Lower bounds

1. The following example requires the ability to solve network flow with lower bounds on the edges.
2. This can be reduced to regular network flow (we are not going to show the details – they are a bit tedious).
3. The integrality property holds – if there is an integral solution our network flow with lower bounds solver would compute such a solution.

Airline Scheduling

Problem

Given information about flights that an airline needs to provide, generate a profitable schedule.

- Input: detailed information about “legs” of flight.
- \( F \): set of flights by
- Purpose: find minimum \# airplanes needed.
Example
(i) a set \( F \) of flights that have to be served, and (ii) the corresponding graph \( G \) representing these flights.

1: Boston (depart 6 A.M.) - Washington DC (arrive 7 A.M.)
2: Urbana (depart 7 A.M.) - Champaign (arrive 8 A.M.)
3: Washington (depart 8 A.M.) - Los Angeles (arrive 11 A.M.)
4: Urbana (depart 11 A.M.) - San Francisco (arrive 2 P.M.)
5: San Francisco (depart 2:15 P.M.) - Seattle (arrive 3:15 P.M.)
6: Las Vegas (depart 5 P.M.) - Seattle (arrive 6 P.M.)

Flight scheduling...

- Use same airplane for two segments \( i \) and \( j \):
  (a) destination of \( i \) is the origin of the segment \( j \).
  (b) there is enough time in between the two flights.
- Also, airplane can fly from dest(\( i \)) to origin(\( j \)) (assuming time constraints are satisfied).

Example
As a concrete example, consider the flights:
Boston (depart 6 A.M.) - Washington D.C. (arrive 7 A.M.)
Washington (depart 8 A.M.) - Los Angeles (arrive 11 A.M.)
Las Vegas (depart 5 P.M.) - Seattle (arrive 6 P.M.)
This schedule can be served by a single airplane by adding the leg “Los Angeles (depart 12 noon)- Las Vegas (1 P.M.)” to this schedule.

Modeling the problem

- model the feasibility constraints by a graph.
- \( G \): directed graph over flight legs.
- For \( i \) and \( j \) (legs), \((i \to j) \in E(G) \iff \text{same airplane can serve both } i \text{ and } j.\)
- \( G \) is acyclic.
- \( Q \): Can required legs can be served using only \( k \) airplanes?

Solution

- Reduction to computation of circulation.
- Build graph \( H \).
- \( \forall \text{ leg } i \), two new vertices \( u_i, v_i \in VH. \)
  \( s \): source vertex. \( t \): sink vertex.
- Set demand at \( t \) to \( k \), Demand at \( s \) to be \( -k \).
- Flight must be served: New edge \( e_i = (u_i \to v_i) \), for leg \( i \).
  Also \( \ell(e_i) = 1 \) and \( c(e_i) = 1 \).
- If same plane can so \( i \) and \( j \) (i.e., \((i \to j) \in E(G)\)) then add edge \((v_i \to u_j)\) with capacity 1 to \( H \).
- Since any airplane can start the day with flight \( i \): add an edge \((s \to u_i)\) with capacity 1 to \( H \), \( \forall i \).
- Add edge \((v_j \to t)\) with capacity 1 to \( G \), \( \forall j \).
- Overflow airplanes: “overflow” edge \((s \to t)\) with capacity \( k \).
  Let \( H \) denote the resulting graph.
Example of resulting graph

The resulting graph $H$ for the instance of airline scheduling show before.

Lemma

Lemma

$\exists$ way perform all flights of $\mathcal{F} \leq k$ planes $\iff \exists$ circulation in $H.$

Proof.

- Given feasible solution $\rightarrow$ translate into valid circulation.
- Given feasible circulation...
- ... extract paths from flow.
- ... every path is a plane.

Extensions and limitations

- a lot of other considerations:
  - (i) airplanes have to undergo long term maintenance treatments every once in awhile,
  - (ii) one needs to allocate crew to these flights,
  - (iii) schedule differ between days, and
  - (iv) ultimately we interested in maximizing revenue.

- Network flow is used in practice, real world problems are complicated, and network flow can capture only a few aspects.
- ... a good starting point.

Pennant Race
Pennant Race: Example

Can Boston win the pennant?
No, because Boston can win at most 91 games.

Another Example

Can Boston win the pennant?
Not clear unless we know what the remaining games are!

Refining the Example

Can Boston win the pennant? Suppose Boston does
- Boston wins both its games to get 92 wins
- New York must lose both games; now both Baltimore and Toronto have at least 92
- Winner of Baltimore-Toronto game has 93 wins!

Abstracting the Problem

Given
- A set of teams \( S \)
- For each \( x \in S \), the current number of wins \( w_x \)
- For any \( x, y \in S \), the number of remaining games \( g_{xy} \) between \( x \) and \( y \)
- A team \( z \)

Can \( z \) win the pennant?
Towards a Reduction

\( \mathcal{Z} \) can win the pennant if

1. \( \mathcal{Z} \) wins at least \( m \) games
   - to maximize \( \mathcal{Z} \)'s chances we make \( \mathcal{Z} \) win all its remaining games and hence \( m = w_z + \sum_{x \in S} g_{xz} \)
2. no other team wins more than \( m \) games
   - for each \( x, y \in S \) the \( g_{xy} \) games between them have to be assigned to either \( x \) or \( y \).
3. each team \( x \neq \mathcal{Z} \) can win at most \( m - w_x - g_{xz} \) remaining games

Is there an assignment of remaining games to teams such that no team \( x \neq \mathcal{Z} \) wins more than \( m - w_x \) games?

Flow Network: The basic gadget

- \( s \): source
- \( t \): sink
- \( x, y \): two teams
- \( g_{xy} \): number of games remaining between \( x \) and \( y \).
- \( w_x \): number of points \( x \) has.
- \( m \): maximum number of points \( x \) can win before team of interest is eliminated.

Flow Network: An Example

Can Boston win?

<table>
<thead>
<tr>
<th>Team</th>
<th>Won</th>
<th>Left</th>
<th>NY</th>
<th>Bal</th>
<th>Tor</th>
<th>Bos</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>90</td>
<td>11</td>
<td>-</td>
<td>1</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Baltimore</td>
<td>88</td>
<td>6</td>
<td>1</td>
<td>-</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Toronto</td>
<td>87</td>
<td>11</td>
<td>6</td>
<td>1</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>Boston</td>
<td>79</td>
<td>12</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>-</td>
</tr>
</tbody>
</table>

\( m = 79 + 12 = 91 \): Boston can get at most 91 points.

Constructing Flow Network

Notations

- \( S \): set of teams,
- \( w_x \): wins for each team, and
- \( g_{xy} \) games left between \( x \) and \( y \).
- \( m \) be the maximum number of wins for \( \mathcal{Z} \), and \( S' = S \setminus \{ \mathcal{Z} \} \).

Reduction

Construct the flow network \( G \) as follows

- One vertex \( v_x \) for each team \( x \in S' \), one vertex \( u_{xy} \) for each pair of teams \( x \) and \( y \) in \( S' \)
- A new source vertex \( s \) and sink \( t \)
- Edges \((u_{xy}, v_x)\) and \((u_{xy}, v_y)\) of capacity \( \infty \)
- Edges \((s, u_{xy})\) of capacity \( g_{xy} \)
- Edges \((v_x, t)\) of capacity equal \( m - w_x \)
Correctness of reduction

Theorem

$G'$ has a maximum flow of value $g^* = \sum_{x,y \in S'} g_{xy}$ if and only if $z$ can win the most number of games (including possibly tie with other teams).

Proof of Correctness

Proof.

Existence of $g^*$ flow $\Rightarrow$ $z$ wins pennant

1. An integral flow saturating edges out of $s$, ensures that each remaining game between $x$ and $y$ is added to win total of either $x$ or $y$
2. Capacity on $(v_x, t)$ edges ensures that no team wins more than $m$ games

Conversely, $z$ wins pennant $\Rightarrow$ flow of value $g^*$

1. Scenario determines flow on edges; if $x$ wins $k$ of the games against $y$, then flow on $(u_{xy}, v_x)$ edge is $k$ and on $(u_{xy}, v_y)$ edge is $g_{xy} - k$

Proof that $z$ cannot win the pennant

1. Suppose $z$ cannot win the pennant since $g^* < g$. How do we prove to some one compactly that $z$ cannot win the pennant?
2. Show them the min-cut in the reduction flow network!

Project Scheduling

Problem:

1. $n$ projects/tasks $1, 2, \ldots, n$
2. dependencies between projects: $i$ depends on $j$ implies $i$ cannot be done unless $j$ is done. dependency graph is acyclic
3. each project $i$ has a cost/profit $p_i$
   - $p_i < 0$ implies $i$ requires a cost of $-p_i$ units
   - $p_i > 0$ implies that $i$ generates $p_i$ profit

Goal: Find projects to do so as to maximize profit.
Project selection example

Notation

For a set $A$ of projects:

- $A$ is a valid solution if $A$ is dependency closed, that is for every $i \in A$, all projects that $i$ depends on are also in $A$.
- $\text{profit}(A) = \sum_{i \in A} p_i$. Can be negative or positive.

Goal: find valid $A$ to maximize $\text{profit}(A)$.

Idea: Reduction to Minimum-Cut

Finding a set of projects is partitioning the projects into two sets: those that are done and those that are not done.

Can we express this is a minimum cut problem?

Several issues:

- We are interested in maximizing profit but we can solve minimum cuts.
- We need to convert negative profits into positive capacities.
- Need to ensure that chosen projects is a valid set.
- The cut value captures the profit of the chosen set of projects.

Reduction to Minimum-Cut

Note: We are reducing a maximization problem to a minimization problem.

- projects represented as nodes in a graph
- if $i$ depends on $j$ then $(i, j)$ is an edge
- add source $s$ and sink $t$
- for each $i$ with $p_i > 0$ add edge $(s, i)$ with capacity $p_i$
- for each $i$ with $p_i < 0$ add edge $(i, t)$ with capacity $-p_i$
- for each dependency edge $(i, j)$ put capacity $\infty$ (more on this later)
Reduction: Flow Network Example

Reduction contd

Algorithm:
1. form graph as in previous slide
2. compute \( s-t \) minimum cut \((A, B)\)
3. output the projects in \( A - \{s\} \)

Understanding the Reduction

Let \( C = \sum_{i: p_i > 0} p_i \): maximum possible profit.

Observation: The minimum \( s-t \) cut value is \( \leq C \). Why?

Lemma
Suppose \((A, B)\) is an \( s-t \) cut of finite capacity (no \( \infty \) edges. Then projects in \( A - \{s\} \) are a valid solution.

Proof.
If \( A - \{s\} \) is not a valid solution then there is a project \( i \in A \) and a project \( j \notin A \) such that \( i \) depends on \( j \).

Since \((i, j)\) capacity is \( \infty \), implies \((A, B)\) capacity is \( \infty \), contradicting assumption.
Reduction: Flow Network Example

Correctness of Reduction

Recall that for a set of projects $X$, $\text{profit}(X) = \sum_{i \in X} p_i$.

Lemma

Suppose $(A, B)$ is an $s$-$t$ cut of finite capacity (no $\infty$) edges. Then $c(A, B) = C - \text{profit}(A - \{s\})$.

Proof.

Edges in $(A, B)$:
- $(s, i)$ for $i \in B$ and $p_i > 0$: capacity is $p_i$
- $(i, t)$ for $i \in A$ and $p_i < 0$: capacity is $-p_i$
- cannot have $\infty$ edges

Proof contd

For project set $A$ let
- $\text{cost}(A) = \sum_{i \in A, p_i < 0} -p_i$
- $\text{benefit}(A) = \sum_{i \in A, p_i > 0} p_i$
- $\text{profit}(A) = \text{benefit}(A) - \text{cost}(A)$.

Proof.

Let $A' = A \cup \{s\}$.

$$c(A', B) = \text{cost}(A) + \text{benefit}(B) = \text{cost}(A) - \text{benefit}(A) + \text{benefit}(A) + \text{benefit}(B) = -\text{profit}(A) + C = C - \text{profit}(A)$$

Correctness of Reduction contd

We have shown that if $(A, B)$ is an $s$-$t$ cut in $G$ with finite capacity then
- $A - \{s\}$ is a valid set of projects
- $c(A, B) = C - \text{profit}(A - \{s\})$

Therefore a minimum $s$-$t$ cut $(A^*, B^*)$ gives a maximum profit set of projects $A^* - \{s\}$ since $C$ is fixed.

Question: How can we use $\infty$ in a real algorithm?

Set capacity of $\infty$ arcs to $C + 1$ instead. Why does this work?
Lower Bounds and Costs

Two generalizations:
1. flow satisfies $f(e) \leq c(e)$ for all $e$. suppose we are given lower bounds $\ell(e)$ for each $e$. can we find a flow such that $\ell(e) \leq f(e) \leq c(e)$ for all $e$?
2. suppose we are given a cost $w(e)$ for each edge. cost of routing flow $f(e)$ on edge $e$ is $w(e)f(e)$. can we (efficiently) find a flow (of at least some given quantity) at minimum cost?

Many applications.

Flows with Lower Bounds

Definition
A flow in a network $G = (V, E)$, is a function $f : E \to \mathbb{R}^\geq 0$ such that
- Capacity Constraint: For each edge $e$, $f(e) \leq c(e)$
- Lower Bound Constraint: For each edge $e$, $f(e) \geq \ell(e)$
- Conservation Constraint: For each vertex $v$
  \[ \sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e) \]

Question: Given $G$ and $c(e)$ and $\ell(e)$ for each $e$, is there a flow? As difficult as finding an $s$-$t$ maximum-flow without lower bounds!

Regular flow via lower bounds

Given usual flow network $G$ with source $s$ and sink $t$, create lower-bound flow network $G'$ as follows:
- set $\ell(e) = 0$ for each $e$ in $G$
- add new edge $(t, s)$ with lower bound $\nu$ and upper bound $\infty$

Claim
There exists a flow of value $\nu$ from $s$ to $t$ in $G$ if and only if there exists a feasible flow with lower bounds in $G'$.

Above reduction show that lower bounds on flows are naturally related to circulations. With lower bounds, cannot guarantee acyclic flows from $s$ to $t$.

Flows with Lower Bounds

- Flows with lower bounds can be reduced to standard maximum flow problem. See text book. Reduction goes via circulations.
- If all bounds are integers then there is a flow that is integral. Useful in applications.
Survey Design
Application of Flows with Lower Bounds

- Design survey to find information about $n_1$ products from $n_2$ customers.
- Can ask customer questions only about products purchased in the past.
- Customer can only be asked about at most $c'_i$ products and at least $c_i$ products.
- For each product need to ask at least $p_i$ consumers and at most $p'_i$ consumers.

Minimum Cost Flows

- **Input:** Given a flow network $G$ and also edge costs, $w(e)$ for edge $e$, and a flow requirement $F$.
- **Goal:** Find a minimum cost flow of value $F$ from $s$ to $t$.

Given flow $f: E \rightarrow R^+$, cost of flow $= \sum_{e \in E} w(e)f(e)$.

Minimum Cost Flow: Facts

- problem can be solved efficiently in polynomial time
  - $O(nm \log C \log(nW))$ time algorithm where $C$ is maximum edge capacity and $W$ is maximum edge cost
  - $O(m \log n(m + n \log n))$ time strongly polynomial time algorithm
- for integer capacities there is always an optimum solutions in which flow is integral
How much damage can a single path cause?

Consider the following network. All the edges have capacity 1. Clearly the maximum flow in this network has value 4.

The network

Why removing the shortest path might ruin everything
- However... The shortest path between $s$ and $t$ is the blue path.
- And if we remove the shortest path, $s$ and $t$ become disconnected, and the maximum flow drop to 0.