Discrete mathematics

Petr Kovář & Tereza Kovářová
petr.kovar@vsb.cz

VŠB – Technical University of Ostrava

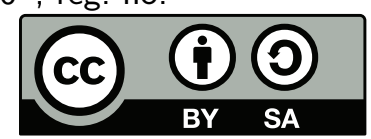
Winter Term 2022/2023
DiM 470-2301/02, 470-2301/04, 470-2301/06

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION,
YOUTH AND SPORTS

The translation was co-financed by the European Union and the Ministry of Education, Youth and Sports from the Operational Programme Research, Development and Education, project "Technology for the Future 2.0", reg. no. CZ.02.2.69/0.0/0.0/18_058/0010212.

This work is licensed under a Creative Commons "Attribution-ShareAlike 4.0 International" license.



About this file

This file is meant to be a guideline for the lecturer. Many important pieces of information are not in this file, they are to be delivered in the lecture: said, shown or drawn on board. The file is made available with the hope students will easier catch up with lectures they missed.

For study the following resources are better suitable:

- Meyer: Lecture notes and readings for an http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-fall-2005/readings/" (weeks 1-5, 8-10, 12-13), MIT, 2005.
- Diestel: Graph theory http://diestel-graph-theory.com/ (chapters 1-6), Springer, 2010.

See also http://homel.vsb.cz/~kov16/predmety_dm.php

Lecture overview

Chapter Flow in a network

- motivation
- definition of a network
- maximum flow algorithm
- network generalization
- further applications

Flow in a network

Motivation Graph theory solves many problems on networks. We are given a network (computer network, pipelines, . . .), where edges represent connections and vertices form crossings or routers.
It is natural to give a bound on capacity of each edge (capacity = number).

Question

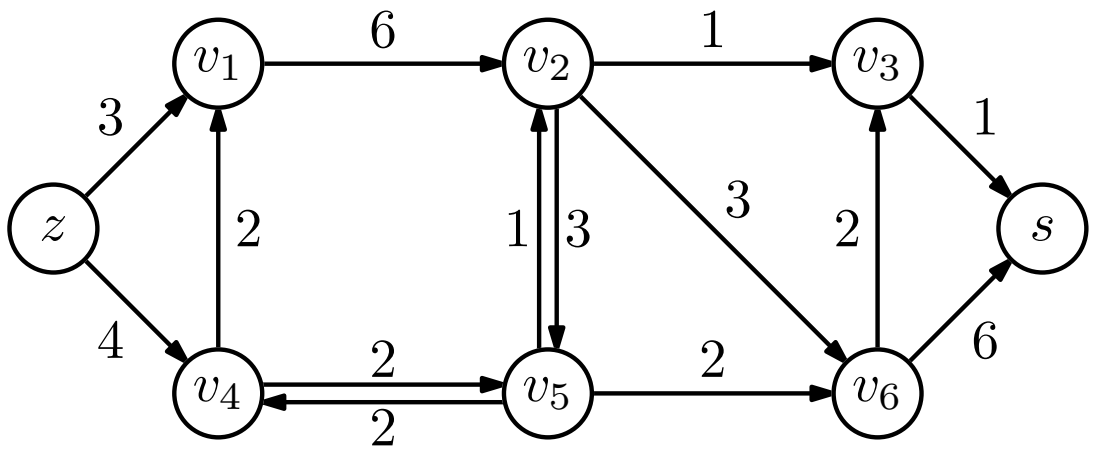
What is the largest possible number of units that can be transfered through the network (with given constraints) from z (source) to s (sink).

A network is a graph in which the capacities, the source and sink are given.

Definition

Network is a four-tuple $S = (G, z, s, w)$, where

- G is an oriented graph,
- vertices $z \in V(G)$, $s \in V(G)$ are the source and sink,
- $w : E(G) \to \mathbb{R}^+$ is a positive labeling of edges, called edge capacity.



Question

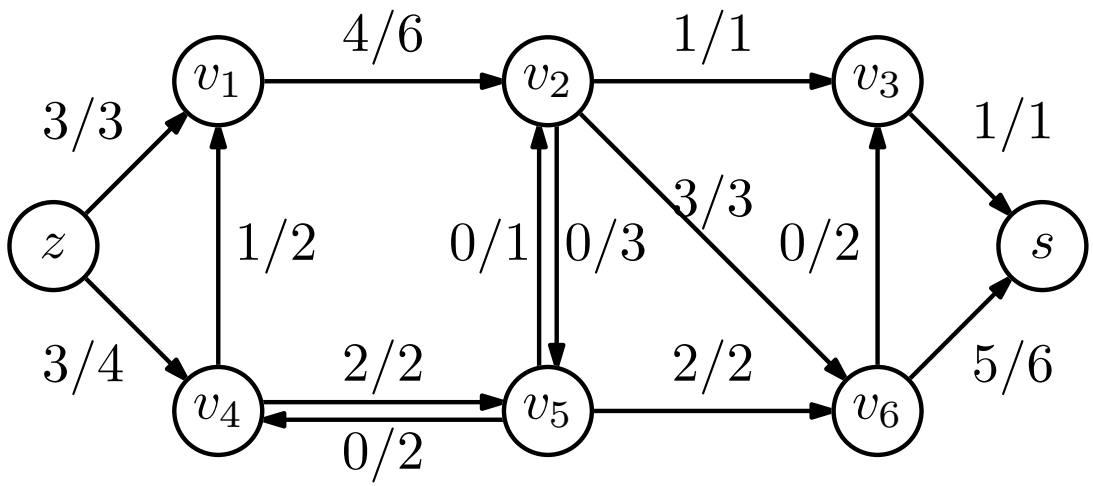
What is the largest possible number of units that can be transfered through the network G from the source z to the place of consumption s (sink). Of course obeying the *maximal capacity* of each edge.

A more complex problem is a network with

- given capacities of vertices
- multiple sources and sinks
- more products to be transfered in a network

In some cases this complex problem can be translated into to basic network defined above. We show how.

Notice that even for a maximum flow through a network the full capacities of all edges do not have to be achieved. This is when we define the flow correctly (flow/capacity).



Note

The edge capacity does not really have to be a capacity (can represent width, automobiles per minute, thickness of a pipe, resistance. . .) We will use terminology from liquid flows: amount „entering“ and „leaving“ a certain vertex.

By $e \to v$ we denote *incoming* edges to v, by $e \leftarrow v$ *outgoing* edges.

Definition

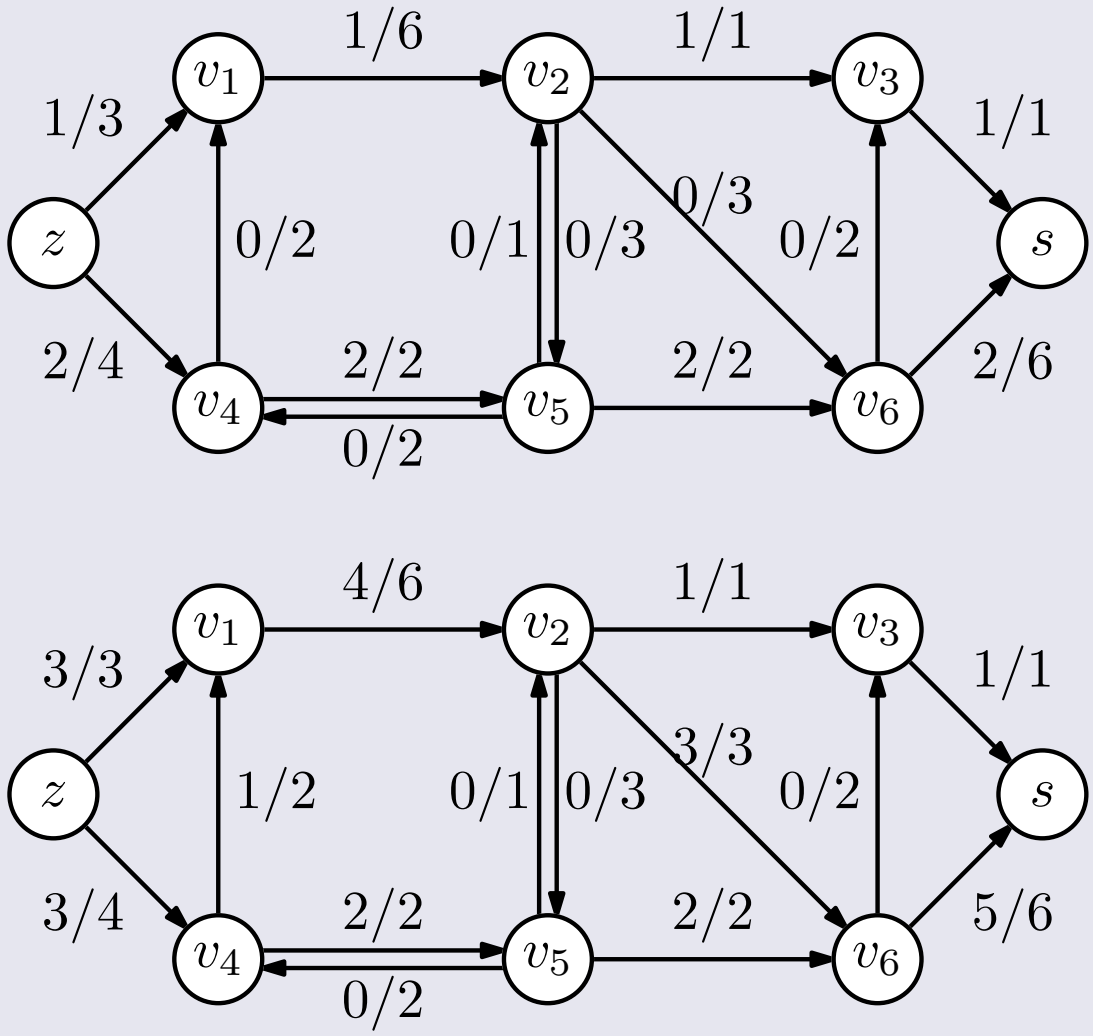
A flow in network $S = (G, z, s, w)$ is a function $f : E(G) \to \mathbb{R}_0^+$, where

- no edge capacity is exceeded: $\forall e \in E(G) : 0 \leq f(e) \leq w(e)$,
- the conservation-of-flow equation hold: $\forall v \in V(G), v \neq z, s : \sum_{e \to v} f(e) = \sum_{e \leftarrow v} f(e)$.

The value of a flow f is

$$\|f\| = \sum_{e \leftarrow z} f(e) - \sum_{e \to z} f(e).$$

Example



Flow and a maximum flow in a network (G, z, s, w).

Source and sink are exceptional vertices in the network.

The conservation-of-flow equations do not hold for them!

- from the source "issues" more than comes in
- the sink "drains" more than comes out

The difference for both these vertices is the same.

Lemma

By f_z we denote the sum of flows on outgoing edges minus the sum of flows on the incoming edges to the source z. By f_s we denote the sum of flows on outgoing edges minus the sum of flows on the incoming edges to s. Then $f_z = -f_s$ holds.

Proof

$$0 = \sum_e (f(e) - f(e)) = \sum_v \sum_{e \leftarrow v} f(e) - \sum_v \sum_{e \rightarrow v} f(e) = \sum_{v \in \{z,s\}} \left(\sum_{e \leftarrow v} f(e) - \sum_{e \rightarrow v} f(e) \right).$$

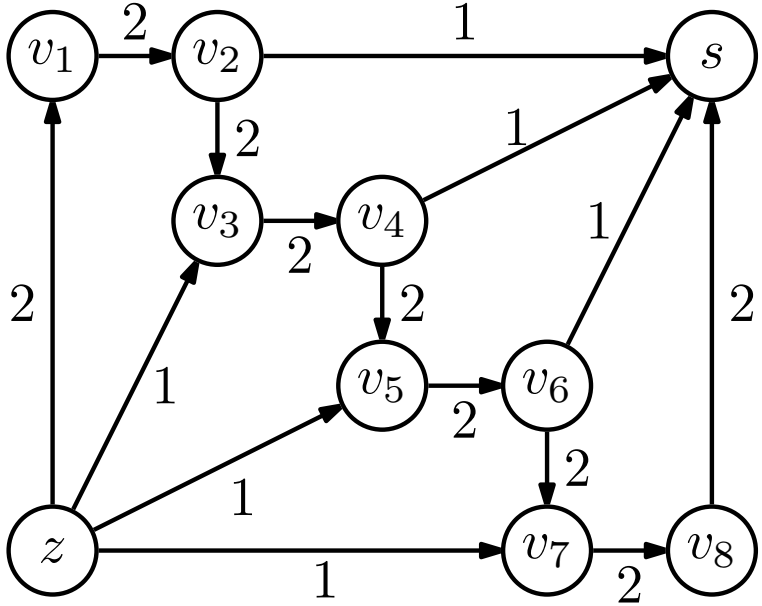
Double sums cancel out for all vertices in the network except z and s.

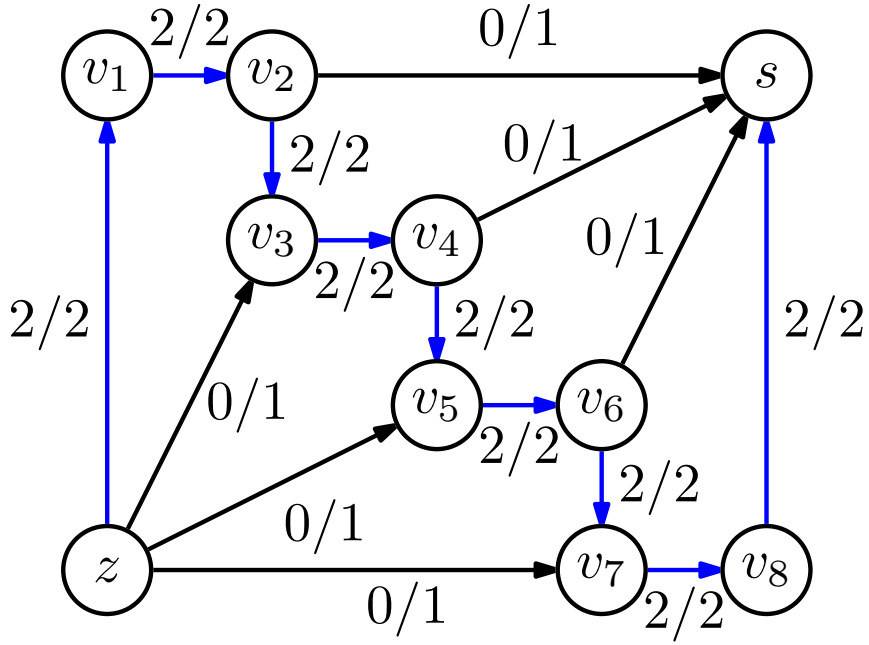
$$\left(\sum_{e \leftarrow z} f(e) - \sum_{e \rightarrow z} f(e) \right) = - \left(\sum_{e \leftarrow s} f(e) - \sum_{e \rightarrow s} f(e) \right). \qquad \square$$

Maximum flow algorithm

Our goal is to find the maximum possible flow from source z to sink s in a network (G) with capacities w.

A greedy algorithm does not give the maximum flow! (see figure)





A greedy algorithm yields a total flow of value 2.

The flow cannot be increased by simply adding some path with non-zero flow, but this is not the maximum flow. There exists a flow of value 5.

Definition

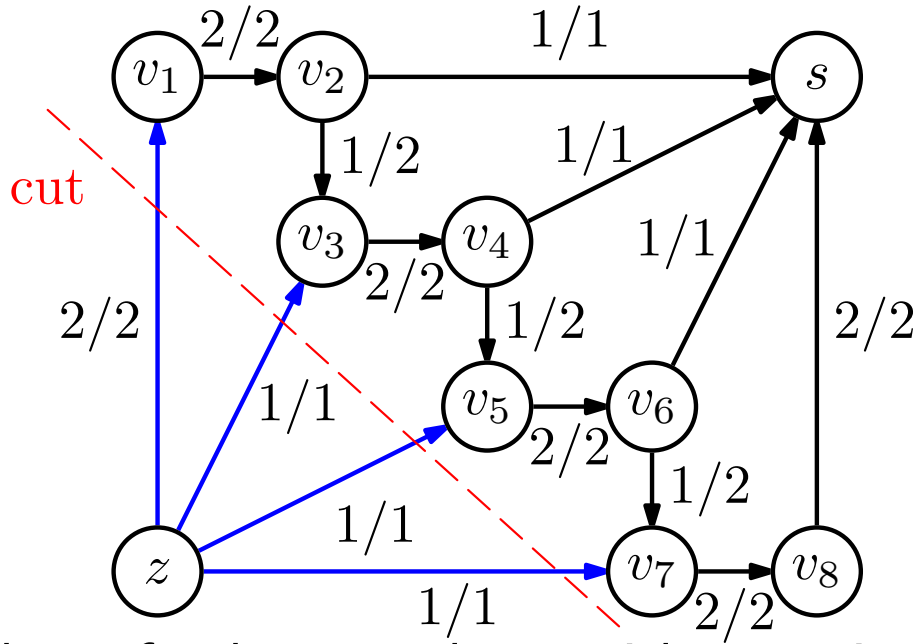
A cut in a network $S = (G, z, s, w)$ is such a subset of edges $C \subseteq E(G)$, that in $G - C$ (G with edges of C deleted) no oriented path from z to s remains.

Capacity of the cut C is the sum of capacities of edges in C, i.e. $\|C\| = \sum_{e \in C} w(e)$.

Theorem

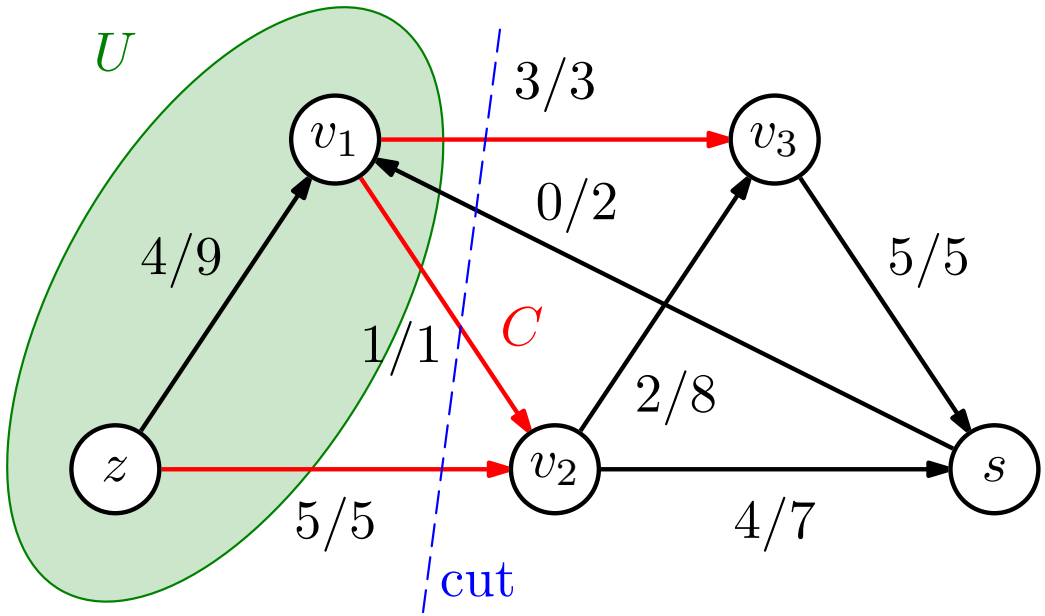
Value of the maximum flow equals the capacity of the minimum cut.

Proof later...



Flow of value 5 *and cut with capacity* 5.

Beware, one has to know what is a cut!



A cut contains only edges leading out of set U.

The theorem characterizes nicely the maximum flow:

The flow of value x is maximum, if there is a cut of capacity x.

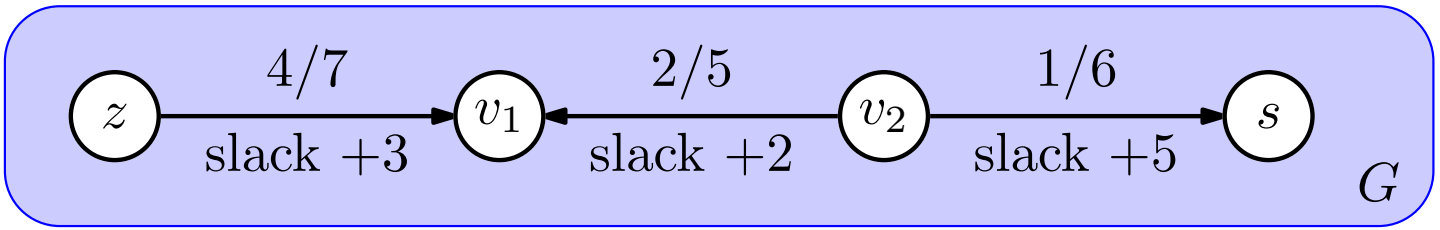
Definition

Let S be a network and let f be a flow in this network. An unsaturated path in S is an unoriented path $e_1, e_2, \ldots, e_m$ in G from vertex u to vertex v (usually from z to s), where

- $f(e_i) < w(e_i)$ for e_i oriented "along" the path from u to v,
- $f(e_i) > 0$ for e_i oriented the opposite way.

The value $w(e_i) - f(e_i)$ for edges e_i oriented from u to v and the value $f(e_i)$ for edges e_i in the opposite way is called the slack of the edge e_i.

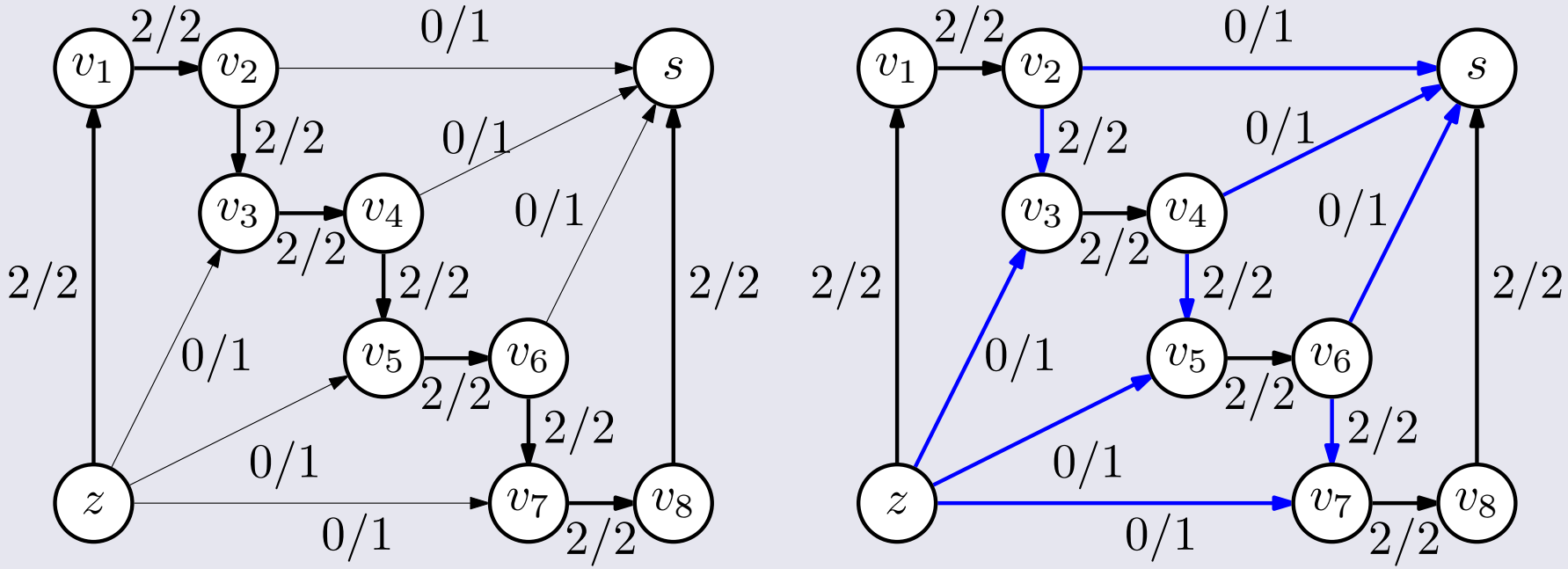
An unsaturated path has positive slacks δ on all edges.



Path with slack 2.

Example

Find several unsaturated paths in the given network.



Example of three unsaturated paths.

Algorithm Ford–Fulkerson's algorithm

```
input:  network S = (G,z,s,w);
initial flow is zero on every edge;

do {
    searching through G find the set U of all vertices
    reachable from z on unsaturated paths in G;
    if (s in U) {
       P = unsaturated path (found above) in S from z to s;
       increase the flow f by the slack of P;
} while (s in U);
output:  print the maximum flow f;
output:  print the min. cut as all edges from U to V(G)-U.
```

Proof We give a direct proof.

Obviously holds $\|f\| \leq \|C\|$.
If at the end of the algorithm we have a flow f and in network S there is a cut of the same capacity $\|C\| = \|f\|$, obviously we have the maximum possible flow in the network S. At the same time we prove Theorem on maximum flow.
It is enough to show that at the end of the algorithm the equality $\|f\| = \|C\|$ holds, where C is the cut between U and the remaining vertices in G.

Suppose we have a flow f in S and no unsaturated path from z to s exists. Thus the set U from the algorithm does not contain s (it is not reachable via unsaturated paths).

Since from U lead no unsaturated paths (nor edges), has every edge $e \leftarrow U$ (outgoing from U) full capacity $f(e) = w(e)$ and each edge $e \rightarrow U$ (incoming to U) flow $f(e) = 0$. The value of the flow f from z to s is

$$\|f\| = \sum_{e \leftarrow U} f(e) - \sum_{e \rightarrow U} f(e) = \sum_{e \leftarrow U} f(e) - 0 = \sum_{e \in C} w(e) = \|C\|.$$

This completes the proof. □

The algorithm finds the maximum flow with value $\|f\|$. Moreover, after the algorithm stops, one can easily find the minimum cut with value $\|C\|$. First we give the following observation:

Corollary

If all capacities in the network S have nonnegative integer values, has the maximum flow also integer value.

The maximum flow always fully saturates the edges of some edge cut, thus the value of such flow equals the sum of capacities of this edge cut. Especially, if all weighs are integer values, so will be the maximum flow.

Note

We point out that omitting the requirement on integer capacities one can come up with examples of simple networks with irrational capacities for which Ford-Fulkerson algorithm does not stop after finitely many steps.

Moreover, the iterated flow does not have to converge toward the maximum flow.

Generalizations of networks and their applications

The method shown above we can generalize to

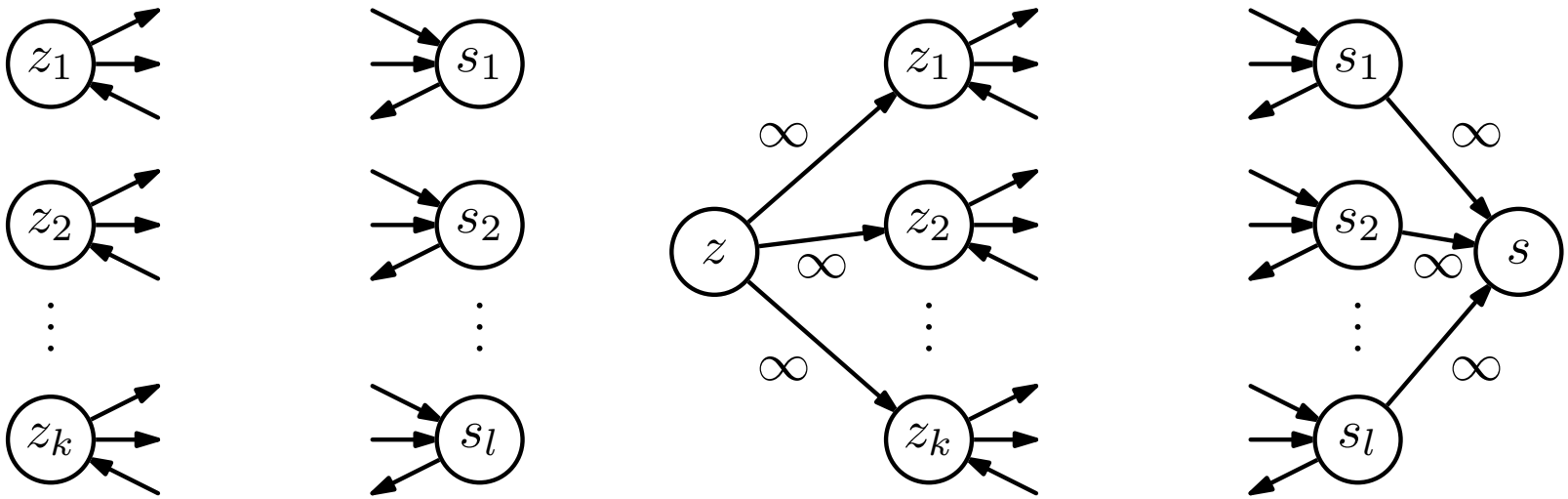
1. multiple sources and sinks,
2. allow unoriented edges in a network,
3. introduce capacities of vertices,
4. transport several products in one network,
5. pose minimal capacities on edges (something has to flow).

Instead of providing new of modifying the previous algorithms for each of the problems, we show how to modify the network.

The more complex problems will be transformed to the basic problem for which Ford-Fulkerson's algorithm can be used.

1) Multiple sources and sinks

If there are multiple sources or sinks in the network, we can transform the problem easily to a single source/sink problem.

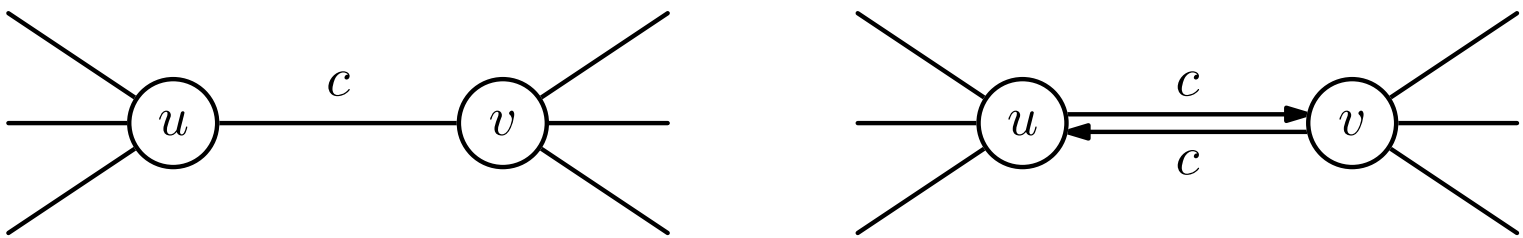


One can also introduce the source/sink capacities by taking the corresponding edge capacities instead of ∞.

2) Unoriented edges

In some real life application the orientation of edges is given (e.g. in sewage system, traffic corridors in road networks, . . .) In other applications the orientation of edges can be arbitrary (information or computer networks). Ford-Fulkerson algorithm is designed for oriented graphs.

Unoriented edges can be simply represented by a pair of edges with the same capacity and opposite orientation.

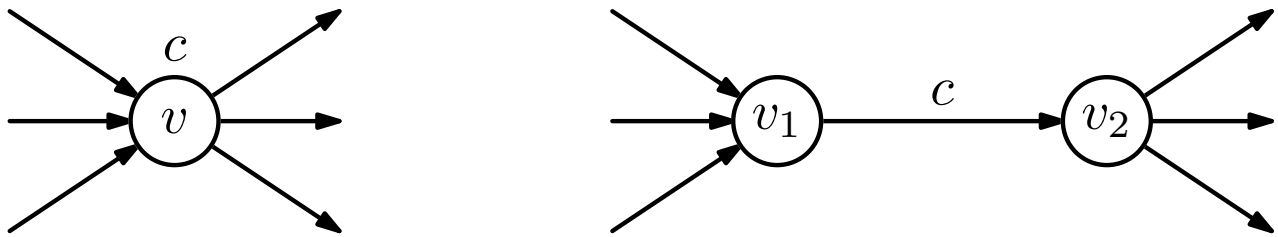


Yet, once the algorithm stops, the flow on the unoriented edge is given by the difference of flows on both oriented edges.

3) Vertex capacities

Naturally, constraints can arise not only for vertices, but also for vertices (crossings, nodes).

A network with vertex capacities can easily be translated into a network where just edges have capacities.



Each vertex with a given capacity c we replace by a pair of vertices joined by an edge with capacity c (we double the vertex).

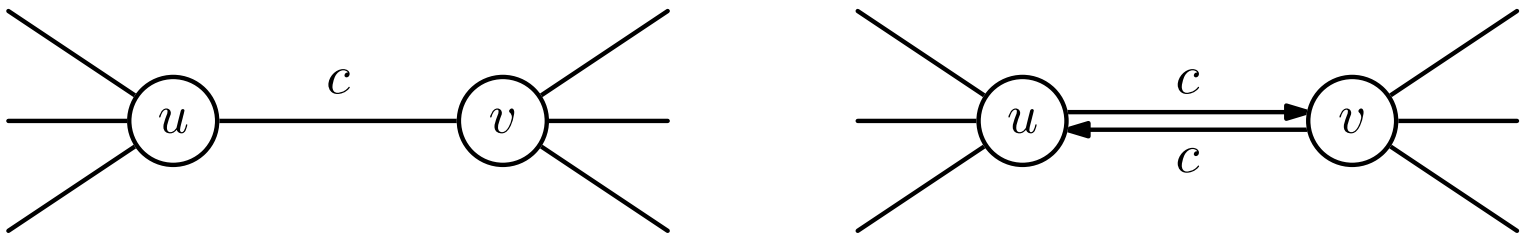
Arcs incoming to v will come into v_1 and arcs outgoing from v will go out from v_2.

4) Several products in one network

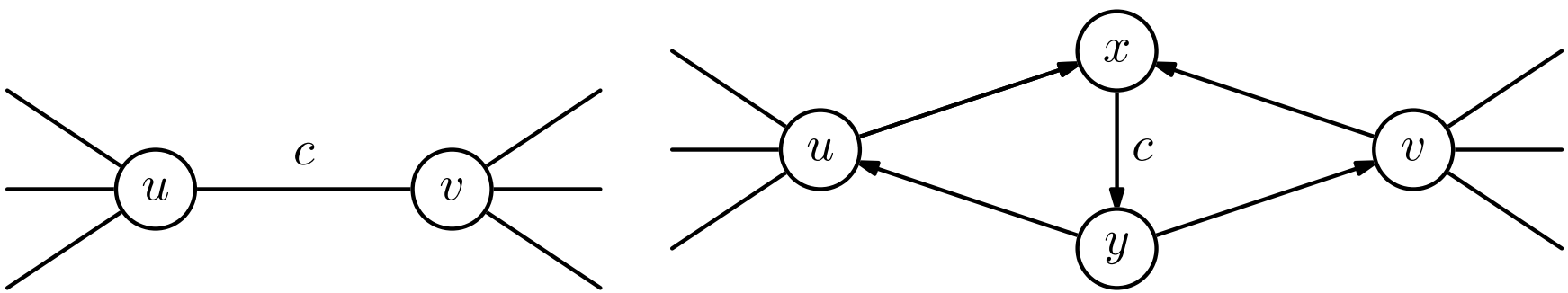
For multiple products transfered in one network the problem is complex. The algorithm for a maximum multi-product flow is beyond this course.

We show, how to translate an unoriented graph into an oriented network with given capacities.

Beware, it is not sufficient to take two opposite arcs:



The sum of flows transfered in one direction and another product transfered in the opposite direction must not exceed the capacity.



This conversion guarantees the total capacity for several products.

5) Minimal capacities of edges

If there are besides the maximal capacities also minimal capacities given, i.e. there is a nonzero flow required, the solution does not need to exist.

This problem is beyond the scope of this course.

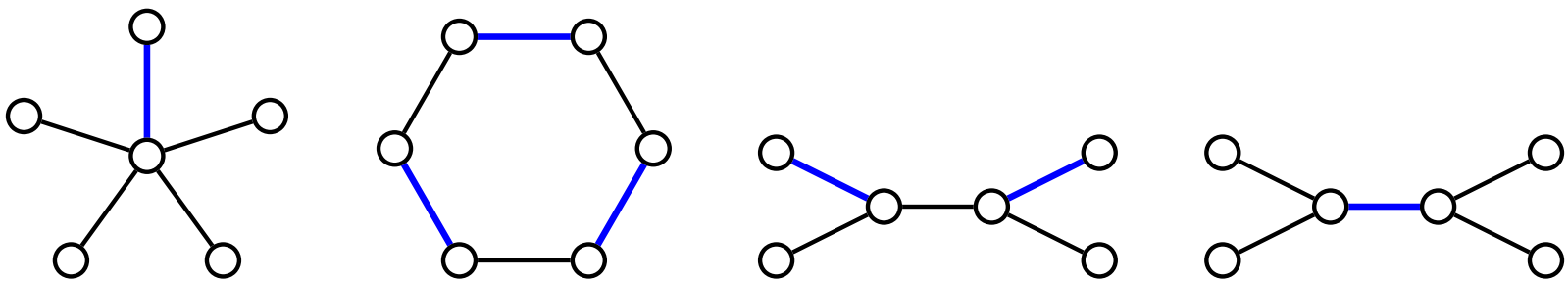
FYI: J. Demel, Grafy, SNTL, Praha, (1989).

Matching in bipartite graphs

There is a surprising variety of applications of the maximum flow algorithm. We show how to translate the search for maximum matching into the search for maximum flow.

Definition

Matching in (bipartite) graph G is a subset of independent edges $M \subseteq E(G)$ (no two edges in M share a vertex).



There are two graphs on six vertices. In the graph on the left the matching has at most one edge, while in the second graph there is a matching that covers all vertices.

Algorithm Maximum matching in a bipartite graph

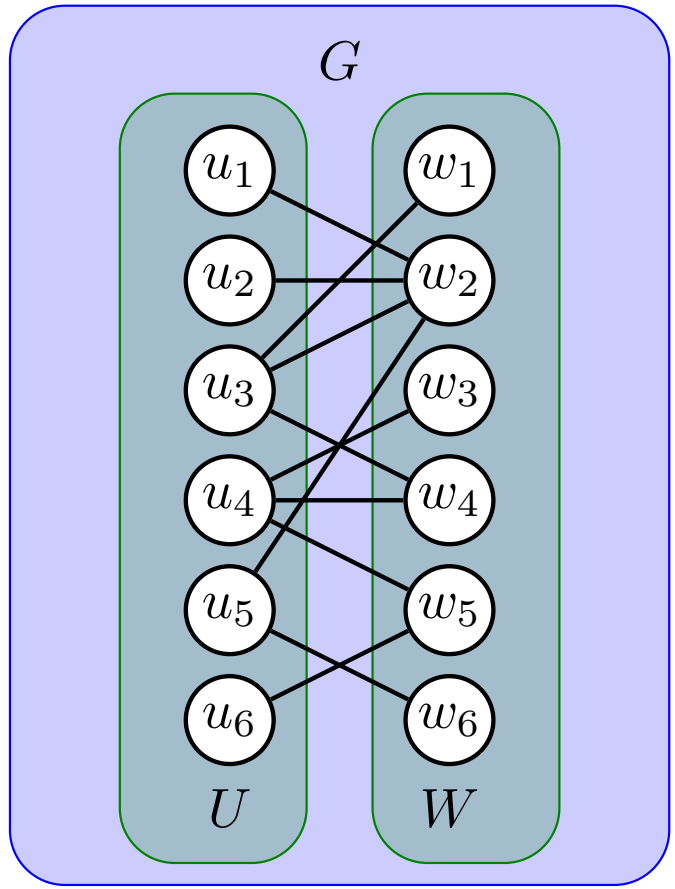
Let G be a bipartite graph vertex set split into two partite sets U and W:

1. we construct a network S: we add the source z and sink s, all vertices in U we join to z and all vertices in W to s, all edges are oriented from the source to the sink; their capacity is 1;
2. we find a (integer) maximum flow in S using previous algorithm;
3. maximum matching in G contains edges with non-zero flow.

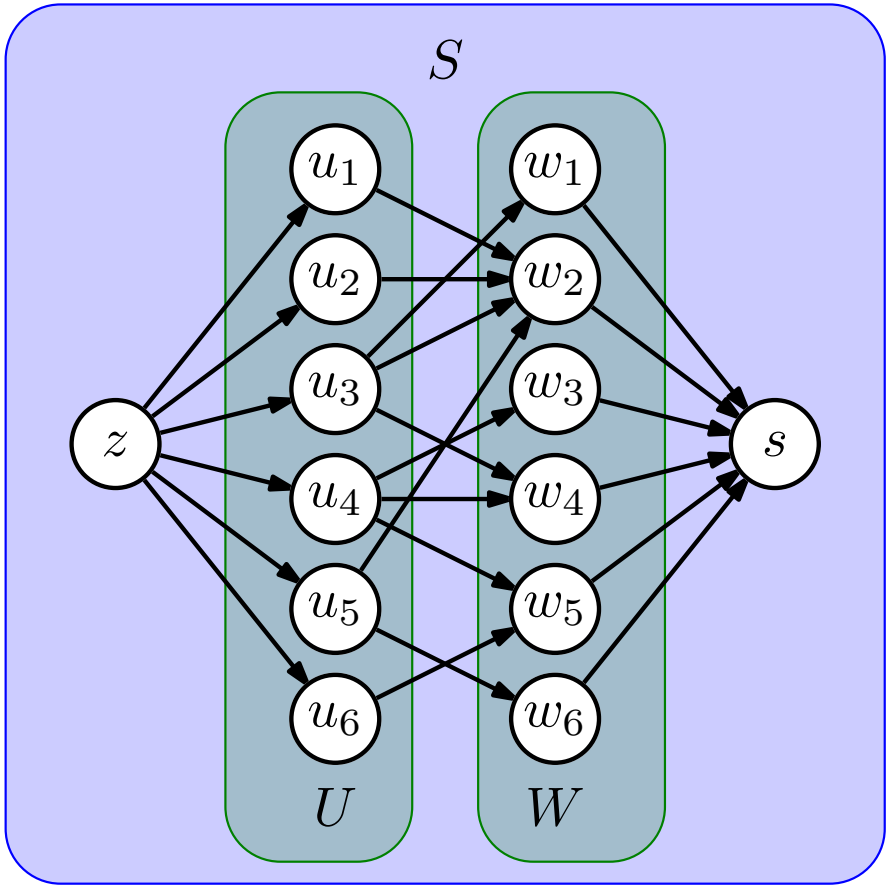
Proof By a corollary the maximum flow will have integer flow, the flow on each edge is either 0 or 1.

Each vertex in U is the end-vertex of precisely one edge with capacity 1, thus each vertex will be in at most one edge of the matching. Similarly for vertices in W. Thus the edges with non-zero flow form a matching, they share no end-vertex.

The matching is maximum. A matching with more edges would correspond to a flow in S with higher value, which leads to a contradiction. □

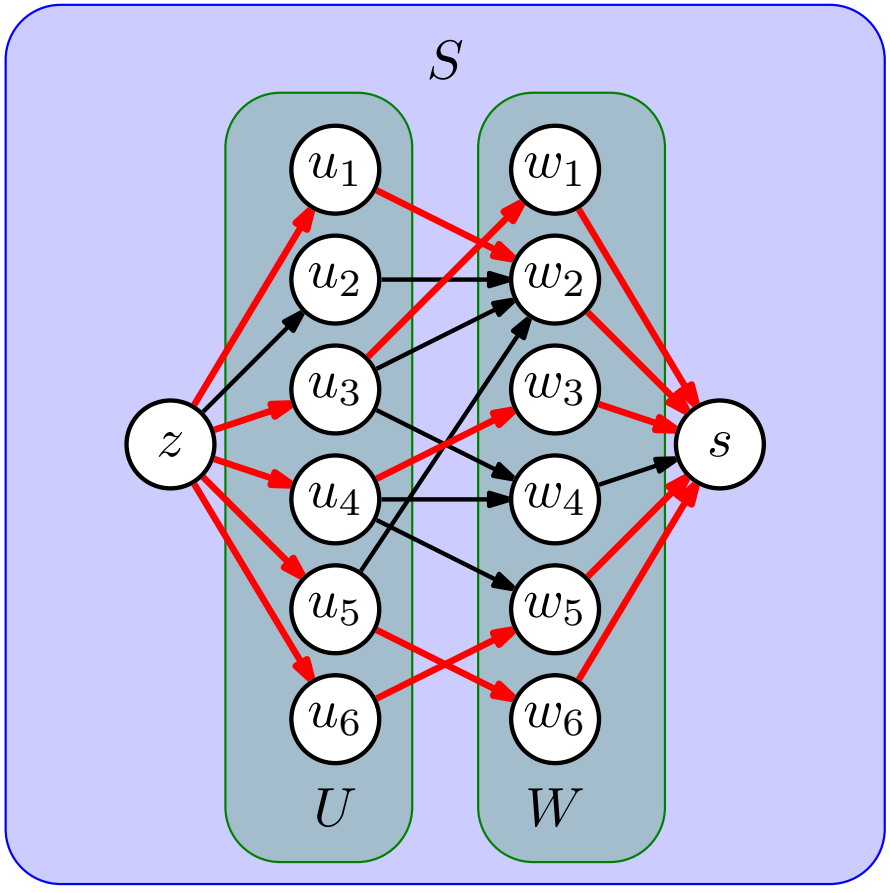


We are given a bipartite graph G.

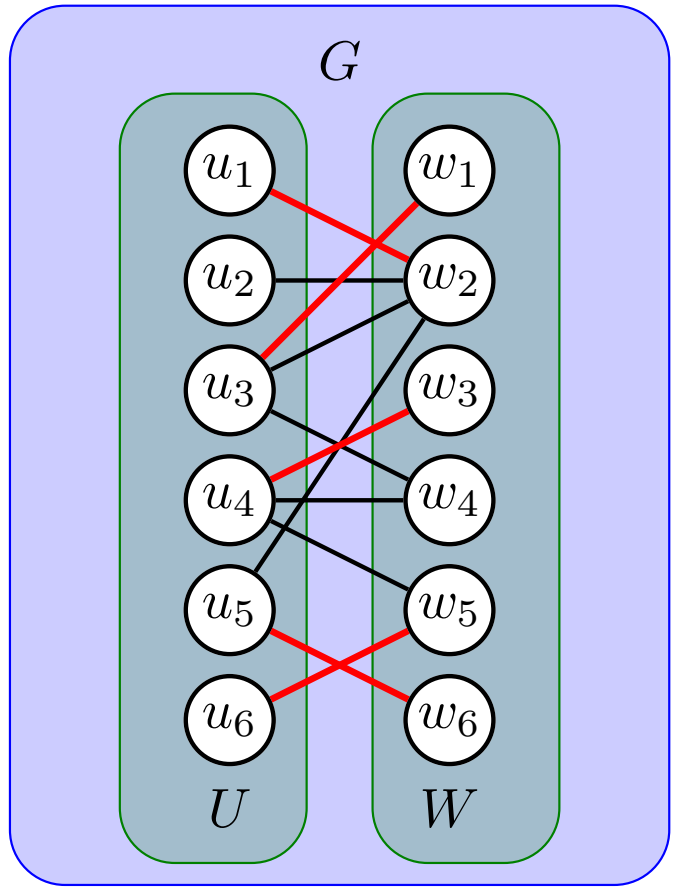


Let us construct the corresponding network S:

- we add a source vertex z and a sink vertex s,
- all vertices in U we join with z and all vertices in W we join with s,
- all edges of the network S are oriented from z to s; all capacities are 1.



Using the Algorithm we find the maximum flow in the network S. By Corollary all values of the maximum flow are integers.



The maximum matching contains just those edges of G with non-zero flow.

k-connectivity

Earlier we defined k-connectivity of graphs. Without proof we stater Menger's Theorem:

Theorem (Menger's theorem)

Graph G is k-edge connected if and only if there are at least k edge-disjoint paths between any two vertices (the paths can share vertices).
Graph G is k-vertex connected if and only if there are at least k internally-disjoint paths between any two vertices (the paths share only end-vertices).

Now we prove the theorem using the algorithm for maximum flow in a network.

First we notice that by definition of (vertex) k-connectivity holds:

Lemma

Let u, v be two vertices in G and $k > 0$ a natural number. Then between u and v there exist in G at least k edge-disjoint paths if and only if after removing any $k-1$ edges remain u and v in the same component.

Proof

"$\Rightarrow$" Follows by the definition of edge k-connectivity of a graph.

"$\Leftarrow$" Let G be a graph and let u and v be any pair of vertices in G. Let vertex u be the source and v the sink, we assign capacity 1 to each edge. Using Ford-Fulkerson's algorithm find the maximum flow from u to v.

The value of the flow is at least k, otherwise the value of the minimum cut is smaller than k. By removing the cut-edges we get a disconnected graph, while there are less than k removed edges which is not possible.

Thus, the edges with flow 1 form different (edge-disjoint) paths from u to v. (The second Mengers Theorem is proven similarly.) □

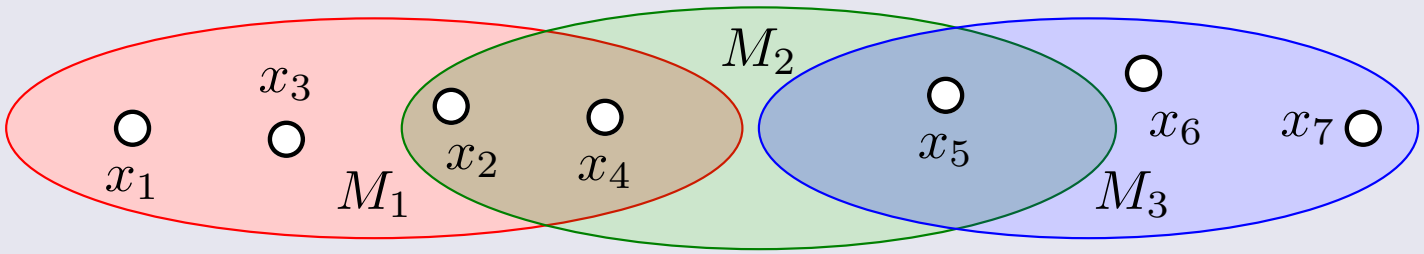
System of distinct representatives

Definition

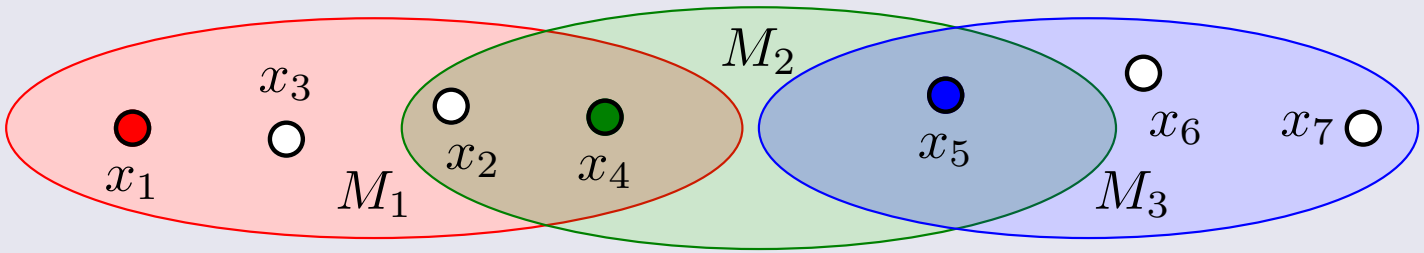
Let $M_1, M_2, \ldots, M_k$ be non-empty sets. System of distinct representatives for $M_1, M_2, \ldots, M_k$ is a sequence of *distinct* elements $(m_1, m_2, \ldots, m_k)$, such that $m_i \in M_i$ for $i = 1, 2, \ldots, k$.

Example

Find the distinct representatives for the given system.



One of the solutions.



An important and well known result is the following theorem

Marriage Theorem

Let $M_1, M_2, \ldots, M_k$ for $k > 0$ be non-empty sets. There exists a system of distinct representatives for these sets if and only if

$$\forall J \subset \{1, 2, \ldots, k\} : \left|\bigcup_{j \in J} M_j\right| \geq |J|,$$

meaning the union of and j sets in this system has at least j elements.

Marriage Theorem gives a sufficient and necessary condition for a set if distinct representatives to exist for a given set-system.

The find the system of representatives is difficult, yet sometimes by choosing a certain collection of sets one can disprove the existence of distinct representatives.

Note

Marriage Theorem is in some literature called „Hall's Theorem".

Proof of the Marriage Theorem

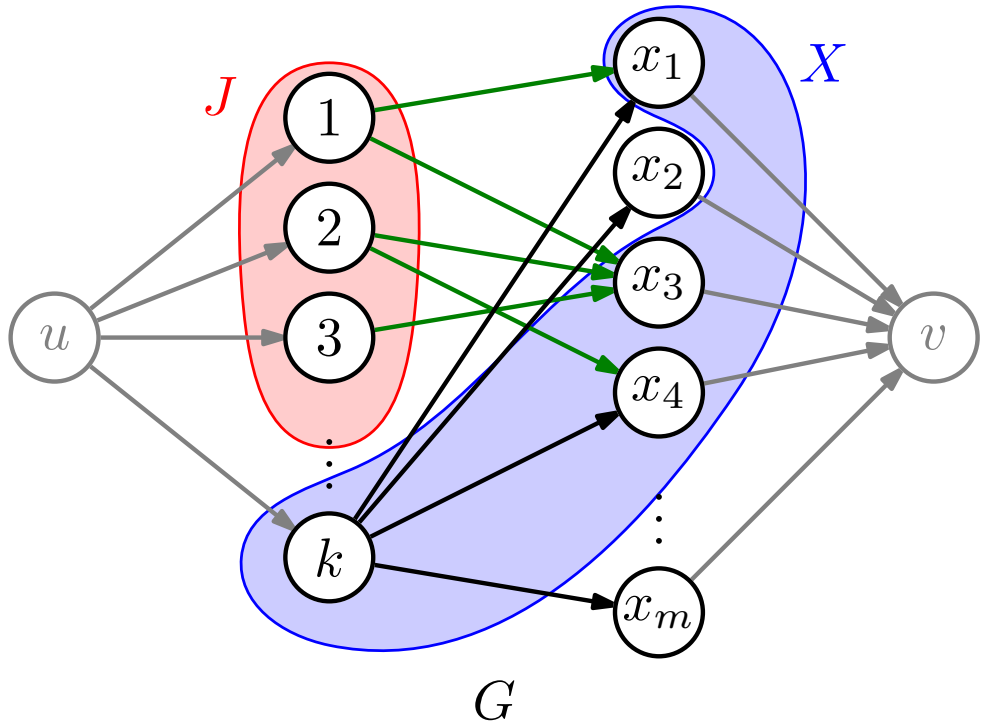
First implication "$\Rightarrow$" is straight forward. Idea of the proof of "$\Leftarrow$":

Denote by $x_1, x_2, \ldots, x_m$ all vertices in the union $M_1 \cup M_2 \cup \cdots \cup M_k$. We define a network S on the vertices $\{1, 2, \ldots, k\} \cup \{x_1, x_2, \ldots, x_m\} \cup \{u, v\}$. Moreover we add edges $\{u, i\}$ for $i = 1, 2, \ldots, k$, $\{x_j, v\}$ for $j = 1, 2, \ldots, m$ and edges $\{i, x_j\}$ for $x_j \in M_i$.

The construction of the network S is analogous as in Algorithm for maximum matching.
Each path from u to v is of the form u, i, x_j, v. It describes each representative $x_j \in M_i$ uniquely. The system of distinct representatives correspond to k vertex-disjoint paths from u to v.

Let X be any minimal subset of vertices in G, such that removing all vertices of X from G no path from u to v remains. By a lemma have all such sets a system of distinct representatives if and only if each such separating set X has et least k elements.

We define $J = \{1, 2, \ldots, k\} \setminus X$.



Now each edge leaving J (besides u) goes to vertices in $X \cap \{x_1, \ldots, x_m\}$, because no path from u to v exists. Thus

$$\left|\bigcup_{j \in J} M_j\right| = |X \cap \{x_1, \ldots, x_m\}| = |X| - |X \cap \{1, \ldots, k\}| = |X| - k + |J|.$$

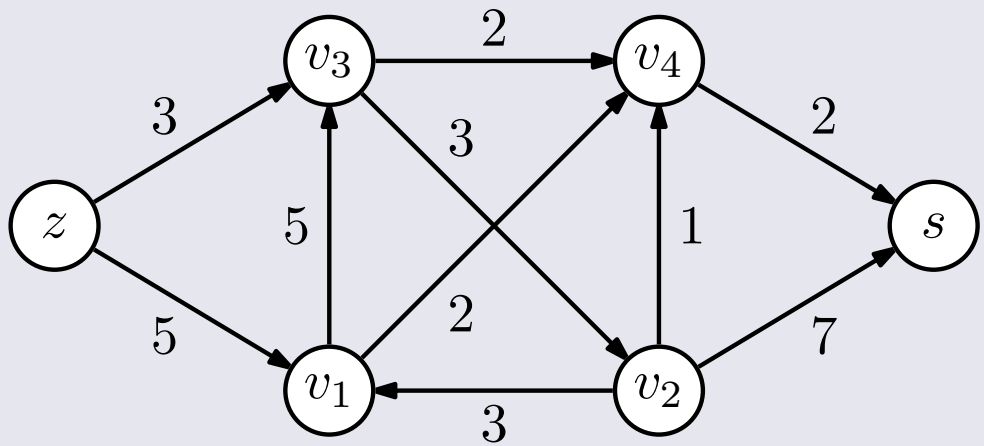
From this $|X| \geq k$ for all (minimal) separating sets X if and only if $\left|\bigcup_{j \in J} M_j\right| \geq |J|$ for all J. This completes the proof. $\square$

Example

We finish the lecture by providing a couple of examples on the maximum flow and minimum cut algorithm.

Example

What is the maximum flow in this network (G, z, s, w)?
And where is the minimum cut in the network?

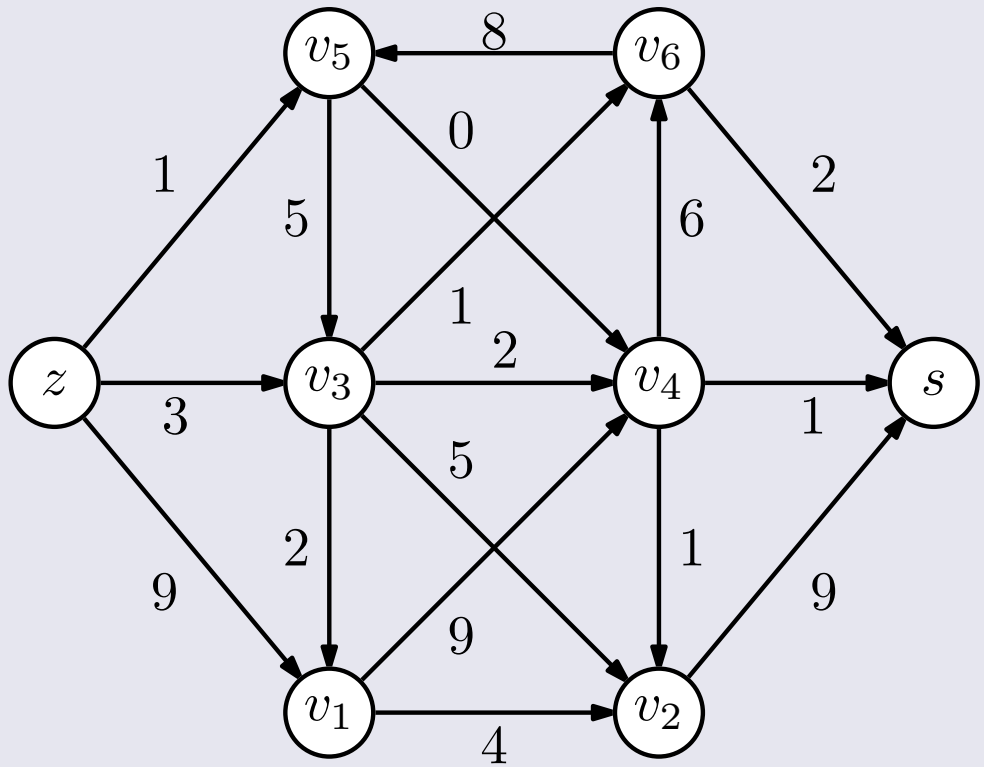


Network (G, z, s, w).

Last example

Example

What is the maximum flow in this network (G, z, s, w)?
And where is the minimum cut in the network?



Network (G, z, s, w).

The end

Thank you for your attention

Please do not forget about the evaluation.
Especially comments.

Exam dates

- “early” exam (?)

Good luck with your exam!