Applications of Network Flow

T. M. Murali

April 12, 14, 2021

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.
  - Network connectivity.
  - Data mining.
  - Project selection.
  - Airline scheduling.
  - Baseball elimination.
  - Image segmentation.
  - Open-pit mining.
  - Network reliability.
  - Distributed computing.
  - Egalitarian stable matching.
  - Security of statistical data.
  - Network intrusion detection.
  - Multi-camera scene reconstruction.
  - Gene function prediction.

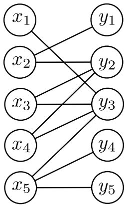
Maximum Flow and Minimum Cut

- Two rich algorithmic problems.
- Fundamental problems in combinatorial optimization.
- Beautiful mathematical duality between flows and cuts.
- Numerous non-trivial applications:
  - Bipartite matching.
  - Network connectivity.
  - Data mining.
  - Project selection.
  - Airline scheduling.
  - Baseball elimination.
  - Image segmentation.
  - 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

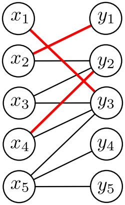
- Two rich algorithmic problems.
- Fundamental problems in combinatorial optimization.
- Beautiful mathematical duality between flows and cuts.
- Numerous non-trivial applications:
  - Bipartite matching.
  - Network connectivity.
  - Data mining.
  - Project selection.
  - Airline scheduling.
  - Baseball elimination.
  - Image segmentation.
  - 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



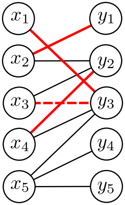
- *Bipartite Graph*: a graph $G(V, E)$ where
  1. $V = X \cup Y$, X and Y are disjoint and
  2. $E \subseteq X \times Y$.
- Bipartite graphs model situations in which objects are matched with or assigned to other objects: e.g., marriages, residents/hospitals, jobs/machines.

Matching in Bipartite Graphs



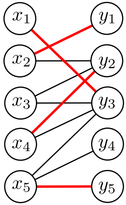
- *Bipartite Graph*: a graph $G(V, E)$ where
  1. $V = X \cup Y$, X and Y are disjoint and
  2. $E \subseteq X \times Y$.
- Bipartite graphs model situations in which objects are matched with or assigned to other objects: e.g., marriages, residents/hospitals, jobs/machines.
- A *matching* in a bipartite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



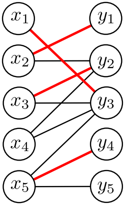
- *Bipartite Graph*: a graph $G(V, E)$ where
  1. $V = X \cup Y$, X and Y are disjoint and
  2. $E \subseteq X \times Y$.
- Bipartite graphs model situations in which objects are matched with or assigned to other objects: e.g., marriages, residents/hospitals, jobs/machines.
- A *matching* in a bipartite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



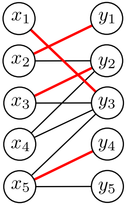
- *Bipartite Graph*: a graph $G(V, E)$ where
  1. $V = X \cup Y$, X and Y are disjoint and
  2. $E \subseteq X \times Y$.
- Bipartite graphs model situations in which objects are matched with or assigned to other objects: e.g., marriages, residents/hospitals, jobs/machines.
- A *matching* in a bipartite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



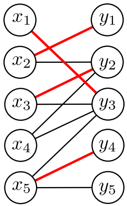
- *Bipartite Graph*: a graph $G(V, E)$ where
  1. $V = X \cup Y$, X and Y are disjoint and
  2. $E \subseteq X \times Y$.
- Bipartite graphs model situations in which objects are matched with or assigned to other objects: e.g., marriages, residents/hospitals, jobs/machines.
- A *matching* in a bipartite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



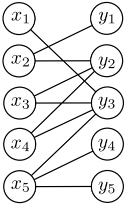
- *Bipartite Graph*: a graph $G(V, E)$ where
  1. $V = X \cup Y$, X and Y are disjoint and
  2. $E \subseteq X \times Y$.
- Bipartite graphs model situations in which objects are matched with or assigned to other objects: e.g., marriages, residents/hospitals, jobs/machines.
- A *matching* in a bipartite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

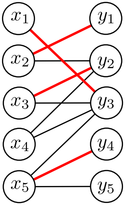
Matching in Bipartite Graphs



- *Bipartite Graph*: a graph $G(V, E)$ where
  1. $V = X \cup Y$, X and Y are disjoint and
  2. $E \subseteq X \times Y$.
- 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



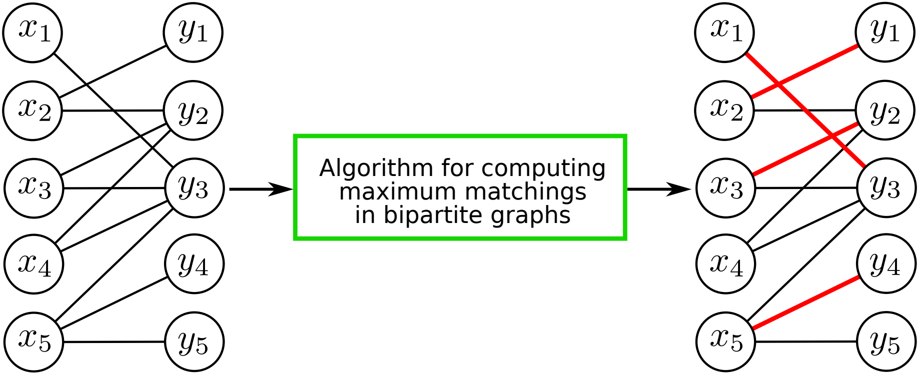


BIPARTITE MATCHING

INSTANCE: A Bipartite graph G.

SOLUTION: The matching of largest size in G.

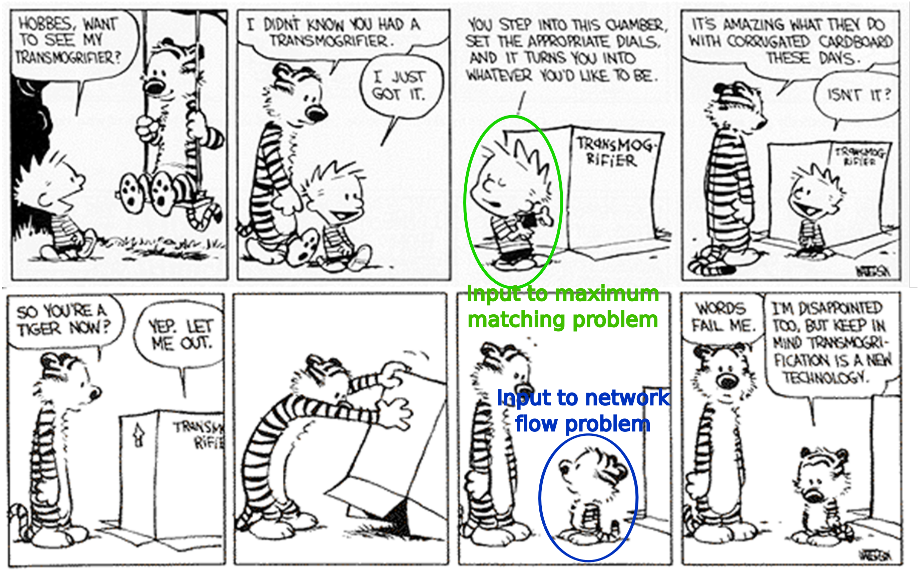
Normal Approach for Solving a Problem



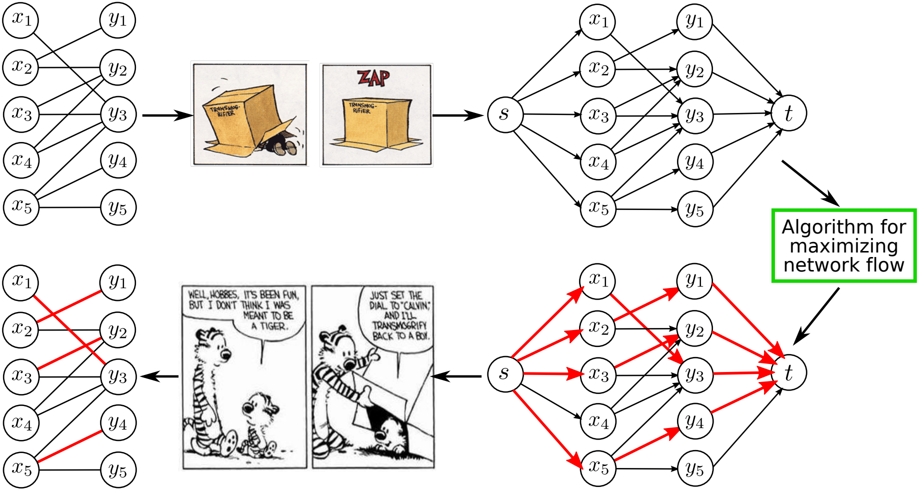
- Develop algorithm for computing maximum matchings in bipartite graphs.
- Prove that the algorithm is correct, i.e., for every possible inputs, it compute the size of the largest matching in the bipartite graph accurately.
- Analyze running time of the algorithm.

Alternative Approach for Solving a Problem

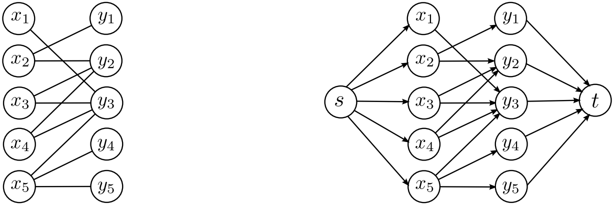
Alternative Approach for Solving a Problem



Alternative Approach for Solving a Problem



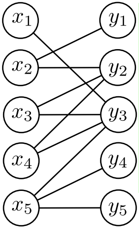
Algorithm for Bipartite Graph Matching



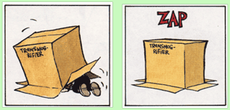
1. Convert G to a flow network G': direct edges from X to Y, add nodes s and t, connect s to each node in X, connect each node in Y to t, set all edge capacities to 1.
2. Compute the maximum flow in G'.
3. Convert the maximum flow in G' into a matching in G.

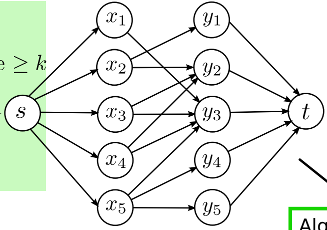
- Claim: the value of the maximum flow in G' equals the size of the maximum matching in G.
- In general, there is matching with size k in G if and only if there is a (integer-valued) flow of value k in G'.

Strategy for Proving Correctness

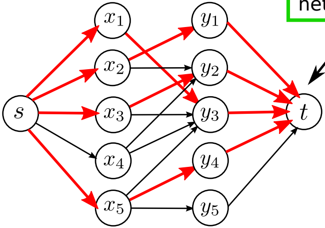


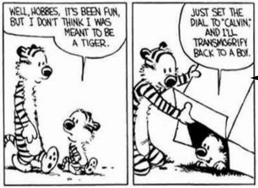
Matching $\Rightarrow$ flow:
If G has a matching with k edges,
then G' must have a flow with value $\geq k$

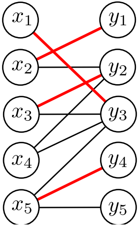




Algorithm for maximizing network flow

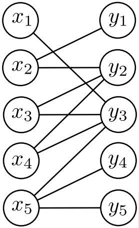


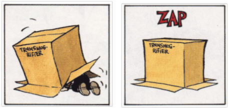


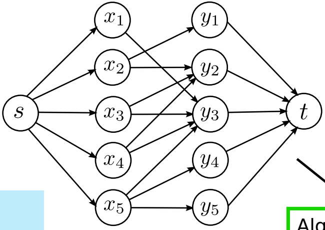


Preclude the possibility that G has a matching with k edges but G' has a flow of small value.

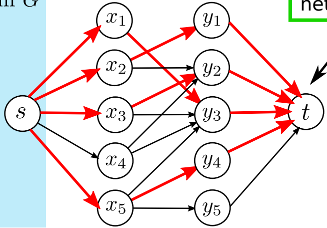
Strategy for Proving Correctness



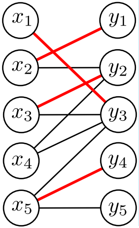




Algorithm for maximizing network flow

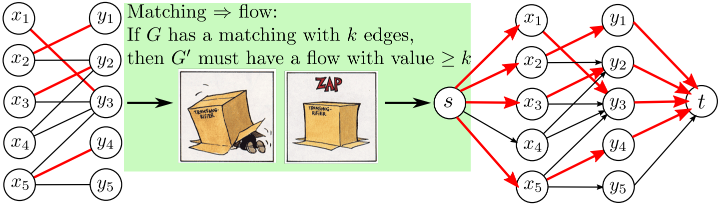


Flow $\Rightarrow$ matching:
If G' has a flow of size k,
then we can construct a matching in G with k edges



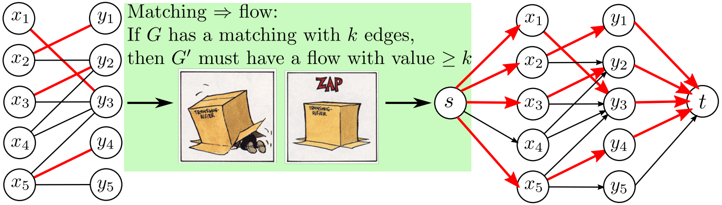
Preclude the possibility that G' has a flow of value k but we cannot construct a matching in G with k edges.

Correctness of Bipartite Graph Matching Algorithm



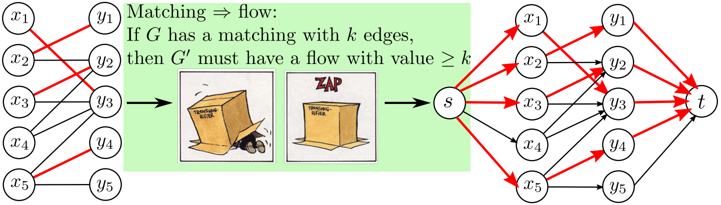
- Matching $\Rightarrow$ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.

Correctness of Bipartite Graph Matching Algorithm



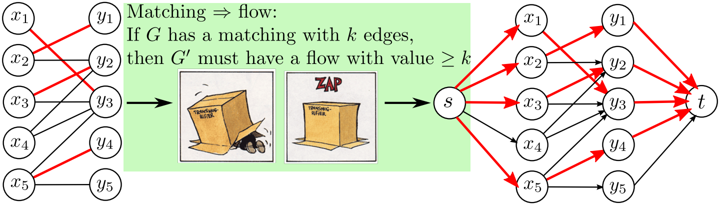
- Matching $\Rightarrow$ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.
- How do we construct this flow? Thought experiment.

Correctness of Bipartite Graph Matching Algorithm



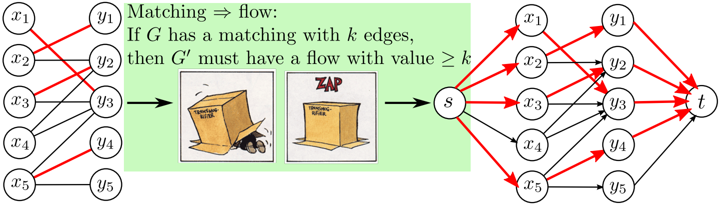
- Matching ⇒ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.
- How do we construct this flow? Thought experiment.
  - Consider every edge (u, v) in the matching: $u \in X$ and $v \in Y$.
  - Send one unit of flow along the path $s \to u \to v \to t$.

Correctness of Bipartite Graph Matching Algorithm



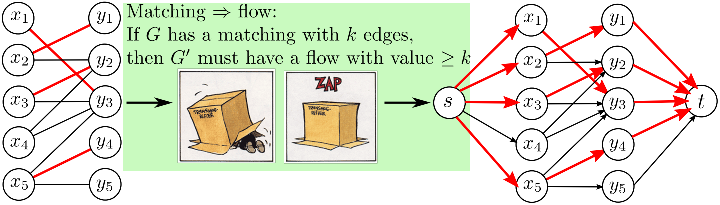
- Matching $\Rightarrow$ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.
- How do we construct this flow? Thought experiment.
  - Consider every edge (u, v) in the matching: $u \in X$ and $v \in Y$.
  - Send one unit of flow along the path $s \to u \to v \to t$.
- Why have we constructed a flow?

Correctness of Bipartite Graph Matching Algorithm



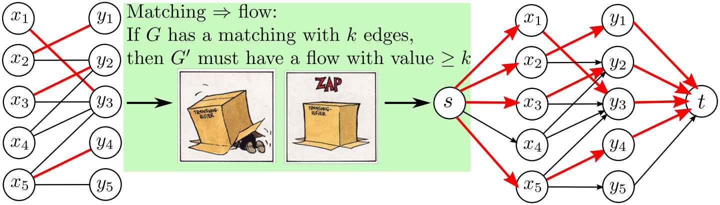
- Matching $\Rightarrow$ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.
- How do we construct this flow? Thought experiment.
  - Consider every edge (u, v) in the matching: $u \in X$ and $v \in Y$.
  - Send one unit of flow along the path $s \rightarrow u \rightarrow v \rightarrow t$.
- Why have we constructed a flow?
  - Capacity constraint:
  - Conservation constraint:

Correctness of Bipartite Graph Matching Algorithm



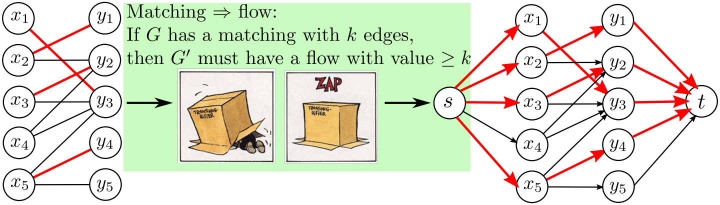
- Matching $\Rightarrow$ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.
- How do we construct this flow? Thought experiment.
  - Consider every edge (u, v) in the matching: $u \in X$ and $v \in Y$.
  - Send one unit of flow along the path $s \rightarrow u \rightarrow v \rightarrow t$.
- Why have we constructed a flow?
  - Capacity constraint: No edge receives a flow > 1 because we started with a matching.
  - Conservation constraint:

Correctness of Bipartite Graph Matching Algorithm



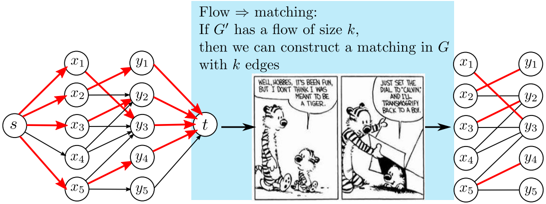
- Matching $\Rightarrow$ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.
- How do we construct this flow? Thought experiment.
  - Consider every edge (u, v) in the matching: $u \in X$ and $v \in Y$.
  - Send one unit of flow along the path $s \rightarrow u \rightarrow v \rightarrow t$.
- Why have we constructed a flow?
  - Capacity constraint: No edge receives a flow > 1 because we started with a matching.
  - Conservation constraint: Every node other than s and t has one incoming unit and one outgoing unit of flow because we started with a matching.

Correctness of Bipartite Graph Matching Algorithm



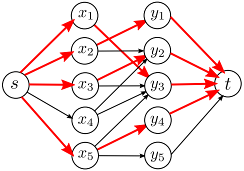
- Matching ⇒ flow: if there is a matching with k edges in G, there is an s-t flow of value k in G'.
- How do we construct this flow? Thought experiment.
  - Consider every edge (u, v) in the matching: $u \in X$ and $v \in Y$.
  - Send one unit of flow along the path $s \rightarrow u \rightarrow v \rightarrow t$.
- Why have we constructed a flow?
  - Capacity constraint: No edge receives a flow > 1 because we started with a matching.
  - Conservation constraint: Every node other than s and t has one incoming unit and one outgoing unit of flow because we started with a matching.
- What is the value of the flow? k, since exactly that many nodes out of s carry flow.

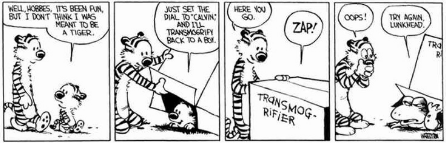
Correctness of Bipartite Graph Matching Algorithm

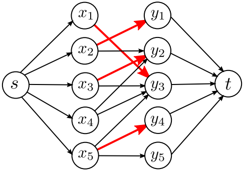


- Flow ⇒ matching: if there is a flow f' in G' with value k, there is a matching M in G with k edges.

Correctness of Bipartite Graph Matching Algorithm

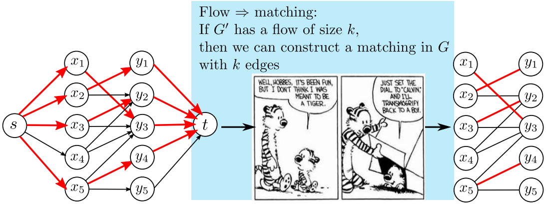






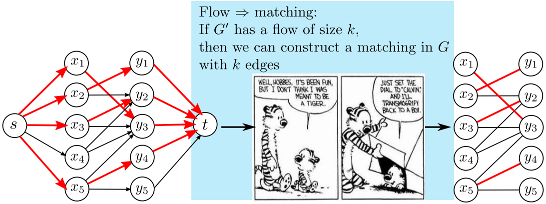
- Flow $\Rightarrow$ matching: if there is a flow f' in G' with value k, there is a matching M in G with k edges.

Correctness of Bipartite Graph Matching Algorithm



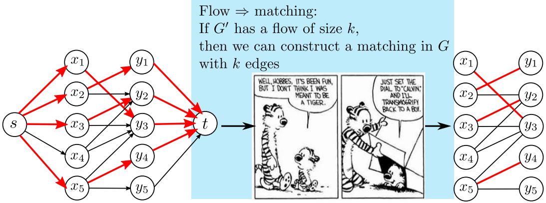
- Flow ⇒ matching: if there is a flow f' in G' with value k, there is a matching M in G with k edges.
  - There is an integer-valued flow f' of value k ⇒ flow along any edge is 0 or 1.

Correctness of Bipartite Graph Matching Algorithm



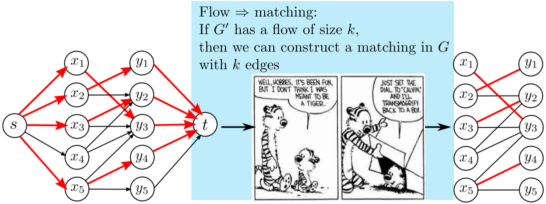
- Flow ⇒ matching: if there is a flow f' in G' with value k, there is a matching M in G with k edges.
  - There is an integer-valued flow f' of value k ⇒ flow along any edge is 0 or 1.
  - Let M be the set of edges not incident on s or t with flow equal to 1.

Correctness of Bipartite Graph Matching Algorithm



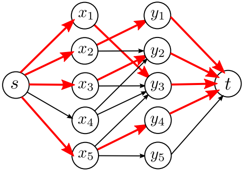
- Flow ⇒ matching: if there is a flow f' in G' with value k, there is a matching M in G with k edges.
  - There is an integer-valued flow f' of value k ⇒ flow along any edge is 0 or 1.
  - Let M be the set of edges not incident on s or t with flow equal to 1.
  - Claim: M contains k edges.

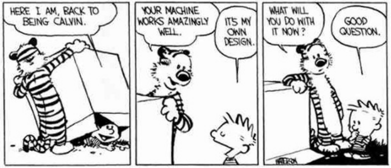
Correctness of Bipartite Graph Matching Algorithm

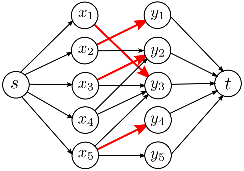


- Flow ⇒ matching: if there is a flow f' in G' with value k, there is a matching M in G with k edges.
  - There is an integer-valued flow f' of value k ⇒ flow along any edge is 0 or 1.
  - Let M be the set of edges not incident on s or t with flow equal to 1.
  - Claim: M contains k edges.
  - Claim: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

Correctness of Bipartite Graph Matching Algorithm

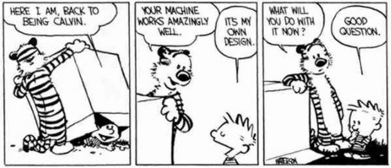






- Flow $\Rightarrow$ matching: if there is a flow f' in G' with value k, there is a matching M in G with k edges.
  - There is an integer-valued flow f' of value $k \Rightarrow$ flow along any edge is 0 or 1.
  - Let M be the set of edges not incident on s or t with flow equal to 1.
  - Claim: M contains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



- 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.

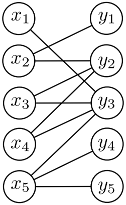
Running time of Bipartite Graph Matching Algorithm

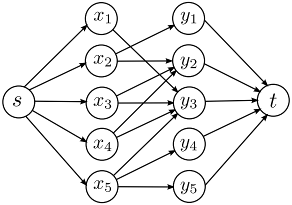
- Suppose G has m edges and n nodes in X and in Y.

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.

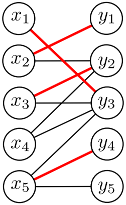
Bipartite Graphs without Perfect Matchings

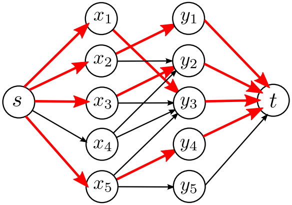




- How do we determine if a bipartite graph G has a perfect matching?

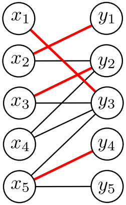
Bipartite Graphs without Perfect Matchings

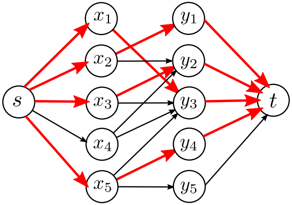




- How do we determine if a bipartite graph G has a perfect matching? Find the maximum matching and check if it is perfect.

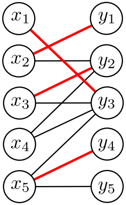
Bipartite Graphs without Perfect Matchings

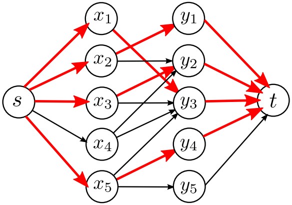




- How do we determine if a bipartite graph G has a perfect matching? Find the maximum matching and check if it is perfect.
- Suppose G has no perfect matching. Can we exhibit a short "certificate" of that fact? What can such certificates look like?

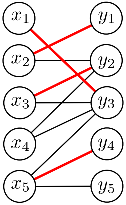
Bipartite Graphs without Perfect Matchings

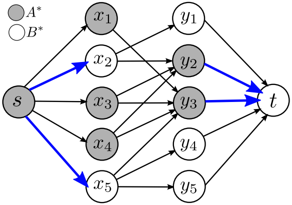




- How do we determine if a bipartite graph G has a perfect matching? Find the maximum matching and check if it is perfect.
- Suppose G has no perfect matching. Can we exhibit a short "certificate" of that fact? What can such certificates look like?
- G has no perfect matching iff

Bipartite Graphs without Perfect Matchings



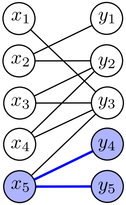


- How do we determine if a bipartite graph G has a perfect matching? Find the maximum matching and check if it is perfect.
- Suppose G has no perfect matching. Can we exhibit a short "certificate" of that fact? What can such certificates look like?
- G has no perfect matching iff there is a cut in G'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

Bipartite Graphs without Perfect Matchings

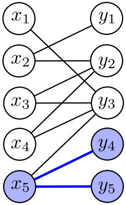
- We would like the certificate in terms of G.

Bipartite Graphs without Perfect Matchings



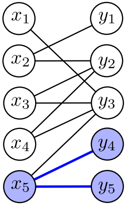
- We would like the certificate in terms of G.
  - For example, two nodes in Y with one incident edge each with the same neighbour in X.

Bipartite Graphs without Perfect Matchings



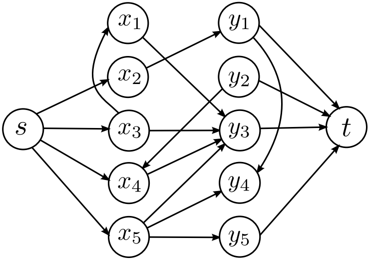
- We would like the certificate in terms of G.
  - For example, two nodes in Y with one incident edge each with the same neighbour in X.
  - Generally, a subset $A \subseteq X$ with neighbours $\Gamma(A) \subseteq Y$, such that $|A| > |\Gamma(A)|$.
- Hall's Theorem: Let $G(X \cup Y, E)$ be a bipartite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

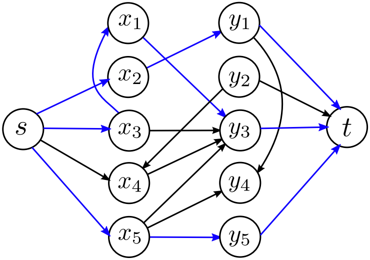
Bipartite Graphs without Perfect Matchings



- 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 $|A| > |\Gamma(A)|$. We can compute a perfect matching or such a subset in $O(mn)$ time. Read proof in the textbook.

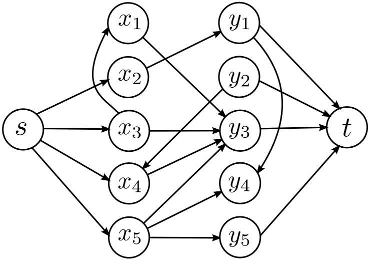
Edge-Disjoint Paths

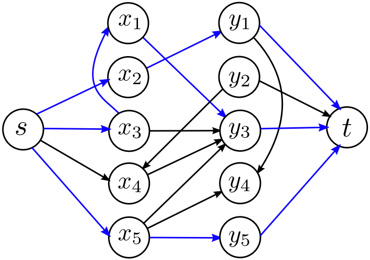




- A set of paths in a graph G is *edge disjoint* if each edge in G appears in at most one path.

Edge-Disjoint Paths





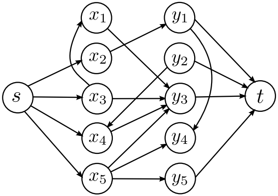
- A set of paths in a graph G is *edge disjoint* if each edge in G appears in at most one path.

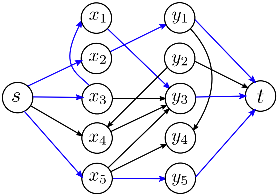
DIRECTED EDGE-DISJOINT PATHS

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.

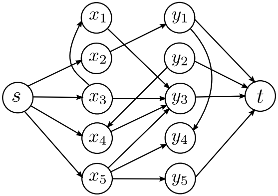
Mapping to the Max-Flow Problem

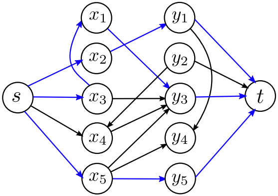




- Convert G into a flow network:

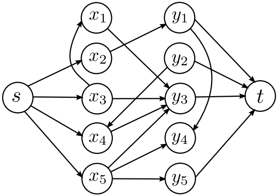
Mapping to the Max-Flow Problem

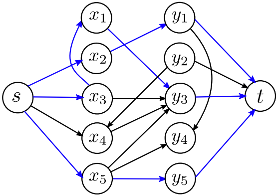




- Convert G into a flow network: s is the source, t is the sink, each edge has capacity 1.
- Claim: There are k edge-disjoint paths from s to t in a directed graph G if and only if there is a s-t flow in G with value $\geq k$.

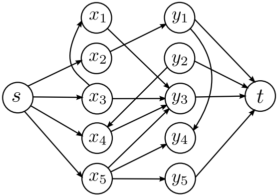
Mapping to the Max-Flow Problem

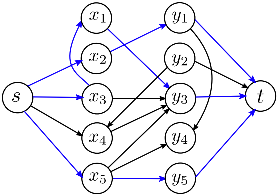




- Convert G into a flow network: s is the source, t is the sink, each edge has capacity 1.
- Claim: There are k edge-disjoint paths from s to t in a directed graph G if and only if there is a s-t flow in G with value $\geq k$.
- Paths $\Rightarrow$ flow: if there are k edge-disjoint paths from s to t,

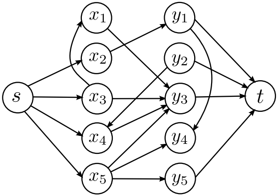
Mapping to the Max-Flow Problem

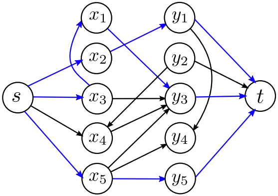




- Convert G into a flow network: s is the source, t is the sink, each edge has capacity 1.
- Claim: There are k edge-disjoint paths from s to t in a directed graph G if and only if there is a s-t flow in G with value $\geq k$.
- Paths $\Rightarrow$ flow: if there are k edge-disjoint paths from s to t,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

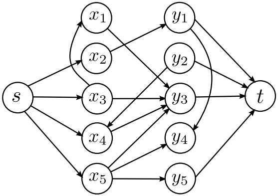
Mapping to the Max-Flow Problem

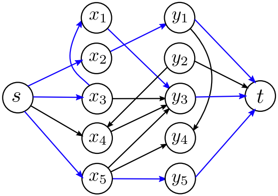




- Convert G into a flow network: s is the source, t is the sink, each edge has capacity 1.
- Claim: There are k edge-disjoint paths from s to t in a directed graph G if and only if there is a s-t flow in G with value $\geq k$.
- Paths $\Rightarrow$ flow: if there are k edge-disjoint paths from s to t,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

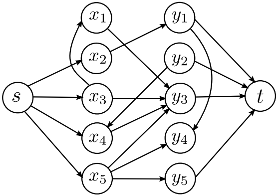
Mapping to the Max-Flow Problem

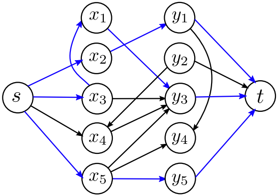




- Convert G into a flow network: s is the source, t is the sink, each edge has capacity 1.
- Claim: There are k edge-disjoint paths from s to t in a directed graph G if and only if there is a s-t flow in G with value $\geq k$.
- Paths $\Rightarrow$ flow: if there are k edge-disjoint paths from s to t, 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 there is a s-t flow in G with value $\geq k$.
- 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 $\nu(f) = k$, then the set of edges with flow $f(e) = 1$ contains a set of k edge-disjoint paths.

Completing the Proof

- Claim: if f is a 0-1 valued flow of value $\nu(f) = k$, then the set of edges with flow $f(e) = 1$ contains a set of k 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 $\nu(f) = k$, then the set of edges with flow $f(e) = 1$ contains a set of k 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 $\nu(f') < k$ carrying flow on $\kappa(f') < \kappa(f)$ edges or
  (b) value $\nu(f') = k$ 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 $\nu(f) = k$, then the set of edges with flow $f(e) = 1$ contains a set of k 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 $\nu(f') < k$ carrying flow on $\kappa(f') < \kappa(f)$ edges or

  (b) value $\nu(f') = k$ 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 k s-t paths from f. Work out by hand.

Completing the Proof

- Claim: if f is a 0-1 valued flow of value $\nu(f) = k$, then the set of edges with flow $f(e) = 1$ contains a set of k 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 $\nu(f') < k$ carrying flow on $\kappa(f') < \kappa(f)$ edges or
  (b) value $\nu(f') = k$ 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 k s-t paths from f. Work out by hand.
- 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., $\nu(f') < k$ 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 $\nu(f') = k$ but carrying flow on fewer edges, i.e., $\kappa(f') < \kappa(f)$ edges.
- You can combine both situations in the inductive hypothesis.

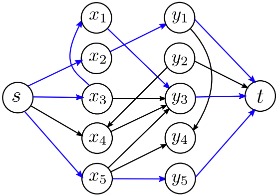
Running Time of the Edge-Disjoint Paths Algorithm

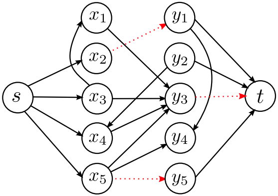
- Given a flow of value k, how quickly can we determine the k edge-disjoint paths?

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.

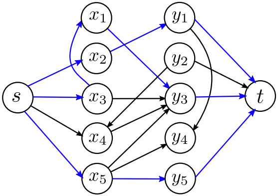
Certificate for Edge-Disjoint Paths Algorithm

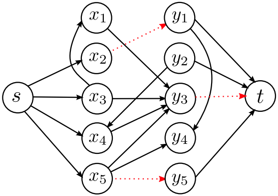




- A set $F \subseteq E$ of edge separates s and t if the graph $(V, E - F)$ contains no s-t paths.

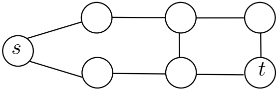
Certificate for Edge-Disjoint Paths Algorithm





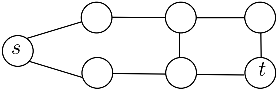
- A set $F \subseteq E$ of edge separates s and t if the graph $(V, E - F)$ contains no s-t paths.
- Menger's Theorem: In every directed graph with nodes s and t, the maximum number of edge-disjoint s-t paths is equal to the minimum number of edges whose removal disconnects s from t.

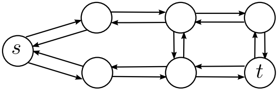
Edge-Disjoint Paths in Undirected Graphs



- Can extend the theorem to *undirected* graphs.

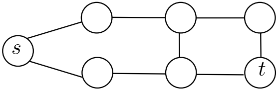
Edge-Disjoint Paths in Undirected Graphs

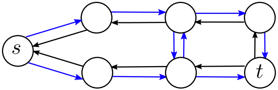




- Can extend the theorem to *undirected* graphs.
- Replace each edge with two directed edges of capacity 1 and apply the algorithm for directed graphs.

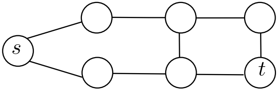
Edge-Disjoint Paths in Undirected Graphs

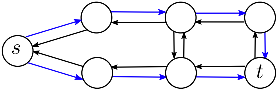




- Can extend the theorem to *undirected* graphs.
- Replace each edge with two directed edges of capacity 1 and apply the algorithm for directed graphs.
- Problem: Both counterparts of an undirected edge (u, v) may be used by different edge-disjoint paths in the directed graph.

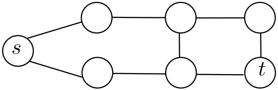
Edge-Disjoint Paths in Undirected Graphs

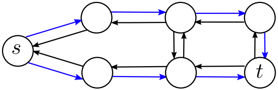




- Can extend the theorem to *undirected* graphs.
- Replace each edge with two directed edges of capacity 1 and apply the algorithm for directed graphs.
- Problem: Both counterparts of an undirected edge (u, v) may be used by different edge-disjoint paths in the directed graph.
- Can obtain an integral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

Edge-Disjoint Paths in Undirected Graphs



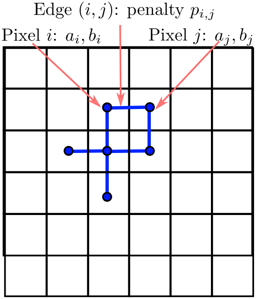


- Can extend the theorem to *undirected* graphs.
- Replace each edge with two directed edges of capacity 1 and apply the algorithm for directed graphs.
- Problem: Both counterparts of an undirected edge (u, v) may be used by different edge-disjoint paths in the directed graph.
- Can obtain an integral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

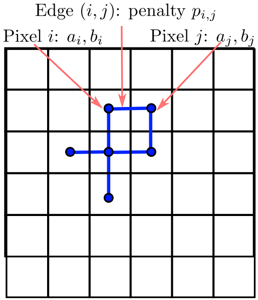
- A fundamental problem in computer vision is that of segmenting an image into coherent regions.
- A basic segmentation problem is that of partitioning an image into a foreground and a background: label each pixel in the image as belonging to the foreground or the background.
  - Note that the image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

Formulating the Image Segmentation Problem



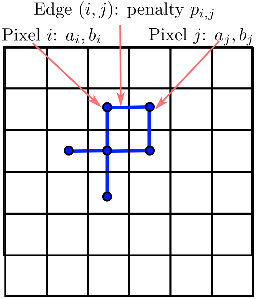
- Let V be the set of pixels in an image.
- Let E be the set of pairs of neighbouring pixels.
- V and E yield an undirected graph $G(V, E)$.

Formulating the Image Segmentation Problem



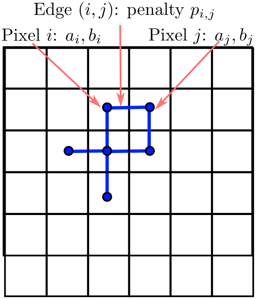
- Let V be the set of pixels in an image.
- Let E be the set of pairs of neighbouring pixels.
- V and E yield an undirected graph $G(V, E)$.
- Each pixel i has a likelihood $a_i > 0$ that it belongs to the foreground and a likelihood $b_i > 0$ that it belongs to the background.
- These likelihoods are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

Formulating the Image Segmentation Problem



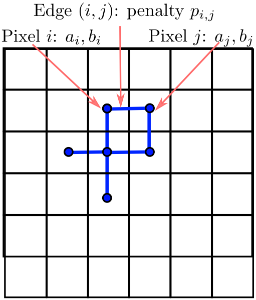
- Let V be the set of pixels in an image.
- Let E be the set of pairs of neighbouring pixels.
- V and E yield an undirected graph $G(V, E)$.
- Each pixel i has a likelihood $a_i > 0$ that it belongs to the foreground and a likelihood $b_i > 0$ that it belongs to the background.
- These likelihoods are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



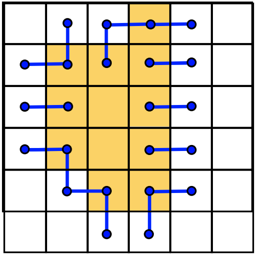


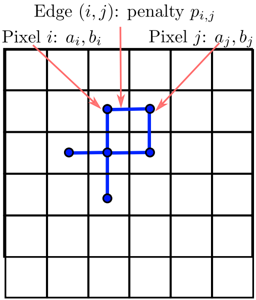
IMAGE SEGMENTATION

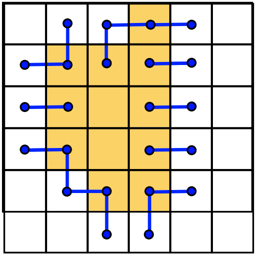
INSTANCE: Pixel graphs $G(V, E)$, likelihood functions $a, b : V \rightarrow \mathbb{R}^+$, penalty function $p : E \rightarrow \mathbb{R}^+$

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

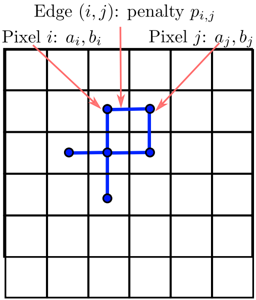


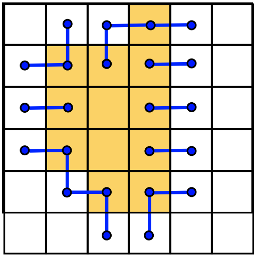


$$q(A, B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}|=1}} p_{ij}$$

- There is a similarity between labellings and ▸ Poll
- But there are differences:

Developing an Algorithm for Image Segmentation

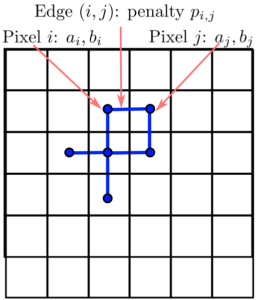


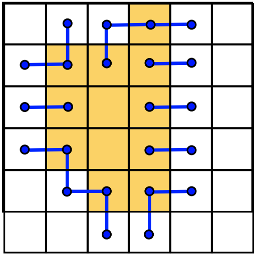


$$q(A, B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

- There is a similarity between labellings and cuts.
- But there are differences: Poll

Developing an Algorithm for Image Segmentation





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

- There is a similarity between labellings and cuts.
- 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

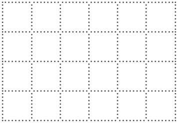
- Solve the other issues like we did earlier.

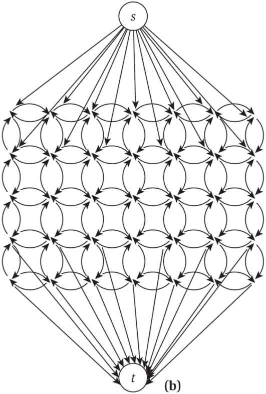
Solving the Other Issues

- Solve the other 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 other 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: Poll

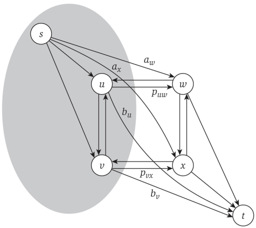


Figure 7.19 An s-t cut on a graph constructed from four pixels. Note how the three types of terms in the expression for $q'(A, B)$ are captured by the cut.

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}.

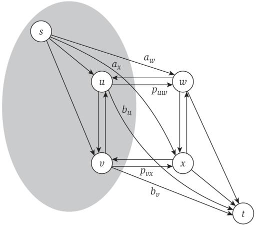


Figure 7.19 An s-t cut on a graph constructed from four pixels. Note how the three types of terms in the expression for $q'(A, B)$ are captured by the cut.

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}.

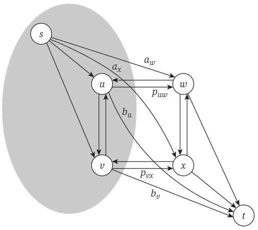


Figure 7.19 An s-t cut on a graph constructed from four pixels. Note how the three types of terms in the expression for $q'(A, B)$ are captured by the cut.

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

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.