BBM402-Lecture 17: Applications of Network Flows

Lecturer: Lale Özkahya

Resources for the presentation:
https://courses.engr.illinois.edu/cs473/fa2016/lectures.html

Is the flow always integral?

Let G be an integral instance of network flow (i.e., all numbers are integers). Consider the following statements:

(I) The value of the maximum flow is an integer number.

(II) If f is a maximum flow, then $f(e)$ is an integer, for any edge $e \in \mathsf{E}(\mathsf{G})$.

(III) There always exists a max flow g, such that g is a maximum flow, and $g(e)$ is an integer, for any edge $e \in \mathsf{E}(\mathsf{G})$.

We have the following:

(A) All the above statements are false.

(B) All the above statements are true.

(C) (I) is true, (II) and (III) are false.

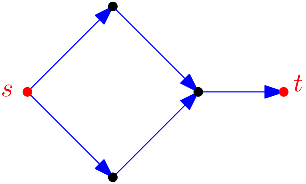
(D) (I) and (II) are true, and (III) is false.

(E) (I) and (III) are true, and (II) is false.

Why max-flow does not have to be integral...

...but the one we compute always is!

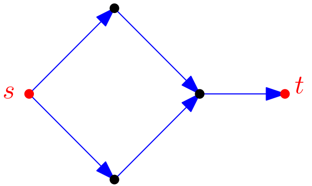
Consider the graph with all capacities being one.



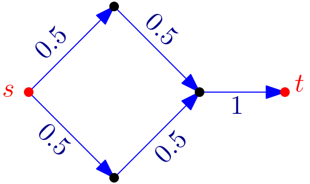
Why max-flow does not have to be integral...

...but the one we compute always is!

Consider the graph with all capacities being one.



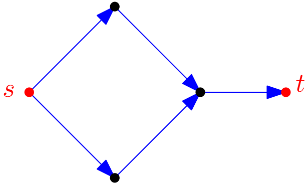
One possible max flow:



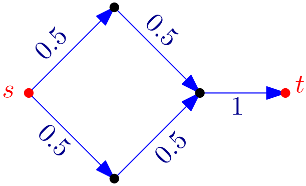
Why max-flow does not have to be integral...

...but the one we compute always is!

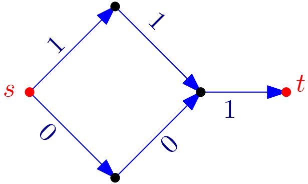
Consider the graph with all capacities being one.



One possible max flow:



Max flow as computed by **algEdmondsKarp** or **algFordFulkerson**:



Network Flow: Facts to Remember

Flow network: directed graph G, capacities c, source s, sink t.

1. Maximum s-t flow can be computed:
    1. Using Ford-Fulkerson algorithm in $O(mC)$ time when capacities are integral and C is an upper bound on the flow.
    2. Using variant of algorithm, in $O(m^2 \log C)$ time, when capacities are integral. (Polynomial time.)
    3. Using Edmonds-Karp algorithm, in $O(m^2 n)$ time, when capacities are rational (strongly polynomial time algorithm).
    4. There is an $O(mn)$ time algorithm due to Orlin which is the currently fastest strongly polynomial-time algorithm.

Network Flow

Even more facts to remember

1. If capacities are integral then there is a maximum flow that is integral and above algorithms give an integral max flow. This is known as **integrality of flow**.
2. Given a flow of value v, can decompose into $O(m+n)$ flow paths of same total value v. Integral flow implies integral flow on paths.
3. Maximum flow is equal to the minimum cut and minimum cut can be found in $O(m+n)$ time given any maximum flow.

Paths, Cycles and Acyclicity of Flows

Definition

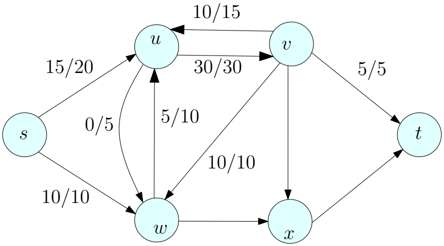
Given a flow network $G = (V, E)$ and a flow $f : E \to \mathbb{R}^{\geq 0}$ on the edges, the **support** of f is the set of edges $E' \subseteq E$ with non-zero flow on them. That is, $E' = \{e \in E \mid f(e) > 0\}$.

Paths, Cycles and Acyclicity of Flows

Definition

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

Question:Given a flow f, can there by cycles in its support?

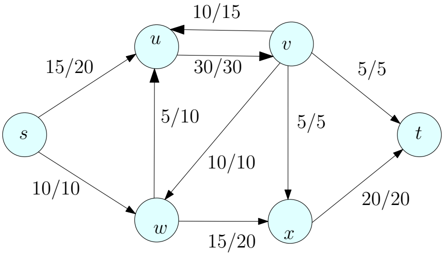


Paths, Cycles and Acyclicity of Flows

Definition

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

Question:Given a flow f, can there by cycles in its support?

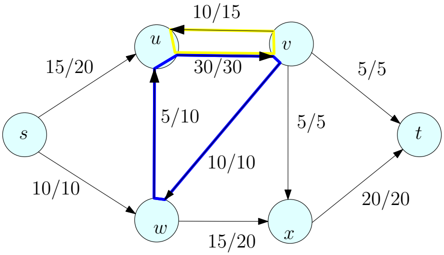


Paths, Cycles and Acyclicity of Flows

Definition

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

Question:Given a flow f, can there by cycles in its support?



How fast can we detect a cycle in the flow

Given a flow network G with n vertices, and m edges, and a flow f on it, then detecting a cycle in the flow can be done in time

(A) $O(m + n)$.

(B) $O(mC)$.

(C) $O(mn)$.

(D) $O(m^2 n)$.

(E) $O(mn^2)$.

Acyclicity of Flows

Proposition

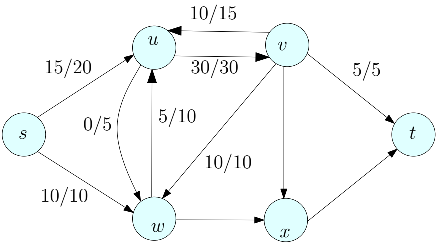
In any flow network, if f is a flow then there is another flow f' such that the support of f' is an acyclic *graph and $v(f') = v(f)$. Further if f is an integral flow then so is f'.*

Proof.

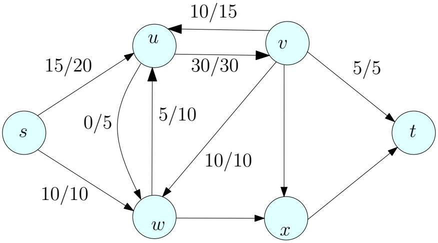
1. $E' = \{e \in E \mid f(e) > 0\}$, support of f.
2. Suppose there is a directed cycle C in E'
3. Let e' be the edge in C with least amount of flow
4. For each $e \in C$, reduce flow by $f(e')$. Remains a flow. Why?
5. Flow on e' is reduced to 0.
6. Claim: Flow value from s to t does not change. Why?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

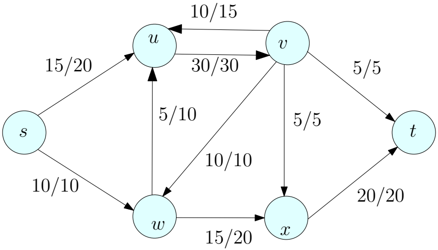
□

Example



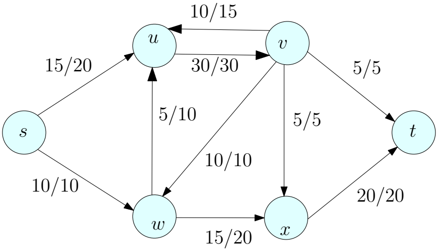
Example

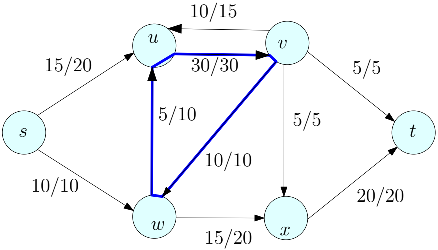




Throw away edge with no flow on it

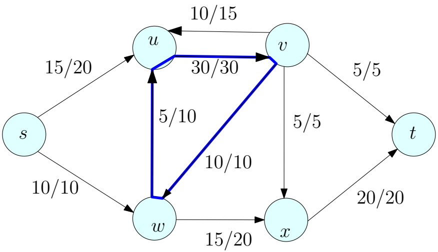
Example

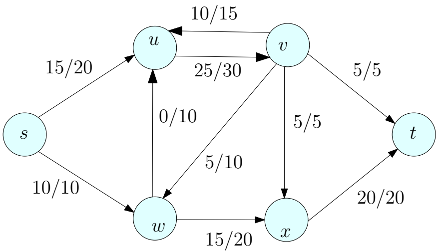




Find a cycle in the support/flow

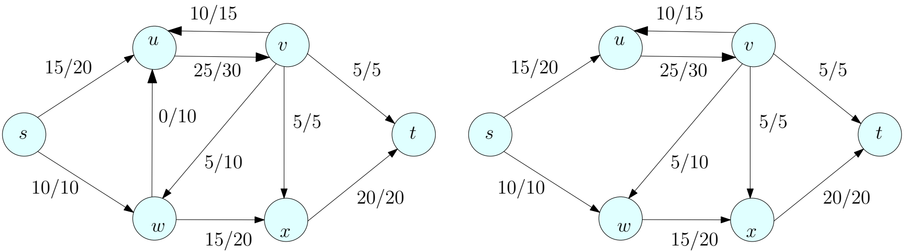
Example





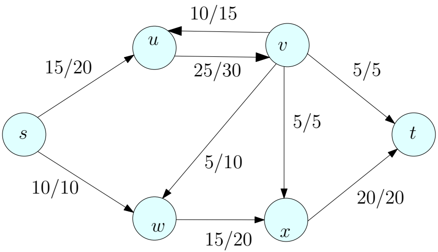
Reduce flow on cycle as much as possible

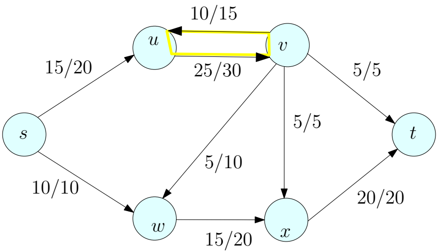
Example



Throw away edge with no flow on it

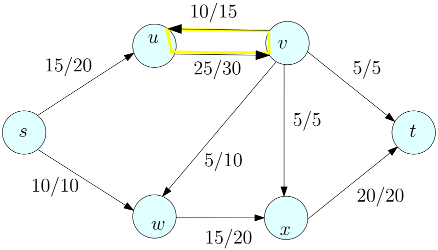
Example

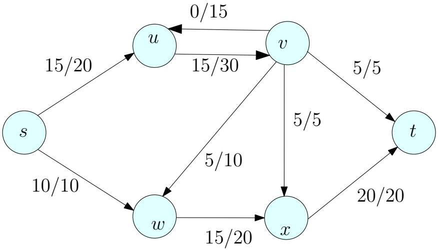




Find a cycle in the support/flow

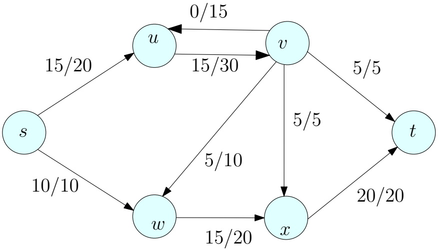
Example

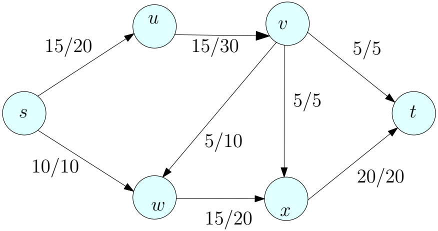




Reduce flow on cycle as much as possible

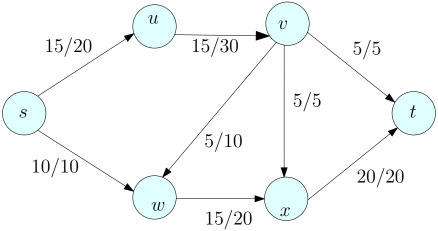
Example



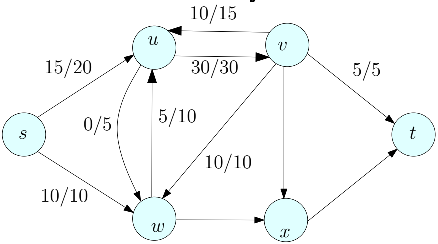


Throw away edge with no flow on it

Example



Viola!!! An equivalent flow with no cycles in it. Original flow:



Flow Decomposition

Lemma

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

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

Flow Decomposition

Lemma

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

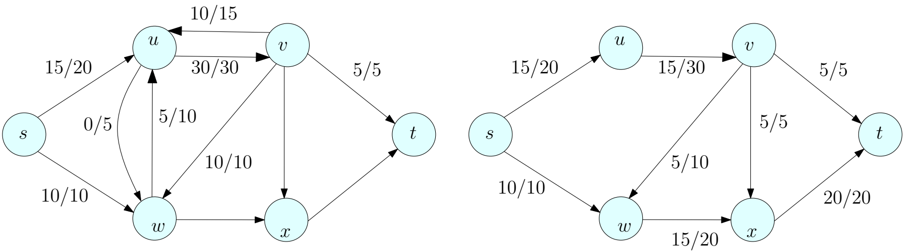
1. $|\mathcal{P} \cup \mathcal{C}| \leq m$
2. *for each* $e \in E$, $\sum_{P \in \mathcal{P} : e \in P} f'(P) + \sum_{C \in \mathcal{C} : e \in C} f'(C) = f(e)$
3. $v(f) = \sum_{P \in \mathcal{P}} f'(P)$.
4. *if* f *is integral then so are* $f'(P)$ *and* $f'(C)$ *for all* P *and* C

Proof Idea.

1. Remove all cycles as in previous proposition.
2. Next, decompose into paths as in previous lecture.
3. Exercise: verify claims.

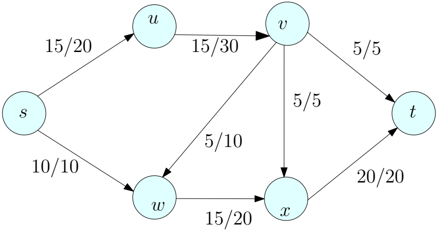
□

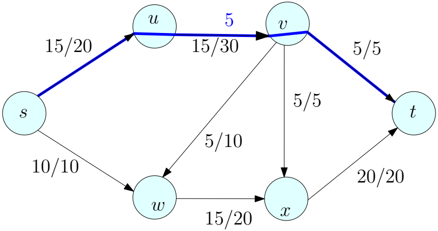
Example



Find cycles as shown before

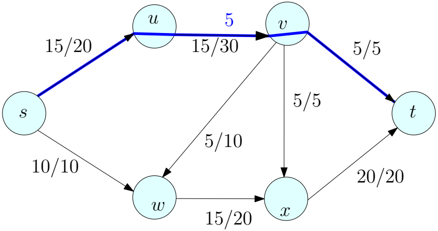
Example

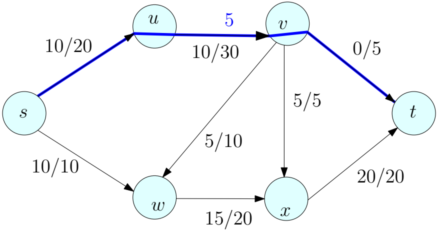




Find a source to sink path, and push max flow along it (5 unites)

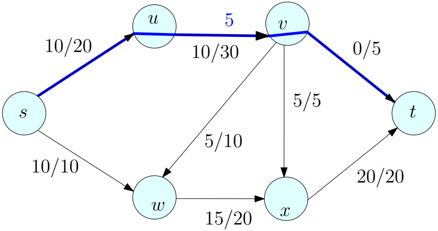
Example

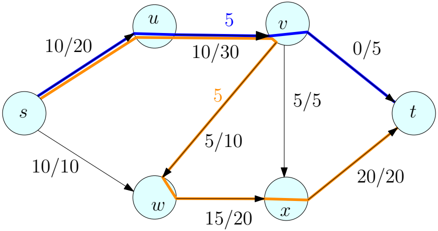




Compute remaining flow

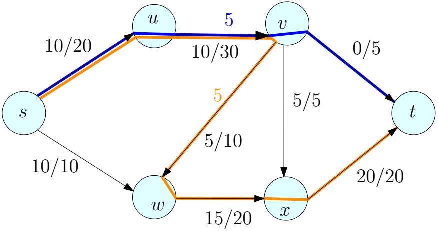
Example

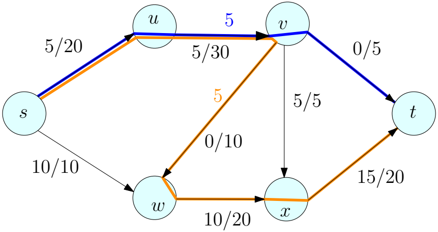




Find a source to sink path, and push max flow along it (5 unites). Edges with **0** flow on them can not be used as they are no longer in the support of the flow.

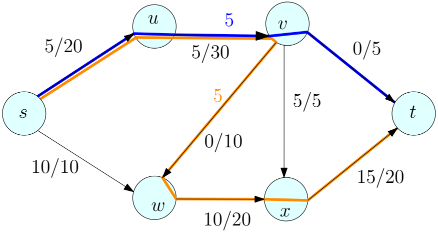
Example

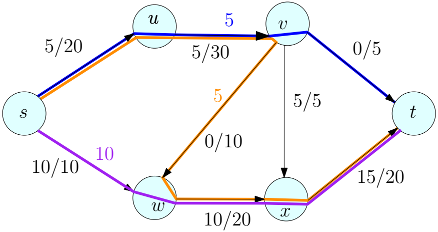




Compute remaining flow

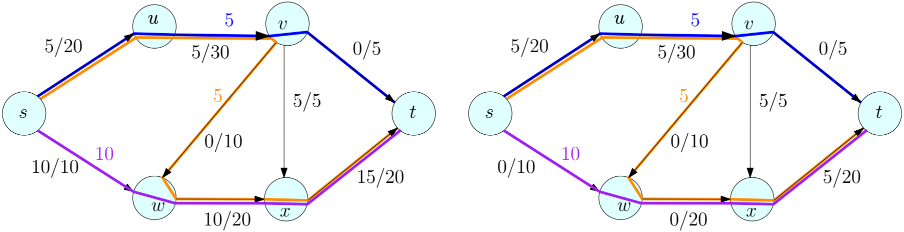
Example





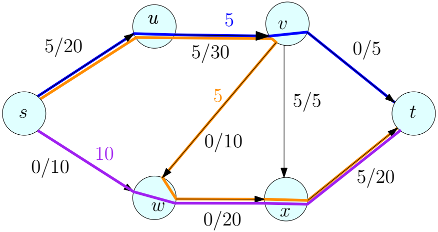
Find a source to sink path, and push max flow along it (10 unites).

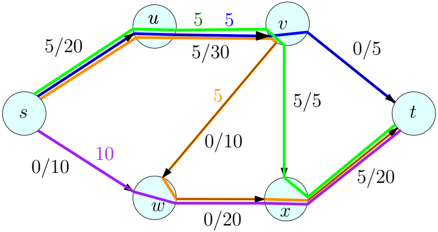
Example



Compute remaining flow

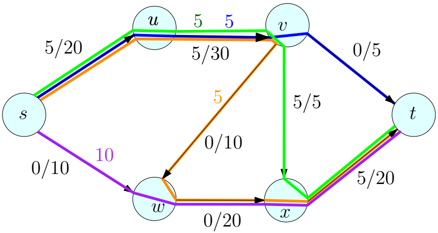
Example

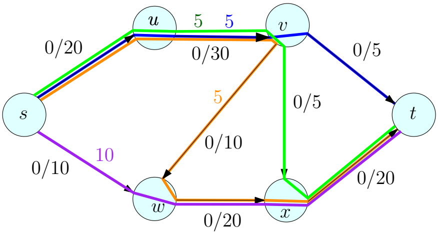




Find a source to sink path, and push max flow along it (5 unites).

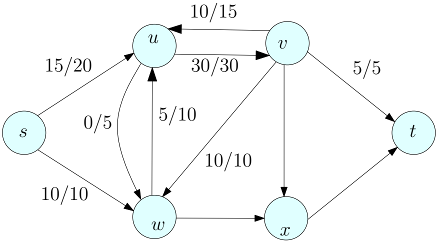
Example

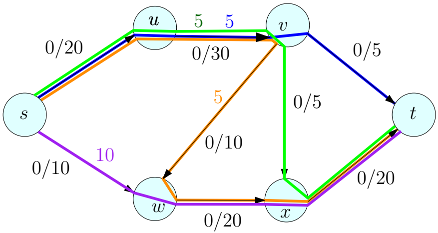




Compute remaining flow

Example





No flow remains in the graph. We fully decomposed the flow into flow on paths. Together with the cycles, we get a decomposition of the original flow into m flows on paths and cycles.

Flow Decomposition

Lemma

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

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

Above flow decomposition can be computed in $O(mn)$ *time.*

Exercise: Naive implementation of flow-decomposition takes $O(m^2)$ time. Show how to implement in $O(mn)$ time.

Flow decomposition into paths and cycles

Consider an integral flow network G, and two maximum flows f and g in G. Assume both f and g are acyclic. Let P_f and P_g be the decomposition of the two flows into paths. Then:

(A) $P_f = P_g$ (paths are the same).

(B) $|P_f| = |P_g|$ (i.e., number of paths is the same).

(C) $|P_f| + |P_g| = m$.

(D) $|P_f| * |P_g| = nm$.

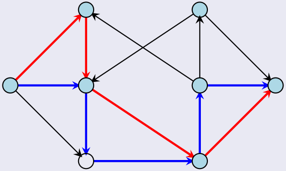
(E) None of the above.

Part I

Network Flow Applications I

Edge-Disjoint Paths in Directed Graphs

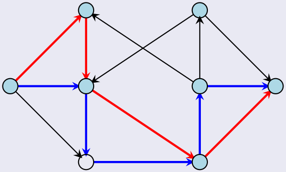
Definition



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

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 G with two special vertices s and t, find the maximum number of edge disjoint paths from s to t.

Reduction

Consider G as a flow network with edge capacities 1, and compute max-flow.

Correctness of Reduction

Lemma

If G has k edge disjoint paths $P_1, P_2, \ldots, P_k$ then there is an s-t flow of value k in G.

Correctness of Reduction

Lemma

If G has k edge disjoint paths $P_1, P_2, \ldots, P_k$ then there is an s-t flow of value k in G.

Proof.

Set $f(e) = 1$ if e belongs to one of the paths $P_1, P_2, \ldots, P_k$; other-wise set $f(e) = 0$. This defines a flow of value k. $\square$

Correctness of Reduction

Lemma

If G has a flow of value k then there are k edge disjoint paths between s and t.

Correctness of Reduction

Lemma

If G has a flow of value k then there are k edge disjoint paths between s and t.

Proof.

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

Running Time

Theorem

The number of edge disjoint paths in a simple graph G can be found in $O(mn)$ time.

Proof.

1. Set capacities of edges in G to 1.
2. Run Ford-Fulkerson algorithm.
3. Maximum value of flow is n and hence run-time is $O(nm)$.
4. Decompose flow into k paths ($k \leq n$).
   Takes $O(k \times m) = O(km) = O(mn)$ time. □

Running Time

Theorem

The number of edge disjoint paths in a simple graph G can be found in $O(mn)$ time.

Proof.

1. Set capacities of edges in G to 1.
2. Run Ford-Fulkerson algorithm.
3. Maximum value of flow is n and hence run-time is $O(nm)$.
4. Decompose flow into k paths ($k \leq n$).
   Takes $O(k \times m) = O(km) = O(mn)$ time. □

Remark

The algorithm also computes a set of edge-disjoint paths realizing this optimal solution.

Menger's Theorem

Theorem

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.

Menger's Theorem

Theorem

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. $\square$

Menger's Theorem

Theorem

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.

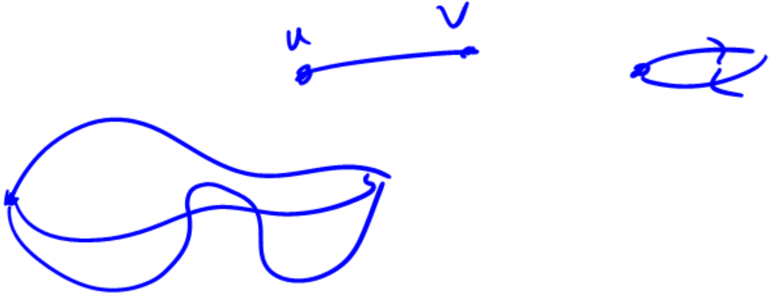
Maxflow-mincut theorem and integrality of flow. □

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



Edge Disjoint Paths in Undirected Graphs

Problem

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

Reduction:

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

Edge Disjoint Paths in Undirected Graphs

Problem

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

Reduction:

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

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

Edge Disjoint Paths in Undirected Graphs

Problem

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

Reduction:

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

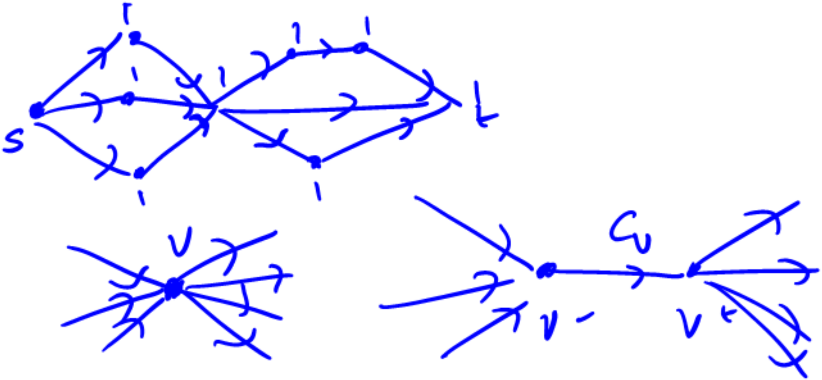
Problem: Both edges (u, v) and (v, u) may have non-zero flow!

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.

Node Disjoint Paths and Meger's theorem

Definition

A set of s-t paths $\mathcal{P}$ are *internally* node-disjoint if no two paths in $\mathcal{P}$ share a node other than s, t.



Node Disjoint Paths and Meger's theorem

Definition

A set of s-t paths $\mathcal{P}$ are *internally* node-disjoint if no two paths in $\mathcal{P}$ share a node other than s, t.

Theorem

Let G be an undirected graph. The minimum number of nodes in $V \setminus \{s, t\}$ whose removal disconnects s from t is equal to the maximum number of internally node-disjoint paths in G between s and t.

Node Disjoint Paths and Meger's theorem

Definition

A set of s-t paths $\mathcal{P}$ are *internally* node-disjoint if no two paths in $\mathcal{P}$ share a node other than s, t.

Theorem

Let G be an undirected graph. The minimum number of nodes in $V \setminus \{s, t\}$ whose removal disconnects s from t is equal to the maximum number of internally node-disjoint paths in G between s and t.

Theorem

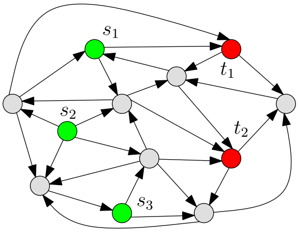
The max number of internally node-disjoint paths between s and t in G can be computed in $O(mn)$ time.

Via reductions to directed graph *edge*-disjoint case!

Multiple Sources and Sinks

Input:

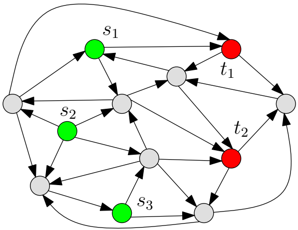
1. A directed graph G with edge capacities $c(e)$.
2. Source nodes $s_1, s_2, \ldots, s_k$.
3. Sink nodes $t_1, t_2, \ldots, t_\ell$.
4. Sources and sinks are *disjoint*.



Multiple Sources and Sinks

Input:

1. A directed graph G with edge capacities $c(e)$.
2. Source nodes $s_1, s_2, \dots, s_k$.
3. Sink nodes $t_1, t_2, \dots, t_\ell$.
4. Sources and sinks are *disjoint*.

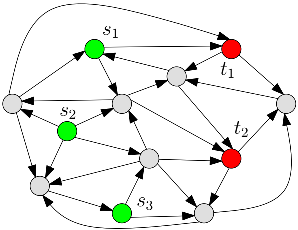


Maximum Flow: Send as much flow as possible from the sources to the sinks. *Sinks don't care which source they get flow from.*

Multiple Sources and Sinks

Input:

1. A directed graph G with edge capacities $c(e)$.
2. Source nodes $s_1, s_2, \ldots, s_k$.
3. Sink nodes $t_1, t_2, \ldots, t_\ell$.
4. 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 E' such that removing E' disconnects every source from every sink.

Multiple Sources and Sinks: Formal Definition

Input:

1. A directed graph G with edge capacities $c(e)$.
2. Source nodes $s_1, s_2, \ldots, s_k$.
3. Sink nodes $t_1, t_2, \ldots, t_\ell$.
4. Sources and sinks are *disjoint*.

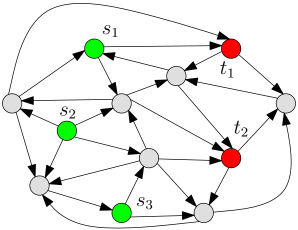
A function $f : E \to \mathbb{R}^{\geq 0}$ is a **flow** if:

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

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

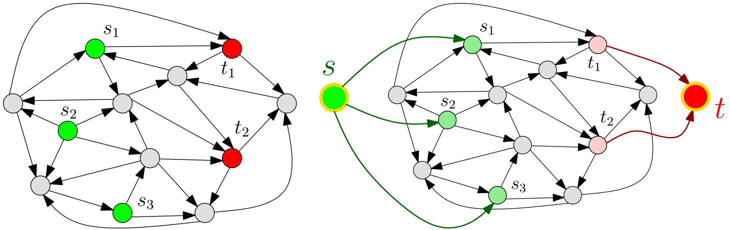
Reduction to Single-Source Single-Sink

1. Add a **source** node s and a **sink** node t.
2. Add edges $(s, s_1), (s, s_2), \ldots, (s, s_k)$.
3. Add edges $(t_1, t), (t_2, t), \ldots, (t_\ell, t)$.
4. Set the capacity of the new edges to be ∞.



Reduction to Single-Source Single-Sink

1. Add a **source** node s and a **sink** node t.
2. Add edges $(s, s_1), (s, s_2), \ldots, (s, s_k)$.
3. Add edges $(t_1, t), (t_2, t), \ldots, (t_\ell, t)$.
4. Set the capacity of the new edges to be ∞.

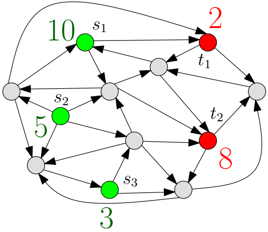


Supplies and Demands

A further generalization:

1. source s_i has a supply of $S_i \geq 0$
2. 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 $f^{\text{out}}(s_i) - f^{\text{in}}(s_i) \leq S_i$ for each source s_i and $f^{\text{in}}(t_j) - f^{\text{out}}(t_j) \geq D_j$ for each sink t_j.

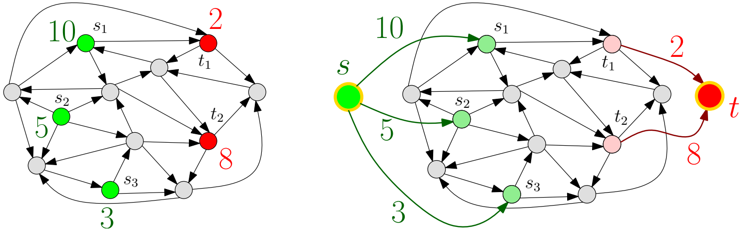


Supplies and Demands

A further generalization:

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

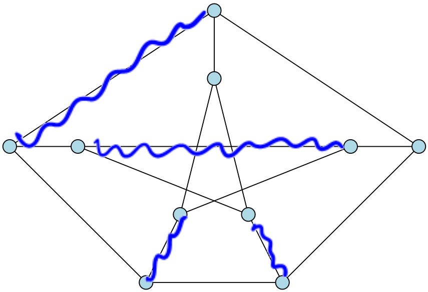
Question: is there a flow from source to sinks such that supplies are not exceeded and demands are met? Formally we have the additional constraints that $f^{\text{out}}(s_i) - f^{\text{in}}(s_i) \leq S_i$ for each source s_i and $f^{\text{in}}(t_j) - f^{\text{out}}(t_j) \geq D_j$ for each sink t_j.



Matching

Problem (Matching)

Input: *Given a (undirected) graph* $\mathbf{G = (V, E)}$.
Goal: *Find a matching of maximum cardinality.*

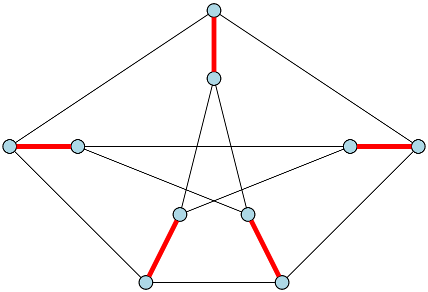


Matching

Problem (Matching)

Input: *Given a (undirected) graph* $G = (V, E)$.
Goal: *Find a matching of maximum cardinality.*

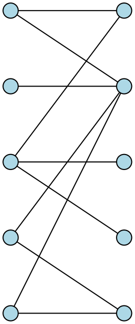
1. *A matching is* $M \subseteq E$ *such that at most one edge in* M *is incident on any vertex*



Bipartite Matching

Problem (Bipartite matching)

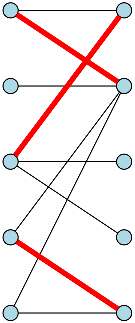
Input: *Given a bipartite graph* $G = (L \cup R, E)$.
Goal: *Find a matching of maximum cardinality*



Bipartite Matching

Problem (Bipartite matching)

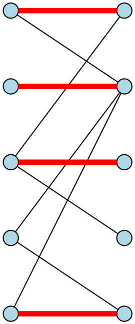
Input: *Given a bipartite graph* $G = (L \cup R, E)$.
Goal: *Find a matching of maximum cardinality*



Bipartite Matching

Problem (Bipartite matching)

Input: *Given a bipartite graph* $G = (L \cup R, E)$.
Goal: *Find a matching of maximum cardinality*

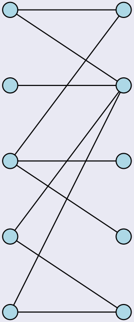


Maximum matching has 4 edges

Reduction of bipartite matching to max-flow

Max-Flow Construction

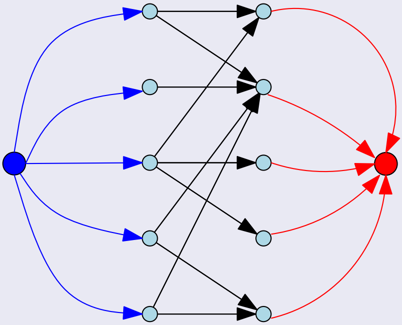
Given graph $G = (L \cup R, E)$ create flow-network $G' = (V', E')$ as follows:



Reduction of bipartite matching to max-flow

Max-Flow Construction

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

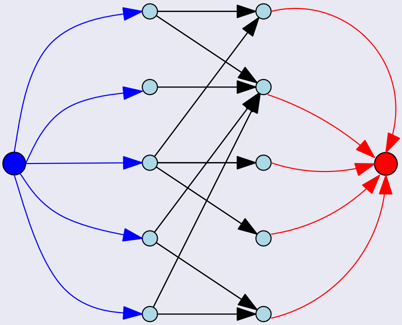


1. $V' = L \cup R \cup \{s, t\}$ where s and t are the new source and sink.

Reduction of bipartite matching to max-flow

Max-Flow Construction

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

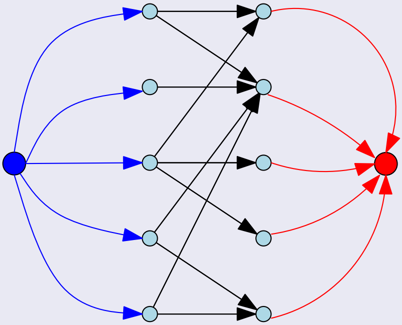


1. $V' = L \cup R \cup \{s, t\}$ where s and t are the new source and sink.
2. Direct all edges in E from L to R, and add edges from s to all vertices in L and from each vertex in R to t.

Reduction of bipartite matching to max-flow

Max-Flow Construction

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

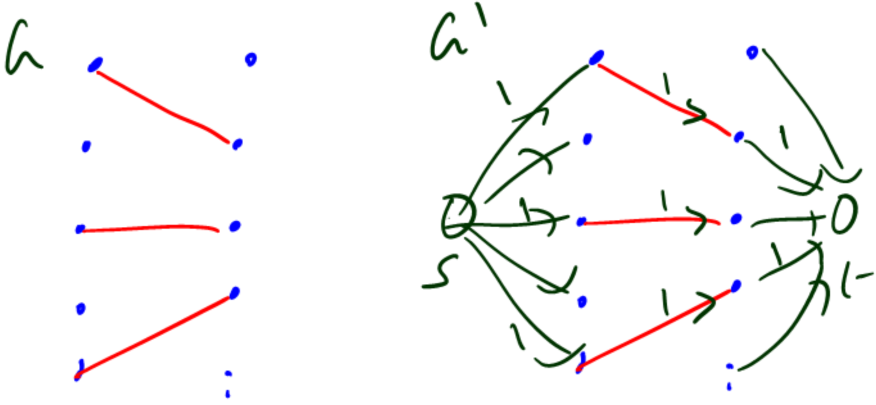


1. $V' = L \cup R \cup \{s, t\}$ where s and t are the new source and sink.
2. Direct all edges in E from L to R, and add edges from s to all vertices in L and from each vertex in R to t.
3. Capacity of every edge is 1.

Correctness: Matching to Flow

Proposition

If G has a matching of size k then G' has a flow of value k.



Correctness: Matching to Flow

Proposition

If G has a matching of size k then G' has a flow of value k.

Proof.

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

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

Verify that f is a flow of value k (because M is a matching). □

Correctness: Flow to Matching

Proposition

If G' has a flow of value k then G has a matching of size k.

Proof.

Consider flow f of value k.

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

$\square$

Correctness of Reduction

Theorem

The maximum flow value in G' = maximum cardinality of matching in G.

Consequence

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

Running Time

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

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

Running Time

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

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

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

Perfect Matchings

Definition

A matching M is said to be **perfect** if every vertex has one edge in M incident upon it.

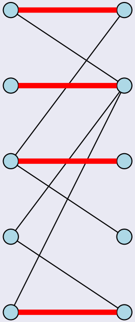


Figure: This graph does not have a perfect matching

Characterizing Perfect Matchings

Problem

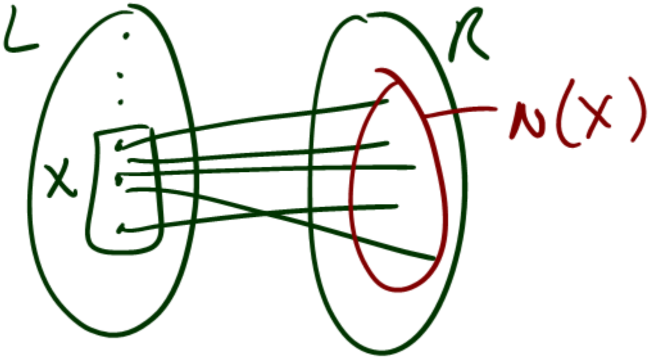
When does a bipartite graph have a perfect matching?

1. Clearly $|L| = |R|$
2. Are there any necessary and sufficient conditions?

A Necessary Condition

Lemma

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



A Necessary Condition

Lemma

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

Proof.

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

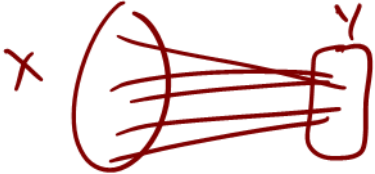
Hall's Theorem

Theorem (Frobenius-Hall)

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

One direction is the necessary condition.

$|X| d$



$|Y| d$

Hall's Theorem

Theorem (Frobenius-Hall)

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

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

1. Create flow network G' from G.
2. If $|N(X)| \geq |X|$ for all X, show that minimum s-t cut in G' is of capacity $n = |L| = |R|$.
3. Implies that G has a perfect matching.

Proof of Sufficiency

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

Proof of Sufficiency

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

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

$\geq |N(X)| - |Y|$

$\geq |L| - |X| + |Y| + \underline{E(X, R-Y)}$

Proof of Sufficiency

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

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

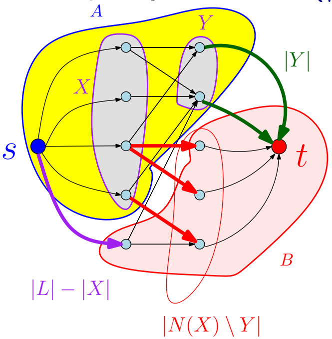
1. Let $X = A \cap L$ and $Y = A \cap R$.

Proof of Sufficiency

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

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

1. Let $X = A \cap L$ and $Y = A \cap R$.
2. Cut capacity is at least $(|L| - |X|) + |Y| + |N(X) \setminus Y|$



Because there are...

1. $|L| - |X|$ edges from s to $L \cap B$.
2. $|Y|$ edges from Y to t.
3. there are at least $|N(X) \setminus Y|$ edges from X to vertices on the right side that are not in Y.

Proof of Sufficiency

Continued...

1. By the above, cut capacity is at least
$$\alpha = (|L| - |X|) + |Y| + |N(X) \setminus Y|.$$
2. $|N(X) \setminus Y| \geq |N(X)| - |Y|$.
(This holds for any two sets.)
3. By assumption $|N(X)| \geq |X|$ and hence
$$|N(X) \setminus Y| \geq |N(X)| - |Y| \geq |X| - |Y|.$$
4. Cut capacity is therefore at least
$$\begin{aligned}\alpha &= (|L| - |X|) + |Y| + |N(X) \setminus Y| \\ &\geq |L| - |X| + |Y| + |X| - |Y| \geq |L| = n.\end{aligned}$$
5. Any s-t cut capacity is at least $n \implies$ max flow at least n units $\implies$ perfect matching. **QED**

Hall's Theorem: Generalization

Theorem (Frobenius-Hall)

Let $G = (L \cup R, E)$ *be a bipartite graph with* $|L| \leq |R|$. G *has a matching that matches all nodes in* L *if and only if for every* $X \subseteq L$, $|N(X)| \geq |X|$.

Proof is essentially the same as the previous one.

Assigning jobs to people

1. n jobs, $n/2$ people
2. For each job: a set of people who can do that job.
3. Each person j has to do exactly two jobs.
4. Goal: find an assignment of 2 jobs to each person, such that all jobs are assigned.

Solution: Build bipartite graph, compute maximum matching, remove it, compute another maximum matching. Both matchings together form a valid solution if it exists. This algorithm is

(A) Correct.

(B) Incorrect.

Application: Assigning jobs to people

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

Application: Assigning jobs to people

1. n jobs or tasks
2. m people
3. for each job a set of people who can do that job
4. for each person j a limit on number of jobs k_j
5. 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

1. Create directed graph $G = (V, E)$ as follows
   1. $V = \{s, t\} \cup L \cup R$: L set of n jobs, R set of m people
   2. add edges (s, i) for each job $i \in L$, capacity 1
   3. add edges (j, t) for each person $j \in R$, capacity k_j
   4. if job i can be done by person j add an edge (i, j), capacity 1
2. Compute max s-t flow. There is an assignment if and only if flow value is 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 was until very recenlty $O(m\sqrt{n})$ due to Hopcroft and Karp. Now there is another algorithm that runs in $\tilde{O}(m^{10/7})$-time due to Madry (2015).

Part I

Baseball Pennant Race

Pennant Race

TUESDAY, SEPTEMBER 10, 1996

San Francisco Chronicle

The Gate
Sports Online
▶ http://www.sfgate.com

SPORTING G

49ers, Young Get Big Brea

Quarterback m

By Gary Swan
Chronicle Staff Writer

The bye week has come at a perfect time for the 49ers and quarterback Steve Young. If they had a game next Sunday, there's a good chance Young would not play.

But the pulled groin muscle on his up-

Giants Officially Leave the NL West Race

By Nancy Gay
Chronicle Staff Writer

With the smack of another National League West bat 500 miles away, the Giants' run at the division title ended last night, just as they were handing the visiting St. Louis Cardinals an even bigger lead in the NL Central.

CARDINALS 6
GIANTS 2

In San Diego, Greg Vaughn's three-run homer in the eighth pushed the Padres over the Pirates and officially shoved the rest of the Giants' season into the background. On the heels of their tedious 6-2 loss before an announced crowd of 10,307 at Candlestick Park, the Giants fell 19½ games off the lead.

As it is, the worst the Padres (80-65) can finish is 80-82. The Giants have fallen to 59-83 with 20 games left; they cannot win 80 games. Coming off a miserable 2-8 mark on a three-city road trip that saw their road record drop to 27-47, the Giants were hoping to get off on the right foot in their longest homestand of the year (15 games, 14 days).

Financing in Place For Giants' New Stadium
SEE PAGE B1, MAIN NEWS

"Where we are, you're going to be eliminated sooner or later," Baker said quietly. "But it doesn't alter the fact that we've still got to play ball. You've still got to play hard, the fans come out to watch you play. You've got to play for the fact of loving to play, no matter where you are in the standings.

"You've got to play the role of spoiler, to not make it easier on

GIANTS: Page D5 Col 3

Pennant Race: Example

Example

Team	Won	Left
New York	92	2
Baltimore	91	3
Toronto	91	3
Boston	89	2

Can Boston win the pennant?

Pennant Race: Example

Example

Team	Won	Left
New York	92	2
Baltimore	91	3
Toronto	91	3
Boston	89	2

Can Boston win the pennant?
No, because Boston can win at most 91 games.

Another Example

Example

Team	Won	Left
New York	92	2
Baltimore	91	3
Toronto	91	3
Boston	90	2

Can Boston win the pennant?

Another Example

Example

Team	Won	Left
New York	92	2
Baltimore	91	3
Toronto	91	3
Boston	90	2

Can Boston win the pennant?
Not clear unless we know what the remaining games are!

Refining the Example

Example

Team	Won	Left	NY	Bal	Tor	Bos
New York	92	2	—	1	1	0
Baltimore	91	3	1	—	1	1
Toronto	91	3	1	1	—	1
Boston	90	2	0	1	1	—

Can Boston win the pennant?

Refining the Example

Example

Team	Won	Left	NY	Bal	Tor	Bos
New York	92	2	—	1	1	0
Baltimore	91	3	1	—	1	1
Toronto	91	3	1	1	—	1
Boston	90	2	0	1	1	—

Can Boston win the pennant? Suppose Boston does

Refining the Example

Example

Team	Won	Left	NY	Bal	Tor	Bos
New York	92	2	—	1	1	0
Baltimore	91	3	1	—	1	1
Toronto	91	3	1	1	—	1
Boston	90	2	0	1	1	—

Can Boston win the pennant? Suppose Boston does

1. Boston wins both its games to get 92 wins

Refining the Example

Example

Team	Won	Left	NY	Bal	Tor	Bos
New York	92	2	—	1	1	0
Baltimore	91	3	1	—	1	1
Toronto	91	3	1	1	—	1
Boston	90	2	0	1	1	—

Can Boston win the pennant? Suppose Boston does

1. Boston wins both its games to get 92 wins
2. New York must lose both games

Refining the Example

Example

Team	Won	Left	NY	Bal	Tor	Bos
New York	92	2	—	1	1	0
Baltimore	91	3	1	—	1	1
Toronto	91	3	1	1	—	1
Boston	90	2	0	1	1	—

Can Boston win the pennant? Suppose Boston does

1. Boston wins both its games to get 92 wins
2. New York must lose both games; now both Baltimore and Toronto have at least 92

Refining the Example

Example

Team	Won	Left	NY	Bal	Tor	Bos
New York	92	2	—	1	1	0
Baltimore	91	3	1	—	1	1
Toronto	91	3	1	1	—	1
Boston	90	2	0	1	1	—

Can Boston win the pennant? Suppose Boston does

1. Boston wins both its games to get 92 wins
2. New York must lose both games; now both Baltimore and Toronto have at least 92
3. Winner of Baltimore-Toronto game has 93 wins!

Can Boston win the penant?

Team	Won	Left	NY	Bal	Tor	Bos
New York	3	6	—	2	3	1
Baltimore	5	4	2	—	1	1
Toronto	4	6	3	1	—	2
Boston	2	4	1	1	2	—

(A) Yes.

(B) No.

Abstracting the Problem

Given

1. A set of teams S
2. For each $x \in S$, the current number of wins w_x
3. For any $x, y \in S$, the number of remaining games g_{xy} between x and y
4. A team z

Can z win the pennant?

Towards a Reduction

$\overline{z}$ can win the pennant if

1. $\overline{z}$ wins at least m games
2. no other team wins more than m games

Towards a Reduction

$\overline{z}$ can win the pennant if

1. $\overline{z}$ wins at least m games
   1. to maximize $\overline{z}$'s chances we make $\overline{z}$ win all its remaining games and hence $m = w_{\overline{z}} + \sum_{x \in S} g_{x\overline{z}}$
2. no other team wins more than m games

Towards a Reduction

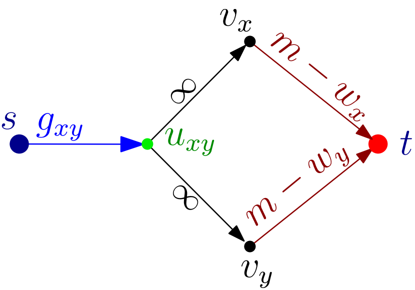
$\overline{z}$ can win the pennant if

1. $\overline{z}$ wins at least m games
   1. to maximize $\overline{z}$'s chances we make $\overline{z}$ win all its remaining games and hence $m = w_{\overline{z}} + \sum_{x \in S} g_{x\overline{z}}$
2. no other team wins more than m games
   1. for each $x, y \in S$ the g_{xy} games between them have to be *assigned* to either x or y.
   2. each team $x \neq \overline{z}$ can win at most $m - w_x - g_{x\overline{z}}$ remaining games

Is there an assignment of remaining games to teams such that no team $x \neq \overline{z}$ wins more than $m - w_x$ games?

Flow Network: The basic gadget

1. s: source
2. t: sink
3. x, y: two teams
4. g_{xy}: number of games remaining between x and y.
5. w_x: number of points x has.
6. m: maximum number of points x can win before team of interest is eliminated.

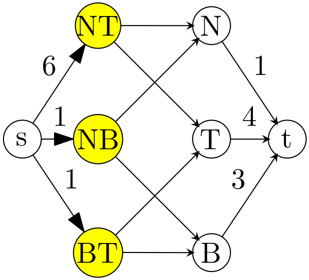


Flow Network: An Example

Can Boston win?

Team	Won	Left	NY	Bal	Tor	Bos
New York	90	11	—	1	6	4
Baltimore	88	6	1	—	1	4
Toronto	87	11	6	1	—	4
Boston	79	12	4	4	4	—

1. $m = 79 + 12 = 91$: Boston can get at most **91** points.



Constructing Flow Network

Notations

1. S: set of teams,
2. w_x wins for each team, and
3. g_{xy} games left between x and y.
4. m be the maximum number of wins for $\bar{z}$,
5. and $S' = S \setminus \{\bar{z}\}$.

Reduction

Construct the flow network G as follows

1. One vertex v_x for each team $x \in S'$, one vertex u_{xy} for each pair of teams x and y in S'
2. A new source vertex s and sink t
3. Edges (u_{xy}, v_x) and (u_{xy}, v_y) of capacity ∞
4. Edges (s, u_{xy}) of capacity g_{xy}
5. Edges (v_x, t) of capacity equal $m - w_x$

Correctness of reduction

Theorem

G' has a maximum flow of value $g^* = \sum_{x,y \in S'} g_{xy}$ if and only if $\overline{z}$ can win the most number of games (including possibly tie with other teams).

Proof of Correctness

Proof.

Existence of g^* flow $\Rightarrow \overline{z}$ wins pennant

1. An integral flow saturating edges out of s, ensures that each remaining game between x and y is added to win total of either x or y
2. Capacity on (v_x, t) edges ensures that no team wins more than m games

Conversely, $\overline{z}$ wins pennant $\Rightarrow$ flow of value g^*

1. Scenario determines flow on edges; if x wins k of the games against y, then flow on (u_{xy}, v_x) edge is k and on (u_{xy}, v_y) edge is $g_{xy} - k$ □

Proof that $\overline{z}$ cannot with the pennant

1. Suppose $\overline{z}$ cannot win the pennant since $g^* < g$. How do we *prove* to some one *compactly* that $\overline{z}$ cannot win the pennant?

Proof that $\overline{z}$ cannot with the pennant

1. Suppose $\overline{z}$ cannot win the pennant since $g^* < g$. How do we *prove* to some one *compactly* that $\overline{z}$ cannot win the pennant?
2. Show them the min-cut in the reduction flow network!

Proof that $\overline{z}$ cannot with the pennant

1. Suppose $\overline{z}$ cannot win the pennant since $g^* < g$. How do we *prove* to some one *compactly* that $\overline{z}$ cannot win the pennant?
2. Show them the min-cut in the reduction flow network!
3. See Kleinberg-Tardos book for a natural interpretation of the min-cut as a certificate.

The biggest loser?

Given an input as above for the pennant competition, deciding if a team can come in the last place can be done in

(A) Can be done using the same reduction as just seen.

(B) Can not be done using the same reduction as just seen.

(C) Can be done using flows but we need lower bounds on the flow, instead of upper bounds.

(D) The problem is **NP-Hard** and requires exponential time.

(E) Can be solved by negating all the numbers, and using the above reduction.

(F) Can be solved efficiently only by running a reality show on the problem.

Part II

An Application of Min-Cut to Project Scheduling

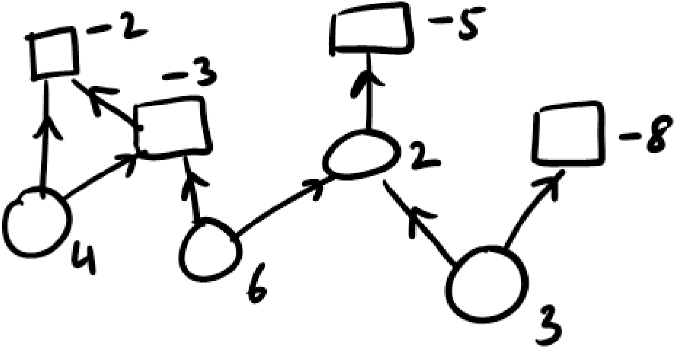
Project Scheduling

Problem:

1. n projects/tasks $1, 2, \ldots, n$
2. *dependencies* between projects: i depends on j implies i cannot be done unless j is done. dependency graph is *acyclic*
3. each project i has a cost/profit p_i
    1. $p_i < 0$ implies i requires a cost of $-p_i$ units
    2. $p_i > 0$ implies that i generates p_i profit

Goal: Find projects to do so as to *maximize* profit.

Example



Notation

For a set $\boldsymbol{A}$ of projects:

1. $\boldsymbol{A}$ is a *valid* solution if $\boldsymbol{A}$ is *dependency closed*, that is for every $\boldsymbol{i} \in \boldsymbol{A}$, all projects that $\boldsymbol{i}$ depends on are also in $\boldsymbol{A}$.

Notation

For a set A of projects:

1. A is a *valid* solution if A is *dependency closed*, that is for every $i \in A$, all projects that i depends on are also in A.
2. $profit(A) = \sum_{i \in A} p_i$. Can be negative or positive.

Notation

For a set A of projects:

1. A is a *valid* solution if A is *dependency closed*, that is for every $i \in A$, all projects that i depends on are also in A.
2. $profit(A) = \sum_{i \in A} p_i$. Can be negative or positive.

Goal: find valid A to maximize $profit(A)$.

Idea: Reduction to Minimum-Cut

Finding a set of projects is partitioning the projects into two sets: those that are done and those that are not done.

Can we express this is a minimum cut problem?

Idea: Reduction to Minimum-Cut

Finding a set of projects is partitioning the projects into two sets: those that are done and those that are not done.

Can we express this is a minimum cut problem?

Several issues:

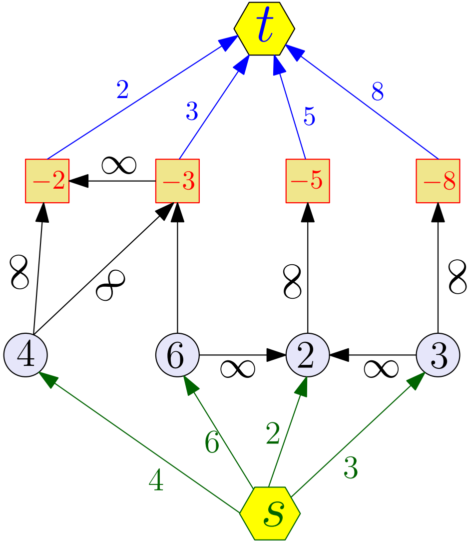
1. We are interested in maximizing profit but we can solve minimum cuts.
2. We need to convert negative profits into positive capacities.
3. Need to ensure that chosen projects is a valid set.
4. The cut value captures the profit of the chosen set of projects.

Reduction to Minimum-Cut

Note: We are reducing a *maximization* problem to a *minimization* problem.

1. projects represented as nodes in a graph
2. if i depends on j then (i, j) is an edge
3. add source s and sink t
4. for each i with $p_i > 0$ add edge (s, i) with capacity p_i
5. for each i with $p_i < 0$ add edge (i, t) with capacity $-p_i$
6. for each dependency edge (i, j) put capacity ∞ (more on this later)

Reduction: Flow Network Example



Reduction contd

Algorithm:

1. form graph as in previous slide
2. compute s-t minimum cut (A, B)
3. output the projects in $A - \{s\}$

Understanding the Reduction

Let $C = \sum_{i:p_i>0} p_i$: maximum possible profit.

Understanding the Reduction

Let $C = \sum_{i:p_i>0} p_i$: maximum possible profit.

Observation: The minimum s-t cut value is $\leq C$. Why?

Understanding the Reduction

Let $C = \sum_{i:p_i>0} p_i$: maximum possible profit.

Observation: The minimum s-t cut value is $\leq C$. Why?

Lemma

Suppose (A, B) is an s-t cut of finite capacity (no ∞) edges. Then projects in $A - \{s\}$ are a valid *solution.*

Understanding the Reduction

Let $C = \sum_{i:p_i>0} p_i$: maximum possible profit.

Observation: The minimum s-t cut value is $\leq C$. Why?

Lemma

Suppose (A, B) is an s-t cut of finite capacity (no ∞) edges. Then projects in $A - \{s\}$ are a valid *solution.*

Proof.

If $A - \{s\}$ is not a valid solution then there is a project $i \in A$ and a project $j \notin A$ such that i depends on j

Understanding the Reduction

Let $C = \sum_{i:p_i>0} p_i$: maximum possible profit.

Observation: The minimum s-t cut value is $\leq C$. Why?

Lemma

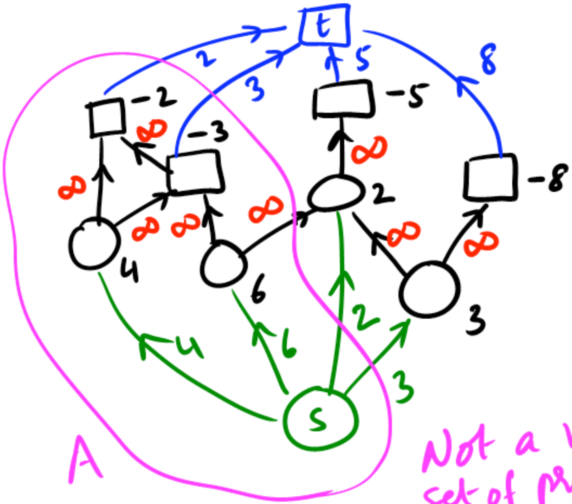
Suppose (A, B) is an s-t cut of finite capacity (no ∞) edges. Then projects in $A - \{s\}$ are a valid *solution.*

Proof.

If $A - \{s\}$ is not a valid solution then there is a project $i \in A$ and a project $j \notin A$ such that i depends on j

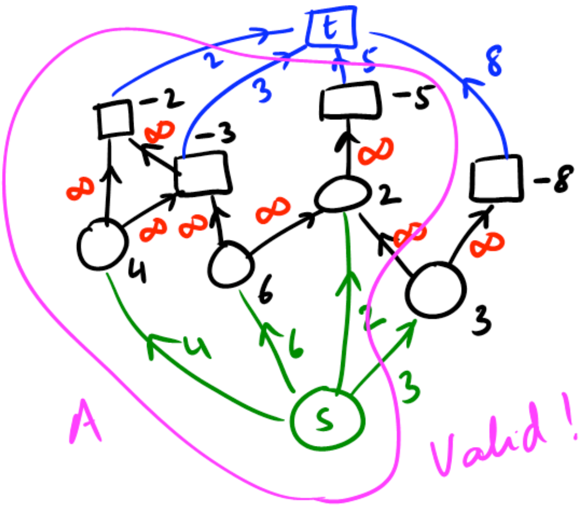
Since (i, j) capacity is ∞, implies (A, B) capacity is ∞, contradicting assumption. □

Example



Not a valid set of projects

Example



Correctness of Reduction

Recall that for a set of projects X, $profit(X) = \sum_{i \in X} p_i$.

Correctness of Reduction

Recall that for a set of projects X, $profit(X) = \sum_{i \in X} p_i$.

Lemma

Suppose (A, B) is an s-t cut of finite capacity (no ∞) edges. Then $c(A, B) = C - profit(A - \{s\})$.

Correctness of Reduction

Recall that for a set of projects X, $profit(X) = \sum_{i \in X} p_i$.

Lemma

Suppose (A, B) is an s-t cut of finite capacity (no ∞) edges. Then $c(A, B) = C - profit(A - \{s\})$.

Proof.

Edges in (A, B):

1. (s, i) for $i \in B$ and $p_i > 0$: capacity is p_i
2. (i, t) for $i \in A$ and $p_i < 0$: capacity is $-p_i$
3. cannot have ∞ edges

$\square$

Proof contd

For project set A let

1. $cost(A) = \sum_{i \in A: p_i < 0} -p_i$
2. $benefit(A) = \sum_{i \in A: p_i > 0} p_i$
3. $profit(A) = benefit(A) - cost(A)$.

Proof.

Let $A' = A \cup \{s\}$.

$$
\begin{aligned}
c(A', B) &= cost(A) + benefit(B) \\
&= cost(A) - benefit(A) + benefit(A) + benefit(B) \\
&= -profit(A) + C \\
&= C - profit(A)
\end{aligned}
$$

□

Correctness of Reduction contd

We have shown that if (A, B) is an s-t cut in G with finite capacity then

1. $A - \{s\}$ is a valid set of projects
2. $c(A, B) = C - \mathit{profit}(A - \{s\})$

Correctness of Reduction contd

We have shown that if (A, B) is an s-t cut in G with finite capacity then

1. $A - \{s\}$ is a valid set of projects
2. $c(A, B) = C - \mathit{profit}(A - \{s\})$

Therefore a *minimum* s-t cut (A^*, B^*) gives a *maximum* profit set of projects $A^* - \{s\}$ since C is fixed.

Correctness of Reduction contd

We have shown that if (A, B) is an s-t cut in G with finite capacity then

1. $A - \{s\}$ is a valid set of projects
2. $c(A, B) = C - profit(A - \{s\})$

Therefore a *minimum* s-t cut (A^*, B^*) gives a *maximum* profit set of projects $A^* - \{s\}$ since C is fixed.

Question: How can we use ∞ in a real algorithm?

Correctness of Reduction contd

We have shown that if (A, B) is an s-t cut in G with finite capacity then

1. $A - \{s\}$ is a valid set of projects
2. $c(A, B) = C - profit(A - \{s\})$

Therefore a *minimum* s-t cut (A^*, B^*) gives a *maximum* profit set of projects $A^* - \{s\}$ since C is fixed.

Question: How can we use ∞ in a real algorithm?

Set capacity of ∞ arcs to $C + 1$ instead. Why does this work?