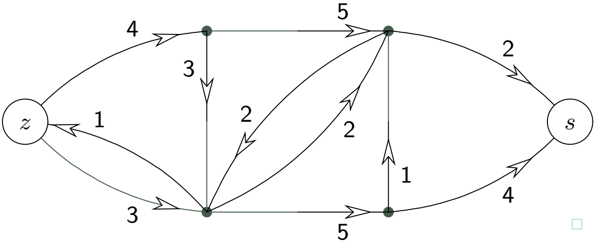
6 Network Flow Problems

Yet another area of rich applications of graphs (actually digraphs) deals with so called *networks* and "commodity flows" in them.

This time, the main optimization task is to maximize a *flow* from the designated *source* to the designated *sink*, respecting given constrains – *capacities* of network arcs.



Brief outline of this lecture

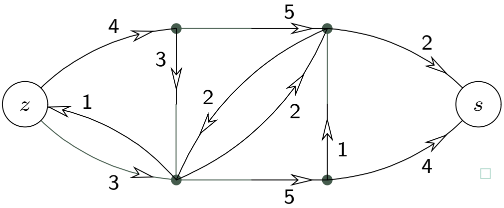
- Networks (weighted directed graphs) and flows in them.
- A network–flow algorithm(s) based on augmenting paths.
- Applications in connectivity, matching, and SDR problems.

6.1 Defining a flow network

The underlying structure of a flow network is a directed graph, with its vertices representing network nodes, and arcs representing the (existing or possible) connections between nodes. □

Definition 6.1. A flow network is a quadruple $S = (G, z, s, w)$ such that

- G is a digraph,
- the vertices $z \in V(G)$, $s \in V(G)$ are the *source* and the *sink*, respectively,
- and $w : E(G) \to \boldsymbol{R}^+$ is a positive weighting of the arcs (edges) of G, these weights are called *edge capacities*.



□

Remark: In reality, more than one source or sink may exist in a flow network, but that is not a problem — we simply create a single artificial source and draw arcs from it to all the real sources (even with source capacities!).

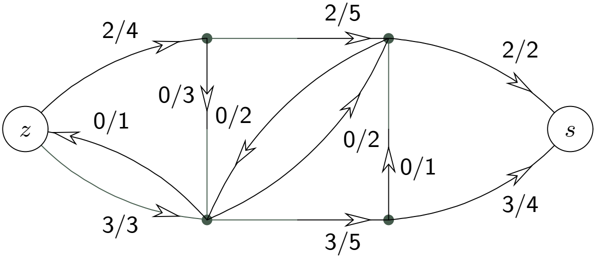
Notation: For simplicity, we shall write $e \to v$ to mean that an arc e "comes to" (has its head in) the vertex v, and $e \leftarrow v$ analogously for e "leaving" (having tail in) v. □

Definition 6.2. **A network flow,** in a flow network $S = (G, z, s, w)$, is an assignment $f : E(G) \to \boldsymbol{R}_0^+$ satisfying (we say f is *admissible*)

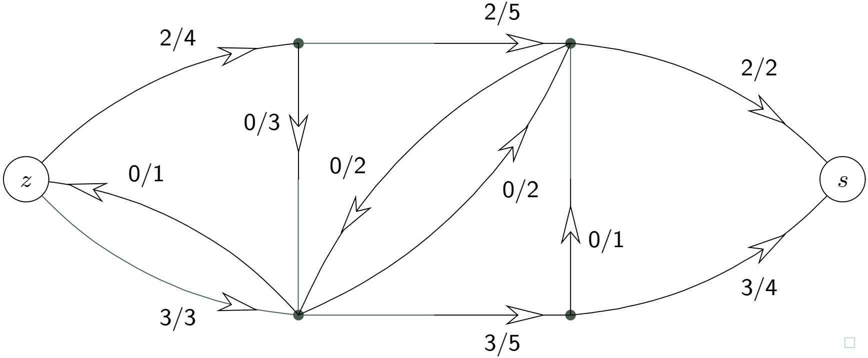
- $\forall e \in E(G) :\ 0 \le f(e) \le w(e)$,
- $\forall v \in V(G), v \neq z, s :\ \sum_{e \to v} f(e) = \sum_{e \leftarrow v} f(e)$. □

The *size (value)* of a flow f is the quantity $\|f\| = \sum_{e \leftarrow z} f(e) - \sum_{e \to z} f(e)$. □

Notation: The flow value F and the capacity C of an arc in a picture of a network will be shortly denoted by F/C, respectively.



So what is the value of the depicted flow? $\quad 5$



Remark: Notice the following simple identity

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

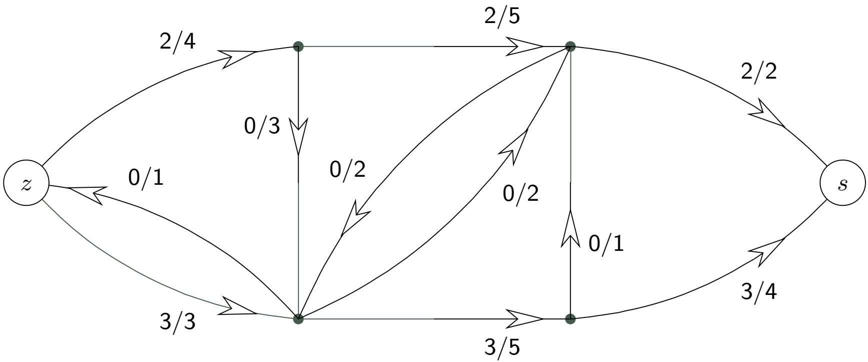
What interesting does it tell us? Briefly, that the (negative) flow value can be analogously defined at the sink s.

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

6.2 Finding the maximum flow value

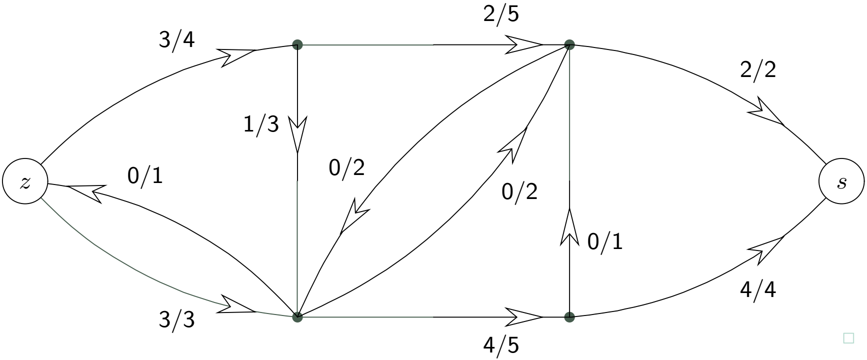
There exist quite simple and fast algorithms to determine the maximum flow value in a given network.

Problem 6.3. The Max–Flow problem
Given a flow network $S = (G, z, s, w)$, the task is to find a flow f in S from z to s such that the value $\|f\|$ is maximized (among all admissible flows in S). □



Is the depicted flow really maximum possible?

And what about this flow of value 6?



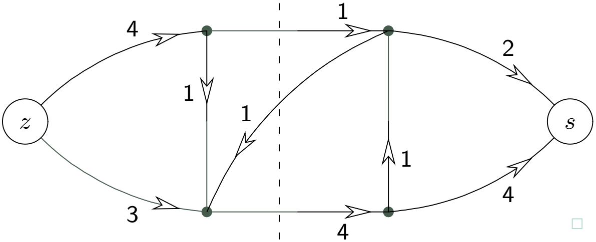
The main question is; how can we certify that no better flow exists in S?

Fortunately, a nice and simply understandable criterion exists there — notice that there is an "arc cut" between z and s of total value 6, and hence obviously there cannot be a larger flow!

Definition 6.4. **A cut** in a flow network $S = (G, z, s, w)$
is a subset of edges (arcs) $C \subset E(G)$ such that there is no $z \to s$ directed path in G completely avoiding C (i.e. a directed $z \to s$ path existing in $G - C$).
The *size* of a cut C is the sum of the capacities of arcs in C, i.e. $\|C\| = \sum_{e \in C} w(e)$. □

Theorem 6.5. *The maximum (admissible) flow value in a network S equals the minimum cut size in S.*

See the following example with a cut of size 5 (in the middle), and so the flow value 5 is maximum.



□

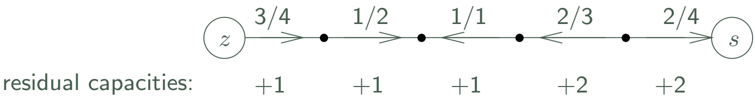
Remark: Theorem 6.5 is an example of a so called good characterization of a property — not having a larger flow can be certified by showing an obvious constrain, a cut of the corresponding size.

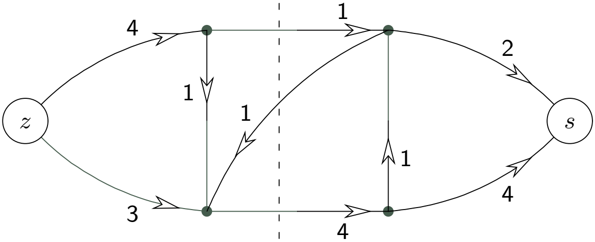
Residual and augmenting paths

Definition: Consider a flow network S and a flow f in it. A *residual z–s path* (in S w.r.t. f) is an undirected path in G from the source z to the sink s, i.e. a sequence of adjacent edges $e_1, e_2, \ldots, e_m$, such that $f(e_i) < w(e_i)$ if e_i is directed from z, and $f(e_i) > 0$ if e_i is directed from s. □

The quantity $w(e_i) - f(e_i)$, or $f(e_i)$, respectively, is called the *residual capacity* of the edge e_i. □

A residual path is that of strictly positive residual capacities. . .





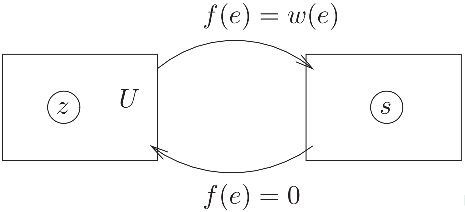
Algorithm 6.6. Ford–Fulkerson's for network flows.

`input` $<$ *flow network* $S=(G,z,s,w)$;
flow $f\equiv 0$;
`repeat` $\{$
    Search (BFS) the graph G *to find the set* U *of those vertices*
        accessible from the source z *along residual paths*;
    `if (` $s\in U$ `)` $\{$
        `P =` *any residual* z–s *path in* S *(this* P *then called an augmenting path)*;
        Augment ("enlarge") f *by the minimal residual capacity along* P;
    $\}$
`until (` $s\notin U$ `);`
`output` $>$ *a maximum flow* f *in* S;
`output` $>$ *a minimum cut in* S *from* U *to* $V(G)-U$.

Proof of Algorithm 6.6:

For any flow f and any cut C in S, it holds $\|f\| \leq \|C\|$. If the algorithm stops with a flow f in S and a cut C such that $\|C\| = \|f\|$, then it is clear that f is a maximum flow in S. (We have, however, not proved yet that the algorithm stops!) □

So to prove that whenever the algorithm stops with f, C, then $\|f\| = \|C\|$, we use the following schematic picture (in which s does not belong to the "accessible" set U):



Since no further vertex than U is accessible along residual paths, every arc e leaving U has full flow $f(e) = w(e)$, and every arc e entering U has zero flow $f(e) = 0$. Therefore;

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

That is nice, and it remains to argue that $\|C\| = \sum_{e \leftarrow U} f(e) - \sum_{e \rightarrow U} f(e) = \|f\|$, finishing the proof. □

The proof of Algorithm 6.6 shows several interesting things:

Fact: If we can prove that Algorithm 6.6 stops, we prove also Theorem 6.5. □

Fact: If the edge capacities in S are integral, then Algorithm 6.6 always stops.

Corollary 6.7. *If the edge capacities in S are integral, then Algorithm 6.6 outputs an integral flow.* □

Improved flow algorithms

Generally, one cannot guarantee that Algorithm 6.6 stops, since counterexamples exist with real capacities, but we can do better as follows. □

Algorithm 6.8. Edmonds–Karp's for network flows.
As in Algorithm 6.6, we always enlarge one of the shortest residual paths in G (i.e. find the path P using BFS, cf. Corollary 3.4).

This implementation is guaranteed to stop after $O(|V(G)| \cdot |E(G)|)$ iterations, so in total computing time $O(|V(G)| \cdot |E(G)|^2)$.

Even better, we can use the following “clever” algorithms.

Algorithm 6.9. **Dinitz’s for network flows** (a sketch).
We modify Algorithm 6.6 with the following iteration:

- *Using BFS, we find all the shortest residual paths in S, creating a “layered” residual network.*
- *The layered network is then completely saturated in one run.*

This implementation makes only $O(|V(G)|)$ iterations of the main cycle. Total computing time now is $O(|V(G)|^2 \cdot |E(G)|)$. □

Algorithm 6.10. **MPM “Three Indians”** (a sketch).
Same as Algorithm 6.9, except that a layered network is saturated faster, and the total computing time is $O(|V(G)|^3)$.

6.3 Generalized network flow settings

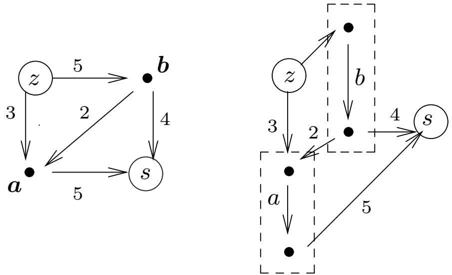
Among the many ways flow networks can be generalized, we briefly mention three which are interesting and often used.

Networks with vertex capacities

In a flow network with *vertex capacities* (of course, retaining edge capacities as well), the weight function is $w : E(G) \cup V(G) \to \boldsymbol{R}^+$.

The meaning for admissible flows is that the total sum of incoming flow to any vertex x is not more than $w(x)$. (Differently applicable to the source or the sink, though. . .)

Fact. Such a generalized flow network can easily be translated to an ordinary network via "doubling" the capacitated vertices (replacing them with new arcs between the two copies), as follows.



Networks with lower capacities

In a flow network with *lower capacities*, in addition to the weight function w, there is another weight function $\ell : E(G) \to \boldsymbol{R}_0^+$ giving the lower edge capacities.

A flow f is admissible in such a network if $\ell(e) \leq f(e) \leq w(e)$ for every edge e of the network. □ Notice that an admissible flow may not exist in such a lower-capacitated network. □

Algorithm 6.11. Flows in lower-capacitated network

For this kind of a flow network, the solution has two steps.

– *First, an admissible circulation is found (with a "back-arc" sz), respecting both the lower an upper bounds ℓ, w. This is done by finding a maximum flow in an artificial network modelling the "surplus" of lower capacities at every vertex...*

– *Second, this admissible circulation is enlarged by a maximum possible excessive flow from z to s (the capacities are now $w(e) - r(e)$ where r is the circulation found above).* □

Multicommodity flows

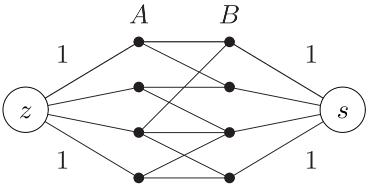
This is a difficult problem setting reaching beyond the scope of our lecture.

6.4 Other applications of network flows

Bipartite matching

Definition: A *matching* in a graph G (bipartite here) is a subset of edges $M \subset E(G)$ such that no two edges from M share a vertex. □

Algorithm 6.12. Finding maximum matching in a bipartite graph
Given a bipartite graph G with the vertex parts A, B, we construct the following flow network S:



All the edges are implicitly directed from the source towards the sink, and all capacities are 1. We run Algorithm 6.6, and form the resulting matching M by those edges of G having nonzero flow at the end. □

Proof of Algorithm 6.12: By Corollary 6.7, the maximum flow found by our algorithm is integral, which now means that each flow unit is 0 or 1. Hence no two edges of M could share a vertex. Conversely, any matching M' gives an admissible flow in S. □

Higher graph connectivity

Let us consider a graph G as a generalized (symmetrically–oriented) network with all vertex capacities equal to 1. Then the network flow theorem immediately says:

Corollary 6.13. *Let u, v be two vertices of G and $k > 0$ be an integer. Then there exists at least k internally disjoint u–v paths in G if, and only if, removing any subset of at most $k-1$ vertices of G (other than u, v) does not disconnect u from v.* □

This statement immediately implies Theorem 2.6 (Menger's)!

Systems of distinct representatives (SDR)

Definition: Let $M_1, M_2, \ldots, M_k$ be a collection of nonempty sets. A *system of distinct representatives (SDR)* of the set family $\{M_1, M_2, \ldots, M_k\}$ is a sequence of pairwise distinct elements $(x_1, x_2, \ldots, x_k)$ such that $x_i \in M_i$ for $i = 1, 2, \ldots, k$. □

Theorem 6.14. (Hall) *Let $\{M_1, M_2, \ldots, M_k\}$ be a family of nonempty sets. Then there exists a system of its distinct representatives if, and only if,*

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

i.e., the union of any subfamily of these sets has at least that many elements as the number of sets in it.

Necessity of Hall's condition in this theorem is obvious, and its sufficiency can be proved by an application of network flows again.