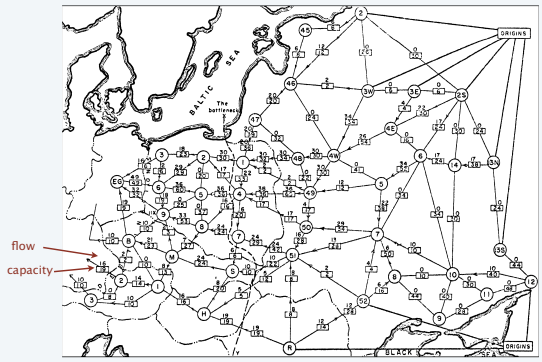
CSCI 355: Algorithm Design and Analysis
9. Network Flow

- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*
- *bipartite matching*
- *disjoint paths*
- *other applications*

Maximum flows (Tolstoĭ, 1930s)

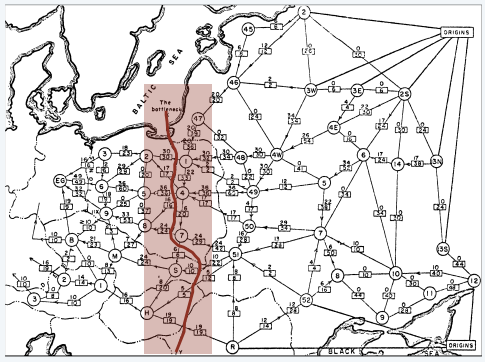
Soviet Union's goal. Maximize flow of supplies to Eastern Europe.



rail network connecting Soviet Union with Eastern European countries
(map declassified by Pentagon in 1999)

Minimum cuts (RAND, 1950s)

United States' goal. Cut supplies (if Cold War turns into real war).

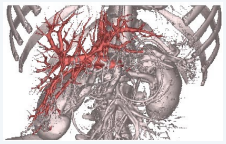


rail network connecting Soviet Union with Eastern European countries
(map declassified by Pentagon in 1999)

Max-flow and min-cut

A widely applicable model.

- Data mining.
- Open-pit mining.
- Bipartite matching.
- Network reliability.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Markov random fields.
- Distributed computing.
- Security of statistical data.
- Egalitarian stable matching.
- Network intrusion detection.
- Multi-camera scene reconstruction.
- Sensor placement for homeland security.
- Many, many, more.



liver and hepatic vascularization segmentation

CSCI 355: Algorithm Design and Analysis
9. Network Flow

- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*
- *bipartite matching*
- *disjoint paths*
- *other applications*

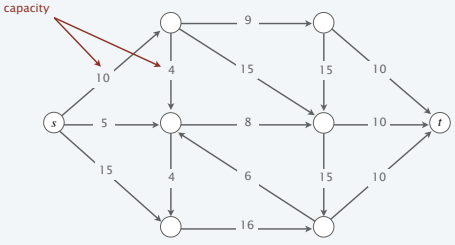
Flow network

A flow network is a tuple $G = (V, E, s, t, c)$.

- Digraph (V, E) with source $s \in V$ and sink $t \in V$.
- Capacity $c(e) \geq 0$ for each $e \in E$.

assume all vertices are reachable from s

Intuition. Material flowing through a transportation network; material originates at the source and is sent to the sink.



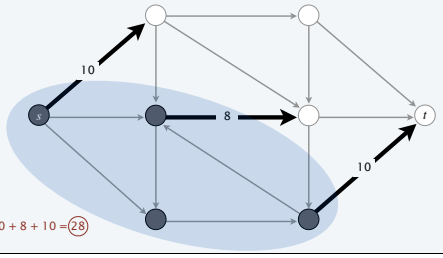
Minimum-cut problem

Def. An st-cut (cut) is a partition (A, B) of the vertices with $s \in A$ and $t \in B$.

Def. The capacity of a cut is the sum of the capacities of edges from A to B.

$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$

Min-cut problem. Find a cut of minimum capacity.



capacity = 10 + 8 + 10 = 28

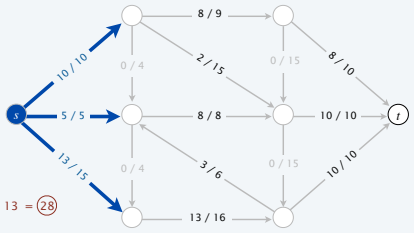
Maximum-flow problem

Def. An st-flow (flow) f is a function that satisfies:

- For each $e \in E$: $0 \le f(e) \le c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

Def. The value of a flow f is: $val(f) = \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ in to } s} f(e)$

Max-flow problem. Find a flow of maximum value.



value = 10 + 5 + 13 = 28

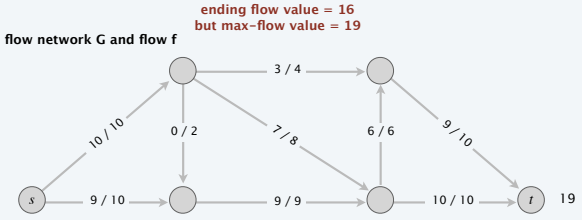
CSCI 355: Algorithm Design and Analysis
9. Network Flow

- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*
- *bipartite matching*
- *disjoint paths*
- *other applications*

Toward a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

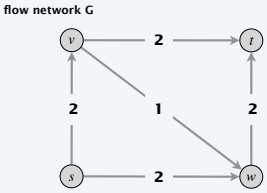


Why the greedy algorithm fails

Q. Why does the greedy algorithm fail?
A. Once the greedy algorithm increases the flow on an edge, it never decreases it.

Ex. Consider the flow network G.

- The unique max flow f^* has $f^*(v, w) = 0$.
- Greedy algorithm could choose $s \rightarrow v \rightarrow w \rightarrow t$ as first path.

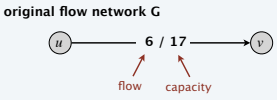


Bottom line. Need some mechanism to "undo" a bad decision.

Residual networks

Original edge. $e = (u, v) \in E$.

- Flow $f(e)$.
- Capacity $c(e)$.

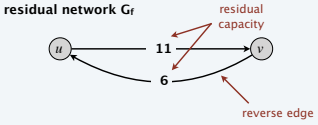


Reverse edge. $e^{\text{reverse}} = (v, u)$.

- "Undo" flow sent.

Residual capacity.

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^{\text{reverse}} \in E \end{cases}$$



Residual network. $G_f = (V, E_f, s, t, c_f)$.

- $E_f = \{e : f(e) < c(e)\} \cup \{e^{\text{reverse}} : f(e) > 0\}$.

edges with positive residual capacity

where the flow on a reverse edge negates the flow on the corresponding forward edge

Key property. f' is a flow in G_f iff $f + f'$ is a flow in G.

Augmenting paths

Def. An augmenting path is a simple $s \rightsquigarrow t$ path in the residual network G_f.

Def. The bottleneck capacity of an augmenting path P is the minimum residual capacity of any edge in P.

Key property. Let f be a flow and let P be an augmenting path in G_f. Then, after calling $f' \leftarrow \text{AUGMENT}(f, c, P)$, the resulting f' is a flow and $val(f') = val(f) + bottleneck(G_f, P)$.

```
AUGMENT(f, c, P)
δ ← bottleneck capacity of augmenting path P.
FOREACH edge e ∈ P :
   IF (e ∈ E) f(e) ← f(e) + δ.
   ELSE       f(e^reverse) ← f(e^reverse) − δ.
RETURN f.
```


Ford–Fulkerson algorithm

see 9A

Ford–Fulkerson augmenting path algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P in the residual network G_f.
- Augment flow along path P.
- Repeat until you get stuck.

Ford

Fulkerson

```
FORD–FULKERSON(G)
FOREACH edge e ∈ E : f(e) ← 0.
G_f ← residual network of G with respect to flow f.
WHILE (there exists an s↝t path P in G_f)
   f ← AUGMENT(f, c, P).        ← augmenting path
   Update G_f.
RETURN f.
```


CSCI 355: Algorithm Design and Analysis
9. Network Flow

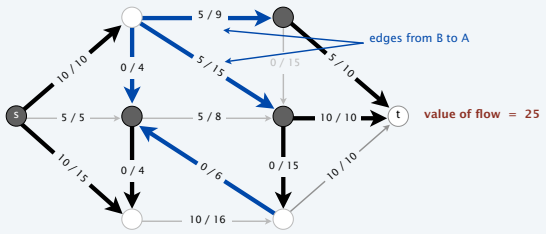
- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*
- *bipartite matching*
- *disjoint paths*
- *other applications*

Relationship between flows and cuts

Flow value lemma. Let f be any flow and let (A, B) be any cut. Then the value of the flow f equals the net flow across the cut (A, B).

$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$

net flow across cut = (10 + 10 + 5 + 10 + 0 + 0) – (5 + 5 + 0 + 0) = 25



Relationship between flows and cuts

Flow value lemma. Let f be any flow and let (A, B) be any cut. Then the value of the flow f equals the net flow across the cut (A, B).

$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$

Pf.

$$val(f) = \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ in to } s} f(e)$$

by flow conservation, all terms except for $v = s$ are 0

$$= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) \right)$$

$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \quad \blacksquare$$

Relationship between flows and cuts

Weak duality. Let f be any flow and (A, B) be any cut. Then $val(f) \le cap(A, B)$.

Pf.

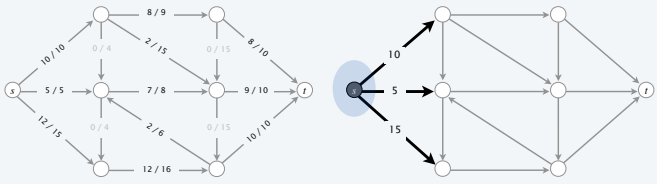
$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$

flow value lemma

$$\le \sum_{e \text{ out of } A} f(e)$$

$$\le \sum_{e \text{ out of } A} c(e)$$

$$= cap(A, B) \quad \blacksquare$$



value of flow = 27 ≤ capacity of cut = 30

Certificate of optimality

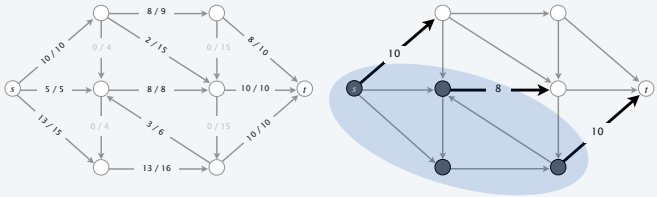
Corollary. Let f be a flow and let (A, B) be any cut.
If $val(f) = cap(A, B)$, then f is a max flow and (A, B) is a min cut.

Pf.

weak duality

- For any flow f': $val(f') \le cap(A, B) = val(f)$.
- For any cut (A', B'): $cap(A', B') \ge val(f) = cap(A, B)$. ▪

weak duality

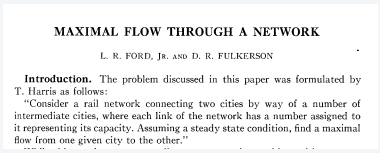


value of flow = 28 = capacity of cut = 28

Max-flow min-cut theorem

Max-flow min-cut theorem. Value of a max flow = capacity of a min cut.

strong duality

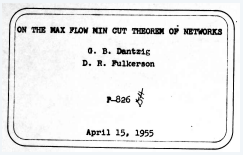


MAXIMAL FLOW THROUGH A NETWORK

L. R. FORD, JR. AND D. R. FULKERSON

Introduction. The problem discussed in this paper was formulated by T. Harris as follows:

"Consider a rail network connecting two cities by way of a number of intermediate cities, where each link of the network has a number assigned to it representing its capacity. Assuming a steady state condition, find a maximal flow from one given city to the other."

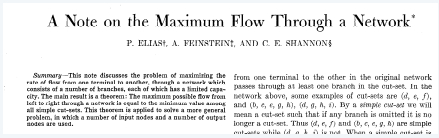


ON THE MAX FLOW MIN CUT THEOREM OF NETWORKS

G. B. Dantzig
D. R. Fulkerson

P-826

April 15, 1955



A Note on the Maximum Flow Through a Network*

P. ELIAS†, A. FEINSTEIN‡, AND C. E. SHANNON§

Max-flow min-cut theorem

Max-flow min-cut theorem. Value of a max flow = capacity of a min cut.

Augmenting path theorem. A flow f is a max flow iff there are no augmenting paths.

Pf. The following three conditions are equivalent for any flow f:

i. There exists a cut (A, B) such that $cap(A, B) = val(f)$.
ii. f is a max flow.
iii. There is no augmenting path with respect to f.

← if Ford–Fulkerson terminates, then f is a max flow

[i ⇒ ii]

- This is the weak duality corollary. ▪

Max-flow min-cut theorem

Max-flow min-cut theorem. Value of a max flow = capacity of a min cut.

Augmenting path theorem. A flow f is a max flow iff there are no augmenting paths.

Pf. The following three conditions are equivalent for any flow f:

i. There exists a cut (A, B) such that $cap(A, B) = val(f)$.

ii. f is a max flow.

iii. There is no augmenting path with respect to f.

[ii $\Rightarrow$ iii] We prove the contrapositive: $\neg$ iii $\Rightarrow \neg$ ii.

- Suppose that there is an augmenting path with respect to f.
- We can improve the flow f by sending the flow along this path.
- Thus, f is not a max flow. ▪

Max-flow min-cut theorem

Pf.

[iii $\Rightarrow$ i]

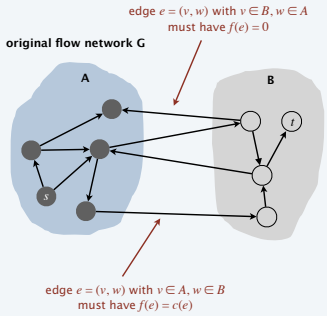
- Let f be a flow with no augmenting paths.
- Let A = set of vertices reachable from s in the residual network G_f.
- By the definition of A: $s \in A$.
- By the definition of flow f: $t \notin A$.

$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$

flow value lemma

$$= \sum_{e \text{ out of } A} c(e) - 0$$

$$= cap(A, B) \quad \blacksquare$$

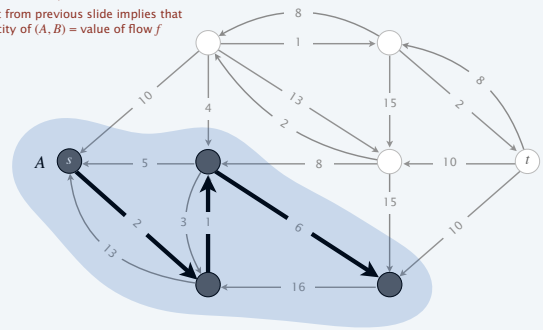


Computing a minimum cut from a maximum flow

Theorem. Given any max flow f, we can compute a min cut (A, B) in $O(m)$ time.

Pf. Let A = set of vertices reachable from s in the residual network G_f. ▪

argument from previous slide implies that capacity of (A, B) = value of flow f



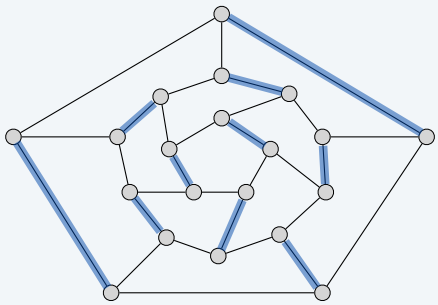
CSCI 355: Algorithm Design and Analysis
9. Network Flow

- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*
- *bipartite matching*
- *disjoint paths*
- *other applications*

Matchings

Def. Given an undirected graph $G = (V, E)$, a subset of edges $M \subseteq E$ is a matching if each vertex appears in at most one edge in M.

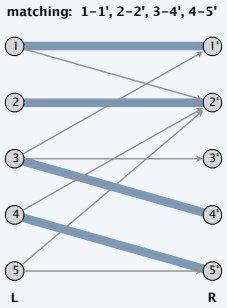
Max matching. Given a graph G, find a maximum cardinality matching.



Bipartite matching

Def. A graph G is bipartite if the vertices can be partitioned into two subsets L and R such that every edge connects a vertex in L with a vertex in R.

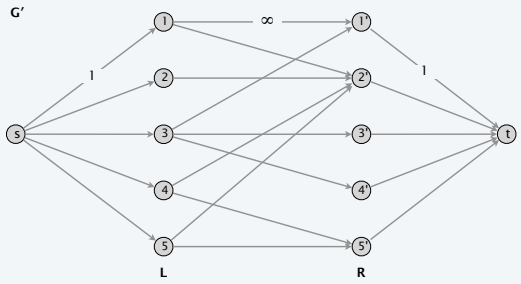
Bipartite matching. Given a bipartite graph $G = (L \cup R, E)$, find a maximum cardinality matching.



Bipartite matching: max-flow formulation

Formulation.

- Create a digraph $G' = (L \cup R \cup \{s, t\},\ E')$.
- Direct all edges from L to R, and assign infinite (or unit) capacity.
- Add unit-capacity edges from s to each vertex in L.
- Add unit-capacity edges from each vertex in R to t.



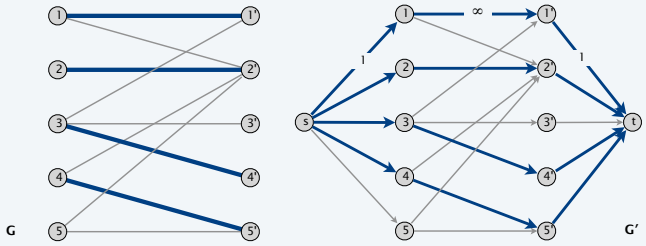
Max-flow formulation: proof of correctness

Theorem. There is a 1–1 correspondence between matchings of cardinality k in G and integral flows of value k in G'.

for each edge e: $f(e) \in \{0, 1\}$

Pf. [$\Rightarrow$]

- Let M be a matching in G of cardinality k.
- Consider the flow f that sends 1 unit on each of the k corresponding paths.
- f is a flow of value k. ▪



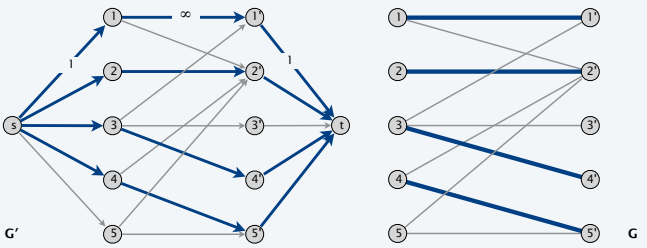
Max-flow formulation: proof of correctness

Theorem. There is a 1–1 correspondence between matchings of cardinality k in G and integral flows of value k in G'.

for each edge e: $f(e) \in \{0, 1\}$

Pf. [$\Leftarrow$]

- Let f be an integral flow in G' of value k.
- Consider M = set of edges from L to R with $f(e) = 1$.
  - Each vertex in L and R participates in at most one edge in M
  - $|M| = k$: apply the flow-value lemma to cut $(L \cup \{s\}, R \cup \{t\})$ ▪

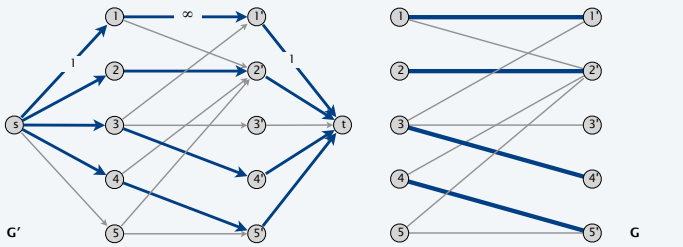


Max-flow formulation: proof of correctness

Corollary. We can solve the bipartite matching problem via max-flow formulation.

Pf.

- Integrality theorem $\Rightarrow$ there exists a max flow f^* in G' that is integral.
- 1–1 correspondence $\Rightarrow$ f^* corresponds to a max-cardinality matching. ▪



Perfect matchings in bipartite graphs

Def. Given a graph $G = (V, E)$, a subset of edges $M \subseteq E$ is a perfect matching if each vertex appears in exactly one edge in M.

Q. When does a bipartite graph have a perfect matching?

Structure of bipartite graphs with perfect matchings.

- Clearly, we must have $|L| = |R|$.
- What other conditions are necessary?
- What other conditions are sufficient?

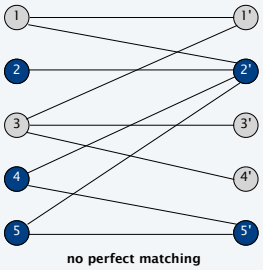
Perfect matchings in bipartite graphs

Notation. Let S be a subset of vertices, and let $N(S)$ be the set of vertices adjacent to vertices in S.

Observation. If a bipartite graph $G = (L \cup R, E)$ has a perfect matching, then $|N(S)| \geq |S|$ for all subsets $S \subseteq L$.

Pf. Each vertex in S has to be matched to a different vertex in $N(S)$. ▪

S = { 2, 4, 5 }
N(S) = { 2', 5' }



no perfect matching

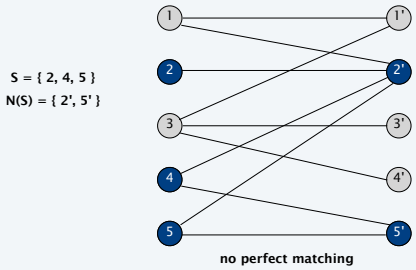
Hall's marriage theorem

Theorem. [Frobenius 1917, Hall 1935] Let $G = (L \cup R, E)$ be a bipartite graph with $|L| = |R|$. Then G has a perfect matching iff $|N(S)| \geq |S|$ for all subsets $S \subseteq L$.

Pf. [$\Rightarrow$] This was the previous observation.


Frobenius


Hall

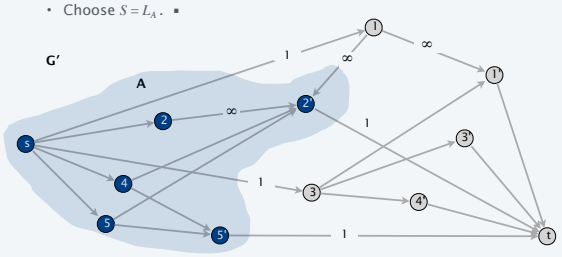


no perfect matching

Hall's marriage theorem

Pf. [$\Leftarrow$] Suppose G does not have a perfect matching.

- Formulate as a max-flow problem and let (A, B) be a min cut in G'.
- By the max-flow min-cut theorem, $cap(A, B) < |L|$.
- Define $L_A = L \cap A$, $L_B = L \cap B$, $R_A = R \cap A$.
- $cap(A, B) = |L_B| + |R_A| \Rightarrow |R_A| < |L_A|$.
- Min cut can't use ∞ edges $\Rightarrow N(L_A) \subseteq R_A$.
- $|N(L_A)| \leq |R_A| < |L_A|$.
- Choose $S = L_A$. ▪



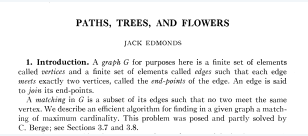
L_A = {2, 4, 5}
L_B = {1, 3}
R_A = {2', 5'}
$N(L_A)$ = {2', 5'}

Non-bipartite matching

Problem. Given an undirected graph, find a max-cardinality matching.

- The structure of non-bipartite graphs is more complicated...
- ...but well understood. [Tutte–Berge formula, Edmonds–Gallai]

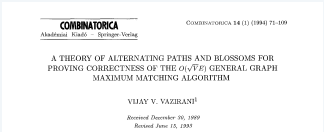
- Blossom algorithm: $O(n^4)$. [Edmonds 1965]
- Best known: $O(m\, n^{1/2})$. [Micali–Vazirani 1980, Vazirani 1994]


PATHS, TREES, AND FLOWERS

JACK EDMONDS

1. Introduction. A *graph* G for purposes here is a finite set of elements called *vertices* and a finite set of elements called *edges* such that each edge *meets* exactly two vertices, called the *end-points* of the edge. An edge is said to *join* its end-points.

A *matching* in G is a subset of its edges such that no two meet the same vertex. We describe an efficient algorithm for finding in a given graph a matching of maximum cardinality. This problem was posed and partly solved by C. Berge; see Sections 3.7 and 3.8.


COMBINATORICA

Combinatorica 14 (1) (1994) 71–109

A THEORY OF ALTERNATING PATHS AND BLOSSOMS FOR PROVING CORRECTNESS OF THE $O(\sqrt{V}E)$ GENERAL GRAPH MAXIMUM MATCHING ALGORITHM

VIJAY V. VAZIRANI[1]

Received December 30, 1989
Revised June 15, 1993

Historical significance (Edmonds, 1965)

2. Digression. An explanation is due on the use of the words "efficient algorithm." First, what I present is a conceptual description of an algorithm and not a particular formalized algorithm or "code."

For practical purposes computational details are vital. However, my purpose is only to show as attractively as I can that there is an efficient algorithm. According to the dictionary, "efficient" means "adequate in operation or performance." This is roughly the meaning I want—in the sense that it is conceivable for maximum matching to have no efficient algorithm. Perhaps a better word is "good."

I am claiming, as a mathematical result, the existence of a *good* algorithm for finding a maximum cardinality matching in a graph.

There is an obvious finite algorithm, but that algorithm increases in difficulty exponentially with the size of the graph. It is by no means obvious whether *or not* there exists an algorithm whose difficulty increases only algebraically with the size of the graph.

Edmonds

CSCI 355: Algorithm Design and Analysis
9. Network Flow

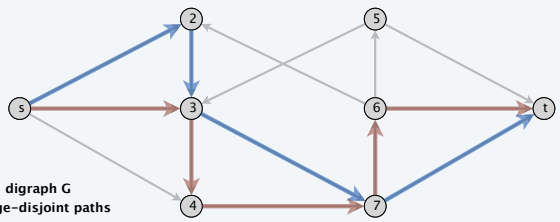
- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*
- *bipartite matching*
- *disjoint paths*
- *other applications*

Edge-disjoint paths

Def. Two paths are edge-disjoint if they have no edge in common.

Edge-disjoint paths problem. Given a digraph $G = (V, E)$ and two vertices s and t, find the max number of edge-disjoint $s \rightsquigarrow t$ paths.

Ex. Communication networks.



digraph G
2 edge-disjoint paths

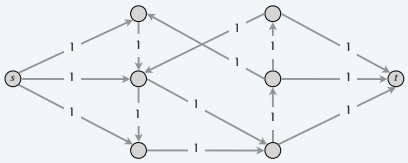
Edge-disjoint paths

Max-flow formulation. Assign unit capacity to every edge.

Theorem. There is a 1–1 correspondence between k edge-disjoint $s \rightsquigarrow t$ paths in G and integral flows of value k in G'.

Pf. [$\Rightarrow$]

- Let $P_1, \ldots, P_k$ be k edge-disjoint $s \rightsquigarrow t$ paths in G.
- Set $f(e) = \begin{cases} 1 & \text{edge } e \text{ participates in some path } P_j \\ 0 & \text{otherwise} \end{cases}$
- Since paths are edge-disjoint, f is a flow of value k. ▪



Edge-disjoint paths

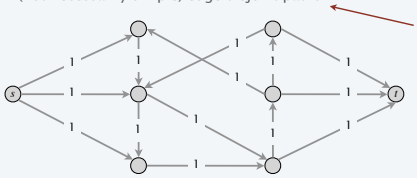
Max-flow formulation. Assign unit capacity to every edge.

Theorem. There is a 1–1 correspondence between k edge-disjoint $s \rightsquigarrow t$ paths in G and integral flows of value k in G'.

Pf. [$\Leftarrow$]

- Let f be an integral flow in G' of value k.
- Consider an edge (s, u) with $f(s, u) = 1$.
  - By flow conservation, there exists an edge (u, v) with $f(u, v) = 1$
  - Continue until we reach t, always choosing a new edge
- Produces k (not necessarily simple) edge-disjoint paths. ▪

we can eliminate cycles to get simple paths in $O(mn)$ time if desired (flow decomposition)



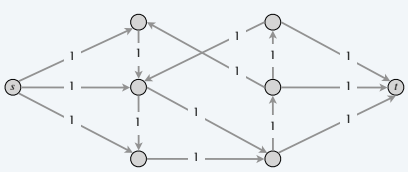
Edge-disjoint paths

Max-flow formulation. Assign unit capacity to every edge.

Corollary. We can solve the edge-disjoint paths problem via max-flow formulation.

Pf.

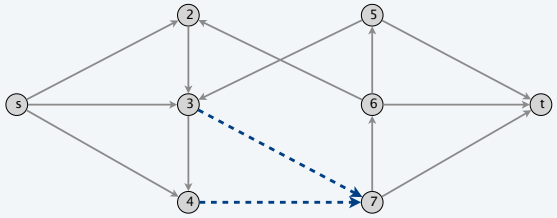
- Integrality theorem $\Rightarrow$ there exists a max flow f^* in G' that is integral.
- 1–1 correspondence $\Rightarrow$ f^* corresponds to max number of edge-disjoint $s \rightsquigarrow t$ paths in G. ▪



Network connectivity

Def. A set of edges $F \subseteq E$ disconnects t from s if every $s \rightsquigarrow t$ path uses at least one edge in F.

Network connectivity. Given a digraph $G = (V, E)$ and two vertices s and t, find the min number of edges whose removal disconnects t from s.

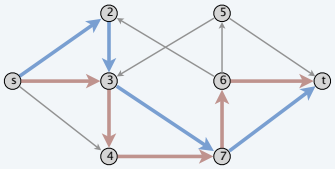


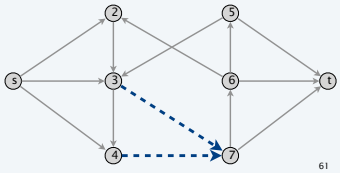
Menger's theorem

Theorem. [Menger 1927] The max number of edge-disjoint $s \rightsquigarrow t$ paths equals the min number of edges whose removal disconnects t from s.

Pf. [$\leq$]

- Suppose the removal of $F \subseteq E$ disconnects t from s, and $|F| = k$.
- Every $s \rightsquigarrow t$ path uses at least one edge in F.
- Hence, the number of edge-disjoint paths is $\leq k$. ▪



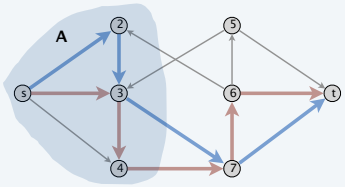


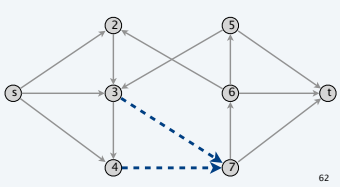
Menger's theorem

Theorem. [Menger 1927] The max number of edge-disjoint $s \rightsquigarrow t$ paths equals the min number of edges whose removal disconnects t from s.

Pf. [$\geq$]

- Suppose the max number of edge-disjoint $s \rightsquigarrow t$ paths is k.
- Then the value of the max flow = k.
- Max-flow min-cut theorem $\Rightarrow$ there exists a cut (A, B) of capacity k.
- Let F be the set of edges going from A to B.
- $|F| = k$ and disconnects t from s. ▪





CSCI 355: Algorithm Design and Analysis
9. Network Flow

- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*
- *bipartite matching*
- *disjoint paths*
- *other applications*

Survey design

- Design a survey asking n_1 consumers about n_2 products. ← one survey question per product
- We can survey a consumer i about a product j only if they own it.
- We ask a consumer i between c_i and c_i' questions.
- We ask between p_j and p_j' consumers about a product j.

Goal. Design a survey that meets these specs, if possible.

Bipartite perfect matching. Special case when $c_i = c_i' = p_j = p_j' = 1$.

Survey design

Max-flow formulation. Model as a circulation problem with lower bounds.

- Add edge (i, j) if consumer j owns product i.
- Add edge from s to consumer j.
- Add edge from product i to t.
- Add edge from t to s.
- All demands = 0.
- Integer circulation $\iff$ feasible survey design.

↖ all supplies and demands are 0

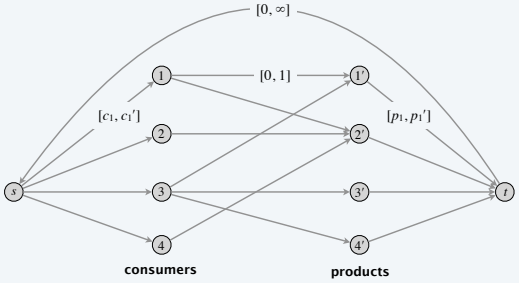
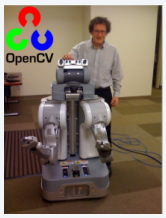


Image segmentation

Image segmentation.

- Need to divide an image into coherent regions.
- A central problem in image processing.

Ex. Separate the human and the robot from the background scene.



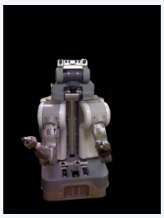
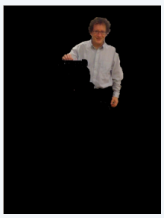
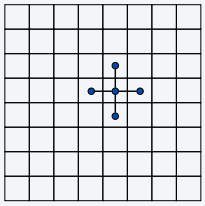



Image segmentation

Foreground / background segmentation.

- Label each pixel in the picture as belonging to foreground or background.
- V = set of pixels, E = pairs of neighbouring pixels.
- $a_i \geq 0$ is likelihood pixel i in foreground.
- $b_i \geq 0$ is likelihood pixel i in background.
- $p_{ij} \geq 0$ is separation penalty for labeling one of i and j as foreground, and the other as background.



Goals.

- Accuracy: if $a_i > b_i$ in isolation, we should prefer to label i in foreground.
- Smoothness: if many neighbours of i are labeled foreground, we should be inclined to label i as foreground.
- Find partition (A, B) that maximizes: $\sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$

  (A: foreground, B: background)

Image segmentation

Formulate as min cut problem.

- Maximization.
- No source or sink.
- Undirected graph.

Turn into minimization problem.

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

  is equivalent to minimizing

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

  (a constant)

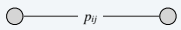
  or, alternatively, $\sum_{j \in B} a_j + \sum_{i \in A} b_i + \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$

Image segmentation

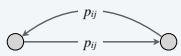
Formulate as min cut problem $G' = (V', E')$.

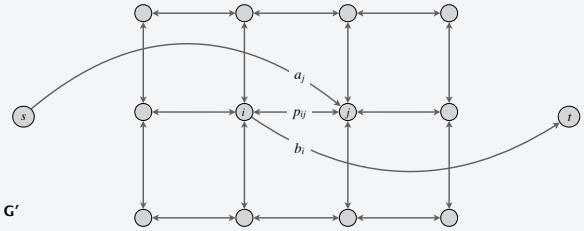
- Include vertex for each pixel.
- Use two antiparallel edges instead of one undirected edge.
- Add source s to correspond to foreground.
- Add sink t to correspond to background.

edge in G



two antiparallel edges in G′





G′

Image segmentation

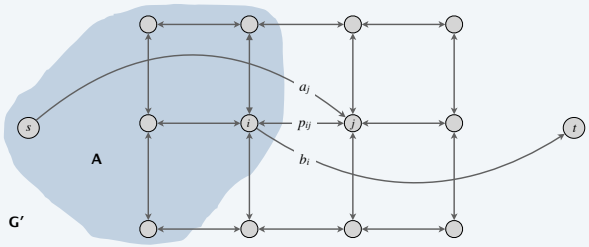
Consider a min cut (A, B) in G'.

- A = foreground.

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

← if i and j are on different sides, then p_{ij} is counted exactly once

- Precisely the quantity we want to minimize.



G′

Airline scheduling

Airline scheduling.

- A complex problem faced by airline carriers.
- Must produce schedules that are efficient in terms of equipment usage, crew allocation, and customer satisfaction. ← even in presence of unpredictable events, such as weather and breakdowns
- Airlines are one of the largest consumers of high-powered algorithmic techniques.

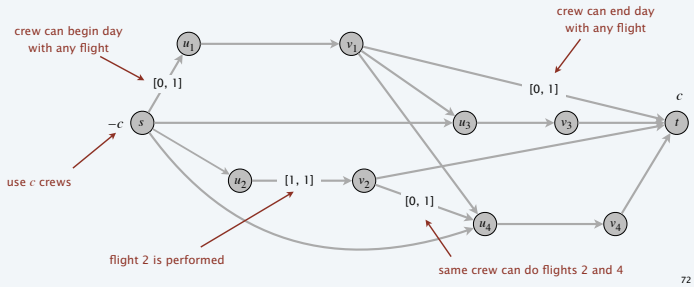
"Toy problem."

- Manage flight crews by reusing them over multiple flights.
- Input: set of k flights for a given day.
- Flight i leaves origin o_i at time s_i and arrives at destination d_i at time f_i.
- Minimize number of flight crews.

Airline scheduling

Circulation formulation. [to see if c crews suffice]

- For each flight i, include two vertices u_i and v_i. ← u_i = start of flight i, v_i = end of flight i
- Add source s with demand $-c$, and edges (s, u_i) with capacity 1.
- Add sink t with demand c, and edges (v_i, t) with capacity 1.
- For each i, add edge (u_i, v_i) with lower bound and capacity 1.
- if flight j is reachable from i, add edge (v_i, u_j) with capacity 1.



Airline scheduling

Remark. We solved a toy version of a real problem.

Real-world problem models countless other factors:

- Union regulations: e.g., flight crews can fly only a certain number of hours in a given time window.
- Need optimal schedule over planning horizon, not just one day.
- Deadheading has a cost.
- Flights don't always leave or arrive on schedule.
- Simultaneously optimize both flight schedule and fare structure.

Message.

- Our solution is a generally useful technique for efficient reuse of limited resources but trivializes real airline scheduling problem.
- Flow techniques useful for solving airline scheduling problems (and are widely used in practice).
- Running an airline efficiently is a very difficult problem.