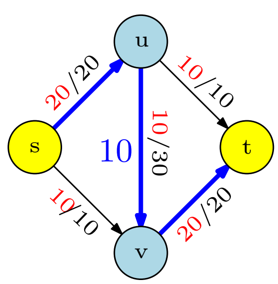
Chapter 18

Network Flow Algorithms

OLD CS 473: Fundamental Algorithms, Spring 2015
March 31, 2015

18.1 Algorithm(s) for Maximum Flow

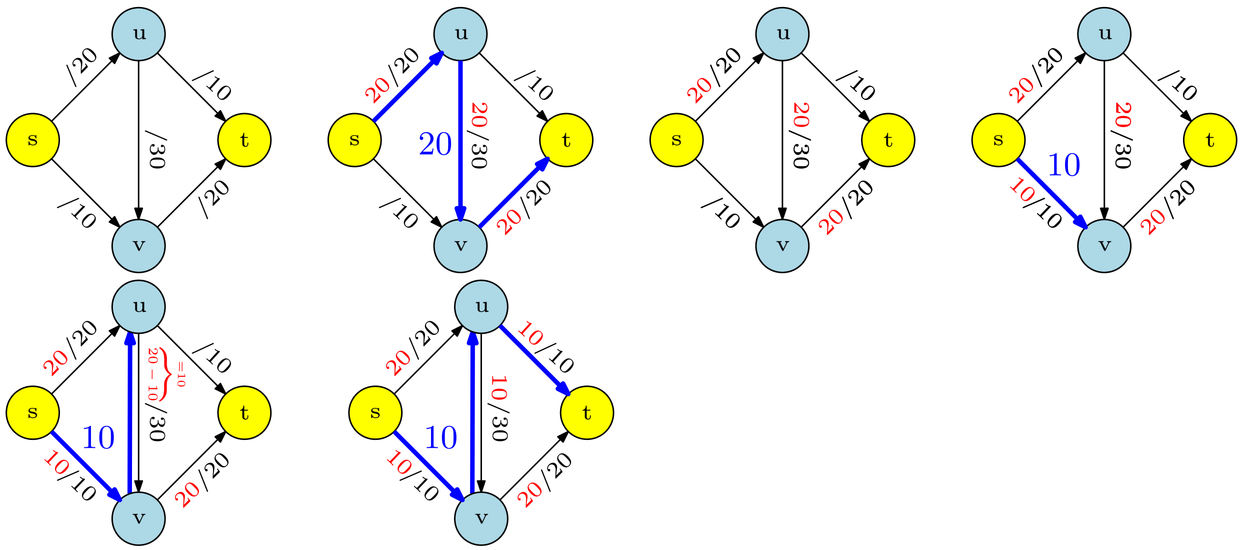
18.1.0.1 Greedy Approach



(A) Begin with $f(e) = 0$ for each edge.
(B) Find a s-t path P with $f(e) < c(e)$ for every edge $e \in P$.
(C) ***Augment*** flow along this path.
(D) Repeat augmentation for as long as possible.

18.1.1 Greedy Approach: Issues

18.1.1.1 Issues = What is this nonsense?



(A) Begin with $f(e) = 0$ for each edge
(B) Find a s-t path P with $f(e) < c(e)$ for every edge $e \in P$
(C) Augment flow along this path
(D) Repeat augmentation for as long as possible.

(A) Greedy can get stuck in sub-optimal flow!
(B) Need to "push-back" flow along edge (u, v).

18.2 Ford-Fulkerson Algorithm

18.2.1 Residual Graph

18.2.1.1 The "leftover" graph

Definition 18.2.1. *For a network $G = (V, E)$ and flow f, the* **residual graph** $G_f = (V', E')$ *of G with respect to f is*

(A) $V' = V$,
(B) **Forward Edges**: *For each edge $e \in E$ with $f(e) < c(e)$, we add $e \in E'$ with capacity $c(e) - f(e)$.*
(C) **Backward Edges**: *For each edge $e = (u, v) \in E$ with $f(e) > 0$, we add $(v, u) \in E'$ with capacity $f(e)$.*

18.2.1.2 Residual Graph Example

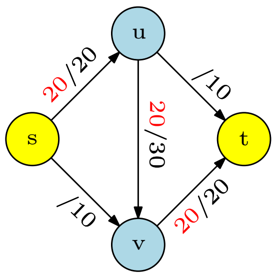


Figure 18.1: Flow on edges is indicated in red

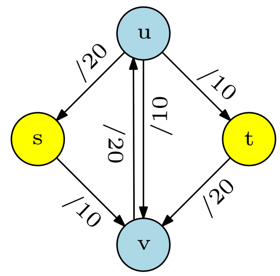


Figure 18.2: Residual Graph

18.2.1.3 Residual capacity

(A) f flow f in network G.
(B) c capacities on the edges.
(C) The ***residual capacity*** is the leftover capacity on each edge. Formally:

$$c_f\Big((u,v)\Big) = \begin{cases} c(u,v) - f(u,v) & (u,v) \in \mathsf{E}(\mathsf{G}) \\ -f(v,u) & (v,u) \in \mathsf{E}(\mathsf{G}) \end{cases}$$

(D) ...assumed that G does not contain both (u,v) and (v,u).
(E) G_f with c_f is a new instance of network flow!

18.2.1.4 Residual graph properties

(A) **Observation:** Residual graph captures the "residual" problem exactly.
(B) Flow in residual graph improves overall flow:

Lemma 18.2.2. *Let f be a flow in G and G_f be the residual graph. If f' is a flow in G_f then $f+f'$ is a flow in G of value $v(f)+v(f')$.*

(C) If there is a bigger flow, we will find it:

Lemma 18.2.3. *Let f and f' be two flows in G with $v(f') \geq v(f)$. Then there is a flow f'' of value $v(f')$-$v(f)$ in G_f.*

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

18.2.1.5 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
```

18.2.1.6 Residual capacity of an augmenting path

(A) f current flow in G_f.
(B) π: A path π in residual graph G_f.
(C) c_f: Residual capacities in G_f.
(D) The ***residual capacity*** of π is

$$c_f(\pi) = \min_{\mathsf{e} \in \mathsf{E}(\pi)} c_f(\mathsf{e}).$$

(E) $c_f(\pi)$ = maximum amount of flow that can be pushed on π in G_f without violating capacities (i.e., c_f).

18.2.1.7 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
```

18.3 Correctness and Analysis

18.3.1 Termination

18.3.1.1 Properties about Augmentation: Flow

Lemma 18.3.1. *If f is a flow and P is a simple s-t path in G_f, then $f' = \mathtt{augment}(f, P)$ is also a flow.*

Proof: Verify that f' is a flow. Let b be augmentation amount.

(A) **Capacity constraint:** If $(u, v) \in P$ is a forward edge then $f'(e) = f(e) + b$ and $b \le c(e) - f(e)$. If $(u, v) \in P$ is a backward edge, then letting $e = (v, u)$, $f'(e) = f(e) - b$ and $b \le f(e)$. Both cases $0 \le f'(e) \le c(e)$.

(B) **Conservation constraint:** Let v be an internal node. Let e_1, e_2 be edges of P incident to v. Four cases based on whether e_1, e_2 are forward or backward edges. Check cases (see fig next slide).

■

18.3.2 Properties of Augmentation

18.3.2.1 Conservation Constraint

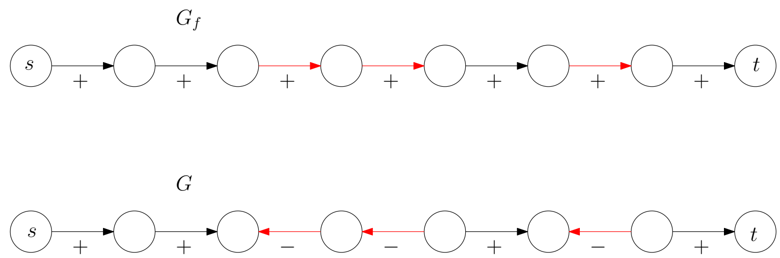


Figure 18.3: Augmenting path P in G_f and corresponding change of flow in G. Red edges are backward edges.

18.3.3 Properties of Augmentation

18.3.3.1 Integer Flow

Lemma 18.3.2. *At every stage of the Ford-Fulkerson algorithm, the flow values on the edges (i.e., $f(e)$, for all edges e) and the residual capacities in G_f are integers.*

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

18.3.3.2 Progress in Ford-Fulkerson

Proposition 18.3.3. *Let f be a flow and f' be flow after one augmentation. Then $v(f) < v(f')$.*

Proof: Let P be an augmenting path, i.e., P is a simple s-t path in residual graph. We have the following.

(A) First edge e in P must leave s.
(B) Original network G has no incoming edges to s; hence e is a forward edge.
(C) P is simple and so never returns to s.
(D) Thus, value of flow increases by the flow on edge e.

■

18.3.3.3 Termination proof for integral flow

Theorem 18.3.4. *Let C be the minimum cut value; in particular $C \leq \sum_{e \text{ out of } s} c(e)$. Ford-Fulkerson algorithm terminates after finding at most C augmenting paths.*

Proof: The value of the flow increases by at least 1 after each augmentation. Maximum value of flow is at most C. ■

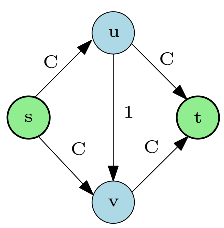
Running time
(A) Number of iterations $\leq C$.
(B) Number of edges in $G_f \leq 2m$.
(C) Time to find augmenting path is $O(n + m)$.
(D) Running time is $O(C(n + m))$ (or $O(mC)$).

18.3.3.4 Efficiency of Ford-Fulkerson

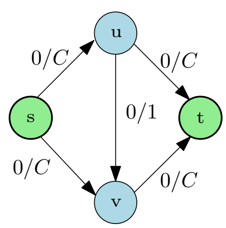
Running time $= O(mC)$ is not polynomial. Can the running time be as $\Omega(mC)$ or is our analysis weak?

18.3.4 Efficiency of Ford-Fulkerson

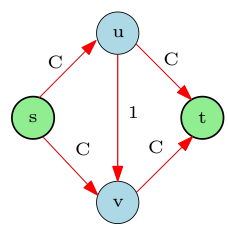
18.3.4.1 Flip-flop 1



Input network + capacities.



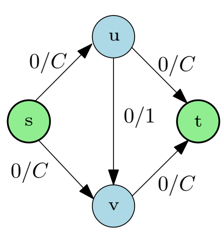
Initial empty flow



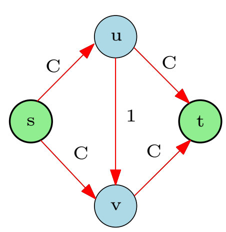
Residual graph (Boring)

18.3.5 Efficiency of Ford-Fulkerson

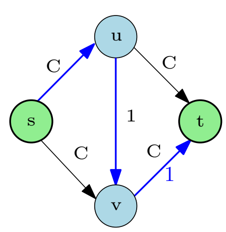
18.3.5.1 Flip-flop 2



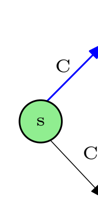
f_0: Initial empty flow



Residual graph



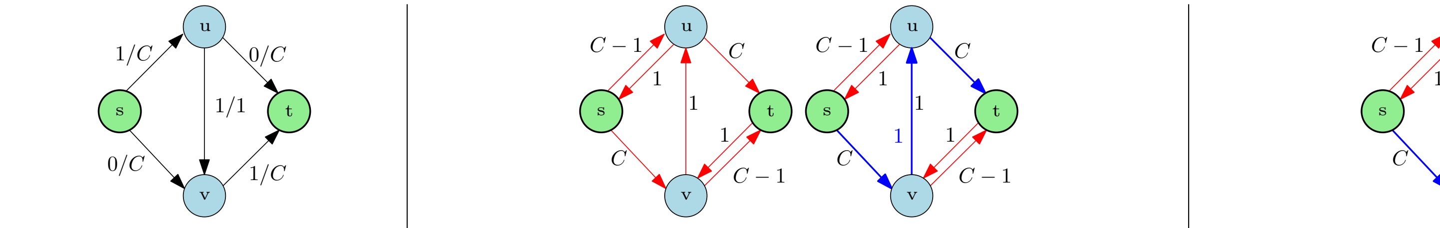
Augmenting path



Augmenting path

18.3.6 Efficiency of Ford-Fulkerson

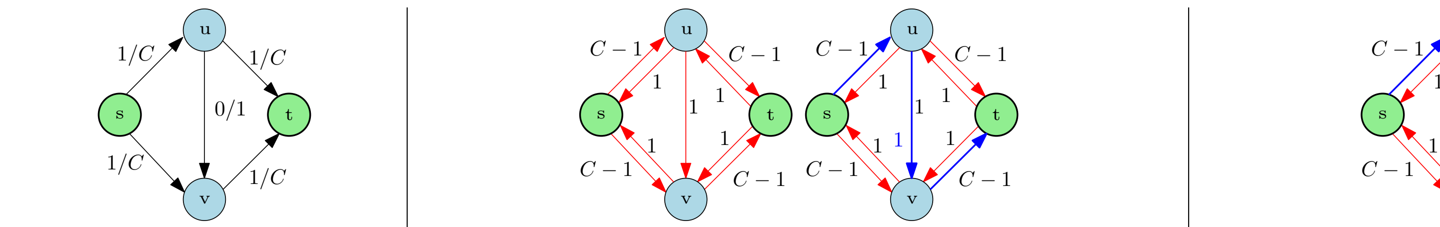
18.3.6.1 Flip-flop 3



f_1: current flow	G_{f_1}: Residual graph	Augmenting path	Augmenting path

18.3.7 Efficiency of Ford-Fulkerson

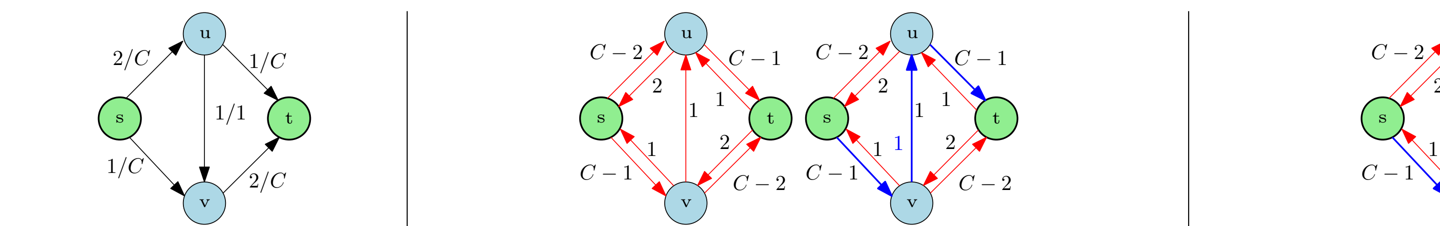
18.3.7.1 Flip-flop 4



f_2: current flow	G_{f_2}: Residual graph	Augmenting path	Augmenting path

18.3.8 Efficiency of Ford-Fulkerson

18.3.8.1 Flip-flop 5



f_3: current flow	G_{f_3}: Residual graph	Augmenting path	Augmenting path

And so it continues for $2C$ iterations...

18.3.8.2 Efficiency of Ford-Fulkerson

(A) Running time $= O(mC)$ is not polynomial.
(B) Can the running time be as $\Omega(mC)$ or is our analysis weak?
(C) Previous example shows this is tight!.
(D) Ford-Fulkerson can take $\Omega(C)$ iterations.

18.3.9 Correctness

18.3.10 Correctness of Ford-Fulkerson

18.3.10.1 Why the augmenting path approach works

(A) **Question:** When the algorithm terminates, is the flow computed the maximum s-t flow?
(B) Proof idea: show a cut of value equal to the flow. Also shows that maximum flow is equal to minimum cut!

18.3.10.2 Recalling Cuts

(A) Definition:

Definition 18.3.5. *Given a flow network an* **s-t cut** *is a set of edges $E' \subset E$ such that removing E'* disconnects *s from t: in other words there is no directed $s \to t$ path in $E - E'$.* **Capacity** *of cut E' is $\sum_{e \in E'} c(e)$.*

(B) ***Vertex cut***: Let $A \subset V$ such that
  (A) $s \in A$, $t \notin A$, and
  (B) $B = V \setminus -A$ and hence $t \in B$.
(C) Define $(A, B) = \{(u, v) \in E \mid u \in A, v \in B\}$

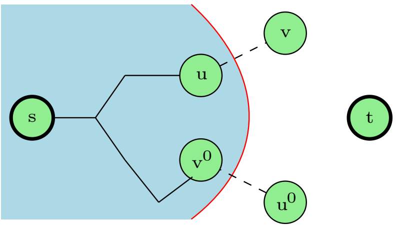
Claim 18.3.6. *(A, B) is an s-t cut.*

(D) Recall: Every *minimal* s-t cut E' is a cut of the form (A, B).

18.3.10.3 Ford-Fulkerson Correctness

Lemma 18.3.7. *If there is no s-t path in G_f then there is some cut (A, B) such that $v(f) = c(A, B)$*

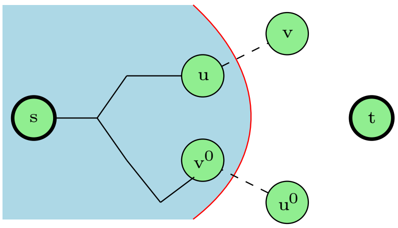
Proof: Let A be all vertices reachable from s in G_f; $B = V \setminus A$.



(A) $s \in A$ a
(B) If $e = ($
$c(e)$ (**sa**
from s

18.3.10.4 Lemma Proof Continued

Proof:

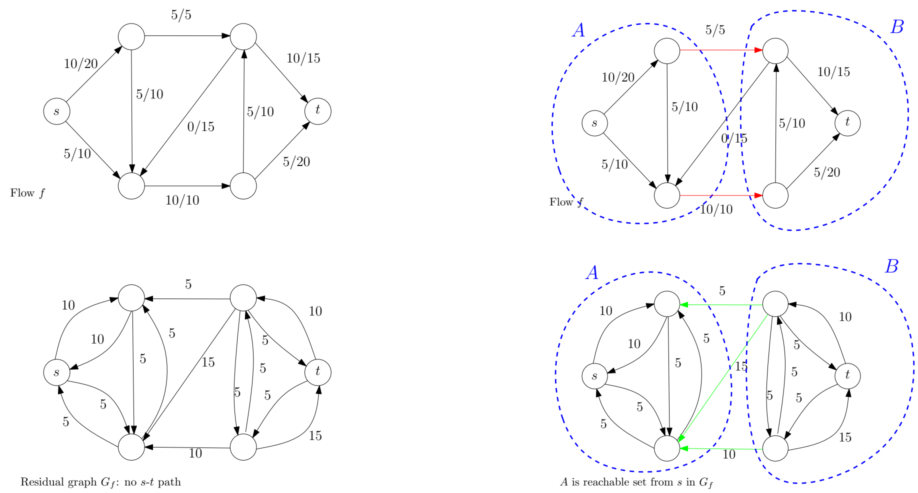


(A) If $e = (u', v') \in G$ with $u' \in B$ and $v' \in A$, then $f(e) = 0$ because otherwise u' is reachable from s in G_f
(B) Thus,

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

■

18.3.10.5 Example



18.3.10.6 Ford-Fulkerson Correctness

Theorem 18.3.8. *The flow returned by the algorithm is the maximum flow.*

Proof:
(A) For any flow f and s-t cut (A, B), $v(f) \le c(A, B)$.
(B) For flow f^* returned by algorithm, $v(f^*) = c(A^*, B^*)$ for some s-t cut (A^*, B^*).
(C) Hence, f^* is maximum.

■

18.3.10.7 Max-Flow Min-Cut Theorem and Integrality of Flows

Theorem 18.3.9. *For any network G, the value of a maximum s-t flow is equal to the capacity of the minimum s-t cut.*

Proof: Ford-Fulkerson algorithm terminates with a maximum flow of value equal to the capacity of a (minimum) cut. ■

18.3.10.8 Max-Flow Min-Cut Theorem and Integrality of Flows

Theorem 18.3.10. *For any network G with integer capacities, there is a maximum s-t flow that is integer valued.*

Proof: Ford-Fulkerson algorithm produces an integer valued flow when capacities are integers. ■

18.4 Polynomial Time Algorithms

18.4.0.9 Efficiency of Ford-Fulkerson

(A) Running time $= O(mC)$ is not polynomial.
(B) Can the upper bound be achieved?
(C) Yes - saw an example.

18.4.0.10 Polynomial Time Algorithms

(A) **Question:** Is there a polynomial time algorithm for max-flow?
(B) **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?
(C) Yes! Two variants.
  (A) Choose the augmenting path with largest bottleneck capacity.
  (B) Choose the shortest augmenting path.

18.4.1 Capacity Scaling Algorithm

18.4.1.1 Augmenting Paths with Large Bottleneck Capacity

(A) Pick augmenting paths with largest bottleneck capacity in each iteration of Ford-Fulkerson.
(B) How do we find path with largest bottleneck capacity?
  (A) Assume we know Δ the bottleneck capacity
  (B) Remove all edges with residual capacity $\leq \Delta$
  (C) Check if there is a path from s to t
  (D) Do binary search to find largest Δ
  (E) Running time: $O(m \log C)$
(C) 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.

18.4.1.2 Augmenting Paths with Large Bottleneck Capacity

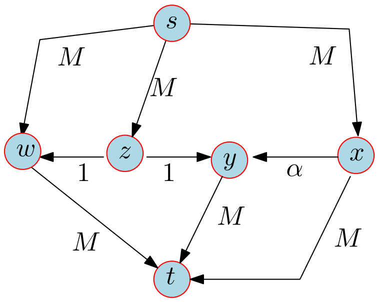
(A) How do we find path with largest bottleneck capacity?
  (A) Max bottleneck capacity is one of the edge capacities. Why?
  (B) Can do binary search on the edge capacities. First, sort the edges by their capacities and then do binary search on that array as before.
  (C) Algorithm's running time is $O(m \log m)$.
  (D) Different algorithm that also leads to $O(m \log m)$ time algorithm by adapting Prim's algorithm.

18.4.1.3 Removing Dependence on C

(A) **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).
(B) Further improvements can yield algorithms running in $O(mn \log n)$, or $O(n^3)$.

18.5 Not for lecture: Non-termination of Ford-Fulkerson

18.5.0.4 Ford-Fulkerson runs in vain



(A) M: large positive integer.
(B) $\alpha = (\sqrt{5} - 1)/2 \approx 0.618$.
(C) $\alpha < 1$,
(D) $1 - \alpha < \alpha$.
(E) Maximum flow in this network is: $2M+1$.

18.5.0.5 Some algebra...

For $\alpha = \dfrac{\sqrt{5}-1}{2}$:

$$\begin{aligned}
\alpha^2 &= ([)]\frac{\sqrt{5}-1}{2}^2 = \frac{1}{4}([)]\sqrt{5}-1^2 = \frac{1}{4}([)]5 - 2\sqrt{5} + 1 \\
&= 1 + \frac{1}{4}([)]2 - 2\sqrt{5} \\
&= 1 + \frac{1}{2}([)]1 - \sqrt{5} \\
&= 1 - \frac{\sqrt{5}-1}{2} \\
&= 1 - \alpha.
\end{aligned}$$

18.5.0.6 Some algebra...

Claim 18.5.1. *Given:* $\alpha = (\sqrt{5}-1)/2$ *and* $\alpha^2 = 1 - \alpha$.

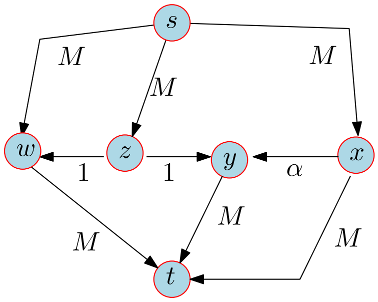
$$\implies \forall i \qquad \alpha^i - \alpha^{i+1} = \alpha^{i+2}$$

Proof:

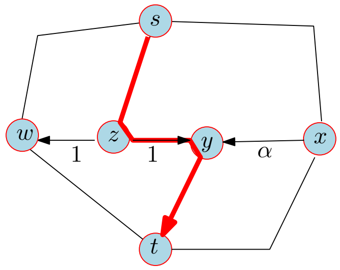
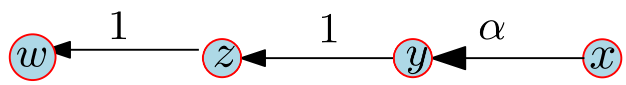
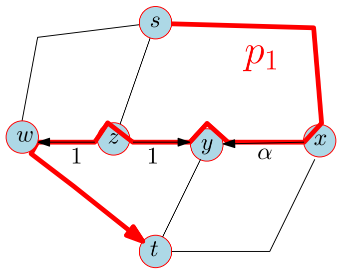
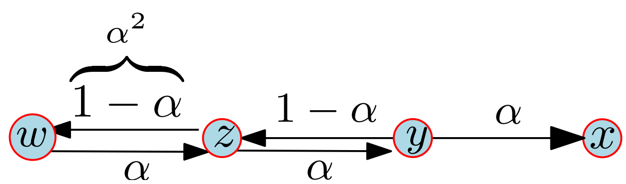
$$\alpha^i - \alpha^{i+1} = \alpha^i(1-\alpha) = \alpha^i \alpha^2 = \alpha^{i+2}.$$

■

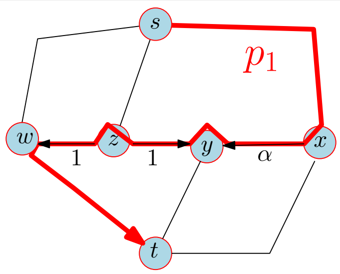
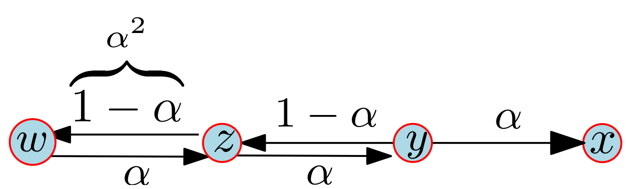
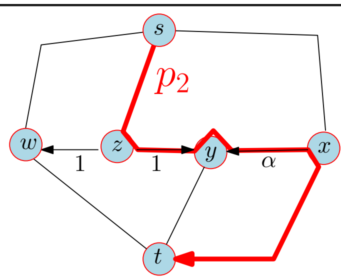
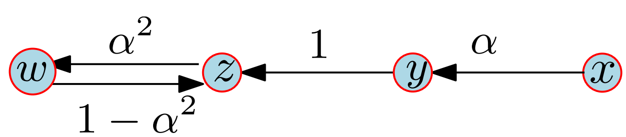
18.5.0.7 The network



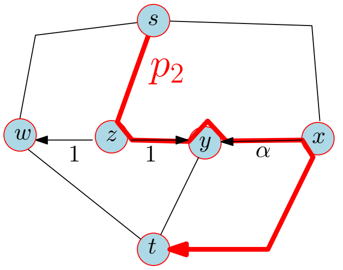
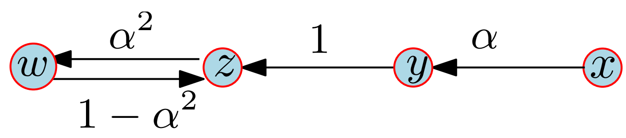
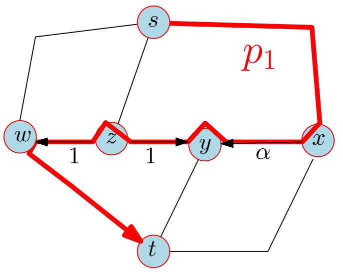
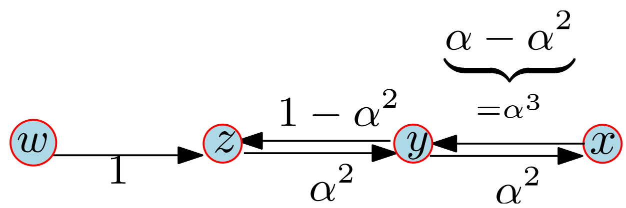
18.5.0.8 Let it flow...

#	Augment. path π	c_π	New residual network
0.		1	
1.		α	

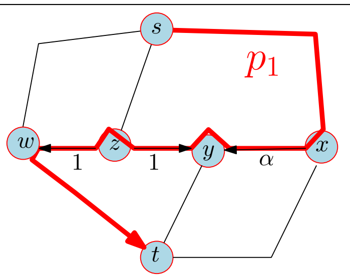
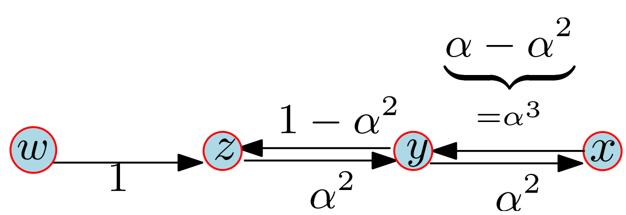
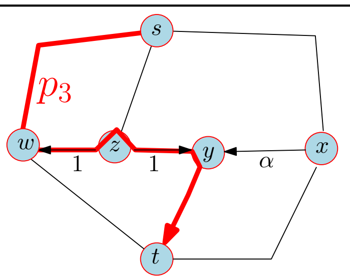
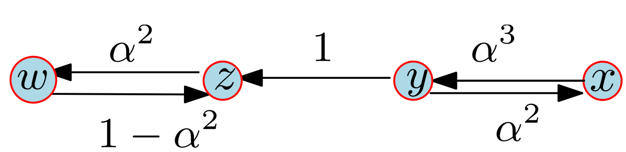
18.5.0.9 Let it flow II

#	Augment. path π	c_π	New residual network
1.		α	
2.		α	

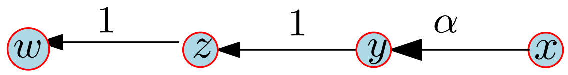
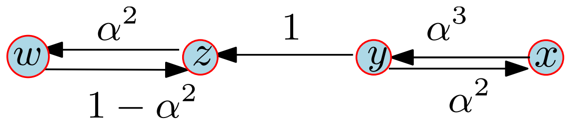
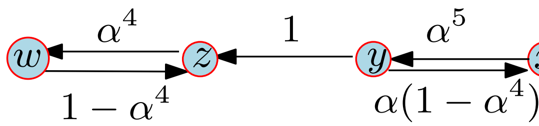
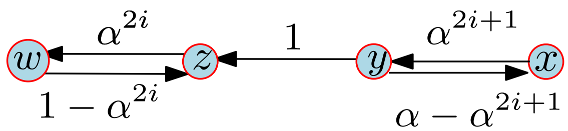
18.5.0.10 Let it flow II

2.		α^2	
3.		α^2	

18.5.0.11 Let it flow III

3.		α^2	
4.		α^2	

18.5.0.12 Let it flow III

<table>
<tr><th>moves</th><th>Residual network after</th></tr>
<tr><td>0</td><td></td></tr>
<tr><td>moves $0, (1, 2, 3, 4)$</td><td></td></tr>
<tr><td>moves $0, (1, 2, 3, 4)^2$</td><td></td></tr>
<tr><td>$0.(1, 2, 3, 4)^i$</td><td></td></tr>
</table>

Namely, the algorithm never terminates.

Bibliography

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

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