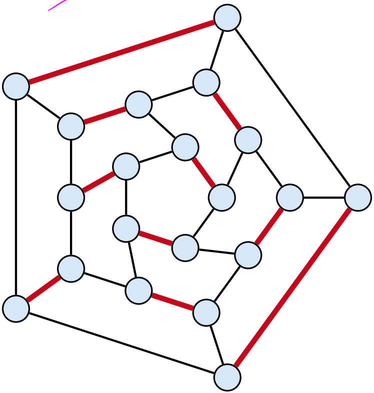
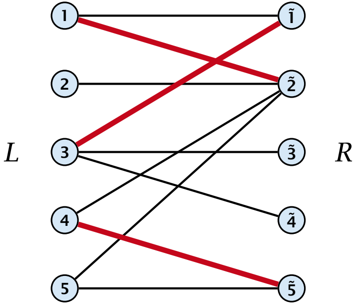
Matching

- Input: undirected graph $G = (V, E)$.
- $M \subseteq E$ is a matching if each node appears in at most one edge in M.
- Maximum Matching: find a matching of maximum cardinality



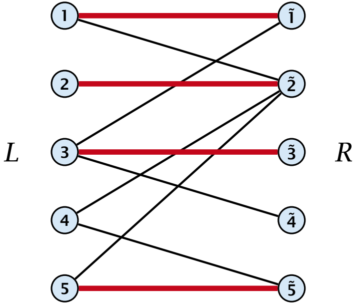
Bipartite Matching

- Input: undirected, bipartite graph $G = (L \uplus R, E)$.
- $M \subseteq E$ is a matching if each node appears in at most one edge in M.
- Maximum Matching: find a matching of maximum cardinality



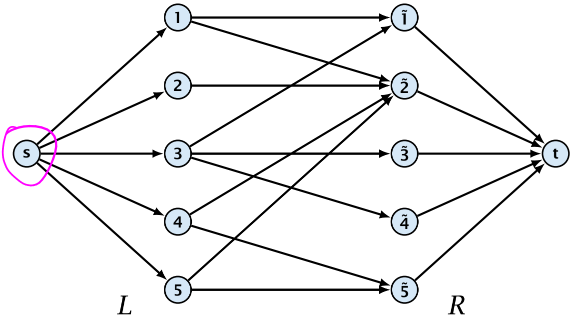
Bipartite Matching

- Input: undirected, bipartite graph $G = (L \uplus R, E)$.
- $M \subseteq E$ is a matching if each node appears in at most one edge in M.
- Maximum Matching: find a matching of maximum cardinality



Maxflow Formulation

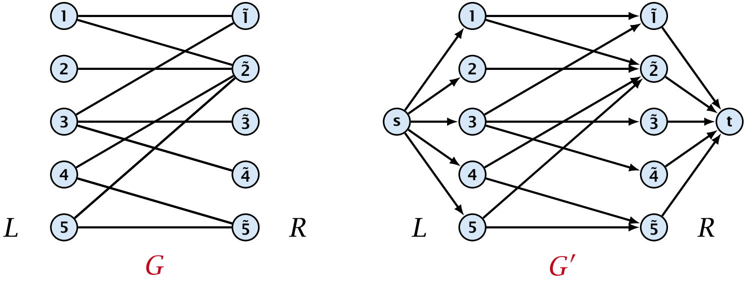
- Input: undirected, bipartite graph $G = (L \uplus R \uplus \{s,t\}, E')$.
- Direct all edges from L to R.
- Add source s and connect it to all nodes on the left.
- Add t and connect all nodes on the right to t.
- All edges have unit capacity.



Proof

Max cardinality matching in $G \leq$ value of maxflow in G'

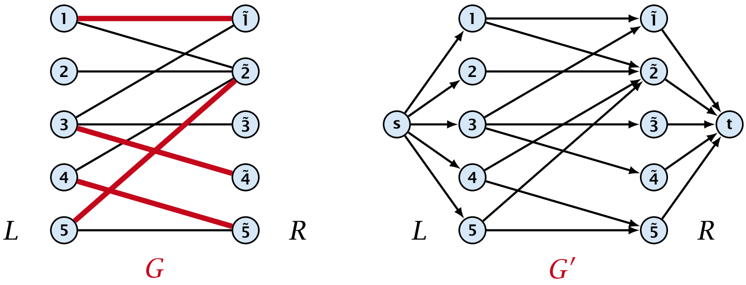
- Given a maximum matching M of cardinality k.
- Consider flow f that sends one unit along each of k paths.
- f is a flow and has cardinality k.



Proof

Max cardinality matching in $G \leq$ value of maxflow in G'

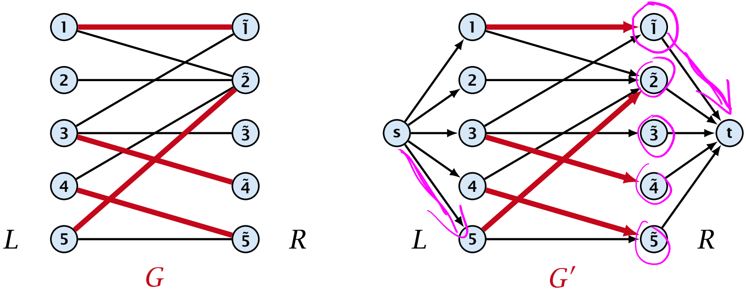
- Given a maximum matching M of cardinality k.
- Consider flow f that sends one unit along each of k paths.
- f is a flow and has cardinality k.



Proof

Max cardinality matching in $G \leq$ value of maxflow in G'

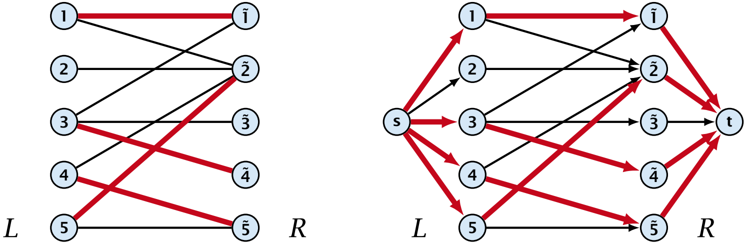
- Given a maximum matching M of cardinality k.
- Consider flow f that sends one unit along each of k paths.
- f is a flow and has cardinality k.



Proof

Max cardinality matching in $G \leq$ value of maxflow in G'

- Given a maximum matching M of cardinality k.
- Consider flow f that sends one unit along each of k paths.
- f is a flow and has cardinality k.

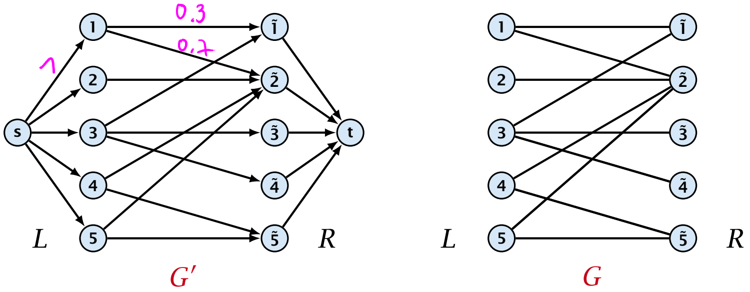


G $\quad$ G'

Proof

Max cardinality matching in $G \geq$ value of maxflow in G'

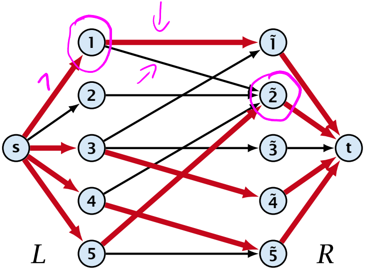
- Let f be a maxflow in G' of value k
- Integrality theorem $\Rightarrow$ k integral; we can assume f is 0/1.
- Consider M= set of edges from L to R with $f(e) = 1$.
- Each node in L and R participates in at most one edge in M.
- $|M| = k$, as the flow must use at least k middle edges.



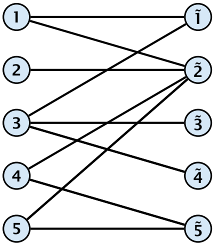
Proof

Max cardinality matching in $G \geq$ value of maxflow in G'

- Let f be a maxflow in G' of value k
- Integrality theorem $\Rightarrow k$ integral; we can assume f is 0/1.
- Consider M= set of edges from L to R with $f(e) = 1$.
- Each node in L and R participates in at most one edge in M.
- $|M| = k$, as the flow must use at least k middle edges.



G'

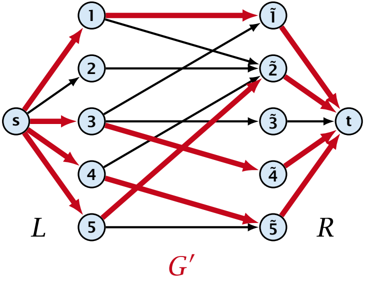


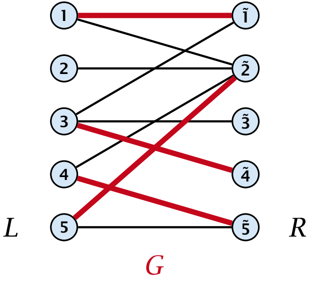
G

Proof

Max cardinality matching in $G \geq$ value of maxflow in G'

- Let f be a maxflow in G' of value k
- Integrality theorem $\Rightarrow$ k integral; we can assume f is 0/1.
- Consider M= set of edges from L to R with $f(e) = 1$.
- Each node in L and R participates in at most one edge in M.
- $|M| = k$, as the flow must use at least k middle edges.





12.1 Matching

Which flow algorithm to use?

- Generic augmenting path: $\mathcal{O}(m\,\mathrm{val}(f^*)) = \mathcal{O}(mn)$.
- Capacity scaling: $\mathcal{O}(m^2 \log(C+1)) = \mathcal{O}(m^2)$.
- Shortest augmenting path: $\mathcal{O}(mn^2)$.

For unit capacity simple graphs shortest augmenting path can be implemented in time $\mathcal{O}(m\sqrt{n})$.

Baseball Elimination

team i	wins w_i	losses ℓ_i	remaining games Atl	Phi	NY	Mon
Atlanta	$\boxed{83}$	71	–	1	6	1
Philadelphia	80	79	1	–	0	2
New York	78	78	6	0	–	0
Montreal	77	82	1	2	0	–

Which team can end the season with most wins?

- Montreal is eliminated, since even after winning all remaining games there are only 80 wins.
- But also Philadelphia is eliminated. Why?

Baseball Elimination

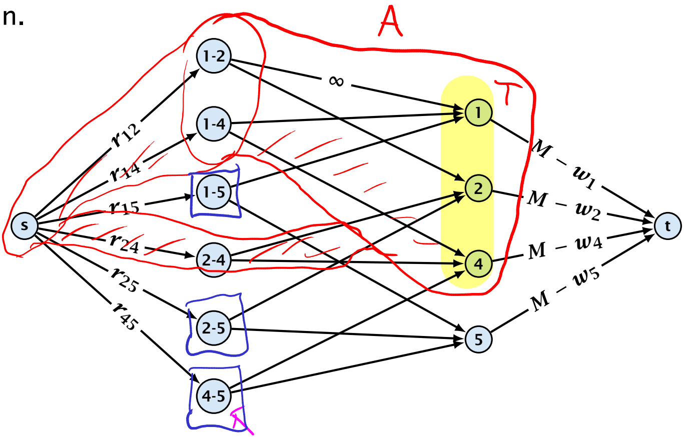
Formal definition of the problem:

- Given a set S of teams, and one specific team $z \in S$.
- Team x has already won w_x games.
- Team x still has to play team y, r_{xy} times.
- Does team z still have a chance to finish with the most number of wins.

Baseball Elimination

1, ..., 5

Flow network for $\boxed{z = 3.}$ M is number of wins Team 3 can still obtain.



Idea. Distribute the results of remaining games in such a way that no team gets too many wins.

Certificate of Elimination

Let $T \subseteq S$ be a subset of teams. Define

$$\underbrace{w(T)}_{\text{wins of teams in } T} := \sum_{i \in T} w_i, \qquad \underbrace{r(T)}_{\text{remaining games among teams in } T} := \sum_{i,j \in T, i<j} r_{ij}$$

If $\frac{w(T)+r(T)}{|T|} > M$ then one of the teams in T will have more than M wins in the end. A team that can win at most M games is therefore eliminated.

Theorem 63

A team z is eliminated if and only if the flow network for z does not allow a flow of value $\sum_{ij\in S\setminus\{z\},i<j} r_{ij}$.

Theorem 63

A team z is eliminated if and only if the flow network for z does not allow a flow of value $\sum_{ij\in S\setminus\{z\},i<j} r_{ij}$.

Proof (⇐)

- Consider the mincut A in the flow network. Let T be the set of team-nodes in A.

Theorem 63

A team z is eliminated if and only if the flow network for z does not allow a flow of value $\sum_{ij\in S\setminus\{z\},i<j} r_{ij}$.

Proof ($\Leftarrow$)

- Consider the mincut A in the flow network. Let T be the set of team-nodes in A.
- If for node x-y not both team-nodes x and y are in T, then x-$y \notin A$ as otw. the cut would cut an infinite capacity edge.

Theorem 63

A team z is eliminated if and only if the flow network for z does not allow a flow of value $\sum_{ij\in S\setminus\{z\},i<j} r_{ij}$.

Proof (⇐)

- Consider the mincut A in the flow network. Let T be the set of team-nodes in A.
- If for node x-y not both team-nodes x and y are in T, then x-$y \notin A$ as otw. the cut would cut an infinite capacity edge.
- We don't find a flow that saturates all source edges:

$$r(S \setminus \{z\})$$

Theorem 63

A team z is eliminated if and only if the flow network for z does not allow a flow of value $\sum_{ij\in S\setminus\{z\},i<j} r_{ij}$.

Proof (⇐)

- Consider the mincut A in the flow network. Let T be the set of team-nodes in A.
- If for node x-y not both team-nodes x and y are in T, then x-$y \notin A$ as otw. the cut would cut an infinite capacity edge.
- We don't find a flow that saturates all source edges:

$$r(S \setminus \{z\}) > \mathrm{cap}(A, V \setminus A)$$

Theorem 63

A team z is eliminated if and only if the flow network for z does not allow a flow of value $\sum_{ij\in S\setminus\{z\},i<j} r_{ij}$.

Proof (⇐)

- Consider the mincut A in the flow network. Let T be the set of team-nodes in A.
- If for node x-y not both team-nodes x and y are in T, then x-$y \notin A$ as otw. the cut would cut an infinite capacity edge.
- We don't find a flow that saturates all source edges:

$$\begin{aligned} r(S\setminus\{z\}) &> \mathrm{cap}(A, V\setminus A) \\ &\geq \sum_{i<j:\, i\notin T\vee j\notin T} r_{ij} + \sum_{i\in T}(M - w_i) \end{aligned}$$

Theorem 63

A team z is eliminated if and only if the flow network for z does not allow a flow of value $\sum_{ij\in S\setminus\{z\},i<j} r_{ij}$.

Proof (⇐)

- Consider the mincut A in the flow network. Let T be the set of team-nodes in A.
- If for node x-y not both team-nodes x and y are in T, then x-$y \notin A$ as otw. the cut would cut an infinite capacity edge.
- We don't find a flow that saturates all source edges:

$$
\begin{aligned}
r(S\setminus\{z\}) &> \mathrm{cap}(A, V\setminus A) \\
&\geq \sum_{i<j:\, i\notin T \vee j\notin T} r_{ij} + \sum_{i\in T} (M - w_i) \\
&\geq r(S\setminus\{z\}) - r(T) + |T|M - w(T)
\end{aligned}
$$

Theorem 63

A team z is eliminated if and only if the flow network for z does not allow a flow of value $\sum_{ij \in S \setminus \{z\}, i<j} r_{ij}$.

Proof ($\Leftarrow$)

- Consider the mincut A in the flow network. Let T be the set of team-nodes in A.
- If for node x-y not both team-nodes x and y are in T, then x-$y \notin A$ as otw. the cut would cut an infinite capacity edge.
- We don't find a flow that saturates all source edges:

$$\begin{aligned}
\cancel{r(S \setminus \{z\})} &> \mathrm{cap}(A, V \setminus A) \\
&\geq \sum_{i<j:\, i \notin T \vee j \notin T} r_{ij} + \sum_{i \in T} (M - w_i) \\
&\geq \cancel{r(S \setminus \{z\})} - \underline{r(T)} + |T|M - \underline{w(T)}
\end{aligned}$$

- This gives $M < (w(T) + r(T))/|T|$, i.e., z is eliminated.

Baseball Elimination

Proof (⇒)

- Suppose we have a flow that saturates all source edges.
- We can assume that this flow is integral.
- For every pairing x-y it defines how many games team x and team y should win.
- The flow leaving the team-node x can be interpreted as the additional number of wins that team x will obtain.
- This is less than $M - w_x$ because of capacity constraints.
- Hence, we found a set of results for the remaining games, such that no team obtains more than M wins in total.
- Hence, team z is not eliminated.

Baseball Elimination

Proof (⇒)

- Suppose we have a flow that saturates all source edges.
- We can assume that this flow is integral.
- For every pairing x-y it defines how many games team x and team y should win.
- The flow leaving the team-node x can be interpreted as the additional number of wins that team x will obtain.
- This is less than $M - w_x$ because of capacity constraints.
- Hence, we found a set of results for the remaining games, such that no team obtains more than M wins in total.
- Hence, team z is not eliminated.

Baseball Elimination

Proof (⇒)

- Suppose we have a flow that saturates all source edges.
- We can assume that this flow is integral.
- For every pairing x-y it defines how many games team x and team y should win.
- The flow leaving the team-node x can be interpreted as the additional number of wins that team x will obtain.
- This is less than $M - w_x$ because of capacity constraints.
- Hence, we found a set of results for the remaining games, such that no team obtains more than M wins in total.
- Hence, team z is not eliminated.

Baseball Elimination

Proof (⇒)

- Suppose we have a flow that saturates all source edges.
- We can assume that this flow is integral.
- For every pairing x-y it defines how many games team x and team y should win.
- The flow leaving the team-node x can be interpreted as the additional number of wins that team x will obtain.
- This is less than $M - w_x$ because of capacity constraints.
- Hence, we found a set of results for the remaining games, such that no team obtains more than M wins in total.
- Hence, team z is not eliminated.

Baseball Elimination

Proof (⇒)

- Suppose we have a flow that saturates all source edges.
- We can assume that this flow is integral.
- For every pairing x-y it defines how many games team x and team y should win.
- The flow leaving the team-node x can be interpreted as the additional number of wins that team x will obtain.
- This is less than $M - w_x$ because of capacity constraints.
- Hence, we found a set of results for the remaining games, such that no team obtains more than M wins in total.
- Hence, team z is not eliminated.

Baseball Elimination

Proof (⇒)

- Suppose we have a flow that saturates all source edges.
- We can assume that this flow is integral.
- For every pairing x-y it defines how many games team x and team y should win.
- The flow leaving the team-node x can be interpreted as the additional number of wins that team x will obtain.
- This is less than $M - w_x$ because of capacity constraints.
- Hence, we found a set of results for the remaining games, such that no team obtains more than M wins in total.
- Hence, team z is not eliminated.

Baseball Elimination

Proof (⇒)

- Suppose we have a flow that saturates all source edges.
- We can assume that this flow is integral.
- For every pairing x-y it defines how many games team x and team y should win.
- The flow leaving the team-node x can be interpreted as the additional number of wins that team x will obtain.
- This is less than $M - w_x$ because of capacity constraints.
- Hence, we found a set of results for the remaining games, such that no team obtains more than M wins in total.
- Hence, team z is not eliminated.

Project Selection

Project selection problem:

- Set P of possible projects. Project v has an associated profit p_v (can be positive or negative).
- Some projects have requirements (taking course EA2 requires course EA1).
- Dependencies are modelled in a graph. Edge (u, v) means "can't do project u without also doing project v."
- A subset A of projects is feasible if the prerequisites of every project in A also belong to A.

Goal: Find a feasible set of projects that maximizes the profit.

Project Selection

Project selection problem:

- Set P of possible projects. Project v has an associated profit p_v (can be positive or negative).
- Some projects have requirements (taking course EA2 requires course EA1).
- Dependencies are modelled in a graph. Edge (u,v) means "can't do project u without also doing project v."
- A subset A of projects is feasible if the prerequisites of every project in A also belong to A.

Goal: Find a feasible set of projects that maximizes the profit.

Project Selection

Project selection problem:

- Set P of possible projects. Project v has an associated profit p_v (can be positive or negative).
- Some projects have requirements (taking course EA2 requires course EA1).
- Dependencies are modelled in a graph. Edge (u,v) means "can't do project u without also doing project v."
- A subset A of projects is feasible if the prerequisites of every project in A also belong to A.

Goal: Find a feasible set of projects that maximizes the profit.

Project Selection

Project selection problem:

- Set P of possible projects. Project v has an associated profit p_v (can be positive or negative).
- Some projects have requirements (taking course EA2 requires course EA1).
- Dependencies are modelled in a graph. Edge (u, v) means “can’t do project u without also doing project v.”
- A subset A of projects is feasible if the prerequisites of every project in A also belong to A.

Goal: Find a feasible set of projects that maximizes the profit.

Project Selection

Project selection problem:

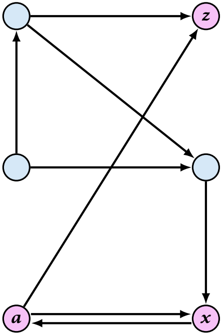
- Set P of possible projects. Project v has an associated profit p_v (can be positive or negative).
- Some projects have requirements (taking course EA2 requires course EA1).
- Dependencies are modelled in a graph. Edge (u,v) means "can't do project u without also doing project v."
- A subset A of projects is feasible if the prerequisites of every project in A also belong to A.

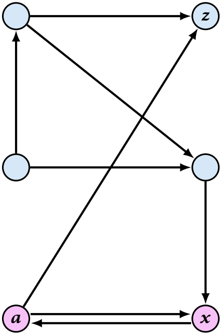
Goal: Find a feasible set of projects that maximizes the profit.

Project Selection

The prerequisite graph:

- $\{x, a, z\}$ is a feasible subset.
- $\{x, a\}$ is infeasible.

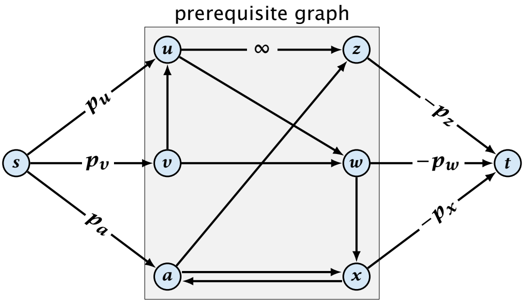




Project Selection

Mincut formulation:

- Edges in the prerequisite graph get infinite capacity.
- Add edge (s, v) with capacity p_v for nodes v with positive profit.
- Create edge (v, t) with capacity $-p_v$ for nodes v with negative profit.



Theorem 64

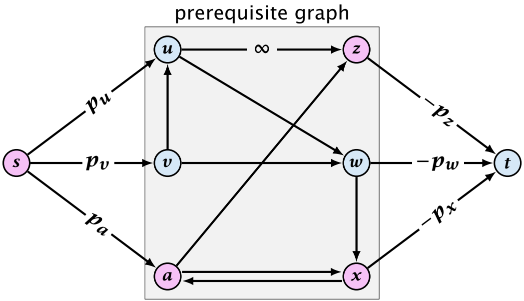
A is a mincut if $A \setminus \{s\}$ is the optimal set of projects.

Theorem 64

A is a mincut if $A \setminus \{s\}$ is the optimal set of projects.

Proof.

- A is feasible because of capacity infinity edges.

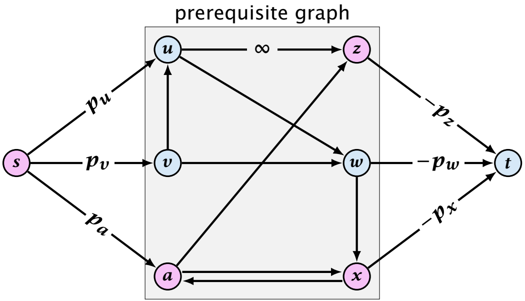


Theorem 64

A is a mincut if $A \setminus \{s\}$ is the optimal set of projects.

Proof.

- A is feasible because of capacity infinity edges.
- $\mathrm{cap}(A, V \setminus A)$

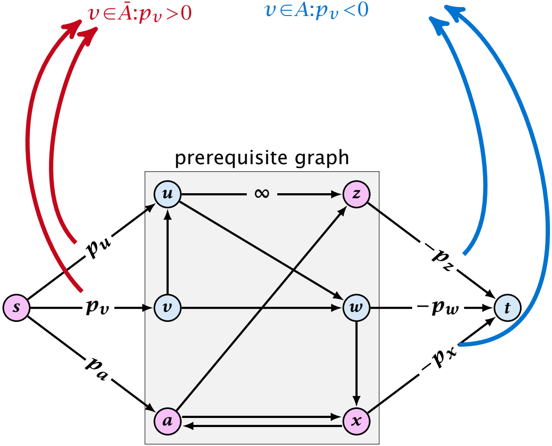


Theorem 64

A is a mincut if $A \setminus \{s\}$ is the optimal set of projects.

Proof.

- A is feasible because of capacity infinity edges.
- $\operatorname{cap}(A, V \setminus A) = \displaystyle\sum_{v \in \bar{A}: p_v > 0} p_v + \sum_{v \in A: p_v < 0} (-p_v)$



Theorem 64

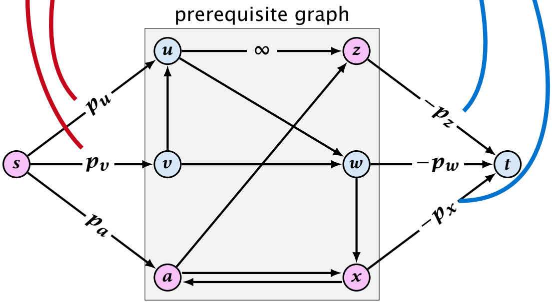
A is a mincut if $A \setminus \{s\}$ is the optimal set of projects.

Proof.

- A is feasible because of capacity infinity edges.
- $\text{cap}(A, V \setminus A) = \boxed{\sum_{v \in \bar{A}: p_v > 0} p_v} + \sum_{v \in A: p_v < 0} (-p_v)$

$= \boxed{\sum_{v: p_v > 0} p_v} - \sum_{v \in A} p_v$

$\sum_{v \in A: p_v > 0} p_v$



Preflows

Definition 65

An (s,t)-preflow is a function $f : E \mapsto \mathbb{R}^+$ that satisfies

1. For each edge e
$$0 \le f(e) \le c(e) \ .$$
2. For each $v \in V \setminus \{s,t\}$
$$\sum_{e \in \mathrm{out}(v)} f(e) \le \sum_{e \in \mathrm{into}(v)} f(e) \ .$$

Preflows

Definition 65

An (s,t)-preflow is a function $f : E \mapsto \mathbb{R}^+$ that satisfies

1. For each edge e
$$0 \le f(e) \le c(e)\ .$$
(capacity constraints)

2. For each $v \in V \setminus \{s,t\}$
$$\sum_{e\in\text{out}(v)} f(e) \le \sum_{e\in\text{into}(v)} f(e)\ .$$

Preflows

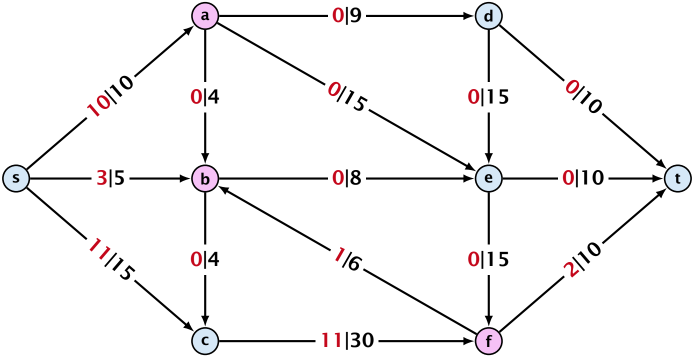
Definition 65

An (s,t)-preflow is a function $f : E \mapsto \mathbb{R}^+$ that satisfies

1. For each edge e
$$0 \leq f(e) \leq c(e) \ .$$
(capacity constraints)
2. For each $v \in V \setminus \{s,t\}$
$$\sum_{e\in \mathrm{out}(v)} f(e) \leq \sum_{e\in \mathrm{into}(v)} f(e) \ .$$

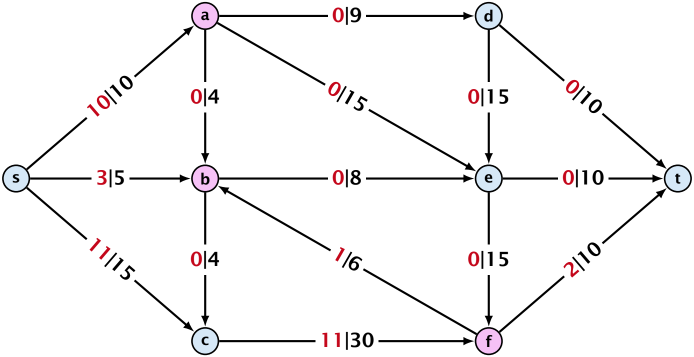
Preflows

Example 66



Preflows

Example 66



A node that has $\sum_{e\in\text{out}(v)} f(e) < \sum_{e\in\text{into}(v)} f(e)$ is called an active node.

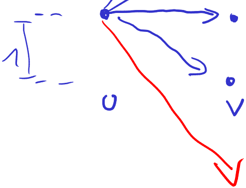
Preflows

Preflows

Definition:

A labelling is a function $\ell : V \to \mathbb{N}$. It is valid for preflow f if

- $\ell(u) \leq \ell(v) + 1$ for all edges (u, v) in the residual graph G_f (only non-zero capacity edges!!!)



Preflows

Definition:

A labelling is a function $\ell : V \to \mathbb{N}$. It is valid for preflow f if

- $\ell(u) \leq \ell(v) + 1$ for all edges (u, v) in the residual graph G_f (only non-zero capacity edges!!!)
- $\ell(s) = n$

Preflows

Definition:

A labelling is a function $\ell : V \to \mathbb{N}$. It is valid for preflow f if

- $\ell(u) \leq \ell(v) + 1$ for all edges (u, v) in the residual graph G_f (only non-zero capacity edges!!!)
- $\ell(s) = n$
- $\ell(t) = 0$

Preflows

Definition:

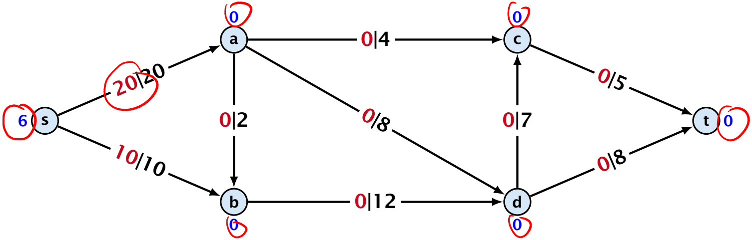
A labelling is a function $\ell : V \to \mathbb{N}$. It is valid for preflow f if

- $\ell(u) \leq \ell(v) + 1$ for all edges (u, v) in the residual graph G_f (only non-zero capacity edges!!!)
- $\ell(s) = n$
- $\ell(t) = 0$

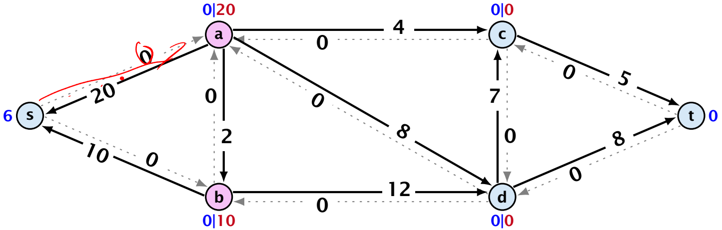
Intuition:

The labelling can be viewed as a height function. Whenever the height from node u to node v decreases by more than 1 (i.e., it goes very steep downhill from u to v), the corresponding edge must be saturated.

Preflows

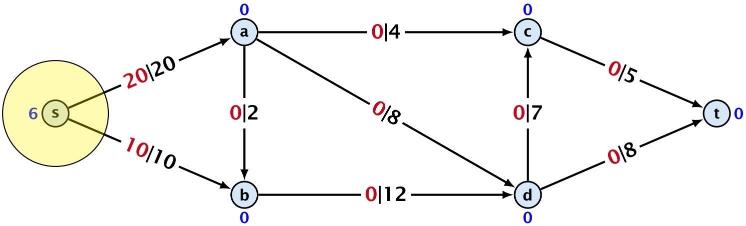


a
b
c
d
s
t
0
0
0
0
6
0
20|20
10|10
0|4
0|2
0|8
0|7
0|5
0|8
0|12

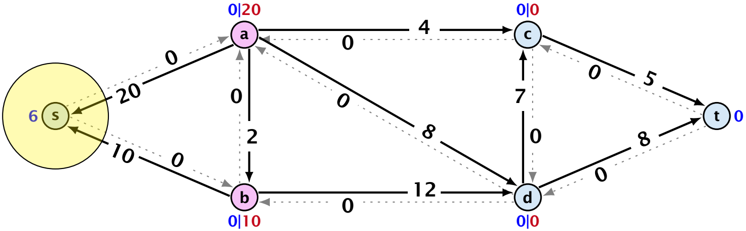


0|20
0|0
0|10
0|0
6
0
a
b
c
d
s
t
20
0
10
0
4
0
0
2
0
8
7
0
0
5
8
0
12
0

Preflows



0
a
0|4
0
c
20|20
0/5
6
s
0|2
0/8
0|7
t
0
10/10
0/8
b
0|12
d
0
0



0|20
a
4
0
0|0
c
0
20
0
0
5
6
s
7
t
0
0
8
10
0
2
8
0
b
12
0
d
0|10
0|0

Preflows

Preflows

Lemma 67

A preflow that has a valid labelling saturates a cut.

Preflows

Lemma 67

A preflow that has a valid labelling saturates a cut.

Proof:

- There are n nodes but $n+1$ different labels from $0,\ldots,n$.

Preflows

Lemma 67

A preflow that has a valid labelling saturates a cut.

Proof:

- There are n nodes but $n + 1$ different labels from $0, \ldots, n$.
- There must exist a label $d \in \{0, \ldots, n\}$ such that none of the nodes carries this label.

Preflows

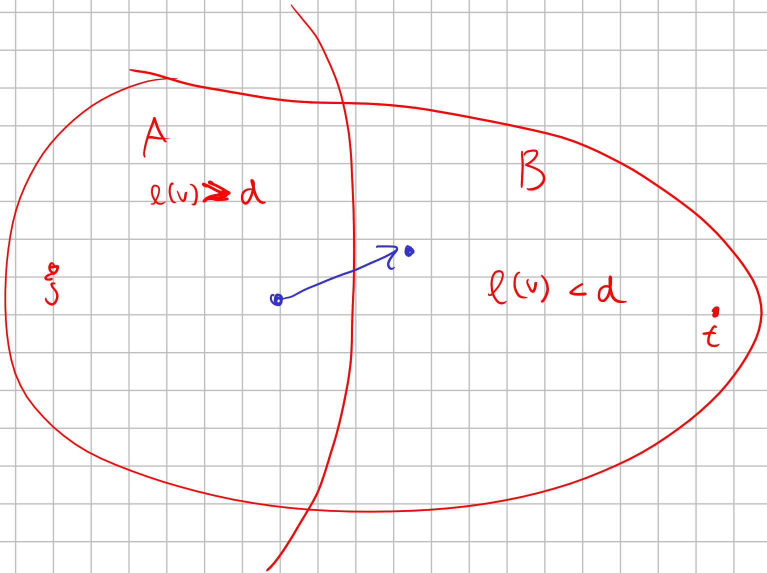
Lemma 67

A preflow that has a valid labelling saturates a cut.

Proof:

- There are n nodes but $n+1$ different labels from $0,\ldots,n$.
- There must exist a label $d \in \{0,\ldots,n\}$ such that none of the nodes carries this label.
- Let $A = \{v \in V \mid \ell(v) > d\}$ and $B = \{v \in V \mid \ell(v) < d\}$.

$s \in A$ $t \in B$


A
ℓ(v) ≥ d
B
ℓ(v) < d
s
t

Preflows

Lemma 67

A preflow that has a valid labelling saturates a cut.

Proof:

- There are n nodes but $n+1$ different labels from $0,\ldots,n$.
- There must exist a label $d\in\{0,\ldots,n\}$ such that none of the nodes carries this label.
- Let $A=\{v\in V\mid \ell(v)>d\}$ and $B=\{v\in V\mid \ell(v)<d\}$.
- We have $s\in A$ and $t\in B$ and there is no edge from A to B in the residual graph G_f; this means that (A,B) is a saturated cut.

Preflows

Lemma 67

A preflow that has a valid labelling saturates a cut.

Proof:

- There are n nodes but $n + 1$ different labels from $0, \ldots, n$.
- There must exist a label $d \in \{0, \ldots, n\}$ such that none of the nodes carries this label.
- Let $A = \{v \in V \mid \ell(v) > d\}$ and $B = \{v \in V \mid \ell(v) < d\}$.
- We have $s \in A$ and $t \in B$ and there is no edge from A to B in the residual graph G_f; this means that (A, B) is a saturated cut.

Lemma 68

A flow that has a valid labelling is a maximum flow.

Push Relabel Algorithms

Push Relabel Algorithms

Idea:

- start with some preflow and some valid labelling

Push Relabel Algorithms

Idea:

- start with some preflow and some valid labelling
- successively change the preflow while maintaining a valid labelling

Push Relabel Algorithms

Idea:

- start with some preflow and some valid labelling
- successively change the preflow while maintaining a valid labelling
- stop when you have a flow (i.e., no more active nodes)

Changing a Preflow

An arc (u, v) with $c_f(u, v) > 0$ in the residual graph is admissible if $\ell(u) = \ell(v) + 1$ (i.e., it goes downwards w.r.t. labelling ℓ).

The push operation

Consider an active node u with excess flow $f(u) = \sum_{e \in \text{into}(u)} f(e) - \sum_{e \in \text{out}(u)} f(e)$ and suppose $e = (u, v)$ is an admissible arc with residual capacity $c_f(e)$.

We can send flow $\min\{c_f(e), f(u)\}$ along e and obtain a new preflow. The old labelling is still valid (!!!).

- saturating push: $\min\{c_f(e), f(u)\} = c_f(e)$
  the arc e is deleted from the residual graph
- deactivating push: $\min\{c_f(e), f(u)\} = f(u)$
  the node u becomes inactive

Changing a Preflow

An arc (u, v) with $c_f(u, v) > 0$ in the residual graph is admissible if $\ell(u) = \ell(v) + 1$ (i.e., it goes downwards w.r.t. labelling ℓ).

The push operation

Consider an active node u with excess flow $f(u) = \sum_{e \in \mathrm{into}(u)} f(e) - \sum_{e \in \mathrm{out}(u)} f(e)$ and suppose $e = (u, v)$ is an admissible arc with residual capacity $c_f(e)$.

We can send flow $\min\{c_f(e), f(u)\}$ along e and obtain a new preflow. The old labelling is still valid (!!!).

- saturating push: $\min\{f(u), c_f(e)\} = c_f(e)$
  the arc e is deleted from the residual graph
- deactivating push: $\min\{f(u), c_f(e)\} = f(u)$
  the node u becomes inactive

Changing a Preflow

An arc (u,v) with $c_f(u,v) > 0$ in the residual graph is admissible if $\ell(u) = \ell(v) + 1$ (i.e., it goes downwards w.r.t. labelling ℓ).

The push operation
Consider an active node u with excess flow $f(u) = \sum_{e\in\text{into}(u)} f(e) - \sum_{e\in\text{out}(u)} f(e)$ and suppose $e = (u,v)$ is an admissible arc with residual capacity $c_f(e)$.

We can send flow $\min\{c_f(e), f(u)\}$ along e and obtain a new preflow. The old labelling is still valid (!!!).

- saturating push: $\min\{f(u), c_f(e)\} = c_f(e)$
  the arc e is deleted from the residual graph
- non-saturating push: $\min\{f(u), c_f(e)\} = f(u)$
  the node u becomes inactive

Changing a Preflow

An arc (u, v) with $c_f(u, v) > 0$ in the residual graph is admissible if $\ell(u) = \ell(v) + 1$ (i.e., it goes downwards w.r.t. labelling ℓ).

The push operation
Consider an active node u with excess flow $f(u) = \sum_{e \in \text{into}(u)} f(e) - \sum_{e \in \text{out}(u)} f(e)$ and suppose $e = (u, v)$ is an admissible arc with residual capacity $c_f(e)$.

We can send flow $\min\{c_f(e), f(u)\}$ along e and obtain a new preflow. The old labelling is still valid (!!!).

Changing a Preflow

An arc (u, v) with $c_f(u, v) > 0$ in the residual graph is admissible if $\ell(u) = \ell(v) + 1$ (i.e., it goes downwards w.r.t. labelling ℓ).

The push operation

Consider an active node u with excess flow $f(u) = \sum_{e \in \mathrm{into}(u)} f(e) - \sum_{e \in \mathrm{out}(u)} f(e)$ and suppose $e = (u, v)$ is an admissible arc with residual capacity $c_f(e)$.

We can send flow $\min\{c_f(e), f(u)\}$ along e and obtain a new preflow. The old labelling is still valid (!!!).

- saturating push: $\min\{f(u), c_f(e)\} = c_f(e)$
  the arc e is deleted from the residual graph
- deactivating push: $\min\{f(u), c_f(e)\} = f(u)$
  the node u becomes inactive

Changing a Preflow

An arc (u,v) with $c_f(u,v) > 0$ in the residual graph is admissible if $\ell(u) = \ell(v) + 1$ (i.e., it goes downwards w.r.t. labelling ℓ).

The push operation

Consider an active node u with excess flow $f(u) = \sum_{e\in\text{into}(u)} f(e) - \sum_{e\in\text{out}(u)} f(e)$ and suppose $e = (u,v)$ is an admissible arc with residual capacity $c_f(e)$.

We can send flow $\min\{c_f(e), f(u)\}$ along e and obtain a new preflow. The old labelling is still valid (!!!).

- saturating push: $\min\{f(u), c_f(e)\} = c_f(e)$
  the arc e is deleted from the residual graph
- deactivating push: $\min\{f(u), c_f(e)\} = f(u)$
  the node u becomes inactive

Push Relabel Algorithms

Push Relabel Algorithms

The relabel operation

Consider an active node u that does not have an outgoing admissible arc.

Push Relabel Algorithms

The relabel operation

Consider an active node u that does not have an outgoing admissible arc.

Increasing the label of u by 1 results in a valid labelling.

Push Relabel Algorithms

The relabel operation

Consider an active node u that does not have an outgoing admissible arc.

Increasing the label of u by 1 results in a valid labelling.

- Edges (w, u) incoming to u still fulfill their constraint $\ell(w) \leq \ell(u) + 1$.

Push Relabel Algorithms

The relabel operation

Consider an active node u that does not have an outgoing admissible arc.

Increasing the label of u by 1 results in a valid labelling.

- Edges (w, u) incoming to u still fulfill their constraint $\ell(w) \leq \ell(u) + 1$.
- An outgoing edge (u, w) had $\ell(u) < \ell(w) + 1$ before since it was not admissible. Now: $\ell(u) \leq \ell(w) + 1$.

Push Relabel Algorithms

Intuition:

We want to send flow downwards, since the source has a height/label of n and the target a height/label of 0. If we see an active node u with an admissible arc we push the flow at u towards the other end-point that has a lower height/label. If we do not have an admissible arc but excess flow into u it should roughly mean that the level/height/label of u should rise. (If we consider the flow to be water then this would be natural.)

Note that the above intuition is very incorrect as the labels are integral, i.e., they cannot really be seen as the height of a node.

Reminder

- In a preflow nodes may not fulfill conservation constraints; a node may have more incoming flow than outgoing flow.
- Such a node is called active.
- A labelling is valid if for every edge (u,v) in the residual graph $\ell(u) \leq \ell(v) + 1$.
- An arc (u,v) in residual graph is admissible if $\ell(u) = \ell(v) + 1$.
- A saturating push along e pushes an amount of $c(e)$ flow along the edge, thereby saturating the edge (and making it dissappear from the residual graph).
- A deactivating push along $e = (u,v)$ pushes a flow of $f(u)$, where $f(u)$ is the excess flow of u. This makes u inactive.

Push Relabel Algorithms

Algorithm 19 maxflow(G, s, t, c)

1: find initial preflow f
2: **while** there is active node u **do**
3:     **if** there is admiss. arc e out of u **then**
4:         push(G, e, f, c)
5:     **else**
6:         relabel(u)
7: **return** f