11 Augmenting Path Algorithms

Greedy-algorithm:
- start with \( f(e) = 0 \) everywhere
- find an \( s\)-\( t \) path with \( f(e) < c(e) \) on every edge
- augment flow along the path
- repeat as long as possible

The Residual Graph

From the graph \( G = (V, E, c) \) and the current flow \( f \) we construct an auxiliary graph \( G_f = (V, E_f, c_f) \) (the residual graph):
- Suppose the original graph has edges \( e_1 = (u, v) \), and \( e_2 = (v, u) \) between \( u \) and \( v \).
- \( G_f \) has edge \( e_1' \) with capacity \( \max\{0, c(e_1) - f(e_1) + f(e_2)\} \) and \( e_2' \) with capacity \( \max\{0, c(e_2) - f(e_2) + f(e_1)\} \).

Augmenting Path Algorithm

Definition 1
An augmenting path with respect to flow \( f \), is a path from \( s \) to \( t \) in the auxiliary graph \( G_f \) that contains only edges with non-zero capacity.

Algorithm 1 FordFulkerson(\( G = (V, E, c) \))
1: Initialize \( f(e) \leftarrow 0 \) for all edges.
2: while \( \exists \) augmenting path \( p \) in \( G_f \) do
3: augment as much flow along \( p \) as possible.
Augmenting Path Algorithm

Theorem 2
A flow \( f \) is a maximum flow iff there are no augmenting paths.

Theorem 3
The value of a maximum flow is equal to the value of a minimum cut.

Proof.
Let \( f \) be a flow. The following are equivalent:
1. There exists a cut \( A \) such that \( \text{val}(f) = \text{cap}(A,V \setminus A) \).
2. Flow \( f \) is a maximum flow.
3. There is no augmenting path w.r.t. \( f \).

11.1 The Generic Augmenting Path Algorithm 11. Apr. 2018
Ernst Mayr, Harald Räcke

Analysis

Assumption:
All capacities are integers between 1 and \( C \).

Invariant:
Every flow value \( f(e) \) and every residual capacity \( c_f(e) \) remains integral throughout the algorithm.

Augmenting Path Algorithm

\[
\text{val}(f) = \sum_{e \in \text{out}(A)} f(e) - \sum_{e \in \text{into}(A)} f(e) = \sum_{e \in \text{out}(A)} c(e) = \text{cap}(A,V \setminus A)
\]

This finishes the proof.

Here the first equality uses the flow value lemma, and the second exploits the fact that the flow along incoming edges must be 0 as the residual graph does not have edges leaving \( A \).
Lemma 4
The algorithm terminates in at most \( \text{val}(f^*) \leq nC \) iterations, where \( f^* \) denotes the maximum flow. Each iteration can be implemented in time \( O(m) \). This gives a total running time of \( O(nmC) \).

Theorem 5
If all capacities are integers, then there exists a maximum flow for which every flow value \( f(e) \) is integral.

A Bad Input
Problem: The running time may not be polynomial.

Question:
Can we tweak the algorithm so that the running time is polynomial in the input length?

A Pathological Input
Let \( r = \frac{1}{2} (\sqrt{5} - 1) \). Then \( r^{n+2} = r^n - r^{n+1} \).

Running time may be infinite!!!
How to choose augmenting paths?
▶ We need to find paths efficiently.
▶ We want to guarantee a small number of iterations.

Several possibilities:
▶ Choose path with maximum bottleneck capacity.
▶ Choose path with sufficiently large bottleneck capacity.
▶ Choose the shortest augmenting path.

Overview: Shortest Augmenting Paths

These two lemmas give the following theorem:

**Theorem 8**
The shortest augmenting path algorithm performs at most \( O(mn) \) augmentations. This gives a running time of \( O(m^2n) \).

**Proof.**
▶ We can find the shortest augmenting paths in time \( O(m) \) via BFS.
▶ \( O(m) \) augmentations for paths of exactly \( k < n \) edges.

Overview: Shortest Augmenting Paths

**Lemma 6**
The length of the shortest augmenting path never decreases.

**Lemma 7**
After at most \( O(m) \) augmentations, the length of the shortest augmenting path strictly increases.

Shortest Augmenting Paths

Define the level \( \ell(v) \) of a node as the length of the shortest \( s-v \) path in \( G_f \).

Let \( L_G \) denote the subgraph of the residual graph \( G_f \) that contains only those edges \((u,v)\) with \( \ell(v) = \ell(u) + 1 \).

A path \( P \) is a shortest \( s-u \) path in \( G_f \) if it is a an \( s-u \) path in \( L_G \).
In the following we assume that the residual graph $G_f$ does not contain zero capacity edges.

This means, we construct it in the usual sense and then delete edges of zero capacity.

---

**Shortest Augmenting Path**

**First Lemma:**
The length of the shortest augmenting path never decreases.

After an augmentation $G_f$ changes as follows:
- Bottleneck edges on the chosen path are deleted.
- Back edges are added to all edges that don’t have back edges so far.

These changes cannot decrease the distance between $s$ and $t$.

**Second Lemma:** After at most $m$ augmentations the length of the shortest augmenting path strictly increases.

Let $E_L$ denote the set of edges in graph $L_G$ at the beginning of a round when the distance between $s$ and $t$ is $k$.

An $s$-$t$ path in $G_f$ that uses edges not in $E_L$ has length larger than $k$, even when considering edges added to $G_f$ during the round.

In each augmentation one edge is deleted from $E_L$.

**Theorem 9**
The shortest augmenting path algorithm performs at most $O(mn)$ augmentations. Each augmentation can be performed in time $O(m)$.

**Theorem 10 (without proof)**
There exist networks with $m = \Theta(n^2)$ that require $O(mn)$ augmentations, when we restrict ourselves to only augment along shortest augmenting paths.

**Note:**
There always exists a set of $m$ augmentations that gives a maximum flow (why?).
Shortest Augmenting Paths

When sticking to shortest augmenting paths we cannot improve (asymptotically) on the number of augmentations.

However, we can improve the running time to $O(mn^2)$ by improving the running time for finding an augmenting path (currently we assume $O(m)$ per augmentation for this).

Suppose that the initial distance between $s$ and $t$ in $G_f$ is $k$.

$E_L$ is initialized as the level graph $L_G$.

Perform a DFS search to find a path from $s$ to $t$ using edges from $E_L$.

Either you find $t$ after at most $n$ steps, or you end at a node $v$ that does not have any outgoing edges.

You can delete incoming edges of $v$ from $E_L$.

Let a phase of the algorithm be defined by the time between two augmentations during which the distance between $s$ and $t$ strictly increases.

Initializing $E_L$ for the phase takes time $O(m)$.

The total cost for searching for augmenting paths during a phase is at most $O(mn)$, since every search (successful i.e., reaching $t$) or unsuccessful) decreases the number of edges in $E_L$ and takes time $O(n)$.

The total cost for performing an augmentation during a phase is only $O(n)$. For every edge in the augmenting path one has to update the residual graph $G_f$ and has to check whether the edge is still in $E_L$ for the next search.

There are at most $n$ phases. Hence, total cost is $O(mn^2)$. 
How to choose augmenting paths?

- We need to find paths efficiently.
- We want to guarantee a small number of iterations.

Several possibilities:

- Choose path with maximum bottleneck capacity.
- Choose path with sufficiently large bottleneck capacity.
- Choose the shortest augmenting path.

**Capacity Scaling**

**Intuition:**
- Choosing a path with the highest bottleneck increases the flow as much as possible in a single step.
- Don’t worry about finding the exact bottleneck.
- Maintain scaling parameter $\Delta$.
- $G_f(\Delta)$ is a sub-graph of the residual graph $G_f$ that contains only edges with capacity at least $\Delta$.

**Algorithm 2 maxflow($G, s, t, c$)**

1. foreach $e \in E$ do $f_e \leftarrow 0$
2. $\Delta \leftarrow 2^{\lceil \log_2 C \rceil}$
3. while $\Delta \geq 1$ do
4.   $G_f(\Delta) \leftarrow \Delta$-residual graph
5.   while there is augmenting path $P$ in $G_f(\Delta)$ do
6.      $f \leftarrow$ augment($f, c, P$)
7.      update($G_f(\Delta)$)
8.   $\Delta \leftarrow \Delta / 2$
9. return $f$

**Assumption:**
All capacities are integers between 1 and $C$.

**Invariant:**
All flows and capacities are/remain integral throughout the algorithm.

**Correctness:**
The algorithm computes a maxflow:
- because of integrality we have $G_f(1) = G_f$
- therefore after the last phase there are no augmenting paths anymore
- this means we have a maximum flow.
Lemma 11
There are $\lceil \log C \rceil + 1$ iterations over $\Delta$.
Proof: obvious.

Lemma 12
Let $f$ be the flow at the end of a $\Delta$-phase. Then the maximum flow is smaller than $\text{val}(f) + m\Delta$.
Proof: less obvious, but simple:
- There must exist an $s$-$t$ cut in $G_f(\Delta)$ of zero capacity.
- In $G_f$ this cut can have capacity at most $m\Delta$.
- This gives me an upper bound on the flow that I can still add.

Theorem 14
We need $O(m \log C)$ augmentations. The algorithm can be implemented in time $O(m^2 \log C)$.