CSE 101, Winter 2016
Design and Analysis of Algorithms
Lecture 12: Network Flows and Cuts (+ Linear Prog)
Class URL: http://vlsicad.ucsd.edu/courses/cse101-w16/
One Big Picture

(various – e.g., Matching)

Network Flow

Linear Programming

more specialized

more efficient

(reduces to)

(reduces to)

boils down to ...

boils down to

Network Flow

more general

[still, "efficient"

in sense of polynomial time complexity in "n"]
Flow Network: Oil Through Pipelines

- Directed graph \( G = (V,E) \)
- Identified source \( S \) and sink \( T \)
- Edge capacities \( c_e \)

How much oil can be shipped from \( S \) to \( T \)?
A Feasible Flow in the Network

- Directed graph $G = (V, E)$
- Identified source $S$ and sink $T$
- Edge capacities $c_e$
- The flow along an edge is $\leq$ capacity

How much oil can be shipped from $S$ to $T$?

5 units of flow – is this the maximum possible?

Yes. (see “cut”)
Many Problems Reduce to Max-Flow (1)

- The UCSD Algorithms Club has N members.
- There are M committees of ASUCSD to which the club can send a representative. (Matching in a bipartite graph)
- Each club member is suited to some subset of committees.
- Can we assign club members so that each committee has a distinct representative? (No one serves on >1 committee.)
Many Problems Reduce to Max-Flow (2)

- Can model multiple sources, sinks
- But: “multi-commodity flow” is difficult to solve – e.g., autos, grain, coal, refrigerators shipped from multiple S’s to multiple T’s over the same railway network

(can solve with L. Programming (fractional LP vs. integer ILP...))
Formal Definition of Flow

- A flow on a graph $G$ is a function $f : E \rightarrow \mathbb{R}$ such that:
  - $0 \leq f(e) \leq c_e$ for all edges $e \in E$
  - Flow into a node = flow out of that node
  - Flow never exceeds capacity
  - Conservation of flow at all nodes except source, sink
  - Amount of flow leaving the source

Size of flow: $\text{size}(f) = \sum_{(s,u) \in E} f(s,u)$

$(2 + 2 + 1) \leq (4 + 1)$
Flows and Cuts

• The size of a flow can be measured across any cut

• A cut \((L, R)\) satisfies:
  – \(V = L \cup R\), with \(L \cap R = \emptyset\) disjoint partition of \(V\)
  – \(S \in L, T \in R\) source is in \(L\), sink is in \(R\)

• Flow across an \((L, R)\) cut:
  \[
  \sum_{(u, w) \in E} f(u, w) - \sum_{(w, z) \in E} f(w, z)
  \]

\(4 - 3 = 1\)
Flows and Cuts

- The size of a flow can be measured across any cut.
- A cut (L, R) satisfies:
  - \( V = L \cup R \), with \( L \cap R = \emptyset \) disjoint partition of \( V \)
  - \( S \in L, T \in R \) source is in L, sink is in R
- Flow across an (L, R) cut: \( \sum_{u \in L, w \in R} f(u, w) - \sum_{w \in R, z \in L} f(w, z) \)

L = \{S\}
R = \{A, B, C, D, E, T\}
\( S - 0 = 5 \)
Observations

• The flow across an \((L,R)\) cut cannot exceed the capacity of the cut

\[
\text{capacity}(L,R) = \sum_{u \in L, w \in R} c(u,w)
\]

• For any flow \(f\) and any cut \(C\), \(\text{size}(f) \leq \text{capacity}(C)\)

  “maximum flow \(\leq\) minimum cut”

• Previous example had \(\text{size}(f) = 5\) and \(\text{capacity}(C) = 5\)
  – The cut is a certificate of optimality (maximality) of the flow
  – The flow is a certificate of optimality (minimality) of the cut
Ford-Fulkerson Algorithm – Basic Idea

• Start with zero flow

REPEAT:
  – Find a path from S to T along which flow can be increased
  – Increase the flow along this path
Motivating Example

- Network with capacities

- First choose:

- Next choose:
Important: Canceling Flow

- If we first choose:
  ![Diagram](image)
  
- Then we **must** allow:
  ![Diagram](image)

---

**Concept of a "residual network"**

**Cancel an existing flow !!!**
Residual Graph

• We have some flow, and want to improve it
• Two ways to “push” or “advance” flow in $G$
  – Find some **unused capacity** on an edge
  – Find some **cancelable flow** on an edge

→ residual graph $G_f$ = “what’s unused or cancelable”

**Flow $f$**

$S \quad A \quad B \quad T$

$0/1 \quad 1/1 \quad 0/1 \quad 1/1$

$S \quad 0/1 \quad B \quad 1/1 \quad T$

**Residual Graph $G_f$**

$S \quad A \quad B \quad T$

$1 \quad 1 \quad 1 \quad 1$

REPEAT: Find an S-T path in $G_f$; Increase $f$ along this path as much as possible
Recipe for Constructing $G_f$

- $G_f = (V, E_f)$
- $E_f \subseteq E \cup E^R$
- For any $(u,w)$ in $E$ or $E^R$
  
  \[
  \text{capacity } c_f(u,w) = c(u,w) - f(u,w) + f(w,u)
  \]
  
- Note 1: Can ignore edges with $c_f(u,w) = 0$
- Note 2: If $(u,w) \not\in E$, write $c(u,w) = 0, f(u,w) = 0$

REPEAT: Find an S-T path in $G_f$;
Increase $f$ along this path as much as possible
Worked Example

• Initial $G_f$

Augment flow along a path

New $G_f$
Worked Example

Final $G_f$

What is the significance of this cut?
Ford-Fulkerson Algorithm – Summary

• Initialize \( f = 0 \)
• REPEAT:
  – Construct the residual graph \( G_f \)
  – Find a path \( P \) from \( S \) to \( T \) in \( G_f \)
  – If there is no such path, HALT
  – \( c_p = \) minimum \( c_f \)-capacity edge on path \( P \)
  – Increase \( f \) by \( c_p \) units along path \( P \)

• Flow always increases \( \rightarrow \) the algorithm terminates
• But, if capacities are \( B \)-bit integers, can take up to \(|E| \cdot 2^B\) iterations in worst case
The Max-Flow Min-Cut Theorem

• **Ford-Fulkerson constructively proves that the maximum flow equals the minimum cut**

• **Define an (L,R) cut as follows:**
  - L = nodes reachable from S in final residual graph G_f
  - R = rest of nodes = V – L  **Note that T cannot be in L → T must be in R**

• **Consider edges between L and R in G**
  - Edges e going from L → R : must have flow = capacity  (by def. of L, R)
  - Edges e’ going from R → L : must have flow = 0  (by def. of L, R)

→ The flow across this cut = Σ_{L→R} edges e c(e) = the capacity of the cut
→ Since any flow is ≤ any cut, we have max flow = min cut
A Taste of Linear Programming
Linear Programming (LP)

- **Tool for optimal allocation of scarce resources**
  - Optimizations subject to “compatibility constraints”

- **Powerful and general problem-solving method**
  - Shortest paths, maximum flows, min-cost flows, MST, matching, 2-person games, …

- **Significance and Practice**
  - Among most important scientific advances of 20th century
  - Dominates industrial practice
    - Delta Airlines: $100M/year benefit from use of LP
  - Commercial solvers (CPLEX, COIN, OSL), modeling languages (AMPL)
  - General tool for attacking intractable (NP-hard) optimization problems
LP Example: Production of Bowls vs. Mugs

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>RESOURCE COSTS</th>
<th>REVENUE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Labor (hr/unit)</td>
<td>Clay (lb/unit)</td>
</tr>
<tr>
<td>Bowl</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Mug</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Constraints: There are 40 hours of labor and 120 pounds of clay available each day.

Decision variables: \( x_1 = \text{number of bowls to produce} \)
\( x_2 = \text{number of mugs to produce} \)

• Note 1: \( x_1, x_2 \) must be non-negative
• Note 2: \( x_1, x_2 \) can be fractional (non-integer)
LP Example: Production of Bowls vs. Mugs

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>Labor (hr/unit)</th>
<th>Clay (lb/unit)</th>
<th>Revenue ($/unit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bowl</td>
<td>1</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>Mug</td>
<td>2</td>
<td>3</td>
<td>50</td>
</tr>
</tbody>
</table>

Constraints: There are 40 hours of labor and 120 pounds of clay available each day

Decision variables: \(x_1 = \) number of bowls to produce \(x_2 = \) number of mugs to produce

Maximize \[ Z = 40x_1 + 50x_2 \]

Subject to
\[ x_1 + 2x_2 \leq 40 \text{ hr (labor constraint)} \]
\[ 4x_1 + 3x_2 \leq 120 \text{ lb (clay constraint)} \]
\[ x_1, x_2 \geq 0 \]

Example solution: \(x_1 = 24\) bowls \(x_2 = 8\) mugs
Revenue = $1,360
Geometric Interpretation

The feasible region is the area common to both constraints:

\[ 4x_1 + 3x_2 \leq 120 \text{ lb} \]

\[ x_1 + 2x_2 \leq 40 \text{ hr} \]
Feasible Region Has Extreme Points

• The **feasible region** is an **intersection of half-spaces** that arise from the constraints
  – Vertices of the feasible region = where 2 constraints are tight
  – Vertices are like “corners”

![Diagram showing the feasible region with vertices A, B, and C, and their corresponding values for \( x_1 \) and \( x_2 \).]
Solution of Simultaneous Equations

\[
\begin{align*}
4x_1 + 3x_2 &\leq 120 \text{ lb} \\
4x_1 + 2x_2 &\leq 40 \text{ hr}
\end{align*}
\]

\[
\begin{align*}
x_1 + 2x_2 &= 40 \\
4x_1 + 3x_2 &= 120 \\
4x_1 + 8x_2 &= 160 \\
-4x_1 - 3x_2 &= -120 \\
5x_2 &= 40
\end{align*}
\]

\[
x_2 = 8
\]

\[
x_1 + 2(8) = 40 \\
x_1 = 24
\]

\[
Z = $50(24) + $50(8) = $1,360
\]
Geometry: Concave, Convex, and Linear

- Inequalities induce *halfspaces* with respect to *hyperplanes*
- Bounded feasible region: convex *polygon* or *polytope*
- Feasible region = convex set: If a, b feasible, so is (a+b)/2
- Extreme point: Feasible solution x that cannot be written as (a+b)/2 for two distinct feasible solutions a and b

![Diagram showing convex and non-convex regions with extreme points](image-url)
Geometry: Convex Sets, Convex Functions

- Inequalities induce *halfspaces* with respect to *hyperplanes*
- Bounded feasible region: convex *polygon* or *polytope*
- Feasible region = convex set: If a, b feasible, so is (a+b)/2
- Extreme point: Feasible solution x that cannot be written as (a+b)/2 for two distinct feasible solutions a and b

!![Diagram showing convex and non-convex polygons]

- Function f is convex if $f[\lambda x_1 + (1-\lambda)x_2] \leq \lambda f(x_1) + (1-\lambda)f(x_2)$
  - f defined over a convex domain
  - If f is convex, then a local minimum is a global minimum
Geometry: Concave, Convex, and Linear

• **Function f is convex** if $f[\lambda x_1 + (1-\lambda)x_2] \leq \lambda f(x_1) + (1-\lambda)f(x_2)$
  - $f$ defined over a convex domain
  - If $f$ is convex, then a local minimum is a global minimum

• **Function f is concave** if $-f$ is convex
  - If $f$ is concave, then a local minimum can occur only at an extreme point of the domain of $f$

• **KEY OBSERVATION:** A linear function is both convex and concave (!!!)
  - Local minima occur only at extreme points (from concavity)
  - Any local minimum is a global minimum (from convexity)
  - $\rightarrow$ To find a global minimum, only need to look at extreme points