Flow Algorithms

Analysis of Algorithms
Week 9, Lecture 2

Prepared by
John Reif, Ph.D.
Distinguished Professor of Computer Science
Duke University
Flow Algorithms

a) Max-flow, min-cut Theorem
b) Augmenting Paths
c) O-1 flow
d) Vertex Connectivity
e) Planar Flow
Readings on Flow Algorithms

- Reading Selection:
  - CLR, Chapter 26
Network Definition

\[
\begin{align*}
\text{digraph} & \quad G = (V,E) \\
\text{distinguished vertices:} & \\
& \quad \text{source} \quad s \in V \\
& \quad \text{sink} \quad t \in V \\
\text{edge capacities:} & \quad c: E \rightarrow \mathbb{R}^+ 
\end{align*}
\]
Reverse Edges

$E^R = \text{reverse of edges in } E$

$(u,v)^R = (v,u) = \text{the reverse of edge } (u,v)$

$u \leftarrow v \quad u \rightarrow v$
Definition of Network Flow

flow $f: (E \cup E^R) \to \mathbb{R}^+$

(1) $f(e) = -f(e^R)$ for all $e \in E$
(2) $f(e) \leq c(e)$
(3) $\sum_{(v,u) \in E} f(v,u) = 0$ for all $v \in V - \{s,t\}$
Value of Flow $f$

- **Flow** $f$

- **Value** \( (f) = \sum_{v \in V} f(s,v) \)
  = sum of flow from source $s$
Min Cut = Max Flow

• \( \text{cut} \)

\[ X, \overline{X} \text{ is partition of } V \]
where \( s \in X, \ t \in \overline{X} \)

\[ f(X, \overline{X}) = \sum_{v \in X, \ u \in \overline{X}} f(v, u) \]
Proof that Min Cut = Max Flow

• **Lemma**

The flow across any cut $X, \overline{X}$ is equal to the value($f$).

• **Proof**

\[ f(X, \overline{X}) = \sum_{u \in X} f(u, v) = \sum_{v \in X} f(v, w) \]

\[ = \sum_{v, w \in X} f(v, w) = \text{value}(f) - 0 = \text{value}(f). \]

Q.E.D.
Residual Flow and Augmenting paths

- **residual capacity** of edge $e$: $res(e) = c(e) - f(e)$
- **residual graph** $R$: use modified capacities $c'(e) = res(e)$ for $res(e) > 0$
- **augmenting path** $p$ for flow $f$ is path in $R$ from $s$ to $t$
- $res(p) = \min_{e \in p} (res(e))$
Characterization of Residual Flow

- **Lemma:** R has max flow value
  value ($f^*$) – value ($f^\prime$), where $f^*$ is the max flow of G.

- **Proof:** If $f^\prime$ is flow in R, then $f + f^\prime$ is flow of G. Also, $f^\prime = f^* - f$ is a flow in R.

Q.E.D.
Flow $f$ on a path

- Flow $f$
  - on path $p = (e_1, e_2, ..., e_k)$
Augmenting Flow

- **residual** \( \text{res}(p) = \Delta = \min_{e \in p} (c(e) - f(e)) \)

- gives Augmented flow \( f + \text{res}(p) \)
Flow $f$ and Augmenting Flow in Residual Network

Network with Flow

Residual Network

Augmenting Path $(s,b,d,a,c,t)$
Min Cut = Max Flow

- **min cut**: cut of minimum capacity
- **max flow**: \( \max (\text{value}(f)) \)

\( f \) is flow
Example of Min Cut = Max Flow

Max Flow = Min Cut = 6

Edges Labeled (Capacity, Flow)
Ford-Fulkerson Proof of Min Cut = Max Flow

Ford - Fulkersons:

Theorem: The max flow $f$ is equal to the min cut $X$, $\overline{X}$.

Proof: (1) If $f$ is max flow, then there can be no augmenting path from $s$ to $t$. Let $X = $ vertices in $V$, reachable from $s$ in residual graph $R$.

$$\overline{X} = V - X$$

$$Value(f) = \sum_{u \in X} f(u,v) = \sum_{u \in X} c(u,v) = c(X, \overline{X})$$

(2) Clearly, $f$ has value at most $c(X, \overline{X})$ for any cut $X$, $\overline{X}$.

Q.E.D.
Bounding Flow augmentations give Max Flow

- **Lemma:** At most $|E|$ flow augmentations are required to construct max flow.
Improving Flow by Augmenting Flow from Residue graph

Proof: Suppose \( f^* \) is max flow in \( G \).

Let \( G^* \) be subgraph of \( G \) with pos. flow.

\[
i \leftarrow 0
\]

\[
\text{while } \exists \text{ path from } s \text{ to } t \text{ do}
\]

\[
i \leftarrow i + 1
\]

find a path \( p_i \) from \( s \) to \( t \) in \( G^* \)

let \( \Delta_i = \min_{e \in p_i} f^*(e) \)

for all \( e \in p \) do

\[
\text{if } f^*(e) \leq 0, \text{ then delete } e \text{ from } G^*
\]

od

od

Note: Deletes at least one edge per step!
Blocking Flow

Definitions given flow $f$

- saturated edge $e$ has $f(e) = c(e)$

- blocking flow $f$: every path from $s$ to $t$ has saturated edge (so cannot augment flow!)

Idea: Re-route flow if it is blocking
Shortest Augmenting paths via level Graph

- **Level Graph** $L$ is subgraph of residual graph $R$
  
  \[
  \text{level} (v) = \text{length of shortest path from } s \text{ to } v \text{ in } R \\
  L \text{ contains only edges } (v,u) \in R \text{ s.t.} \\
  \text{level} (u) = \text{level} (v) + 1
  \]

- **Note**
  $L$ gives shortest augmenting paths
  Construct $L$ in $O(|V| + |E|)$ time by Breadth First Search of $R$
Dinic’s Flow Algorithm

- **Input:** network $G=(V,E)$ s.t. capacities $c_i: (E \cup E^R) \rightarrow \mathbb{R}^+$

- **Initialization:**
  
  for all $e$ assign $f(e) \leftarrow 0$
Dinic’s Flow Algorithm (cont’d)

- Loop:

  [1] Construct level graph \( L \) for \( f \) by Breadth First Search
  [2] By augmentations, find blocking flow \( f' \) in \( L \) from \( f \).
  [3] for all \( e \) assign \( f(e) \leftarrow f(e) + f'(e) \)
  [4] If \( t \) is not in level graph,
      then return \( f \)
      else go to [1]
Proof of Dinic’s Flow Algorithm (cont’d)

• Theorem
  Dinic’s Algorithm halts after $|V|$ blocking steps

• Proof
  Suppose $f$ is flow with
  • $R = \text{residual graph (currently)}$
  • $\text{level } (v) = \text{min length path from } s \text{ to } v \text{ in } R$
  • $R' = \text{new residual graph}$
  • $\text{level}' (v) = \text{min length of path } s \text{ to } v \text{ in } R'$
Proof of Dinic’s Flow Algorithm (cont’d)

- **Claim** \( \text{level}' (t) > \text{level} (t) \)
- **Proof** (by contradiction)
  - If \( \text{level} (t) = \text{level}' (t) \),
  - then \( \text{level} (w) = \text{level} (v)+1 \) for every edge \( (v,w) \in L \).
  - This contradicts the fact that at least one edge is saturated (on the blocking flow) on any path \( p \) in \( L \).

Q.E.D.

Hence \( n \) steps suffice for the algorithm
Example of Dinic’s Flow Algorithm (cont’d)

- Network

[Diagram of a network with nodes labeled a, b, c, d, s, t and edges with capacities 1, 2, 3, 4 as shown in the image.]
Example of Dinic’s Flow Algorithm (cont’d)

- 1st Level Graph with Blocking Flow
Example of Dinic’s Flow Algorithm (cont’d)

- 2nd Level Graph with Blocking Flow
Example of Dinic’s Flow Algorithm (cont’d)

• 3rd Level Graph with Blocking Flow

![Diagam of 3rd Level Graph with Blocking Flow]
Example of Dinic’s Flow Algorithm (cont’d)

- Final Flow
Finding a Blocking Flow

- by Karzanov
- Preflow j:
  1. Satisfies capacities’ constraints
  2. May have unbalanced vertices

\[
\Delta f(u) = \sum_{u \in v} f(u, v) > 0
\]
Finding a Blocking Flow (cont’d)

- Wave method:
  - begin with blocking preflow $f$
    (saturates on edge on every path $s$ to $t$)
  - balance vertices so $\Delta f(v) = 0$ to get blocking flow
Finding a Blocking Flow (cont’ d)

• To balance blocked vertex v:

Repeat (until $\Delta f(v) = 0$) do
  choose edge $(u,v)$ with $f(u,v) > 0$
  decrease $f(u,v)$ by $\min (f(uv), \Delta f(v))$
Finding a Blocking Flow (cont’d)

- To attempt to balance unblocked vertex \( v \):

  Repeat (until \( \Delta f(v) = 0 \), or there is not an unsaturated edge \( (v, w) \) where \( w \) is unblocked).

  do choose some such edge \( (v, w) \) and decrease \( f(v, w) \) by \( \min(C(v, w) - f(v, w), \Delta f(v)) \).
Finding a Blocking Flow (cont’ d)

- Wave Algorithm for Blocking Flow
  Initialize: with preflow that saturates every edge out of $s$ and otherwise 0.
Example of Finding a Blocking Flow (cont’d)

Edges Labeled (Capacity, Flow)

Increased Flow to Blocked vertex d

Decreased Flow Balanced at vertex d

But Blocked vertex c
Example of Finding a Blocking Flow (cont’d)

Decreased Flow
Balanced vertex \( c \)
But Blocked vertex \( a \)

Edges Labeled (Capacity, Flow)
Finding a Blocking Flow (cont’d)

- Set s blocked, and set $V-\{s\}$ all unblocked.  
  Repeat until there are no unbalanced vertices  

  Increase flow: Scan all vertices between $t,s$ in topological order, balancing every vertex $v$ that is unbalanced and unblocked.  (If balancing fails, make $v$ blocked.)

  Decrease flow: Scan vertices in reverse topological order, balancing each vertex that is unbalanced and blocked.
Proof of $O(n^2)$ Time for Finding a Blocking Flow

- **Theorem**: Wave Algorithm computes a blocking flow in $O(n^2)$ time (and hence a max flow in $O(n^3)$ time).

- **Proof** (use invariants):
  1. If $v$ blocked then every path from $v$ to $t$ has saturated edge.
  2. The preflows constructed by algorithm are blocking.
Proof of $O(n^2)$ Time for Finding a Blocking Flow (cont’ d)

- **Modify:** $s$ blocked, and departing edges saturated.

- **Inductive Step:**
  (a) Scanning in topological order in increase flow guarantees no unblocked, unbalanced vertices.
  (b) Scanning in reverse topological order guarantees every blocked vertex gets balanced.
Proof of $O(n^2)$ Time for Finding a Blocking Flow (cont’d)

- **Note:** Each step blocks at least 1 vertex

- So at most $n$ steps flow on edge $e$ increases and decreases at most once

- So total time $O(|V|^2 + |E|) = O(|V|^2) = O(n^2)$
Improved Flow Algorithms

• Can use data structures to decrease blocking flow algorithms to $O(|E| \log |V|)$ time, giving...

• Theorem
  Max flow can be computed in $O(|V||E| \log |V|)$ time.
0-1 Flow Algorithms

• Special Case:

0–1 Flow, if for all \( e \in E \), \( c(e) = 1 \)

• Theorem (Evan and Tarjan)

\[
0 - 1 \text{ Flow requires } \min \left( \left| V \right|^{\frac{2}{3}}, \left| E \right|^{\frac{1}{2}} \right) \text{ blocking steps of Dinic's Algorithm, so total time } O(\min(\left| V \right|^{\frac{2}{3}}, \left| E \right|^{\frac{1}{2}}) \left| E \right| \log(V)).
\]
Unit Flow

- **Unit Network:** All capacities are integers and every vertex $v$ other than $s$ or $t$ has \{single entering edge or single departing edge.\}

- **Claim:** If Unit Network $G$ has max flow $f$, then max level is

\[
\leq \left( \sqrt[\text{value}(f)]{V} \right) + 1
\]

- **Proof:** $G$ can be decomposed into value($f$) vertex-disjoint paths from $s$ to $t$. so \[\text{value}(f) \cdot (\text{level}-1) \leq |V|\]
Illustration of Unit Flow

\[ G \] is decomposed into \textit{value}(f) vertex-disjoint paths from \( s \) to \( t \).
**Theorem:** Dinic’s Algorithm has $O\left(|V|^{\frac{1}{2}}\right)$ steps on unit networks.

**Proof:**

1. If $\text{value}(f) \leq |V|^{\frac{1}{2}} \Rightarrow \# \text{ steps} \leq |V|^{\frac{1}{2}}$

2. If $\text{value}(f) > |V|^{\frac{1}{2}} \Rightarrow \text{level} \leq \frac{|V|}{|V|^{\frac{1}{2}}} + 1,$

so $\# \text{ steps} \leq O\left(|V|^{\frac{1}{2}}\right)$.

**Q.E.D.**

**Total Time Unit Flow is** $O\left(|V|^{\frac{1}{2}}|E|\log|E|\right).$
s-t Vertex Separator

s-t Vertex Separator $S \subseteq V$:
if all paths from $s$ to $t$ contain $v \in S$.

Menger’s Theorem: The size of the smallest $s,t$ Vertex Separator $S$ is exactly the same as the number of vertex disjoint paths from $s$ to $t$. 
Illustration of $s$-$t$ Vertex Separator $S$
Solving Vertex Connectivity via Flow

- Transform **Vertex Connectivity** to **Unit Network Flow Problem**
Time for Solving Vertex Connectivity via Flow

- Total Time $O(|V| |E| \log(E))$ to compute $s$-$t$ Vertex Connectivity $N(s,t)$ (from $s$ to $t$).
- $N(s,t) = \text{number of disjoint paths from } s \text{ to } t$. 
Vertex Connectivity

• \( N(u,v) = \text{min vertex cut size for } (G, u, v) \)
• \( G \) undirected

\[
\text{Vertex Connectivity: } c(G) = \begin{cases} 
  n-1 & \text{if } G \text{ is complete graph} \\
  \min \{ N(u,v) \} & \text{else} \\
  u,v \in V \\
  (u,v) \not\in E
\end{cases}
\]
Bounding Vertex Connectivity

- Lemma
  \[ c(G) \leq \frac{2|E|}{|V|} \]

- Proof
  Connectivity \leq \min_{v \in V} \text{degree } v,
  \[ \sum_{v \in V} \text{degree } (v) = 2|E| \]
  Hence, \( c(G) \leq \frac{2|E|}{|V|} \)

- Q.E.D. (also true for edge connectivity)
Algorithm for Vertex Connectivity

• **Lemma:**

If $S$ is $(u, v)$ Vertex Separator with $|S| = c(G)$, then
\[
c(G) = \min_{(a,b) \notin E} N(a, b) \text{ for all } a \in V - S
\]

• **Proof:** $G - S$ has at least 2 - components
Proof of Algorithm for Vertex Connectivity

• Let $b$ be any node in a component of $G-S$ which does not have $a$.

Thus, $N(a,b) \leq |S| = c(G)$.

Q.E.D.
Idea for Randomized Algorithm for Vertex Connectivity

- **Idea**: Choose at random $a \in V$.

**Time Cost**: $O\left(\log\left(\frac{1}{\varepsilon}\right)|V|^{3/2}|E| \log|E|\right)$
Randomized Algorithm for Vertex Connectivity

• (Melhorn & Students)

Input: \( G = (V,E) \), error bound, \( \varepsilon \), \( 0 < \varepsilon < 1 \)

[0] \( \mu \leftarrow |V| - 2 \)

[1] for \( i=1,2,... \) until \( i \geq \frac{\log \left( \frac{1}{\varepsilon} \right)}{\log \left( \frac{|V|}{\mu} \right)} \)

    do select \( a_i \in V \) at random

    \( \mu \leftarrow \min (\mu, \min_{b \in V} N(a_i, b)) \)

    od

[2] output \( \mu \)
Randomized Algorithm for Vertex Connectivity (cont’d)

- **Theorem:** \( \text{Prob} (\mu \neq c(G)) \leq \varepsilon \)
- **Proof:** Let \( S \) be a Vertex Separator with

\[
|S| = c(G). \text{ If } \mu > c(G), \text{ then } a_1, a_2, \ldots, a_k \text{ all belong to } S, \text{ where} \\

k \geq \log \left( \frac{1}{\varepsilon} \right) / \log\left( \frac{|V|}{c(G)} \right) \\

Hence, \( \text{prob} (\mu > c(G)) = \text{prob} (a_1, \ldots, a_k \in S) \)

\[
= \left( \frac{|S|}{|V|} \right)^k = \left( \frac{c(G)}{|V|} \right)^k = 2^{-\log (\frac{1}{\varepsilon})} = 2^{\log \varepsilon} = \varepsilon.
\]
Definition of Planar Graph

• $G = (V,E)$ is a planar graph if $G$ can be embedded on plane so no two edges cross.
Dual of Planar Graph

- **Dual**: \( D(G) = (F, D(E)) \)
  
  - \( F \) = faces of embedding
  
  - \( D(E) = \{ \{F_i, F_j\} | e \in E \text{ is between } F_i, F_j \} \)
**Lemma:** If $G$ is a planar embedded network, then max flow in $G$ is same as min cost cycle in $D(G)$ separating $s, t$.

**Proof:** We assume $c(F_i, F_j) = c(e)$, if $e$ is between $F_i, F_j$. Then, by min-cost cut theorem, flow value = min cut $X, \bar{X}$ between $s, t$ 

$= \min$ cost cycle in $D(G)$ separating $s, t$
Min Cost Cycle in Dual Graph $D(G)$ separating $s, t$
Definition of Outerplanar Embedded Graph

- G is outerplanar embedded if the planar embedding has face $F_0$ incident to all vertices.
Algorithm for Max Flow in Outerplanar Embedded Graph

- Idea: To reduce to Min Cost Path
  Add new edge \((s,t)\) with weight \(\infty\).
Time Cost of Algorithm for Max Flow in Outerplanar Embedded Graph

Find min cost path from $F_0$ to $F_0'$ in $D(G)$

$= \min s \cdot t$ cut in $G$

$= \max$ flow value in $G$

**Theorem:** If $G$ is outerplanar, we can find max flow in $O(|V|\log|V|)$ time.
Developing an Efficient Algorithm for Max Flow in a General Planar Graph

- **Lemma**: [Reif] If $\mu(s,t)$ is a minimum cost path in $D(G)$ from a face bounding on $s$ to a face bounding on $t$, then any min cost cycle in $D(G)$ separating $s,t$ must contain an edge of $\mu(s,t)$. 
Illustration of $\mu(s,t)$ Path Separating $s$ and $t$ in Dual Graph $D(G)$
Proof of Lemma

• Proof:
  Suppose not. Then we can shortcut any cycle of $D(G)$ separating $s, t$ to get a shorter one, using edges of the $\mu(s, t)$ path.
Efficient Divide and Conquer Algorithm for Max Flow in a General Planar Graph

- **Theorem:** [Reif] The min cost flow in a planar graph can be computed in \( O(|V| \log^2 |V|) \) time.

- **Proof:** Idea: use \( \mu(s,t) \) cut in \( D(G) \) to guide a recursive divided and conquer algorithm. On each step, divide the \( \mu(s,t) \) path in half and solve the problem on each half, separately, using \( s,t \) cut as separator.
Illustration of an Efficient Divide and Conquer Algorithm for Max Flow in a General Planar Graph

Requires $O(\log|V|)$ steps
Each step $O(|V|\log|V|)$ time
Flow Algorithms

Analysis of Algorithms
Week 9, Lecture 2

Prepared by
John Reif, Ph.D.
Distinguished Professor of Computer Science
Duke University