How to choose augmenting paths?

- We need to find paths efficiently.
How to choose augmenting paths?

- We need to find paths efficiently.
- We want to guarantee a small number of iterations.
How to choose augmenting paths?

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

Several possibilities:
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 $\Delta$. 

Ernst Mayr, Harald Räcke
Capacity Scaling

Intuition:

▶ Choosing a path with the highest bottleneck increases the flow as much as possible in a single step.
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.
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 $\Delta$. 
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 $\Delta$. 
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 $\Delta$.

![Graphs](https://via.placeholder.com/150)
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: \hspace{1em} $G_f(\Delta) \leftarrow \Delta$-residual graph
5: \hspace{1em} while there is augmenting path $P$ in $G_f(\Delta)$ do
6: \hspace{2em} $f \leftarrow$ augment$(f, c, P)$
7: \hspace{2em} update$(G_f(\Delta))$
8: \hspace{1em} $\Delta \leftarrow \Delta/2$
9: return $f$
Capacity Scaling

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
▶ therefore after the last phase there are no augmenting paths anymore
▶ this means we have a maximum flow.
Capacity Scaling

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

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

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

Correctness:
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.
Capacity Scaling

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 $G_f(1) = G_f$
Capacity Scaling

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
Capacity Scaling

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.
Capacity Scaling

Lemma 1
There are ⌈log C⌉ iterations over ∆.
Proof: obvious.

Lemma 2
Let f be the flow at the end of a ∆-phase. Then the maximum flow is smaller than val(f) + m ∆.
Proof: less obvious, but simple:
▶ There must exist an s-t cut in G f(∆) of zero capacity.
▶ In G f this cut can have capacity at most m ∆.
▶ This gives me an upper bound on the flow that I can still add.
Capacity Scaling

Lemma 1

There are \([\log C]\) iterations over \(\Delta\).

Proof: obvious.

Lemma 2

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.
Capacity Scaling

Lemma 1

There are \( \lceil \log C \rceil \) iterations over \( \Delta \).

Proof: obvious.

Lemma 2

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:
Lemma 1

There are \(\lceil \log C \rceil\) iterations over \(\Delta\).

Proof: obvious.

Lemma 2

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.
Lemma 1

There are $\lceil \log C \rceil$ iterations over $\Delta$.

Proof: obvious.

Lemma 2

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 $m\Delta$. 
Capacity Scaling

Lemma 1
There are $\lceil \log C \rceil$ iterations over $\Delta$.

Proof: obvious.

Lemma 2
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.
Capacity Scaling

Lemma 3
There are at most $2^m$ augmentations per scaling-phase.

Proof:
Let $f$ be the flow at the end of the previous phase.

$\text{val}(f^+)$ $\leq$ $\text{val}(f) + 2^m \Delta$

Each augmentation increases flow by $\Delta$.

Theorem 4
We need $O(m \log C)$ augmentations. The algorithm can be implemented in time $O(m^2 \log C)$. 

11.3 Capacity Scaling
Ernst Mayr, Harald Räcke
Lemma 3

There are at most $2m$ augmentations per scaling-phase.
Lemma 3

There are at most $2m$ augmentations per scaling-phase.

Proof:

- Let $f$ be the flow at the end of the previous phase.
Lemma 3

There are at most $2m$ augmentations per scaling-phase.

Proof:

- Let $f$ be the flow at the end of the previous phase.
- $\text{val}(f^*) \leq \text{val}(f) + 2m\Delta$
Lemma 3

There are at most $2m$ augmentations per scaling-phase.

Proof:

- Let $f$ be the flow at the end of the previous phase.
- $\text{val}(f^*) \leq \text{val}(f) + 2m\Delta$
- Each augmentation increases flow by $\Delta$. 
Lemma 3
There are at most $2m$ augmentations per scaling-phase.

Proof:
- Let $f$ be the flow at the end of the previous phase.
- $\text{val}(f^*) \leq \text{val}(f) + 2m\Delta$
- Each augmentation increases flow by $\Delta$.

Theorem 4
We need $\Theta(m \log C)$ augmentations. The algorithm can be implemented in time $\Theta(m^2 \log C)$. 