Applications of Network Flows

Lecture 18

March 31, 2011

Network Flow: Facts to Remember

Flow network: directed graph $\mathbf{G}$, capacities $\mathbf{c}$, source $\mathbf{s}$, sink $\mathbf{t}$

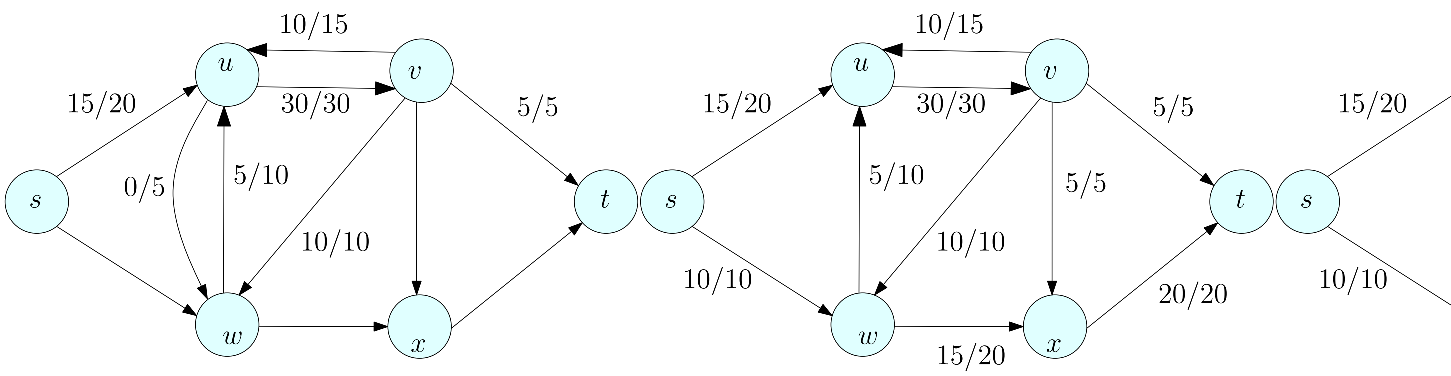
- maximum $\mathbf{s}$-$\mathbf{t}$ flow can be computed
  - using Ford-Fulkerson algorithm in $\mathbf{O(mC)}$ time when capacities are integral and $\mathbf{C}$ is an upper bound on the flow
  - using variant of algorithm in homework in $\mathbf{O(m^2 \log C)}$ time when capacities are integral
  - using Edmonds-Karp algorithm in $\mathbf{O(m^2 n)}$ time when capacities are rational (strongly polynomial time algorithm)
- if capacities are integral then there is a maximum flow that is integral and above algorithms give an integral max flow
- given a flow of value $\mathbf{v}$, can decompose into $\mathbf{O(m+n)}$ flow paths of same total value $\mathbf{v}$. integral flow implies integral flow on paths.
- maximum flow is equal to the minimum cut and minimum cut can be found in $\mathbf{O(m+n)}$ time given any maximum flow

Paths, Cycles and Acyclicity of Flows

Definition

Given a flow network $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ and a flow $\mathbf{f} : \mathbf{E} \rightarrow \mathbb{R}^{\geq 0}$ on the edges, the support of $\mathbf{f}$ is the set of edges $\mathbf{E}' \subseteq \mathbf{E}$ with non-zero flow on them. That is, $\mathbf{E}' = \{\mathbf{e} \in \mathbf{E} \mid \mathbf{f}(\mathbf{e}) > 0\}$.

Question: Given flow $\mathbf{f}$, can there by cycles in its support?



Acyclicity of Flows

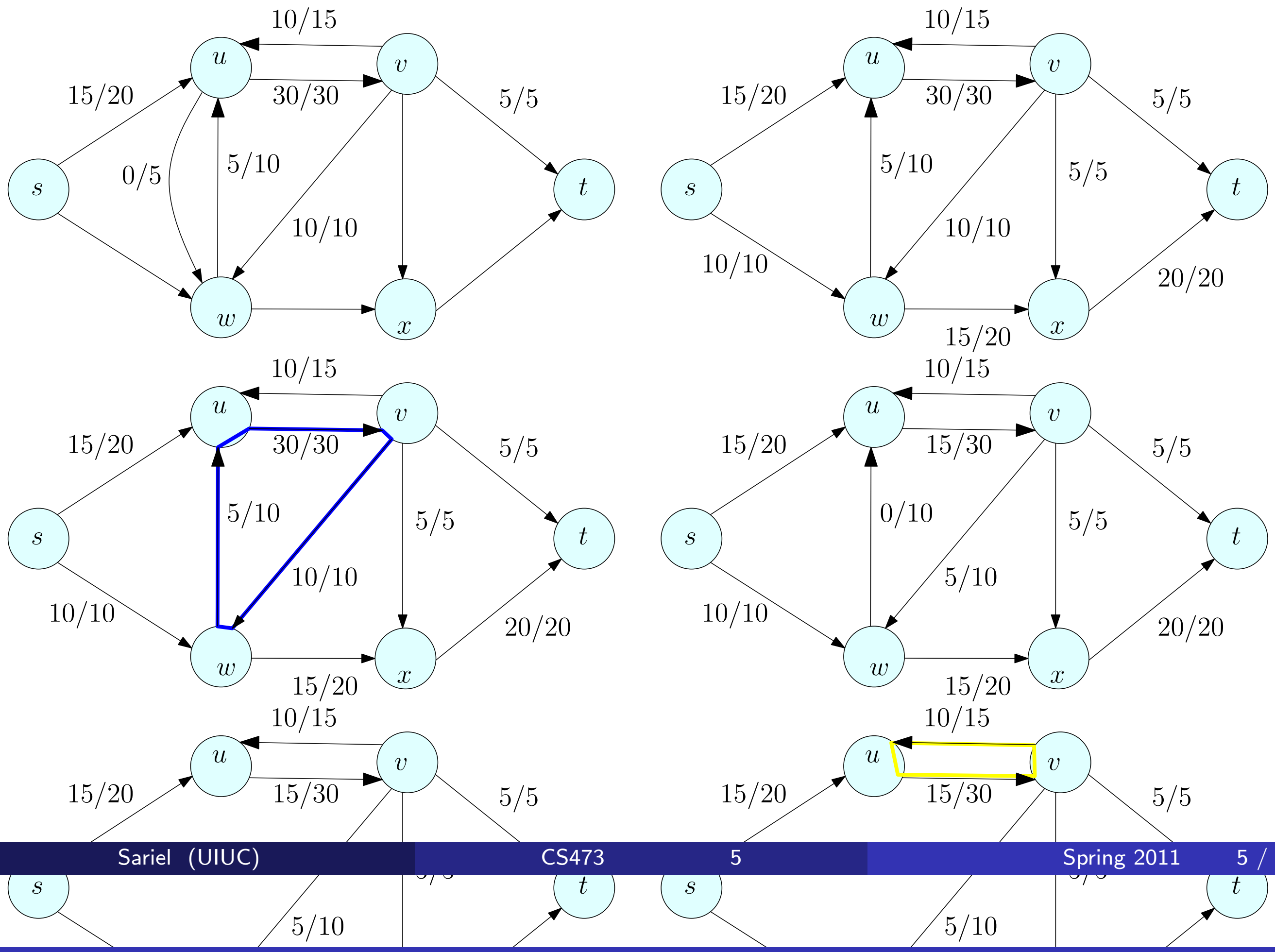
Proposition

In any flow network, if $\mathbf{f}$ is a flow then there is another flow $\mathbf{f}'$ such that the support of $\mathbf{f}'$ is an acyclic *graph and $\mathbf{v}(\mathbf{f}') = \mathbf{v}(\mathbf{f})$. Further if $\mathbf{f}$ is an integral flow then so is $\mathbf{f}'$.*

Proof.

- $\mathbf{E}' = \{\mathbf{e} \in \mathbf{E} \mid \mathbf{f}(\mathbf{e}) > 0\}$, support of $\mathbf{f}$.
- Suppose there is a directed cycle $\mathbf{C}$ in $\mathbf{E}'$
- Let $\mathbf{e}'$ be the edge in $\mathbf{C}$ with least amount of flow
- For each $\mathbf{e} \in \mathbf{C}$, reduce flow by $\mathbf{f}(\mathbf{e}')$. Remains a flow. Why?
- flow on $\mathbf{e}'$ is reduced to $\mathbf{0}$
- Claim: flow value from $\mathbf{s}$ to $\mathbf{t}$ does not change. Why?
- Iterate until no cycles $\square$

Example



Flow Decomposition

Lemma

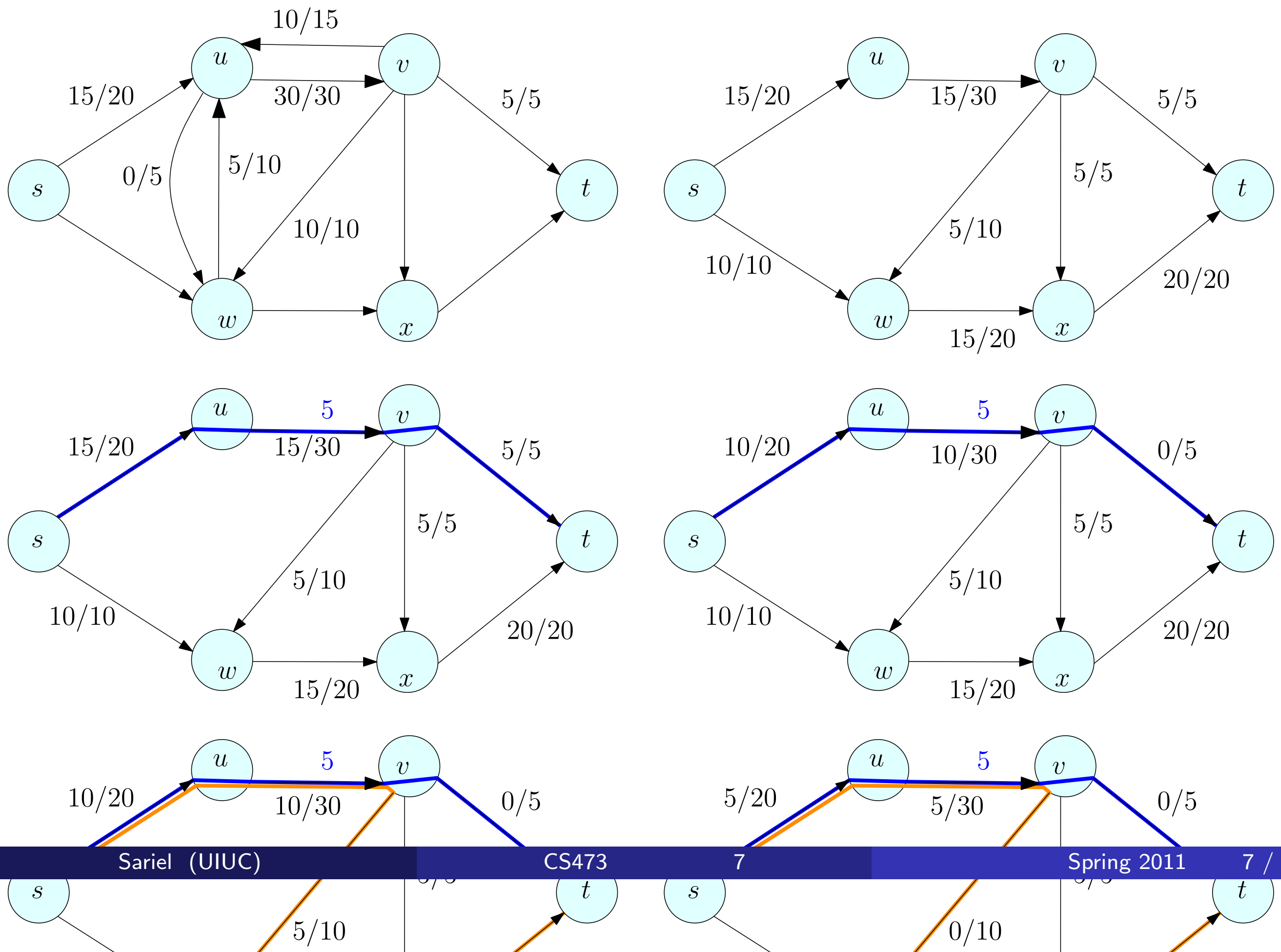
Given an edge based flow $\mathbf{f} : \mathbf{E} \rightarrow \mathbb{R}^{\geq 0}$, *there exists a collection of paths* $\mathcal{P}$ *and cycles* $\mathcal{C}$ *and an assignment of flow to them* $\mathbf{f}' : \mathcal{P} \cup \mathcal{C} \rightarrow \mathbb{R}^{\geq 0}$ *such that:*

- $|\mathcal{P} \cup \mathcal{C}| \leq \mathbf{m}$
- *for each* $\mathbf{e} \in \mathbf{E}$, $\sum_{\mathbf{P} \in \mathcal{P} : \mathbf{e} \in \mathbf{P}} \mathbf{f}'(\mathbf{P}) + \sum_{\mathbf{C} \in \mathcal{C} : \mathbf{e} \in \mathbf{C}} \mathbf{f}'(\mathbf{C}) = \mathbf{f}(\mathbf{e})$
- $\mathbf{v}(\mathbf{f}) = \sum_{\mathbf{P} \in \mathcal{P}} \mathbf{f}'(\mathbf{P})$.
- *if* $\mathbf{f}$ *is integral then so are* $\mathbf{f}'(\mathbf{P})$ *and* $\mathbf{f}'(\mathbf{C})$ *for all* $\mathbf{P}$ *and* $\mathbf{C}$

Proof Idea.

- first remove all cycles as in previous proposition
- then decompose into paths as in previous lecture.
- Exercise: verify claims. □

Example



Flow Decomposition

Lemma

Given an edge based flow $\mathbf{f : E \to \mathbb{R}^{\geq 0}}$*, there exists a collection of paths* $\mathcal{P}$ *and cycles* $\mathcal{C}$ *and an assignment of flow to them* $\mathbf{f' : \mathcal{P} \cup \mathcal{C} \to \mathbb{R}^{\geq 0}}$ *such that:*

- $|\mathcal{P} \cup \mathcal{C}| \leq \mathbf{m}$
- *for each* $\mathbf{e \in E}$, $\sum_{\mathbf{P \in \mathcal{P} : e \in P}} \mathbf{f'(P)} + \sum_{\mathbf{C \in \mathcal{C} : e \in C}} \mathbf{f'(C)} = \mathbf{f(e)}$
- $\mathbf{v(f)} = \sum_{\mathbf{P \in \mathcal{P}}} \mathbf{f'(P)}$.
- *if* $\mathbf{f}$ *is integral then so are* $\mathbf{f'(P)}$ *and* $\mathbf{f'(C)}$ *for all* $\mathbf{P}$ *and* $\mathbf{C}$

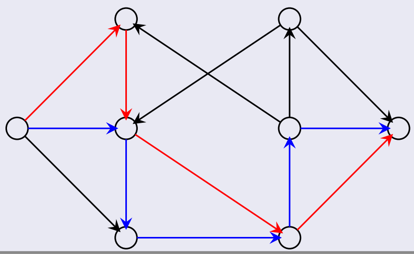
Above flow decomposition can be computed in $\mathbf{O(m^2)}$ *time.*

Part I

Network Flow Applications I

Edge-Disjoint Paths in Directed Graphs

Definition



A set of paths is edge disjoint if no two paths share an edge.

Problem

Given a directed graph with two special vertices **s** and **t**, find the *maximum* number of edge disjoint paths from **s** to **t**

Applications: Fault tolerance in routing — edges/nodes in networks can fail. Disjoint paths allow for planning backup routes in case of failures.

Reduction to Max-Flow

Problem

Given a directed graph $\mathbf{G}$ with two special vertices $\mathbf{s}$ and $\mathbf{t}$, find the maximum number of edge disjoint paths from $\mathbf{s}$ to $\mathbf{t}$

Reduction

Consider $\mathbf{G}$ as a flow network with edge capacities 1, and find max-flow.

Correctness of Reduction

Lemma

If $\mathbf{G}$ has $\mathbf{k}$ edge disjoint paths $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_k$ then there is an $\mathbf{s}$-$\mathbf{t}$ flow of value $\mathbf{k}$.

Proof.

Set $\mathbf{f(e) = 1}$ if $\mathbf{e}$ belongs to one of the paths $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_k$; other-wise set $\mathbf{f(e) = 0}$. This defines a flow of value $\mathbf{k}$. □

Correctness of Reduction

Lemma

If $\mathbf{G}$ has a flow of value $\mathbf{k}$ then there are $\mathbf{k}$ edge disjoint paths between $\mathbf{s}$ and $\mathbf{t}$.

Proof.

- Capacities are all $\mathbf{1}$ and hence there is integer flow of value $\mathbf{k}$, that is $\mathbf{f(e) = 0}$ or $\mathbf{f(e) = 1}$ for each $\mathbf{e}$.
- Decompose flow into paths of same value
- Flow on each path is either $\mathbf{1}$ or $\mathbf{0}$
- Hence there are $\mathbf{k}$ paths $\mathbf{P_1, P_2, \ldots, P_k}$ with flow of $\mathbf{1}$ each
- Paths are edge-disjoint since capacities are $\mathbf{1}$.

$\square$

Running Time

Theorem

The number of edge disjoint paths in $\mathbf{G}$ can be found in $\mathbf{O(mn)}$ time.

Run Ford-Fulkerson algorithm. Maximum possible flow is $\mathbf{n}$ and hence run-time is $\mathbf{O(nm)}$.

Menger's Theorem

Theorem (Menger)

Let G be a directed graph. The minimum number of edges whose removal disconnects s from t (the minimum-cut between s and t) is equal to the maximum number of edge-disjoint paths in G between s and t.

Proof.

Maxflow-mincut theorem and integrality of flow. □

Menger proved his theorem before Maxflow-Mincut theorem!
Maxflow-Mincut theorem is a generalization of Menger's theorem to capacitated graphs.

Edge Disjoint Paths in Undirected Graphs

Problem

Given an undirected graph G, find the maximum number of edge disjoint paths in G

Reduction:

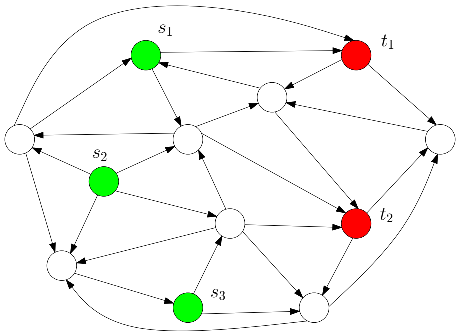
- create directed graph H by adding directed edges (u, v) and (v, u) for each edge uv in G.
- compute maximum s-t flow in H

Problem: Both edges (u, v) and (v, u) may have non-zero flow!

Not a Problem! Can assume maximum flow in H is acyclic and hence cannot have non-zero flow on both (u, v) and (v, u). Reduction works. See book for more details.

Multiple Sources and Sinks

- Directed graph $\mathbf{G}$ with edge capacities $\mathbf{c(e)}$
- source nodes $\mathbf{s_1, s_2, \ldots, s_k}$
- sink nodes $\mathbf{t_1, t_2, \ldots, t_\ell}$
- sources and sinks are *disjoint*



Multiple Sources and Sinks

- Directed graph $\mathbf{G}$ with edge capacities $\mathbf{c(e)}$
- source nodes $\mathbf{s_1, s_2, \ldots, s_k}$
- sink nodes $\mathbf{t_1, t_2, \ldots, t_\ell}$
- sources and sinks are *disjoint*

Maximum Flow: send as much flow as possible from the sources to the sinks. *Sinks don't care which source they get flow from.*

Minimum Cut: find a minimum capacity set of edge $\mathbf{E'}$ such that removing $\mathbf{E'}$ disconnects every source from every sink.

Multiple Sources and Sinks: Formal Definition

- Directed graph $\mathbf{G}$ with edge capacities $\mathbf{c(e)}$
- source nodes $\mathbf{s_1, s_2, \ldots, s_k}$
- sink nodes $\mathbf{t_1, t_2, \ldots, t_\ell}$
- sources and sinks are *disjoint*

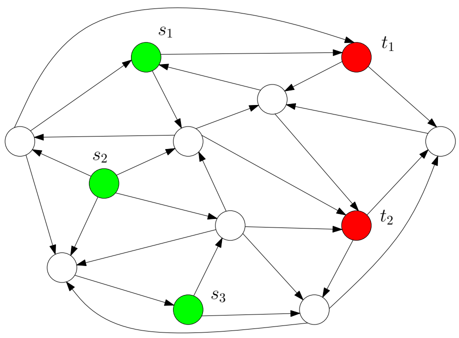
A function $\mathbf{f : E \to \mathbb{R}^{\geq 0}}$ is a flow if:

- for each $\mathbf{e \in E}$, $\mathbf{f(e) \leq c(e)}$ and
- for each $\mathbf{v}$ which is not a source or a sink $\mathbf{f^{in}(v) = f^{out}(v)}$.

Goal: max $\sum_{i=1}^{k}(\mathbf{f^{out}(s_i) - f^{in}(s_i)})$, that is, flow out of sources

Reduction to Single-Source Single-Sink

- Add a *source* node $\mathbf{s}$ and a *sink* node $\mathbf{t}$
- Add edges $\mathbf{(s, s_1), (s, s_2), \ldots, (s, s_k)}$
- Add edges $\mathbf{(t_1, t), (t_2, t), \ldots, (t_\ell, t)}$
- Set the capacity of the new edges to be ∞

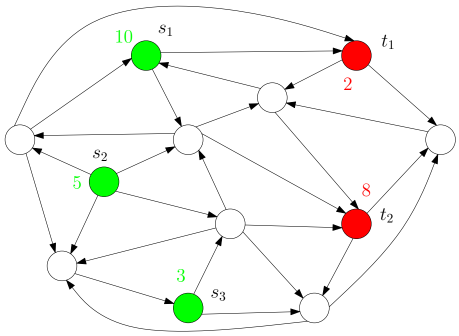


Supplies and Demands

A further generalization:

- source s_i has a supply of $S_i \geq 0$
- since t_j has a demand of $D_j \geq 0$ units

Question: is there a flow from source to sinks such that supplies are not exceeded and demands are met? Formally we have the additional constraints that $f^{out}(s_i) - f^{in}(s_i) \leq S_i$ for each source s_i and $f^{in}(t_j) - f^{out}(t_j) \leq D_j$ for each sink t_j.

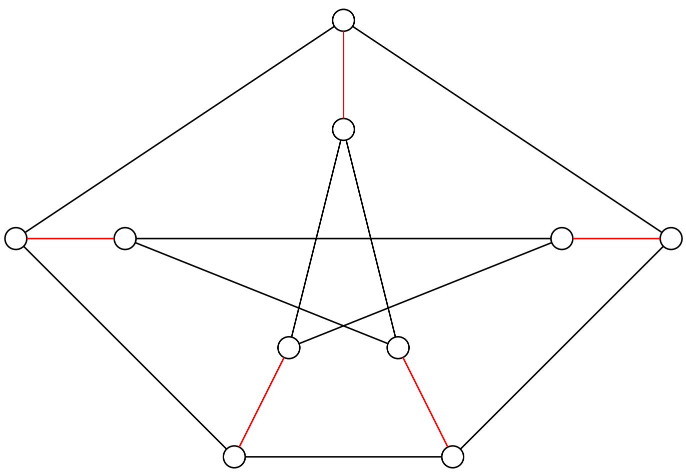


Matching

Input Given a (undirected) graph $G = (V, E)$

Goal Find a matching of maximum cardinality

- A matching is $M \subseteq E$ such that at most one edge in M is incident on any vertex



Bipartite Matching

Input Given a bipartite graph $\mathbf{G = (L \cup R, E)}$

Goal Find a matching of maximum cardinality

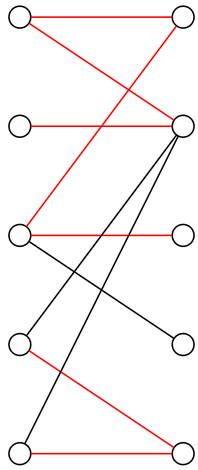
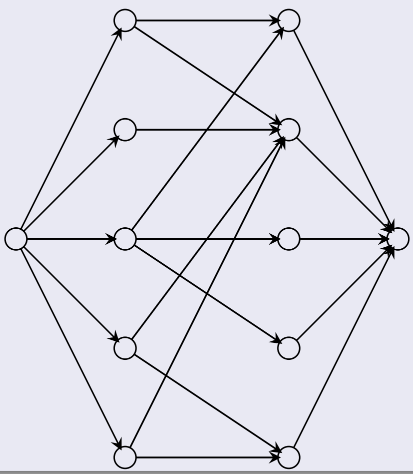


Figure: Maximum matching has 4 edges

Reduction to Max-Flow

Max-Flow Construction

Given graph $\mathbf{G = (L \cup R, E)}$ create flow-network $\mathbf{G' = (V', E')}$ as follows:



- $\mathbf{V' = L \cup R \cup \{s, t\}}$ where $\mathbf{s}$ and $\mathbf{t}$ are the new source and sink
- Direct all edges in $\mathbf{E}$ from $\mathbf{L}$ to $\mathbf{R}$, and add edges from $\mathbf{s}$ to all vertices in $\mathbf{L}$ and from each vertex in $\mathbf{R}$ to $\mathbf{t}$
- Capacity of every edge is $\mathbf{1}$

Correctness: Matching to Flow

Proposition

If $\mathbf{G}$ has a matching of size $\mathbf{k}$ then $\mathbf{G}'$ has a flow of value $\mathbf{k}$.

Proof.

Let $\mathbf{M}$ be matching of size $\mathbf{k}$. Let $\mathbf{M} = \{(u_1, v_1), \ldots, (u_k, v_k)\}$. Consider following flow $\mathbf{f}$ in $\mathbf{G}'$:

- $f(s, u_i) = 1$ and $f(v_i, t) = 1$ for $1 \leq i \leq k$
- $f(u_i, v_i) = 1$ for $1 \leq i \leq k$
- for all other edges flow is zero.

Verify that $\mathbf{f}$ is a flow of value $\mathbf{k}$ (because $\mathbf{M}$ is a matching). □

Correctness: Flow to Matching

Proposition

If $\mathbf{G}'$ has a flow of value $\mathbf{k}$ then $\mathbf{G}$ has a matching of size $\mathbf{k}$.

Proof.

Consider flow $\mathbf{f}$ of value $\mathbf{k}$.

- Can assume $\mathbf{f}$ is integral. Thus each edge has flow $\mathbf{1}$ or $\mathbf{0}$
- Consider the set $\mathbf{M}$ of edges from $\mathbf{L}$ to $\mathbf{R}$ that have flow 1
  - $\mathbf{M}$ has $\mathbf{k}$ edges because value of flow is equal to the number of non-zero flow edges crossing cut $(L \cup \{s\}, R \cup \{t\})$
  - Each vertex has at most one edge in $\mathbf{M}$ incident upon it. Why?

□

Correctness of Reduction

Theorem

The maximum flow value in $\mathbf{G}'$ = maximum cardinality of matching in $\mathbf{G}$

Consequence

Thus, to find maximum cardinality matching in $\mathbf{G}$, we construct $\mathbf{G}'$ and find the maximum flow in $\mathbf{G}'$. Note that the matching itself (not just the value) can be found efficiently from the flow.

Running Time

For graph $\mathbf{G}$ with $\mathbf{n}$ vertices and $\mathbf{m}$ edges $\mathbf{G}'$ has $\mathbf{O(n+m)}$ edges, and $\mathbf{O(n)}$ vertices.

- Generic Ford-Fulkerson: Running time is $\mathbf{O(mC) = O(nm)}$ since $\mathbf{C = n}$
- Capacity scaling: Running time is $\mathbf{O(m^2 \log C) = O(m^2 \log n)}$

Better known running time: $\mathbf{O(m\sqrt{n})}$

Perfect Matchings

Definition

A matching $\mathbf{M}$ is said to be perfect if every vertex has one edge in $\mathbf{M}$ incident upon it.

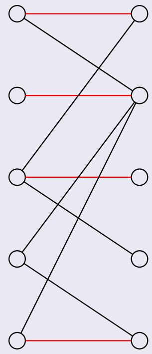


Figure: This graph does not have a perfect matching

Characterizing Perfect Matchings

Problem

When does a bipartite graph have a perfect matching?

- Clearly $|\mathbf{L}| = |\mathbf{R}|$
- Are there any necessary and sufficient conditions?

A Necessary Condition

Lemma

If $\mathbf{G} = (\mathbf{L} \cup \mathbf{R}, \mathbf{E})$ *has a perfect matching then for any* $\mathbf{X} \subseteq \mathbf{L}$, $|\mathbf{N}(\mathbf{X})| \geq |\mathbf{X}|$, *where* $\mathbf{N}(\mathbf{X})$ *is the set of neighbors of vertices in* $\mathbf{X}$

Proof.

Since $\mathbf{G}$ has a perfect matching, every vertex of $\mathbf{X}$ is matched to a different neighbor, and so $|\mathbf{N}(\mathbf{X})| \geq |\mathbf{X}|$ $\square$

Hall's Theorem

Theorem (Frobenius-Hall)

Let $\mathbf{G} = (\mathbf{L} \cup \mathbf{R}, \mathbf{E})$ *be a bipartite graph with* $|\mathbf{L}| = |\mathbf{R}|$. $\mathbf{G}$ *has a perfect matching if and only if for every* $\mathbf{X} \subseteq \mathbf{L}$, $|\mathbf{N}(\mathbf{X})| \geq |\mathbf{X}|$

One direction is the necessary condition.
For the other direction we will show the following:

- create flow network $\mathbf{G}'$ from $\mathbf{G}$
- if $|\mathbf{N}(\mathbf{X})| \geq |\mathbf{X}|$ for all $\mathbf{X}$, show that minimum $\mathbf{s}$-$\mathbf{t}$ cut in $\mathbf{G}'$ is of capacity $\mathbf{n} = |\mathbf{L}| = |\mathbf{R}|$
- implies that $\mathbf{G}$ has a perfect matching

Proof of Sufficiency

Assume $|N(X)| \geq |X|$ for each $X \in L$. Then show that min s-t cut in G' is of capacity n.

Let (A, B) be an *arbitrary* s-t cut in G'

- let $X = A \cap L$ and $Y = A \cap R$
- cut capacity is at least $(|L| - |X|) + |Y| + |N(X) - Y|$
- $|N(X) - Y| \geq |N(X)| - |Y|$ and by assumption $|N(X)| \geq |X|$ and hence $|N(X) - Y| \geq |X| - |Y|$
- cut capacity is therefore at least $|L| - |X| + |Y| + |X| - |Y| \geq |L| = n$.

Application: assigning jobs to people

- n jobs or tasks
- m people
- for each job a set of people who can do that job
- for each person j a limit on number of jobs k_j
- Goal: find an assignment of jobs to people so that all jobs are assigned and no person is overloaded

Reduce to max-flow similar to matching.

Arises in many settings. Using *minimum-cost flows* can also handle the case when assigning a job i to person j costs c_{ij} and goal is assign all jobs but minimize cost of assignment.

Reduction to Maximum Flow

- Create directed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ as follows
  - $\mathbf{V} = \{\mathbf{s}, \mathbf{t}\} \cup \mathbf{L} \cup \mathbf{R}$: $\mathbf{L}$ set of $\mathbf{n}$ jobs, $\mathbf{R}$ set of $\mathbf{m}$ people
  - add edges $(\mathbf{s}, \mathbf{i})$ for each job $\mathbf{i} \in \mathbf{L}$, capacity $\mathbf{1}$
  - add edges $(\mathbf{j}, \mathbf{t})$ for each person $\mathbf{j} \in \mathbf{R}$, capacity $\mathbf{k_j}$
  - if job $\mathbf{i}$ can be done by person $\mathbf{j}$ add an edge $(\mathbf{i}, \mathbf{j})$, capacity $\mathbf{1}$
- Compute max $\mathbf{s}$-$\mathbf{t}$ flow. There is an assignment if and only if flow value is $\mathbf{n}$.

Matchings in General Graphs

Matchings in general graphs more complicated.

There is a polynomial time algorithm to compute a maximum matching in a general graph. Best known running time is $\mathbf{O}(\mathbf{m}\sqrt{\mathbf{n}})$.