminus S)$, and so the value of this cut is zero.

Definition. (s-t cuts). We define an s-t cut to be a subset $S \subset V$, where $s \in S$ and $t \in V \setminus S$.

A decision problem: *"Given an instance of the Maximum Flow problem, is there a flow from s to t such that $\boldsymbol{f} \neq \mathbf{0}$?"*

If YES: We can decompose this flow into s-t path flows and cycle flows.

If NO: There is no flow path from s to t. Let S be the set of vertices reachable from source s. Then $(S, V \setminus S)$ is a cut in the graph, with no edges crossing from S to $V \setminus S$. Figure 1 gives an example.

Upper bounding the maximum possible flow value. How can we recognize a maximum flow? Is there a way to confirm that a flow is maximum value?

We can now introduce the *Minimum Cut* problem.

$$\min_{S \subseteq V} \quad c_G(S) \tag{2}$$
$$\text{s.t. } s \in V \text{ and } t \notin V$$

The Minimum Cut problem can also be phrased as a linear program, although we won't see that today.

We'd like to obtain a tight connection between flows and cuts in the graph. As a first step, we won't get that, but we can at least observe that the value of any s-t cut provides an upper bound to the maximum possible s-t flow value.

Theorem 2.5 (Max Flow $\leq$ Min Cut)**.** *The maximum s-t flow value in a directed graph G (Program* (1)*) is upper bounded by the minimum value of any s-t cut (Program* (2)*. I.e. if S is an s-t cut, and $\boldsymbol{f}$ a feasible s-t flow then*

$$\text{val}(\boldsymbol{f}) \leq c_G(S)$$

And in particular, this holds for any minimum cut S^* and maximum flow $\boldsymbol{f}^*$. I.e. $\text{val}(\boldsymbol{f}^*) \leq c_G(S^*)$.

Proof. Consider any feasible flow $0 \leq \boldsymbol{f} \leq \boldsymbol{c}$ and a cut S, T such that $S \cup T = V$. Consider a path-cycle decomposition of $\boldsymbol{f}$, where each s-t path must cross the cut going forward from S to T at least once. We pick a cut S, T with source on one side and sink on the other side as shown in the Figure (2). Every time the path flow passes through the cut, it has to use one of the edges that connect S and T. Total amount of flow crossing the cut is bounded above by total amount of capacity of the cut, otherwise the capacities would be violated, thus $\text{val}(\boldsymbol{f}) \leq c_G(S, T) = \sum_{e \in E \cap S \times T} \boldsymbol{c}(e)$.

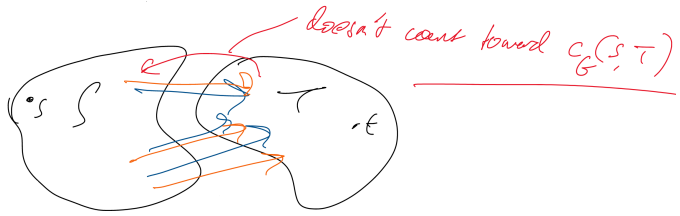


Figure 2: s-t Cut

Note that the edges from T to S do not count towards the cut. The above equation holds for all flows with all s-t cuts. This implies that **Max flow** $\leq$ **Min cut**. □

This theorem is an instance of a general pattern, known as *weak duality*. Weak duality is a relationship between two optimization programs, a maximization problem and a mimization problem, where any solution to the former has its value upper bounded by the value of any solution to the latter.

2.3 Algorithms for Max flow

How can we find a good flow?

Algorithm 1: A first attempt - bad idea?

$\boldsymbol{f} \leftarrow \mathbf{0}$;
repeat
  Find an s-t path flow $\tilde{\boldsymbol{f}}$ that is feasible with respect to $\boldsymbol{c} - \boldsymbol{f}$.
  $\boldsymbol{f} \leftarrow \boldsymbol{f} - \tilde{\boldsymbol{f}}$

Does Algorithm 1 work?
Consider the graph below with directed edges with capacities 1 at every edge If we make a single path update as shown by the orange lines in Figure 3, then afterwards, using the remaining capacity, there's no path flow we can route, as shown in Figure 3. But the max flow is 2, as shown by the flow on orange edges in Figure 5. So, the above algorithm does not always find a maximum flow: It can get stuck at a point where it cannot make progress despite not having reached the maximum value.

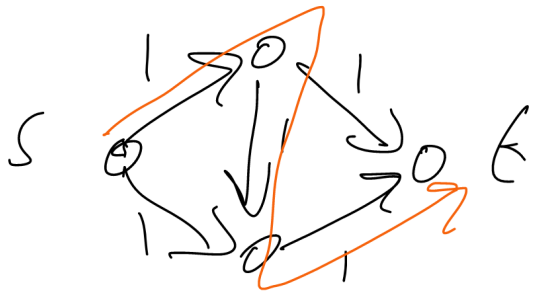


Figure 3: Sending a unit s-t path flow through the graph.

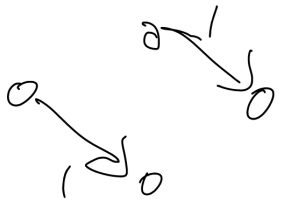


Figure 4: Remaining edge capacities after sending a path flow through the graph as depicted in Figure 3.

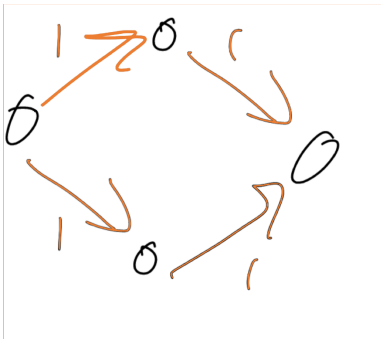


Figure 5: The flow depicted routes two units from s to t.

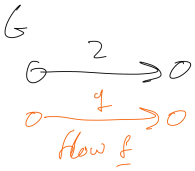
A better approach. It turns out we can fix the problem with the previous algorithm using a simple fix. This idea is known as *residual graphs*.

Algorithm 2: Better Idea (Residual Graph)

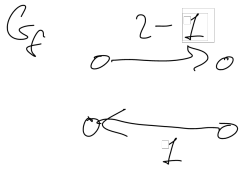
$\boldsymbol{f} \leftarrow \mathbf{0}$;
repeat
  Find an s-t path flow $\tilde{\boldsymbol{f}}$ that is feasible with respect to $-\boldsymbol{f} \leq \tilde{\boldsymbol{f}} \leq \boldsymbol{c} - \boldsymbol{f}$.
  $\boldsymbol{f} \leftarrow \boldsymbol{f} - \tilde{\boldsymbol{f}}$

The $-\boldsymbol{f}$ can be treated as edge going in the other direction with capacity $\boldsymbol{f}(e)$. By convention, an edge in G with $\boldsymbol{f}(e) = \boldsymbol{c}(e)$ is called *saturated*, and we do not include the edge in G_f. The graph defined above with such capacities is called **the residual graph of $\boldsymbol{f}$**, $G_{\boldsymbol{f}}$. $G_{\boldsymbol{f}}$ is only defined for feasible $\boldsymbol{f}$, since otherwise the constraint $\tilde{\boldsymbol{f}} \leq \boldsymbol{c} - \boldsymbol{f}$ gives trouble.

Suppose we start with a graph having a single edge with capacity 2 and we introduce a flow of 1 unit.



The residual graph $G_{\boldsymbol{f}}$ has an edge with capacity 1 going forward and -1 capacity going forward, but we can treat the latter as $+1$ capacity going backwards. So it is an edge that allows you to undo the choice made to send flow along that direction.



Let us consider the same example with Residual Graph. The original graph is shown in figure 6

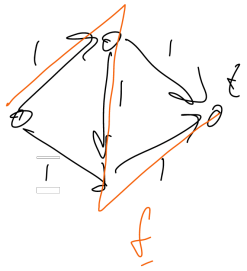


Figure 6: Original graph G and an s-t path flow in G shown in orange.

The residual graph for the same is shown in Figure 7 :

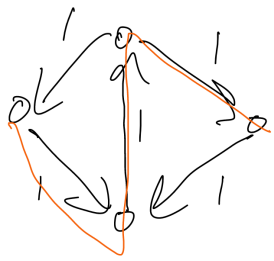


Figure 7: The residual graph w.r.t the flow from Figure 6, and new s-t path flow which is feasible in the residual graph.

Adding both the flows together, we get the paths as shown in Figure 8 with value 2, which is the optimum:

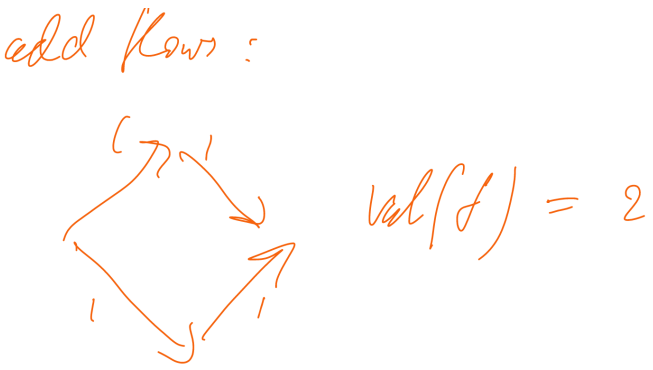


Figure 8: Max flow in the graph

Let's prove some important properties of residual graphs.

Lemma 2.6. *Suppose $\boldsymbol{f}$, $\hat{\boldsymbol{f}}$ are feasible in G. Then this implies that $\hat{\boldsymbol{f}} - \boldsymbol{f}$ (where negative entries count as flow in opposite direction) is feasible in $G_{\boldsymbol{f}}$.*

Proof. $0 \leq \boldsymbol{f} \leq \boldsymbol{c}$ and $0 \leq \tilde{\boldsymbol{f}} \leq \boldsymbol{c}$, this implies $\boldsymbol{f} \leq \tilde{\boldsymbol{f}} - \boldsymbol{f} \leq \boldsymbol{c} - \boldsymbol{f}$. Hence, proved. □

Lemma 2.7. *Suppose that $\boldsymbol{f}$ is feasible in G and $\hat{\boldsymbol{f}}$ is feasible in $G_{\boldsymbol{f}}$. Then, $\boldsymbol{f} + \hat{\boldsymbol{f}}$ is feasible in G.*

Proof. $0 \leq \boldsymbol{f} \leq \boldsymbol{c}$ and $\boldsymbol{f} \leq \tilde{\boldsymbol{f}} \leq \boldsymbol{c} - \boldsymbol{f}$, this implies $0 \leq \tilde{\boldsymbol{f}} + \boldsymbol{f} \leq \boldsymbol{c}$ □

Lemma 2.8. *A feasible $\boldsymbol{f}$ is optimal if and only if t is not reachable from s in $G_{\boldsymbol{f}}$.*

Proof. Let $\boldsymbol{f}$ be optimal, and suppose t is reachable from s in $G_{\boldsymbol{f}}$ then, we can find an s-t path flow $\tilde{\boldsymbol{f}}$ that is feasible in $G_{\boldsymbol{f}}$, and $\text{val}(\boldsymbol{f} + \tilde{\boldsymbol{f}}) > \text{val}(\boldsymbol{f})$. $\boldsymbol{f} + \tilde{\boldsymbol{f}}$ is feasible in G by Lemma 2.7. This is a contradiction, as we assumed $\boldsymbol{f}$ was supposed to be optimal.

Suppose t is not reachable from s in $G_{\boldsymbol{f}}$, and $\boldsymbol{f}$ is feasible, but not optimal. Let $\boldsymbol{f}^*$ be optimal, then by Lemma 2.6, the flow $\boldsymbol{f}^* - \boldsymbol{f}$ is feasible in $G_{\boldsymbol{f}}$ and $\text{val}(\boldsymbol{f}^* - \boldsymbol{f}) > 0$. So there exists an s-t path flow from s to t in $G_{\boldsymbol{f}}$ (as we can do a path decomposition of $\boldsymbol{f}^* - \boldsymbol{f}$). But, this is a contradiction as t is not reachable from s in $G_{\boldsymbol{f}}$. □

Theorem 2.9 (Max Flow = Min Cut theorem)**.** *The maximum flow in a directed graph G equals the minimum cut.*

Proof. Consider the set $S = \{\text{vertices reachable from } s \text{ in } G_{\boldsymbol{f}}\}$. Note that $\boldsymbol{f}$ saturates the edge capacities in cut $S, V \setminus S$ in G: Consider any edge from S to T in G. Since this edge does not exist in the residual graph, we must have $\boldsymbol{f}(e) = \boldsymbol{c}(e)$.

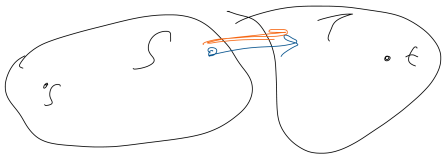


Figure 9: The cut between vertices reachable from S and everything else in G_f must have all outgoing edges saturated by $\boldsymbol{f}$.

This means that

$$\text{val}(\boldsymbol{f}) \geq c_G(S, V \setminus S)$$

Since we already know the opposite inequality by weak duality, we have shown that

$$\text{val}(\boldsymbol{f}) = c_G(S, V \setminus S)$$

This proves the **Max Flow = Min Cut** theorem, which is also called strong duality. □

Ford-Fulkerson Algorithm:

Algorithm 3: Ford-Fulkerson Algorithm

repeat
| Add update by arbitrary s-t path flow in $G_{\boldsymbol{f}}$ (augment the flow $\boldsymbol{f}$ by the path flow)

2.3.1 Convergence properties and Analysis of runtime of Ford-Fulkerson algorithm

- **Does this algorithm terminate?**
  The algorithm terminates if the capacities are integers. However for irrational capacities the algorithm may not terminate.
- **Does it converge towards the max flow value?**
  No, it does not converge to max flow value if the updates are poor and the capacities are irrational.

Lemma 2.10. *Consider Ford-Fulkerson algorithm with integer capacities. The algorithm terminates in* $\text{val}(\boldsymbol{f}^*)$ *augmentations i.e.* $O(m\,\text{val}(\boldsymbol{f}^*))$ *time.*

Proof. Each iteration increases the flow by at least one unit as the capacities in $G_{\boldsymbol{f}}$ are integral and each iteration can be computed in $O(m)$ time. □

Can we do better than this? Suppose we pick the maximum bottleneck capacity (minimum capacity along path) augmenting path. This gives an algorithm that is better in some regimes.

How to pick the maximum bottleneck capacity augmenting path? We are going to perform a binary search on the capacities in $G_{\boldsymbol{f}}$, to find a path with maximum bottleneck capacity. Each time our binary search has picked a threshold capacity, we then try to find an s-t path flow in $G_{\boldsymbol{f}}$ using only edges with capacity above that threshold. If we find a path, the binary search next tries a higher capacity. If don't find a path, the binary search next tries a lower capacity.

Using this approach, the time to find a single path is $O(m\log(n))$ where m is number of edges in the graph. This path must carry at least a $\frac{1}{m}$ fraction of the total amount of flow left in $G_{\boldsymbol{f}}$. For instance, if $\hat{F}$ is the amount of flow left in $G_{\boldsymbol{f}}$, then the path must carry $\frac{\hat{F}}{m}$ flow (from the path decomposition lemma, there are at most m paths, and the one carrying the most flow must carry at least the average amount).

So if the flow is integral, the algorithms completes when

$$\left(1-\frac{1}{m}\right)^{T}\text{val}(\boldsymbol{f}^*)<1$$

where T is number of augmentations.

This means

$$T=m\log F$$

$$\begin{aligned}\text{Total time} &= O(m\log nT)\\ &= O(m^2\log n\log F)\\ &\leq O(m^2\log n\log mU)\text{ as } F\leq mU \text{ where U is the maximum capacity}\end{aligned}$$

Current state of art approaches for Max Flow:

- Strongly polynomial time algorithm:

  - has to work with real valued capacities, then the best time $O(mn)$ by Orlin.

- Restricted to unit capacities
  - Old algorithm that runs in $O(m^{1.5})$.
  - The time for this was improved in 2014 to $O(m^{\frac{10}{7}})$ by Madry.
  - Run time improved again in 2020 to $\tilde{O}(m^{\frac{4}{3}})$ by Liu and Sidford.
- General integer capacities: $O(m \min(\sqrt{m}, n^{2/3}) \log(n) \log(U))$ where U is the maximum capacity. This result is by Goldberg and Rao.