Max-flow and min-cut problems

Maximum Matching problem

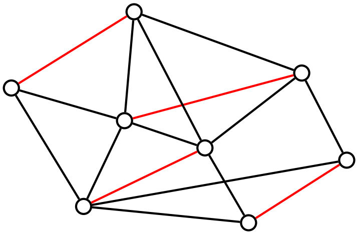
Given an undirected graph $G = (V, E)$ a subset of edges $M \subseteq E$ is a matching if each node appears at most in one edge in M (a node may not appear at all).

Maximum Matching problem:
Given a graph G, find a matching with maximum cardinality.

MAXIMUM MATCHING problem

Given an undirected graph $G = (V, E)$ a subset of edges $M \subseteq E$ is a matching if each node appears at most in one edge in M (a node may not appear at all).

MAXIMUM MATCHING problem:
Given a graph G, find a matching with maximum cardinality.



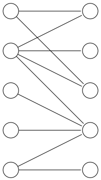
Maximum matching in bipartite graphs

A graph $G = (V, E)$ is bipartite if there is a partition of V in L and R, $(L \cup R = V$ and $L \cap R = \emptyset)$, such that every $e \in E$ connects a vertex in L with a vertex in R.

Maximum matching in bipartite graphs

A graph $G = (V, E)$ is bipartite if there is a partition of V in L and R, $(L \cup R = V$ and $L \cap R = \emptyset)$, such that every $e \in E$ connects a vertex in L with a vertex in R.

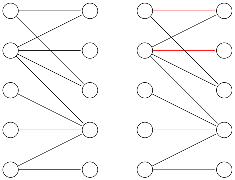
We want to solve the MAXIMUM MATCHING problem on bipartite graphs



Maximum matching in bipartite graphs

A graph $G = (V, E)$ is bipartite if there is a partition of V in L and R, ($L \cup R = V$ and $L \cap R = \emptyset$), such that every $e \in E$ connects a vertex in L with a vertex in R.

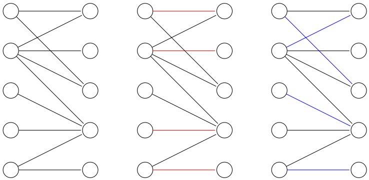
We want to solve the Maximum Matching problem on bipartite graphs



Maximum matching in bipartite graphs

A graph $G = (V, E)$ is bipartite if there is a partition of V in L and R, ($L \cup R = V$ and $L \cap R = \emptyset$), such that every $e \in E$ connects a vertex in L with a vertex in R.

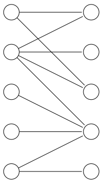
We want to solve the MAXIMUM MATCHING problem on bipartite graphs

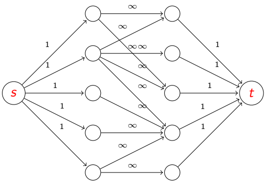


Maximum Matching: Network formulation

From $G = (L \cup R, E)$ construct $\mathcal{N} = (\hat{V}, \hat{E}, c, s, t)$:

- Add vertices s and t: $\hat{V} = L \cup R \cup \{s, t\}$.
- Add directed edges $s \to L$ with capacity 1. Add directed edges $R \to t$ with capacity 1.
- Direct the edges E from L to R, and give them capacity ∞.
- $\hat{E} = \{s \to L\} \cup E \cup \{R \to t\}$.





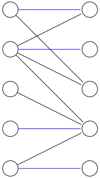
Maximum matching algorithm: Analysis

Theorem

Max flow in $\mathcal{N}$=Max bipartite matching in G.

Proof Matching as flows

Let M be a matching in G with k-edges, consider the flow f that sends 1 unit along each one of the k paths,
$s \to u \to v \to t$, for $(u, v) \in M$.
As M is a matching all these paths are disjoint, so f is a flow and has value k.



Maximum matching algorithm: Analysis

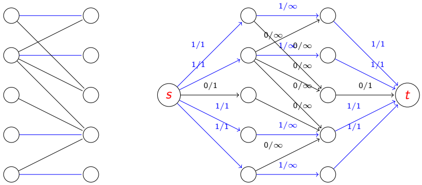
Theorem

Max flow in $\mathcal{N}$=Max bipartite matching in G.

Proof Matching as flows

Let M be a matching in G with k-edges, consider the flow f that sends 1 unit along each one of the k paths, $s \to u \to v \to t$, for $(u, v) \in M$.

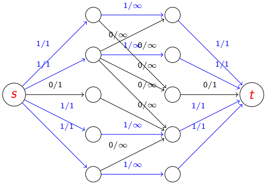
As M is a matching all these paths are disjoint, so f is a flow and has value k.



Maximum matching algorithm: Analysis

Flows as matchings

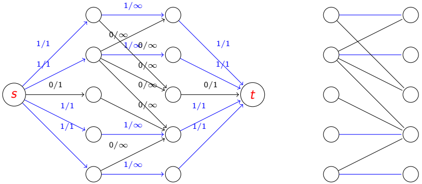
- Consider an integral flow f in $\hat{G}$. Therefore, for any edge e, the flow is either 0 or 1.
- Consider the cut $C = (\{s\} \cup L, R \cup \{t\})$ in $\hat{G}$.
- Let M be the set of edges in the cut C with flow=1, then $|M| = |f|$.
- Each node in L is in at most one $e \in M$ and every node in R is in at most one head of an $e \in F$
- Therefore, M is a matching in G with $|M| = |f|$



Maximum matching algorithm: Analysis

Flows as matchings

- Consider an integral flow f in $\hat{G}$. Therefore, for any edge e, the flow is either 0 or 1.
- Consider the cut $C = (\{s\} \cup L, R \cup \{t\})$ in $\hat{G}$.
- Let M be the set of edges in the cut C with flow=1, then $|M| = |f|$.
- Each node in L is in at most one $e \in M$ and every node in R is in at most one head of an $e \in F$
- Therefore, M is a matching in G with $|M| = |f|$



Maximum matching algorithm: Analysis

As $\mathcal{N}$ has integer capacities there is an integral maximum flow f^*, the associated matching is a maximum matching. $\square$

Maximum matching algorithm: Analysis

What is the cost of the algorithm?

Maximum matching algorithm: Analysis

What is the cost of the algorithm?

- The bipartite graph, has n vertices and m edges. The capacities are integers. We need an integral solution.
- The algorithm: (1) constructs $\mathcal{N}$, (2) runs FF on $\mathcal{N}$ to obtain a maxflow f, (3) from f obtain a maximum matching M.

Maximum matching algorithm: Analysis

What is the cost of the algorithm?

- The bipartite graph, has n vertices and m edges. The capacities are integers. We need an integral solution.
- The algorithm: (1) constructs $\mathcal{N}$, (2) runs FF on $\mathcal{N}$ to obtain a maxflow f, (3) from f obtain a maximum matching M.
- $\mathcal{N}$ has $n+2$ vertices and $m+2n$ edge, (1) takes $O(n+m)$

Maximum matching algorithm: Analysis

What is the cost of the algorithm?

- The bipartite graph, has n vertices and m edges. The capacities are integers. We need an integral solution.
- The algorithm: (1) constructs $\mathcal{N}$, (2) runs FF on $\mathcal{N}$ to obtain a maxflow f, (3) from f obtain a maximum matching M.
- $\mathcal{N}$ has $n+2$ vertices and $m+2n$ edge, (1) takes $O(n+m)$
- The maximum value of a flow in $\mathcal{N}$ is at most n, (2) takes time $O(|f|(n+m)) = O(n(n+m))$

Maximum matching algorithm: Analysis

What is the cost of the algorithm?

- The bipartite graph, has n vertices and m edges. The capacities are integers. We need an integral solution.
- The algorithm: (1) constructs $\mathcal{N}$, (2) runs FF on $\mathcal{N}$ to obtain a maxflow f, (3) from f obtain a maximum matching M.
- $\mathcal{N}$ has $n+2$ vertices and $m+2n$ edge, (1) takes $O(n+m)$
- The maximum value of a flow in $\mathcal{N}$ is at most n, (2) takes time $O(|f|(n+m)) = O(n(n+m))$
- (3) can be done in time $O(n+m)$.

Maximum matching algorithm: Analysis

What is the cost of the algorithm?

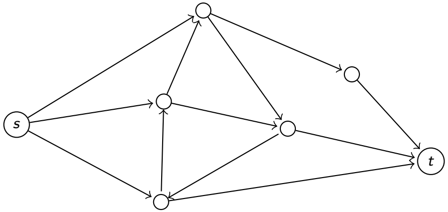
- The bipartite graph, has n vertices and m edges. The capacities are integers. We need an integral solution.
- The algorithm: (1) constructs $\mathcal{N}$, (2) runs FF on $\mathcal{N}$ to obtain a maxflow f, (3) from f obtain a maximum matching M.
- $\mathcal{N}$ has $n+2$ vertices and $m+2n$ edge, (1) takes $O(n+m)$
- The maximum value of a flow in $\mathcal{N}$ is at most n, (2) takes time $O(|f|(n+m)) = O(n(n+m))$
- (3) can be done in time $O(n+m)$.

So, the cost is $O(n(n+m))$.

DISJOINT PATH problem

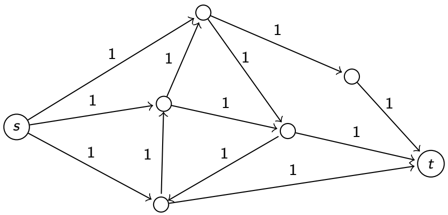
Given a digraph $G = (V, E)$ and two vertices $s, t \in V$, a set of paths is edge-disjoint if their edges are disjoint (although they might share some vertex)

DISJOINT PATH problem: Given a digraph $G = (V, E)$ and two vertices $s, t \in V$, find a set of $s \rightsquigarrow t$ edge-disjoint paths of maximum cardinality



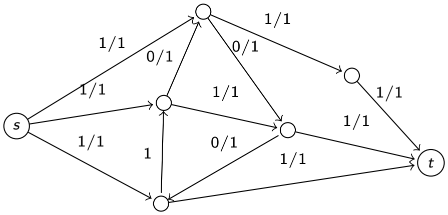
DISJOINT PATH: Max flow formulation

Thinking in terms of flow a path from s to t can be seen as a way of transporting a unit of flow.
We construct a network $\mathcal{N}$ assigning unit capacity to every edge.



Disjoint Path: Max flow formulation

Thinking in terms of flow a path from s to t can be seen as a way of transporting a unit of flow.
We construct a network $\mathcal{N}(G)$ assigning unit capacity to every edge
We solve MaxFlow for $\mathcal{N}(G)$.



DISJOINT PATH: Max flow formulation

Thinking in terms of flow a path from s to t can be seen as a way of transporting a unit of flow.
We construct a network $\mathcal{N}(G)$ assigning unit capacity to every edge

DISJOINT PATH: Max flow formulation

Thinking in terms of flow a path from s to t can be seen as a way of transporting a unit of flow.
We construct a network $\mathcal{N}(G)$ assigning unit capacity to every edge

Theorem

The max number of edge disjoint paths $s \rightsquigarrow t$ in G is equal to the max flow value in $\mathcal{N}(G)$

Disjoint Path: Proof of the Theorem

Proof.

Number of disjoints paths $\leq$ max flow

If we have k edge-disjoints paths $s \rightsquigarrow t$ in G then making $f(e) = 1$ for each e in a path, we get a valid flow f with $|f| = k$

Disjoint Path: Proof of the Theorem

Number of disjoints paths $\geq$ max flow

- If the max flow value is k, there exists a 0-1 flow f^* with value k.
- Consider the graph $G^* = (V, E')$ where E' is formed by all edges e with $f(e) = 1$.
- We repeatedly compute a $s \rightsquigarrow t$ simple path in G^*, and remove its edges from G^*.
- Each time that we remove a path, the value of the flow in the network is reduced by one, so we can apply the process k times.
- None of the paths share an edge, so we get k disjoint paths. $\square$

Disjoint Path: Proof of the Theorem

Number of disjoints paths $\geq$ max flow

- If the max flow value is k, there exists a 0-1 flow f^* with value k.
- Consider the graph $G^* = (V, E')$ where E' is formed by all edges e with $f(e) = 1$.
- We repeatedly compute a $s \rightsquigarrow t$ simple path in G^*, and remove its edges from G^*.
- Each time that we remove a path, the value of the flow in the network is reduced by one, so we can apply the process k times.
- None of the paths share an edge, so we get k disjoint paths. $\square$

End Proof

DISJOINT PATH: Max flow + path extraction algorithm

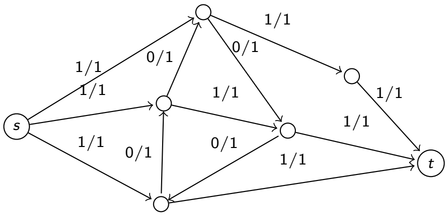
Algorithm

1. Construct the network $\mathcal{N}(G)$ assigning unit capacity to every edge
2. Solve MaxFlow for $\mathcal{N}(G)$
3. Extract the set of disjoint paths on the graph restricted to edges with flow > 0

Disjoint Path: Max flow + path extraction algorithm

Algorithm

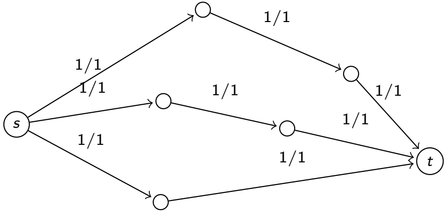
1. Construct the network $\mathcal{N}(G)$ assigning unit capacity to every edge
2. Solve MaxFlow for $\mathcal{N}(G)$
3. Extract the set of disjoint paths on the graph restricted to edges with flow > 0



DISJOINT PATH: Max flow + path extraction algorithm

Algorithm

1. Construct the network $\mathcal{N}$ assigning unit capacity to every edge
2. Solve MaxFlow for $\mathcal{N}$
3. Extract the set of disjoint paths on the graph restricted to edges with flow > 0



Disjoint paths algorithm: Analysis

What is the cost of the algorithm?

Disjoint paths algorithm: Analysis

What is the cost of the algorithm?

- The graph, has n vertices and m edges. The capacities are integers. We need an integral solution.
- The algorithm: (1) constructs $\mathcal{N}$, (2) runs FF on $\mathcal{N}$ to obtain a max flow f, (3) from f obtains $|f|$ edge disjoint paths.

Disjoint paths algorithm: Analysis

What is the cost of the algorithm?

- The graph, has n vertices and m edges. The capacities are integers. We need an integral solution.
- The algorithm: (1) constructs $\mathcal{N}$, (2) runs FF on $\mathcal{N}$ to obtain a max flow f, (3) from f obtains $|f|$ edge disjoint paths.
- $\mathcal{N}$ has n vertices and m edges, (1) takes $O(n+m)$

Disjoint paths algorithm: Analysis

What is the cost of the algorithm?

- The graph, has n vertices and m edges. The capacities are integers. We need an integral solution.
- The algorithm: (1) constructs $\mathcal{N}$, (2) runs FF on $\mathcal{N}$ to obtain a max flow f, (3) from f obtains $|f|$ edge disjoint paths.
- $\mathcal{N}$ has n vertices and m edges, (1) takes $O(n+m)$
- The maximum value of a flow in $\mathcal{N}$ is at most n, (2) takes time $O(|f|(n+m)) = O(n(n+m))$

Disjoint paths algorithm: Analysis

What is the cost of the algorithm?

- The graph, has n vertices and m edges. The capacities are integers. We need an integral solution.
- The algorithm: (1) constructs $\mathcal{N}$, (2) runs FF on $\mathcal{N}$ to obtain a max flow f, (3) from f obtains $|f|$ edge disjoint paths.
- $\mathcal{N}$ has n vertices and m edges, (1) takes $O(n+m)$
- The maximum value of a flow in $\mathcal{N}$ is at most n, (2) takes time $O(|f|(n+m)) = O(n(n+m))$
- (3) can be done in time $O(n+m)$ per path, i.e., $O(|f|(n+m))$.

Disjoint paths algorithm: Analysis

What is the cost of the algorithm?

- The graph, has n vertices and m edges. The capacities are integers. We need an integral solution.
- The algorithm: (1) constructs $\mathcal{N}$, (2) runs FF on $\mathcal{N}$ to obtain a max flow f, (3) from f obtains $|f|$ edge disjoint paths.
- $\mathcal{N}$ has n vertices and m edges, (1) takes $O(n+m)$
- The maximum value of a flow in $\mathcal{N}$ is at most n, (2) takes time $O(|f|(n+m)) = O(n(n+m))$
- (3) can be done in time $O(n+m)$ per path, i.e., $O(|f|(n+m))$.

So the cost is $O(n(n+m))$.

Vertex disjoint paths

Can we do something similar to get the maximum number of vertex disjoint paths?

The case of undirected graphs

If we have an undirected graph, with two distinguised nodes u, v, how would you apply the max flow formulation to solve the problem of finding the max number of disjoint paths between u and v?

The case of undirected graphs

Dinic and Edmonds-Karp algorithm

Maximum matching in Bip graphs
Disjoint paths problem
Edmonds Karp alg
Generalized assignment problems
Circulations
Demands
Lower bounds
Survey design
Joint rounding
Min cost Max Flow

J.Edmonds, R. Karp: *Theoretical improvements in algorithmic efficiency for network flow problems*. Journal ACM 1972.

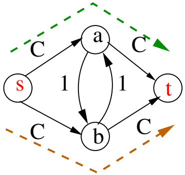
Yefim Dinic: *Algorithm for solution of a problem of maximum flow in a network with power estimation*. Doklady Ak.N. 1970

Choosing a good augmenting path can lead to a faster algorithm.
Use BFS to find an augmenting paths in G_f.

Edmonds-Karp algorithm

FF algorithm but using BFS: choose the augmenting path in G_f with the smallest length (number of edges).

Edmonds-Karp(G, c, s, t)
For all $e = (u, v) \in E$ let $f(u, v) = 0$
$G_f = G$
while there is an $s \rightsquigarrow t$ path in G_f
do
  $P = \text{BFS}(G_f, s, t)$
  $f = \text{Augment}(f, P)$
  Compute G_f
return f



The BFS in EK will choose: or

BFS paths on G_f

For $\mathcal{N} = (V, E, c, s, t)$ and a flow f in $\mathcal{N}$, assuming that G_f has an augmenting path, let f' be the next flow after executing one step of the EK algorithm.

BFS paths on G_f

For $\mathcal{N} = (V, E, c, s, t)$ and a flow f in $\mathcal{N}$, assuming that G_f has an augmenting path, let f' be the next flow after executing one step of the EK algorithm.

- The path from s to t in a BFS traversal starting at s, is a path $s \rightsquigarrow t$ with minimum number of edges, i.e., a shortest length path.

BFS paths on G_f

For $\mathcal{N} = (V, E, c, s, t)$ and a flow f in $\mathcal{N}$, assuming that G_f has an augmenting path, let f' be the next flow after executing one step of the EK algorithm.

- The path from s to t in a BFS traversal starting at s, is a path $s \rightsquigarrow t$ with minimum number of edges, i.e., a shortest length path.
- For $\in V$, let $\delta_f(s, v)$ denote length of a shortest length path from s to v in G_f.

Some properties of G_f and $G_{f'}$

Some properties of G_f and $G_{f'}$

How can we have $(u, v) \in E_{f'}$ but $(u, v) \notin E_f$?

- (u, v) is a forward edge saturated in f and not in f''.
- (u, v) is a backward edge in G_f and $f(v, u) = 0$

Some properties of G_f and $G_{f'}$

How can we have $(u, v) \in E_{f'}$ but $(u, v) \notin E_f$?

- (u, v) is a forward edge saturated in f and not in f''.
- (u, v) is a backward edge in G_f and $f(v, u) = 0$

In any of the two cases, the augmentation must have modified the flow from v to u, so (u, v) must form part of the augmenting path.

EK and the shortest length distances

Lemma

If the EK-algorithm runs on $\mathcal{N} = (V, E, c, s, t)$, for all vertices $v \neq s$, $\delta_f(s, v)$ increases monotonically with each flow augmentation.

Proof. By contradiction.

Let f be the first flow such that, for some $u \neq s$,

$$\delta_{f'}(s, u) < \delta_f(s, u).$$

EK and the shortest length distances

Proof (cont)

Let v be the vertex with the minimum $\delta_{f'}(s, v)$ whose distance was decreased.

- Let $P : s \rightsquigarrow u \rightarrow v$ be a shortest length path from s to v in $G_{f'}$
- Then, $\delta_{f'}(s, v) = \delta_{f'}(s, u) + 1$ and $\delta_{f'}(s, u) \geq \delta_f(s, u)$.
- If $(u, v) \in E_f$,
  $\delta_f(s, v) \leq \delta_f(s, u) + 1 \leq \delta_{f'}(s, u) + 1 = \delta_{f'}(s, v)$
- So, $(u, v) \notin E_f$

EK and the shortest length distances

Proof (cont)

How can we have ?

- $(u, v) \in E_{f'}$ but $(u, v) \notin E_f$
- If so, (v, u) appears in the augmenting path.

EK and the shortest length distances

Proof (cont)

How can we have ?

- $(u, v) \in E_{f'}$ but $(u, v) \notin E_f$
- If so, (v, u) appears in the augmenting path.
- Then, the shortest length path from s to u in G_f has (v, u) as it last edge.
  $\delta_f(s, v) \leq \delta_f(s, u) - 1 \leq \delta_{f'}(s, u) - 1 = \delta_{f'}(s, v) - 1 - 1$

EK and the shortest length distances

Proof (cont)

How can we have ?

- $(u, v) \in E_{f'}$ but $(u, v) \notin E_f$
- If so, (v, u) appears in the augmenting path.
- Then, the shortest length path from s to u in G_f has (v, u) as it last edge.
  $\delta_f(s, v) \leq \delta_f(s, u) - 1 \leq \delta_{f'}(s, u) - 1 = \delta_{f'}(s, v) - 1 - 1$
- which contradicts $\delta_{f'}(s, v) < \delta_f(u, v)$. $\square$

Some properties of G_f and $G_{f'}$

Some properties of G_f and $G_{f'}$

Let P be an augmenting path in G_f.

$(u, v) \in P$ is critical if $b(P) = c_f(u, v)$.

Some properties of G_f and $G_{f'}$

Let P be an augmenting path in G_f.

$(u, v) \in P$ is critical if $b(P) = c_f(u, v)$.

Critical edges do not appear in $G_{f'}$.

- (u, v) forward, $f'(u, v) = c(u, v)$
- (u, v) backward, $f'(v, u) = 0$

EK and critical edges

Lemma

In the EK algorithm, each one of the edges can become critical at most $|V|/2$ times.

Proof:

- Let $(u, v) \in E$, when (u, v) is critical for the first time, $\delta_f(s, v) = \delta_f(s, u) + 1$

EK and critical edges

Lemma

In the EK algorithm, each one of the edges can become critical at most $|V|/2$ times.

Proof:

- Let $(u, v) \in E$, when (u, v) is critical for the first time, $\delta_f(s, v) = \delta_f(s, u) + 1$
- After this step (u, v) disappears from the residual graph until after the flow in (u, v) changes.

EK and critical edges

Lemma

In the EK algorithm, each one of the edges can become critical at most $|V|/2$ times.

Proof:

- Let $(u,v) \in E$, when (u,v) is critical for the first time, $\delta_f(s,v) = \delta_f(s,u) + 1$
- After this step (u,v) disappears from the residual graph until after the flow in (u,v) changes.
- At this point, (v,u) forms part of the augmenting path in $G_{f'}$, and $\delta_{f'}(s,u) = \delta_{f'}(s,v) + 1$,

$$\delta_{f'}(s,u) = \delta_{f'}(s,v) + 1 \geq \delta_f(s,v) + 1 \geq \delta_f(s,u) + 2$$

EK and critical edges

Lemma

In the EK algorithm, each one of the edges can become critical at most $|V|/2$ times.

Proof:

- Let $(u, v) \in E$, when (u, v) is critical for the first time, $\delta_f(s, v) = \delta_f(s, u) + 1$
- After this step (u, v) disappears from the residual graph until after the flow in (u, v) changes.
- At this point, (v, u) forms part of the augmenting path in $G_{f'}$, and $\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1$,

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1 \geq \delta_f(s, v) + 1 \geq \delta_f(s, u) + 2$$

- So, the distance has increased by at least 2.

Complexity of Edmonds-Karp algorithm

Theorem

The EK algorithms runs in $O(mn(n+m))$ steps. Therefore it is a polynomial time algorithm.

Complexity of Edmonds-Karp algorithm

Theorem

The EK algorithms runs in $O(mn(n+m))$ steps. Therefore it is a polynomial time algorithm.

Proof:

- Need time $O(m+n)$ to find the augmenting path using BFS.
- By the previous Lemma, there are $O(mn)$ augmentations. □

Finding a min-cut

Given (G, s, t, c) to find a min-cut:

1. Compute the max-flow f^* in G.
2. Obtain G_{f^*}.
3. Find the set $S = \{v \in V | s \rightsquigarrow v\}$ in G_{f^*}.
4. Output the cut $(S, V - \{S\}) = \{(v, u) | v \in S and u \in V - \{S\}\}$ in G.

The running time is the same than the algorithm to find the max-flow.

The max-flow problems: History

- Ford-Fulkerson (1956) $O(mC)$, where C is the max flow val.
- Dinic (1970) (blocking flow) $O(n^2 m)$
- Edmond-Karp (1972) (shortest augmenting path) $O(nm^2)$
- Karzanov (1974), $O(n^2 m)$ Goldberg-Tarjant (1986) (push re-label preflow + dynamic trees) $O(nm \lg(n^2/m))$ (uses parallel implementation)
- King-Rao-Tarjan (1998) $O(nm \log_{m/n \lg n} n)$.
- J. Orlin (2013) $O(nm)$ (clever follow up to KRT-98)
- Chen, Kyng, Liu, Peng, Gutenberg, Sachdeva (2022) $O(m^{1+o(1)})$ (polynomially bounded integral capacities) You can read Quanta Magazine article.

Applications: Generalized assignment problems

- In a generalized assignment problem $\mathcal{GP}$, we have as input d finite sets $X_1, \ldots, X_d$, each representing a different set of resources.

Applications: Generalized assignment problems

- In a generalized assignment problem $\mathcal{GP}$, we have as input d finite sets $X_1, \ldots, X_d$, each representing a different set of resources.
- Our goal is to chose the "largest" number of d-tuples, each d-tuple containing exactly one element from each X_i, subject to the constrains:

Applications: Generalized assignment problems

- In a generalized assignment problem $\mathcal{GP}$, we have as input d finite sets $X_1, \ldots, X_d$, each representing a different set of resources.
- Our goal is to chose the "largest" number of d-tuples, each d-tuple containing exactly one element from each X_i, subject to the constrains:
  - For each $i \in [d]$, each $x \in X_i$ can appears in at most $c(x)$ selected tuples.
  - For each $i \in [d]$, any two $x \in X_i$ and $y \in X_{i+1}$ can appear in at most $c(x, y)$ selected tuples.
  - The values for $c(x)$ and $c(x, y)$ are either in $\mathbb{Z}^+$ or ∞.

Applications: Generalized assignment problems

- In a generalized assignment problem $\mathcal{GP}$, we have as input d finite sets $X_1, \ldots, X_d$, each representing a different set of resources.
- Our goal is to chose the "largest" number of d-tuples, each d-tuple containing exactly one element from each X_i, subject to the constrains:
  - For each $i \in [d]$, each $x \in X_i$ can appears in at most $c(x)$ selected tuples.
  - For each $i \in [d]$, any two $x \in X_i$ and $y \in X_{i+1}$ can appear in at most $c(x, y)$ selected tuples.
  - The values for $c(x)$ and $c(x, y)$ are either in $\mathbb{Z}^+$ or ∞.
- Notice that only pairs of objects between adjacent X_i and X_{i+1} are constrained.

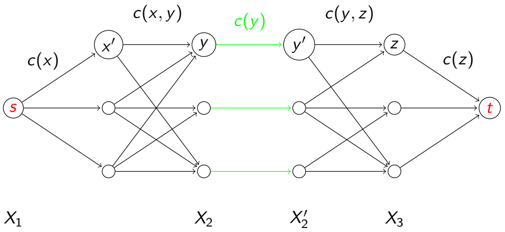
Applications: Generic reduction to Max-Flow

Make the reduction from $\mathcal{GP}$ to the following network $\mathcal{N}$:

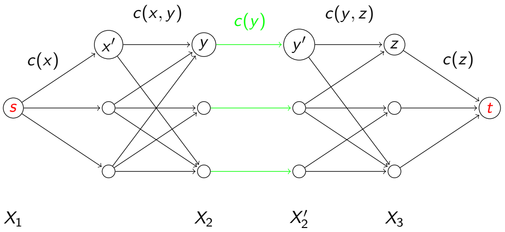
- V contains a vertex x, for each element x in each X_i, and a copy x', for each element $x \in X_i$ for $1 \leq i < d$.
- We add vertex s and vertex t.
- Add an edge $s \rightarrow x$ for each $x \in X_1$ and add an edge $y \rightarrow t$ for every $y \in X_d$. Give capacities $c(s,x) = c(x)$ and $c(y,t) = c(y)$.
- Add an edge $x' \rightarrow y$ for every pair $x \in X_i$ and $y \in X_{i+1}$. Give a capacity $c(x,y)$. Omit the edges with capacity 0.
- For every $x \in X_i$ for $1 \leq i < d$, add an edge $x \rightarrow x'$ with $c(x,x') = c(x)$.

Every path $s \rightsquigarrow t$ in $\mathcal{N}$ identifies a feasible d-tuple, conversely every d-tuple determines a a path $s \rightsquigarrow t$.

Flow Network: The reduction

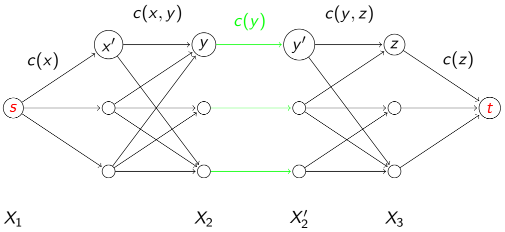


Flow Network: The reduction



- To solve $\mathcal{GP}$, we construct $\mathcal{N}$, and then we find an integer maximum flow f^*.
- In the subgraph formed by edges with $f^*(e) > 0$, we find a (s,t) path P (a d-tuple), decrease in 1 the flow in each edge of P, remove edges with 0 flow.

Flow Network: The reduction



- To solve $\mathcal{GP}$, we construct $\mathcal{N}$, and then we find an integer maximum flow f^*.
- In the subgraph formed by edges with $f^*(e) > 0$, we find a (s, t) path P (a d-tuple), decrease in 1 the flow in each edge of P, remove edges with 0 flow.
- Repeat $|f^*|$ times. In this way we obtain a set of d-tuples with maximum size verifying all the restrictions.

Circulation with demands

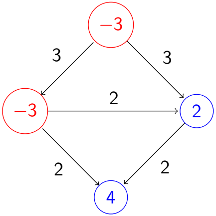
- We introduce another flow problem, to deal with supply and demand inside a network.
- Instead of having a pair source/sink the new setting consider a producer/consumer scenario.

Circulation with demands

- We introduce another flow problem, to deal with supply and demand inside a network.
- Instead of having a pair source/sink the new setting consider a producer/consumer scenario.
- Some nodes are able to produce a certain amount of flow.
- Some nodes are willing to consume flow.
- The question is whether it is possible to route "all" the produced flow to the consumers. When possible the flow assignment is called a circulation

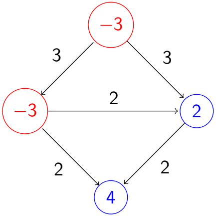
Network with demands

A network with demands $\mathcal{N}$ is a tuple (V, E, c, d) where c assigns a positive capacity to each edge, and d is a function associating a demand $d(v)$, to $v \in V$.



Network with demands

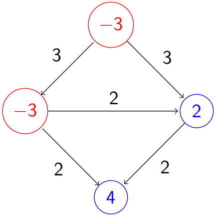
A network with demands $\mathcal{N}$ is a tuple (V, E, c, d) where c assigns a positive capacity to each edge, and d is a function associating a demand $d(v)$, to $v \in V$.



- When $d(v) > 0$, v can receive $d(v)$ units of flow more than it sends, v is a sink,.
- If $d(v) < 0$, v can send $d(v)$ units of flow more than it receives, v is a source.
- If $d(v) = 0$, v is neither a source or a sink.

Network with demands

A network with demands $\mathcal{N}$ is a tuple (V, E, c, d) where c assigns a positive capacity to each edge, and d is a function associating a demand $d(v)$, to $v \in V$.



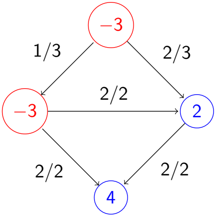
- When $d(v) > 0$, v can receive $d(v)$ units of flow more than it sends, v is a sink,.
- If $d(v) < 0$, v can send $d(v)$ units of flow more than it receives, v is a source.
- If $d(v) = 0$, v is neither a source or a sink.
- Define S to be the set of sources and T the set of sinks.

Network with demands: circulation

Given a network $\mathcal{N} = (V, E, c, d)$, a circulation is a flow assignment $f : E \to \mathbb{R}^+$ s.t.

1. capacity: For each $e \in E$, $0 \leq f(e) \leq c(e)$,
2. conservation: For each $v \in V$,

$$\sum_{(u,v)\in E} f(u,v) - \sum_{(v,z)\in E} f(v,z) = d(v).$$

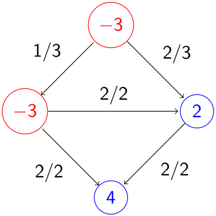


Network with demands: circulation

Given a network $\mathcal{N} = (V, E, c, d)$, a circulation is a flow assignment $f : E \to \mathbb{R}^+$ s.t.

1. capacity: For each $e \in E$, $0 \leq f(e) \leq c(e)$,
2. conservation: For each $v \in V$,

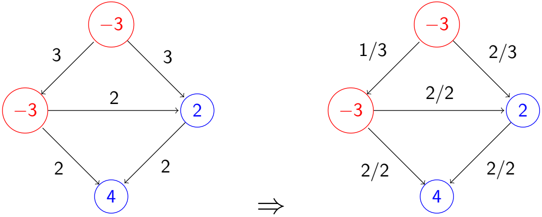
$$\sum_{(u,v)\in E} f(u,v) - \sum_{(v,z)\in E} f(v,z) = d(v).$$



Take into account that a circulation might not exist.

Network with demands: circulation problem

Circulation problem: Given $\mathcal{N} = (V, E, c, d)$ with $c > 0$, obtain a circulation provided it does exists.



A first conditions for a circulation to exists

If f is a circulation for $\mathcal{N} = (V, E, c, d)$,

$$\sum_{v \in V} d(v) = \sum_{v \in V} \left(\underbrace{\sum_{(u,v) \in E} f(u, v)}_{\text{edges to } v} - \underbrace{\sum_{(v,z) \in E} f(v, z)}_{\text{edges out of } v} \right).$$

A first conditions for a circulation to exists

If f is a circulation for $\mathcal{N} = (V, E, c, d)$,

$$\sum_{v \in V} d(v) = \sum_{v \in V} \left(\underbrace{\sum_{(u,v) \in E} f(u, v)}_{\text{edges to } v} - \underbrace{\sum_{(v,z) \in E} f(v, z)}_{\text{edges out of } v} \right).$$

For $e = (u, v) \in E$, $f(e)$ appears in the sum of edges to v and in the sum of edges out of u. Both terms cancel!

A first conditions for a circulation to exists

If f is a circulation for $\mathcal{N} = (V, E, c, d)$,

$$\sum_{v \in V} d(v) = \sum_{v \in V} \left(\underbrace{\sum_{(u,v) \in E} f(u, v)}_{\text{edges to } v} - \underbrace{\sum_{(v,z) \in E} f(v, z)}_{\text{edges out of } v} \right).$$

For $e = (u, v) \in E$, $f(e)$ appears in the sum of edges to v and in the sum of edges out of u. Both terms cancel!

Then, $\sum_{v \in V} d(v) = 0$.

A first conditions for a circulation to exists

If there is a circulation, then $\sum_{v\in V} d(v) = 0$.

Recall that
$S = \{v \in V | d(v) < 0\}$ and
$T = \{v \in V | d(v) > 0\}$.

Define $D = -\sum_{v\in S} d(v) = \sum_{v\in T} d(v)$.

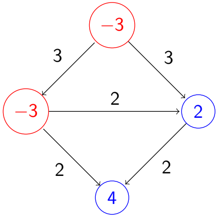
A first conditions for a circulation to exists

If there is a circulation, then $\sum_{v \in V} d(v) = 0$.

Recall that
$S = \{v \in V | d(v) < 0\}$ and
$T = \{v \in V | d(v) > 0\}$.

Define $D = -\sum_{v \in S} d(v) = \sum_{v \in T} d(v)$.

D is the total amount of extra flow that has to be transported from the sources to the sinks.

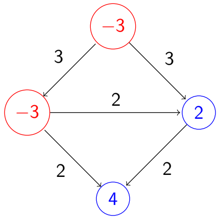


Circulation problem: reduction to Max-flow

From $\mathcal{N} = (V, E, c, d)$, define a flow network $\mathcal{N}' = (V', E', c', s, t)$:

- $V' = V \cup \{s, t\}$, we add a source s and a sink t.
- For $v \in S$ $(d(v) < 0)$, add (s, v) with capacity $-d(v)$.
- For $v \in T$ $(d(v) > 0)$, add (v, t) with capacity $d(v)$.
- Keep E and, for $e \in E$, $c'(e) = c(e)$.

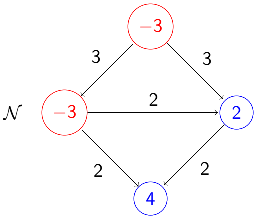
$\mathcal{N}$

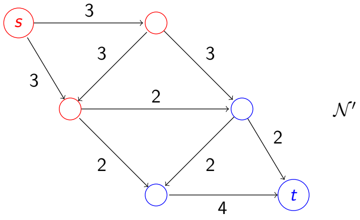


Circulation problem: reduction to Max-flow

From $\mathcal{N} = (V, E, c, d)$, define a flow network $\mathcal{N}' = (V', E', c', s, t)$:

- $V' = V \cup \{s, t\}$, we add a source s and a sink t.
- For $v \in S$ $(d(v) < 0)$, add (s, v) with capacity $-d(v)$.
- For $v \in T$ $(d(v) > 0)$, add (v, t) with capacity $d(v)$.
- Keep E and, for $e \in E$, $c'(e) = c(e)$.





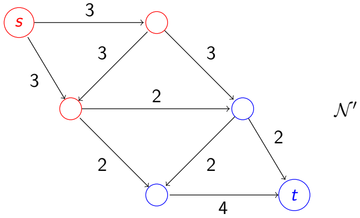
Circulation problem: reduction to Max-flow

1.- Every flow f' in $\mathcal{N}'$ verifies $|f'| \leq D$

Circulation problem: reduction to Max-flow

1.- Every flow f' in $\mathcal{N}'$ verifies $|f'| \leq D$

The capacity $c'(\{s\}, V) = D$, by the capacity restriction on flows, $|f'| \leq D$.



Circulation problem: reduction to Max-flow

2.- If there is a circulation f in $\mathcal{N}$, we have a max-flow f' in $\mathcal{N}'$ with $|f'| = D$.

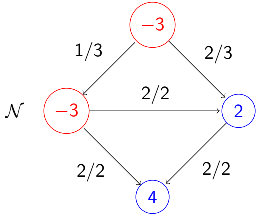
Extend f to a flow f', assigning $f'(s,v) = -d(v)$, for $v \in S$, and $f'(u,t) = d(u)$, for $u \in T$.

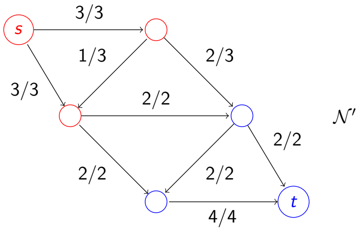
Circulation problem: reduction to Max-flow

2.- If there is a circulation f in $\mathcal{N}$, we have a max-flow f' in $\mathcal{N}'$ with $|f'| = D$.

Extend f to a flow f', assigning $f'(s,v) = -d(v)$, for $v \in S$, and $f'(u,t) = d(u)$, for $u \in T$.

By the circulation condition, f' is a flow in $\mathcal{N}'$. Furthermore, $|f'| = D$.

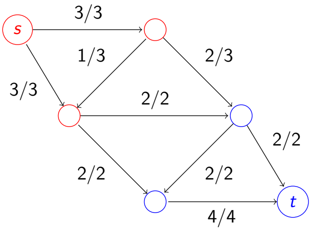




3.- If there is a flow f' in $\mathcal{N}'$ with $|f'| = D$, $\mathcal{N}$ has a circulation

For $e \in E$, define $f(e) = f'(e)$.

- As $|f'| = D$, all edges $(s, v) \in E'$ and $(u, t) \in E'$ are saturated by f'.
- By flow conservation, f satisfies $d(v) = \underbrace{\sum_{(u,v)\in E} f(u,v)}_{\text{edges to } v} - \underbrace{\sum_{(v,z)\in E} f(v,z)}_{\text{edge.s out of } v}$.
- So, f is a circulation for $\mathcal{N}$.



Circulation: main results

From the previous discussion, we can conclude:

Theorem (Necessary and sufficient condition)

There is a circulation for $\mathcal{N} = (V, E, c, d)$ iff the maxflow in $\mathcal{N}'$ has value D.

Circulation: main results

From the previous discussion, we can conclude:

Theorem (Necessary and sufficient condition)

There is a circulation for $\mathcal{N} = (V, E, c, d)$ iff the maxflow in $\mathcal{N}'$ has value D.

Theorem (Circulation integrality theorem)

If all capacities and demands are integers, and there exists a circulation, then there exists an integer valued circulation.

Sketch Proof Max-flow formulation + integrality theorem for max-flow $\square$

Circulation: main results

Theorem

There is a polynomial time algorithm to solve the circulation problem.

The cost of the algorithm is the same as the cost of the algorithm used for the MaxFlow computation.

Circulation: main results

Theorem

There is a polynomial time algorithm to solve the circulation problem.

The cost of the algorithm is the same as the cost of the algorithm used for the MaxFlow computation.

Theorem

If all capacities and demands are integers, and there exists a circulation, then we can obtain an integer valued circulation in time $O(Dm)$.

Networks with demands and lower bounds

Generalization of the previous problem: besides satisfy demands at nodes, we want to force the flow to use certain edges.

Networks with demands and lower bounds

Generalization of the previous problem: besides satisfy demands at nodes, we want to force the flow to use certain edges.

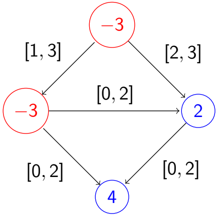
Introduce a new constrain $\ell(e)$ on each $e \in E$, indicating the min-value the flow must be on e.

Networks with demands and lower bounds

Generalization of the previous problem: besides satisfy demands at nodes, we want to force the flow to use certain edges.

Introduce a new constrain $\ell(e)$ on each $e \in E$, indicating the min-value the flow must be on e.

A network $\mathcal{N}$ with demands and lower bounds is a tuple (V, E, c, ℓ, d) with $c(e) \geq \ell(e) \geq 0$, for each $e \in E$,

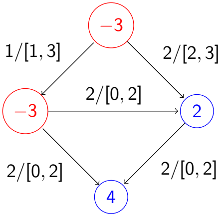


Networks with demands and lower bounds: circulation

Given a network $\mathcal{N} = (V, E, c, \ell, d)$ a circulation as a flow assignment $f : E \to \mathbb{R}^+$ s.t.

1. capacity: For each $e \in E$, $\ell(e) \leq f(e) \leq c(e)$,
2. conservation: For each $v \in V$,

$$\sum_{(u,v)\in E} f(u,v) - \sum_{(v,z)\in E} f(v,z) = d(v).$$

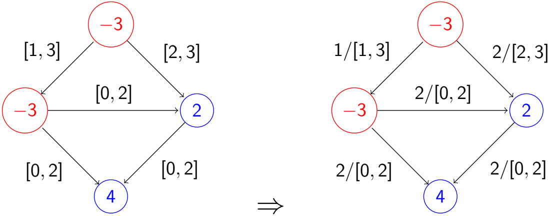


A circulation might not exist.

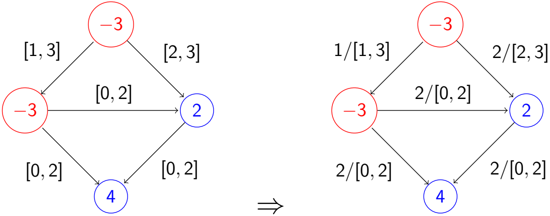
Maximum matching in Bip graphs
Disjoint paths problem
Edmonds Karp alg
Generalized assignment problems
Circulations
Demands
Lower bounds
Survey design
Joint rounding
Min cost Max Flow

Circulations with demands and lower bounds problem

Circulation with demands and lower bounds problem: Given $\mathcal{N} = (V, E, c, \ell, d)$, obtain a circulation for $\mathcal{N}$, provided it does exists



Circulation with demands and lower bounds problem: Given $\mathcal{N} = (V, E, c, \ell, d)$, obtain a circulation for $\mathcal{N}$, provided it does exists



We devise an algorithm to the problem by a reduction to a circulation with demands problem.

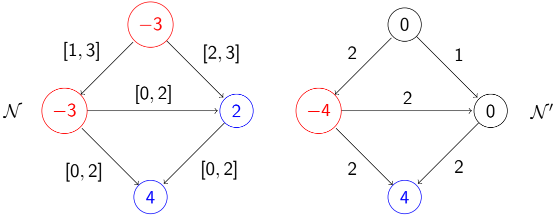
Circulations with demands and lower bounds: the reduction

Let $\mathcal{N} = (V, E, c, \ell, d)$, construct a network $\mathcal{N}' = (V, E, c', d')$ with only demands as follows:

Initially set $c' = c$ and $d' = d$.

For each $e = (u, v) \in E$, with $\ell(e) > 0$:

- $c'(e) = c(e) - \ell(e)$.
- Update the demands on both ends of e: $d'(u) = d'(u) + \ell(e)$ and $d'(v) = d'(v) - \ell(e)$



Circulations with demands and lower bounds: the reduction

1.- If f is a circulation in $\mathcal{N}$, $f'(e) = f(e) - \ell(e)$, for $e \in E$, is a circulation in $\mathcal{N}'$.

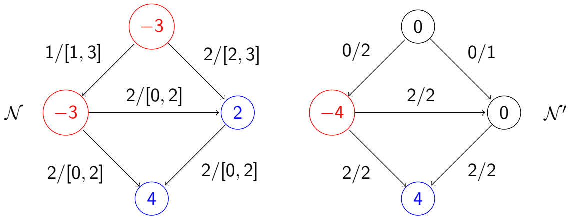
Circulations with demands and lower bounds: the reduction

1.- If f is a circulation in $\mathcal{N}$, $f'(e) = f(e) - \ell(e)$, for $e \in E$, is a circulation in $\mathcal{N}'$.

By construction of $\mathcal{N}'$, f' verifies the capacity constraint.

Besides, for (u, v) with $\ell(u, v) > 0$, the flow out of u and the flow in v is decreased by $\ell(u, v)$.

f is a circulation in $\mathcal{N}$ so, the flow imbalance of f' matches the demand d' at each node.



Circulations with demands and lower bounds: the reduction

2.- If f' is a circulation in $\mathcal{N}'$, $f(e) = f'(e) + \ell(e)$, for $e \in E$, is a circulation in $\mathcal{N}$.

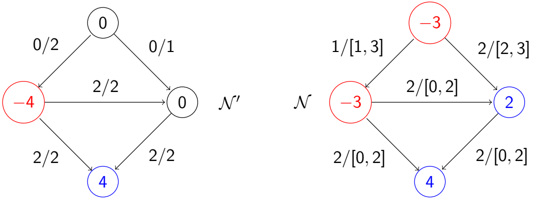
Circulations with demands and lower bounds: the reduction

2.- If f' is a circulation in $\mathcal{N}'$, $f(e) = f'(e) + \ell(e)$, for $e \in E$, is a circulation in $\mathcal{N}$.

f verifies the capacity constraint $0 \leq f(e) \leq c(e) - \ell(e)$, so $\ell(e) \leq f'(e) \leq c(e)$.

f' is a circulation, the f' imbalance at u is $d'(u)$.

Therefore, for (u, v) with $\ell(u, v) > 0$, the increase of flow in (u, v) balances $\ell(u, v)$ units of flow out of u with $\ell(u, v)$ units of flow entering v. Thus the f imbalance at u is $d(u)$.



Main result

Theorem

There exists a circulation in $\mathcal{N}$ iff there exists a circulation in $\mathcal{N}'$. Moreover, if all demands, capacities and lower bounds in $\mathcal{N}$ are integers, and $\mathcal{N}$ admits a circulation, there is a circulation in $\mathcal{N}$ that is integer-valued.

The integer-valued circulation part is a consequence of the integer-value circulation Theorem for f' in G'.

Circulation with demands and lower bounds: main results

Theorem

There is a polynomial time algorithm to solve the circulation with demands and lower bounds problem.

The cost of the algorithm is the same as the cost of the algorithm used for the circulation with demands computation.

Circulation with demands and lower bounds: main results

Theorem

There is a polynomial time algorithm to solve the circulation with demands and lower bounds problem.

The cost of the algorithm is the same as the cost of the algorithm used for the circulation with demands computation.

Theorem

If all capacities, lower bounds, and demands are integers, and there exists a circulation, then we can obtain an integer valued circulation in time $O((D + L)m)$ where L is the sum of al lower bounds.

SURVEY DESIGN problem

Problem: Design a survey among customers of products (KT-7.8)

- Customer i can only be asked about a bought product and must receive a questionnaire for at least c_i such products, those values are determined as function of the purchased products.
- For each product j, we want to collect data from a minimum of p_j customers.
- The c and p values cannot exceed the number of purchased, sold products.

SURVEY DESIGN problem

The input to the problem is:
A set C of n customers and a set P of m products.

- For each customer $i \in C$, a list of purchased products and the two values $c_i \leq c'_i$.
- For each product $j \in P$, two values p_j and p'_j.

Alternatively,

- The information about purchases can be represented as a bipartite graph $G = (C \cup P, E)$, where C is the set of customers and P is the set of products.
- $(i, j) \in E$ means $i \in C$ has purchased product $j \in P$.

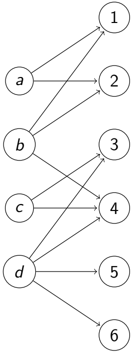
Survey Design: Input

Customers $C = \{a, b, c, d\}$
Products $P = \{1, 2, 3, 4, 5, 6\}$

Customer	Bought	c
a	1,2	1
b	1,2,4	1
c	3,6	1
d	3,4,5,6	2

Prod.	1	2	3	4	5	6
d	1	1	1	1	0	1

G



Survey Design: Circulation with lower bounds formulation

We construct a network $\mathcal{N} = (V', E', c, \ell)$ from G as follows:

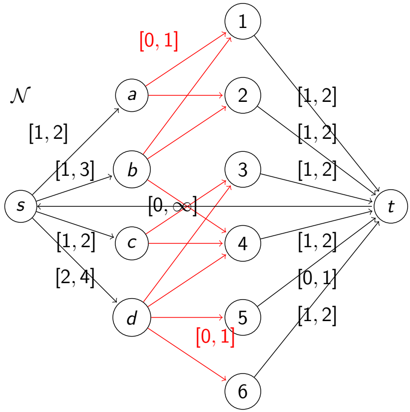
- Nodes: $V' = V \cup \{s, t\}$
- Edges: E' contains E and edges $s \to \{C\}$, $\{P\} \to t$, and (t, s).
- Capacities and lower bounds:
  - $c(t, s) = \infty$ and $\ell(t, s) = 0$
  - For $i \in C$, $\ell(s, i) = c_i$ and $c(s, i) =$ the number of purchased products.
  - For $j \in P$, $\ell(j, t) = p_j$ and $c(j, t) =$ number of customers that purchased j.
  - For $(i, j) \in E$, $c(i, j) = 1$, and $\ell(i, j) = 0$.

SURVEY DESIGN: Circulation with lower bounds formulation

Customers $C = \{a, b, c, c\}$
Products $P = \{1, 2, 3, 4, 5, 6\}$

Customer	Bought	c
a	1,2	1
b	1,2,4	1
c	3,6	1
d	3,4,5,6	2

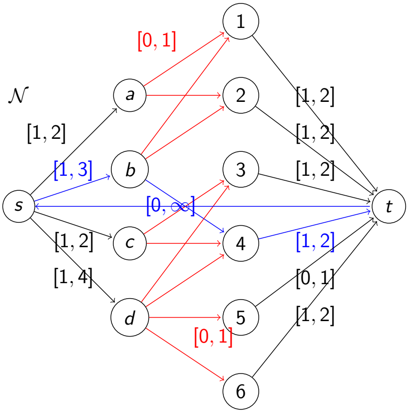
Prod.	1	2	3	4	5	6
d	1	1	1	1	0	1



Survey Design: Circulation interpretation

If f is a circulation in $\mathcal{N}$:

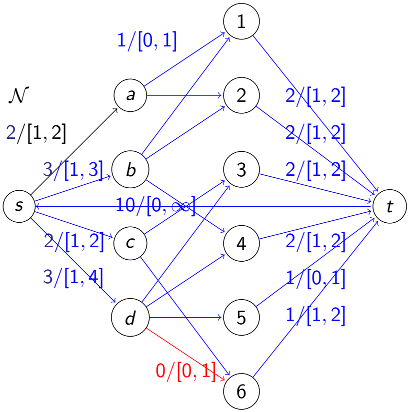
- one unit of flow circulates $s \to i \to j \to t \to s$.
- $f(i,j) = 1$ means ask i about j,
- $f(s,i)$ # products to ask i for opinion,
- $f(j,t) =$ # customers to be asked to review j,
- $f(t,s)$ is the total number of questionnaires.



SURVEY DESIGN: Circulation vs solutions

A solution

- Ask a about $1, 2$.
- Ask b about $1, 2, 4$.
- Ask c about $3, 6$.
- Ask d about $3, 4, 5$.



Main result

Theorem $\mathcal{N}$ has a circulation iff there is a feasible way to design the survey.

Proof if there is a feasible way to design the survey:

- if i is asked about j then $f(i,j) = 1$,
- $f(s,i) =$ number questions asked to i ($\geq c_i$).
- $f(j,t) =$ number of customers who were asked about j ($\geq d_j$),
- $f(t,s) =$ total number of questions.
- easy to verify that f is a circulation in $\mathcal{N}$

If there is an integral circulation in $\mathcal{N}$:

- if $f(i,j) = 1$ then i will be asked about j,
- the constrains will be satisfied by the capacity rule. $\square$

Cost of the algorithm

- $\mathcal{N}$ has $N = n + m + 2$ vertices and $E = n + m + nm$ edges
- $L = \sum_e \ell(e) \leq nm$.
- Obtain $\mathcal{N}$ and extract the information from the circulation has cost $O(nm)$.
- FF analysis, the cost of obtaining a circulaton $O(L(N+M)) = O(n^2m^2)$.
- EK analysis, the cost of obtaining a circulaton $O(NM(N+M)) = O((n+m)n^2m^2)$.
- The algorithm has cost $O(n^2m^2)$.

Joint rounding

Consider a matrix $A = (a_{ij})$ with dimension $n \times n$, where each $a_{ij} \in \mathbb{R}^+ \cup \{0\}$ and furthermore the sum of each row/column is an integer value. We want to round, if possible, each value a_{ij} to $\lfloor a_{ij} \rfloor$ or $\lceil a_{ij} \rceil$ without changing the value of the sum per row/column.

Joint rounding

Consider a matrix $A = (a_{ij})$ with dimension $n \times n$, where each $a_{ij} \in \mathbb{R}^+ \cup \{0\}$ and furthermore the sum of each row/column is an integer value. We want to round, if possible, each value a_{ij} to $\lfloor a_{ij} \rfloor$ or $\lceil a_{ij} \rceil$ without changing the value of the sum per row/column.

$$\begin{pmatrix} 10.9 & 2.5 & 1.3 & 9.3 \\ 3.8 & 9.2 & 2.2 & 11.8 \\ 7.9 & 5.2 & 7.3 & 0.6 \\ 3.4 & 13.1 & 1.2 & 6.3 \end{pmatrix} \rightarrow \begin{pmatrix} 11 & 3 & 1 & 9 \\ 4 & 9 & 2 & 12 \\ 7 & 5 & 8 & 1 \\ 4 & 13 & 2 & 6 \end{pmatrix}$$

Such a rounding is called a joint rounding.

Joint rounding

Note that:

- The elements in A that are integers cannot be modified.
- Let $r_i = \sum_{j=1}^{n}(a_{ij} - \lfloor a_{ij} \rfloor)$ and $c_j = \sum_{i=1}^{n}(a_{ij} - \lfloor a_{ij} \rfloor)$

Joint rounding

Note that:

- The elements in A that are integers cannot be modified.
- Let $r_i = \sum_{j=1}^{n}(a_{ij} - \lfloor a_{ij} \rfloor)$ and $c_j = \sum_{i=1}^{n}(a_{ij} - \lfloor a_{ij} \rfloor)$
- As the rows/columns of A add up to an integer, r_i and c_j are integer values.

$$\begin{pmatrix} 10.9 & 2.5 & 1.3 & 9.3 \\ 3.8 & 9.2 & 2.2 & 11.8 \\ 7.9 & 5.2 & 7.3 & 0.6 \\ 3.4 & 13.1 & 1.2 & 6.3 \end{pmatrix}$$

	1	2	3	4
r	2	2	2	1
c	3	1	1	2

Joint rounding

Note that:

- The elements in A that are integers cannot be modified.
- Let $r_i = \sum_{j=1}^{n}(a_{ij} - \lfloor a_{ij} \rfloor)$ and $c_j = \sum_{i=1}^{n}(a_{ij} - \lfloor a_{ij} \rfloor)$
- As the rows/columns of A add up to an integer, r_i and c_j are integer values.

$$\begin{pmatrix} 10.9 & 2.5 & 1.3 & 9.3 \\ 3.8 & 9.2 & 2.2 & 11.8 \\ 7.9 & 5.2 & 7.3 & 0.6 \\ 3.4 & 13.1 & 1.2 & 6.3 \end{pmatrix}$$

	1	2	3	4
r	2	2	2	1
c	3	1	1	2

- Furthermore, $\sum_i r_i = \sum_j c_j$.

Joint rounding

- In order to solve the problem, we perform a reduction from this problem to a circulation problem.

Joint rounding

- In order to solve the problem, we perform a reduction from this problem to a circulation problem.
- One unit of flow on an edge (i, j) corresponds to rounding $a_{i,j}$ to $\lceil a_{i,j} \rceil$ or rounding $(a_{ij} - \lfloor a_{ij} \rfloor)$ to 1.

Joint rounding

- In order to solve the problem, we perform a reduction from this problem to a circulation problem.
- One unit of flow on an edge (i, j) corresponds to rounding $a_{i,j}$ to $\lceil a_{i,j} \rceil$ or rounding $(a_{ij} - \lfloor a_{ij} \rfloor)$ to 1.
- Zero flow on an edge (i, j) corresponds to rounding $a_{i,j}$ to $\lfloor a_{i,j} \rfloor$ or rounding $(a_{ij} - \lfloor a_{ij} \rfloor)$ to 0.

Joint rounding

- In order to solve the problem, we perform a reduction from this problem to a circulation problem.
- One unit of flow on an edge (i, j) corresponds to rounding $a_{i,j}$ to $\lceil a_{i,j} \rceil$ or rounding $(a_{ij} - \lfloor a_{ij} \rfloor)$ to 1.
- Zero flow on an edge (i, j) corresponds to rounding $a_{i,j}$ to $\lfloor a_{i,j} \rfloor$ or rounding $(a_{ij} - \lfloor a_{ij} \rfloor)$ to 0.
- The total up roundings in row i should be r_i.

Joint rounding

- In order to solve the problem, we perform a reduction from this problem to a circulation problem.
- One unit of flow on an edge (i, j) corresponds to rounding $a_{i,j}$ to $\lceil a_{i,j} \rceil$ or rounding $(a_{ij} - \lfloor a_{ij} \rfloor)$ to 1.
- Zero flow on an edge (i, j) corresponds to rounding $a_{i,j}$ to $\lfloor a_{i,j} \rfloor$ or rounding $(a_{ij} - \lfloor a_{ij} \rfloor)$ to 0.
- The total up roundings in row i should be r_i.
- The total up roundings in column j should be c_j.

Build the network with demands $\mathcal{N} = (V, E, c, d)$ where:

- **Vertices:** $V = \{x_i, y_i | 1 \leq i \leq n\}$. x's vertices represent rows and y vertices columns.
- **Edges:** $E = \{(x_i, y_j) | 1 \leq i, j \leq n \text{ i } a_{i,j} \notin \mathbb{Z}\}$
- **Capacities:** $c(x_i, y_j) = 1$.
- **Demands:** $d(x_i) = -r_i$, $1 \leq i \leq n$, i $d(y_j) = c_j$, $1 \leq j \leq n$.

$\mathcal{N}$ has $O(n)$ vertices and $O(n^2)$ edges.

If there is a joint rounding of A, $\mathcal{N}$ has a circulation with integer values in $\{0, 1\}$.

- Let B a joint rounding of A, we define a new matrix D where
$$d_{ij} = \begin{cases} 1 & \text{if } b_{ij} > a_{ij} \\ 0 & \text{otherwise} \end{cases}$$
- As B is a joint rounding, $\sum_j d_{ij} = r_i$ i $\sum_i d_{ij} = c_i$.
- Therefore, the flow assignment $f(i,j) = d_{ij}$ is a circulation in $\mathcal{N}$.

If $\mathcal{N}$ has a circulation with integer values in $\{0,1\}$, there is a joint rounding of A.

- Let f be a circulation in $\mathcal{N}$,
- we define matrix B as

$$b_{i,j} = \begin{cases} a_{ij} & \text{if } a_{ij} \in \mathbb{Z} \\ \lceil a_{i,j} \rceil & \text{if } a_{ij} \notin \mathbb{Z} \text{ i } f(i,j) = 1 \\ \lfloor a_{i,j} \rfloor & \text{otherwise} \end{cases}$$

- As f is a circulation, $\sum_j b_{ij} = \sum_j a_{ij}$ and $\sum_i b_{ij} = \sum_i a_{ij}$.
- Therefore, B is a joint rounding of A.

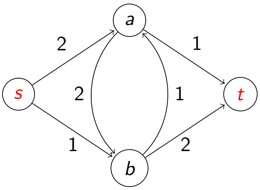
The construction of $\mathcal{N}$ has cost $O(n^2)$.

Ford-Fulkerson algorithm requires $O(D|E|)$, where D is the sum of the positive demands, i.e., $D = \sum r_i = O(n^2)$. As $|E| = O(n^2)$, the total cost is $O(n^4)$.

Flow Network with costs

A network $\mathcal{N} = (V, E, c, \$, s, t)$ is formed by

- a digraph $G = (V, E)$,
- a source vertex $s \in V$
- a sink vertex $t \in V$,
- and edge capacities $c : E \to \mathbb{R}^+$
- and unit flow cost $\$: E \to \mathbb{R}^+$



e	$\$$	e	$\$$
(s, a)	0.2	(s, b)	0.1
(a, b)	0.1	(a, t)	0.1
(b, a)	0.5	(b, t)	0.2

A flow in a network

Given a network $\mathcal{N} = (V, E, c, s, t)$
A Flow is an assignment $f : E \to \mathbb{R}^+ \cup \{0\}$ that follows the Kirchoff's laws:

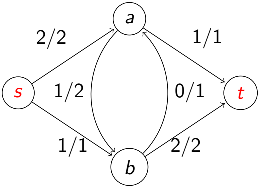
- $\forall (u, v) \in E$, $0 \leq f(u, v) \leq c(u, v)$,
- (Flow conservation) $\forall v \in V - \{s, t\}$, $\sum_{u \in V} f(u, v) = \sum_{z \in V} f(v, z)$

The value of a flow f is

$$|f| = \sum_{v \in V} f(s, v) = f(s, V) = f(V, t).$$

The cost of a flow f is

$$\$(f) = \sum_{e \in E} \$(e) f(e).$$

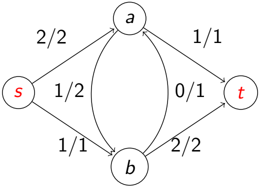


e	\$	e	\$
(s, a)	0.2	(s, b)	0.1
(a, b)	0.1	(a, t)	0.1
(b, a)	0.5	(b, t)	0.2

A flow in a network

Given a network $\mathcal{N} = (V, E, c, s, t)$
A Flow is an assignment $f : E \to \mathbb{R}^+ \cup \{0\}$ that follows the Kirchoff's laws:

- $\forall (u, v) \in E$, $0 \leq f(u, v) \leq c(u, v)$,
- (Flow conservation) $\forall v \in V - \{s, t\}$, $\sum_{u \in V} f(u, v) = \sum_{z \in V} f(v, z)$



The value of a flow f is

$$|f| = \sum_{v \in V} f(s, v) = f(s, V) = f(V, t).$$

e	\$	e	\$
(s, a)	0.2	(s, b)	0.1
(a, b)	0.1	(a, t)	0.1
(b, a)	0.5	(b, t)	0.2

The cost of a flow f is

$$\$(f) = \sum_{e \in E} \$(e) f(e).$$

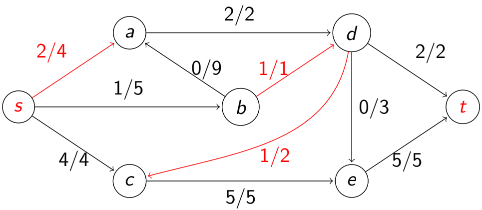
$\$(f) = 0.4 + 0.1 + 0.1 + 0.1 + 0.4 = 1.1$

The Min cost Maximum flow problem

INPUT: A network flow with costs $\mathcal{N} = (V, E, c, \$, s, t,)$
QUESTION: Find a flow of maximum value on $\mathcal{N}$ having minimum cost.

The Min cost Maximum flow problem

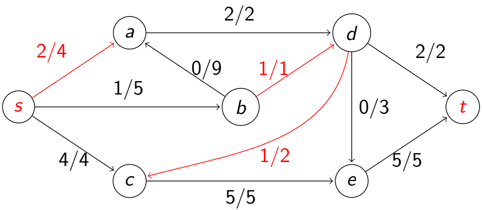
INPUT: A network flow with costs $\mathcal{N} = (V, E, c, \$, s, t,)$
QUESTION: Find a flow of maximum value on $\mathcal{N}$ having minimum cost.



Red edges have unit cost 0.5 and all others unit cost 0.1

The Min cost Maximum flow problem

INPUT: A network flow with costs $\mathcal{N} = (V, E, c, \$, s, t,)$
QUESTION: Find a flow of maximum value on $\mathcal{N}$ having minimum cost.

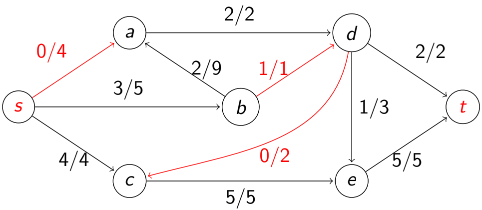


Red edges have unit cost 0.5 and all others unit cost 0.1

$$|f| = 7 \text{ (it is maximum)}$$
$$\$(f) = 1.9 + 2 = 3.9$$

The Min cost Maximum flow problem

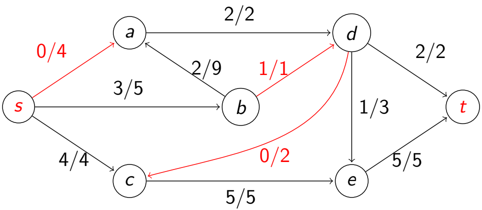
INPUT: A network flow with costs $\mathcal{N} = (V, E, c, \$, s, t,)$
QUESTION: Find a flow of maximum value on $\mathcal{N}$ having minimum cost.



Red edges have unit cost 0.5 and all others unit cost 0.1

The Min cost Maximum flow problem

INPUT: A network flow with costs $\mathcal{N} = (V, E, c, \$, s, t,)$
QUESTION: Find a flow of maximum value on $\mathcal{N}$ having minimum cost.



Red edges have unit cost 0.5 and all others unit cost 0.1

$$|f| = 7 \text{ (it is maximum)}$$
$$\$(f) = 2.4 + 0.5 = 3.4$$

Flows and cycles in the residual graph

Given a network with costs $\mathcal{N} = (V, E, s, \$, t, c)$ together with a flow f on it, the residual graph, $(G_f = (V, E_f, c_f, \$_f)$ is a weighted digraph on the same vertex set and with edge set:

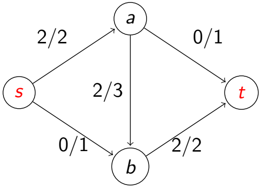
- if $c(u, v) - f(u, v) > 0$, then $(u, v) \in E_f$ and $c_f(u, v) = c(u, v) - f(u, v) > 0$ and $\$(u, v) = \$(u, v)$ (forward edges)
- if $f(u, v) > 0$, then $(v, u) \in E_f$ and $c_f(v, u) = f(u, v)$ and $\$(v, u) = -\$(u, v)$ (backward edges).

Let C be a (simple) cycle in G_f, the bottleneck, $b(C)$, is the minimum (residual) capacity of the edges in P.

Cycle redistribution

Let $\mathcal{N} = (V, E, c, s, \$, t)$ and let f be a flow in $\mathcal{N}$,

Redistribute(C, f)
b=bottleneck (C)
for each $(u, v) \in C$ **do**
  if (u, v) is a forward edge **then**
    Increase $f(u, v)$ by b
  else
    Decrease $f(v, u)$ by b
return f

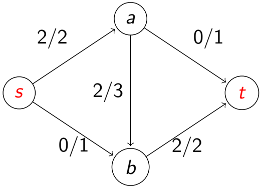


$\mathcal{N}, f$

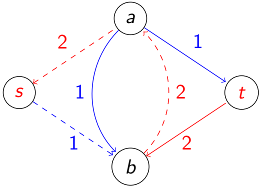
Cycle redistribution

Let $\mathcal{N} = (V, E, c, s, \$, t)$ and let f be a flow in $\mathcal{N}$,

Redistribute(C, f)
b=bottleneck (C)
for each $(u, v) \in C$ **do**
  if (u, v) is a forward edge **then**
    Increase $f(u, v)$ by b
  else
    Decrease $f(v, u)$ by b
return f



$\mathcal{N}, f$

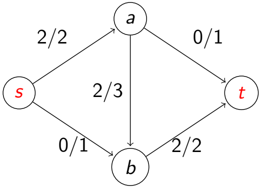


G_f, $P = (s, a, t)$, $b(P) = 1$

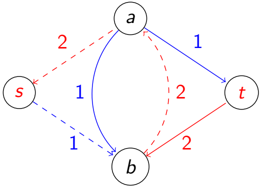
Cycle redistribution

Let $\mathcal{N} = (V, E, c, s, \$, t)$ and let f be a flow in $\mathcal{N}$,

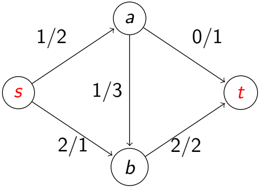
Redistribute(C, f)
b=bottleneck (C)
for each $(u, v) \in C$ **do**
    if (u, v) is a forward edge **then**
        Increase $f(u, v)$ by b
    else
        Decrease $f(v, u)$ by b
return f



$\mathcal{N}, f$



$G_f,\ P = (s, a, t),\ b(P) = 1$



$\mathcal{N}, f'$

Redistribute: flow conservation

Lemma

Let $f' =$Redistribute(C, f), then f' is a flow in $\mathcal{N}$ and $|f'| = |f|$.

Proof.

We have to prove the two flow properties.

- Capacity law
  - Forward edges $(u, v) \in P$, we increase $f(u, v)$ by b, as $b \leq c(u, v) - f(u, v)$ then $f'(u, v) = f(u, v) + b \leq c(u, v)$.
  - Backward edges $(u, v) \in P$ we decrease $f(v, u)$ by b, as $b \leq f(v, u), f'(v, u) = f(u, v) - b \geq 0$.

Redistribute: flow conservation

- Conservation law, $\forall v \in P \setminus \{s, t\}$ let u be the predecessor of v in P and let w be its successor.
- As the cycle is simple only the alterations due to (u, v) and (v, w) can change the flow that goes trough v. As we did for the algorithm Augment, for augmenting path, a case by case analysis shows that the conservation law is preserved.

Redistribute: flow conservation

Now we have to prove that the value of the flow does not change. We have two cases:

Redistribute: flow conservation

Now we have to prove that the value of the flow does not change. We have two cases:

- $s \notin C$. As Redistribute only changes the in/out flow of the vertices in C, the flow out of s is not changed. Therefore, $|f'| = |f|$.

Redistribute: flow conservation

Now we have to prove that the value of the flow does not change. We have two cases:

- $s \notin C$. As Redistribute only changes the in/out flow of the vertices in C, the flow out of s is not changed. Therefore, $|f'| = |f|$.
- $s \in C$. As s has not incoming edges, any cycle in G_f containing s must involve a backward edge entering s and a forward edge out of s. Therefore, we reduce by b the flow in one edge out of s and increase by b another such edges. Again, $|f'| = |f|$.

EndProof

Redistribute: cost

Lemma

Let $f' =$*Redistribute*(C, f), *then* $\$(f') = \$(f) + b\$_f(C)$.

Redistribute: cost

Lemma

Let $f' =$Redistribute(C, f), then $\$(f') = \$(f) + b\,\$_f(C)$.

Proof.

The changes done by Redistribute are (1) subtract b units of flow from the backward edges in C and (2) add b units of flow to the forward edges in C.
According to the definition of $\$_f$ the total change in cost is given by $b\,\$_f(C)$. $\square$

Min cost Max Flow

Theorem

f is a minimum cost maximum flow for $\mathcal{N} = (V, E, c, s, \$, t)$ iff f is a maximum flow in $\mathcal{N} = (V, E, c, s, t)$ and the residual graph G_f has no negative cost cycles.

Proof.

- If there is a negative cost cycle C, f has maximum value but not minimum cost as Redistribute(f, C) will provide a flow with maximum value and smaller cost.

Min cost Max Flow

- If there is no negative cycle,
  - We can compute the shortest distance $\delta(v)$ from s to every node v in G_f according to edge weight $\$_f$.
  - As we have seen when discussing Johnson's algorithm, under the reduced cost $c(v, w) = \$_f(v, w) + \delta(v) - \delta(w)$, all edges in G_f have non-negative costs. This means that any change in f, cannot decrease the reduced cost of f.
  - By the path/cycle invariant of the reduced cost, any change in f cannot decrease its cost.

EndProof.

Cycle-canceling algorithm

Cycle-canceling algorithm

Note that when $f' =$Redistribute(F, C) for some negative cost cycle C in G_f, C does not form in $G_{f'}$.

Cycle-canceling algorithm

Note that when $f' =$Redistribute(F, C) for some negative cost cycle C in G_f, C does not form in $G_{f'}$.

Morton Klein, *A Primal Method for Minimal Cost Flows with Applications to the Assignment and Transportation Problems*, Management Science, INFORMS, vol. 14(3), pages 205-220, November 1967.

Cycle Canceling$(G, s, t, c, \$)$
$f =$ MaxFlow(G, s, t)
Compute G_f
while there is a negative cost cycle C in G_f **do**
  $f =$ **Redistribute**(f, C, G_f)
  Compute G_f
return f

Networks with integer capacities

Using the same arguments as for the Ford Fulkerson algorithm.

Lemma (Integrality invariant)

Let $\mathcal{N} = (V, E, c, \$, s, t)$ where $c : E \to \mathbb{Z}^+$. At every iteration of the Cycle Canceling algorithm, the flow values $f(e)$ are integers.

Theorem (Integrality theorem)

Let $\mathcal{N} = (V, E, c, \$, s, t)$ where $c : E \to \mathbb{Z}^+$. There exists a min cost max-flow f^* such that $f^*(e)$ is an integer, for any $e \in E$.

Networks with integer capacities and costs

Lemma

Let $\mathcal{N} = (V, E, c, \$, s, t)$ where $c, \$: E \to \mathbb{Z}^+$. Let C be the min cut capacity, the Cycle Canceling algorithm terminates after finding at most C augmenting paths and after performing at most $\$(C)$ redistribution calls.

Proof.

The value of the flow increases by ≥ 1 after each augmentation and the cost of a maximum flow after a redistribute call decreases at least by 1. $\square$

Networks with integer capacities and costs: running time

- Computing the MaxFlow f^* takes $O(|f^*|(n+m))$.

Networks with integer capacities and costs: running time

- Computing the MaxFlow f^* takes $O(|f^*|(n+m))$.
- For the second part of the Cycle Canceling algorithm:

Networks with integer capacities and costs: running time

- Computing the MaxFlow f^* takes $O(|f^*|(n+m))$.
- For the second part of the Cycle Canceling algorithm:
  - Constructing G_f, takes $O(m)$ time.
  - $O(nm)$ time to decide if G_f has a negative cycle and if so computing one (use Bellman-Ford algorithm).
  - A call to Redistribute requires $O(m)$ steps

Networks with integer capacities and costs: running time

- Computing the MaxFlow f^* takes $O(|f^*|(n+m))$.
- For the second part of the Cycle Canceling algorithm:
  - Constructing G_f, takes $O(m)$ time.
  - $O(nm)$ time to decide if G_f has a negative cycle and if so computing one (use Bellman-Ford algorithm).
  - A call to Redistribute requires $O(m)$ steps
- Let $C = \max_{e \in E} c(e)$ and $K = \max_{e \in E} \$(e)$, $\$(f^*) \leq CKm$

Networks with integer capacities and costs: running time

- Computing the MaxFlow f^* takes $O(|f^*|(n+m))$.
- For the second part of the Cycle Canceling algorithm:
  - Constructing G_f, takes $O(m)$ time.
  - $O(nm)$ time to decide if G_f has a negative cycle and if so computing one (use Bellman-Ford algorithm).
  - A call to Redistribute requires $O(m)$ steps
- Let $C = \max_{e \in E} c(e)$ and $K = \max_{e \in E} \$(e)$, $\$(f^*) \leq CKm$
- Total running time is $O(|f^*|(n+m) + CKnm)$

Networks with integer capacities and costs: running time

- Computing the MaxFlow f^* takes $O(|f^*|(n+m))$.
- For the second part of the Cycle Canceling algorithm:
  - Constructing G_f, takes $O(m)$ time.
  - $O(nm)$ time to decide if G_f has a negative cycle and if so computing one (use Bellman-Ford algorithm).
  - A call to Redistribute requires $O(m)$ steps
- Let $C = \max_{e \in E} c(e)$ and $K = \max_{e \in E} \$(e)$, $\$(f^*) \leq CKm$
- Total running time is $O(|f^*|(n+m) + CKnm)$
- Thus, we have a pseudo polynomial algorithm.

Networks with integer capacities and costs: running time

- Computing the MaxFlow f^* takes $O(|f^*|(n+m))$.
- For the second part of the Cycle Canceling algorithm:
  - Constructing G_f, takes $O(m)$ time.
  - $O(nm)$ time to decide if G_f has a negative cycle and if so computing one (use Bellman-Ford algorithm).
  - A call to Redistribute requires $O(m)$ steps
- Let $C = \max_{e \in E} c(e)$ and $K = \max_{e \in E} \$(e)$, $\$(f^*) \leq CKm$
- Total running time is $O(|f^*|(n+m) + CKnm)$
- Thus, we have a pseudo polynomial algorithm.

Improving the cost

- Like Ford-Fulkerson algorithm, more careful choices of which cycle to cancel lead to more efficient algorithms.

Improving the cost

- Like Ford-Fulkerson algorithm, more careful choices of which cycle to cancel lead to more efficient algorithms.
- In 1980, Goldberg and Tarjan developed an algorithm that cancels the minimum-mean cycle, the cycle whose average cost per edge is smallest. A clever implementation of the algorithm achieves running time $O(nm^2 \log V)$

Improving the cost

- Like Ford-Fulkerson algorithm, more careful choices of which cycle to cancel lead to more efficient algorithms.
- In 1980, Goldberg and Tarjan developed an algorithm that cancels the minimum-mean cycle, the cycle whose average cost per edge is smallest. A clever implementation of the algorithm achieves running time $O(nm^2 \log V)$
- Combining Edmonds-Karp algorithm with Goldberg and Tarjan's, we get a polynomial time algorithm solving the Min cost Maximum flow problem.

Improving the cost

- Like Ford-Fulkerson algorithm, more careful choices of which cycle to cancel lead to more efficient algorithms.
- In 1980, Goldberg and Tarjan developed an algorithm that cancels the minimum-mean cycle, the cycle whose average cost per edge is smallest. A clever implementation of the algorithm achieves running time $O(nm^2 \log V)$
- Combining Edmonds-Karp algorithm with Goldberg and Tarjan's, we get a polynomial time algorithm solving the Min cost Maximum flow problem.
- The , Chen, Kyng, Liu, Peng, Gutenberg, Sachdeva (2022) $O(m^{1+o(1)})$ solves also the Min cost Maximum flow algorithm.

Min cost circulations

- The Cycle Canceling algorithm can be extended to compute min cost circulations in flow networks with demands and lower bounds, provided a circulation exists.
- The algorithms, have the same asymptotic cost as the ones for the minimum cost maximum flow problem