Applications of Network Flow

T. M. Murali

March 31, April 2, 2008

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.

Matching in Bipartite Graphs

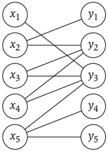


Figure 7.1 A bipartite graph.

- *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

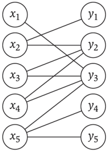


Figure 7.1 A bipartite graph.

- *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.

Bipartite Graph Matching Problem

BIPARTITE MATCHING

INSTANCE: A Bipartite graph G.

SOLUTION: The matching of largest size in G.

Algorithm for Bipartite Graph Matching

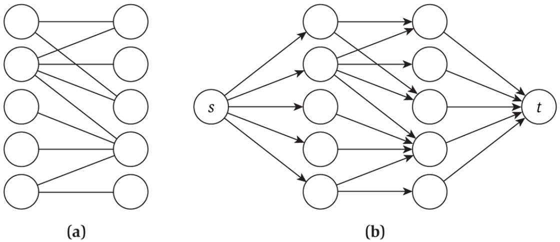


Figure 7.9 (a) A bipartite graph. (b) The corresponding flow network, with all capacities equal to 1.

- 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.
- Compute the maximum flow in G'.
- Claim: the value of the maximum flow is the size of the maximum matching.

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 an integer-valued 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 an integer-valued 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

- 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 an integer-valued 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

- 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 an integer-valued 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

- 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 an integer-valued 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.
  - 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'.

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? $O(m^2 \log n)$ 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 the maximum capacity of a cut in G' is less than n. Therefore, the cut is a certificate.

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 the maximum capacity of a cut in G' is less than n. Therefore, the cut is a certificate.
- But we would like the certificate in terms of G.

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 the maximum capacity of a cut in G' is less than n. Therefore, the cut is a certificate.
- But we would like the certificate in terms of G.
  - For example, two nodes in X with one incident edge each with the same neighbour in Y.

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 the maximum capacity of a cut in G' is less than n. Therefore, the cut is a certificate.
- But we would like the certificate in terms of G.
    - For example, two nodes in X with one incident edge each with the same neighbour in Y.
    - Generally, a subset $A \subseteq X$ with neighbours $\Gamma(A) \subseteq Y$, such that $|A| > |\Gamma(A)|$.

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 the maximum capacity of a cut in G' is less than n. Therefore, the cut is a certificate.
- But we would like the certificate in terms of G.
  - For example, two nodes in X with one incident edge each with the same neighbour in Y.
  - 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 $A \subseteq X$ such that $|A| > |\Gamma(A)|$. A perfect matching or such a subset can be computed in $O(mn)$ time.

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.

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.
- 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 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.
- 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 k. Are there k edge-disjoint paths? If so, what are they?
- Construct k edge-disjoint paths from a flow of value $\geq k$.
  - 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.
- 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 k. Are there k edge-disjoint paths? If so, what are they?
- Construct k edge-disjoint paths from a flow of value $\geq k$.
  - 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 ν, then the set of edges with flow $f(e) = 1$ contains a set of ν edge-disjoint paths.

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.
- 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 k. Are there k edge-disjoint paths? If so, what are they?
- Construct k edge-disjoint paths from a flow of value $\geq k$.
  - 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 ν, 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.

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.
- 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 k. Are there k edge-disjoint paths? If so, what are they?
- Construct k edge-disjoint paths from a flow of value $\geq k$.
  - 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 ν, 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.
- We just proved: there are k edge-disjoint paths from s to t in a directed graph G iff the maximum value of an s-t flow in G is $\geq k$.

Running Time of the Edge-Disjoint Paths Algorithm

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

Running Time of the Edge-Disjoint Paths Algorithm

- Given a flow of value $\geq k$, 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

Running Time of the Edge-Disjoint Paths Algorithm

- Given a flow of value $\geq k$, 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 two counterparts of an undirected edge (u, v) may be used by the 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 two counterparts of an undirected edge (u, v) may be used by the 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 two counterparts of an undirected edge (u, v) may be used by the 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.

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 $c : E \rightarrow \mathbb{Z}^+$ 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.

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 $c : E \rightarrow \mathbb{Z}^+$ 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 internal 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 internal 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 there exist a circulation that is *feasible*, 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

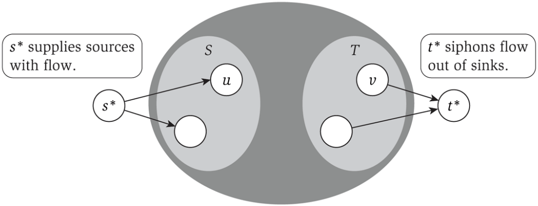
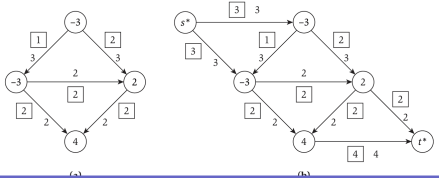


Figure 7.14 Reducing the Circulation Problem to the Maximum-Flow Problem.

- Create a new graph $G' = G$ and
  1. create two new nodes in G': a source s^* and a sink t^*;
  2. connect s^* to each node in S using an edge with capacity $-d_v$;
  3. connect each node in T to t^* using an edge with capacity d_v.



Computing a Feasible Circulation

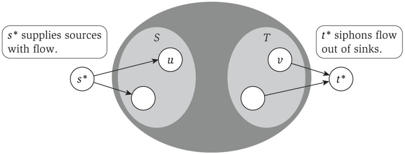


Figure 7.14 Reducing the Circulation Problem to the Maximum-Flow Problem.

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

Computing a Feasible Circulation

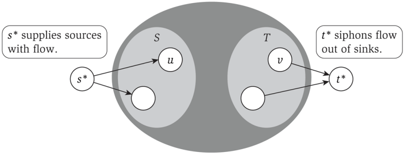


Figure 7.14 Reducing the Circulation Problem to the Maximum-Flow Problem.

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

Computing a Feasible Circulation

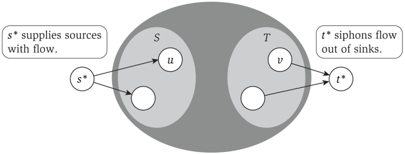


Figure 7.14 Reducing the Circulation Problem to the Maximum-Flow Problem.

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

Computing a Feasible Circulation

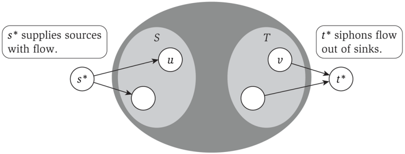


Figure 7.14 Reducing the Circulation Problem to the Maximum-Flow Problem.

- 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.

Computing a Feasible Circulation

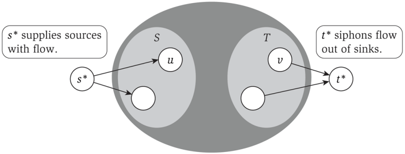


Figure 7.14 Reducing the Circulation Problem to the Maximum-Flow Problem.

- 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.
- Flow $\rightarrow$ circulation. If there is an s-t flow of value D in G',

Computing a Feasible Circulation

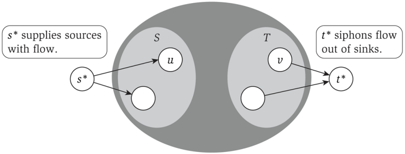


Figure 7.14 Reducing the Circulation Problem to the Maximum-Flow Problem.

- 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.
- 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.

Computing a Feasible Circulation

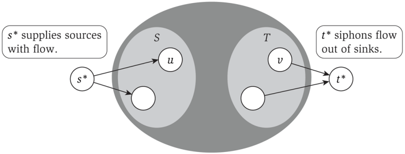


Figure 7.14 Reducing the Circulation Problem to the Maximum-Flow Problem.

- 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.
- 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.
- We have just 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 is a function $f : E \rightarrow \mathbb{R}^+$ 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 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 internal 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 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 internal 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 must 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 must 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 must 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 must 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: lower bound on each edge is 0, capacity of edge e is $c(e) - l(e)$, and demand of node v is $d_v - L_v$.
- Claim: there is a feasible circulation in G iff there is a feasible circulation in G'.

Data Mining

- Algorithmic study of unexpected patterns in large quantities of data.
- Study customer preferences is an important topic.
  - Customers who buy diapers also buy beer:
    - http://www.dssresources.com/newsletters/66.php
    - http://www.forbes.com/forbes/1998/0406/6107128s1.html
  - People who bought “Harry Potter and the Deathly Hallows” also bought “Making Money (Discworld)”.
- Store cards allow companies to keep track of your history of shopping.

Survey Design

- Company sells k products.
- Company has a database of purchase histories of many customers.
- Company wants to send a customised survey to each of its n customers to further understand their preferences.

Survey Design

- Company sells k products.
- Company has a database of purchase histories of many customers.
- Company wants to send a customised survey to each of its n customers to further understand their preferences.
- Survey must satisfy certain constraints:
  1. Each customer receives questions about a subset of products.
  2. A customer receives questions only about products he/she has bought.
  3. The questionnaire must be informative but not too long: each customer i should be asked about a number of products between c_i and c'_i.
  4. Each product must have enough data collected: between p_j and p'_j customers should be asked about product j.

Survey Design

- Company sells k products.
- Company has a database of purchase histories of many customers.
- Company wants to send a customised survey to each of its n customers to further understand their preferences.
- Survey must satisfy certain constraints:
  1. Each customer receives questions about a subset of products.
  2. A customer receives questions only about products he/she has bought.
  3. The questionnaire must be informative but not too long: each customer i should be asked about a number of products between c_i and c'_i.
  4. Each product must have enough data collected: between p_j and p'_j customers should be asked about product j.
- Is it possible to design a survey that satisfies this constraints?

Formalising the Survey Design Problem

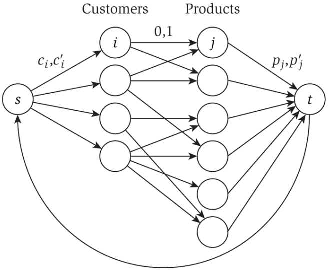
- Input is a bipartite graph G:
  - Nodes are n customers and k products.
  - There is an edge between customer i and product j iff the customer has ever purchased the product.
  - For each customer $1 \leq i \leq n$, limits $c_i \leq c'_i$ on the number of products he or she can be asked about.
  - For each product $1 \leq j \leq k$, limits $p_j \leq p'_j$ on the number of distinct customers asked about the product.

Solving the Survey Design Problem

- Reduce the problem to a circulation problem on a flow network G' with demands and lower bounds (lbs).

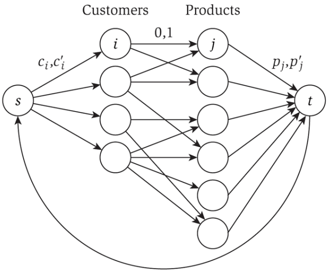
Solving the Survey Design Problem

- Reduce the problem to a circulation problem on a flow network G' with demands and lower bounds (lbs).
- Orient edges in G from customers to products: capacity 1, lb 0.
- Add node s, edges (s, i) to each customer: capacity c'_i, lb c_i.
- Add node t, edges (j, t) from each product: capacity p'_i, lb p_i.
- Set node demands to .



Solving the Survey Design Problem

- Reduce the problem to a circulation problem on a flow network G' with demands and lower bounds (lbs).
- Orient edges in G from customers to products: capacity 1, lb 0.
- Add node s, edges (s, i) to each customer: capacity c'_i, lb c_i.
- Add node t, edges (j, t) from each product: capacity p'_i, lb p_i.
- Set node demands to 0.
- Add edge from t to s: capacity $\sum_i c'_i$, lb $\sum_i c_i$.



Solving the Survey Design Problem

- Reduce the problem to a circulation problem on a flow network G' with demands and lower bounds (lbs).
- Orient edges in G from customers to products: capacity 1, lb 0.
- Add node s, edges (s, i) to each customer: capacity c'_i, lb c_i.
- Add node t, edges (j, t) from each product: capacity p'_i, lb p_i.
- Set node demands to 0.
- Add edge from t to s: capacity $\sum_i c'_i$, lb $\sum_i c_i$.
- Claim: G' has a feasible circulation iff there is a feasible survey.

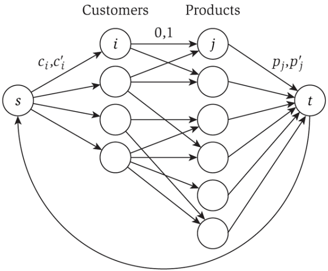


Image Segmentation

- 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.

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, assign 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

- 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 in E with two directed edges of capacity 1.

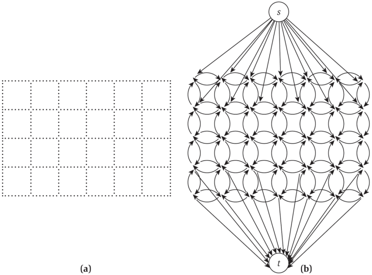


Figure 7.18 (a) A pixel graph. (b) A sketch of the corresponding flow graph. Not all edges from the source or to the sink are drawn.

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

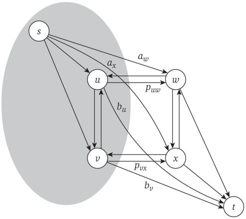


Figure 7.19 An *s*-*t* cut on a graph constructed from four pixels. Note how the three types of terms in the expression for $q'(A, B)$ are captured by the cut.

- 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:

Cuts in the Flow Network

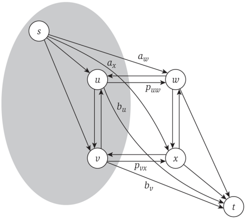


Figure 7.19 An s-t cut on a graph constructed from four pixels. Note how the three types of terms in the expression for $q'(A, B)$ are captured by the cut.

- 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:
  - $(s, j), j \in B$ contributes a_j.
  - $(i, t), i \in A$ contributes b_i.
  - $(i, j), i \in A, j \in B$ contributes p_{ij}.

Cuts in the Flow Network

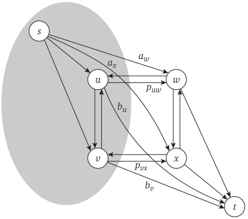


Figure 7.19 An s-t cut on a graph constructed from four pixels. Note how the three types of terms in the expression for $q'(A, B)$ are captured by the cut.

- 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:
  - $(s, j), j \in B$ contributes a_j.
  - $(i, t), i \in A$ contributes b_i.
  - $(i, j), i \in A, j \in B$ contributes p_{ij}.

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

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.