Part I

Baseball Pennant Race
Pennant Race: Example

### Example

<table>
<thead>
<tr>
<th>Team</th>
<th>Won</th>
<th>Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>92</td>
<td>2</td>
</tr>
<tr>
<td>Baltimore</td>
<td>91</td>
<td>3</td>
</tr>
<tr>
<td>Toronto</td>
<td>91</td>
<td>3</td>
</tr>
<tr>
<td>Boston</td>
<td>89</td>
<td>2</td>
</tr>
</tbody>
</table>

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

Example

<table>
<thead>
<tr>
<th>Team</th>
<th>Won</th>
<th>Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>92</td>
<td>2</td>
</tr>
<tr>
<td>Baltimore</td>
<td>91</td>
<td>3</td>
</tr>
<tr>
<td>Toronto</td>
<td>91</td>
<td>3</td>
</tr>
<tr>
<td>Boston</td>
<td>90</td>
<td>2</td>
</tr>
</tbody>
</table>

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

Refining the Example

Example

<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>92</td>
<td>2</td>
<td>—</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Baltimore</td>
<td>91</td>
<td>3</td>
<td>1</td>
<td>—</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Toronto</td>
<td>91</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>1</td>
</tr>
<tr>
<td>Boston</td>
<td>90</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>—</td>
</tr>
</tbody>
</table>

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

\( z \) can win the pennant if
- \( z \) wins at least \( m \) games
  - to maximize \( z \)'s chances we make \( z \) win all its remaining games and hence \( m = w_z + \sum_{x \in S} g_{xz} \)
- 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 \( x \) and \( y \).
  - each team \( x \neq z \) can win at most \( m - w_x - g_{xz} \) remaining games

Is there an assignment of remaining games to teams such that no team \( x \neq z \) wins more than \( m - w_x - g_{xz} \) 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.

\[ s \rightarrow g_{xy} \rightarrow u_{xy} \rightarrow t \]

\( v_x \rightarrow m - w_x \) and \( v_y \rightarrow m - w_y \)

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

Reduction
Let \( S \) be the set of teams, \( w_x \) wins for each team, and \( g_{xy} \) games left between \( x \) and \( y \). Let \( m \) be the maximum number of wins for \( z \), and \( S' = S \setminus \{z\} \). 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 $\implies$ $z$ wins pennant
- 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 $y$
- Capacity on $(v_x, t)$ edges ensures that no team wins more than $m$ games

Conversely, $z$ wins pennant $\implies$ flow of value $g^*$
- 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 $g_{xy} - k$

---

**Proof that $z$ cannot win the pennant**

Suppose $z$ cannot win the pennant since $g^* < g$. How do we prove to some one *compactly* that $z$ cannot win the pennant?

Show them the min-cut in the reduction flow network!

See text book for a natural interpretation of the min-cut as a certificate.
Part II

An Application of Min-Cut to Project Scheduling

Project Scheduling

Problem:
- **n** projects/tasks 1, 2, . . ., n
- *dependencies* between projects: i depends on j implies i cannot be done unless j is done. dependency graph is *acyclic*
- 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.
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

Algorithm:
- form graph as in previous slide
- compute s-t minimum cut \((A, B)\)
- 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 $j \not\in A$ such that $i$ depends on $j$.

Since $(i, j)$ capacity is $\infty$, implies $(A, B)$ capacity is $\infty$, contradicting assumption.

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.

$$c(A, B) = \text{cost}(A) + \text{benefit}(B)$$

$$= \text{cost}(A) - \text{benefit}(A) + \text{benefit}(A) + \text{benefit}(B)$$

$$= -\text{profit}(A) + C$$

$$= C - \text{profit}(A)$$

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?
Part III

Extensions to Maximum-Flow Problem

Lower Bounds and Costs

Two generalizations:

- 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 $e$?

- 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 $f : E \rightarrow \mathbb{R}_{\geq 0}$ 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 $v$ and upper bound $\infty$

**Claim**: there exists a flow of value $v$ 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 east \( p_i \) consumers and at most \( p_i' \) consumers
Reduction to Circulation

- include edge \((i, j)\) is customer \(i\) has bought product \(j\)
- Add edge \((t, s)\) with lower bound \(0\) and upper bound \(\infty\).
- Consumer \(i\) is asked about product \(j\) if the integral flow on edge \((i, j)\) is 1

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 \(f : E \rightarrow \mathbb{R}^+\), 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