Contents Lecture 7

- The maximum flow problem
- The Ford-Fulkerson algorithm
- Maximum flows and minimum cuts
- The preflow-push maximum flow algorithm
A flow network

- A directed graph $G(V, E)$
- Each edge $e \in E$ has a nonnegative capacity $c(e)$
- A source node $s \in V$ with no predecessor
- A sink node $t \in V$ with no successor
- An example:

![Flow Network Diagram](image)

- In Lab 6 you will act as an CCCP party member and solve a problem for railway transportations passing Minsk, using capacities estimated by US spies
Terminology

- An *st-cut* is a partition \((A, B)\) with \(s \in A\) and \(t \in B\). Also called simply a *cut*
- The **capacity** of a cut is
  \[
  \text{cap}(A, B) = \sum_{e \text{ out from } A} c(e)
  \]
- For the previous graph, \(\text{cap}({s}, V - \{s\}) = 3 + 8\)
- The **min-cut problem** is to find a cut of minimum capacity
- Useful information when bombing enemy railroads for instance
A flow

- A flow is a function $f$ which says how much of the capacity of each edge is used.
- Often we want to use the edges to maximize the total flow from $s$ to $t$.
- This can be used to reduce the traffic jams with self-driving cars (if the cars somehow collaborate).
- The algorithm design techniques we have studied so far are insufficient to solve this problem!
- The capacity constraint says: for each $e \in E$, $0 \leq f(e) \leq c(e)$.
- We cannot press more cars across the Öresund bridge to Copenhagen than the road’s capacity (vehicles per unit time).
- And it is not permitted to use the Denmark to Sweden lanes in reverse gear to increase the capacity towards Denmark.
Flow conservation constraint

- The flow coming in to a vertex $v$ must equal the flow going out from $v$
- This **flow conservation constraint** does not apply to the source $s$ and the sink $t$

$$v \in V - \{s, t\} : \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out from } v} f(e)$$

- Too many cars going in to a parking area leads to certain problems. They must leave as well
- Or, the same number of cars going in to a round-about must also leave it (the round-about is a node)
The maximum flow problem

- The value of a flow $f$ is $\sum_{e \text{ out from } s} f(e)$
- The **maximum flow problem** is to find a flow $f$ with maximum value
The basic idea is very simple

1. Start with a flow $f(e) = 0$ for every $e \in E$
2. Look for a simple path $p$ from $s$ to $t$ such that on every edge $(u, v)$ in $p$ we can increase the flow in the direction from $u$ to $v$
3. If we could not find any such path, we already have the maximum flow and we take vacation (or at least print the flow and go home)
4. Let each edge $e = (u, v)$ on $p$ have a value $\delta(e)$, which means room for improvement, or how much we can increase the flow on that edge
5. Let $\Delta$ be the minimum of all $\delta(e)$ on $p$
6. Increase the flow by $\Delta$ along the path $p$
7. goto 2
What does it mean that $\delta(u, v) > 0$?

Answer: $f(u, v) < c(u, v)$

It is clear that if we find such a path $p$ we can increase the flow on each edge of that path $p$ by $\Delta$

From what we have so far, we cannot decrease the flow of any edge, so we still easily can get stuck

But consider an edge $e = (u, v)$ with a flow $f(e)$

To decrease this flow, we can instead increase the flow of a new edge $(v, u)$ by up to the amount $f(e)$

We thus need additional edges and therefore create a new graph $G_f$ for that
The residual graph

- We create a **residual graph** $G_f$ with the same nodes as $G$
- An edge in $G$ becomes either one or two edges in $G_f$ (one of them with reversed direction)
- Edges in $G_f$ have the capacities:

$$c_f(u, v) = \begin{cases} 
  c(u, v) - f(u, v) & \text{if } (u, v) \in E \text{ a forward edge} \\
  f(v, u) & \text{if } (v, u) \in E \text{ a backward edge} \\
  0 & \text{otherwise}
\end{cases}$$

- If $c(u, v) = a$, $f(u, v) = b$, and $a > b$, two edges are created in $G_f$:
  - One forward edge $(u, v)$ with capacity $a - b$ since this is how much we can increase the flow
  - One backward edge $(v, u)$ with capacity $b$ since this is how much we can decrease the flow
- If $f(u, v) = c(u, v)$ in $G$ then only a backward edge is created in $G_f$
- With $f(u, v) = c(u, v)$ in $G$ we can only decrease the flow on $(u, v)$
procedure \texttt{ford\_fulkerson}(G, s, t, c)

\begin{itemize}
\item for each $e \in E$ do $f(e) \leftarrow 0$
\item $G_f \leftarrow$ create initial residual graph
\item while $(p \leftarrow \text{find\_path}(G_f)) \neq \text{nil}$ do
\begin{itemize}
\item augment $G$ using $p$
\item update $f$ in $G$
\item update $c^R$ in $G^R$
\end{itemize}
\end{itemize}
Ford and Fulkerson did not specify how the path should be found.

Different options result in different time complexity and therefore it is sometimes called a **method** and not an algorithm — such as in Cormen, Leiserson, Rivest and Stein *Introduction to Algorithms* — i.e. CLRS (about 1300 pages).

If breadth first search is used, it is called the Edmond-Karp algorithm.

We will use the name Ford-Fulkerson algorithm.
Correctness of the Ford-Fulkerson algorithm

- We need to show that after updating $G$ (and $G_f$) they satisfy the two constraints for being a network flow, the capacity and conservation constraints.
- We also need to prove that it actually terminates — maybe it does not?
Termination of the Ford-Fulkerson algorithm

- Will it eventually terminate?
- It depends. If we use infinite precision of the representation of the capacities and flows, and the capacities are carefully selected irrational numbers, it will not terminate.
- Showing this is beyond the scope of the course.
- In practice this is not a problem because real numbers are represented as floating point numbers which means they really are rational numbers.
- If the capacities are integers, then all flows will also be integers and the algorithm clearly will terminate since it improves the flow at least by one each iteration (exactly by $\Delta$), and the sum of flows out from $s$ is an upper bound on the maximum flow.
Running time of the Ford-Fulkerson algorithm

- As usual, \( n \) is the number of nodes and \( m \) the number of edges in \( G \)
- Assume all capacities are integers
- Let the sum of capacities out from \( s \) be \( C \)
- We assume \( m \geq n/2 \) since each node has at least one neighbor, and this makes our analysis simpler

**Lemma**

*The Ford-Fulkerson algorithm can be implemented to run in \( O(Cm) \) time*

**Proof.**

At most \( C \) iterations to find a path are needed. Finding a path using e.g. breadth-first search and an adjacency list representation, can be done in \( O(n + m) \) and by our assumption this is equal to \( O(m) \). Updating \( G \) and \( G_f \) using the path also needs \( O(m) \) time
Recall a partitioning of $V$ into $A$ and $B$ means
- $V = A \cup B$, and
- $A \cap B = \emptyset$

A cut is a partitioning $(A, B)$ such that $s \in A$ and $t \in B$

How are cuts and flows related?
The value of a flow is denoted by $\nu(f)$

Consider any cut $(A, B)$

$f^{in}(s) = 0$

$\nu(f) = f^{out}(s)$

And also: $\nu(f) = f^{out}(s) - f^{in}(s)$

For all nodes $u \in V - \{s, t\}$ we have $f^{out}(u) = f^{in}(u)$

Thus for all nodes $u \in A - \{s\}$ we have $f^{out}(u) - f^{in}(u) = 0$

Therefore we can write: $\nu(f) = f^{out}(s) = \sum_{u \in A} f^{out}(u) - f^{in}(u)$

See next slide
Edges, flows, and cuts

- \( v(f) = \sum_{u \in A} f^{\text{out}}(u) - f^{\text{in}}(u) \)

Consider any edge \( e \in E \). We have four cases:

1. No end in \( A \): The edge does not affect the flow in \( A \)
2. From \( B \) to \( A \): the flow will be counted only as \(-f^{\text{in}}(u)\)
3. From \( A \) to \( B \): the flow will be counted only as \(f^{\text{out}}(u)\)
4. Both ends in \( A \): the flow will be counted both as \(f^{\text{out}}(u)\) and as \(f^{\text{in}}(u)\) above and thus cancels (by different terms in the sum)

Thus:

\[ v(f) = \sum_{u \in A} f^{\text{out}}(u) - f^{\text{in}}(u) = \sum_{\text{e out of } A} f^{\text{out}}(e) - \sum_{\text{e in to } A} f^{\text{in}}(e) \]

We have just shown:

Lemma

Let \( f \) be any \( s - t \) flow and \((A, B)\) any \( s - t \) cut. Then

\[ v(f) = f^{\text{out}}(A) - f^{\text{in}}(A) \]
Viewing the flow from \( t \)

- The value of a flow \( f \) can also be written \( \nu(f) = f^\text{in}(t) \)
- This is clear but we can also see it follows from what we just saw
- Since the edges out of \( A \) are the edges in to \( B \), we have \( f^\text{out}(A) = f^\text{in}(B) \)
- And since the edges out of \( B \) are the edges in to \( A \) we have \( f^\text{out}(B) = f^\text{in}(A) \)
- Therefore \( \nu(f) = f^\text{in}(B) - f^\text{out}(B) \)
- Since \( f^\text{out}(t) = 0 \) we have with \( B = \{ t \} \) the expected \( \nu(f) = f^\text{in}(t) \)
The capacity of a cut \((A, B)\) is \(\sum_{e \text{ out of } A} c(e)\) and it is denoted \(c(A, B)\).

We have:

\[
\begin{align*}
\nu(f) &= f^{\text{out}}(A) - f^{\text{in}}(A) \\
&\leq f^{\text{out}}(A) \\
&= \sum_{e \text{ out of } A} f(e) \\
&\leq \sum_{e \text{ out of } A} c(e) \\
&= c(A, B)
\end{align*}
\]

Therefore:

**Lemma**

The value of any flow is limited by the capacity of any cut: \(\nu(f) \leq c(A, B)\).
Exploiting $v(f) \leq c(A, B)$

- If we can show that the flow $f$ found by the Ford-Fulkerson algorithm is equal to the capacity of any cut $(A, B)$ then we know the algorithm finds the maximum flow.

**Lemma**

*If there is an $s$–$t$ flow $f$ in $G$ such that there is no $s$–$t$ path in $G_f$ then there is a cut $(A, B)$ in $G$ for which $v(f) = c(A, B)$, and, therefore, $f$ has the a maximum flow in $G$ and $(A, B)$ the minimum cut capacity in $G$.*

- A convenient shorter version:

**Lemma**

*If there is an $s$–$t$ flow $f$ in $G$ such that there is no $s$–$t$ path in $G_f$ then $f$ has the a maximum flow in $G$.*

- See the next slides for the proof.
No s-t path in $G_f$ means $f(e) = c(e)$ for $e$ crossing cut

Proof.

- Let $A$ be the set of nodes reachable from $s$ in $G_f$ and $B = V - A$
- Since $s$ is reachable from itself, $s \in A$ and therefore $A$ is not empty
- By assumption, there is no $s - t$ path in $G_f$ and therefore $t \in B$ and $B$ is not empty
- Thus $(A, B)$ is both a partition and a cut
- For any edge $e = (u, v)$ such that $u \in A$ and $v \in B$ we will next see that $f(e) = c(e)$
- Assume in contradiction that $f(e) < c(e)$. Since $c(e) - f(e) > 0$ there exists a forward edge $e$ in $G_f$ with $c_f(e) > 0$. Since $u \in A$ there is a path from $s$ to $v$ in $G_f$. Since this is a contradiction, $f(e) = c(e)$
No s-t path in $G_f$ means no flow back across cut

**Proof.**

- For any edge $e = (v, u)$ such that $v \in B$ and $u \in A$ we will next see that $f(e) = 0$

- Assume in contradiction that $f(e) > 0$. Since $f(e) > 0$ there exists a backward edge $e' = (u, v)$ with $c(e') > 0$ in $G_f$. But $e'$ makes $v$ reachable from $s$ in $G_f$ which is a contradiction, and therefore $f(e) = 0$

- We have showed that all edges $e$ out from $A$ have $f(e) = c(e)$ and all edges $e$ in to $A$ have $f(e) = 0$
We have shown that the flow computed by the Ford-Fulkerson algorithm is equal to a cut, and this means it is optimal.

We can compute a minimal cut \((A, B)\) in time \(O(m)\) using the method described in the previous proof.
The max-flow min-cut theorem (proved in 1956)

- Recall: the value of any flow is limited by the capacity of any cut:
  \[ v(f) \leq c(A, B) \]

**Theorem**

The maximum flow is equal to the minimum cut

**Proof.**

- Consider any flow \( f \) and cut \((A, B)\) such that \( v(f) = c(A, B) \)
- Assume in contradiction there exists a flow \( v'(f) > v(f) \)
- This would contradict \( v(f') \leq c(A, B) \), and therefore \( f \) is maximum
- Also assume in contradiction there exists a cut \((A', B')\) with \( c(A', B') < c(A, B) \)
- Again this would contradict \( v(f) \leq c'(A, B) \), and therefore \( c \) is minimum
Recall that the flow $f$ is incremented by the smallest value of $c_f(u, v)$ on the $s - t$ path in the residual graph $G_f$ (which we called $\Delta$). Therefore it is useful to find a path with a high $\Delta$.

If $C = \sum_{e \text{ out from } s} c(e)$ is a huge number this is particularly important. In this and many other situations it is not worthwhile to find the optimal value of a parameter (here $\Delta$) used to speed up an algorithm. We can look for paths with $\Delta \geq C/2^i$ for $i = 1, 2, \ldots$.

Another idea is to search for paths with the fewest number of edges. We will instead next look at a completely different approach.
Variants of the preflow-push maximum network flow algorithm, we will study next, are the fastest algorithms for finding the maximum network flow.

For brevity we simply call it the preflow-push algorithm.

The preflow-push algorithm also uses of the residual graph.

Instead of maintaining a valid flow which satisfies both the conservation constraint and the capacity constraint, it uses a weaker type of flow which only satisfies the capacity constraint.

The weaker flow is called a preflow.

At algorithm termination, the preflow will have become a valid flow.

In addition, it uses a height function for each node.
The preflow

- For each edge \( e \in E \) we have \( 0 \leq f(e) \leq c(e) \)
- Thus the capacity constraint is always satisfied
- Instead of the conservation constraint, a node \( u \neq s \) is allowed to have more incoming flow than outgoing
- Thus for each node \( u \in V - \{s\} \) we have

\[
\sum_{e \text{ into } u} f(e) \geq \sum_{e \text{ out from } u} f(e)
\]

- The excess preflow of a node \( u \) is

\[
e_f(u) = \sum_{e \text{ into } u} f(e) - \sum_{e \text{ out from } u} f(e)
\]

- Only \( s \) has a negative excess preflow
The height function

- There is a height function $h : V \rightarrow \mathbb{N}$
- $h(s) = n$
- $h(t) = 0$
- For $s$ and $t$ the heights cannot change and for other nodes they start at 0 and can increase
- The preflow on an edge $(u, v)$ can only increase if $h(u) = h(v) + 1$
- As we will see, $0 \leq h(u) < 2n - 1$ for $u \neq s$
The height function \( h \) and a preflow \( f \) are **compatible** if the following conditions are satisfied:

1. \( h(s) = n \) and \( h(t) = 0 \)
2. For all edges \( (v, w) \in E_f \) we have \( h(v) \leq h(w) + 1 \)

This means that on a simple path in \( G_f \) \( p = v_1, v_2, \ldots, v_k \) the height of \( v_{i+1} \) is at most one less than the height of \( v_i \).

Thus, on a simple path \( v_0, v_1, v_2, \ldots, v_k \) in \( G_f \) with \( v_0 = s \), the height of \( v_k \) is at least \( n - k \).
Preflow paths in $G_f$

Lemma

There can be no $s-t$ path in $G_f$ for a preflow $f$ compatible with $h$

Proof.

- Assume in contradiction there is a simple $st$ path $p$ in $G_f$
- Let $p = v_0, v_1, v_2, \ldots, v_k$, i.e. $s = v_0$ and $t = v_k$
- Then $h(t) \geq n - k$ and since $h(t) = 0$ it must be the case that $k = n$, and that the length of $p$ is $n$.
- This path cannot be simple. A contradiction.
Finding a maximum flow using $h$

**Lemma**

*If an $s-t$ flow $f$ is compatible with a height function $h$, then $f$ is a maximum flow.*

**Proof.**

- Recall: if there is an $s-t$ flow $f$ in $G$ such that there is no $s-t$ path in $G_f$ then $f$ has the a maximum flow in $G$.
- Since a flow $f$ also satisfies the conservation constraint, $f$ is a preflow.
- Therefore for a flow $f$ compatible with a height function $h$, there cannot be an $s-t$ path in $G_f$ (from previous slide).
- And no $s-t$ path in $G_f$ means $f$ is maximal.

- If we can transform a preflow to a flow compatible with a height function $h$, we have found a maximal flow.
We start with a preflow $f$ which, as we will see, is not a flow since it violates the conservation constraint.

The preflow $f$ is compatible with the height function $h$ and thus there is no $s-t$ path in $G_f$.

We will maintain the preflow so it remains compatible with an $h$.

The preflow will be modified until it becomes a flow $f$ which then will be a maximum flow.

Instead of maintaining valid but suboptimal flows which are improved, we will work towards a valid optimal flow.

The height of a node $u \in V - \{s, t\}$ can be at most $2n - 1$. 
Initial preflow and height function

- Each edge \((s, u)\) is assigned the initial preflow \(f(s, u) = c(s, u)\)
- For all other edges \(f(u, v) = 0\)
- \(h(s) = n\) and \(h(u) = 0\) for every node \(u \neq s\)
Three conditions must be satisfied for a push:

1. \( e_f(v) > 0 \)
2. \( h(v) > h(w) \)
3. \( (v, w) \in G_f \)

procedure \textit{push}(f, h, v, w) 

assert \( e_f(v) > 0 \) and \( h(v) > h(w) \) and \( (v, w) \in G_f \)

\( e \leftarrow (v, w) \)

if \( e \) is a forward edge then

\( \delta \leftarrow \min(e_f(v), c(e) - f(e)) \)

increase \( f(e) \) by \( \delta \)

else

\( e \leftarrow (w, v) \)

\( \delta \leftarrow \min(e_f(v), f(e)) \)

decrease \( f(e) \) by \( \delta \)
The purpose of a relabel is to increase the height of a node. It is done when the node has excess flow but nowhere to push it due to neighbors having too high height.

**procedure** `relabel(f, h, v)`

assert $e_f(v) > 0$ and for all edges $(v, w) \in E_f$ we have $h(w) \geq h(v)$

$h(v) \leftarrow h(v) + 1$
The preflow push algorithm

function preflow_push(G, s, t)
    h(s) ← n
    for each node u ≠ s do h(u) ← 0
    for each edge (s, v) do f(s, v) ← c(s, v)
    for each edge (u, v) such that u ≠ s do f(u, v) ← 0
    while there is a node v ≠ t with ef(v) > 0 do
        if there is a node w such that h(v) > h(w) and (v, w) ∈ Gf then
            push(h, f, v, w)
        else
            relabel(h, f, v)
    return f
Correctness of the preflow-push algorithm

- Initially the preflow $f$ and height function $h$ are compatible
- Each push satisfies the capacity constraints due to how the $\delta$ is calculated
- Each relabel increases the height of a node $v$ by one.
- This could violate the compatibility of $f$ and $h$
- The relevant condition for compatibility is:
  
  \[
  \text{For all edges } (v, w) \in E_f \text{ we have } h(v) \leq h(w) + 1
  \]
- If it is the case $h(v) > h(w)$ then a push and not a relabel is performed
- In the other case, $h(v) = h(w)$ the height of $v$ is incremented by one, and this still satisfies this condition since now $h(v) = h(w) + 1$.
- Therefore after a relabel, $f$ and $h$ remain compatible
Correctness of the preflow-push algorithm

- The algorithm terminates when no node other than \( t \) has excess flow.
- When this happens the preflow is a flow and as proved earlier, this is a maximum flow.
Paths to $s$ in $G_f$

**Lemma**

A node $v$ with $f_e(v) > 0$ has a path in $G_f$ to $s$

**Proof.**

- Let $A$ be the set of nodes with a path to $s$ in $G_f$, and $B = V - A$.
- $s \in A$
- No edge $(v, w)$ with $v \in A$ and $w \in B$ can have a positive excess flow since it would create a backward edge $(w, v)$ in $G_f$ so that then $w \in A$, which contradicts the assumption that $w \in B$.
- The sum of excess flow of nodes in $B$ is nonnegative (since only $s \in A$ has negative excess flow) can be written:

$$0 \leq \sum_{w \in B} e_f(w) = \sum_{w \in B} f^{\text{in}}(w) - \sum_{w \in B} f^{\text{out}}(w)$$
Proof.

- From the previous slide

\[ 0 \leq \sum_{w \in B} e_f(w) = \sum_{w \in B} f^\text{in}(w) - \sum_{w \in B} f^\text{out}(w) \]

- Considering edges which contribute to the above sums we have different cases.

- For an edge \((u, v)\) with \(u, v \in B\) these cancel.

- For an edge \((u, v)\) with \(u \in A\) and \(v \in B\) its flow is 0 as shown on the previous slide.

- Only edges \((u, v)\) with \(u \in B\) and \(v \in A\) remain

\[ 0 \leq \sum_{w \in B} e_f(w) = - \sum_{w \in B} f^\text{out}(w) \]
Paths to $s$ in $G_f$

Proof.

- From the previous slide

$$0 \leq \sum_{w \in B} e_f(w) = - \sum_{w \in B} f^{\text{out}}(w)$$

- But flows are nonnegative which implies they are all zero.
- Therefore, all nodes with excess are in the set $A$ and the claim follows.
**Lemma**

\[ h(u) \leq 2n - 1 \]

**Proof.**

- A height is increased by a relabel operation, which is applicable to nodes other than \( s \) and \( t \)
- As was proved by the previous lemma, a node \( u \) with \( f_e(u) > 0 \) has a simple path \( p \) to \( s \) in \( G_f \)
- The length of this path is at most \( n - 1 \).
- For a compatible \( h \) and \( f \) the heights on this path decrease at most by the length of the path, i.e. at most \( n - 1 \)
- Since \( h(s) = n \) we have \( h(u) - h(s) \leq n - 1 \) i.e. \( h(u) \leq 2n - 1 \)
- Since each node can have height at most \( 2n - 1 \) and there are \( n \) nodes, the number of relabel operations is limited to less than \( 2n^2 \)
Push operations

- A push operation increases the flow along an edge \((v, w)\).
- As much excess flow \(e_f(v)\) as possible is added to \(f(v, w)\).
- There are two limits:
  1. At most \(e_f(v)\) can be used since excess flow can never be negative.
  2. The capacity of the edge cannot be exceeded.
- A push at an edge \((v, w)\) is **saturating** if the limit was edge capacity:
  1. \((v, w)\) is a forward edge and \(\delta = c(v, w) - f(v, w)\), and
  2. \((v, w)\) is a backward edge and \(\delta = f(v, w)\).
- All other push operations are **nonsaturating** and were limited by the amount of excess flow for \(v\).
- After a nonsaturating push, \(v\) no longer has any excess flow: \(e_f(v) = 0\).
Saturating push operations

Lemma

The number of saturating push operations is less than $2nm$.

Proof.

- Consider any two nodes $v$ and $w$ such that they have an edge $(v, w)$.
- At a saturating push at the edge $(v, w)$ we have $h(v) = h(w) + 1$.
- Before a new push at the same edge, the height of $w$ must be increased by 2. Since the height of any node always is less than $2n - 1$, any node can increase by 2 at most $n - 1$ times.
- Counting both $v$ and $w$ the number of saturating pushes between them is less than $2n$.
- Since there are $m$ edges the total number of saturating pushes is less than $2nm$. 
Lemma

The number of nonsaturating push operations is at most $4n^2m$.

Proof.

- This lemma is proved using the potential function method.
- For a given preflow $f$ and height function $h$ we define

$$
\Phi(f, h) = \sum_{v: e_f(v) > 0} h(v)
$$

- Initially $\Phi(f, h) = 0$
- After a nonsaturating push $\Phi$ is reduced by $h(v)$ since $v$ no longer has any excess flow
- In addition $\Phi$ is incremented by $h(w)$ if $e_f(w) = 0$ before the push and not incremented if $w$ already had excess flow
- $\Phi(f, h)$ is decremented at least by 1 by a nonsaturating push
Lemma

The number of nonsaturating push operations is at most $4n^2m$.

Proof.

- A relabel operation increases $\Phi$ by 1 and since there are at most $2n^2$ relabel operations, they can contribute to $\Phi$ by at most $2n^2$.
- A saturating push $(v, w)$ may increase $e_f(w)$ from 0 and therefore increase $\Phi$ by at most $2n - 1$.
- With at most $2nm$ saturating push operations, $\Phi$ can be increased at most to $4n^2m$.
- Since $\Phi$ is decremented by at least 1 each nonsaturating push and is nonnegative, there can be at most $4n^2m$ nonsaturating push operations.
Maximum flow running times

Ford-Fulkerson \( O(Cm) \)
Preflow-push \( O(4mn^2) \)
- Both relabel and push take constant time
- The theoretical limitation of preflow-push is the number of nonsaturating push operations
- The preflow-push algorithm has \( O(mn^2) \) nonsaturating push operations
- In a dense graph this is \( O(n^4) \)
- It can be shown that if we always take the node \( v \) with \( c_f(v) > 0 \) and maximum height \( h(v) \) the number of nonsaturating push operations is at most \( 4n^3 \)