Network Flow Algorithms

Lecture 17

April 1, 2014

Part I

Algorithm(s) for Maximum Flow

Max-flow and min-cut?

Given a network G with capacities on the edges, and vertices $\mathbf{s}$ and $\mathbf{t}$, consider the maximum flow $\mathbf{f}$ between $\mathbf{s}$ and $\mathbf{t}$, and the minimum cut $\mathbf{(S, T)}$ between $\mathbf{s}$ and $\mathbf{t}$. Then, we have that

(A) $\mathbf{v(f) < c(S, T)}$.

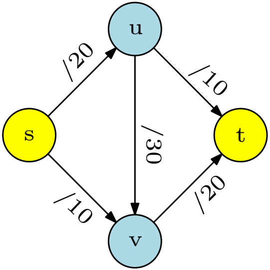
(B) $\mathbf{v(f) \leq c(S, T)}$.

(C) $\mathbf{v(f) > c(S, T)}$.

(D) $\mathbf{v(f) \geq c(S, T)}$.

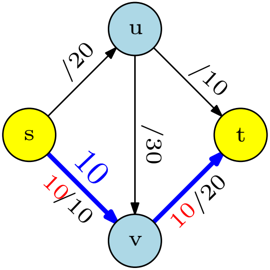
(E) $\mathbf{v(f) = c(S, T)}$.

Greedy Approach



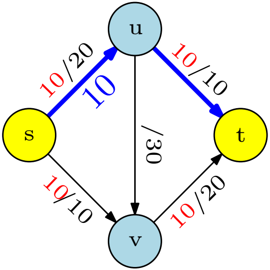
1. Begin with $\mathbf{f(e) = 0}$ for each edge.
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e} \in \mathbf{P}$.
3. **Augment** flow along this path.
4. Repeat augmentation for as long as possible.

Greedy Approach



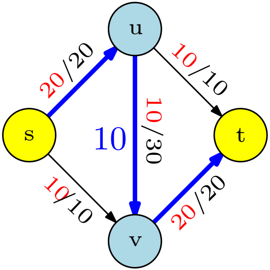
1. Begin with $\mathbf{f(e) = 0}$ for each edge.
2. Find a **s-t** path **P** with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e} \in \mathbf{P}$.
3. **Augment** flow along this path.
4. Repeat augmentation for as long as possible.

Greedy Approach



1. Begin with $\mathbf{f(e) = 0}$ for each edge.
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e} \in \mathbf{P}$.
3. **Augment** flow along this path.
4. Repeat augmentation for as long as possible.

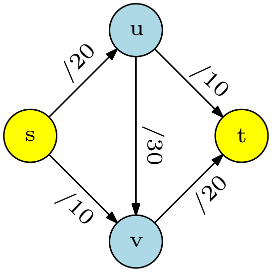
Greedy Approach



1. Begin with $\mathbf{f(e) = 0}$ for each edge.
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e} \in \mathbf{P}$.
3. **Augment** flow along this path.
4. Repeat augmentation for as long as possible.

Greedy Approach: Issues

Issues = What is this nonsense?

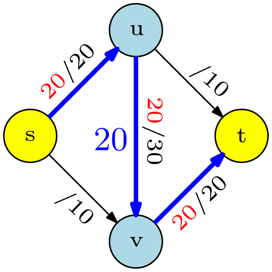


1. Begin with $\mathbf{f(e) = 0}$ for each edge
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e \in P}$
3. Augment flow along this path
4. Repeat augmentation for as long as possible.

Greedy can get stuck in sub-optimal flow!
Need to "push-back" flow along edge $(\mathbf{u}, \mathbf{v})$.

Greedy Approach: Issues

Issues = What is this nonsense?

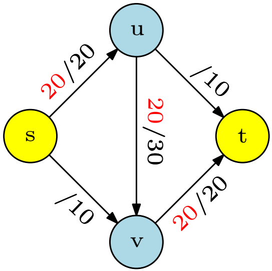


1. Begin with $\mathbf{f(e) = 0}$ for each edge
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e \in P}$
3. Augment flow along this path
4. Repeat augmentation for as long as possible.

Greedy can get stuck in sub-optimal flow!
Need to "push-back" flow along edge $\mathbf{(u, v)}$.

Greedy Approach: Issues

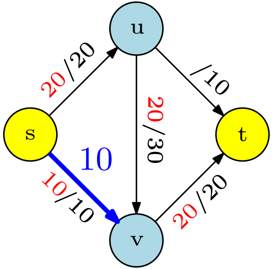
Issues = What is this nonsense?



1. Begin with $\mathbf{f(e) = 0}$ for each edge
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e} \in \mathbf{P}$
3. Augment flow along this path
4. Repeat augmentation for as long as possible.

Greedy can get stuck in sub-optimal flow!

Need to "push-back" flow along edge $\mathbf{(u, v)}$.



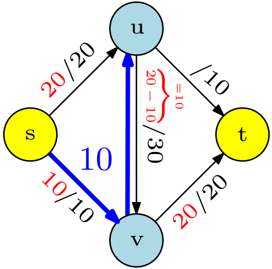
1. Begin with $\mathbf{f(e) = 0}$ for each edge
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e} \in \mathbf{P}$
3. Augment flow along this path
4. Repeat augmentation for as long as possible.

Greedy can get stuck in sub-optimal flow!

Need to "push-back" flow along edge $\mathbf{(u, v)}$.

Greedy Approach: Issues

Issues = What is this nonsense?

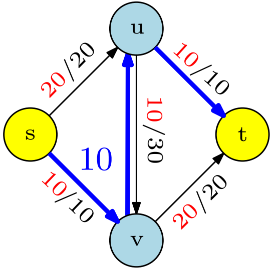


1. Begin with $\mathbf{f(e) = 0}$ for each edge
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e \in P}$
3. Augment flow along this path
4. Repeat augmentation for as long as possible.

Greedy can get stuck in sub-optimal flow!
Need to "push-back" flow along edge $\mathbf{(u, v)}$.

Greedy Approach: Issues

Issues = What is this nonsense?



1. Begin with $\mathbf{f(e) = 0}$ for each edge
2. Find a $\mathbf{s}$-$\mathbf{t}$ path $\mathbf{P}$ with $\mathbf{f(e) < c(e)}$ for every edge $\mathbf{e} \in \mathbf{P}$
3. Augment flow along this path
4. Repeat augmentation for as long as possible.

Greedy can get stuck in sub-optimal flow!
Need to "push-back" flow along edge $\mathbf{(u, v)}$.

Residual Graph

The "leftover" graph

Definition

For a network $\mathbf{G = (V, E)}$ and flow $\mathbf{f}$, the **residual graph** $\mathbf{G_f = (V', E')}$ of $\mathbf{G}$ with respect to $\mathbf{f}$ is

1. $\mathbf{V' = V}$,
2. **Forward Edges**: For each edge $\mathbf{e \in E}$ with $\mathbf{f(e) < c(e)}$, we add $\mathbf{e \in E'}$ with capacity $\mathbf{c(e) - f(e)}$.
3. **Backward Edges**: For each edge $\mathbf{e = (u, v) \in E}$ with $\mathbf{f(e) > 0}$, we add $\mathbf{(v, u) \in E'}$ with capacity $\mathbf{f(e)}$.

Residual Graph Example

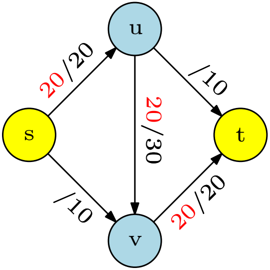


Figure : Flow on edges is indicated in red

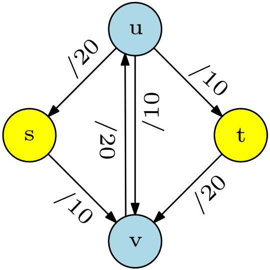


Figure : Residual Graph

Residual graph has...

Given a network with $\mathbf{n}$ vertices and $\mathbf{m}$ edges, and a valid flow $\mathbf{f}$ in it, the residual network $\mathbf{G_f}$, has

- **(A)** $\mathbf{m}$ edges.
- **(B)** $\leq \mathbf{2m}$ edges.
- **(C)** $\leq \mathbf{2m + n}$ edges.
- **(D)** $\mathbf{4m + 2n}$ edges.
- **(E)** $\mathbf{nm}$ edges.
- **(F)** just the right number of edges - not too many, not too few.

Residual Graph Property

Observation: Residual graph captures the "residual" problem exactly.

Lemma

Let $\mathbf{f}$ be a flow in $\mathbf{G}$ and $\mathbf{G_f}$ be the residual graph. If $\mathbf{f'}$ is a flow in $\mathbf{G_f}$ then $\mathbf{f} + \mathbf{f'}$ is a flow in $\mathbf{G}$ of value $\mathbf{v(f)} + \mathbf{v(f')}$.

Lemma

Let $\mathbf{f}$ and $\mathbf{f'}$ be two flows in $\mathbf{G}$ with $\mathbf{v(f')} \geq \mathbf{v(f)}$. Then there is a flow $\mathbf{f''}$ of value $\mathbf{v(f')} - \mathbf{v(f)}$ in $\mathbf{G_f}$.

Definition of $+$ and - for flows is intuitive and the above lemmas are easy in some sense but a bit messy to formally prove.

Residual Graph Property

Observation: Residual graph captures the "residual" problem exactly.

Lemma

Let $\mathbf{f}$ be a flow in $\mathbf{G}$ and $\mathbf{G_f}$ be the residual graph. If $\mathbf{f'}$ is a flow in $\mathbf{G_f}$ then $\mathbf{f} + \mathbf{f'}$ is a flow in $\mathbf{G}$ of value $\mathbf{v(f)} + \mathbf{v(f')}$.

Lemma

Let $\mathbf{f}$ and $\mathbf{f'}$ be two flows in $\mathbf{G}$ with $\mathbf{v(f')} \geq \mathbf{v(f)}$. Then there is a flow $\mathbf{f''}$ of value $\mathbf{v(f')} - \mathbf{v(f)}$ in $\mathbf{G_f}$.

Definition of $+$ and - for flows is intuitive and the above lemmas are easy in some sense but a bit messy to formally prove.

Residual Graph Property

Observation: Residual graph captures the "residual" problem exactly.

Lemma

Let $\mathbf{f}$ be a flow in $\mathbf{G}$ and $\mathbf{G_f}$ be the residual graph. If $\mathbf{f'}$ is a flow in $\mathbf{G_f}$ then $\mathbf{f} + \mathbf{f'}$ is a flow in $\mathbf{G}$ of value $\mathbf{v(f)} + \mathbf{v(f')}$.

Lemma

Let $\mathbf{f}$ and $\mathbf{f'}$ be two flows in $\mathbf{G}$ with $\mathbf{v(f')} \geq \mathbf{v(f)}$. Then there is a flow $\mathbf{f''}$ of value $\mathbf{v(f')} - \mathbf{v(f)}$ in $\mathbf{G_f}$.

Definition of $+$ and - for flows is intuitive and the above lemmas are easy in some sense but a bit messy to formally prove.

Residual Graph Property

Observation: Residual graph captures the "residual" problem exactly.

Lemma

Let $\mathbf{f}$ be a flow in $\mathbf{G}$ and $\mathbf{G_f}$ be the residual graph. If $\mathbf{f'}$ is a flow in $\mathbf{G_f}$ then $\mathbf{f} + \mathbf{f'}$ is a flow in $\mathbf{G}$ of value $\mathbf{v(f) + v(f')}$.

Lemma

Let $\mathbf{f}$ and $\mathbf{f'}$ be two flows in $\mathbf{G}$ with $\mathbf{v(f') \geq v(f)}$. Then there is a flow $\mathbf{f''}$ of value $\mathbf{v(f') - v(f)}$ in $\mathbf{G_f}$.

Definition of $+$ and - for flows is intuitive and the above lemmas are easy in some sense but a bit messy to formally prove.

Residual Graph Property: Implication

Recursive algorithm for finding a maximum flow:

```
MaxFlow(G, s, t):
    if the flow from s to t is 0 then
        return 0
    Find any flow f with v(f) > 0 in G
    Recursively compute a maximum flow f' in G_f
    Output the flow f + f'
```

Iterative algorithm for finding a maximum flow:

```
MaxFlow(G, s, t):
    Start with flow f that is 0 on all edges
    while there is a flow f' in G_f with v(f') > 0 do
        f = f + f'
        Update G_f

    Output f
```

Residual Graph Property: Implication

Recursive algorithm for finding a maximum flow:

```
MaxFlow(G, s, t):
    if the flow from s to t is 0 then
        return 0
    Find any flow f with v(f) > 0 in G
    Recursively compute a maximum flow f' in G_f
    Output the flow f + f'
```

Iterative algorithm for finding a maximum flow:

```
MaxFlow(G, s, t):
    Start with flow f that is 0 on all edges
    while there is a flow f' in G_f with v(f') > 0 do
        f = f + f'
        Update G_f

    Output f
```

Ford-Fulkerson Algorithm

algFordFulkerson

```
for every edge e, f(e) = 0
G_f is residual graph of G with respect to f
while G_f has a simple s-t path do
    let P be simple s-t path in G_f
    f = augment(f, P)
    Construct new residual graph G_f.
```

augment(f, P)

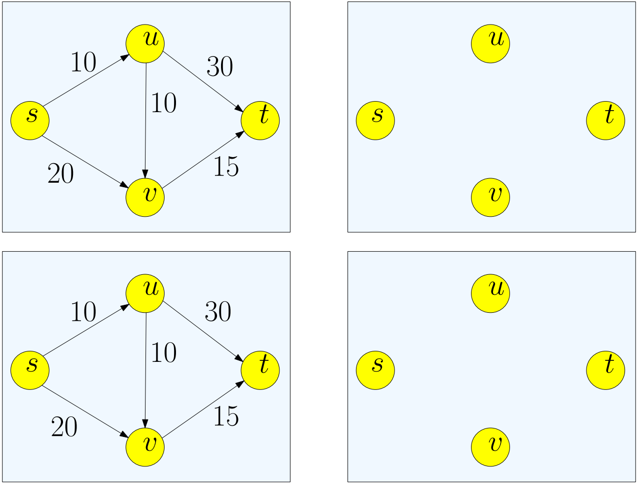
```
let b be bottleneck capacity,
    i.e., min capacity of edges in P (in G_f)
for each edge (u, v) in P do
    if e = (u, v) is a forward edge then
        f(e) = f(e) + b
    else (* (u, v) is a backward edge *)
        let e = (v, u) (* (v, u) is in G *)
        f(e) = f(e) - b
return f
```

Ford-Fulkerson Algorithm

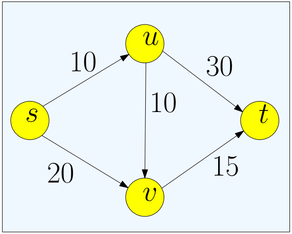
```
algFordFulkerson
    for every edge e, f(e) = 0
    G_f is residual graph of G with respect to f
    while G_f has a simple s-t path do
        let P be simple s-t path in G_f
        f = augment(f, P)
        Construct new residual graph G_f.
```

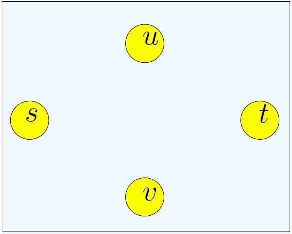
```
augment(f,P)
    let b be bottleneck capacity,
        i.e., min capacity of edges in P (in G_f)
    for each edge (u, v) in P do
        if e = (u, v) is a forward edge then
            f(e) = f(e) + b
        else (* (u, v) is a backward edge *)
            let e = (v, u) (* (v, u) is in G *)
            f(e) = f(e) − b
    return f
```

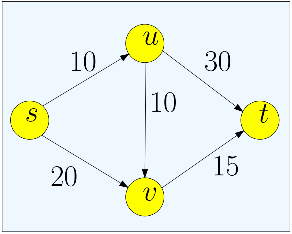
Example

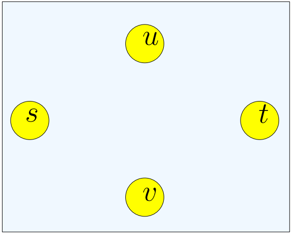


Example continued

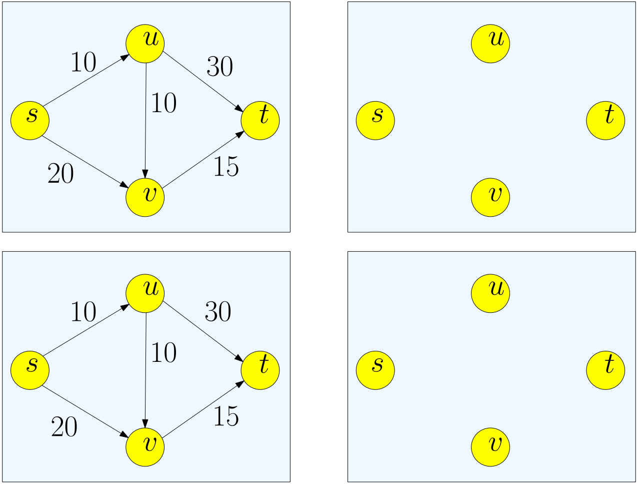




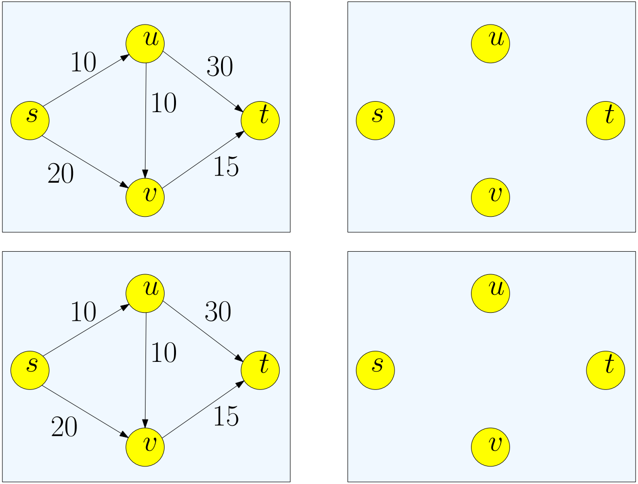




Example continued



Example continued



Properties about Augmentation: Flow

Lemma

If $\mathbf{f}$ is a flow and $\mathbf{P}$ is a simple $\mathbf{s}$-$\mathbf{t}$ path in $\mathbf{G_f}$, then $\mathbf{f' = \mathtt{augment}(f, P)}$ is also a flow.

Proof.

Verify that $\mathbf{f'}$ is a flow. Let $\mathbf{b}$ be augmentation amount.

1. Capacity constraint: If $\mathbf{(u,v) \in P}$ is a forward edge then $\mathbf{f'(e) = f(e) + b}$ and $\mathbf{b \le c(e) - f(e)}$. If $\mathbf{(u,v) \in P}$ is a backward edge, then letting $\mathbf{e = (v,u)}$, $\mathbf{f'(e) = f(e) - b}$ and $\mathbf{b \le f(e)}$. Both cases $\mathbf{0 \le f'(e) \le c(e)}$.
2. Conservation constraint: Let $\mathbf{v}$ be an internal node. Let $\mathbf{e_1, e_2}$ be edges of $\mathbf{P}$ incident to $\mathbf{v}$. Four cases based on whether $\mathbf{e_1, e_2}$ are forward or backward edges. Check cases (see fig next slide). $\square$

Properties about Augmentation: Flow

Lemma

If $\mathbf{f}$ *is a flow and* $\mathbf{P}$ *is a simple* $\mathbf{s}$-$\mathbf{t}$ *path in* $\mathbf{G_f}$*, then* $\mathbf{f}' = \mathtt{augment}(\mathbf{f}, \mathbf{P})$ *is also a flow.*

Proof.

Verify that $\mathbf{f}'$ is a flow. Let $\mathbf{b}$ be augmentation amount.

1. Capacity constraint: If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a forward edge then $\mathbf{f}'(\mathbf{e}) = \mathbf{f}(\mathbf{e}) + \mathbf{b}$ and $\mathbf{b} \leq \mathbf{c}(\mathbf{e}) - \mathbf{f}(\mathbf{e})$. If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a backward edge, then letting $\mathbf{e} = (\mathbf{v}, \mathbf{u})$, $\mathbf{f}'(\mathbf{e}) = \mathbf{f}(\mathbf{e}) - \mathbf{b}$ and $\mathbf{b} \leq \mathbf{f}(\mathbf{e})$. Both cases $\mathbf{0} \leq \mathbf{f}'(\mathbf{e}) \leq \mathbf{c}(\mathbf{e})$.
2. Conservation constraint: Let $\mathbf{v}$ be an internal node. Let $\mathbf{e}_1, \mathbf{e}_2$ be edges of $\mathbf{P}$ incident to $\mathbf{v}$. Four cases based on whether $\mathbf{e}_1, \mathbf{e}_2$ are forward or backward edges. Check cases (see fig next slide). □

Properties about Augmentation: Flow

Lemma

If $\mathbf{f}$ *is a flow and* $\mathbf{P}$ *is a simple* $\mathbf{s}$-$\mathbf{t}$ *path in* $\mathbf{G_f}$*, then* $\mathbf{f'} = \mathtt{augment}(\mathbf{f}, \mathbf{P})$ *is also a flow.*

Proof.

Verify that $\mathbf{f'}$ is a flow. Let $\mathbf{b}$ be augmentation amount.

1. Capacity constraint: If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a forward edge then $\mathbf{f'(e)} = \mathbf{f(e)} + \mathbf{b}$ and $\mathbf{b} \leq \mathbf{c(e)} - \mathbf{f(e)}$. If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a backward edge, then letting $\mathbf{e} = (\mathbf{v}, \mathbf{u})$, $\mathbf{f'(e)} = \mathbf{f(e)} - \mathbf{b}$ and $\mathbf{b} \leq \mathbf{f(e)}$. Both cases $\mathbf{0} \leq \mathbf{f'(e)} \leq \mathbf{c(e)}$.
2. Conservation constraint: Let $\mathbf{v}$ be an internal node. Let $\mathbf{e_1}, \mathbf{e_2}$ be edges of $\mathbf{P}$ incident to $\mathbf{v}$. Four cases based on whether $\mathbf{e_1}, \mathbf{e_2}$ are forward or backward edges. Check cases (see fig next slide). □

Properties about Augmentation: Flow

Lemma

If $\mathbf{f}$ *is a flow and* $\mathbf{P}$ *is a simple* $\mathbf{s}$-$\mathbf{t}$ *path in* $\mathbf{G_f}$, *then* $\mathbf{f'} = \texttt{augment}(\mathbf{f}, \mathbf{P})$ *is also a flow.*

Proof.

Verify that $\mathbf{f'}$ is a flow. Let $\mathbf{b}$ be augmentation amount.

1. Capacity constraint: If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a forward edge then $\mathbf{f'(e) = f(e) + b}$ and $\mathbf{b \leq c(e) - f(e)}$. If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a backward edge, then letting $\mathbf{e = (v, u)}$, $\mathbf{f'(e) = f(e) - b}$ and $\mathbf{b \leq f(e)}$. Both cases $\mathbf{0 \leq f'(e) \leq c(e)}$.
2. Conservation constraint: Let $\mathbf{v}$ be an internal node. Let $\mathbf{e_1, e_2}$ be edges of $\mathbf{P}$ incident to $\mathbf{v}$. Four cases based on whether $\mathbf{e_1, e_2}$ are forward or backward edges. Check cases (see fig next slide). □

Properties about Augmentation: Flow

Lemma

If $\mathbf{f}$ *is a flow and* $\mathbf{P}$ *is a simple* $\mathbf{s}$-$\mathbf{t}$ *path in* $\mathbf{G_f}$, *then* $\mathbf{f'} = \mathtt{augment}(\mathbf{f}, \mathbf{P})$ *is also a flow.*

Proof.

Verify that $\mathbf{f'}$ is a flow. Let $\mathbf{b}$ be augmentation amount.

1. Capacity constraint: If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a forward edge then $\mathbf{f'(e) = f(e) + b}$ and $\mathbf{b \leq c(e) - f(e)}$. If $(\mathbf{u}, \mathbf{v}) \in \mathbf{P}$ is a backward edge, then letting $\mathbf{e = (v, u)}$, $\mathbf{f'(e) = f(e) - b}$ and $\mathbf{b \leq f(e)}$. Both cases $\mathbf{0 \leq f'(e) \leq c(e)}$.
2. Conservation constraint: Let $\mathbf{v}$ be an internal node. Let $\mathbf{e_1, e_2}$ be edges of $\mathbf{P}$ incident to $\mathbf{v}$. Four cases based on whether $\mathbf{e_1, e_2}$ are forward or backward edges. Check cases (see fig next slide). $\square$

Properties of Augmentation

Conservation Constraint

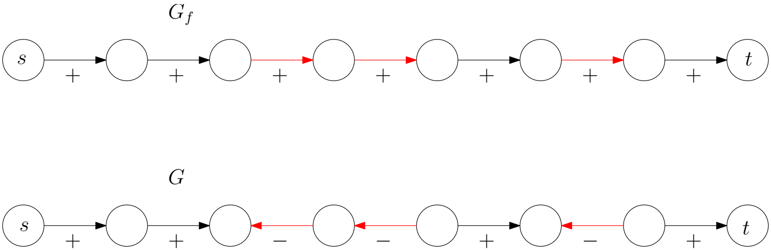


Figure : Augmenting path $\mathbf{P}$ in $\mathbf{G_f}$ and corresponding change of flow in $\mathbf{G}$. Red edges are backward edges.

Rational, integer or real?

Consider a network flow instance where all the numbers are integers. **algFordFulkerson** on this network outputs a flow such that its value is

(A) Since the algorithm runs on a RAM machine, and it can perform any arithmetic operation, the output is a real number.

(B) The algorithm does only subtract, add, divide and multiply operations. Thus the output is a rational number.

(C) The algorithm does only subtract and add operations on numbers. Thus the output is an integer number.

(D) **algFordFulkerson** does not necessarily terminates, so the question is ill defined.

(E) If the capacities are negative, the algorithm might output $+\infty$ (which is not an integer, rational or real number).

Properties of Augmentation

Integer Flow

Lemma

At every stage of the Ford-Fulkerson algorithm, the flow values on the edges (i.e., $\mathbf{f(e)}$, for all edges $\mathbf{e}$) and the residual capacities in $\mathbf{G_f}$ are integers.

Proof.

Initial flow and residual capacities are integers. Suppose lemma holds for $\mathbf{j}$ iterations. Then in $\mathbf{(j+1)}$st iteration, minimum capacity edge $\mathbf{b}$ is an integer, and so flow after augmentation is an integer. □

Progress in Ford-Fulkerson

Proposition

Let $\mathbf{f}$ *be a flow and* $\mathbf{f'}$ *be flow after one augmentation. Then* $\mathbf{v(f) < v(f')}$.

Proof.

Let $\mathbf{P}$ be an augmenting path, i.e., $\mathbf{P}$ is a simple $\mathbf{s}$-$\mathbf{t}$ path in residual graph. We have the following.

1. First edge $\mathbf{e}$ in $\mathbf{P}$ must leave $\mathbf{s}$.
2. Original network $\mathbf{G}$ has no incoming edges to $\mathbf{s}$; hence $\mathbf{e}$ is a forward edge.
3. $\mathbf{P}$ is simple and so never returns to $\mathbf{s}$.
4. Thus, value of flow increases by the flow on edge $\mathbf{e}$. □

Termination proof for integral flow

Theorem

Let $\mathbf{C}$ *be the minimum cut value; in particular* $\mathbf{C} \leq \sum_{\mathbf{e} \textit{ out of } \mathbf{s}} \mathbf{c(e)}$*. Ford-Fulkerson algorithm terminates after finding at most* $\mathbf{C}$ *augmenting paths.*

Proof.

The value of the flow increases by at least $\mathbf{1}$ after each augmentation. Maximum value of flow is at most $\mathbf{C}$. □

Running time

1. Number of iterations $\leq \mathbf{C}$.
2. Number of edges in $\mathbf{G_f} \leq \mathbf{2m}$.
3. Time to find augmenting path is $\mathbf{O(n + m)}$.
4. Running time is $\mathbf{O(C(n + m))}$ (or $\mathbf{O(mC)}$).

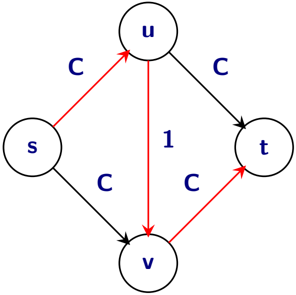
Efficiency of Ford-Fulkerson

Running time $= O(mC)$ is not polynomial. Can the running time be as $\Omega(mC)$ or is our analysis weak?

Ford-Fulkerson can take $\Omega(C)$ iterations.

Efficiency of Ford-Fulkerson

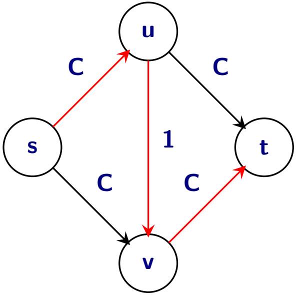
Running time $= \mathbf{O(mC)}$ is not polynomial. Can the running time be as $\mathbf{\Omega(mC)}$ or is our analysis weak?

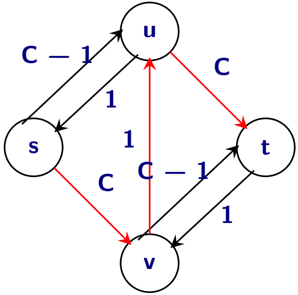


Ford-Fulkerson can take $\mathbf{\Omega(C)}$ iterations.

Efficiency of Ford-Fulkerson

Running time $= \mathbf{O(mC)}$ is not polynomial. Can the running time be as $\mathbf{\Omega(mC)}$ or is our analysis weak?





Ford-Fulkerson can take $\mathbf{\Omega(C)}$ iterations.

Correctness of Ford-Fulkerson

Why the augmenting path approach works

Question: When the algorithm terminates, is the flow computed the maximum **s**-**t** flow?

Proof idea: show a cut of value equal to the flow. Also shows that maximum flow is equal to minimum cut!

Correctness of Ford-Fulkerson

Why the augmenting path approach works

Question: When the algorithm terminates, is the flow computed the maximum **s**-**t** flow?

Proof idea: show a cut of value equal to the flow. Also shows that maximum flow is equal to minimum cut!

Recalling Cuts

Definition

Given a flow network an **s-t cut** is a set of edges $\mathbf{E}' \subset \mathbf{E}$ such that removing $\mathbf{E}'$ *disconnects* $\mathbf{s}$ from $\mathbf{t}$: in other words there is no directed $\mathbf{s} \rightarrow \mathbf{t}$ path in $\mathbf{E} - \mathbf{E}'$. **Capacity** of cut $\mathbf{E}'$ is $\sum_{\mathbf{e} \in \mathbf{E}'} \mathbf{c}(\mathbf{e})$.

Let $\mathbf{A} \subset \mathbf{V}$ such that

1. $\mathbf{s} \in \mathbf{A}$, $\mathbf{t} \not\in \mathbf{A}$, and
2. $\mathbf{B} = \mathbf{V} \setminus -\mathbf{A}$ and hence $\mathbf{t} \in \mathbf{B}$.

Define $(\mathbf{A}, \mathbf{B}) = \{(\mathbf{u}, \mathbf{v}) \in \mathbf{E} \mid \mathbf{u} \in \mathbf{A}, \mathbf{v} \in \mathbf{B}\}$

Claim

$(\mathbf{A}, \mathbf{B})$ *is an* $\mathbf{s}$-$\mathbf{t}$ *cut.*

Recall: Every *minimal* $\mathbf{s}$-$\mathbf{t}$ cut $\mathbf{E}'$ is a cut of the form $(\mathbf{A}, \mathbf{B})$.

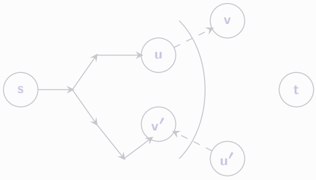
Ford-Fulkerson Correctness

Lemma

If there is no s-t *path in* G_f *then there is some cut* (A, B) *such that* $v(f) = c(A, B)$

Proof.

Let A be all vertices reachable from s in G_f; $B = V \setminus A$.



1. $s \in A$ and $t \in B$. So (A, B) is an s-t cut in G.
2. If $e = (u, v) \in G$ with $u \in A$ and $v \in B$, then $f(e) = c(e)$ (saturated edge) because otherwise v is reachable from s in G_f.

□

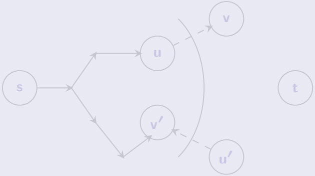
Ford-Fulkerson Correctness

Lemma

If there is no s-t *path in* G_f *then there is some cut* (A, B) *such that* $v(f) = c(A, B)$

Proof.

Let A be all vertices reachable from s in G_f; $B = V \setminus A$.



1. $s \in A$ and $t \in B$. So (A, B) is an s-t cut in G.
2. If $e = (u, v) \in G$ with $u \in A$ and $v \in B$, then $f(e) = c(e)$ (saturated edge) because otherwise v is reachable from s in G_f.

$\square$

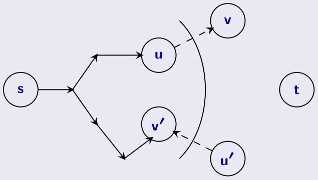
Ford-Fulkerson Correctness

Lemma

If there is no $\mathbf{s}$-$\mathbf{t}$ *path in* $\mathbf{G_f}$ *then there is some cut* $\mathbf{(A, B)}$ *such that* $\mathbf{v(f) = c(A, B)}$

Proof.

Let $\mathbf{A}$ be all vertices reachable from $\mathbf{s}$ in $\mathbf{G_f}$; $\mathbf{B = V \setminus A}$.



1. $\mathbf{s \in A}$ and $\mathbf{t \in B}$. So $\mathbf{(A, B)}$ is an $\mathbf{s}$-$\mathbf{t}$ cut in $\mathbf{G}$.
2. If $\mathbf{e = (u, v) \in G}$ with $\mathbf{u \in A}$ and $\mathbf{v \in B}$, then $\mathbf{f(e) = c(e)}$ (saturated edge) because otherwise $\mathbf{v}$ is reachable from $\mathbf{s}$ in $\mathbf{G_f}$.

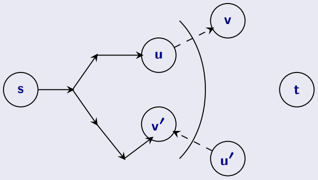
Ford-Fulkerson Correctness

Lemma

If there is no $\mathbf{s}$-$\mathbf{t}$ *path in* G_f *then there is some cut* (A, B) *such that* $v(f) = c(A, B)$

Proof.

Let A be all vertices reachable from s in G_f; $B = V \setminus A$.

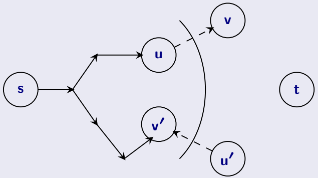


1. $s \in A$ and $t \in B$. So (A, B) is an s-t cut in G.
2. If $e = (u, v) \in G$ with $u \in A$ and $v \in B$, then $f(e) = c(e)$ (saturated edge) because otherwise v is reachable from s in G_f.

□

Lemma Proof Continued

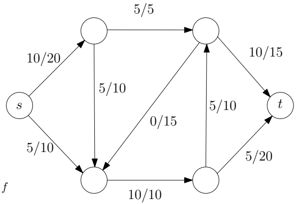
Proof.



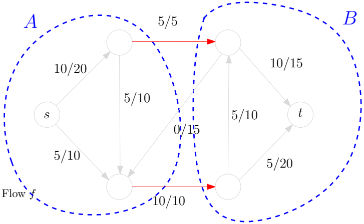
1. If $\mathbf{e} = (\mathbf{u}', \mathbf{v}') \in \mathbf{G}$ with $\mathbf{u}' \in \mathbf{B}$ and $\mathbf{v}' \in \mathbf{A}$, then $\mathbf{f(e) = 0}$ because otherwise $\mathbf{u}'$ is reachable from $\mathbf{s}$ in $\mathbf{G_f}$
2. Thus,

$$
\begin{aligned}
\mathbf{v(f)} &= \mathbf{f}^{\mathrm{out}}(\mathbf{A}) - \mathbf{f}^{\mathrm{in}}(\mathbf{A}) \\
&= \mathbf{f}^{\mathrm{out}}(\mathbf{A}) - \mathbf{0} \\
&= \mathbf{c(A, B) - 0} \\
&= \mathbf{c(A, B)}.
\end{aligned}
$$

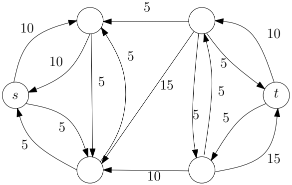
$\square$



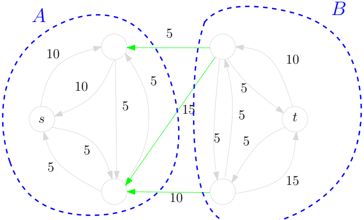
Flow f



Flow f

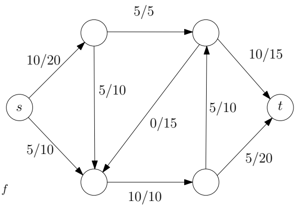


Residual graph G_f: no s-t path

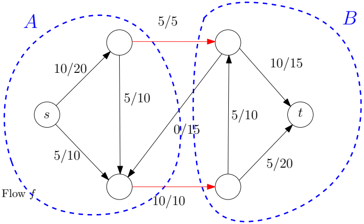


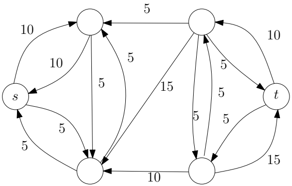
A is reachable set from s in G_f

Example

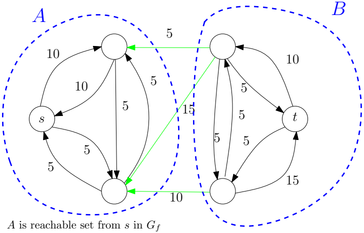


Flow f





Residual graph G_f: no s-t path



A is reachable set from s in G_f

Ford-Fulkerson Correctness

Theorem

The flow returned by the algorithm is the maximum flow.

Proof.

1. For any flow $\mathbf{f}$ and $\mathbf{s}$-$\mathbf{t}$ cut $(\mathbf{A}, \mathbf{B})$, $\mathbf{v(f)} \leq \mathbf{c(A, B)}$.
2. For flow $\mathbf{f}^*$ returned by algorithm, $\mathbf{v(f^*)} = \mathbf{c(A^*, B^*)}$ for some $\mathbf{s}$-$\mathbf{t}$ cut $(\mathbf{A}^*, \mathbf{B}^*)$.
3. Hence, $\mathbf{f}^*$ is maximum.

□

Max-Flow Min-Cut Theorem and Integrality of Flows

Theorem

For any network $\mathbf{G}$*, the value of a maximum* $\mathbf{s}$-$\mathbf{t}$ *flow is equal to the capacity of the minimum* $\mathbf{s}$-$\mathbf{t}$ *cut.*

Proof.

Ford-Fulkerson algorithm terminates with a maximum flow of value equal to the capacity of a (minimum) cut. $\square$

Max-Flow Min-Cut Theorem and Integrality of Flows

Theorem

For any network $\mathbf{G}$ *with integer capacities, there is a maximum* $\mathbf{s}$-$\mathbf{t}$ *flow that is integer valued.*

Proof.

Ford-Fulkerson algorithm produces an integer valued flow when capacities are integers. □

Does it terminate?

(A) **algFordFulkerson** always terminates.

(B) **algFordFulkerson** might not terminate if the input has real numbers.

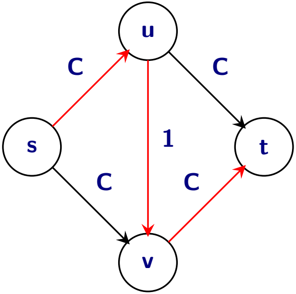
(C) **algFordFulkerson** might not terminate if the input has rational numbers.

(D) **algFordFulkerson** might not terminate if the input is only integer numbers that are sufficiently large.

Efficiency of Ford-Fulkerson

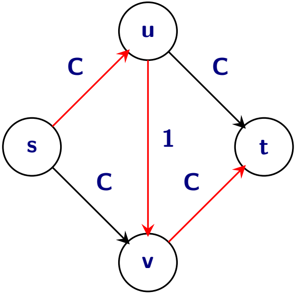
Running time $= \mathbf{O(mC)}$ is not polynomial. Can the upper bound be achieved?

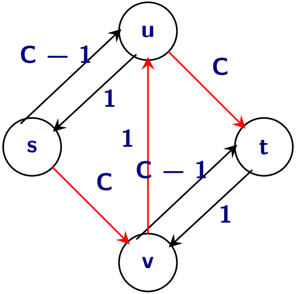
Running time $= \mathbf{O(mC)}$ is not polynomial. Can the upper bound be achieved?



Efficiency of Ford-Fulkerson

Running time $= \mathbf{O(mC)}$ is not polynomial. Can the upper bound be achieved?





Polynomial Time Algorithms

Question: Is there a polynomial time algorithm for maxflow?

Question: Is there a variant of Ford-Fulkerson that leads to a polynomial time algorithm? Can we choose an augmenting path in some clever way? Yes! Two variants.

1. Choose the augmenting path with largest bottleneck capacity.
2. Choose the shortest augmenting path.

Polynomial Time Algorithms

Question: Is there a polynomial time algorithm for maxflow?

Question: Is there a variant of Ford-Fulkerson that leads to a polynomial time algorithm? Can we choose an augmenting path in some clever way? Yes! Two variants.

1. Choose the augmenting path with largest bottleneck capacity.
2. Choose the shortest augmenting path.

Polynomial Time Algorithms

Question: Is there a polynomial time algorithm for maxflow?

Question: Is there a variant of Ford-Fulkerson that leads to a polynomial time algorithm? Can we choose an augmenting path in some clever way? Yes! Two variants.

1. Choose the augmenting path with largest bottleneck capacity.
2. Choose the shortest augmenting path.

Finding path with largest bottleneck capacity

$\mathbf{G_f}$ - residual network with (residual) capacities.
$\mathbf{n}$ vertices and $\mathbf{m}$ edges.
Finding the $\mathbf{s}$-$\mathbf{t}$ path with largest bottleneck capacity can be done (faster is better) in:

- **(A)** $\mathbf{O(n+m)}$
- **(B)** $\mathbf{O(n \log + m)}$
- **(C)** $\mathbf{O(nm)}$
- **(D)** $\mathbf{O(m^2)}$
- **(E)** $\mathbf{O(m^3)}$

time (expected or deterministic is fine here).

Augmenting Paths with Large Bottleneck Capacity

1. Pick augmenting paths with largest bottleneck capacity in each iteration of Ford-Fulkerson.
2. How do we find path with largest bottleneck capacity?
    1. Assume we know Δ the bottleneck capacity
    2. Remove all edges with residual capacity $\leq \Delta$
    3. Check if there is a path from s to t
    4. Do binary search to find largest Δ
    5. Running time: $O(m \log C)$
3. Can we bound the number of augmentations? Can show that in $O(m \log C)$ augmentations the algorithm reaches a max flow. This leads to an $O(m^2 \log^2 C)$ time algorithm.

Augmenting Paths with Large Bottleneck Capacity

1. Pick augmenting paths with largest bottleneck capacity in each iteration of Ford-Fulkerson.
2. How do we find path with largest bottleneck capacity?
    1. Assume we know Δ the bottleneck capacity
    2. Remove all edges with residual capacity $\leq \Delta$
    3. Check if there is a path from s to t
    4. Do binary search to find largest Δ
    5. Running time: $O(m \log C)$
3. Can we bound the number of augmentations? Can show that in $O(m \log C)$ augmentations the algorithm reaches a max flow. This leads to an $O(m^2 \log^2 C)$ time algorithm.

Augmenting Paths with Large Bottleneck Capacity

1. Pick augmenting paths with largest bottleneck capacity in each iteration of Ford-Fulkerson.
2. How do we find path with largest bottleneck capacity?
    1. Assume we know Δ the bottleneck capacity
    2. Remove all edges with residual capacity $\leq \Delta$
    3. Check if there is a path from s to t
    4. Do binary search to find largest Δ
    5. Running time: $O(m \log C)$
3. Can we bound the number of augmentations? Can show that in $O(m \log C)$ augmentations the algorithm reaches a max flow. This leads to an $O(m^2 \log^2 C)$ time algorithm.

Augmenting Paths with Large Bottleneck Capacity

How do we find path with largest bottleneck capacity?

1. Max bottleneck capacity is one of the edge capacities. Why?
2. Can do binary search on the edge capacities. First, sort the edges by their capacities and then do binary search on that array as before.
3. Algorithm's running time is $O(m \log m)$.
4. Different algorithm that also leads to $O(m \log m)$ time algorithm by adapting Prim's algorithm.
5. $O(m)$ time using linear-time median selection algorithm.

Removing Dependence on C

1. **Dinic [1970]**, **Edmonds and Karp [1972]**
   Picking augmenting paths with fewest number of edges yields a $O(m^2 n)$ algorithm, i.e., independent of C. Such an algorithm is called a strongly polynomial time algorithm since the running time does not depend on the numbers (assuming RAM model). (Many implementation of Ford-Fulkerson would actually use shortest augmenting path if they use **BFS** to find an s-t path).
2. Further improvements can yield algorithms running in $O(mn \log n)$, or $O(n^3)$.

Ford-Fulkerson Algorithm

```
algEdmondsKarp
    for every edge e, f(e) = 0
    G_f is residual graph of G with respect to f
    while G_f has a simple s-t path do
        Perform BFS in G_f
        P: shortest s-t path in G_f
        f = augment(f, P)
        Construct new residual graph G_f.
```

Running time $O(m^2n)$.

Finding a Minimum Cut

Question: How do we find an actual minimum $\mathbf{s}$-$\mathbf{t}$ cut?

Proof gives the algorithm!

1. Compute an $\mathbf{s}$-$\mathbf{t}$ maximum flow $\mathbf{f}$ in $\mathbf{G}$
2. Obtain the residual graph $\mathbf{G_f}$
3. Find the nodes $\mathbf{A}$ reachable from $\mathbf{s}$ in $\mathbf{G_f}$
4. Output the cut $\mathbf{(A, B) = \{(u, v) \mid u \in A, v \in B\}}$. Note: The cut is found in $\mathbf{G}$ while $\mathbf{A}$ is found in $\mathbf{G_f}$

Running time is essentially the same as finding a maximum flow.

Note: Given $\mathbf{G}$ and a flow $\mathbf{f}$ there is a linear time algorithm to check if $\mathbf{f}$ is a maximum flow and if it is, outputs a minimum cut. How?

Finding a Minimum Cut

Question: How do we find an actual minimum s-t cut?
Proof gives the algorithm!

1. Compute an s-t maximum flow f in G
2. Obtain the residual graph G_f
3. Find the nodes A reachable from s in G_f
4. Output the cut $(A, B) = \{(u, v) \mid u \in A, v \in B\}$. Note: The cut is found in G while A is found in G_f

Running time is essentially the same as finding a maximum flow.

Note: Given G and a flow f there is a linear time algorithm to check if f is a maximum flow and if it is, outputs a minimum cut. How?

Notes

Notes

Notes

Notes

Dinic, E. A. (1970). Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Math. Doklady*, 11:1277–1280.

Edmonds, J. and Karp, R. M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. *J. Assoc. Comput. Mach.*, 19(2):248–264.