Airline Scheduling

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

- Input: detailed information about “legs” of flight.
- \( \mathcal{F} \): set of flights by
- Purpose: find minimum \# airplanes needed.

**Example**
(i) a set \( \mathcal{F} \) 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 \( \text{dest}(i) \) to \( \text{origin}(j) \) (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** → **j**) ∈ **E(G)** ⟷ same airplane can serve both **i** 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**.
- ∀ leg **i**, two new vertices **u**, **v** ∈ **V_H**.
  - **s**: source vertex. **t**: sink vertex.
- Set demand at **t** to **k**, Demand at **s** to be −**k**.
- Each flight must be served. New edge **e_i** = (**u**, **v**) for leg **i**.
  - Also ℓ(**e_i**) = 1 and **c**(**e_i**) = 1.
- If same plane can so **i** and **j** (i.e., (**i** → **j**) ∈ **E(G)**) then add edge (**v** → **u**) with capacity 1 to **H**.
- Since any airplane can start the day with flight **i**: add an edge (**s** → **u**) with capacity 1 to **H**, ∀**i**.
- Add edge (**v** → **t**) with capacity 1 to **G**, ∀**j**.
- Overflow airplanes: “overflow” edge (**s** → **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.
Extensions and limitations

- a lot of other considerations:
  1. airplanes have to undergo long term maintenance treatments every once in awhile,
  2. one needs to allocate crew to these flights,
  3. schedule differ between days, and
  4. 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.

Image Segmentation

Input is an image.
Partition image into background and foreground.

(i) (ii)

The (i) input image, and (ii) a possible segmentation of the image.

What is the input...

- Input is a bitmap on a grid.
- Every grid node represents a pixel
- Convert grid into a directed graph $G$.

Input:
1. $N \times N$ bitmap. $G = (V, E)$.
2. $\forall$ pixel $i$: foreground value $f_i \geq 0$.
3. $\forall$ pixel $i$: background value $b_i$.
4. $\forall i, j$ adjacent: separation penalty $p_{ij}$. (we assume that $p_{ij} = p_{ji}$)
Problem statement

Problem

Given input as above, partition \( V \) (the set of pixels) into two disjoint subsets \( F \) and \( B \), such that

\[
q(F, B) = \sum_{i \in F} f_i + \sum_{j \in B} b_j - \sum_{(i,j) \in E, |F \cap \{i,j\}| = 1} p_{ij}.
\]

is maximized.

Rewrite \( q(F, B) \) as:

\[
q(F, B) = \sum_{i \in F} f_i + \sum_{j \in B} b_j - \sum_{(i,j) \in E, |F \cap \{i,j\}| = 1} p_{ij}
= \sum_{i \in v} (f_i + b_i) - \left( \sum_{i \in B} f_i + \sum_{j \in F} b_j + \sum_{(i,j) \in E, |F \cap \{i,j\}| = 1} p_{ij} \right).
\]

Solution continued...

1. Compute a minimum cut in a graph. Price = \( u(F, B) \).
2. A toy example:
   - \( \{s, f_i\} \rightarrow \{t\} \)
   - \( \{s\} \rightarrow \{i, t\} \)
   - Two possible cuts in the graph:
     - (i) \( \{s, i\}, \{t\} \): price \( b_i \).
     - (ii) \( \{s\}, \{i, t\} \): price \( f_i \).
   - In particular,
     - Every path of length 2 from \( s \) to \( t \) forces mincut to choose one of edges.
     - Mincut “prefers” the edge with lower price.

Solution continued...

1. Two pixel bitmap:
2. Captures background/foreground prices. But... ignores separation penalties...
3. Price of cut in graph is corresponding value of \( u(F, B) \).
4. mincut-cut in the resulting graph would corresponds to the required segmentation.
Recap...

- Given directed grid graph $G = (V, E)$.
- $s, y$: add two special source and sink vertices.
- $\forall i \in V$: add edge $e_i = (s \rightarrow i)$.
- $c(e_i) = f_i$.
- Add $e'_i = (j \rightarrow t)$ with capacity $c(e'_i) = b_i$.
- $\forall i, j$ adjacent:
  assign the capacity $p_{ij}$ to the edges $(i \rightarrow j)$ and $(j \rightarrow i)$

$H$: resulting graph.

Solution continues...

By the above discussion:

**Lemma**

A minimum cut $(F, B)$ in $H$ minimizes $u(F, B)$.

Using the minimum-cut max-flow theorem, we have:

**Theorem**

One can solve the segmentation problem, in polynomial time, by computing the max flow in the graph $H$.

---

Part II

Projection selection

**Project Selection**

- company which can carry out some projects.
- $P$: set of possible projects.
- $\forall i \in P$: a **revenue** $p_i$.
- $p_i > 0$ is a profitable project and $p_i < 0$ is a losing project.
- There is dependency between projects.
- $G = (P, E)$: $(i \rightarrow j) \in E$ if and only if $j$ is a prerequisite for $i$. 
Definition

A set \( X \subset P \) is **feasible** if for all \( i \in X \), all the prerequisites of \( i \) are also in \( X \). Formally, for all \( i \in X \), with an edge \( (i \to j) \in E \), we have \( j \in X \).

The **profit** associated with a set of projects \( X \subseteq P \) is 
\[
\text{profit}(X) = \sum_{i \in X} p_i.
\]

Problem - Project Selection Problem

Select a feasible set of projects maximizing the overall profit.

The reduction

- Use mincut again.
- Add \( s \) and \( t \) to \( G \).
- Perform the following modifications:
  - \( \forall i \in P \) with \( p_i > 0 \): add edge \( e_i = (s \to i) \).
    with \( c(e_i) = p_i \).
  - \( \forall j \in P \) with \( p_j < 0 \): add edge \( e'_j = (j \to t) \).
    Set \( c(e'_j) = -p_j \).
  - \( C = \sum_{i \in P, p_i > 0} p_i \): upper bound on profit.
- Set capacity of all original (dependency) edges in \( G \) to \( 4C \).

Let \( H \) denote the resulting network.
Solution continued

- \( X \subseteq P \): Set of feasible projects.
- \( X' = X \cup \{s\} \) and \( Y' = (P \setminus X) \cup \{t\} \).
- Consider the \( s-t \) cut \((X', Y')\) in \( H \).
- No \( E(G) \) is in \((X', Y')\) since \( X \) is a feasible set.

\[ X' = X \cup \{s\} \text{ and } Y' = (P \setminus X) \cup \{t\}. \]

\[ \text{Consider the } s-t \text{ cut } (X', Y') \text{ in } H. \]

\[ \text{No } E(G) \text{ is in } (X', Y') \text{ since } X \text{ is a feasible set.} \]

Lemma

\[ c(X', Y') = C - \sum_{i \in X} p_i = C - \text{profit}(X). \]

Proof

- The edges of \( H \) are either:
  (i) original edges of \( G \),
  (ii) emanating from \( s \), and
  (iii) edges entering \( t \).
- \( X \) feasible \( \implies \) no edges of type (i) in cut.
- Edges entering \( t \) contribute:
  \[ \beta = \sum_{i \in X \text{ and } p_i < 0} -p_i. \]

Proof continued

Edges leaving \( s \) contribute:

\[
\gamma = \sum_{i \in X \text{ and } p_i > 0} p_i = \sum_{i \in P \setminus X \text{ and } p_i > 0} p_i - \sum_{i \in X \text{ and } p_i > 0} p_i
\]

\[
= C \sum_{i \in X \text{ and } p_i > 0} p_i,
\]

by the definition of \( C \). The capacity of the cut \((X', Y')\) is

\[
\beta + \gamma = \sum_{i \in X \text{ and } p_i < 0} (-p_i) + \left( C - \sum_{i \in X \text{ and } p_i > 0} p_i \right)
\]

\[
= C - \sum_{i \in X} p_i = C - \text{profit}(X),
\]

Lemma

If \((X', Y')\) is a cut with capacity at most \( C \) in \( G \), then the set \( X = X' \setminus \{s\} \) is a feasible set of projects.

Namely, cuts \((X', Y')\) of capacity \( \leq C \) in \( H \) corresponds one-to-one to feasible sets which are profitable.

Proof.

Since \( c(X', Y') \leq C \) it must not cut any of the edges of \( G \), since the price of such an edge is \( 4C \). As such, \( X \) must be a feasible set.
Putting everything together, we are looking for a feasible set $X$ that maximizes $\text{profit}(X) = \sum_{i \in X} p_i$. This corresponds to a set $X' = X \cup \{s\}$ of vertices in $H$ that minimizes $C - \sum_{i \in X} p_i$, which is also the cut capacity $(X', Y')$. Thus, computing a minimum-cut in $H$ corresponds to computing the most profitable feasible set of projects.

Result

**Theorem**

If $(X', Y')$ is a minimum cut in $H$ then $X = X' \setminus \{s\}$ is an optimum solution to the project selection problem. In particular, using network flow the optimal solution can be computed in polynomial time.

**Proof.**

Indeed, we use network flow to compute the minimum cut in the resulting graph $H$. Note, that it is quite possible that the most profitable project is still a net loss.
Pennant Race: 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

<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

<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>1</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>1</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>1</td>
<td>0</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

1. \( Z \) wins at least \( m \) games
2. to maximize \( Z \)'s chances we make \( Z \) win all its remaining games
   and hence \( m = w_z + \sum_{x \in S} g_{xz} \)
3. 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 \( y \).
4. 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 \( m - w_x \) 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>1</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>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Toronto</td>
<td>87</td>
<td>11</td>
<td>6</td>
<td>1</td>
<td>4</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>4</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 \( Z \),
  and \( S' = S \setminus \{ Z \} \).

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**

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

---

Proof of Correctness

**Proof.**

Existence of \( g^* \) flow \( \Rightarrow \) \( z \) wins pennant

- 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 \( \Rightarrow \) 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 is \( g_{xy} - k \)

---

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.

---

A compact proof of a team being eliminated

Interestingly, once \( z \) is eliminated, we can generate a compact proof of this fact.
Helper claim

Claim

For any two teams $x$ and $y$ for which the vertex $u_{xy}$ exists, we have that $u_{xy} \in S$ if and only if both $x$ and $y$ are in $\hat{S}$.

Proof...

$(x \notin \hat{S} \text{ or } y \notin \hat{S}) \implies u_{xy} \notin S$ If $x$ is not in $\hat{S}$ then $v_x$ is in $T$. But then, if $u_{xy}$ is in $S$ the edge $(u_{xy} \to v_x)$ is in the cut. However, this edge has infinite capacity, which implies this cut is not a minimum cut (in particular, $(S, T)$ is a cut with capacity smaller than $\alpha$). As such, in such a case $u_{xy}$ must be in $T$. This implies that if either $x$ or $y$ are not in $\hat{S}$ then it must be that $u_{xy} \in T$. (And as such $u_{xy} \notin S$.)

Theorem

Suppose that team $z$ has been eliminated. Then there exists a “proof” of this fact of the following form:

- The team $z$ can finish with at most $m$ wins.
- There is a set of teams $\hat{S} \subset S$ so that
  \[ \sum_{s \in \hat{S}} w_s + \sum_{\{x, y\} \subseteq \hat{S}} g_{xy} > m |\hat{S}|. \]

(And hence one of the teams in $\hat{S}$ must end with strictly more than $m$ wins.)
Proof

If \( z \) is eliminated then the max flow in \( G \) has value \( \gamma \), which is smaller than \( \alpha \). By the minimum-cut max-flow theorem, there exists a minimum cut \((S, T)\) of capacity \( \gamma \) in \( G \), and let \( S = \{ x \mid v_x \in S \} \). The above argumentation implies that edges of the type \((u_{xy} \rightarrow v_x)\) can not be in the cut \((S, T)\). As such, there are two type of edges in the cut \((S, T)\): (i) \((v_x \rightarrow t)\), for \( x \in S \), and (ii) \((s \rightarrow u_{xy})\) where at least one of \( x \) or \( y \) is not in \( S \). As such, the capacity of the cut \((S, T)\) is

\[
c(S, T) = \sum_{x \in S} (m - w_x) + \sum_{\{x,y\} \subseteq \bar{S}} g_{xy}
\]

\[
= m |S| - \sum_{x \in S} w_x + \left( \alpha - \sum_{\{x,y\} \subseteq \bar{S}} g_{xy} \right).
\]

However, \( c(S, T) = \gamma < \alpha \), and it follows that

\[
m |S| - \sum_{x \in S} w_x - \sum_{\{x,y\} \subseteq \bar{S}} g_{xy} < \alpha - \alpha = 0.
\]

Namely,

\[
\sum_{x \in S} w_x + \sum_{\{x,y\} \subseteq \bar{S}} g_{xy} > m |S|,
\]

as claimed. ■

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.

**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.