Network Flows

Zdeněk Hanzálek, Přemysl Šůcha
zdenek.hanzalek@cvut.cz

CTU in Prague

April 7, 2020

Table of contents

Network and Network Flow

What is Network?

By network we mean a 5-tuple (G, l, u, s, t), where G denotes the oriented graph, $u : E(G) \to \mathbb{R}_0^+$ and $l : E(G) \to \mathbb{R}_0^+$ denote the maximum and minimum capacity of the arcs and finally s represents the source node (there is an oriented path from the source node to every other node) while t represents the sink node (there is an oriented path from every other node to the sink node).

Network Flow

$f : E(G) \to \mathbb{R}_0^+$ is the flow if Kirchhoff law $\sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e)$ is valid for every node except s and t.

Feasible Flow

Feasible flow must satisfy $f(e) \in \langle l(e), u(e) \rangle$.
There might be no feasible flow when $l(e) > 0$.

Maximum Flow Problem

Maximum flow

Given a network (G, l, u, s, t). The goal is to find the **feasible flow** f from the source to the sink that maximizes $\sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e)$ (i.e. to send the maximum volume of the flow from s to t).

$\delta^+(s)$ is a set of arcs leaving node s
$\delta^-(s)$ is a set of arcs entering node s (often we do not consider these arcs).

Example - Transportation problem: We wish to transport the maximum amount of goods from s to t. The problem is described by the network where the arc represents the route (pipeline, railway, motorway, etc). The flows on the arcs are assumed to be steady and lossless.
Example constraints:

- $u_i = 10$ - arc i is capable of transporting 10 units maximum
- $l_j = 3$ - arc j must transport at least 3 units
- $l_k = u_k = 20$ - arc k transports exactly 20 units

Example Flow.scheduling: Multiprocessor Scheduling Problem with Preemption, Release Date and Deadline

Consider a $P\left|\text{pmtn}, r_j, \widetilde{d_j}\right| -$ problem - we have n **tasks** which we want to assign to R **identical resources** (processors). Each task has its own **processing time** p_j, **release date** r_j and **deadline** $\widetilde{d_j}$. **Preemption** is allowed (including migration from one resource to another).

Example for 3 parallel identical resources:

task	T_1	T_2	T_3	T_4
p_j	1.5	1.25	2.1	3.6
r_j	3	1	3	5
$\widetilde{d_j}$	5	4	7	9

Goal

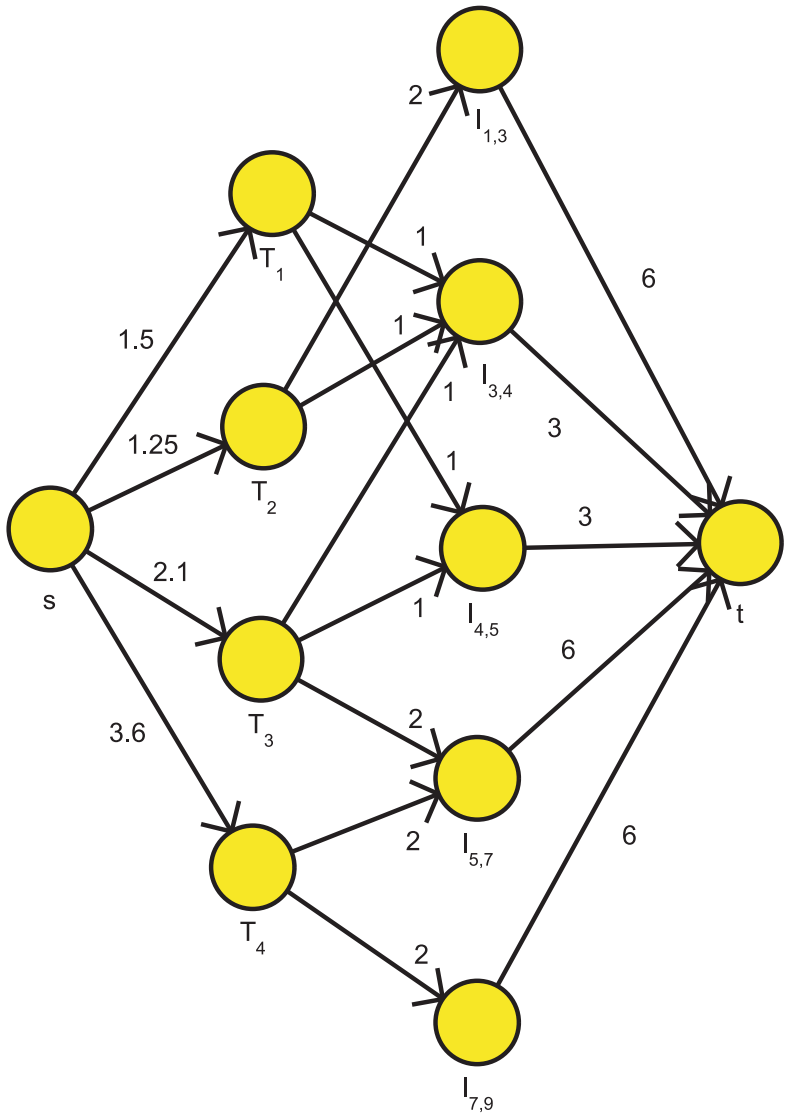
Assign all tasks to processors so that every processor will execute no more than one task at a moment and no task will be executed simultaneously on more than one processor.

Example Flow.scheduling: Multiprocessor Scheduling Problem with Preemption, Release Date and Deadline

3 parallel identical resources

T_j	1	2	3	4
p_j	1.5	1.25	2.1	3.6
r_j	3	1	3	5
$\widetilde{d}_j$	5	4	7	9

1) Nodes $I_{1,3}, I_{3,4}, I_{4,5}, I_{5,7}$ a $I_{7,9}$ represent time interval in which the tasks can be executed (intervals are given by r and $\widetilde{d}$). E.g. $I_{1,3}$ represents time interval $\langle 1, 3)$.
2) The upper bound for the $(T_j, I_{x,y})$ arc is the length of the time interval (i.e. $y - x$), since the tasks are internally sequential.
3) The upper bound for $(I_{x,y}, t)$ is the length of the interval multiplied by the number of resources.



Example Flow.dynamic: Dynamic Flow [Dem02]

Dynamic flow is changing its volume in time. We will show, how to formulate this problem while introducing discrete time and using (static) flow.

For example: let us consider cities $a_1, a_2, \ldots, a_n$ with $q_1, q_2, \ldots, q_n$ cars that should be transported to city a_n in K hours. When the cities a_i, a_j are connected directly, the duration of driving is denoted by d_{ij} and the capacity of the road in number of cars per hour is denoted by u_{ij}. Finally, p_i represents capacity of parking area in city a_i. The objective is to transport the maximum number of cars to city a_n in K hours.

Maximum Flow Problem Formulated as LP

Variable $f(e) \in \mathbb{R}_0^+$ represents flow through arc $e \in E(G)$.

$$\max \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e)$$

$$\text{s.t.} \quad \sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e) \qquad v \in V(G) \setminus \{s, t\}$$
$$l(e) \leq f(e) \leq u(e) \qquad e \in E(G)$$

Note that the following equation is valid for any set A containing source s but not containing sink t :
$\sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e) = \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e)$.

Homework: Prove it while using Kirchhoff's law.

Ford-Fulkerson Algorithm

Pioneers in the field of networks and network flows are L. R. Ford, Jr. and D. R. Fulkerson (on the picture). In 1956, they published a widely known **algorithm for the maximum flow problem**, the Ford-Fulkerson Algorithm.

Ford-Fulkerson Algorithm

It is based on **incremental augmentation** of the flow along an **oriented path** while maintaining the flow's feasibility.
The **path** from source s to sink t **does not respect orientation of arcs**.

Forward arc and backward arc

An arc is called a *forward arc* if it is oriented in the same direction as the path from s to t and a *backward arc* otherwise.

Augmenting path

An augmenting path for flow f is a path from s to t with:

- $f(e) < u(e)$ for all forward arcs on the path ... $f(e)$ can be increased
- $f(e) > l(e)$ for all backward arcs on the path ... $f(e)$ can be decrsed

Capacity of an augmenting path

Capacity $\gamma = min\{min_{e \text{ is forward}} u(e) - f(e), min_{e \text{ is backward}} f(e) - l(e)\}$ is the biggest possible increase of the flow on the augmenting path.

Ford-Fulkerson Algorithm

Input: Network (G, l, u, s, t).
Output: Maximum feasible flow f from s to t.

1. Find the feasible flow $f(e)$ for all $e \in E(G)$.
2. Find an augmenting path P. If none exists then stop.
3. Compute γ, the capacity of an augmenting path P. Augment the flow from s to t and go to 2.

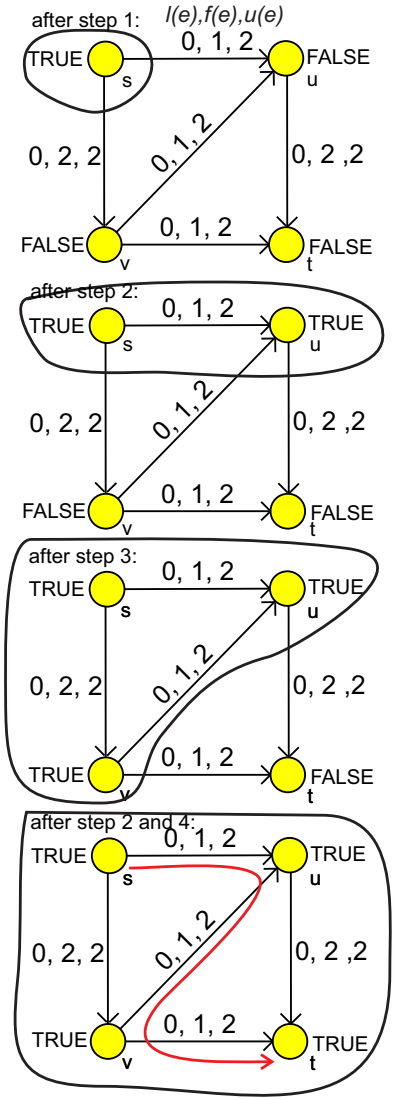
Increase of the flow by γ on forward arcs and decrease of the flow by γ on backward arcs - this preserves feasibility of the flow and Kirchhof's law moreover the flow is augmented by γ.
This augmenting path can't be used again since the flow along this path is the highest possible.

The flow from s to t is the **maximum** if and only if there is no augmenting path.

Finding the Augmenting Path (Labeling Procedure) for Ford-Fulkerson Algorithm

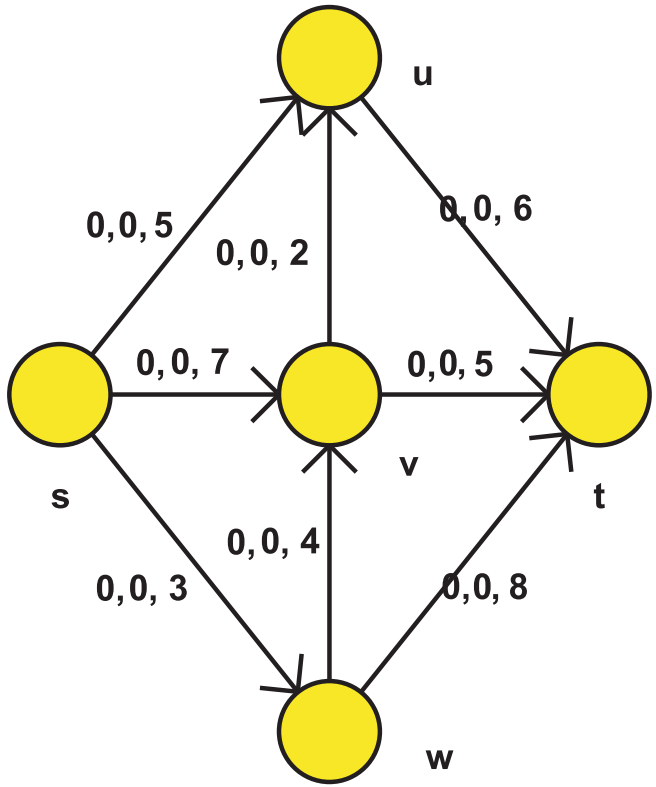
Input: Network (G, l, u, s, t), feasible flow f.
Output: Augmenting path P.

1. Label $m_s = TRUE$, $m_v = FALSE$ $\forall v \in V(G) \setminus s$.
2. **If** there exists $e \in E(G)$ (where e is the edge from v_i to v_j) that satisfies $m_i = TRUE$, $m_j = FALSE$ and $f(e) < u(e)$ **Then** $m_j = TRUE$.
3. **If** there exists $e \in E(G)$ (where e is the edge from v_i to v_j) that satisfies $m_i = FALSE$, $m_j = TRUE$ and $f(e) > l(e)$ **Then** $m_i = TRUE$.
4. If t is reached, then the search stops as we have found the augmenting path P. If it is not possible to mark another node, then P does not exists. In other cases go to step 2.



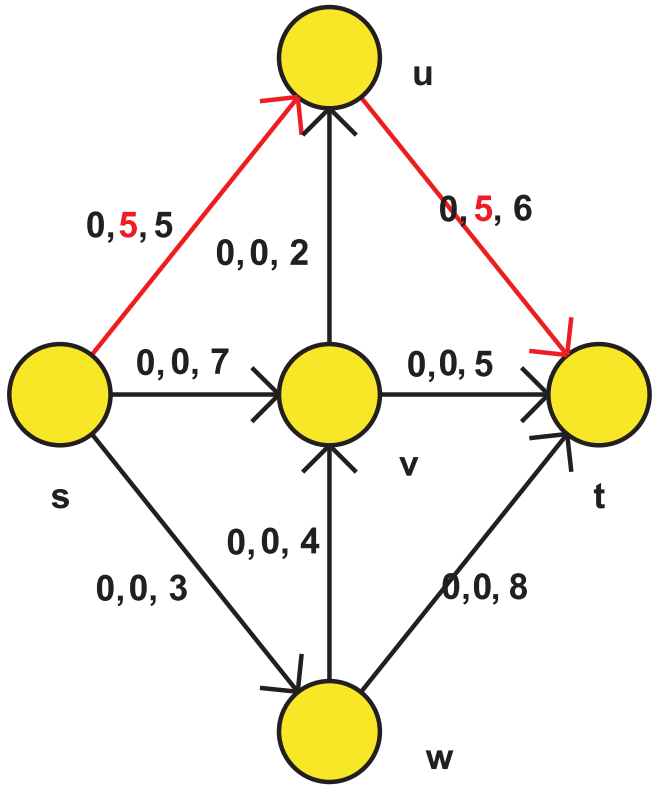
Ford-Fulkerson Algorithm
Simple Example

arcs are labeled by $l(e), f(e), u(e)$



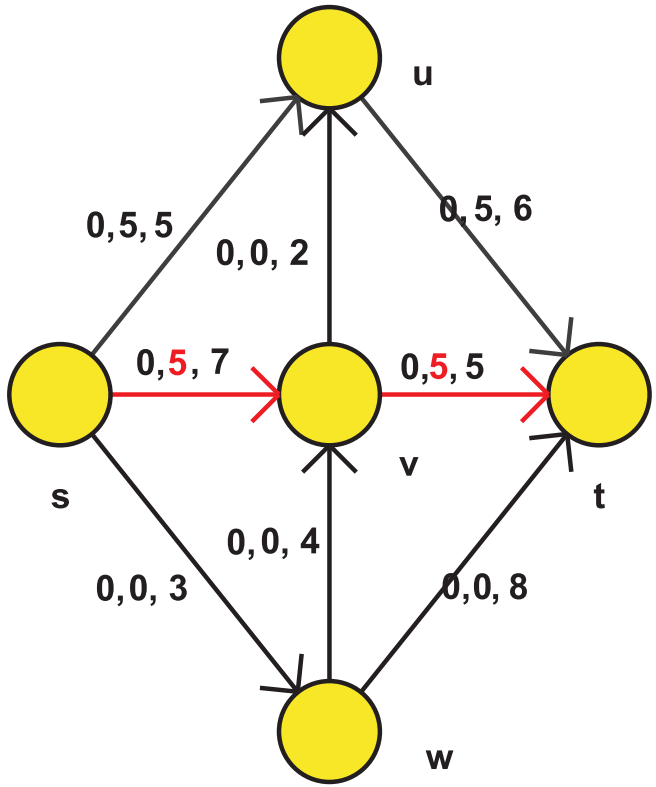
Ford-Fulkerson Algorithm
Simple Example

arcs are labeled by $l(e), f(e), u(e)$



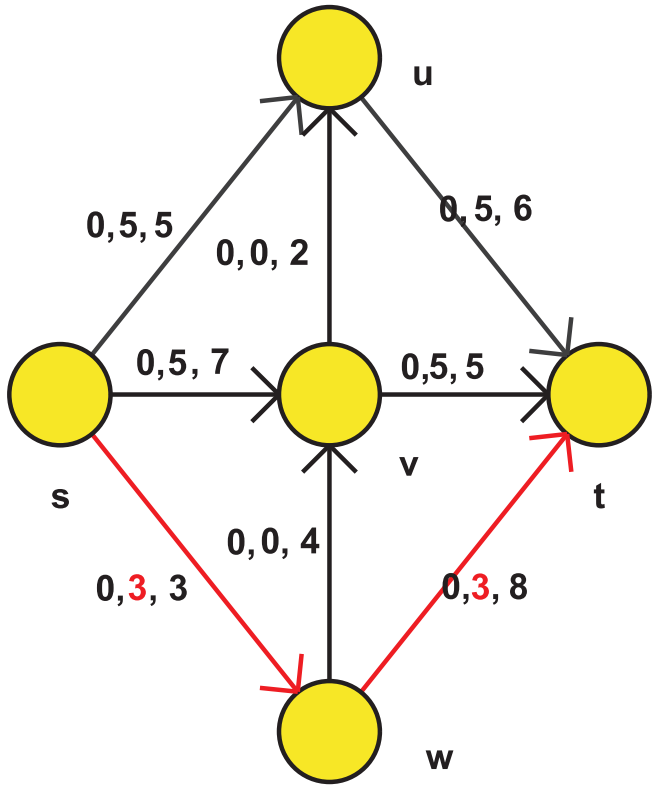
Ford-Fulkerson Algorithm
Simple Example

arcs are labeled by $l(e), f(e), u(e)$



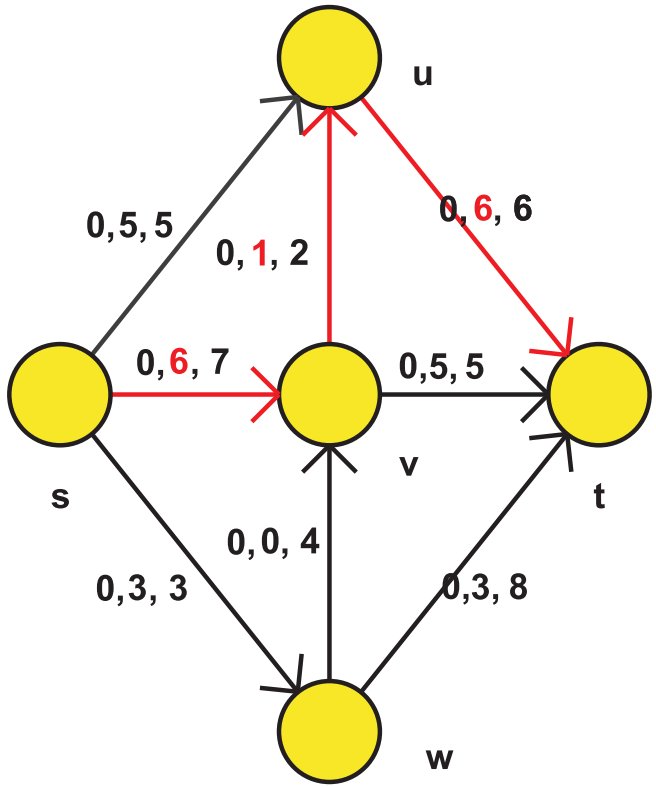
Ford-Fulkerson Algorithm
Simple Example

arcs are labeled by $l(e), f(e), u(e)$



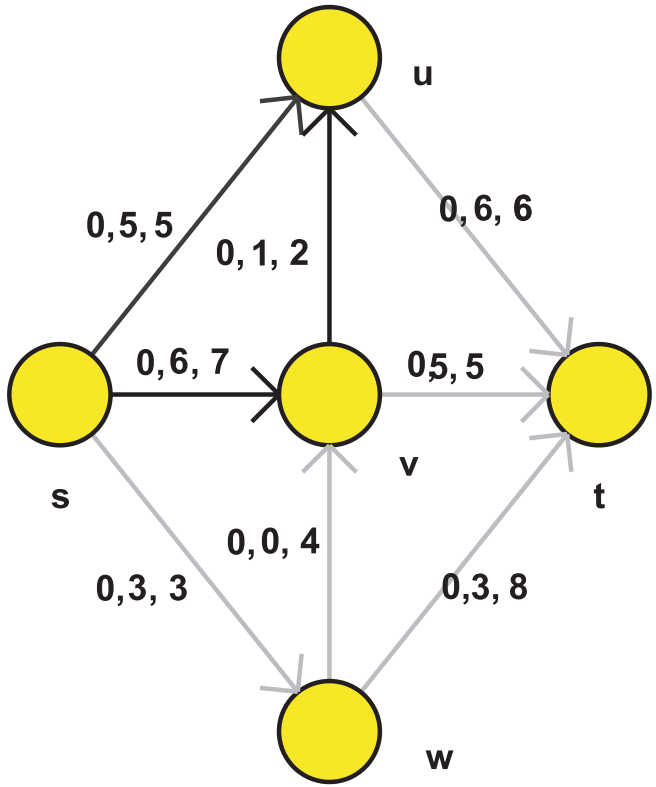
Ford-Fulkerson Algorithm
Simple Example

arcs are labeled by $l(e), f(e), u(e)$



Ford-Fulkerson Algorithm
Simple Example

arcs are labeled by $l(e), f(e), u(e)$

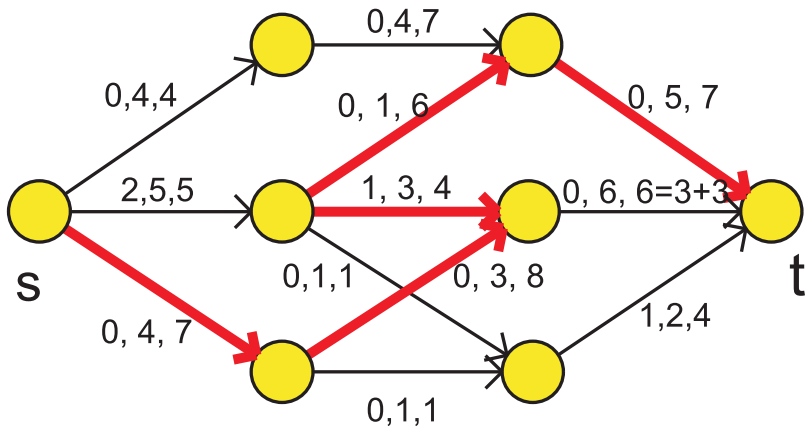


set A=$\{s, u, v\}$ determines the **minimum capacity cut**

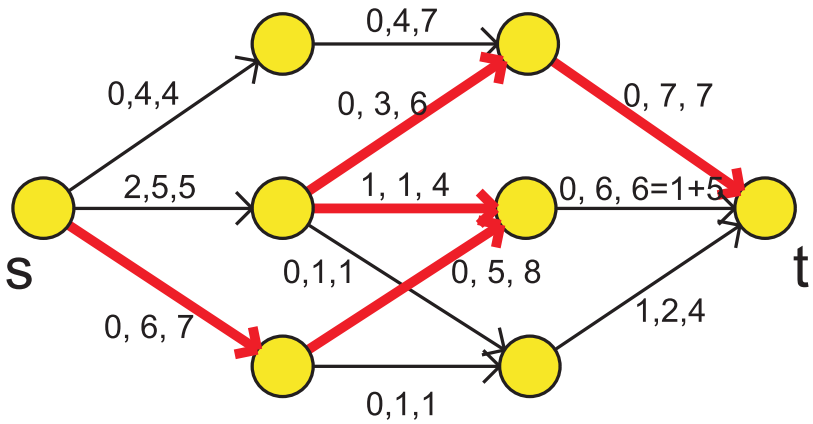
Ford-Fulkerson Algorithm
Example with Backward Edge

This example illustrates that **we can't omit the backward edges** when creating the augmenting path. Otherwise we are not able to obtain the maximum flow (right) from the initial flow (left).
The arcs are labeled by: $l(e), f(e), u(e)$.

Capacity of the augmenting path is equal to 2.



Final flow is the maximal one.
Find a minimum capacity cut.



Minimum Cut Problem

Cut

The cut in G is an edge set $\delta(A) = \delta^+(A) \cup \delta^-(A)$ with $s \in A$ and $t \in V(G) \setminus A$ (i.e. the cut separates nodes s and t). The **minimum cut** is the cut of minimum capacity $C(A) = \sum_{e \in \delta^+(A)} u(e) - \sum_{e \in \delta^-(A)} l(e)$.

Ford and Fulkerson [1956]

The value of the maximum flow from s to t is equal to the capacity of the **minimum cut**, i.e.,
$\sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e) = \sum_{e \in \delta^+(A)} u(e) - \sum_{e \in \delta^-(A)} l(e)$.

This property follows from LP duality.
When the labeling procedure stops, since there is no augmenting path, the minimum cut is given by the labeled vertices (the minimum cut is equal to the set of edges that do not allow further labeling). Therefore, $f(e) = u(e)$ holds for all $e \in \delta^+(A)$ and $f(e) = l(e)$ holds for all $e \in \delta^-(A)$.

Example Flow.cut.blocking: Blocking of Gasoline Supply

A tank is located at one node t in gasoline pipeline directed graph $G(E, V)$. Gasoline supply nodes are denoted by the set $S \subset V$ such that $t \notin S$. Let pipe e has minimum throughput $l(e)$ and maximum throughput $u(e)$. The adversary might decrease the maximum throughput by $\alpha(e) \in \langle l(e), u(e) \rangle$ if he/she makes effort of volume $k\alpha(e)$, where k is a positive constant.

a) Consider $l_{ij} = 0$ and determine the minimal effort required to isolate the tank from a gasoline.

b) How the problem changes when we consider non-zero minimum throughput?

Integrality

Integral Flow Theorem (Dantzig and Fulkerson [1956])

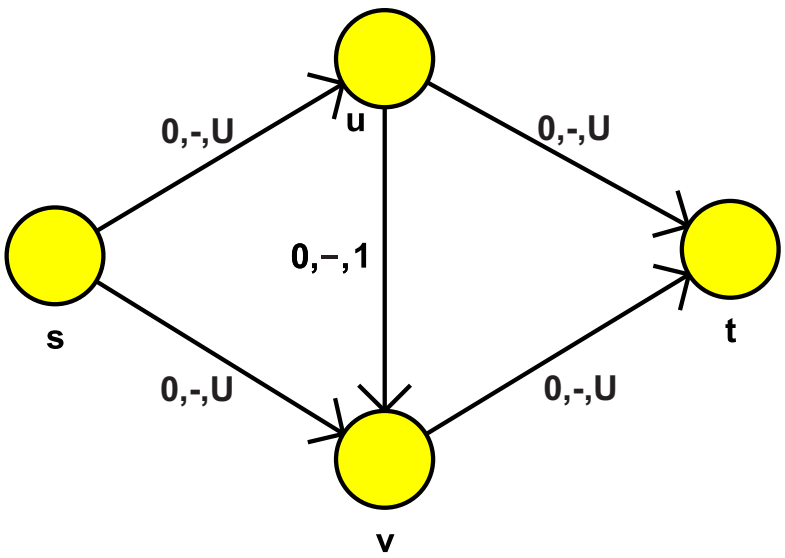
If the capacities of the network are integers, and there exists a feasible flow, then there exists **an integer-valued maximum flow**.

This follows from total unimodularity of the incidence matrix of a digraph G, which is matrix $\mathbf{A}$ in LP formulation $\mathbf{A} \cdot x \leq b$.

Alternatively we can prove it as follows:
If all capacities are integers, γ in 3) of the Ford-Fulkerson algorithm is always an integer. Since there is a maximum flow of finite value, the algorithm terminates after a finite number of steps.

Ford-Fulkerson Algorithm - Time Complexity

To find an augmenting path takes $O(|E|)$ time.



For integer capacities, each iteration increases the flow by at least 1.
The maximum number of iterations is equal to the volume of maximum flow F and it is equal to the capacity of the minimum cut bounded by $|E| \cdot U$ where U is the highest edge capacity, giving alg. complexity of $O(|E| \cdot F)$ and $O(|E|^2 \cdot U)$ respectively.

For non-integer capacities and non-integer flow the algorithm **might not terminate at all** [Korte Vygen] Chapter Flows,Exercise 2.

Edmonds and Karp [1972])

When choosing the augmenting path, if we always choose the **shortest one** (e.g., by Breadth First Search), time complexity is $O\left(|E|^2 \cdot |V|\right)$.

Proof: [Korte Vygen] Theorem 8.14.

Example Flow.lower.representatives: Distinct Representatives - Existence of Lower Bound [AMO93]

Assignment problem with an additional constraint.
Let us have n residents of a town, each of them is a member of at least one club $k_1, \ldots, k_l$ and belongs to exactly one age group $p_1, \ldots, p_r$. Each club must nominate one of its members to the town's governing council so that the number of council members belonging to the age group is constrained by its minimum and maximum. One resident represents at most one club. Find a feasible assignment of the representatives or prove that it does not exist. Formulate as the Maximum Flow Problem.

Example Flow.lower.rounding: [AMO93] Matrix Rounding Problem - Existence of Lower Bound

This application is concerned with **consistent rounding of the elements, row sums, and column sums** of a matrix. We are given a 3×3 matrix of real numbers x_{ij} with row sums vector r and column sums vector c. We can round any real number $a \in \{x_{11} \dots x_{33}, r_1 \dots r_3, c_1 \dots c_3\}$ to the next smaller integer $\lfloor a \rfloor$ or to the next larger integer $\lceil a \rceil$, and the decision is completely up to us. The problem requires that we round all matrix elements and the following constraints hold:

- the sum of the rounded elements in each row is equal to the rounded row sum r_i
- the sum of the rounded elements in each column is equal to the rounded column sum c_i

We refer to such rounding as a *consistent rounding*.
The objective is to maximize the sum of matrix entries (due to the constraint it is equal to the sum of the row sums and at the same time to the sum of the column sums).

Feasible Flow with Balances - Decision Problem

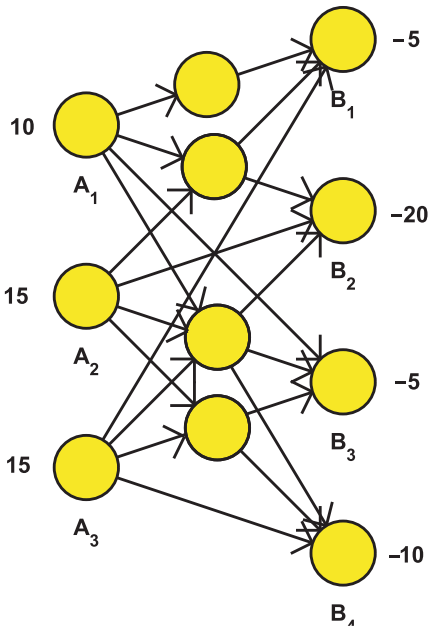
We assume several sources and several sinks.
Later we show its polynomial reduction to the Maximum Flow problem (with one source and one sink).

Feasible flow with balances

- **Instance**: Let (G, l, u, b) be a network where G is a digraph with upper bounds $u : E(G) \to \mathbb{R}_0^+$, $l : E(G) \to \mathbb{R}_0^+$ and with:
  - balance $b : V(G) \to \mathbb{R}$ that represents the supply/consumption of the nodes and satisfies $\sum_{v \in V(G)} b(v) = 0$.
- **Goal**: Decide if there exists a feasible flow f which satisfies $\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = b(v)$ for all $v \in G(V)$.

Example - Transport Problem

There are **suppliers** (represented by nodes with $b(v) > 0$) and **consumers** (represented by nodes with $b(v) < 0$) of a **product**. Our goal is to decide whether it is possible to transport all the product from the suppliers to the consumers through the network with upper bounds u. The problem is described by a network (or graph) where the edges represent pipelines, highways or railways of some transportation capacity.



Goal

The goal is to decide whether it is possible to transport all the product from suppliers A to consumers B through the network with capacities u.

Feasible Flow with Balances - Reduction to Maximum Flow

We transform this problem to the **maximum flow problem**:

1. We add a new node s called the source and add edges (s, v) with the lower bound $l_v = 0$ and the upper bound $u_v = b(v)$ for every node that satisfies $b(v) > 0$
2. We add a new node t called the sink and add edges (v, t) with the lower bound $l_v = 0$ and the upper bound $u_v = -b(v)$ for every node that satisfies $b(v) < 0$
3. We solve the maximum flow problem
4. If the maximum flow saturates all edges leaving s and/or entering t, then the answer to the feasible flow decision problem is YES.

How to Find an Initial Feasible Flow for Ford-Fulkerson Algorithm?

If $\forall e \in E(G)$; $l(e) = 0$ - easy solution - we use zero flow which satisfies Kirchhoff's law.
If $\exists e \in E(G)$; $l(e) > 0$, we transform the feasible flow problem to the **Feasible Flow with Balances and zero lower bounds** as follows:
1) We transform the maximum flow problem (with non-zero lower bounds) to a circulation problem by **adding an arc from t to s of infinite capacity**. Consequently, the Kirchhoff's law applies to nodes s and t. Therefore, a feasible **circulation must satisfy**:

$$l(e) \leq f(e) \leq u(e) \qquad e \in E(G)$$
$$\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = 0 \qquad v \in V(G)$$

How to find an initial feasible flow for Ford-Fulkerson algorithm?

2) Substituting $f(e) = f(e)' + l(e)$, we obtain the transformed problem:

$$0 \leq f(e)' \leq u(e) - l(e) \qquad e \in E(G)$$

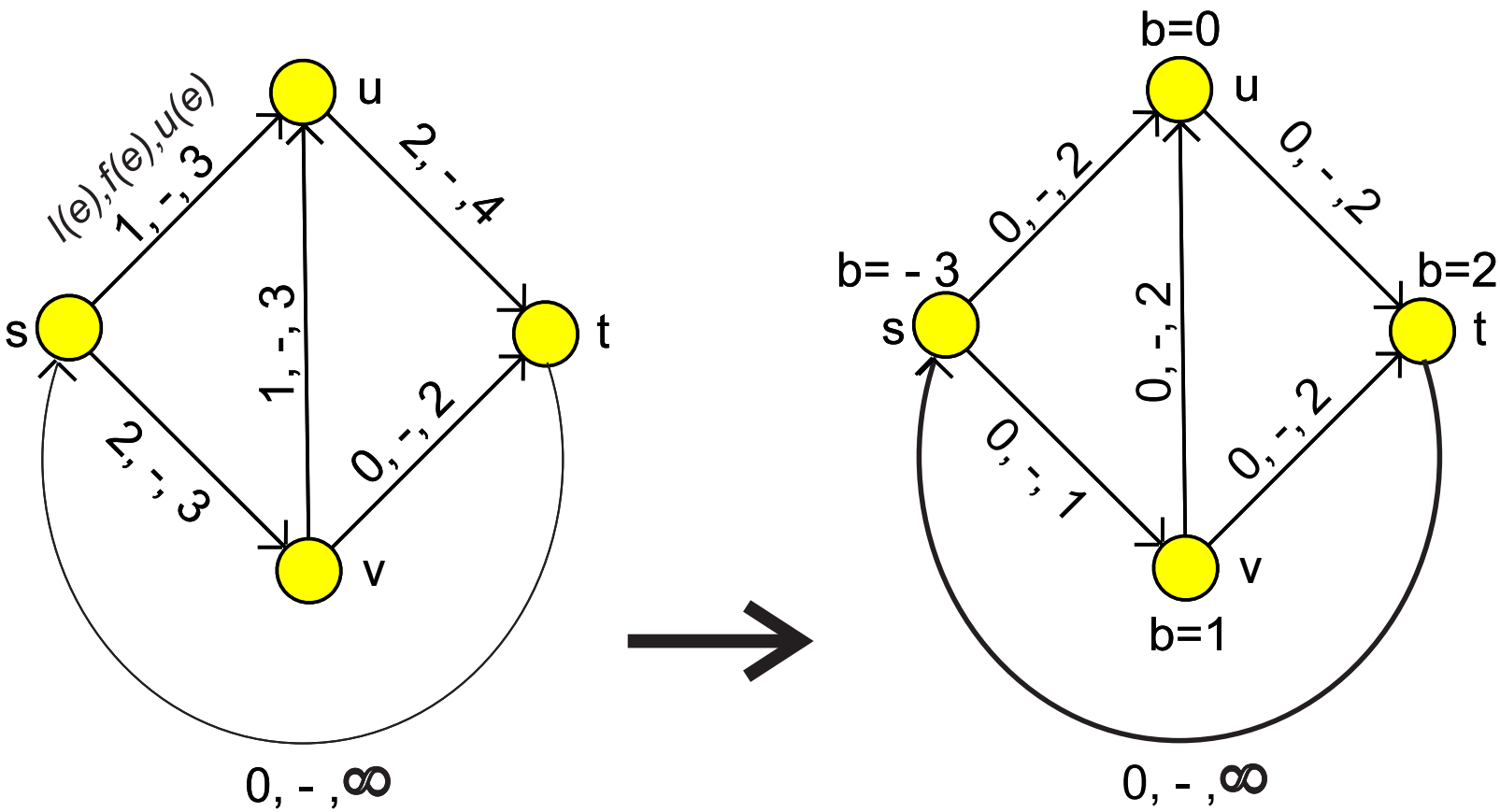
$$\sum_{e \in \delta^+(v)} f(e)' - \sum_{e \in \delta^-(v)} f(e)' = \underbrace{\sum_{e \in \delta^-(v)} l(e) - \sum_{e \in \delta^+(v)} l(e)}_{b(v)} \qquad v \in V(G)$$

3) This is a **Feasible Flow with Balances and zero lower bounds** because $\sum_{v \in V(G)} b(v) = 0$ (notice that $l(e)$ appears twice in summation, once with a positive and once with a negative sign).

4) While solving this decision problem (i.e. adding s', t' and solving the maximum flow problem with zero lower bounds) we obtain the initial feasible circulation/flow or decide that it does not exist.

Conclusion: finding of the **initial flow with nonzero lower bounds** can be transformed to the **Feasible Flow with Balances and zero lower bounds** which can be transformed to the **Maximum Flow with zero lower bounds**.

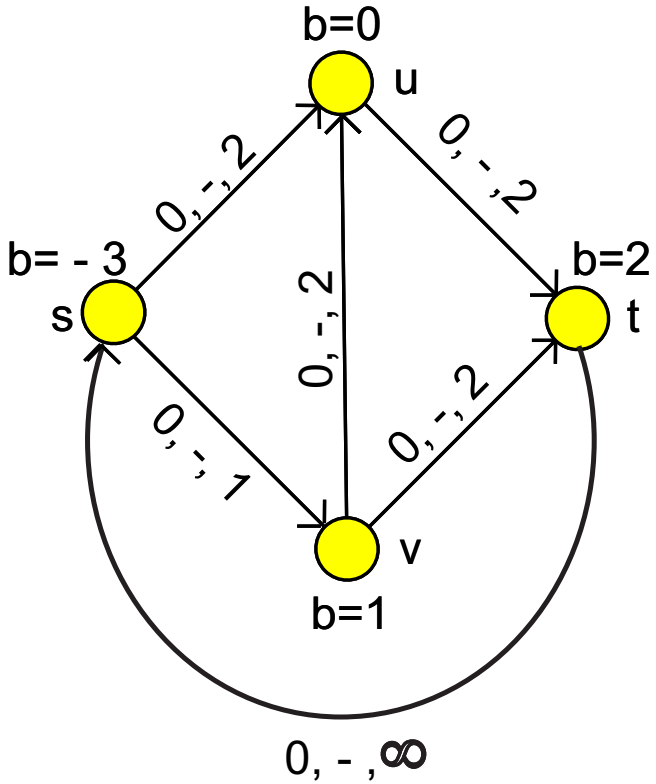
Example: Transformation of MaxFlow with nonzero lower bounds to FeasFlow with Balances and zero lower bounds



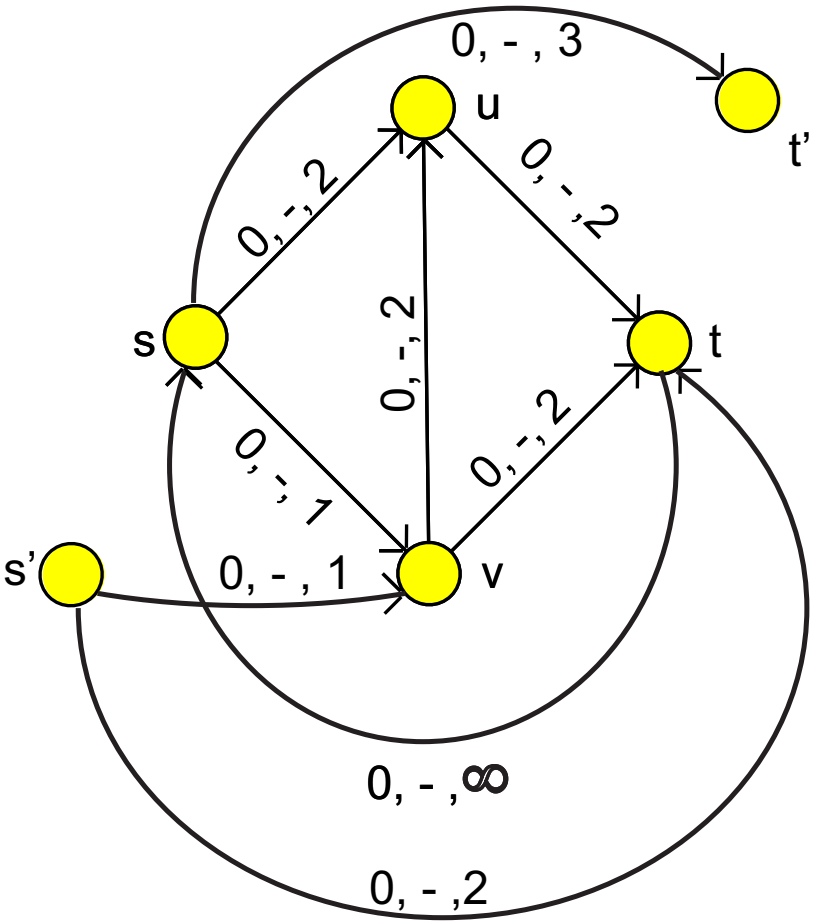
Maximu Flow
with nonzero lower bounds

Feasible Flow with Balances
and zero lower bouds

Example: Transformation of Feasible Flow with Balances to Maximum Flow, both with zero lower bounds



Feasible Flow with Balances and zero lower bouds



Maximum Flow with zero lower bounds

Minimum Cost Flow

Extension of the Maximum flow problem - we consider the edge costs and the supply/consumption of the nodes.

Minimum Cost Flow

- **Instance**: 5-tuple (G, l, u, c, b) where G is a digraph, $u : E(G) \to \mathbb{R}_0^+$ represents the upper and $l : E(G) \to \mathbb{R}_0^+$ the lower bounds and:
  - **cost of arcs** $c : E(G) \to \mathbb{R}$
  - balance $b : V(G) \to \mathbb{R}$ that represents the supply/consumption of the nodes and satisfies $\sum_{v \in V(G)} b(v) = 0$.
- **Goal**: Find the feasible flow f that minimizes $\sum_{e \in E(G)} f(e) \cdot c(e)$ (we want to transport the flow through the network at the lowest possible cost) and satisfies $\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = b(v)$ for all $v \in G(V)$, or decide that it does not exist.

Minimum Cost Flow - LP Formulation

Variable $f(e) \in \mathbb{R}_0^+$ represents the flow on edge $e \in E(G)$.

$$\min \sum_{e \in E(G)} c(e) \cdot f(e)$$

$$\text{s.t.} \quad \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = b(v) \qquad v \in V(G)$$
$$l(e) \leq f(e) \leq u(e) \qquad e \in E(G)$$

The **Maximum Flow problem** can be polynomially reduced to the Minimum Cost Flow problem:

- Add an edge from t to s with the upper bound equal to ∞ and the cost -1.
- Set the cost of every other edge to 0.
- Set $b(v) = 0$ for all nodes including s and t.
- Minimum cost circulation (it will be negative) maximizes the flow on the added edge.

Minimum Cost Flow is quite general problem

The **Shortest Path from s to t** can be reduced to the Min-Cost Flow:

- Use LP formulation of Min-Cost Flow.
- Set $b(s) = 1$ and $b(t) = -1$. Set $b(v) = 0 \ \ \forall v \in V \setminus \{s, t\}$.
- Set $l(e) = 0$ and $u(e) = \infty \ \ \forall e \in E$.
- You obtain (primal) LP formulation of the Shortest Path Problem (follow example of totally unimodular matrix A in ILP lecture).

The **Chinese Postman Problem** (A postman has to deliver the mail within his district given by strongly connected directed graph. To do this, he must start at the post office, walk along each street at least once, and finally return to the post office. The problem is to find a postman's tour of minimum length.) can be reduced to the Min-Cost Flow:

- Set $b(v) = 0 \ \ \forall v \in V$.
- Set $l(e) = 1$ and $u(e) = \infty \ \ \forall e \in E$.

There is a postman's tour using each edge exactly once (i.e. Eulerian walk) iff every vertex has equal indegree and outdegree (i.e. Eulerian digraph).

Minimum Cost Flow - Cycle Canceling Algorithm

Input: Network (G, l, u, b, c).
Output: Minimum cost flow f.

1. Find feasible flow f while solving Feasible Flow with Balances in (G, l, u, b).
2. Build residual graph G_f with respect to f. Find a negative-cost cycle C in residual graph G_f. If none exists then stop.
3. Compute $\gamma = \min_{e \in E(C)} u_f(e)$. Augment f along C by γ. Go to 2.

The negative-cost cycle C in residual graph can be found as follows:

1. Consider residual graph (G_f, u_f, c_f) where all edges have $u_f(e) > 0$
2. Add node s and connect it to every $v \in V(G_f)$ by edge $e(s, v)$ with cost $c_f(e) = 0$.
3. Use, for example, Bellman-Ford algorithm to detect a negative-cost cycle (with respect to cost c_f).
4. Recover negative-cost cycle C or state that it does not exist.

Residual Graph

The residual graph shows where an extra capacity might be found. The residual graph G_f is constructed from network (G, l, u, b, c) and specific feasible flow f as follows:

$$G_f = (V, E_f, u_f, c_f)$$

$$\bar{E} = E \cup \{(j,i) : (i,j) \in E\}$$

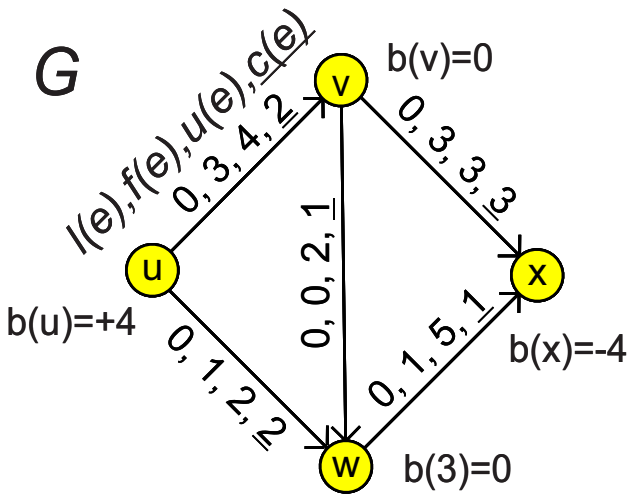
$$u_f(i,j) = u(i,j) - f(i,j) \quad \forall (i,j) \in E$$

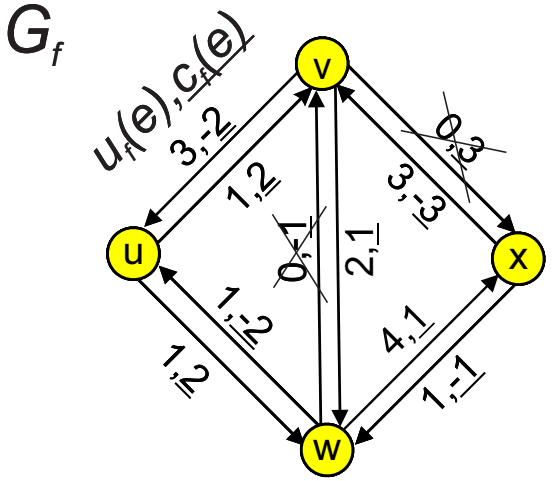
$$u_f(j,i) = f(i,j) - l(i,j) \quad \forall (i,j) \in E$$

$$E_f = \left\{(i,j) \in \bar{E} : u_f(i,j) > 0\right\}$$

$$c_f(i,j) = c(i,j) \quad \forall (i,j) \in E$$

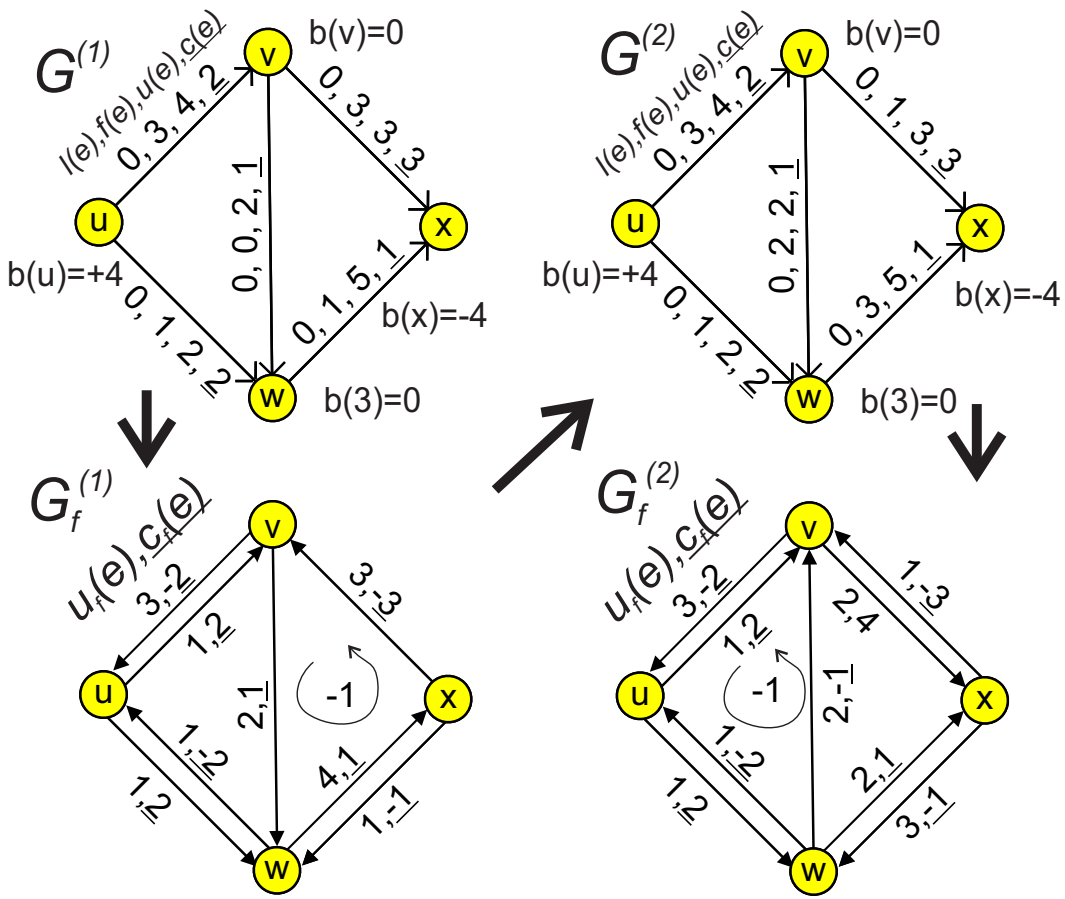
$$c_f(j,i) = -c(i,j) \quad \forall (i,j) \in E$$





Example: Cycle Canceling Algorithm with Residual Graph

- $G^{(1)}$ with a feasible flow in the first iteration.
- $G_f^{(1)}$ with a negative-cost cycle $C = (v, w, x)$, $\gamma = \min_{e \in E(C)} u_f(e) = 2$, min cost criterion will change by $-1 \cdot 2$.
- $G^{(2)}$ with the feasible flow in the second iteration.
- $G_f^{(2)}$ with a negative-cost cycle $C = (v, u, w)$, $\gamma = \min_{e \in E(C)} u_f(e) = 1$, min cost criterion will change by $-1 \cdot 1$, new flow will be feasible.



Cycle Canceling Algorithm - Complexity and its improvements

Analysis of the Cycle Canceling Algorithm:
Let $U = \max_{(i,j)\in E} u(i,j)$ and $C = \max_{(i,j)\in E} |c(i,j)|$.
For any feasible flow f its cost $c(f)$ is bounded as:
$-|E| \cdot C \cdot U \leq c(f) \leq |E| \cdot C \cdot U$
Each iteration decreases the objective by at least 1.
Conclusion: there are at most $2|E| \cdot C \cdot U$ iterations.
The Bellman-Ford complexity is $O(|V| \cdot |E|)$.
Overall time complexity is $O\left(|V| \cdot |E|^2 \cdot C \cdot U\right)$.

Idea for improvement:
Find the most negative cycle. But this problem is NP-hard.

Solution by a Minimum Mean Cycle Canceling Algorithm:
Find a circuit C whose mean weight $\frac{c(E(C))}{|E(C)|}$ is minimum in $O(|V| \cdot |E|)$.
Minimum Mean Cycle Canceling Algorithm runs in $O\left(|V|^2 \cdot |E|^3 \cdot log|V|\right)$.

Minimum Cost Multicommodity Flow

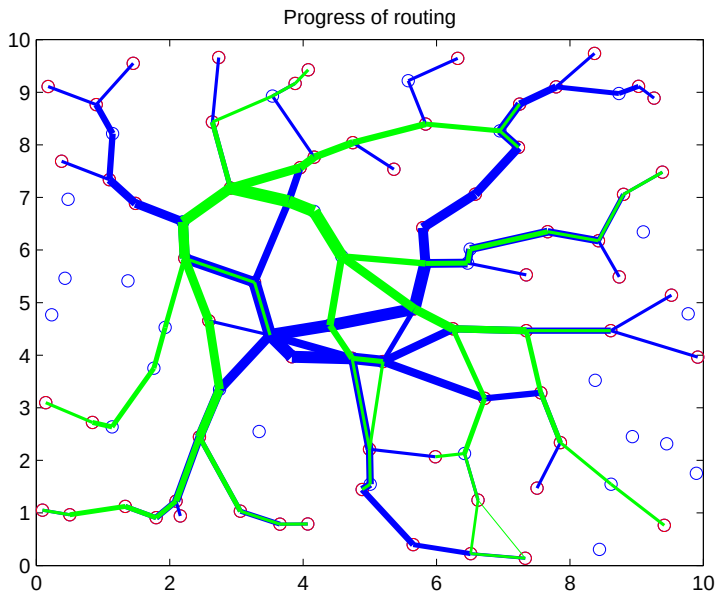
So far, we have assumed to be transporting just one commodity.
Let M be the **commodity set** transported through the network.
Every commodity has several sources and several sinks.
Variable $f^m(e) \in \mathbb{R}_0^+$ is the flow of commodity $m \in M$ along edge $e \in E(G)$.

Example: sensor network with two commodities and one sink for every commodity:

- source nodes are measuring **temperature(green)** and/or **humidity(blue)** and sending the data to one concentrator (sink) for temperature and one for humidity
- flow (amount of data per time unit)

Communication links:

- capacity (amount of data per time unit)



Minimum Cost Multicommodity Flow

Minimum cost multicommodity flow

- **Instance**: 5-tuple $(G, l, u, c, b^1 \dots b^m \dots b^{|M|})$ where G is a digraph with upper bounds $u : E(G) \to \mathbb{R}_0^+$, lower bounds $l : E(G) \to \mathbb{R}_0^+$ and costs $c : E(G) \to \mathbb{R}$ and with:
  - vectors $b^m : V(G) \to \mathbb{R}$ that express (**supply/consumption**) of nodes by commodity m. $\sum_{v \in V(G)} b^m(v) = 0$ **for all commodities** $m \in M$.
- **Goal**: Find the feasible flow f whose cost $\sum_{e \in E(G)} \sum_{m \in M} f^m(e) \cdot c(e)$ is minimal (we want to transport the flow as cheap as possible) or decide that such a flow does not exist. Feasible flow satisfies $\sum_{e \in \delta^+(v)} f^m(e) - \sum_{e \in \delta^-(v)} f^m(e) = b^m(v)$ for all $v \in G(V)$ and all $m \in M$.

Minimum Cost Multicommodity Flow
LP Formulation

Variable $f^m(e) \in \mathbb{R}_0^+$ represents the flow of commodity $m \in M$ along edge $e \in E(G)$.

$$\min \sum_{e \in E(G)} \sum_{m \in M} f^m(e) \cdot c(e)$$

$$\text{s.t.} \quad \sum_{e \in \delta^+(v)} f^m(e) - \sum_{e \in \delta^-(v)} f^m(e) = b^m(v) \qquad v \in V(G), m \in M$$

$$l(e) \leq \sum_{m \in M} f^m(e) \leq u(e) \qquad e \in E(G)$$

- 1st Kirchhoff's law is satisfied in every node for **every commodity**.
- Multicommodity flow can be sloved by LP - **in polynomial time**.
- Integer-valued flow is not assured since matrix A in LP **is not totally unimodular** (see $l(e) \leq \sum_{m \in M} f^m(e) \leq u(e)$)
- Practical experience: ILP formulation which guarantees integer-valued solution can be solved even for big instances in acceptable time.

Matching - Introduction

Matching is the set of arcs $P \subseteq E(G)$ in graph G, such that the endpoints are all different (no arcs from P are incident with the same node).
When all nodes of G are incident with some arc in P, we call P a **perfect matching**.
Problems:

a) **Maximum Cardinality Matching Problem** - we are looking for matching with the maximum number of edges.

b) **Maximum Cardinality Matching in Bipartite Graphs** - special case of problem a.

c) **Minimum Weight Matching** in a weighted graph - the cheapest matching from the set of all Maximum Cardinality Matchings.

d) **Minimum Weight Perfect Matching** in a complete bipartite graph whose parts have the same number of nodes. This problem is also called an **Assignment Problem** and it is a special case of problem c) and also a special case of the **minimum cost flow problem**.

These problems can be solved in polynomial time.
We will present some algorithms for a), b) and d) only.

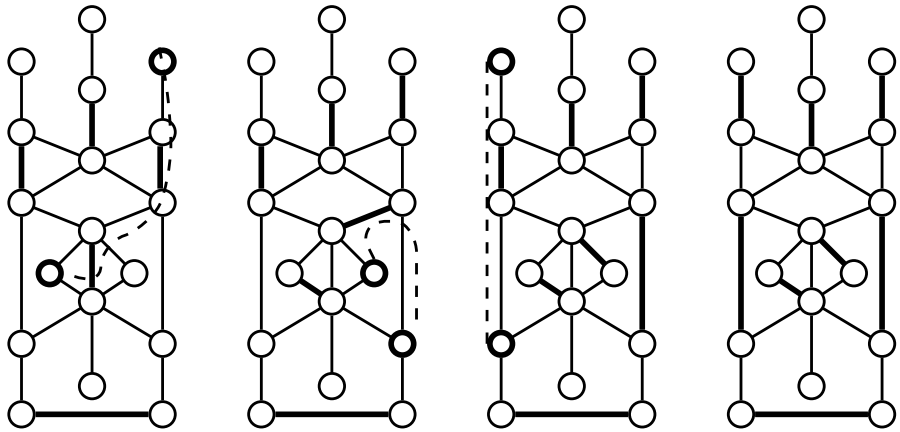
a) Maximum Cardinality Matching Problem - Solution Using M-alternating Path

Definition: Let G be a graph (bipartite or not), and let M be a matching in G. A path P is an M-**alternating path** if $E(P) \setminus M$ is a matching. An M-alternating path is M-**augmenting** if its endpoints are not covered by M.

Theorem: Let G be a graph (bipartite or not) with some matching M. Then M is maximum if and only if there is no M-augmenting path.

Algorithm:

- Find the arbitrary matching.
- Find the M-alternating path with uncovered endpoints. Exchange (i.e. augment) the matching along the alternating path. Repeat as long as such a path does exist.



b) Maximum Cardinality Matching in Bipartite Graphs - Solution by Maximum Flow

Can be transformed to the **maximum flow** problem:

- Add source s and edge (s, i) for all $i \in X$
- Add sink t and edge (j, t) for all $j \in Y$
- Edge orientation should be from s to X, from X to Y and from Y to t
- The upper bound of all edges is equal to 1 and the lower bound is equal to 0
- By solving the maximum flow from s to t we obtain maximum cardinality matching

d) Assignment Problem - Solution by Minimum Cost Flow

We have n employees and n tasks and we know the cost of execution for each possible employee-task pair.

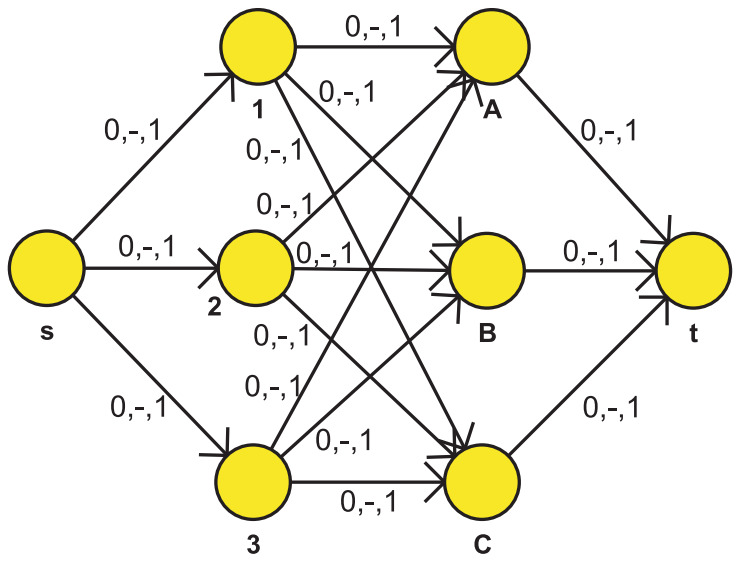
Goal

Assign one task per employee while minimizing the total cost.

Can be solved by transformation to **minimum cost flow problem**:

Example: cost of execution of task 1,2,3 by employee A,B,C

	A	B	C
1	6	2	4
2	3	1	3
3	5	3	4



Add source s, sink t and related edges.

Alternatively we can solve the assignment problem by Hungarian Alg.

Hungarian Algorithm - Solution of Assignment Problem (i.e. Minimum Weight Perfect Matching in Complete Bipartite Graph whose Sets Have the Same Cardinality)

Description:

- G - Complete undirected bipartite graph containing sets X, Y which satisfy $|X| = |Y| = n$.
- Edge costs are arranged in the matrix, where element $c_{ij} \in \mathbb{R}_0^+$ represents the cost of edge $(i, j) \in X \times Y$.

The basic ideas of the Hungarian algorithm:

- **Assign** an arbitrary real number $p(v)$, to every vertex $v \in V(G)$. These numbers are used to transform the costs $c_{ij}^p = c_{ij} - p_i^x - p_j^y$.
- For every perfect matching, this transformation **changes the total cost by the same value** (every node participates exactly once). Thanks to this, the cheapest perfect matching is still given by the **same set of edges**.

Solution of Assignment Problem - Hungarian Algorithm

Definition: We call numbers assigned to nodes p a **feasible rating** if all transformed costs are nonnegative, i.e. $c_{ij}^p \geq 0$.

Definition: When p is the feasible rating, we call G^p an **equality graph** if it is a factor of graph G which contains only edges with **zero cost**.

Theorem

P is the optimal solution to the assignment problem if the equality graph G^p contains perfect matching.

Idea of the proof: The cost of matching P in G^p is equal to zero. There is no cheaper matching, since it is the feasible rating, where $c_{ij}^p \geq 0$.

Hungarian Algorithm

Input: Undirected complete bipartite graph G and costs $c : E(G) \to \mathbb{R}_0^+$.
Output: Perfect matching $P \subseteq E(G)$ whose cost $\sum_{(i,j)\in P} c_{ij}$ is minimal.

1. For all $i \in X$ compute $p_i^x := \min_{j\in Y}\{c_{ij}\}$
   and for all $j \in Y$ compute $p_j^y := \min_{i\in X}\{c_{ij} - p_i^x\}$
2. Construct the equality graph G^p;
   $E(G^p) = \left\{(i,j) \in E(G); c_{ij} - p_i^x - p_j^y = 0\right\}$
3. Find the maximum cardinality matching P in G^p.
   If P is perfect matching, the computation ends.
4. If P is not perfect, find set $A \subseteq X$ and set $B \subseteq Y$ **incident to** A **in** G^p satisfying $|A| > |B|$. Compute
   $$d = min_{i\in A, j\in Y\setminus B}\left\{c_{ij} - p_i^x - p_j^y\right\}$$
   and change the rating of the nodes as follows:
   $$p_i^x := p_i^x + d \text{ for all } i \in A$$
   $$p_j^y := p_j^y - d \text{ for all } j \in B$$
   Go to 2.

Time complexity is $O(n^4)$.

Hungarian algorithm - example

Cost matrix (It is neither an adjacency nor an incidence matrix):

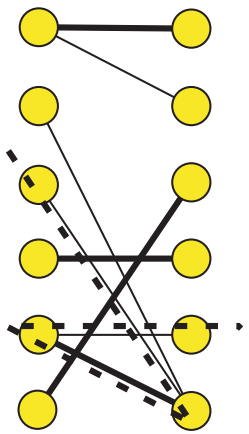
$$\begin{array}{cccccc} 5 & 3 & 7 & 4 & 5 & 4 \\ 10 & 11 & 10 & 7 & 8 & 3 \\ 18 & 7 & 6 & 6 & 6 & 2 \\ 6 & 12 & 2 & 1 & 9 & 8 \\ 8 & 4 & 4 & 4 & 1 & 1 \\ 4 & 8 & 1 & 3 & 7 & 4 \end{array}$$

1. First from each row subtract off the row minimum - we obtain the rating for nodes in X.
   Now subtract the lowest element of each column from that column - we obtain the rating for nodes in Y.

Hungarian Algorithm - Example

2. Create the transformed cost matrix and note p_i^x for every row and p_j^y for every column. Construct the equality graph G^p.
3. Find the maximum cardinality matching in bipartite graph G^p (i.e. solve the problem b)).

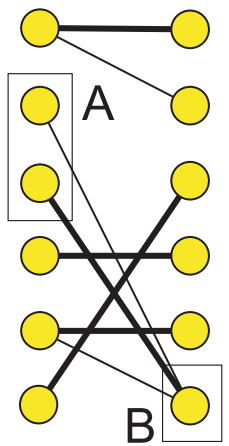
							p_i^x
	0	0	4	1	2	1	3
	5	8	7	4	5	0	3
	14	5	4	4	4	0	2
	3	11	1	0	8	7	1
	5	3	3	3	0	0	1
	1	7	0	2	6	3	1
p_j^y	2	0	0	0	0	0	



Hungarian algorithm - example

4. Since the matching is not perfect yet:
   - find (blue) set A and (green) set B (start the labeling procedure from the uncovered node in X)
   - from the blue elements of the matrix find the minimum $d = 4$

							p_i^x
	0	0	4	1	2	1	3
	5	8	7	4	5	0	3
	14	5	4	4	4	0	2
	3	11	1	0	8	7	1
	5	3	3	3	0	0	1
	1	7	0	2	6	3	1
p_j^y	2	0	0	0	0	0	

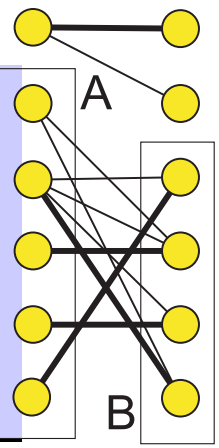


- add d to p_i^x, subtract d from p_j^y and recalculate the cost matrix

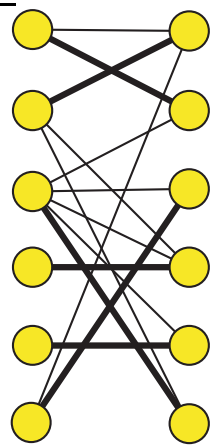
Hungarian Algorithm - Example

2. Several zeros appeared in the matrix and several edges appeared in G^p. On the contrary, edge (5,6) disappeared.
3. The cardinality of the matching cannot be increased now.
4. Find sets A (blue) and B (green). Minimum $d = 1$.
2. Now, perfect matching does exist in graph G^p. The cost is equal to 18 (sum of the ratings).

							p_i^x
	0	0	4	1	2	5	3
	1	4	3	0	1	0	7
	10	1	0	0	0	0	6
	3	11	1	0	8	11	1
	5	3	3	3	0	4	1
	1	7	0	2	6	7	1
p_j^y	2	0	0	0	0	-4	



							p_i^x
	0	0	5	2	3	6	3
	0	3	3	0	1	0	8
	9	0	0	0	0	0	7
	2	10	1	0	8	11	2
	4	2	3	3	0	4	2
	0	6	0	2	6	7	2
p_j^y	2	0	-1	-1	-1	-5	



References


Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin.
Network Flows: Theory, Algorithms, and Applications.
Prentice Hall, 1993.


Jiří Demel.
Grafy a jejich aplikace.
Academia, 2002.


B. H. Korte and Jens Vygen.
Combinatorial Optimization: Theory and Algorithms.
Springer, fourth edition, 2008.