COL351: Analysis and Design of Algorithms

Ragesh Jaiswal, CSE, IITD

Applications of Network Flow

- Suppose there are four teams in IPL with their current number of wins:
  - Daredevils: 10
  - Sunrisers: 10
  - Lions: 10
  - Supergiants: 8
- There are 7 more games to be played. These are as follows:
  - Supergiants plays all other 3 teams.
  - Daredevils Vs Sunrisers, Sunrisers Vs Lions, Daredevils Vs Lions, Sunrisers Vs Daredevils

- Suppose there are four teams in IPL with their current number of wins:
  - Daredevils: 10
  - Sunrisers: 10
  - Lions: 10
  - Supergiants: 8
- There are 7 more games to be played. These are as follows:
  - Supergiants plays all other 3 teams.
  - Daredevils Vs Sunrisers, Sunrisers Vs Lions, Daredevils Vs Lions, Sunrisers Vs Daredevils
- A team is said to be eliminated if it cannot end with maximum number of wins.
- Can we say that Supergiants have been eliminated give the current scenario?

- Suppose there are four teams in IPL with their current number of wins:
  - Daredevils: 10
  - Sunrisers: 10
  - Lions: 9
  - Supergiants: 8
- There are 7 more games to be played. These are as follows:
  - Supergiants plays all other 3 teams.
  - 4 games between Daredevils and Sunrisers.
- Can we say that Supergiants have been eliminated give the current scenario?

Problem

There are n teams. Each team i has a current number of wins denoted by $w(i)$.There are $G(i, j)$ games yet to be played between team i and j. Design an algorithm to determine whether a given team x has been eliminated.

Problem

There are n teams. Each team i has a current number of wins denoted by $w(i)$.There are $G(i, j)$ games yet to be played between team i and j. Design an algorithm to determine whether a given team x has been eliminated.

- Consider the following flow network

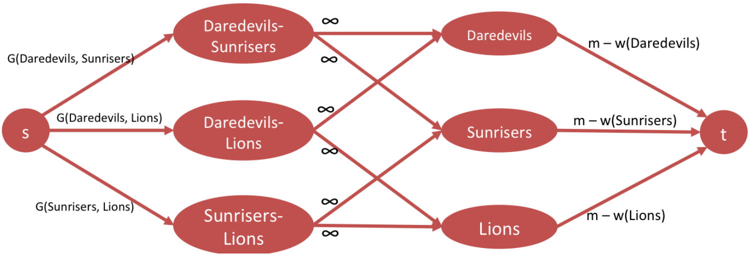


Figure: Team x can end with at most m wins, i.e., $m = w(x) + \sum_j G(x, j)$

Problem

There are n teams. Each team i has a current number of wins denoted by $w(i)$.There are $G(i,j)$ games yet to be played between team i and j. Design an algorithm to determine whether a given team x has been eliminated.

- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.

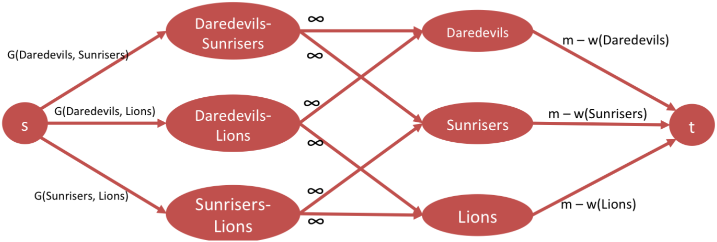
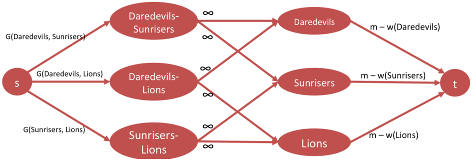


Figure: Team x can end with at most m wins, i.e., $m = w(x) + \sum_j G(x,j)$

Problem

There are n teams. Each team i has a current number of wins denoted by $w(i)$.There are $G(i,j)$ games yet to be played between team i and j. Design an algorithm to determine whether a given team x has been eliminated.

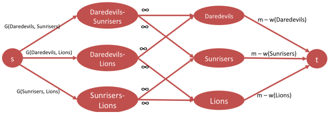
- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.
- Comment: If we can somehow find a subset T of teams (not including x) such that $\sum_{i \in T} w(i) + \sum_{i<j \text{ and } i,j \in T} G(i,j) > m \cdot |T|$. Then we have a witness to the fact that x has been eliminated.



Problem

There are n teams. Each team i has a current number of wins denoted by $w(i)$.There are $G(i,j)$ games yet to be played between team i and j. Design an algorithm to determine whether a given team x has been eliminated.

- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.
- Comment: If we can somehow find a subset T of teams (not including x) such that $\sum_{i \in T} w(i) + \sum_{i<j \text{ and } i,j \in T} G(i,j) > m \cdot |T|$. Then we have a witness to the fact that x has been eliminated.
- Can we find such a subset T?

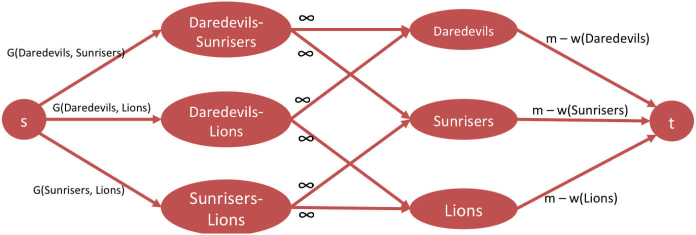


- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.

Proof.

- Claim 1.1: If x has been eliminated, then the max flow in the network is $< g^*$.

$\square$



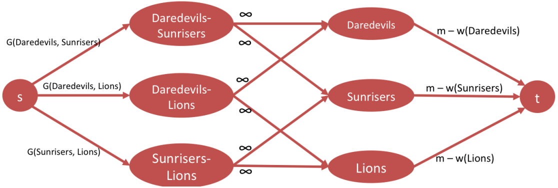
- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.

Proof of Claim 1

- Claim 1.1: If x has been eliminated, then the max flow in the network is $< g^*$.
- Claim 1.2: If the max flow is $< g^*$, then team x has been eliminated.

Proof of Claim 1.2

- Consider any s-t min-cut (A, B) in the graph.
- Claim 1.2.1: If v_{ij} is in A, then both v_i and v_j are in A.



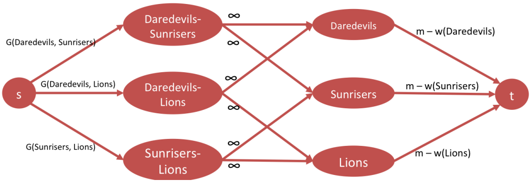
- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.

Proof of Claim 1

- Claim 1.1: If x has been eliminated, then the max flow in the network is $< g^*$.
- Claim 1.2: If the max flow is $< g^*$, then team x has been eliminated.

Proof of Claim 1.2

- Consider any s-t min-cut (A, B) in the graph.
- Claim 1.2.1: If v_{ij} is in A, then both v_i and v_j are in A.
- Claim 1.2.2: If both v_i and v_j are in A, then v_{ij} is in A.



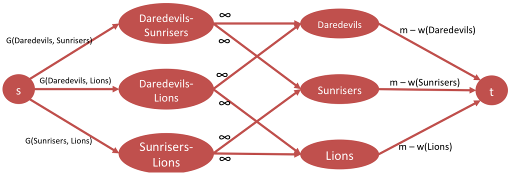
- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.

Proof of Claim 1

- Claim 1.1: If x has been eliminated, then the max flow in the network is $< g^*$.
- Claim 1.2: If the max flow is $< g^*$, then team x has been eliminated.

Proof of Claim 1.2

- Consider any s-t min-cut (A, B) in the graph.
- Claim 1.2.1: If v_{ij} is in A, then both v_i and v_j are in A.
- Claim 1.2.2: If both v_i and v_j are in A, then v_{ij} is in A.
- Let T be the set of teams such that $i \in T$ **iff** $v_i \in A$.



- Claim 1: Team x has been eliminated **iff** the maximum flow in the network is $< g^*$, where $g^* = \sum_{i,j \text{ s.t. } x \notin \{i,j\}} G(i,j)$.

Proof of Claim 1

- Claim 1.1: If x has been eliminated, then the max flow in the network is $< g^*$.
- Claim 1.2: If the max flow is $< g^*$, then team x has been eliminated.

Proof of Claim 1.2

- Consider any s-t min-cut (A, B) in the graph.
- Claim 1.2.1: If v_{ij} is in A, then both v_i and v_j are in A.
- Claim 1.2.2: If both v_i and v_j are in A, then v_{ij} is in A.
- Let T be the set of teams such that $i \in T$ **iff** $v_i \in A$. Then we have:

$$C(A,B) = \sum_{i \in T}(m - w(i)) + \sum_{\{i,j\} \not\subset T} G(i,j) < g^*$$

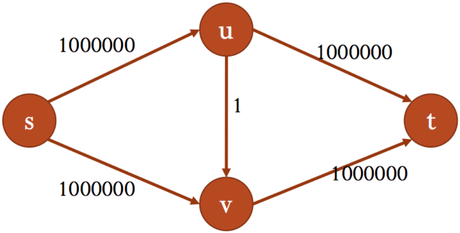
$$\Rightarrow \quad m \cdot |T| - \sum_{i \in T} w(i) + (g^* - \sum_{\{i,j\} \subset T} G(i,j)) < g^*$$

$$\Rightarrow \quad \sum_{i \in T} w(i) + \sum_{\{i,j\} \subset T} G(i,j) > m \cdot |T| \quad \square$$

$\square$

Max flow revisited: Scaling max flow

- Let $C = \sum_{e \text{ out of } s} c(e)$.
- The running time of the `Ford-Fulkerson` algorithm is $O(m \cdot C)$.
- C could be very large compared to the size of the graph.
  - For the example below, we might get a better running time if we could hide the edge with small capacity when looking for an augmenting path.
- General idea: Use all edges with large capacities before considering edges with smaller capacity.



- For an s-t flow and a positive integer Δ, let $G_f(\Delta)$ denote the subgraph of the residual graph G_f that consists of all vertices but only edges with residual capacity of at least Δ.
- Idea: Instead of finding augmenting paths in G_f, we will find augmenting paths in $G_f(\Delta)$ for smaller and smaller values of Δ.

Algorithm

`Scaling-Max-Flow`

- Start with an s-t flow such that for all e, $f(e) = 0$
- $\Delta \leftarrow$ largest power of 2 smaller than C
- While $(\Delta \geq 1)$
  - While there is an s-t path P in $G_f(\Delta)$
    - Augment flow along an augmenting path and let f' be the resulting flow
    - Update f to f' and $G_f(\Delta)$ to $G_{f'}(\Delta)$
  - $\Delta \leftarrow \Delta/2$
- return(f)

Algorithm

`Scaling-Max-Flow`

- Start with an $s-t$ flow such that for all e, $f(e)=0$
- $\Delta \leftarrow$ largest power of 2 smaller than C
- While $(\Delta \geq 1)$
  - While there is an $s-t$ path P in $G_f(\Delta)$
    - Augment flow along an augmenting path and let f' be the resulting flow
    - Update f to f' and $G_f(\Delta)$ to $G_{f'}(\Delta)$
  - $\Delta \leftarrow \Delta/2$
- return(f)

- Claim 1: The algorithm returns max. flow on termination.

Algorithm

`Scaling-Max-Flow`

- Start with an $s-t$ flow such that for all e, $f(e)=0$
- $\Delta \leftarrow$ largest power of 2 smaller than C
- While $(\Delta \geq 1)$
  - While there is an $s-t$ path P in $G_f(\Delta)$
    - Augment flow along an augmenting path and let f' be the resulting flow
    - Update f to f' and $G_f(\Delta)$ to $G_{f'}(\Delta)$
  - $\Delta \leftarrow \Delta/2$
- return(f)

- Claim 1: The algorithm returns max. flow on termination.
- Claim 2: The outer while loop runs for at most $(1+\lceil \log C \rceil)$ steps.

Algorithm

`Scaling-Max-Flow`

- Start with an $s-t$ flow such that for all e, $f(e) = 0$
- $\Delta \leftarrow$ largest power of 2 smaller than C
- While $(\Delta \geq 1)$
  - While there is an $s-t$ path P in $G_f(\Delta)$
    - Augment flow along an augmenting path and let f' be the resulting flow
    - Update f to f' and $G_f(\Delta)$ to $G_{f'}(\Delta)$
  - $\Delta \leftarrow \Delta/2$
- return(f)

- Claim 1: The algorithm returns max. flow on termination.
- Claim 2: The outer while loop runs for at most $(1 + \lceil \log C \rceil)$ steps.
- Claim 3: Each augmentation increases the flow by at least Δ (whatever the current value of Δ is).

Algorithm

`Scaling-Max-Flow`

- Start with an $s-t$ flow such that for all e, $f(e) = 0$
- $\Delta \leftarrow$ largest power of 2 smaller than C
- While $(\Delta \geq 1)$
  - While there is an $s-t$ path P in $G_f(\Delta)$
    - Augment flow along an augmenting path and let f' be the resulting flow
    - Update f to f' and $G_f(\Delta)$ to $G_{f'}(\Delta)$
  - $\Delta \leftarrow \Delta/2$
- return(f)

- Claim 1: The algorithm returns max. flow on termination.
- Claim 2: The outer while loop runs for at most $(1 + \lceil \log C \rceil)$ steps.
- Claim 3: Each augmentation increases the flow by at least Δ (whatever the current value of Δ is).
- Claim 4: Let f be the flow at the end of a Δ-*scaling* phase. Then there is an $s-t$ cut (A, B) such that $c(A, B) \leq v(f) + m \cdot \Delta$.
  - Corollary: The max flow in the graph has value at most $v(f) + m \cdot \Delta$.

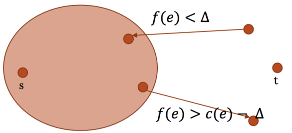
- Claim 4: Let f be the flow at the end of a Δ-*scaling* phase. Then there is an $s-t$ cut (A, B) such that $c(A, B) \leq v(f) + m \cdot \Delta$.
  - Corollary: The max flow in the graph has value at most $v(f) + m \cdot \Delta$.

Proof of Claim 4.

Let A be the set of vertices that are reachable from s in $G_f(\Delta)$ (see figure below). Then we have

$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) \\ &\geq \sum_{e \text{ out of } A} (c(e) - \Delta) - \sum_{e \text{ into } A} \Delta \\ &\geq c(A, B) - m \cdot \Delta. \end{aligned}$$

□



A (all vertices reachable from s in $G_f(\Delta)$.

Algorithm

`Scaling-Max-Flow`

- Start with an $s-t$ flow such that for all e, $f(e) = 0$
- $\Delta \leftarrow$ largest power of 2 smaller than C
- While $(\Delta \geq 1)$
  - While there is an $s-t$ path P in $G_f(\Delta)$
    - Augment flow along an augmenting path and let f' be the resulting flow
    - Update f to f' and $G_f(\Delta)$ to $G_{f'}(\Delta)$
  - $\Delta \leftarrow \Delta/2$
- return(f)

- Claim 1: The algorithm returns max. flow on termination.
- Claim 2: The outer while loop runs for at most $(1 + \lceil \log C \rceil)$ steps.
- Claim 3: Each augmentation increases the flow by at least Δ (whatever the current value of Δ is).
- Claim 4: Let f be the flow at the end of a Δ-*scaling* phase. Then there is an $s-t$ cut (A, B) such that $c(A, B) \leq v(f) + m \cdot \Delta$.
  - Corollary: The max flow in the graph has value at most $v(f) + m \cdot \Delta$.
- Claim 5: The total number of iterations of the inner while loop is at most $2m$.

Algorithm

`Scaling-Max-Flow`
- Start with an $s-t$ flow such that for all e, $f(e) = 0$
- $\Delta \leftarrow$ largest power of 2 smaller than C
- While $(\Delta \geq 1)$
  - While there is an $s-t$ path P in $G_f(\Delta)$
    - Augment flow along an augmenting path and let f' be the resulting flow
    - Update f to f' and $G_f(\Delta)$ to $G_{f'}(\Delta)$
  - $\Delta \leftarrow \Delta/2$
- return(f)

- Claim 1: The algorithm returns max. flow on termination.
- Claim 2: The outer while loop runs for at most $(1 + \lceil \log C \rceil)$ steps.
- Claim 3: Each augmentation increases the flow by at least Δ (whatever the current value of Δ is).
- Claim 4: Let f be the flow at the end of a Δ-*scaling* phase. Then there is an $s-t$ cut (A, B) such that $c(A, B) \leq v(f) + m \cdot \Delta$.
  - Corollary: The max flow in the graph has value at most $v(f) + m \cdot \Delta$.
- Claim 5: The total number of iterations of the inner while loop is at most $2m$.
- Claim 6: The running time of `Scaling-Max-Flow` algorithm is $O(m^2 \cdot \log C)$.

More applications

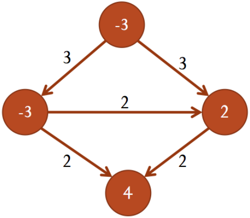
- Given a weighted directed graph representing a transportation network.
- There are multiple supply nodes in the graph denoting the places that has a factory for some product.
- There are multiple demand nodes denoting the consumption points.
- Each supply node v has an associated supply value $s(v)$ denoting the amount the product it can supply.
- Each demand node v has a similar demand value $d(v)$.
- Question: Is there a way to ship product such that all demand and supply goals are met?

Problem

Given a directed graph G with integer edge capacities. For each node v, there is an associated demand value $t(v)$ denoting the demand at the node (*for supply nodes this is* $-s(v)$, *for demand nodes* $d(v)$, *for other nodes* 0). Find whether there exists a flow f such that for all nodes v:

$$f^{in}(v) - f^{out}(v) = t(v)$$

and the capacity constraints are met. Such a flow is called a feasible circulation.



Problem

Given a directed graph G with integer edge capacities. For each node v, there is an associated demand value $t(v)$ denoting the demand at the node (*for supply nodes this is* $-s(v)$, *for demand nodes* $d(v)$, *for other nodes* 0). Find whether there exists a flow f such that for all nodes v:

$$f^{in}(v) - f^{out}(v) = t(v)$$

and the capacity constraints are met. Such a flow is called a feasible circulation.

- Claim 1: For a feasible circulation to exist, $\sum_v t(v) = 0$. (*That means supply equals the demand*)

- Claim 1: For a feasible circulation to exist, $\sum_v t(v) = 0$. (*That means supply equals the demand*)
- Consider the flow network as shown in the diagram below and let $D = \sum_{\text{demand node } v} d(v)$.

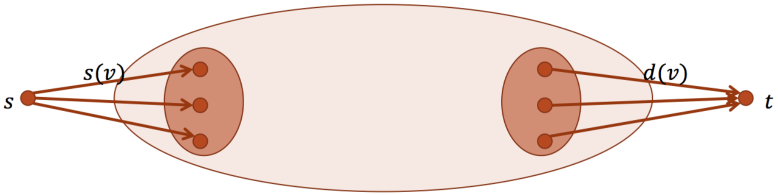


Figure: Connect source to supply nodes and demand nodes to sink.

- <u>Claim 1</u>: For a feasible circulation to exist, $\sum_v t(v) = 0$. (*That means supply equals the demand*)
- Consider the flow network G' as shown in the diagram below and let $D = \sum_{\text{demand node } v} d(v)$.
- <u>Claim 2</u>: There is a feasible circulation in G iff the maximum flow in the network G' is D.

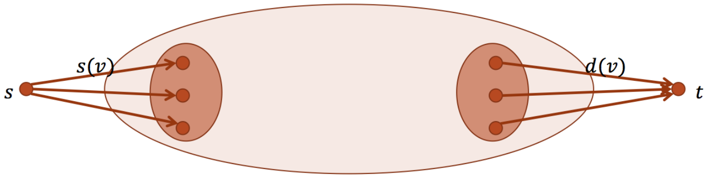


Figure: Connect source to supply nodes and demand nodes to sink.

- Claim 1: For a feasible circulation to exist, $\sum_v t(v) = 0$. (*That means supply equals the demand*)
- Consider the flow network G' as shown in the diagram below and let $D = \sum_{\text{demand node } v} d(v)$.
- Claim 2: There is a feasible circulation in G iff the maximum flow in the network G' is D.
  - (if) Consider the max-flow and remove s, t.
  - (only if) Extend the feasible circulation in the network.

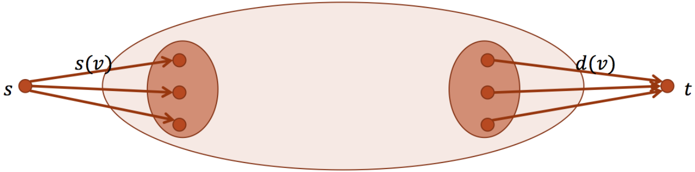


Figure: Connect source to supply nodes and demand nodes to sink.

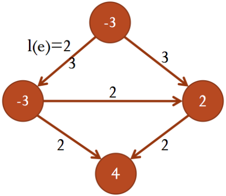
Problem

Given a directed graph G with integer edge capacities $c(e)$ and lower bounds $l(e)$. For each node v, there is an associated demand value $t(v)$ denoting the demand at the node (*for supply nodes this is* $-s(v)$, *for demand nodes* $d(v)$, *for other nodes* 0). Find whether there exists a flow f such that for all nodes v:

$$f^{in}(v) - f^{out}(v) = t(v)$$

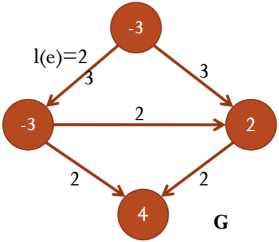
and the following capacity constraints are met. For every edge e:

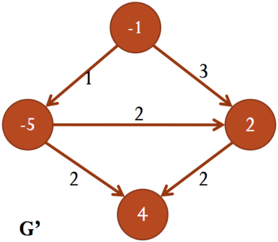
$$l(e) \leq f(e) \leq c(e)$$



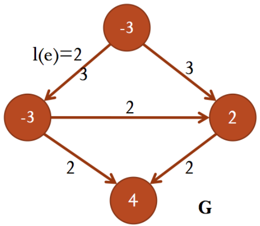
Network Flow
Feasible Circulation with Lower Bounds

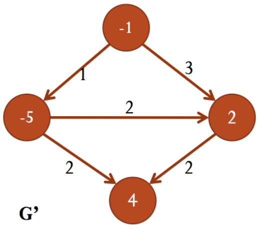
- Consider a flow f such that for all edge e, $f(e) = l(e)$.
- For each vertex v, let $r(v) = f^{in}(v) - f^{out}(v)$.
- Construct a new graph G':
  - Each edge e in G' has capacity $c(e) - l(e)$.
  - Each vertex v in G' has a demand $t(v) - r(v)$.
- <u>Idea</u>: Solve the feasible circulation problem without lower bounds on G'.



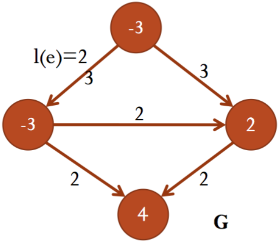


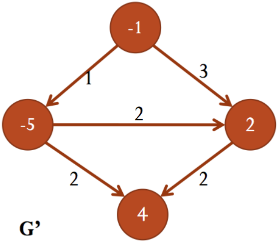
- Consider a flow f such that for all edge e, $f(e) = l(e)$.
- For each vertex v, let $r(v) = f^{in}(v) - f^{out}(v)$.
- Construct a new graph G':
  - Each edge e in G' has capacity $c(e) - l(e)$.
  - Each vertex v in G' has a demand $t(v) - r(v)$.
- Idea: Solve the feasible circulation problem without lower bounds on G'.
- Claim: There is a feasible circulation (with lower bounds) in G iff there is a feasible circulation in G'.





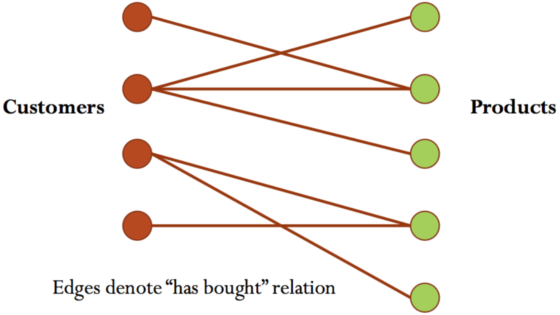
- Claim: There is a feasible circulation (with lower bounds) in G iff there is a feasible circulation in G'.
  - (**if**) Let f' be a feasible circulation in G'. Consider f where $f(e) = f'(e) + l(e)$. Is f a feasible circulation in G?
  - (**only if**) Let f be a feasible circulation in G. Consider f' where $f'(e) = f(e) - l(e)$. Is f' a feasible circulation in G'?





Problem

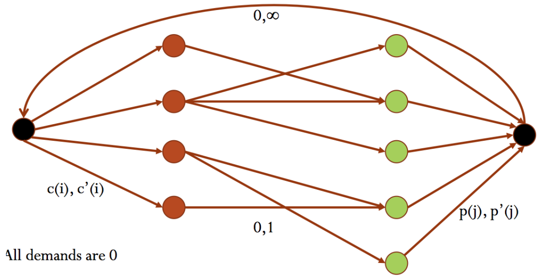
There are n customers and m products. Each customer i is supposed to review between $c(i)$ and $c'(i)$ products that he has bought in the past and each product j should be reviewed by between $p(j)$ and $p'(j)$ customers. Find a way to do the survey.



Problem

There are n customers and m products. Each customer i is supposed to review between $c(i)$ and $c'(i)$ products that he has bought in the past and each product j should be reviewed by between $p(j)$ and $p'(j)$ customers. Find a way to do the survey.

- Consider the flow network set up below.
- Claim: The survey is feasible iff there is a feasible circulation (with lower bounds) in the network.



- You are given an image as a 2-D matrix of pixels.
- We want to determine the foreground and the background pixels.
- Each pixel i, has an integer $a(i)$ associated with it denoting how likely it is to be a foreground pixel.
- Similarly, each pixel i, has an integer $b(i)$ associated with it denoting how likely it is to be a foreground pixel.
- For neighboring pixels, i and j, there is an associated penalty $p(i, j)$ with putting i and j in different sets.

Problem

Find a partition of the pixels into F and B such that:

$$\sum_{i \in F} a(i) + \sum_{i \in B} b(i) - \sum_{i \text{ and } j \text{ are neighbors but in different sets}} p(i, j)$$

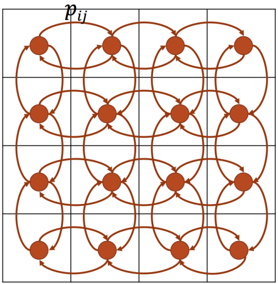
is maximized.

Problem

Find a partition of the pixels into F and B such that:

$$\sum_{i \in F} a(i) + \sum_{i \in B} b(i) - \sum_{i \text{ and } j \text{ are neighbors but in different sets}} p(i,j)$$

is maximized.



Problem

Find a partition of the pixels into F and B such that:

$$\sum_{i \in F} a(i) + \sum_{i \in B} b(i) - \sum_{i \text{ and } j \text{ are neighbors but in different sets}} p(i,j)$$

is maximized.

- Consider the network below:

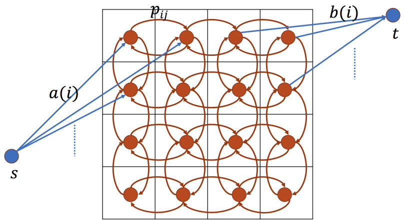


Figure: Idea: The s-t min-cut in the above network gives the optimal partition.

- Let $C = \sum_i a(i) + \sum_i b(i)$.
- Claim 1: Consider a partition (F, B) of the set of pixels. Let $S = F \cup \{s\}$, $T = B \cup \{t\}$. Then the capacity of the s-t cut (S, T) in the network is given by

$$C(S,T) = C - \left(\sum_{i \in F} a(i) + \sum_{i \in B} b(j) - \sum_{i \text{ and } j \text{ are neighbors but in different sets}} p(i,j) \right)$$

- Let $C = \sum_i a(i) + \sum_i b(i)$.
- Claim 1: Consider a partition (F, B) of the set of pixels. Let $S = F \cup \{s\}$, $T = B \cup \{t\}$. Then the capacity of the s-t cut (S, T) in the network is given by

$$C(S,T) = C - \left(\sum_{i \in F} a(i) + \sum_{i \in B} b(j) - \sum_{i \text{ and } j \text{ are neighbors but in different sets}} p(i,j) \right)$$

- Claim 2: Consider an s-t cut (S, T) in the network. Let $F = A \setminus \{s\}$, $B = T \setminus \{t\}$. Then

$$C(S,T) = C - \left(\sum_{i \in F} a(i) + \sum_{i \in B} b(j) - \sum_{i \text{ and } j \text{ are neighbors but in different sets}} p(i,j) \right)$$

- Let $C = \sum_i a(i) + \sum_i b(i)$.
- Claim 1: Consider a partition (F, B) of the set of pixels. Let $S = F \cup \{s\}$, $T = B \cup \{t\}$. Then the capacity of the s-t cut (S, T) in the network is given by

$$C(S,T) = C - \left(\sum_{i \in F} a(i) + \sum_{i \in B} b(j) - \sum_{i \text{ and } j \text{ are neighbors but in different sets}} p(i,j) \right)$$

- Claim 2: Consider an s-t cut (S, T) in the network. Let $F = A \setminus \{s\}$, $B = T \setminus \{t\}$. Then

$$C(S,T) = C - \left(\sum_{i \in F} a(i) + \sum_{i \in B} b(j) - \sum_{i \text{ and } j \text{ are neighbors but in different sets}} p(i,j) \right)$$

- Form Claims 1 and 2, we get that if (S, T) is a s-t min-cut in the network, then $F = S \setminus \{s\}, B = T \setminus \{t\}$ is an optimal solution to the Image Segmentation problem

End