Applications of Network Flow

T. M. Murali

April 7, 12 2016

Maximum Flow and Minimum Cut

- Two rich algorithmic problems.
- Fundamental problems in combinatorial optimization.
- Beautiful mathematical duality between flows and cuts.
- Numerous non-trivial applications:
  - Bipartite matching.
  - Data mining.
  - Project selection.
  - Airline scheduling.
  - Baseball elimination.
  - Image segmentation.
  - Network connectivity.
  - Open-pit mining.
  - Network reliability.
  - Distributed computing.
  - Egalitarian stable matching.
  - Security of statistical data.
  - Network intrusion detection.
  - Multi-camera scene reconstruction.
  - Gene function prediction.

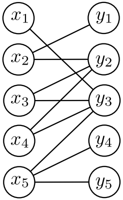
Maximum Flow and Minimum Cut

- Two rich algorithmic problems.
- Fundamental problems in combinatorial optimization.
- Beautiful mathematical duality between flows and cuts.
- Numerous non-trivial applications:
  - Bipartite matching.
  - Data mining.
  - Project selection.
  - Airline scheduling.
  - Baseball elimination.
  - Image segmentation.
  - Network connectivity.
  - Open-pit mining.
  - Network reliability.
  - Distributed computing.
  - Egalitarian stable matching.
  - Security of statistical data.
  - Network intrusion detection.
  - Multi-camera scene reconstruction.
  - Gene function prediction.

Maximum Flow and Minimum Cut

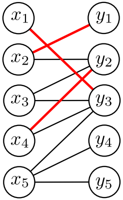
- Two rich algorithmic problems.
- Fundamental problems in combinatorial optimization.
- Beautiful mathematical duality between flows and cuts.
- Numerous non-trivial applications:
  - Bipartite matching.
  - Data mining.
  - Project selection.
  - Airline scheduling.
  - Baseball elimination.
  - Image segmentation.
  - Network connectivity.
  - Open-pit mining.
  - Network reliability.
  - Distributed computing.
  - Egalitarian stable matching.
  - Security of statistical data.
  - Network intrusion detection.
  - Multi-camera scene reconstruction.
  - Gene function prediction.
- We will only sketch proofs. Read details from the textbook.

Matching in Bipartite Graphs



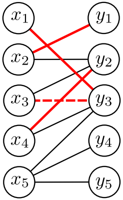
- *Bipartite Graph*: a graph $G(V, E)$ where
  $V = X \cup Y$, X and Y are disjoint and
  $E \subseteq X \times Y$.
- Bipartite graphs model situations in which objects are matched with or assigned to other objects: e.g., marriages, residents/hospitals, jobs/machines.

Matching in Bipartite Graphs



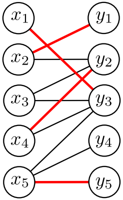
- *Bipartite Graph*: a graph $G(V, E)$ where
  $V = X \cup Y$, X and Y are disjoint and
  $E \subseteq X \times Y$.
- Bipartite graphs model situations in which objects are matched with or assigned to other objects: e.g., marriages, residents/hospitals, jobs/machines.
- A *matching* in a bipartite graph G is a set $M \subseteq E$ of edges such that each node of V is incident on at most edge of M.
- A set of edges M is a *perfect matching* if every node in V is incident on exactly one edge in M.

Matching in Bipartite Graphs



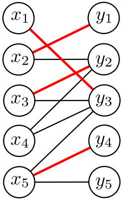
- *Bipartite Graph*: a graph $G(V, E)$ where
  $V = X \cup Y$, X and Y are disjoint and
  $E \subseteq X \times Y$.
- Bipartite graphs model situations in which objects are matched with or assigned to other objects: e.g., marriages, residents/hospitals, jobs/machines.
- A *matching* in a bipartite graph G is a set $M \subseteq E$ of edges such that each node of V is incident on at most edge of M.
- A set of edges M is a *perfect matching* if every node in V is incident on exactly one edge in M.

Matching in Bipartite Graphs



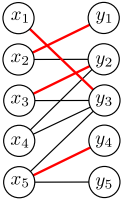
- *Bipartite Graph*: a graph $G(V, E)$ where
  $V = X \cup Y$, X and Y are disjoint and
  $E \subseteq X \times Y$.
- Bipartite graphs model situations in which objects are matched with or assigned to other objects: e.g., marriages, residents/hospitals, jobs/machines.
- A *matching* in a bipartite graph G is a set $M \subseteq E$ of edges such that each node of V is incident on at most edge of M.
- A set of edges M is a *perfect matching* if every node in V is incident on exactly one edge in M.

Matching in Bipartite Graphs



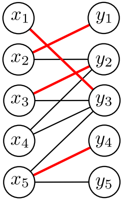
- *Bipartite Graph*: a graph $G(V, E)$ where
  $V = X \cup Y$, X and Y are disjoint and
  $E \subseteq X \times Y$.
- Bipartite graphs model situations in which objects are matched with or assigned to other objects: e.g., marriages, residents/hospitals, jobs/machines.
- A *matching* in a bipartite graph G is a set $M \subseteq E$ of edges such that each node of V is incident on at most edge of M.
- A set of edges M is a *perfect matching* if every node in V is incident on exactly one edge in M.

Matching in Bipartite Graphs



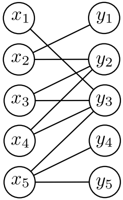
- *Bipartite Graph*: a graph $G(V, E)$ where
  $V = X \cup Y$, X and Y are disjoint and
  $E \subseteq X \times Y$.
- Bipartite graphs model situations in which objects are matched with or assigned to other objects: e.g., marriages, residents/hospitals, jobs/machines.
- A *matching* in a bipartite graph G is a set $M \subseteq E$ of edges such that each node of V is incident on at most edge of M.
- A set of edges M is a *perfect matching* if every node in V is incident on exactly one edge in M.
  - The graph in the figure does not have a perfect matching because

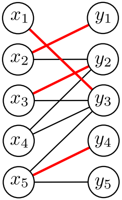
Matching in Bipartite Graphs



- *Bipartite Graph*: a graph $G(V, E)$ where
  $V = X \cup Y$, X and Y are disjoint and
  $E \subseteq X \times Y$.
- Bipartite graphs model situations in which objects are matched with or assigned to other objects: e.g., marriages, residents/hospitals, jobs/machines.
- A *matching* in a bipartite graph G is a set $M \subseteq E$ of edges such that each node of V is incident on at most edge of M.
- A set of edges M is a *perfect matching* if every node in V is incident on exactly one edge in M.
  - The graph in the figure does not have a perfect matching because both y_4 and y_5 are adjacent only to x_5.

Bipartite Graph Matching Problem



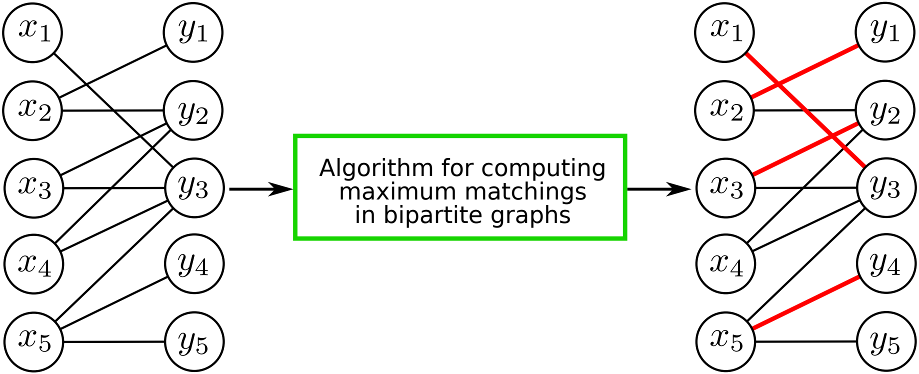


BIPARTITE MATCHING

INSTANCE: A Bipartite graph G.

SOLUTION: The matching of largest size in G.

Normal Approach for Solving a Problem



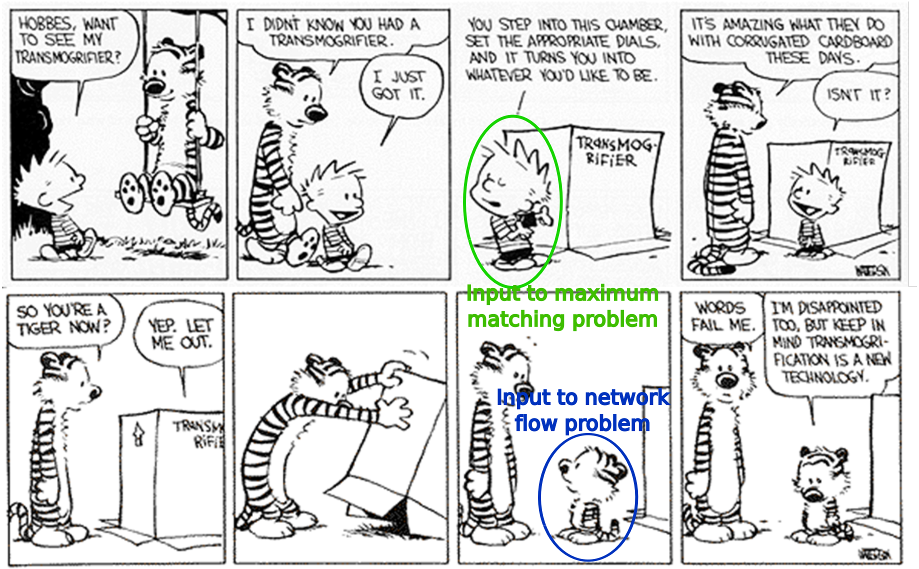
(i) Develop algorithm for computing maximum matchings in bipartite graphs.

(ii) Prove that the algorithm is correct, i.e., for every possible input, it compute the size of the largest matching in the bipartite graph accurately.

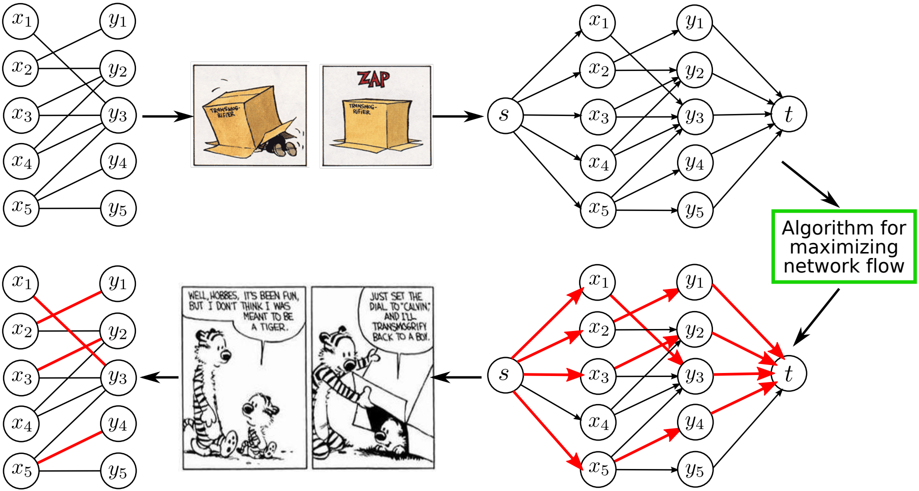
(iii) Analyze running time of the algorithm.

Alternative Approach for Solving a Problem

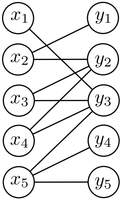
Alternative Approach for Solving a Problem

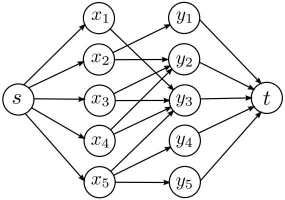


Alternative Approach for Solving a Problem



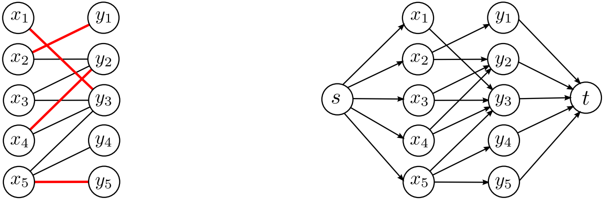
Algorithm for Bipartite Graph Matching





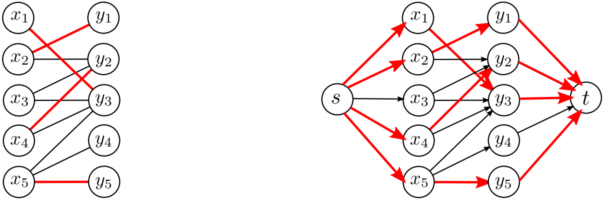
- Convert G to a flow network G': direct edges from X to Y, add nodes s and t, connect s to each node in X, connect each node in Y to t, set all edge capacities to 1.
- Compute the maximum flow in G'.
- Claim: the value of the maximum flow in G' is the size of the maximum matching in G.
- In general, there is matching with size k in G if and only if there is a (integer-valued) flow of value k in G'.

Correctness of Bipartite Graph Matching Algorithm



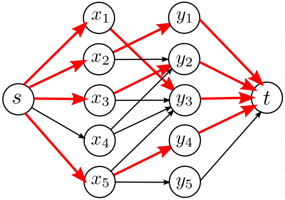
- Matching $\Rightarrow$ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.

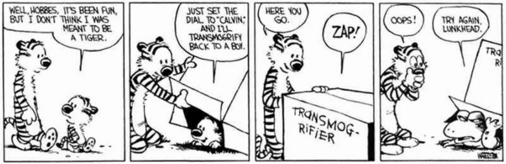
Correctness of Bipartite Graph Matching Algorithm

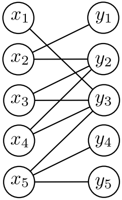


- Matching $\Rightarrow$ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.

Correctness of Bipartite Graph Matching Algorithm

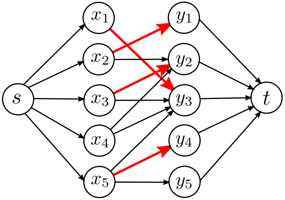


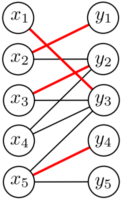




- Matching $\Rightarrow$ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.
- Flow $\Rightarrow$ matching: if there is a flow f' in G' with value k, there is a matching M in G with k edges.

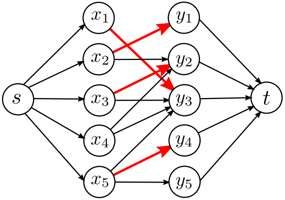
Correctness of Bipartite Graph Matching Algorithm

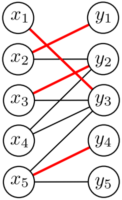




- Matching $\Rightarrow$ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.
- Flow $\Rightarrow$ matching: if there is a flow f' in G' with value k, there is a matching M in G with k edges.
  - There is an integer-valued flow f' of value k $\Rightarrow$ flow along any edge is 0 or 1.

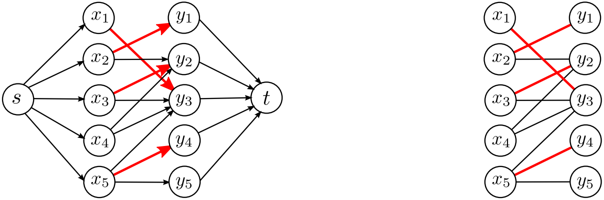
Correctness of Bipartite Graph Matching Algorithm





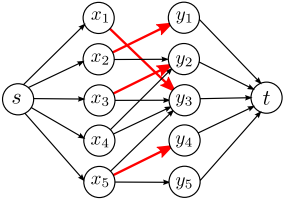
- Matching $\Rightarrow$ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.
- Flow $\Rightarrow$ matching: if there is a flow f' in G' with value k, there is a matching M in G with k edges.
  - There is an integer-valued flow f' of value $k \Rightarrow$ flow along any edge is 0 or 1.
  - Let M be the set of edges not incident on s or t with flow equal to 1.

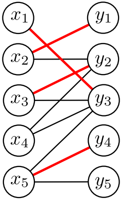
Correctness of Bipartite Graph Matching Algorithm



- Matching $\Rightarrow$ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.
- Flow $\Rightarrow$ matching: if there is a flow f' in G' with value k, there is a matching M in G with k edges.
  - There is an integer-valued flow f' of value $k \Rightarrow$ flow along any edge is 0 or 1.
  - Let M be the set of edges not incident on s or t with flow equal to 1.
  - Claim: M contains k edges.

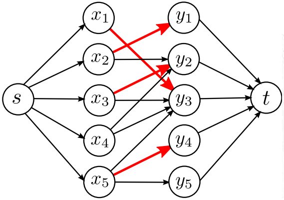
Correctness of Bipartite Graph Matching Algorithm

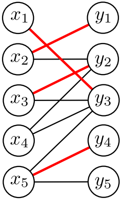




- Matching $\Rightarrow$ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.
- Flow $\Rightarrow$ matching: if there is a flow f' in G' with value k, there is a matching M in G with k edges.
  - There is an integer-valued flow f' of value k $\Rightarrow$ flow along any edge is 0 or 1.
  - Let M be the set of edges not incident on s or t with flow equal to 1.
  - Claim: M contains k edges.
  - Claim: Each node in X (respectively, Y) is the tail (respectively, head) of at most one edge in M.

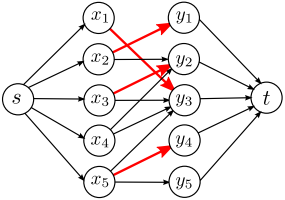
Correctness of Bipartite Graph Matching Algorithm

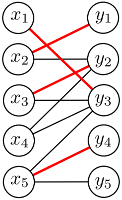




- Matching $\Rightarrow$ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.
- Flow $\Rightarrow$ matching: if there is a flow f' in G' with value k, there is a matching M in G with k edges.
  - There is an integer-valued flow f' of value $k \Rightarrow$ flow along any edge is 0 or 1.
  - Let M be the set of edges not incident on s or t with flow equal to 1.
  - Claim: M contains k edges.
  - Claim: Each node in X (respectively, Y) is the tail (respectively, head) of at most one edge in M.
- Conclusion: size of the maximum matching in G is equal to the value of the maximum flow in G'; the edges in this matching are those that carry flow from X to Y in G'.

Correctness of Bipartite Graph Matching Algorithm





- Matching $\Rightarrow$ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.
- Flow $\Rightarrow$ matching: if there is a flow f' in G' with value k, there is a matching M in G with k edges.
  - There is an integer-valued flow f' of value $k \Rightarrow$ flow along any edge is 0 or 1.
  - Let M be the set of edges not incident on s or t with flow equal to 1.
  - Claim: M contains k edges.
  - Claim: Each node in X (respectively, Y) is the tail (respectively, head) of at most one edge in M.
- Conclusion: size of the maximum matching in G is equal to the value of the maximum flow in G'; the edges in this matching are those that carry flow from X to Y in G'.
- Read the book on what augmenting paths mean in this context.

Running time of Bipartite Graph Matching Algorithm

- Suppose G has m edges and n nodes in X and in Y.

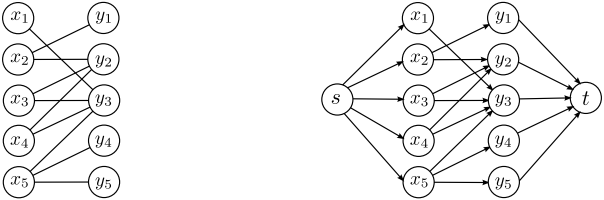
Running time of Bipartite Graph Matching Algorithm

- Suppose G has m edges and n nodes in X and in Y.
- $C \leq n$.
- Ford-Fulkerson algorithm runs in $O(mn)$ time.
- How long does the scaling algorithm take?

Running time of Bipartite Graph Matching Algorithm

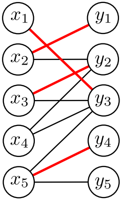
- Suppose G has m edges and n nodes in X and in Y.
- $C \leq n$.
- Ford-Fulkerson algorithm runs in $O(mn)$ time.
- How long does the scaling algorithm take? $O(m^2)$ time ($C = 1$ for this algorithm).

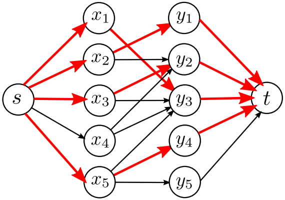
Bipartite Graphs without Perfect Matchings



- How do we determine if a bipartite graph G has a perfect matching?

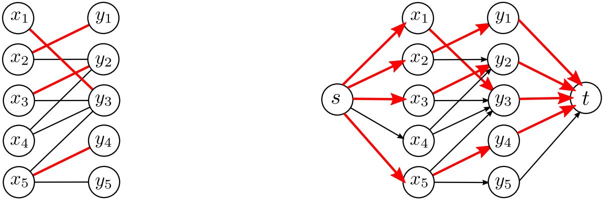
Bipartite Graphs without Perfect Matchings





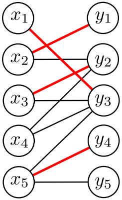
- How do we determine if a bipartite graph G has a perfect matching? Find the maximum matching and check if it is perfect.

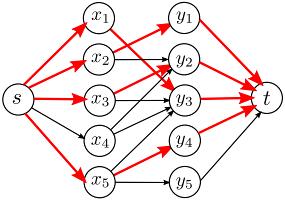
Bipartite Graphs without Perfect Matchings



- How do we determine if a bipartite graph G has a perfect matching? Find the maximum matching and check if it is perfect.
- Suppose G has no perfect matching. Can we exhibit a short "certificate" of that fact? What can such certificates look like?

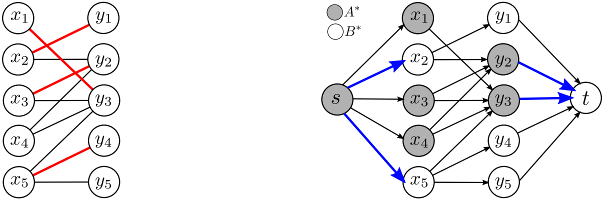
Bipartite Graphs without Perfect Matchings





- How do we determine if a bipartite graph G has a perfect matching? Find the maximum matching and check if it is perfect.
- Suppose G has no perfect matching. Can we exhibit a short "certificate" of that fact? What can such certificates look like?
- G has no perfect matching iff

Bipartite Graphs without Perfect Matchings

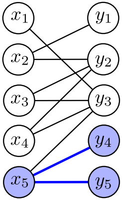


- How do we determine if a bipartite graph G has a perfect matching? Find the maximum matching and check if it is perfect.
- Suppose G has no perfect matching. Can we exhibit a short "certificate" of that fact? What can such certificates look like?
- G has no perfect matching iff there is a cut in G' with capacity less than n. Therefore, the cut is a certificate.

Bipartite Graphs without Perfect Matchings

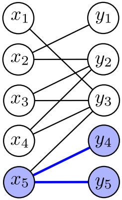
- We would like the certificate in terms of G.

Bipartite Graphs without Perfect Matchings



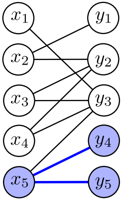
- We would like the certificate in terms of G.
  - For example, two nodes in Y with one incident edge each with the same neighbour in X.

Bipartite Graphs without Perfect Matchings



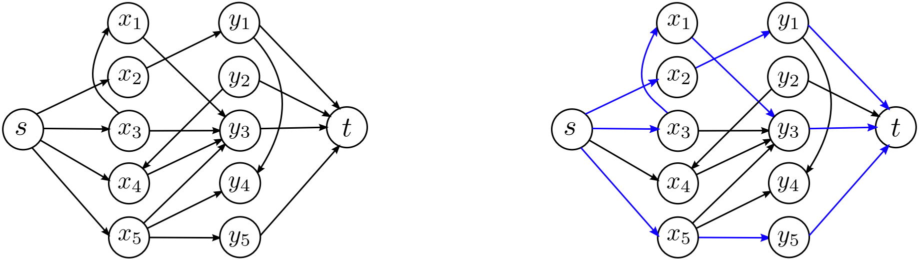
- We would like the certificate in terms of G.
  - For example, two nodes in Y with one incident edge each with the same neighbour in X.
  - Generally, a subset $A \subseteq X$ with neighbours $\Gamma(A) \subseteq Y$, such that $|A| > |\Gamma(A)|$.
- Hall's Theorem: Let $G(X \cup Y, E)$ be a bipartite graph such that $|X| = |Y|$. Then G either has a perfect matching or there is a subset $A \subseteq Y$ such that $|A| > |\Gamma(A)|$. A perfect matching or such a subset can be computed in $O(mn)$ time.

Bipartite Graphs without Perfect Matchings



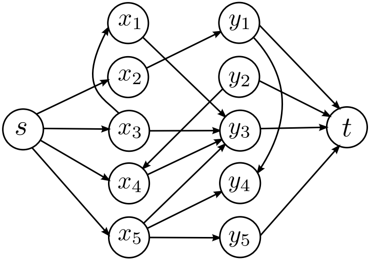
- We would like the certificate in terms of G.
  - For example, two nodes in Y with one incident edge each with the same neighbour in X.
  - Generally, a subset $A \subseteq X$ with neighbours $\Gamma(A) \subseteq Y$, such that $|A| > |\Gamma(A)|$.
- Hall's Theorem: Let $G(X \cup Y, E)$ be a bipartite graph such that $|X| = |Y|$. Then G either has a perfect matching or there is a subset $A \subseteq Y$ such that $|A| > |\Gamma(A)|$. A perfect matching or such a subset can be computed in $O(mn)$ time. Read proof in the textbook.

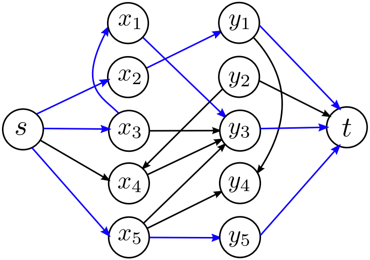
Edge-Disjoint Paths



- A set of paths in a graph G is *edge disjoint* if each edge in G appears in at most one path.

Edge-Disjoint Paths





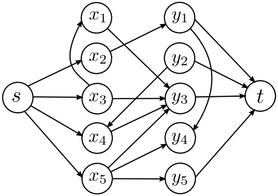
- A set of paths in a graph G is *edge disjoint* if each edge in G appears in at most one path.

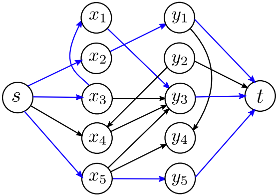
DIRECTED EDGE-DISJOINT PATHS

INSTANCE: Directed graph $G(V, E)$ with two distinguished nodes s and t.

SOLUTION: The maximum number of edge-disjoint paths between s and t.

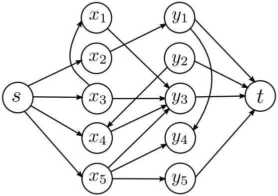
Mapping to the Max-Flow Problem

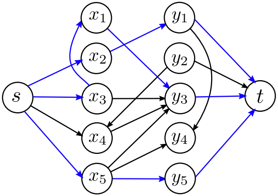




- Convert G into a flow network: s is the source, t is the sink, each edge has capacity 1.
- Claim: There are k edge-disjoint paths from s to t in a directed graph G if and only if the maximum value of an s-t flow in G is $\geq k$.

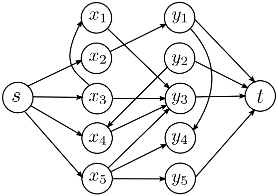
Mapping to the Max-Flow Problem

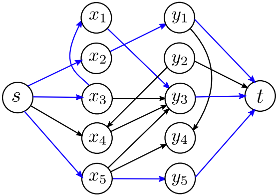




- Convert G into a flow network: s is the source, t is the sink, each edge has capacity 1.
- Claim: There are k edge-disjoint paths from s to t in a directed graph G if and only if the maximum value of an s-t flow in G is $\geq k$.
- Paths $\Rightarrow$ flow: if there are k edge-disjoint paths from s to t,

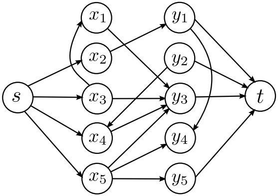
Mapping to the Max-Flow Problem

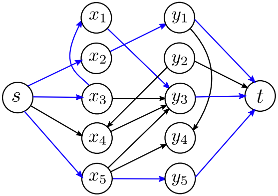




- Convert G into a flow network: s is the source, t is the sink, each edge has capacity 1.
- Claim: There are k edge-disjoint paths from s to t in a directed graph G if and only if the maximum value of an s-t flow in G is $\geq k$.
- Paths $\Rightarrow$ flow: if there are k edge-disjoint paths from s to t, send one unit of flow along each to yield a flow with value k.

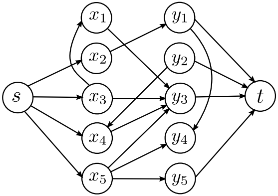
Mapping to the Max-Flow Problem

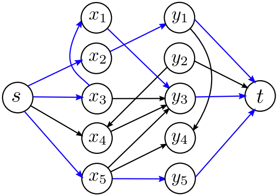




- Convert G into a flow network: s is the source, t is the sink, each edge has capacity 1.
- Claim: There are k edge-disjoint paths from s to t in a directed graph G if and only if the maximum value of an s-t flow in G is $\geq k$.
- Paths $\Rightarrow$ flow: if there are k edge-disjoint paths from s to t, send one unit of flow along each to yield a flow with value k.
- Flow $\Rightarrow$ paths: Suppose there is an integer-valued flow of value at least k. Are there k edge-disjoint paths? If so, what are they?

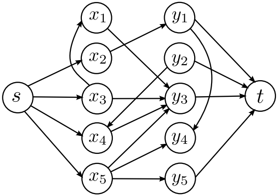
Mapping to the Max-Flow Problem

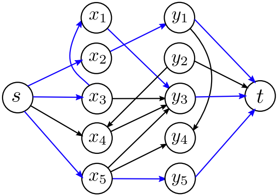




- Convert G into a flow network: s is the source, t is the sink, each edge has capacity 1.
- Claim: There are k edge-disjoint paths from s to t in a directed graph G if and only if the maximum value of an s-t flow in G is $\geq k$.
- Paths $\Rightarrow$ flow: if there are k edge-disjoint paths from s to t, send one unit of flow along each to yield a flow with value k.
- Flow $\Rightarrow$ paths: Suppose there is an integer-valued flow of value at least k. Are there k edge-disjoint paths? If so, what are they?
- Construct k edge-disjoint paths from a flow of value $\geq k$ as follows:
  - There is an integral flow. Therefore, flow on each edge is 0 or 1.

Mapping to the Max-Flow Problem





- Convert G into a flow network: s is the source, t is the sink, each edge has capacity 1.
- Claim: There are k edge-disjoint paths from s to t in a directed graph G if and only if the maximum value of an s-t flow in G is $\geq k$.
- Paths $\Rightarrow$ flow: if there are k edge-disjoint paths from s to t, send one unit of flow along each to yield a flow with value k.
- Flow $\Rightarrow$ paths: Suppose there is an integer-valued flow of value at least k. Are there k edge-disjoint paths? If so, what are they?
- Construct k edge-disjoint paths from a flow of value $\geq k$ as follows:
  - There is an integral flow. Therefore, flow on each edge is 0 or 1.
  - Claim: if f is a 0-1 valued flow of value $\nu(f) = \nu$, then the set of edges with flow $f(e) = 1$ contains a set of ν edge-disjoint paths.

Completing the Proof

- Claim: if f is a 0-1 valued flow of value $\nu(f) = \nu$, then the set of edges with flow $f(e) = 1$ contains a set of ν edge-disjoint paths.
- Prove by induction on the number of edges in f that carry flow. Let this number be $\kappa(f)$.

  Base case: $\nu = 0$. Nothing to prove.

Completing the Proof

- Claim: if f is a 0-1 valued flow of value $\nu(f) = \nu$, then the set of edges with flow $f(e) = 1$ contains a set of ν edge-disjoint paths.
- Prove by induction on the number of edges in f that carry flow. Let this number be $\kappa(f)$.

  Base case: $\nu = 0$. Nothing to prove.

  Inductive hypothesis: For every flow f' in G with
  (a) value $\nu(f') < \nu$ carrying flow on $\kappa(f') < \kappa(f)$ edges or
  (b) value $\nu(f') = \nu$ carrying flow on $\kappa(f') < \kappa(f)$ edges,
  the set of edges with $f'(e) = 1$ contains a set of $\nu(f')$ edge-disjoint s-t paths.

Completing the Proof

- Claim: if f is a 0-1 valued flow of value $\nu(f) = \nu$, then the set of edges with flow $f(e) = 1$ contains a set of ν edge-disjoint paths.
- Prove by induction on the number of edges in f that carry flow. Let this number be $\kappa(f)$.

  Base case: $\nu = 0$. Nothing to prove.

  Inductive hypothesis: For every flow f' in G with
  (a) value $\nu(f') < \nu$ carrying flow on $\kappa(f') < \kappa(f)$ edges or
  (b) value $\nu(f') = \nu$ carrying flow on $\kappa(f') < \kappa(f)$ edges,
  the set of edges with $f'(e) = 1$ contains a set of $\nu(f')$ edge-disjoint s-t paths.

  Inductive step: Construct a set of ν s-t paths from f. Work out on the board.

Completing the Proof

- Claim: if f is a 0-1 valued flow of value $\nu(f) = \nu$, then the set of edges with flow $f(e) = 1$ contains a set of ν edge-disjoint paths.
- Prove by induction on the number of edges in f that carry flow. Let this number be $\kappa(f)$.

  Base case: $\nu = 0$. Nothing to prove.

  Inductive hypothesis: For every flow f' in G with
  (a) value $\nu(f') < \nu$ carrying flow on $\kappa(f') < \kappa(f)$ edges or
  (b) value $\nu(f') = \nu$ carrying flow on $\kappa(f') < \kappa(f)$ edges,
  the set of edges with $f'(e) = 1$ contains a set of $\nu(f')$ edge-disjoint s-t paths.

  Inductive step: Construct a set of ν s-t paths from f. Work out on the board.
- Note: Formulating the inductive hypothesis precisely can be tricky.
- Strategy is to try to prove the inductive step first.
- During this proof, you will observe two types of "smaller" flows:
  (i) When you succeed in finding an s-t path, you get a new flow f' that is smaller, i.e., $\nu(f') < \nu$ carrying flow on fewer edges, i.e., $\kappa(f') < \kappa(f)$.
  (ii) When you run into a cycle, you get a new flow f' with $\nu(f') = \nu$ but carrying flow on fewer edges, i.e., $\kappa(f') < \kappa(f)$ edges.

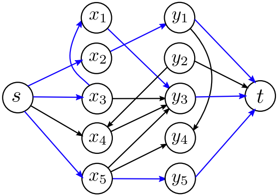
Running Time of the Edge-Disjoint Paths Algorithm

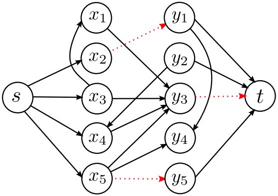
- Given a flow of value k, how quickly can we determine the k edge-disjoint paths?

Running Time of the Edge-Disjoint Paths Algorithm

- Given a flow of value k, how quickly can we determine the k edge-disjoint paths? $O(mn)$ time.
- Corollary: The Ford-Fulkerson algorithm can be used to find a maximum set of edge-disjoint s-t paths in a directed graph G in $O(mn)$ time.

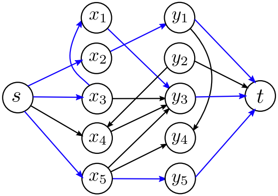
Certificate for Edge-Disjoint Paths Algorithm

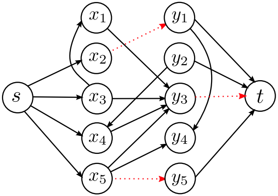




- A set $F \subseteq E$ of edge separates s and t if the graph $(V, E - F)$ contains no s-t paths.

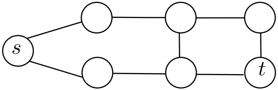
Certificate for Edge-Disjoint Paths Algorithm





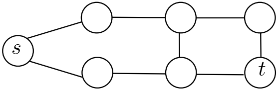
- A set $F \subseteq E$ of edge separates s and t if the graph $(V, E - F)$ contains no s-t paths.
- Menger's Theorem: In every directed graph with nodes s and t, the maximum number of edge-disjoint s-t paths is equal to the minimum number of edges whose removal disconnects s from t.

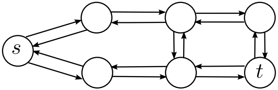
Edge-Disjoint Paths in Undirected Graphs



- Can extend the theorem to *undirected* graphs.

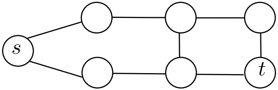
Edge-Disjoint Paths in Undirected Graphs

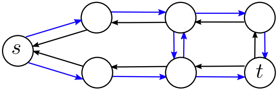




- Can extend the theorem to *undirected* graphs.
- Replace each edge with two directed edges of capacity 1 and apply the algorithm for directed graphs.

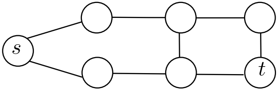
Edge-Disjoint Paths in Undirected Graphs

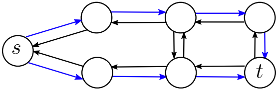




- Can extend the theorem to *undirected* graphs.
- Replace each edge with two directed edges of capacity 1 and apply the algorithm for directed graphs.
- Problem: Both counterparts of an undirected edge (u, v) may be used by different edge-disjoint paths in the directed graph.

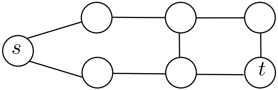
Edge-Disjoint Paths in Undirected Graphs

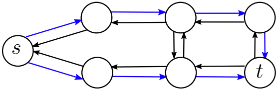




- Can extend the theorem to *undirected* graphs.
- Replace each edge with two directed edges of capacity 1 and apply the algorithm for directed graphs.
- Problem: Both counterparts of an undirected edge (u, v) may be used by different edge-disjoint paths in the directed graph.
- Can obtain an integral flow where only one of the directed counterparts of (u, v) has non-zero flow.

Edge-Disjoint Paths in Undirected Graphs





- Can extend the theorem to *undirected* graphs.
- Replace each edge with two directed edges of capacity 1 and apply the algorithm for directed graphs.
- Problem: Both counterparts of an undirected edge (u, v) may be used by different edge-disjoint paths in the directed graph.
- Can obtain an integral flow where only one of the directed counterparts of (u, v) has non-zero flow.
- We can find the maximum number of edge-disjoint paths in $O(mn)$ time.
- We can prove a version of Menger's theorem for undirected graphs: in every undirected graph with nodes s and t, the maximum number of edge-disjoint s–t paths is equal to the minimum number of edges whose removal separates s from t.

Image Segmentation


- A fundamental problem in computer vision is that of segmenting an image into coherent regions.
- A basic segmentation problem is that of partitioning an image into a foreground and a background: label each pixel in the image as belonging to the foreground or the background.
  - Note that the image on the right shows segmentation into multiple regions but we are interested in the segmentation into two regions.

Formulating the Image Segmentation Problem

- Let V be the set of pixels in an image.
- Let E be the set of pairs of neighbouring pixels.
- V and E yield an undirected graph $G(V, E)$.

Formulating the Image Segmentation Problem


- Let V be the set of pixels in an image.
- Let E be the set of pairs of neighbouring pixels.
- V and E yield an undirected graph $G(V, E)$.
- Each pixel i has a likelihood $a_i > 0$ that it belongs to the foreground and a likelihood $b_i > 0$ that it belongs to the background.
- These likelihoods are specified in the input to the problem.

Formulating the Image Segmentation Problem

- Let V be the set of pixels in an image.
- Let E be the set of pairs of neighbouring pixels.
- V and E yield an undirected graph $G(V, E)$.
- Each pixel i has a likelihood $a_i > 0$ that it belongs to the foreground and a likelihood $b_i > 0$ that it belongs to the background.
- These likelihoods are specified in the input to the problem.
- We want the foreground/background boundary to be smooth:

Formulating the Image Segmentation Problem

- Let V be the set of pixels in an image.
- Let E be the set of pairs of neighbouring pixels.
- V and E yield an undirected graph $G(V, E)$.
- Each pixel i has a likelihood $a_i > 0$ that it belongs to the foreground and a likelihood $b_i > 0$ that it belongs to the background.
- These likelihoods are specified in the input to the problem.
- We want the foreground/background boundary to be smooth: For each pair (i, j) of pixels, there is a separation penalty $p_{ij} \geq 0$ for placing one of them in the foreground and the other in the background.

The Image Segmentation Problem

IMAGE SEGMENTATION

INSTANCE: Pixel graphs $G(V, E)$, likelihood functions $a, b : V \to \mathbb{R}^+$, penalty function $p : E \to \mathbb{R}^+$

SOLUTION: *Optimum labelling*: partition of the pixels into two sets A and B that maximises

$$q(A, B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}.$$

Developing an Algorithm for Image Segmentation

- There is a similarity between cuts and labellings.
- But there are differences:
  - We are maximising an objective function rather than minimising it.
  - There is no source or sink in the segmentation problem.
  - We have values on the nodes.
  - The graph is undirected.

Maximization to Minimization

- Let $Q = \sum_i (a_i + b_i)$.

Maximization to Minimization

- Let $Q = \sum_i (a_i + b_i)$.
- Notice that $\sum_{i \in A} a_i + \sum_{j \in B} b_j = Q - \sum_{i \in A} b_i - \sum_{j \in B} a_j$.
- Therefore, maximising

$$\begin{aligned} q(A,B) &= \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cup \{i,j\}| = 1}} p_{ij} \\ &= Q - \sum_{i \in A} b_i - \sum_{j \in B} a_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij} \end{aligned}$$

is identical to minimising

$$q'(A,B) = \sum_{i \in A} b_i + \sum_{j \in B} a_j + \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

Solving the Other Issues

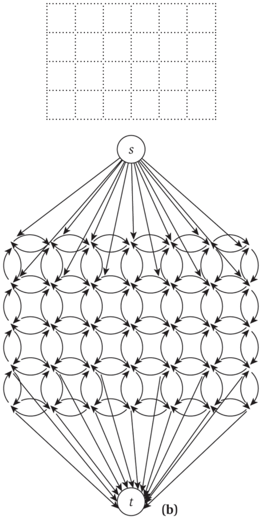
- Solve the issues like we did earlier.

Solving the Other Issues

- Solve the issues like we did earlier.
- Add a new "super-source" s to represent the foreground.
- Add a new "super-sink" t to represent the background.

Solving the Other Issues

- Solve the issues like we did earlier.
- Add a new "super-source" s to represent the foreground.
- Add a new "super-sink" t to represent the background.
- Connect s and t to every pixel and assign capacity a_i to edge (s, i) and capacity b_i to edge (i, t).
- Direct edges away from s and into t.
- Replace each edge (i, j) in E with two directed edges of capacity p_{ij}.



(b)

Cuts in the Flow Network

- Let G' be this flow network and (A, B) an s-t cut.
- What does the capacity of the cut represent?

Cuts in the Flow Network

- Let G' be this flow network and (A, B) an s-t cut.
- What does the capacity of the cut represent?
- Edges crossing the cut are of three types:

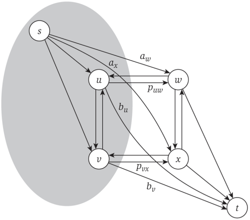


Figure 7.19 An s-t cut on a graph constructed from four pixels. Note how the three types of terms in the expression for $q'(A, B)$ are captured by the cut.

Cuts in the Flow Network

- Let G' be this flow network and (A, B) an s-t cut.
- What does the capacity of the cut represent?
- Edges crossing the cut are of three types:
  - $(s, w), w \in B$ contributes a_w.
  - $(u, t), u \in A$ contributes b_u.
  - $(u, w), u \in A, w \in B$ contributes p_{uw}.

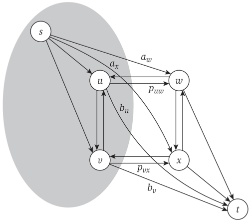


Figure 7.19 An s-t cut on a graph constructed from four pixels. Note how the three types of terms in the expression for $q'(A, B)$ are captured by the cut.

Cuts in the Flow Network

- Let G' be this flow network and (A, B) an s-t cut.
- What does the capacity of the cut represent?
- Edges crossing the cut are of three types:
  - $(s, w), w \in B$ contributes a_w.
  - $(u, t), u \in A$ contributes b_u.
  - $(u, w), u \in A, w \in B$ contributes p_{uw}.

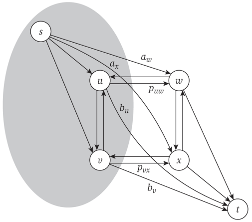


Figure 7.19 An s-t cut on a graph constructed from four pixels. Note how the three types of terms in the expression for $q'(A, B)$ are captured by the cut.

$$c(A, B) \quad = \quad \sum_{i \in A} b_i + \sum_{j \in B} a_j + \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij} = q'(A, B).$$

Solving the Image Segmentation Problem

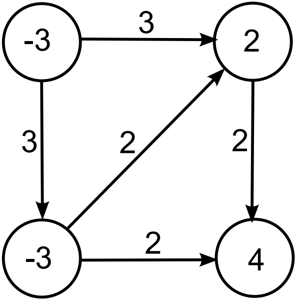
- The capacity of a s-t cut $c(A, B)$ exactly measures the quantity $q'(A, B)$.
- To maximise $q(A, B)$, we simply compute the s-t cut (A, B) of minimum capacity.
- Deleting s and t from the cut yields the desired segmentation of the image.

Extension of Max-Flow Problem

- Suppose we have a set S of multiple sources and a set T of multiple sinks.
- Each source can send flow to any sink.
- Let us not maximise flow here but formulate the problem in terms of demands and supplies.

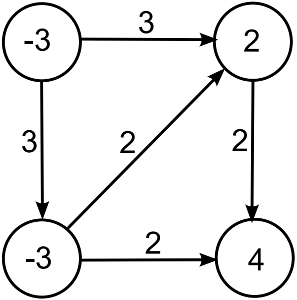
Circulation with Demands

- We are given a graph $G(V, E)$ with capacity function $c : E \rightarrow \mathbb{Z}^+$ and a demand function $d : V \rightarrow \mathbb{Z}$:



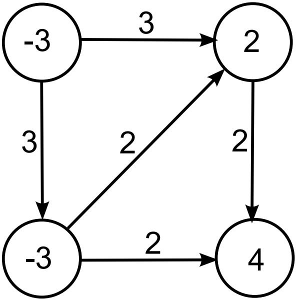
Circulation with Demands

- We are given a graph $G(V, E)$ with capacity function $c : E \to \mathbb{Z}^+$ and a demand function $d : V \to \mathbb{Z}$:
  - $d_v > 0$: node is a sink, it has a "demand" for d_v units of flow.
  - $d_v < 0$: node is a source, it has a "supply" of $-d_v$ units of flow.
  - $d_v = 0$: node simply receives and transmits flow.



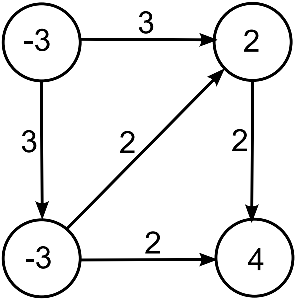
Circulation with Demands

- We are given a graph $G(V, E)$ with capacity function $c : E \rightarrow \mathbb{Z}^+$ and a demand function $d : V \rightarrow \mathbb{Z}$:
  - $d_v > 0$: node is a sink, it has a "demand" for d_v units of flow.
  - $d_v < 0$: node is a source, it has a "supply" of $-d_v$ units of flow.
  - $d_v = 0$: node simply receives and transmits flow.
  - S is the set of nodes with negative demand and T is the set of nodes with positive demand.



Circulation with Demands

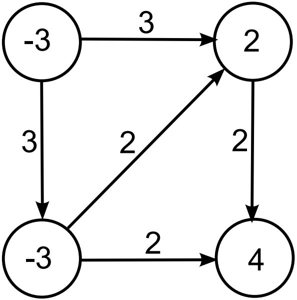
- We are given a graph $G(V, E)$ with capacity function $c : E \rightarrow \mathbb{Z}^+$ and a demand function $d : V \rightarrow \mathbb{Z}$:
  - $d_v > 0$: node is a sink, it has a "demand" for d_v units of flow.
  - $d_v < 0$: node is a source, it has a "supply" of $-d_v$ units of flow.
  - $d_v = 0$: node simply receives and transmits flow.
  - S is the set of nodes with negative demand and T is the set of nodes with positive demand.



- A *circulation* with demands is a function $f : E \rightarrow \mathbb{R}^+$ that satisfies

Circulation with Demands

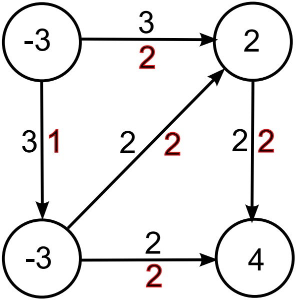
- We are given a graph $G(V, E)$ with capacity function $c : E \to \mathbb{Z}^+$ and a demand function $d : V \to \mathbb{Z}$:
  - $d_v > 0$: node is a sink, it has a "demand" for d_v units of flow.
  - $d_v < 0$: node is a source, it has a "supply" of $-d_v$ units of flow.
  - $d_v = 0$: node simply receives and transmits flow.
  - S is the set of nodes with negative demand and T is the set of nodes with positive demand.



- A *circulation* with demands is a function $f : E \to \mathbb{R}^+$ that satisfies
  (i) (*Capacity conditions*) For each $e \in E$, $0 \leq f(e) \leq c(e)$.
  (ii) (*Demand conditions*) For each node v, $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$.

Circulation with Demands

- We are given a graph $G(V, E)$ with capacity function $c : E \rightarrow \mathbb{Z}^+$ and a demand function $d : V \rightarrow \mathbb{Z}$:
  - $d_v > 0$: node is a sink, it has a "demand" for d_v units of flow.
  - $d_v < 0$: node is a source, it has a "supply" of $-d_v$ units of flow.
  - $d_v = 0$: node simply receives and transmits flow.
  - S is the set of nodes with negative demand and T is the set of nodes with positive demand.



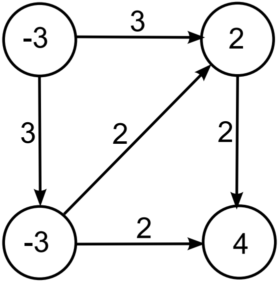
- A *circulation* with demands is a function $f : E \rightarrow \mathbb{R}^+$ that satisfies
  (i) (*Capacity conditions*) For each $e \in E$, $0 \leq f(e) \leq c(e)$.
  (ii) (*Demand conditions*) For each node v, $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$.

Circulation with Demands

INSTANCE: A directed graph $G(V, E)$, $c : E \rightarrow \mathbb{Z}^+$, and $d : V \rightarrow \mathbb{Z}$.

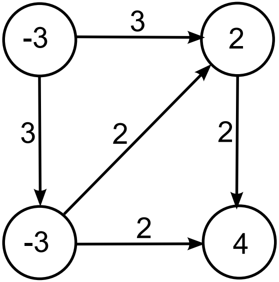
SOLUTION: Does a *feasible* circulation exist, i.e., it meets the capacity and demand conditions?

Properties of Feasible Circulations



- Claim: if there exists a feasible circulation with demands, then $\sum_v d_v = 0$.

Properties of Feasible Circulations



- Claim: if there exists a feasible circulation with demands, then $\sum_v d_v = 0$.
- Corollary: $\sum_{v, d_v > 0} d_v = \sum_{v, d_v < 0} -d_v$. Let D denote this common value.

Mapping Circulation to Maximum Flow

- Create a new graph $G' = G$ and
  (i) create two new nodes in G': a source s^* and a sink t^*;
  (ii) connect s^* to each node v in S using an edge with capacity $-d_v$;
  (iii) connect each node v in T to t^* using an edge with capacity d_v.

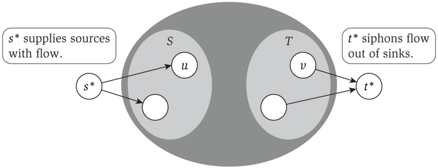
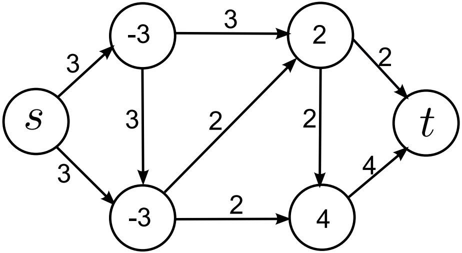
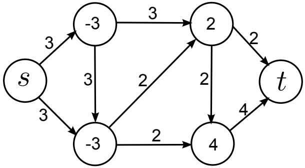
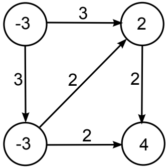


Figure 7.14 Reducing the Circulation Problem to the Maximum-Flow Problem.



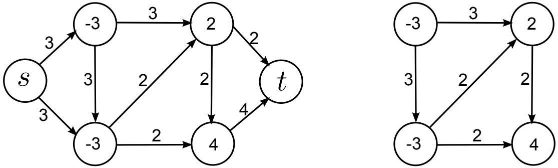
Computing a Feasible Circulation





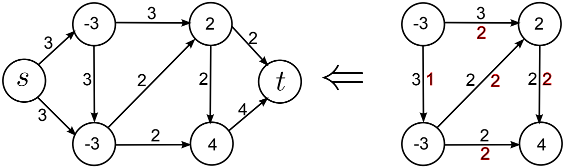
- We will look for a maximum s^*-t^* flow f in G'; $\nu(f)$

Computing a Feasible Circulation



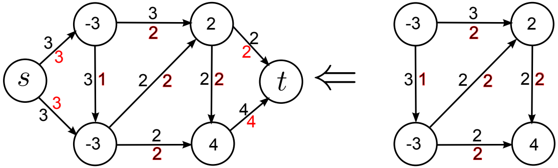
- We will look for a maximum s^*-t^* flow f in G'; $\nu(f) \leq D$.

Computing a Feasible Circulation



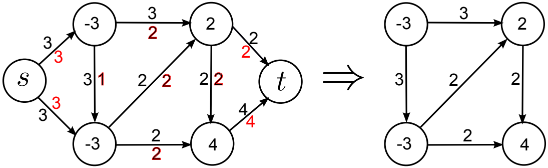
- We will look for a maximum s^*-t^* flow f in G'; $\nu(f) \leq D$.
- Circulation $\Rightarrow$ flow.

Computing a Feasible Circulation



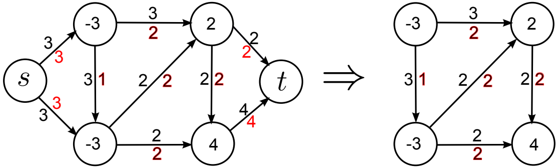
- We will look for a maximum s^*-t^* flow f in G'; $\nu(f) \leq D$.
- Circulation $\Rightarrow$ flow. If there is a feasible circulation, we send $-d_v$ units of flow along each edge (s^*, v) and d_v units of flow along each edge (v, t^*). The value of this flow is D. (Prove it yourself.)

Computing a Feasible Circulation



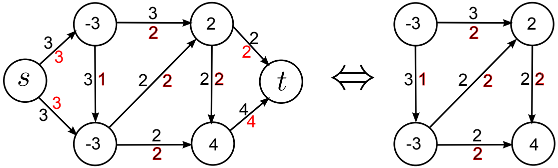
- We will look for a maximum s^*-t^* flow f in G'; $\nu(f) \leq D$.
- Circulation $\Rightarrow$ flow. If there is a feasible circulation, we send $-d_v$ units of flow along each edge (s^*, v) and d_v units of flow along each edge (v, t^*). The value of this flow is D. (Prove it yourself.)
- Flow $\Rightarrow$ circulation. If there is an s^*-t^* flow of value D in G',

Computing a Feasible Circulation



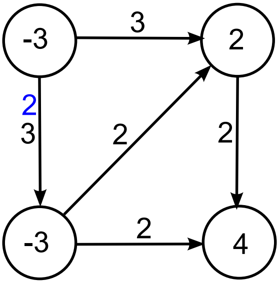
- We will look for a maximum s^*-t^* flow f in G'; $\nu(f) \leq D$.
- Circulation $\Rightarrow$ flow. If there is a feasible circulation, we send $-d_v$ units of flow along each edge (s^*, v) and d_v units of flow along each edge (v, t^*). The value of this flow is D. (Prove it yourself.)
- Flow $\Rightarrow$ circulation. If there is an s^*-t^* flow of value D in G', edges incident on s^* and on t^* must be saturated with flow. Deleting these edges from G' yields a feasible circulation in G. (Prove it yourself.)

Computing a Feasible Circulation



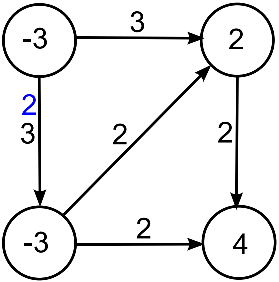
- We will look for a maximum s^*-t^* flow f in G'; $\nu(f) \leq D$.
- Circulation $\Rightarrow$ flow. If there is a feasible circulation, we send $-d_v$ units of flow along each edge (s^*, v) and d_v units of flow along each edge (v, t^*). The value of this flow is D. (Prove it yourself.)
- Flow $\Rightarrow$ circulation. If there is an s^*-t^* flow of value D in G', edges incident on s^* and on t^* must be saturated with flow. Deleting these edges from G' yields a feasible circulation in G. (Prove it yourself.)
- We have proved that there is a feasible circulation with demands in G iff the maximum s^*-t^* flow in G' has value D.

Circulation with Demands and Lower Bounds



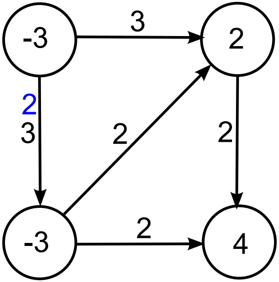
- We want to force the flow to use certain edges.

Circulation with Demands and Lower Bounds



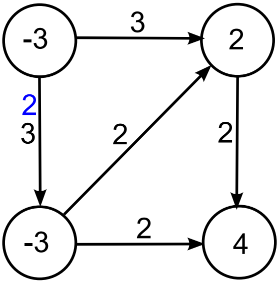
- We want to force the flow to use certain edges.
- We are given a graph $G(V, E)$ with a capacity $c(e)$ and a lower bound $0 \leq l(e) \leq c(e)$ on each edge and a demand d_v on each vertex.

Circulation with Demands and Lower Bounds



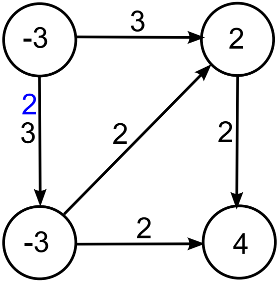
- We want to force the flow to use certain edges.
- We are given a graph $G(V, E)$ with a capacity $c(e)$ and a lower bound $0 \leq l(e) \leq c(e)$ on each edge and a demand d_v on each vertex.
- A *circulation* with demands and lower bounds is a function $f : E \to \mathbb{R}^+$ that satisfies

Circulation with Demands and Lower Bounds



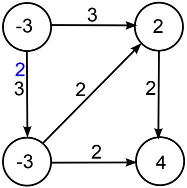
- We want to force the flow to use certain edges.
- We are given a graph $G(V, E)$ with a capacity $c(e)$ and a lower bound $0 \leq l(e) \leq c(e)$ on each edge and a demand d_v on each vertex.
- A *circulation* with demands and lower bounds is a function $f : E \to \mathbb{R}^+$ that satisfies
  (i) (*Capacity conditions*) For each $e \in E$, $l(e) \leq f(e) \leq c(e)$.
  (ii) (*Demand conditions*) For each node v, $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$.

Circulation with Demands and Lower Bounds



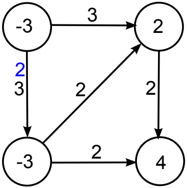
- We want to force the flow to use certain edges.
- We are given a graph $G(V, E)$ with a capacity $c(e)$ and a lower bound $0 \leq l(e) \leq c(e)$ on each edge and a demand d_v on each vertex.
- A *circulation* with demands and lower bounds is a function $f : E \rightarrow \mathbb{R}^+$ that satisfies
  (i) (*Capacity conditions*) For each $e \in E$, $l(e) \leq f(e) \leq c(e)$.
  (ii) (*Demand conditions*) For each node v, $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$.
- Is there a feasible circulation?

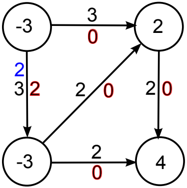
Algorithm for Circulation with Lower Bounds



- Strategy is to reduce the problem to one with no lower bounds on edges.

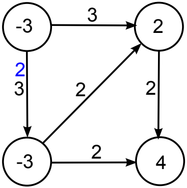
Algorithm for Circulation with Lower Bounds

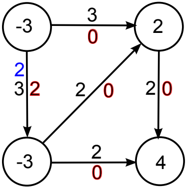




- Strategy is to reduce the problem to one with no lower bounds on edges.
- Suppose we define a circulation f_0 that satisfies lower bounds on all edges, i.e., set $f_0(e) = l(e)$ for all $e \in E$. What can go wrong?

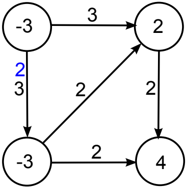
Algorithm for Circulation with Lower Bounds

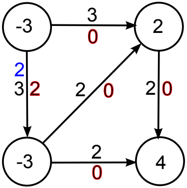


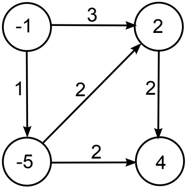


- Strategy is to reduce the problem to one with no lower bounds on edges.
- Suppose we define a circulation f_0 that satisfies lower bounds on all edges, i.e., set $f_0(e) = l(e)$ for all $e \in E$. What can go wrong?
- Demand conditions may be violated. Let $L_v = f_0^{\text{in}}(v) - f_0^{\text{out}}(v) = \sum_{e \text{ into } v} l(e) - \sum_{e \text{ out of } v} l(e)$.

Algorithm for Circulation with Lower Bounds

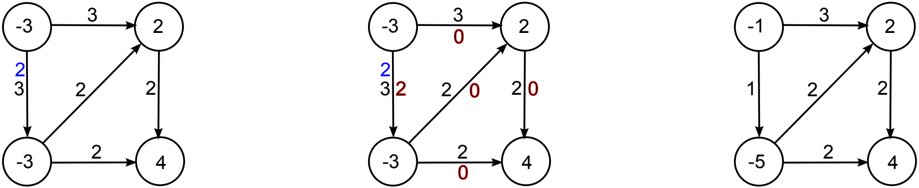






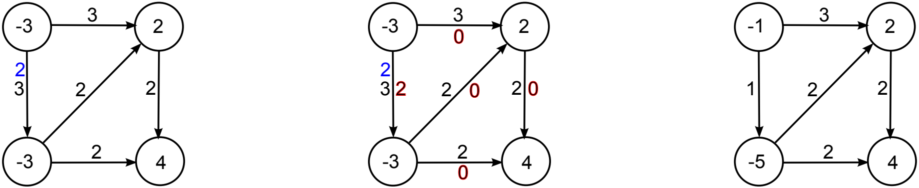
- Strategy is to reduce the problem to one with no lower bounds on edges.
- Suppose we define a circulation f_0 that satisfies lower bounds on all edges, i.e., set $f_0(e) = l(e)$ for all $e \in E$. What can go wrong?
- Demand conditions may be violated. Let $L_v = f_0^{\text{in}}(v) - f_0^{\text{out}}(v) = \sum_{e \text{ into } v} l(e) - \sum_{e \text{ out of } v} l(e)$.
- If $L_v \neq d_v$, we can superimpose a circulation f_1 on top of f_0 such that $f_1^{\text{in}}(v) - f_1^{\text{out}}(v) = d_v - L_v$.

Algorithm for Circulation with Lower Bounds



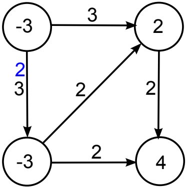
- Strategy is to reduce the problem to one with no lower bounds on edges.
- Suppose we define a circulation f_0 that satisfies lower bounds on all edges, i.e., set $f_0(e) = l(e)$ for all $e \in E$. What can go wrong?
- Demand conditions may be violated. Let $L_v = f_0^{\text{in}}(v) - f_0^{\text{out}}(v) = \sum_{e \text{ into } v} l(e) - \sum_{e \text{ out of } v} l(e)$.
- If $L_v \neq d_v$, we can superimpose a circulation f_1 on top of f_0 such that $f_1^{\text{in}}(v) - f_1^{\text{out}}(v) = d_v - L_v$.
- How much capacity do we have left on each edge?

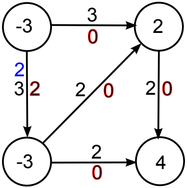
Algorithm for Circulation with Lower Bounds

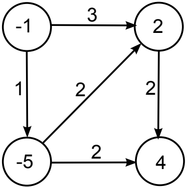


- Strategy is to reduce the problem to one with no lower bounds on edges.
- Suppose we define a circulation f_0 that satisfies lower bounds on all edges, i.e., set $f_0(e) = l(e)$ for all $e \in E$. What can go wrong?
- Demand conditions may be violated. Let $L_v = f_0^{\text{in}}(v) - f_0^{\text{out}}(v) = \sum_{e \text{ into } v} l(e) - \sum_{e \text{ out of } v} l(e)$.
- If $L_v \neq d_v$, we can superimpose a circulation f_1 on top of f_0 such that $f_1^{\text{in}}(v) - f_1^{\text{out}}(v) = d_v - L_v$.
- How much capacity do we have left on each edge? $c(e) - l(e)$.

Algorithm for Circulation with Lower Bounds







- Strategy is to reduce the problem to one with no lower bounds on edges.
- Suppose we define a circulation f_0 that satisfies lower bounds on all edges, i.e., set $f_0(e) = l(e)$ for all $e \in E$. What can go wrong?
- Demand conditions may be violated. Let $L_v = f_0^{\text{in}}(v) - f_0^{\text{out}}(v) = \sum_{e \text{ into } v} l(e) - \sum_{e \text{ out of } v} l(e)$.
- If $L_v \neq d_v$, we can superimpose a circulation f_1 on top of f_0 such that $f_1^{\text{in}}(v) - f_1^{\text{out}}(v) = d_v - L_v$.
- How much capacity do we have left on each edge? $c(e) - l(e)$.
- Approach: define a new graph G' with the same nodes and edges: each edge e has lower bound 0, capacity $c(e) - l(e)$; demand of each node v is $d_v - L_v$.
- Claim: there is a feasible circulation in G iff there is a feasible circulation in G'. Read the proof in the textbook.

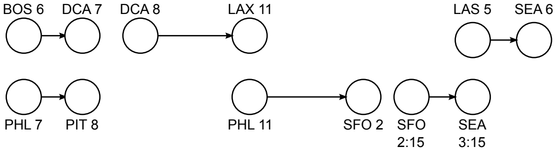
Airline Scheduling

- Airlines face very complex computational problems.
- Produce schedules for thousands of routes.
- Make these schedules efficient in terms of crew allocation, equipment usage, fuel costs, customer satisfaction, etc.

Airline Scheduling

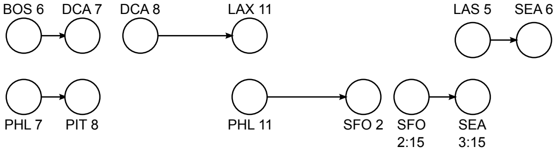
- Airlines face very complex computational problems.
- Produce schedules for thousands of routes.
- Make these schedules efficient in terms of crew allocation, equipment usage, fuel costs, customer satisfaction, etc.
- Modelling these problems realistically is out of the scope of the course.
- We will focus on a "toy" problem that cleanly captures some of the resource allocation problems they have to deal with.

Creating Flight Schedules



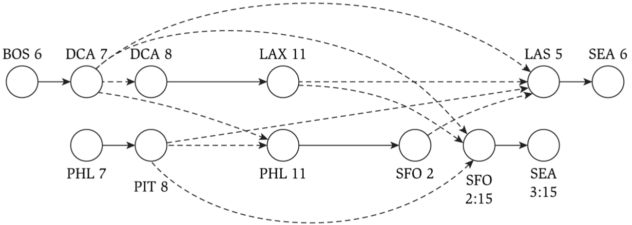
- Desire to serve m specific flight segments.
- Each flight segment (or flight) specified by four parameters: origin airport, destination airport, departure time, arrival time.

Creating Flight Schedules



- Desire to serve m specific flight segments.
- Each flight segment (or flight) specified by four parameters: origin airport, destination airport, departure time, arrival time.
- We can use a single plane for flight i and later for flight j if
  - (i) the destination of i is the same as the origin of j and there is enough time to perform maintenance on the plane between the two flights, or
  - (ii) we can add a flight that takes the plane from the destination of i to the origin of j with enough time for maintenance.
- Goal is to schedule all m flights using at most k planes.

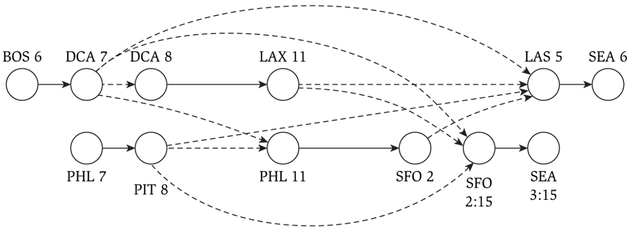
Reachability



(b)

- Flight j is *reachable* from flight i if the same plane can be used for both flights subject to the constraints described earlier.
- Assume input includes pairs (i, j) of reachable flights, i.e., in each pair j is reachable from i.
  - Pairs form a

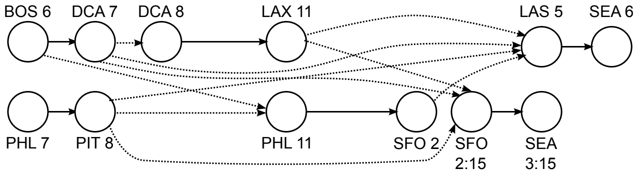
Reachability



(b)

- Flight j is *reachable* from flight i if the same plane can be used for both flights subject to the constraints described earlier.
- Assume input includes pairs (i, j) of reachable flights, i.e., in each pair j is reachable from i.
  - Pairs form a DAG.
  - *Flights* are reachable from one another, not *airports*.
  - Construction of reachable pairs will take maintenance time into account.
  - Definition of reachability can be more complex; input pairs can encode this complexity.

The Airline Scheduling Problem



Airline Scheduling

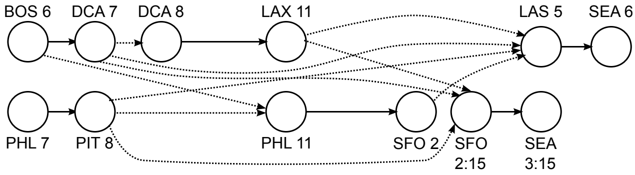
INSTANCE: Set S of m flight segments (u_i, v_i), $1 \leq i \leq m$, a set R of reachable pairs of flights (i, j), $1 \leq i, j \leq m$, and an integer bound k

SOLUTION: *Feasible scheduling*:

(a) Set T of $n \geq 0$ new flight segments (u_j, v_j), $1 \leq j \leq n$ and

(b) A partition of $S \cup T$ into at most k sequences such that in each sequence, flight i is reachable from flight $i-1$, for all $1 < i \leq l$, where l is the length of the sequence.

The Airline Scheduling Problem



AIRLINE SCHEDULING

INSTANCE: Set S of m flight segments (u_i, v_i), $1 \leq i \leq m$, a set R of reachable pairs of flights (i, j), $1 \leq i, j \leq m$, and an integer bound k

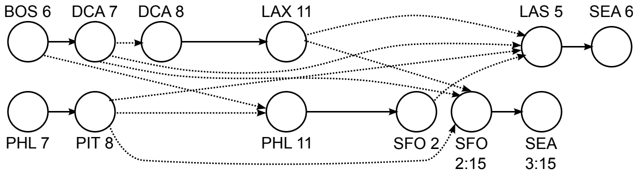
SOLUTION: *Feasible scheduling*:

(a) Set T of $n \geq 0$ new flight segments (u_j, v_j), $1 \leq j \leq n$ and

(b) A partition of $S \cup T$ into at most k sequences such that in each sequence, flight i is reachable from flight $i - 1$, for all $1 < i \leq l$, where l is the length of the sequence.

- Where are flight departure and arrival times in the input?

The Airline Scheduling Problem



AIRLINE SCHEDULING

INSTANCE: Set S of m flight segments (u_i, v_i), $1 \leq i \leq m$, a set R of reachable pairs of flights (i, j), $1 \leq i, j \leq m$, and an integer bound k

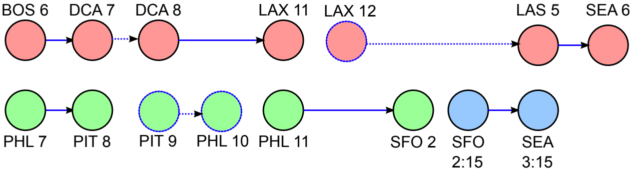
SOLUTION: *Feasible scheduling*:

(a) Set T of $n \geq 0$ new flight segments (u_j, v_j), $1 \leq j \leq n$ and

(b) A partition of $S \cup T$ into at most k sequences such that in each sequence, flight i is reachable from flight $i - 1$, for all $1 < i \leq l$, where l is the length of the sequence.

- Where are flight departure and arrival times in the input? In a flight segment, u_i specifies both origin airport and departure time; v_i specifies both arrival airport and arrival time.

The Airline Scheduling Problem



The dotted circles are meant only to illustrate the new flights added.

AIRLINE SCHEDULING

INSTANCE: Set S of m flight segments (u_i, v_i), $1 \leq i \leq m$, a set R of reachable pairs of flights (i, j), $1 \leq i, j \leq m$, and an integer bound k

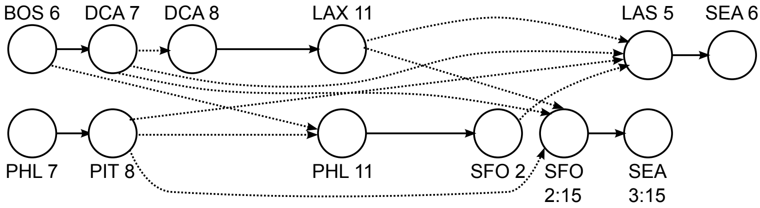
SOLUTION: *Feasible scheduling*:

(a) Set T of $n \geq 0$ new flight segments (u_j, v_j), $1 \leq j \leq n$ and

(b) A partition of $S \cup T$ into at most k sequences such that in each sequence, flight i is reachable from flight $i - 1$, for all $1 < i \leq l$, where l is the length of the sequence.

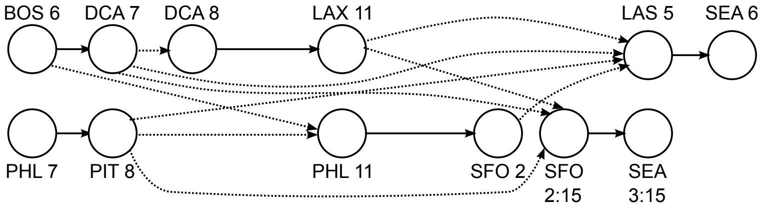
- Where are flight departure and arrival times in the input? In a flight segment, u_i specifies both origin airport and departure time; v_i specifies both arrival airport and arrival time.

Intuition Underlying Algorithm



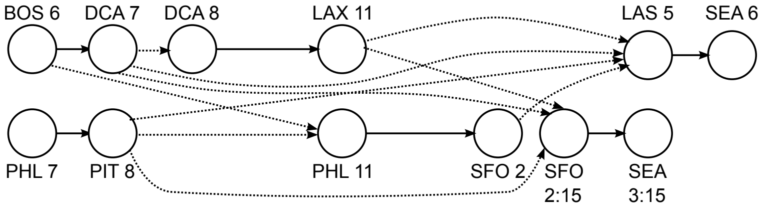
- Nodes in the flow network are airports.
- Planes correspond to units of flow.

Intuition Underlying Algorithm



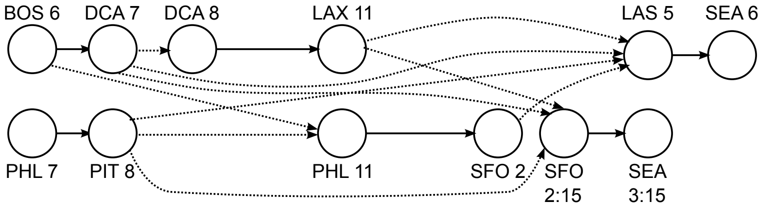
- Nodes in the flow network are airports.
- Planes correspond to units of flow.
- Each flight corresponds to an edge. How do we ensure each flight is served by exactly one plane?

Intuition Underlying Algorithm



- Nodes in the flow network are airports.
- Planes correspond to units of flow.
- Each flight corresponds to an edge. How do we ensure each flight is served by exactly one plane? Lower bound of 1 and a capacity of 1.

Intuition Underlying Algorithm



- Nodes in the flow network are airports.
- Planes correspond to units of flow.
- Each flight corresponds to an edge. How do we ensure each flight is served by exactly one plane? Lower bound of 1 and a capacity of 1.
- How do we represent reachability? If (i, j) is a reachable pair, there is an edge from v_i to u_j with lower bound of 0 and a capacity of 1.

Designing the Flow Network

Nodes:
- For each flight i, graph G has two nodes u_i and v_i.
- G also contains a distinct source node s and a sink node t.

Designing the Flow Network

Nodes:
- For each flight i, graph G has two nodes u_i and v_i.
- G also contains a distinct source node s and a sink node t.

Edges:

Serve each flight: For each $i \in S$ (flight), G contains an edge directed from u_i to v_i with a lower bound of 1 and a capacity of 1.

Same plane for flights i and j: For each $(i, j) \in R$, G contains an edge directed from v_i to u_j with a lower bound of 0 and a capacity of 1.

Start a plane with any flight: For each $i \in S$, G contains an edge directed from s to u_i with a lower bound of 0 and a capacity of 1.

End a plane with any flight: For each $j \in S$, G contains an edge directed from v_j to t with a lower bound of 0 and a capacity of 1.

Excess planes: G contains an edge directed from s to t with lower bound 0 and capacity k.

Designing the Flow Network

Nodes:
- For each flight i, graph G has two nodes u_i and v_i.
- G also contains a distinct source node s and a sink node t.

Edges:

Serve each flight: For each $i \in S$ (flight), G contains an edge directed from u_i to v_i with a lower bound of 1 and a capacity of 1.

Same plane for flights i and j: For each $(i, j) \in R$, G contains an edge directed from v_i to u_j with a lower bound of 0 and a capacity of 1.

Start a plane with any flight: For each $i \in S$, G contains an edge directed from s to u_i with a lower bound of 0 and a capacity of 1.

End a plane with any flight: For each $j \in S$, G contains an edge directed from v_j to t with a lower bound of 0 and a capacity of 1.

Excess planes: G contains an edge directed from s to t with lower bound 0 and capacity k.

Demands: Node s has demand $-k$, node t has demand k, all other nodes have demand 0.

Designing the Flow Network

Nodes:
- For each flight i, graph G has two nodes u_i and v_i.
- G also contains a distinct source node s and a sink node t.

Edges:

Serve each flight: For each $i \in S$ (flight), G contains an edge directed from u_i to v_i with a lower bound of 1 and a capacity of 1.

Same plane for flights i and j: For each $(i, j) \in R$, G contains an edge directed from v_i to u_j with a lower bound of 0 and a capacity of 1.

Start a plane with any flight: For each $i \in S$, G contains an edge directed from s to u_i with a lower bound of 0 and a capacity of 1.

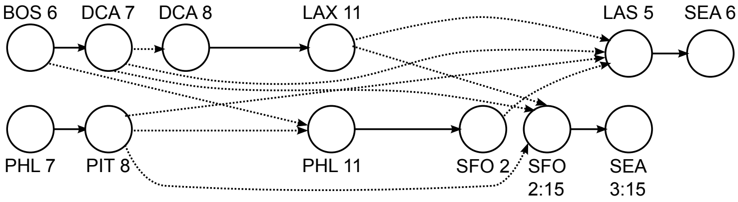
End a plane with any flight: For each $j \in S$, G contains an edge directed from v_j to t with a lower bound of 0 and a capacity of 1.

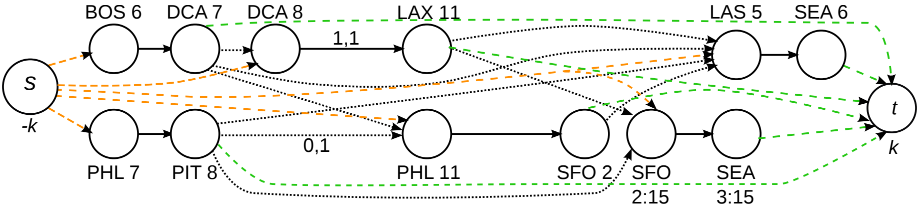
Excess planes: G contains an edge directed from s to t with lower bound 0 and capacity k.

Demands: Node s has demand $-k$, node t has demand k, all other nodes have demand 0.

Goal: Compute whether G has a feasible circulation.

Example of Circulation Formulation



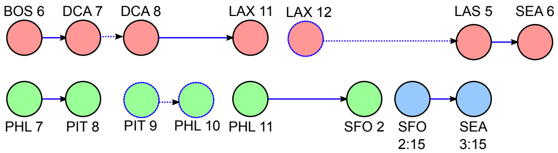


The image does not show the edge between s and t.

Proof of Correctness: Part 1

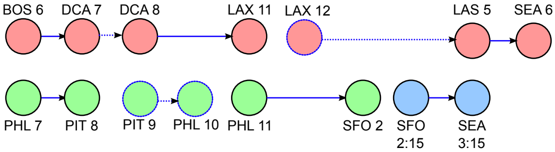
- Claim: We can schedule all flights in S using at most k planes iff G has a feasible circulation.

Proof of Correctness: Part 1



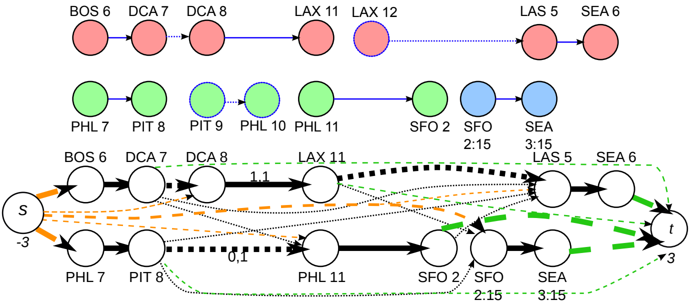
- Claim: We can schedule all flights in S using at most k planes iff G has a feasible circulation.
- Feasible schedule with $k' \leq k$ planes $\Rightarrow$ feasible circulation:

Proof of Correctness: Part 1



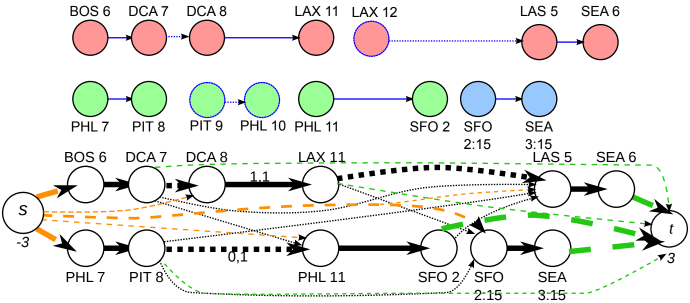
- Claim: We can schedule all flights in S using at most k planes iff G has a feasible circulation.
- Feasible schedule with $k' \leq k$ planes $\Rightarrow$ feasible circulation:
  - Each plane $l, 1 \leq l \leq k'$ flies along a particular path P_l of flights unique to that plane, starting at city s_l and ending at city t_l.
  - Send one unit of flow along the edges of that path P and along the edges (s, s_l) and (t_l, t).

Proof of Correctness: Part 1



- Claim: We can schedule all flights in S using at most k planes iff G has a feasible circulation.
- Feasible schedule with $k' \leq k$ planes $\Rightarrow$ feasible circulation:
  - Each plane $l, 1 \leq l \leq k'$ flies along a particular path P_l of flights unique to that plane, starting at city s_l and ending at city t_l.
  - Send one unit of flow along the edges of that path P and along the edges (s, s_l) and (t_l, t).
  - To satisfy excess demands at s and t, send $k - k'$ units of flow along (s, t).

Proof of Correctness: Part 1

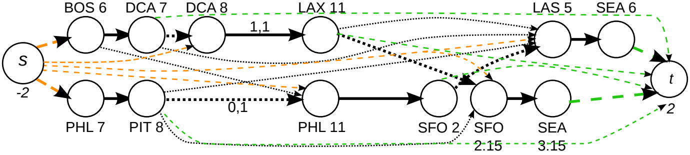


- Claim: We can schedule all flights in S using at most k planes iff G has a feasible circulation.
- Feasible schedule with $k' \leq k$ planes $\Rightarrow$ feasible circulation:
  - Each plane $l, 1 \leq l \leq k'$ flies along a particular path P_l of flights unique to that plane, starting at city s_l and ending at city t_l.
  - Send one unit of flow along the edges of that path P and along the edges (s, s_l) and (t_l, t).
  - To satisfy excess demands at s and t, send $k - k'$ units of flow along (s, t).
  - Why does the resulting circulation satisfy all demand, lower bound, and capacity constraints?

Proof of Correctness: Part 2

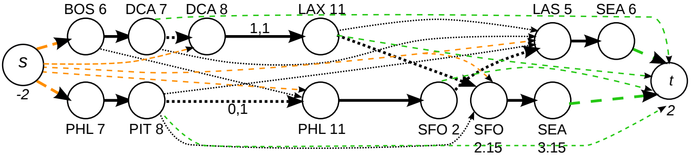
- Claim: We can schedule all flights in S using at most k planes iff G has a feasible circulation.

Proof of Correctness: Part 2



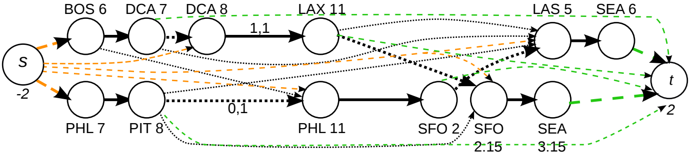
- Claim: We can schedule all flights in S using at most k planes iff G has a feasible circulation.
- Feasible circulation $\Rightarrow$ feasible schedule:

Proof of Correctness: Part 2



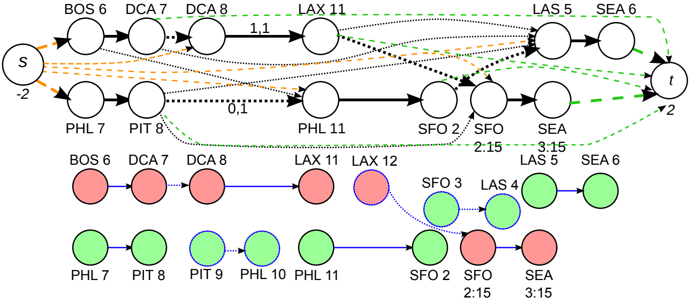
- Claim: We can schedule all flights in S using at most k planes iff G has a feasible circulation.
- Feasible circulation $\Rightarrow$ feasible schedule:
  - Flow on each edge must be 0 or 1. Flow on the edges for flights must be 1.

Proof of Correctness: Part 2



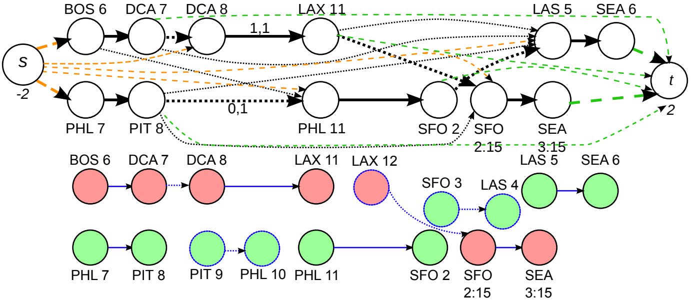
- Claim: We can schedule all flights in S using at most k planes iff G has a feasible circulation.
- Feasible circulation $\Rightarrow$ feasible schedule:
  - Flow on each edge must be 0 or 1. Flow on the edges for flights must be 1.
  - Suppose total flow out of s other than the edge (s, t) is $k' \leq k$.

Proof of Correctness: Part 2



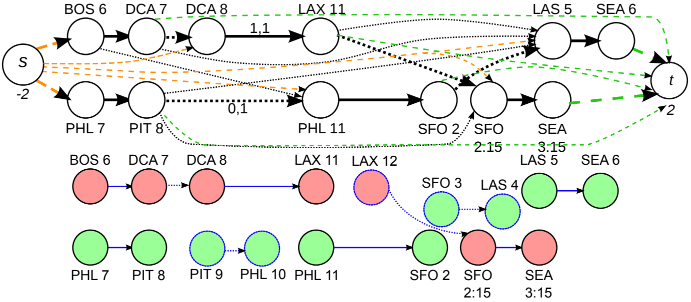
- Claim: We can schedule all flights in S using at most k planes iff G has a feasible circulation.
- Feasible circulation $\Rightarrow$ feasible schedule:
  - Flow on each edge must be 0 or 1. Flow on the edges for flights must be 1.
  - Suppose total flow out of s other than the edge (s, t) is $k' \leq k$.
  - Claim: at most k' planes suffice to satisfy all flights.

Proof of Correctness: Part 2



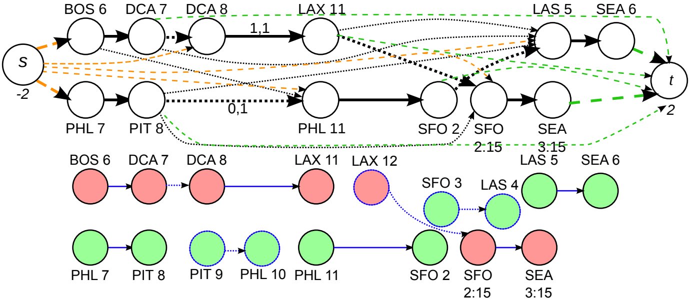
- Claim: We can schedule all flights in S using at most k planes iff G has a feasible circulation.
- Feasible circulation $\Rightarrow$ feasible schedule:
  - Flow on each edge must be 0 or 1. Flow on the edges for flights must be 1.
  - Suppose total flow out of s other than the edge (s,t) is $k' \leq k$.
  - Claim: at most k' planes suffice to satisfy all flights.
  - Convert set of edges that carry flow into k' edge-disjoint s-t paths.

Proof of Correctness: Part 2



- Claim: We can schedule all flights in S using at most k planes iff G has a feasible circulation.
- Feasible circulation $\Rightarrow$ feasible schedule:
  - Flow on each edge must be 0 or 1. Flow on the edges for flights must be 1.
  - Suppose total flow out of s other than the edge (s, t) is $k' \leq k$.
  - Claim: at most k' planes suffice to satisfy all flights.
  - Convert set of edges that carry flow into k' edge-disjoint s-t paths.
  - Each path starts at exactly one of the k' edges of the form (s, u), $u \neq t$ that carry flow. Use the proof for the edge-disjoint paths problem to compute path.

Proof of Correctness: Part 2



- Claim: We can schedule all flights in S using at most k planes iff G has a feasible circulation.
- Feasible circulation $\Rightarrow$ feasible schedule:
  - Flow on each edge must be 0 or 1. Flow on the edges for flights must be 1.
  - Suppose total flow out of s other than the edge (s, t) is $k' \leq k$.
  - Claim: at most k' planes suffice to satisfy all flights.
  - Convert set of edges that carry flow into k' edge-disjoint s-t paths.
  - Each path starts at exactly one of the k' edges of the form (s, u), $u \neq t$ that carry flow. Use the proof for the edge-disjoint paths problem to compute path.
  - Output these paths. Paths define extra flight segments automatically.