Combinatorial Optimization

Zdeněk Hanzálek
`hanzalek@fel.cvut.cz`

CTU FEE Department of Control Engineering

May 3, 2013

Network Flows

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

CTU FEE Department of control engineering

April 2, 2013

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 while t represents 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.

$\delta^+(v)$ is a set of arcs leaving node v
$\delta^-(v)$ is a set of arcs entering node v

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 $P \left| pmtn, r_j, \widetilde{d_j} \right| C_{max}$ 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 $\langle 1, 3 \rangle$.

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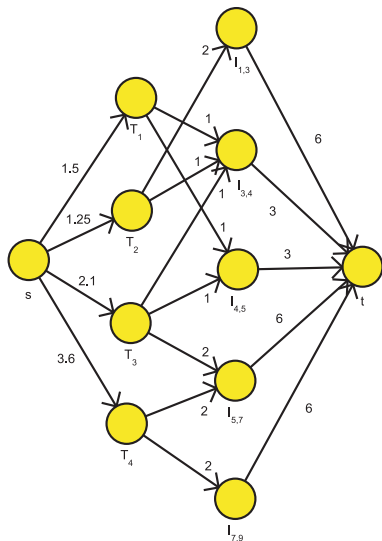
Dynamic flow is changing its volume in time. We will show, how to formulate this problem while introducing discrete time and using (static) flow.

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 manage transport of all cars so that the maximum number of them reaches 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 (in the picture). In 1956, they published a widely known **algorithm for the maximum flow problem**, the Ford-Fulkerson Algorithm.

Ford-Fulkerson Algorithm

The algorithm is based on **incremental augmentation of the flow** while maintaining its feasibility.

Forward arc and backward arc

An arc is called a *forward arc* if it is orientated in the same direction as the path from the source to the sink and a *backward arc* otherwise.

Augmenting path

An augmenting path for flow f is an **undirected** path from source s to sink t with:

- $f(e) < u(e)$ if e is a forward arc ... the flow can be increased
- $f(e) > l(e)$ if e is a backward arc... the flow can be decreased

Capacity of an augmenting path

Capacity γ of the augmenting path 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 flow by γ on forward arcs and decrease 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
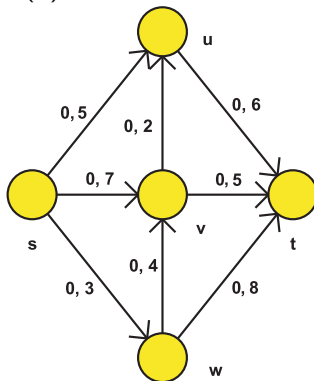
Input: Network (G, l, u, s, t), feasible flow f.
Output: Augmenting path P.

1. Label $m_v = FALSE$ $\forall v \in V(G)$, $m_s = TRUE$ (mark node 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 2 or 3 .
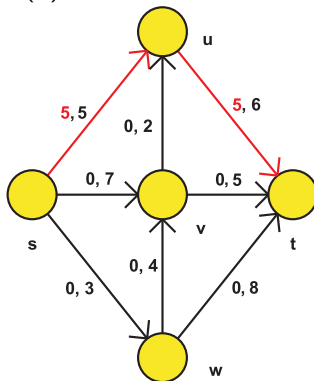
arcs are labeled by $f(e), u(e)$

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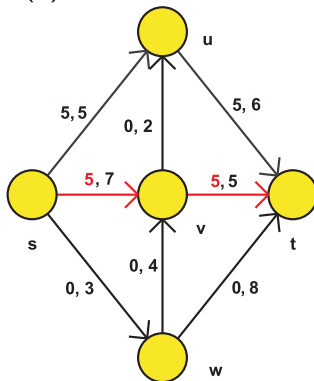
arcs are labeled by $f(e), u(e)$

Ford-Fulkerson Algorithm
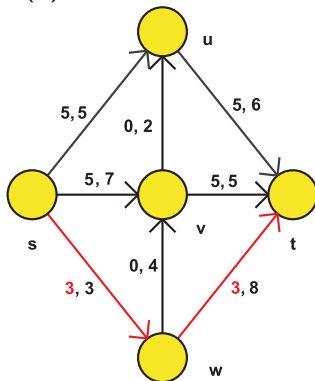# Zero Lower Bounds Example

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

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

arcs are labeled by $f(e), u(e)$
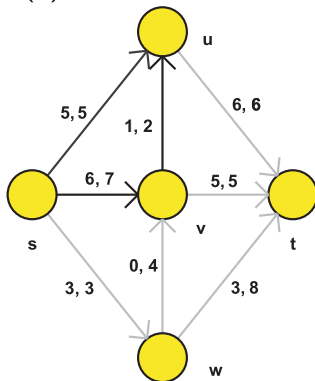
set A=$\{s, u, v\}$ determines the **minimum capacity cut**

Ford-Fulkerson Algorithm
Non-zero Lower Bounds Example

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

Capacity of 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 $\delta(A)$ 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 $C(A) = \sum_{e \in \delta^+(A)} u(e) - \sum_{e \in \delta^-(A)} l(e)$.
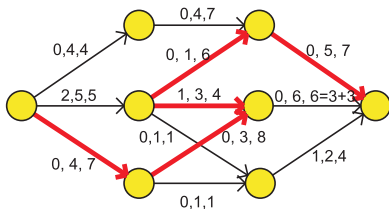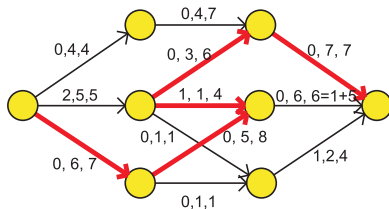
Ford and Fulkerson [1956]

The value of the maximum flow from s to t is equal to the capacity of the **minimum cut**. 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 $e \in \delta^-(A)$.

The value of the max. flow is equal to the capacity of the minimum cut:
$\sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e) = \sum_{e \in \delta^+(A)} u(e) - \sum_{e \in \delta^-(A)} l(e)$.

Integrality

Integral Flow Theorem (Dantzig and Fulkerson [1956])

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

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

When we choose the augmenting path **inappropriately**, we can augment the flow only by unit steps. For non-integer capacities and non-integer flow the algorithm **might not terminate at all**.
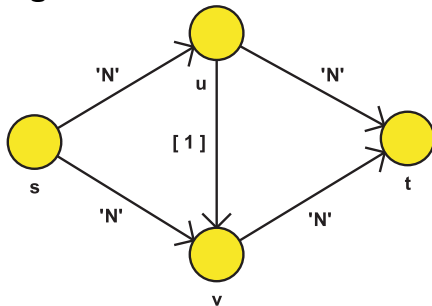
Edmonds and Karp [1972])

When choosing the augmenting path, if we always choose the **shortest one**, time complexity is $O\left(m^2 \cdot n\right)$.

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

Assignment problem with additional constraint.

Let us have n residents of 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.

The objective is to find the maximum number of representatives or prove that it does not exist. Formulate as the Maximum Flow Problem.

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

This application is concerned with consistent rounding of the elements, row sums, and column sums of a matrix. We are given a $p \times q$ matrix of real numbers with row sums vector and column sums vector. We can round any real number a to the next smaller integer or to the next larger integer, 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
- the sum of the rounded elements in each column is equal to the rounded column sum

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 as a Decision Problem

We assume several sources and several sinks. We do not assume lower bounds.

Can be polynomially reduced to the Maximum flow problem (with one source and one sink).

Feasible flow in the network

- **Instance**: Let (G, u, b) be a network where G is a digraph with upper bounds $u : E(G) \rightarrow \mathbb{R}_0^+$ and with:
  - balance $b : V(G) \rightarrow \mathbb{R}$ 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.
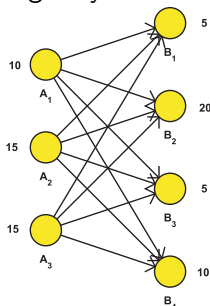
Goal

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.

We transform this problem to the **maximum flow problem with zero lower bounds**:

1. We add a new node s called the source and add edges (s, v) with 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 upper bound $u_v = -b(v)$ for every node that satisfies $b(v) < 0$

3. We solve the maximum flow problem for the lower bounds equal to zero (we start with the initial feasible flow equal to zero).

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 decision problem** 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. Now we can apply Kirchhoff's law to all nodes including s and t.

2. Therefore, we look for a feasible circulation which must satisfy:
$$\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = 0 \qquad v \in V(G)$$
$$l(e) \leq f(e) \leq u(e) \qquad e \in E(G)$$

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

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

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

where $b(v) = \sum_{e \in \delta^-(v)} l(e) - \sum_{e \in \delta^+(v)} l(e)$ $v \in V(G)$

4. This is a **feasible flow decision problem** 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).

5. 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: the problem of finding the **initial flow with nonzero lower bounds** can be transformed to the **feasible flow decision problem** which can be further transformed to the **maximum flow problem 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 $e \in E(G)$.

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

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

The maximum flow problem can be transformed 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.

A commander is located at one node p in a communication network G and his soldiers are located at nodes denoted by the set S. Let c_{ij} be the effort required to eliminate undirected arc i, j from the network. The problem is to determine the minimal effort required to block all communications between the commander and his subordinates. How can you solve this problem in polynomial 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 bipartite graphs only.

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

b) Maximum Cardinality Matching in Bipartite Graphs - Solution Using M-alternating Path

Definition: Let G be a graph (bipartite or not), and let M be some 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.
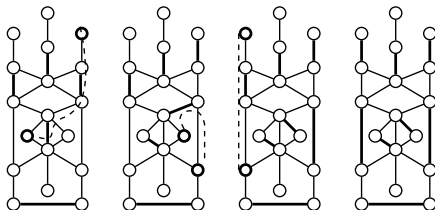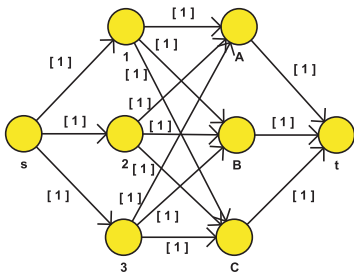
d) Assignment Problem - Example

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.

edges labeled by: $u(e)$
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

We can solve this problem as a **minimum cost flow problem** (on the picture) or as an **assignment problem**.

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 only once). Thanks to this, the cheapest perfect matching is still given by the **same set of edges**.

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 $c : E(G) \rightarrow \mathbb{R}_0^+$.
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:
   $p_i^x := p_i^x + d$ for all $i \in A$
   $p_j^y := p_j^y - d$ for all $j \in B$
   Go to 2. Time complexity is $O(n^4)$.

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

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

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.

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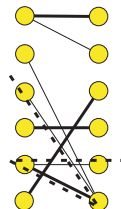
④ Since the matching is not perfect yet:
- find (blue) set A and (green) set B (start the labeling procedure from the uncovered node in X)
- from the blue elements of the matrix find the minimum $d = 4$

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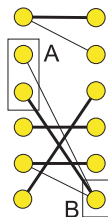
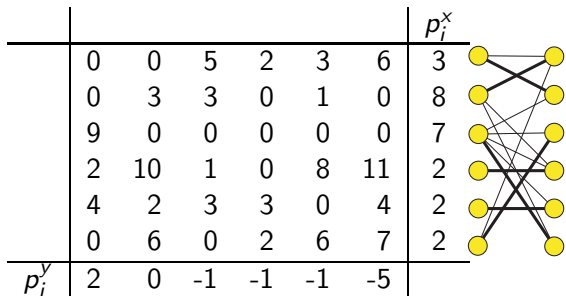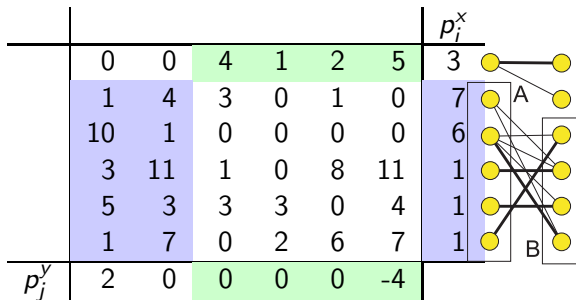- add d to p_i^x, subtract d from p_j^y and recalculate the cost matrix

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.

							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		

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	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		

Multicommodity Flow Problem

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)

Progress of routing
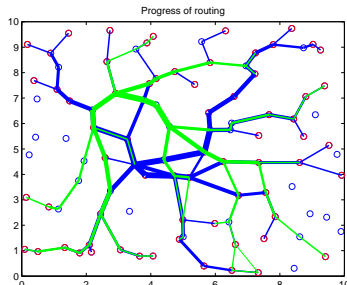
Minimum Cost Multicommodity Flow Problem

Minimum cost multicommodity flow problem

- **Instance**: 5-tuple $(G, l, u, c, b^1 \ldots b^m \ldots b^{|M|})$ 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 that 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 Problem
LP Formulation

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

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

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

- 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.

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.