8 Network Flows

In this and the next chapter we consider flows in networks. We have a digraph G with edge capacities $u : E(G) \to \mathbb{R}_+$ and two specified vertices s (the **source**) and t (the **sink**). The quadruple (G, u, s, t) is sometimes called a **network**.

Our main motivation is to transport as many units as possible simultaneously from s to t. A solution to this problem will be called a maximum flow. Formally we define:

Definition 8.1. *Given a digraph G with capacities $u : E(G) \to \mathbb{R}_+$, a* **flow** *is a function $f : E(G) \to \mathbb{R}_+$ with $f(e) \leq u(e)$ for all $e \in E(G)$. The* **excess** *of a flow f at $v \in V(G)$ is*

$$\mathrm{ex}_f(v) := \sum_{e \in \delta^-(v)} f(e) - \sum_{e \in \delta^+(v)} f(e).$$

We say that f satisfies the **flow conservation rule** *at vertex v if $\mathrm{ex}_f(v) = 0$. A flow satisfying the flow conservation rule at every vertex is called a* **circulation**.

Now given a network (G, u, s, t), an ***s-t*-flow** *is a flow f satisfying $\mathrm{ex}_f(s) \leq 0$ and $\mathrm{ex}_f(v) = 0$ for all $v \in V(G) \setminus \{s, t\}$. We define the* **value** *of an s-t-flow f by* $\mathrm{value}(f) := -\mathrm{ex}_f(s)$.

Now we can formulate the basic problem of this chapter:

MAXIMUM FLOW PROBLEM	
Instance:	A network (G, u, s, t).
Task:	Find an s-t-flow of maximum value.

It causes no loss of generality to assume that G is simple as parallel edges can be united beforehand.

This problem has numerous applications. For example, consider the JOB ASSIGNMENT PROBLEM: given n jobs, their processing times $t_1, \ldots, t_n \in \mathbb{R}_+$ and a nonempty subset $S_i \subseteq \{1, \ldots, m\}$ of employees that can contribute to each job $i \in \{1, \ldots, n\}$, we ask for numbers $x_{ij} \in \mathbb{R}_+$ for all $i = 1, \ldots, n$ and $j \in S_i$ (meaning how long employee j works on job i) such that all jobs are finished, i.e. $\sum_{j \in S_i} x_{ij} = t_i$ for $i = 1, \ldots, n$. Our goal was to minimize the amount of time in which all jobs are done, i.e. $T(x) := \max_{j=1}^{m} \sum_{i : j \in S_i} x_{ij}$. Instead of solving this problem with LINEAR PROGRAMMING we look for a combinatorial algorithm.

B. Korte and J. Vygen *Combinatorial Optimization*,
Algorithms and Combinatorics 21, DOI 10.1007/978-3-642-24488-9_8,

We apply binary search for the optimum $T(x)$. Then for one specific value T we have to find numbers $x_{ij} \in \mathbb{R}_+$ with $\sum_{j \in S_i} x_{ij} = t_i$ for all i and $\sum_{i:j \in S_i} x_{ij} \leq T$ for all j. We model the sets S_i by a (bipartite) digraph with a vertex v_i for each job i, a vertex w_j for each employee j and an edge (v_i, w_j) whenever $j \in S_i$. We introduce two additional vertices s and t and edges (s, v_i) for all i and (w_j, t) for all j. Let this graph be G. We define capacities $u : E(G) \to \mathbb{R}_+$ by $u((s, v_i)) := t_i$ and $u(e) := T$ for all other edges. Then the feasible solutions x with $T(x) \leq T$ evidently correspond to the s-t-flows of value $\sum_{i=1}^{n} t_i$ in (G, u). Indeed, these are maximum flows.

In Section 8.1 we describe a basic algorithm for the MAXIMUM FLOW PROBLEM and use it to prove the Max-Flow-Min-Cut Theorem, one of the best-known results in combinatorial optimization, which shows the relation to the problem of finding a minimum capacity s-t-cut. Moreover we show that, for integral capacities, there always exists an optimum flow which is integral. The combination of these two results also implies Menger's Theorem on disjoint paths as we discuss in Section 8.2.

Sections 8.3, 8.4 and 8.5 contain efficient algorithms for the MAXIMUM FLOW PROBLEM. Then we shift attention to the problem of finding minimum cuts. Section 8.6 describes an elegant way to store the minimum capacity of an s-t-cut (which equals the maximum value of an s-t-flow) for all pairs of vertices s and t. Section 8.7 shows how the edge-connectivity, or a minimum capacity cut in an undirected graph, can be determined more efficiently than by applying several network flow computations.

8.1 Max-Flow-Min-Cut Theorem

The definition of the MAXIMUM FLOW PROBLEM suggests the following LP formulation:

$$\begin{aligned}
\max \quad & \sum_{e \in \delta^+(s)} x_e - \sum_{e \in \delta^-(s)} x_e & & \\
\text{s.t.} \quad & \sum_{e \in \delta^-(v)} x_e = \sum_{e \in \delta^+(v)} x_e & & (v \in V(G) \setminus \{s, t\}) \\
& x_e \leq u(e) & & (e \in E(G)) \\
& x_e \geq 0 & & (e \in E(G))
\end{aligned}$$

Since this LP is obviously bounded and the zero flow $f \equiv 0$ is always feasible, we have the following :

Proposition 8.2. *The* MAXIMUM FLOW PROBLEM *always has an optimum solution.* □

Furthermore, by Theorem 4.18 there exists a polynomial-time algorithm. However, we are not satisfied with this, but will rather look for a combinatorial algorithm (not using Linear Programming).

Recall that an s-t-cut in G is an edge set $\delta^+(X)$ with $s \in X$ and $t \in V(G) \setminus X$. The **capacity** of an s-t-cut is the sum of the capacities of its edges. By a minimum s-t-cut in (G,u) we mean an s-t-cut of minimum capacity (with respect to u) in G.

Lemma 8.3. *For any $A \subseteq V(G)$ such that $s \in A, t \notin A$, and any s-t-flow f,*

(a) $\text{value}(f) = \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e)$.
(b) $\text{value}(f) \leq \sum_{e \in \delta^+(A)} u(e)$.

Proof: (a): Since the flow conservation rule holds for $v \in A \setminus \{s\}$,

$$\begin{aligned}\text{value}(f) &= \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e) \\ &= \sum_{v \in A} \left(\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) \right) \\ &= \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e).\end{aligned}$$

(b): This follows from (a) by using $0 \leq f(e) \leq u(e)$ for $e \in E(G)$. □

In other words, the value of a maximum flow cannot exceed the capacity of a minimum s-t-cut. In fact, we have equality here. To see this, we need the concept of augmenting paths which will reappear in several other chapters.

Definition 8.4. *For a digraph G we define $\overleftrightarrow{G} := \left(V(G), E(G) \,\dot\cup\, \left\{\overleftarrow{e} : e \in E(G)\right\}\right)$, where for $e = (v,w) \in E(G)$ we define $\overleftarrow{e}$ to be a new edge from w to v. We call $\overleftarrow{e}$ the* **reverse edge** *of e and vice versa. Note that if $e = (v,w), e' = (w,v) \in E(G)$, then $\overleftarrow{e}$ and e' are two distinct parallel edges in $\overleftrightarrow{G}$.*

Given a digraph G with capacities $u : E(G) \to \mathbb{R}_+$ and a flow f, we define **residual capacities** *$u_f : E(\overleftrightarrow{G}) \to \mathbb{R}_+$ by $u_f(e) := u(e) - f(e)$ and $u_f(\overleftarrow{e}) := f(e)$ for all $e \in E(G)$. The* **residual graph** *G_f is the graph $\left(V(G), \left\{e \in E\left(\overleftrightarrow{G}\right) : u_f(e) > 0\right\}\right)$.*

Given a flow f and a path (or circuit) P in G_f, to **augment** *f along P by γ means to do the following for each $e \in E(P)$: if $e \in E(G)$ then increase $f(e)$ by γ, otherwise – if $e = \overleftarrow{e_0}$ for $e_0 \in E(G)$ – decrease $f(e_0)$ by γ.*

Given a network (G,u,s,t) and an s-t-flow f, an **f-augmenting path** *is an s-t-path in the residual graph G_f.*

Using this concept, the following algorithm for the MAXIMUM FLOW PROBLEM, due to Ford and Fulkerson [1957], is natural. We first restrict ourselves to integral capacities.

FORD-FULKERSON ALGORITHM

Input: A network (G, u, s, t) with $u : E(G) \to \mathbb{Z}_+$.

Output: An s-t-flow f of maximum value.

① Set $f(e) := 0$ for all $e \in E(G)$.

② Find an f-augmenting path P. **If** none exists **then stop**.

③ Compute $\gamma := \min_{e \in E(P)} u_f(e)$. Augment f along P by γ and **go to** ②.

Edges where the minimum in ③ is attained are sometimes called bottleneck edges. The choice of γ guarantees that f continues to be a flow. Since P is an s-t-path, the flow conservation rule is preserved at all vertices except s and t.

To find an augmenting path is easy (we just have to find any s-t-path in G_f). However, we should be careful how to do this. In fact, if we allow irrational capacities (and have bad luck when choosing the augmenting paths), the algorithm might not terminate at all (Exercise 2).

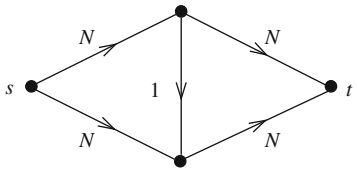

Fig. 8.1.

Even in the case of integer capacities, we may have an exponential number of augmentations. This is illustrated by the simple network shown in Figure 8.1, where the numbers are the edge capacities ($N \in \mathbb{N}$). If we choose an augmenting path of length 3 in each iteration, we can augment the flow by just one unit each time, so we need $2N$ iterations. Observe that the input length is $O(\log N)$, since capacities are of course encoded in binary form. We shall overcome these problems in Section 8.3.

We now claim that when the algorithm stops, then f is indeed a maximum flow:

Theorem 8.5. *An s-t-flow f is maximum if and only if there is no f-augmenting path.*

Proof: If there is an augmenting path P, then ③ of the FORD-FULKERSON ALGORITHM computes a flow of greater value, so f is not maximum. If there is no augmenting path, this means that t is not reachable from s in G_f. Let R be the set of vertices reachable from s in G_f. By the definition of G_f, we have $f(e) = u(e)$ for all $e \in \delta_G^+(R)$ and $f(e) = 0$ for all $e \in \delta_G^-(R)$.

Now Lemma 8.3 (a) says that

$$\text{value}(f) = \sum_{e \in \delta_G^+(R)} u(e)$$

which by Lemma 8.3 (b) implies that f is a maximum flow. □

In particular, for any maximum s-t-flow we have an s-t-cut whose capacity equals the value of the flow. Together with Lemma 8.3 (b) this yields the central result of network flow theory, the Max-Flow-Min-Cut Theorem:

Theorem 8.6. (Ford and Fulkerson [1956], Dantzig and Fulkerson [1956]) *In a network the maximum value of an s-t-flow equals the minimum capacity of an s-t-cut.* □

An alternative proof was proposed by Elias, Feinstein and Shannon [1956]. The Max-Flow-Min-Cut Theorem also follows quite easily from LP duality; see Exercise 9 of Chapter 3.

If all capacities are integers, γ in ③ of the FORD-FULKERSON ALGORITHM is always integral. Since there is a maximum flow of finite value (Proposition 8.2), the algorithm terminates after a finite number of steps. Therefore we have the following important consequence:

Corollary 8.7. (Dantzig and Fulkerson [1956]) *If the capacities of a network are integers, then there exists an integral maximum flow.* □

This corollary – sometimes called the Integral Flow Theorem – can also be proved easily by using the total unimodularity of the incidence matrix of a digraph (Exercise 3).

We close this section with another easy but useful observation, the Flow Decomposition Theorem:

Theorem 8.8. (Gallai [1958], Ford and Fulkerson [1962]) *Let (G, u, s, t) be a network and let f be an s-t-flow in G. Then there exists a family $\mathcal{P}$ of s-t-paths and a family $\mathcal{C}$ of circuits in G along with weights $w : \mathcal{P} \cup \mathcal{C} \to \mathbb{R}_+$ such that $f(e) = \sum_{P \in \mathcal{P} \cup \mathcal{C}: e \in E(P)} w(P)$ for all $e \in E(G)$, $\sum_{P \in \mathcal{P}} w(P) = \text{value}(f)$, and $|\mathcal{P}| + |\mathcal{C}| \leq |E(G)|$.*

Moreover, if f is integral then w can be chosen to be integral.

Proof: We construct $\mathcal{P}$, $\mathcal{C}$ and w by induction on the number of edges with nonzero flow. Assume that there is an edge e with $f(e) > 0$; otherwise the statement is trivial. Consider a maximal walk W containing e in which every edge carries positive flow and no vertex appears twice except possibly if it is one of the endpoints. W contains at most n edges. Moreover, W contains a circuit P, or W is a path P. In the latter case, P begins at s, ends at t, and $f(\delta^-(s)) = f(\delta^+(t)) = 0$ (due to the maximality of W and the flow conservation rule).

Let $w(P) := \min_{e \in E(P)} f(e)$. Set $f'(e) := f(e) - w(P)$ for $e \in E(P)$ and $f'(e) := f(e)$ for $e \notin E(P)$. An application of the induction hypothesis to f' completes the proof. □

The proof also leads to an $O(mn)$-time algorithm for computing such a flow decomposition.

8.2 Menger's Theorem

Consider Corollary 8.7 and Theorem 8.8 in the special case where all capacities are 1. Here integral s-t-flows can be regarded as collections of edge-disjoint s-t-paths and circuits. We obtain the following important theorem:

Theorem 8.9. (Menger [1927]) *Let G be a graph (directed or undirected), let s and t be two vertices, and $k \in \mathbb{N}$. Then there are k edge-disjoint s-t-paths if and only if after deleting any $k-1$ edges t is still reachable from s.*

Proof: Necessity is obvious. To prove sufficiency in the directed case, let (G, u, s, t) be a network with unit capacities $u \equiv 1$ such that t is reachable from s even after deleting any $k-1$ edges. This implies that the minimum capacity of an s-t-cut is at least k. By the Max-Flow-Min-Cut Theorem 8.6 and Corollary 8.7 there is an integral s-t-flow of value at least k. By Theorem 8.8 this flow can be decomposed into integral flows on s-t-paths (and possibly some circuits). Since all capacities are 1 we must have at least k edge-disjoint s-t-paths.

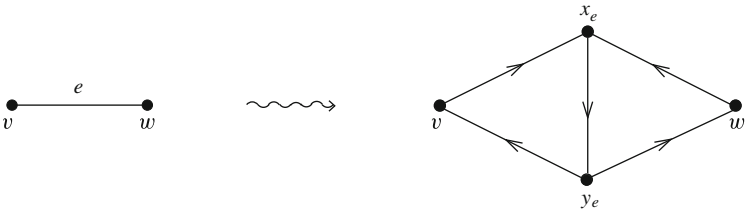

Fig. 8.2.

To prove sufficiency in the undirected case, let G be an undirected graph with two vertices s and t such that t is reachable from s even after deleting any $k-1$ edges. This property obviously remains true if we replace each undirected edge $e = \{v, w\}$ by five directed edges (v, x_e), (w, x_e), (x_e, y_e), (y_e, v), (y_e, w) where x_e and y_e are new vertices (see Figure 8.2). Now we have a digraph G' and, by the first part, k edge-disjoint s-t-paths in G'. These can be easily transformed to k edge-disjoint s-t-paths in G. □

In turn it is easy to derive the Max-Flow-Min-Cut Theorem (at least for rational capacities) from Menger's Theorem. We now consider the vertex-disjoint version of

Menger's Theorem. We call a set of paths **internally disjoint** if no two of them have an edge or an internal vertex in common. Although they may share endpoints, internally disjoint paths are sometimes also called vertex-disjoint (if the set of endpoints is given).

Theorem 8.10. (Menger [1927]) *Let G be a graph (directed or undirected), let s and t be two non-adjacent vertices, and $k \in \mathbb{N}$. Then there are k pairwise internally disjoint s-t-paths if and only if after deleting any $k-1$ vertices (distinct from s and t) t is still reachable from s.*

Proof: Necessity is again trivial. Sufficiency in the directed case follows from the directed part of Theorem 8.9 by the following elementary construction: we replace each vertex v of G by two vertices v' and v'' and an edge (v', v''). Each edge (v, w) of G is replaced by (v'', w'). Any set of $k-1$ edges in the new graph G' whose deletion makes t' unreachable from s'' implies a set of at most $k-1$ vertices in G, containing neither s nor t, whose deletion makes t unreachable from s. Moreover, edge-disjoint s''-t'-paths in the new graph correspond to internally disjoint s-t-paths in the old one.

The undirected version follows from the directed one by the same construction as in the proof of Theorem 8.9 (Figure 8.2). □

The following corollary is an important consequence of Menger's Theorem:

Corollary 8.11. (Whitney [1932]) *An undirected graph G with at least two vertices is k-edge-connected if and only if for each pair $s, t \in V(G)$ with $s \neq t$ there are k edge-disjoint s-t-paths.*

An undirected graph G with more than k vertices is k-connected if and only if for each pair $s, t \in V(G)$ with $s \neq t$ there are k internally disjoint s-t-paths.

Proof: The first statement follows directly from Theorem 8.9.

To prove the second statement let G be an undirected graph with more than k vertices. If G has $k-1$ vertices whose deletion makes the graph disconnected, then it cannot have k internally disjoint s-t-paths for each pair $s, t \in V(G)$.

Conversely, if G does not have k internally disjoint s-t-paths for some $s, t \in V(G)$, then we consider two cases. If s and t are non-adjacent, then by Theorem 8.10 G has $k-1$ vertices whose deletion separates s and t.

If s and t are joined by a set F of parallel edges, $|F| \geq 1$, then $G - F$ has no $k-|F|$ internally disjoint s-t-paths, so by Theorem 8.10 it has a set X of $k-|F|-1$ vertices whose deletion separates s and t. Let $v \in V(G) \setminus (X \cup \{s, t\})$. Then v cannot be reachable from s and from t in $(G-F)-X$, say v is not reachable from s. Then v and s are in different connected components of $G - (X \cup \{t\})$. □

In many applications one looks for edge-disjoint or vertex-disjoint (or internally disjoint) paths between several pairs of vertices. The four versions of Menger's Theorem (directed and undirected, vertex- and edge-disjoint) correspond to four versions of the DISJOINT PATHS PROBLEM:

DIRECTED/UNDIRECTED EDGE-/VERTEX-DISJOINT PATHS PROBLEM	
Instance:	Two directed/undirected graphs (G, H) on the same vertices.
Task:	Find a family $(P_f)_{f \in E(H)}$ of edge-disjoint/internally disjoint paths in G such that for each $f = (t, s)$ or $f = \{t, s\}$ in H, P_f is an s-t-path.

Such a family is called a **solution** of (G, H). We say that P_f **realizes** f. The edges of G are called **supply edges**, the edges of H **demand edges**. A vertex incident to some demand edge is called a **terminal**.

Above we considered the special case when H is just a set of k parallel edges. The general DISJOINT PATHS PROBLEM will be discussed in Chapter 19. Here we only note the following useful special case of Menger's Theorem:

Proposition 8.12. *Let (G, H) be an instance of the* DIRECTED EDGE-DISJOINT PATHS PROBLEM *where H is just a set of parallel edges and $G + H$ is Eulerian. Then (G, H) has a solution.*

Proof: Since $G + H$ is Eulerian, every edge, in particular any $f \in E(H)$, belongs to some circuit C. We take $C - f$ as the first path of our solution, delete C, and apply induction. □

8.3 The Edmonds-Karp Algorithm

In Exercise 2 it is shown that it is necessary to make ② of the FORD-FULKERSON ALGORITHM more precise. Instead of choosing an arbitrary augmenting path it is a good idea to look for a shortest one, i.e. an augmenting path with a minimum number of edges. With this simple idea Edmonds and Karp [1972] obtained the first polynomial-time algorithm for the MAXIMUM FLOW PROBLEM.

EDMONDS-KARP ALGORITHM	
Input:	A network (G, u, s, t).
Output:	An s-t-flow f of maximum value.

① Set $f(e) := 0$ for all $e \in E(G)$.

② Find a shortest f-augmenting path P. **If** there is none **then stop**.

③ Compute $\gamma := \min_{e \in E(P)} u_f(e)$. Augment f along P by γ and **go to** ②.

This means that ② of the FORD-FULKERSON ALGORITHM should be implemented by BFS (see Section 2.3).

Lemma 8.13. *Let $f_1, f_2, \ldots$ be a sequence of flows such that f_{i+1} results from f_i by augmenting along P_i, where P_i is a shortest f_i-augmenting path. Then*

(a) $|E(P_k)| \leq |E(P_{k+1})|$ *for all* k.
(b) $|E(P_k)|+2 \leq |E(P_l)|$ *for all* $k < l$ *such that* $P_k \cup P_l$ *contains a pair of reverse edges.*

Proof: (a): Consider the graph G_1 which results from $P_k \dot{\cup} P_{k+1}$ by deleting pairs of reverse edges. (Edges appearing both in P_k and P_{k+1} appear twice in G_1.) Every simple subgraph of G_1 is a subgraph of G_{f_k}, since any edge in $E(G_{f_{k+1}}) \setminus E(G_{f_k})$ must be the reverse of an edge in P_k.

Let H_1 simply consist of two copies of (t, s). Obviously $G_1 + H_1$ is Eulerian. Thus by Proposition 8.12 there are two edge-disjoint s-t-paths Q_1 and Q_2. Since $E(G_1) \subseteq E(G_{f_k})$, both Q_1 and Q_2 are f_k-augmenting paths. Since P_k was a shortest f_k-augmenting path, $|E(P_k)| \leq |E(Q_1)|$ and $|E(P_k)| \leq |E(Q_2)|$. Thus,

$$2|E(P_k)| \leq |E(Q_1)| + |E(Q_2)| \leq |E(G_1)| \leq |E(P_k)| + |E(P_{k+1})|,$$

implying $|E(P_k)| \leq |E(P_{k+1})|$.

(b): By part (a) it is enough to prove the statement for those k, l such that for $k < i < l$, $P_i \cup P_l$ contains no pair of reverse edges.

As above, consider the graph G_1 which results from $P_k \dot{\cup} P_l$ by deleting pairs of reverse edges. Again, we claim that every simple subgraph of G_1 is a subgraph of G_{f_k}. To see this, observe that $E(P_k) \subseteq E(G_{f_k})$, $E(P_l) \subseteq E(G_{f_l})$, and any edge of $E(G_{f_l}) \setminus E(G_{f_k})$ must be the reverse of an edge in one of $P_k, P_{k+1}, \ldots, P_{l-1}$. But – due to the choice of k and l – among these paths only P_k contains the reverse of an edge in P_l.

Let H_1 again consist of two copies of (t, s). Since $G_1 + H_1$ is Eulerian, Proposition 8.12 guarantees that there are two edge-disjoint s-t-paths Q_1 and Q_2. Again Q_1 and Q_2 are both f_k-augmenting. Since P_k was a a shortest f_k-augmenting path, $|E(P_k)| \leq |E(Q_1)|$ and $|E(P_k)| \leq |E(Q_2)|$. We conclude that

$$2|E(P_k)| \leq |E(Q_1)| + |E(Q_2)| \leq |E(P_k)| + |E(P_l)| - 2$$

(since we have deleted at least two edges). This completes the proof. □

Theorem 8.14. (Edmonds and Karp [1972]) *Regardless of the edge capacities, the* EDMONDS-KARP ALGORITHM *stops after at most* $\frac{mn}{2}$ *augmentations, where* m *and* n *denote the number of edges and vertices, respectively.*

Proof: Let $P_1, P_2, \ldots$ be the augmenting paths chosen during the EDMONDS-KARP ALGORITHM. By the choice of γ in ③ of the algorithm, each augmenting path contains at least one bottleneck edge.

For any edge e, let $P_{i_1}, P_{i_2}, \ldots$ be the subsequence of augmenting paths containing e as a bottleneck edge. Obviously, between P_{i_j} and $P_{i_{j+1}}$ there must be an augmenting path P_k ($i_j < k < i_{j+1}$) containing $\overleftarrow{e}$. By Lemma 8.13 (b), $|E(P_{i_j})| + 4 \leq |E(P_k)| + 2 \leq |E(P_{i_{j+1}})|$ for all j. If e has neither s nor t as endpoint, we have $3 \leq |E(P_{i_j})| \leq n - 1$ for all j, and there can be at most $\frac{n}{4}$

augmenting paths containing e as a bottleneck edge. Otherwise at most one of the augmenting paths contains e or $\overleftarrow{e}$ as bottleneck edge.

Since any augmenting path must contain at least one edge of $\overleftrightarrow{G}$ as a bottleneck edge, there can be at most $|E(\overleftrightarrow{G})|\frac{n}{4} = \frac{mn}{2}$ augmenting paths. □

Corollary 8.15. *The* EDMONDS-KARP ALGORITHM *solves the* MAXIMUM FLOW PROBLEM *in* $O(m^2n)$ *time.*

Proof: By Theorem 8.14 there are at most $\frac{mn}{2}$ augmentations. Each augmentation uses BFS and thus takes $O(m)$ time. □

8.4 Dinic's, Karzanov's, and Fujishige's Algorithm

Around the time when Edmonds and Karp observed how to obtain a polynomial-time algorithm for the MAXIMUM FLOW PROBLEM, Dinic [1970] independently found an even better algorithm. It is based on the following definition:

Definition 8.16. *Given a network* (G, u, s, t) *and an* s-t*-flow* f*. The* **level graph** G_f^L *of* G_f *is the graph*

$$\left(V(G), \left\{e = (x, y) \in E(G_f) : \text{dist}_{G_f}(s, x) + 1 = \text{dist}_{G_f}(s, y)\right\}\right).$$

Note that the level graph is acyclic. The level graph can be constructed easily by BFS in $O(m)$ time. The s-t-paths in G_f^L are precisely the shortest s-t-paths in G_f.

Lemma 8.13(a) says that the length of the shortest augmenting paths in the EDMONDS-KARP ALGORITHM is non-decreasing. Let us call a sequence of augmenting paths of the same length a **phase** of the algorithm. Let f be the flow at the beginning of a phase. The proof of Lemma 8.13 (b) yields that all augmenting paths of this phase must already be augmenting paths in G_f. Therefore all these paths must be s-t-paths in the level graph of G_f. The total of all augmentions in a phase can be regarded as a blocking flow in G_f^L:

Definition 8.17. *Given a network* (G, u, s, t)*, an* s-t*-flow* f *is called* **blocking** *if* $(V(G), \{e \in E(G) : f(e) < u(e)\})$ *contains no* s-t*-path.*

Note that a blocking flow is not necessarily maximum. The above considerations suggest the following algorithmic scheme:

DINIC'S ALGORITHM	
Input:	A network (G, u, s, t).
Output:	An s-t-flow f of maximum value.

① Set $f(e) := 0$ for all $e \in E(G)$.

② Construct the level graph G_f^L of G_f.

③ Find a blocking s-t-flow f' in (G_f^L, u_f). **If** $f' = 0$ **then stop**.

④ Augment f by f' and **go to** ②.

Augmenting f by f' of course means increasing $f(e)$ by $f'(e)$ for each $e \in E(G_f^L) \cap E(G)$ and decreasing $f(e)$ by $f'(\overleftarrow{e})$ for each $e \in E(G)$ with $\overleftarrow{e} \in E(G_f^L)$.

Theorem 8.18. (Dinic [1970]) DINIC'S ALGORITHM *works correctly and stops after at most n iterations.*

Proof: If the algorithm stops, there is no s-t-path in G_f^L, and hence in G_f. It remains to prove that the length of a shortest augmenting path (which must be in $\{1, 2, \ldots, n-1, \infty\}$) increases in each iteration. Consider the flow f at the beginning of some iteration, and let $\bar{f}$ be the augmented flow f after ④. Note that $\mathrm{dist}_{G_f}(s, y) \leq \mathrm{dist}_{G_f}(s, x)$ for all $(x, y) \in E(G_f) \setminus E(G_f^L)$ and also for reverse edges (x, y) of edges in G_f^L, while $\mathrm{dist}_{G_f}(s, y) = \mathrm{dist}_{G_f}(s, x) + 1$ for all $e = (x, y) \in E(G_f^L)$. Since each augmenting path with respect to $\bar{f}$ contains only edges from $E(G_f) \cup \{\overleftarrow{e} : e \in E(G_f^L)\}$, and at least one edge that is not in G_f^L, it must contain more than $\mathrm{dist}_{G_f}(s, t)$ edges. □

Note that ② (using BFS) and ④ can be implemented in linear time. So it remains to show how a blocking flow in an acyclic graph can be found efficiently. Dinic obtained an $O(nm)$ bound for each phase, which is not very difficult to show (Exercise 19). We shall now describe Karzanov's faster algorithm. It is based on the following important definition:

Definition 8.19. (Karzanov [1974]) *Given a network (G, u, s, t), an* ***s*-*t*-preflow** *is a function $f : E(G) \to \mathbb{R}_+$ satisfying $f(e) \leq u(e)$ for all $e \in E(G)$ and $\mathrm{ex}_f(v) \geq 0$ for all $v \in V(G) \setminus \{s\}$. We call a vertex $v \in V(G) \setminus \{s, t\}$* **active** *if $\mathrm{ex}_f(v) > 0$.*

Obviously, an s-t-preflow is an s-t-flow if and only if there are no active vertices. This concept will be used again in the next section.

Theorem 8.20. (Karzanov [1974]) *A blocking flow in a network (G, u, s, t) with an acyclic digraph G can be found in $O(n^2)$ time, where $n = |V(G)|$.*

Proof: First compute a topological order $V(G) = \{v_1, v_2, \ldots, v_n\}$. For each vertex v except s and t, the algorithm maintains the list of its leaving edges and a stack, initially empty, whose elements are pairs from $\delta^-(v) \times \mathbb{R}_+$. All vertices are marked non-frozen initially.

The algorithm maintains an s-t-preflow f throughout. We begin by setting $f(e) := u(e)$ for each edge $e = (s, v) \in \delta^+(s)$ and putting $(e, u(e))$ on the stack

of v. For all other edges e we set $f(e) := 0$ initially. Then the following two steps alternate until there is no active vertex anymore:

The push step scans the vertices in topological order. While a vertex v is active and there is an edge $e = (v, w)$ with $f(e) < u(e)$ and w is not frozen, increase $f(e)$ by $\delta := \min\{u(e) - f(e), \mathrm{ex}_f(v)\}$ and put (e, δ) on the stack of w.

The balancing step deals with the active vertex v_i for which i is maximum. We remove the topmost pair (e, δ) from the stack of v_i (i.e., the one inserted last) and decrease $f(e)$ by $\delta' := \min\{\delta, \mathrm{ex}_f(v_i)\}$. If v_i is still active, we continue with the next pair from the stack. Finally, v_i is marked frozen.

If a vertex v_i is subject to balancing, then at that time all vertices v_j with $j > i$ are inactive, and so their entering edges will never carry less flow anymore than now. This is because flow is reduced during rebalancing according to the stack: first on edges where the flow was increased last. Hence v_i will never become active anymore. Thus each vertex is subject to balancing at most once, and the number of iterations is less than n. Each push step runs in $O(n + p)$ time, where p is the number of edges that receive a saturating push. Note that an edge that is saturated will not be considered anymore by a push step. Hence the total time for all push steps is $O(n^2 + m)$, and so is the total time for rebalancing.

After each push step, t is not reachable from s or any active vertex in $(V(G), \{e \in E(G) : f(e) < u(e)\})$. Hence f is a blocking flow at termination. □

We give another proof, due to Malhotra, Kumar and Maheshwari [1978]:

Second Proof of Theorem 8.20: First compute a topological order of G (cf. Theorem 2.20). For $v \in V(G)$ we write $G_{\leq v}$ and $G_{\geq v}$ for the subgraphs induced by all vertices up to v and from v on, respectively.

We start with $f(e) = 0$ for all $e \in E(G)$. Let

$$\alpha := \min\left\{\min\{u_f(\delta_G^-(v)) : v \in V(G) \setminus \{s\}\}, \min\{u_f(\delta_G^+(v)) : v \in V(G) \setminus \{t\}\}\right\},$$

and let v be a vertex where the minimum is attained.

We first find a v-t-flow g of value α in $G_{\geq v}$ (unless $v = t$). We do this by scanning the vertices in topological order. For each vertex w, where $w = v$ or predecessors have been processed already, set $\beta := \alpha$ if $w = v$ and $\beta := g(\delta_G^-(w))$ otherwise, let $\delta_G^+(w) = \{e_1, \ldots, e_k\}$, and set $g(e_j) := \min\{u_f(e_j), \beta - \sum_{i=1}^{j-1} g(e_i)\}$ for $j = 1, \ldots, k$. Similarly, we find an s-v-flow g' of value α in $G_{\leq v}$ (unless $v = s$) by scanning vertices in reverse topological order. We set $f(e) := f(e) + g(e) + g'(e)$ for all $e \in E(G)$. If $v \in \{s, t\}$, we stop: f is a blocking flow. If $v \notin \{s, t\}$, we delete v and its incident edges and iterate.

This algorithm stops after at most $n-1$ iterations and is obviously correct. It can be implemented such that each edge is scanned at most once after being saturated. In each iteration it suffices to scan at most $n - 2$ edges (one at each vertex) in addition to those that are being saturated in this iteration. By always updating the current values of $u_f(\delta_G^-(v))$ and $u_f(\delta_G^+(v))$, we can find α and v in each iteration in $O(n)$ time. The running time of $O(m + n^2)$ follows. □

Corollary 8.21. *There is an $O(n^3)$-time algorithm for the* MAXIMUM FLOW PROBLEM.

Proof: Use Theorem 8.20 to implement ③ of DINIC'S ALGORITHM. □

Subsequent improvements are due to Cherkassky [1977], Galil [1980], Galil and Namaad [1980], Shiloach [1978], Sleator [1980], and Sleator and Tarjan [1983]. The last two references describe an $O(m \log n)$-algorithm for finding blocking flows in an acyclic network using a data structure called dynamic trees. Using this as a subroutine of DINIC'S ALGORITHM one has an $O(mn \log n)$-algorithm for the MAXIMUM FLOW PROBLEM. However, we do not describe any of the above-mentioned algorithms here (see Tarjan [1983]), because an even faster network flow algorithm will be the subject of the next section.

We close this section by describing the weakly polynomial algorithm by Fujishige [2003], mainly because of its simplicity:

FUJISHIGE'S ALGORITHM

Input: A network (G, u, s, t) with $u : E(G) \to \mathbb{Z}_+$.

Output: An s-t-flow f of maximum value.

① Set $f(e) := 0$ for all $e \in E(G)$. Set $\alpha := \max\{u(e) : e \in E(G)\}$.

② Set $i := 1$, $v_1 := s$, $X := \emptyset$, and $b(v) := 0$ for all $v \in V(G)$.

③ **For** $e = (v_i, w) \in \delta^+_{G_f}(v_i)$ with $w \notin \{v_1, \ldots, v_i\}$ **do**:
Set $b(w) := b(w) + u_f(e)$. **If** $b(w) \geq \alpha$ **then** set $X := X \cup \{w\}$.

④ **If** $X = \emptyset$ **then**:
Set $\alpha := \lfloor \frac{\alpha}{2} \rfloor$. **If** $\alpha = 0$ **then stop else go to** ②.

⑤ Set $i := i + 1$. Choose $v_i \in X$ and set $X := X \setminus \{v_i\}$.
If $v_i \neq t$ **then go to** ③.

⑥ Set $\beta(t) := \alpha$ and $\beta(v) := 0$ for all $v \in V(G) \setminus \{t\}$.
While $i > 1$ **do**:
For $e = (p, v_i) \in \delta^-_{G_f}(v_i)$ with $p \in \{v_1, \ldots, v_{i-1}\}$ **do**:
Set $\beta' := \min\{\beta(v_i), u_f(e)\}$.
Augment f along e by β'.
Set $\beta(v_i) := \beta(v_i) - \beta'$ and $\beta(p) := \beta(p) + \beta'$.
Set $i := i - 1$.

⑦ **Go to** ②.

Theorem 8.22. FUJISHIGE'S ALGORITHM *correctly solves the* MAXIMUM FLOW PROBLEM *for simple digraphs G and integral capacities $u : E(G) \to \mathbb{Z}_+$ in $O(mn \log u_{\max})$ time, where $n := |V(G)|$, $m := |E(G)|$ and $u_{\max} := \max\{u(e) : e \in E(G)\}$.*

Proof: Let us call an iteration a sequence of steps ending with ④ or ⑦. In ②–⑤, $v_1, \ldots, v_i$ is always an order of a subset of vertices such that $b(v_j)$

$= u_f(E^+(\{v_1,\dots,v_{j-1}\},\{v_j\})) \geq \alpha$ for $j = 2,\dots,i$. In ⑥ the flow f is augmented with the invariant $\sum_{v\in V(G)} \beta(v) = \alpha$, and by the above the result is an s-t-flow whose value is α units larger.

Thus after at most $n-1$ iterations, α will be decreased for the first time. When we decrease α to $\alpha' = \lfloor \frac{\alpha}{2} \rfloor \geq \frac{\alpha}{3}$ in ④, we have an s-t-cut $\delta^+_{G_f}(\{v_1,\dots,v_i\})$ in G_f of capacity less than $\alpha(|V(G)|-i)$ because $b(v) = u_f(E^+(\{v_1,\dots,v_i\},\{v\})) < \alpha$ for all $v \in V(G)\setminus\{v_1,\dots,v_i\}$. By Lemma 8.3(b), a maximum s-t-flow in G_f has value less than $\alpha(n-i) < 3\alpha' n$. Hence after less than $3n$ iterations, α will be decreased again. If α is decreased from 1 to 0, we have an s-t-cut of capacity 0 in G_f, so f is maximum.

As α is decreased at most $1 + \log u_{\max}$ times before it reaches 0, and each iteration takes $O(m)$ time, the overall running time is $O(mn \log u_{\max})$. □

Such a scaling technique is useful in many contexts and will reappear in Chapter 9. Fujishige [2003] also described a variant of his algorithm without scaling, where v_i in ⑤ is chosen as a vertex attaining $\max\{b(v) : v \in V(G)\setminus\{v_1,\dots,v_{i-1}\}\}$. The resulting order is called MA order and will reappear in Section 8.7. The running time of this variant is slightly higher than the above and not strongly polynomial either (Shioura [2004]). See Exercise 24.

8.5 The Goldberg-Tarjan Algorithm

In this section we shall describe the PUSH-RELABEL ALGORITHM due to Goldberg and Tarjan [1988]. We shall derive an $O(n^2\sqrt{m})$ bound for the running time.

Sophisticated implementations using dynamic trees (see Sleator and Tarjan [1983]) result in network flow algorithms with running time $O\left(nm \log \frac{n^2}{m}\right)$ (Goldberg and Tarjan [1988]) and $O\left(nm \log\left(\frac{n}{m}\sqrt{\log u_{\max}} + 2\right)\right)$, where $u_{\max}$ is the maximum (integral) edge capacity (Ahuja, Orlin and Tarjan [1989]). The best known bounds today are $O\left(nm \log_{2+m/(n\log n)} n\right)$ (King, Rao and Tarjan [1994]) and

$$O\left(\min\{m^{1/2}, n^{2/3}\} m \log\left(\frac{n^2}{m}\right) \log u_{\max}\right)$$

(Goldberg and Rao [1998]).

By definition and Theorem 8.5, a flow f is a maximum s-t-flow if and only if the following conditions hold:

- $\mathrm{ex}_f(v) = 0$ for all $v \in V(G)\setminus\{s,t\}$;
- There is no f-augmenting path.

In the algorithms discussed so far, the first condition is always satisfied, and the algorithms stop when the second condition is satisfied. The PUSH-RELABEL ALGORITHM starts with an f satisfying the second condition and maintains it throughout. Naturally it stops when the first condition is satisfied as well. So f will not be an

s-t-flow during the algorithm (except at termination), but an s-t-preflow (cf. Definition 8.19).

Definition 8.23. *Let (G, u, s, t) be a network and f an s-t-preflow. A* **distance labeling** *is a function $\psi : V(G) \to \mathbb{Z}_+$ such that $\psi(t) = 0$, $\psi(s) = n := |V(G)|$ and $\psi(v) \leq \psi(w) + 1$ for all $(v, w) \in E(G_f)$. An edge $e = (v, w) \in E(\overleftrightarrow{G})$ is called* **admissible** *if $e \in E(G_f)$ and $\psi(v) = \psi(w) + 1$.*

If ψ is a distance labeling, $\psi(v)$ (for $v \neq s$) must be a lower bound on the distance to t (number of edges in a shortest v-t-path) in G_f.

The PUSH-RELABEL ALGORITHM to be described below always works with an s-t-preflow f and a distance labeling ψ. It starts with the preflow that is equal to the capacity on each edge leaving s and zero on all other edges. The initial distance labeling is $\psi(s) = n$ and $\psi(v) = 0$ for all $v \in V(G) \setminus \{s\}$.

Then the algorithm performs the update operations PUSH (updating f) and RELABEL (updating ψ) in any order.

PUSH-RELABEL ALGORITHM

Input: A network (G, u, s, t).

Output: A maximum s-t-flow f.

① Set $f(e) := u(e)$ for each $e \in \delta^+(s)$.
Set $f(e) := 0$ for each $e \in E(G) \setminus \delta^+(s)$.

② Set $\psi(s) := n := |V(G)|$ and $\psi(v) := 0$ for all $v \in V(G) \setminus \{s\}$.

③ **While** there exists an active vertex **do**:
Let v be an active vertex.
If no $e \in \delta^+_{G_f}(v)$ is admissible
then RELABEL(v),
else let $e \in \delta^+_{G_f}(v)$ be an admissible edge and PUSH(e).

PUSH(e)

① Set $\gamma := \min\{\mathrm{ex}_f(v), u_f(e)\}$, where v is the tail of e.

② Augment f along e by γ.

RELABEL(v)

① Set $\psi(v) := \min\{\psi(w) + 1 : (v, w) \in \delta^+_{G_f}(v)\}$.

Proposition 8.24. *During the execution of the* PUSH-RELABEL ALGORITHM *f is always an s-t-preflow and ψ is always a distance labeling with respect to f. For each $v \in V(G)$, $\psi(v)$ is strictly increased by every* RELABEL(v).

Proof: We have to show that the procedures PUSH and RELABEL preserve these properties. It is clear that after a PUSH operation, f is still an s-t-preflow. A RELABEL operation does not even change f.

If RELABEL(v) is called and ψ was a distance labeling before, then $\psi(v)$ is strictly increased (as no $e \in \delta^+_{G_f}(v)$ was admissible), and ψ remains a distance labeling.

We finally show that after a PUSH operation, ψ is still a distance labeling with respect to the new preflow. We have to check $\psi(a) \leq \psi(b) + 1$ for all new edges (a, b) in G_f. But if we apply PUSH(e) for some $e = (v, w)$, the only possible new edge in G_f is the reverse edge of e, and here we have $\psi(w) = \psi(v) - 1$, since e is admissible. □

Lemma 8.25. *If f is an s-t-preflow and ψ is a distance labeling with respect to f, then:*

(a) *s is reachable from any active vertex v in G_f.*
(b) *If w is reachable from v in G_f for some $v, w \in V(G)$, then $\psi(v) \leq \psi(w)+n-1$.*
(c) *t is not reachable from s in G_f.*

Proof: (a): Let v be an active vertex, and let R be the set of vertices reachable from v in G_f. Then $f(e) = 0$ for all $e \in \delta^-_G(R)$. So

$$\sum_{w \in R} \mathrm{ex}_f(w) \;=\; \sum_{e \in \delta^-_G(R)} f(e) - \sum_{e \in \delta^+_G(R)} f(e) \;\leq\; 0.$$

But v is active, meaning $\mathrm{ex}_f(v) > 0$, and therefore there must exist a vertex $w \in R$ with $\mathrm{ex}_f(w) < 0$. Since f is an s-t-preflow, this vertex must be s.

(b): Suppose there is a v-w-path in G_f, say with vertices $v = v_0, v_1, \ldots, v_k = w$. Since ψ is a distance labeling with respect to f, $\psi(v_i) \leq \psi(v_{i+1}) + 1$ for $i = 0, \ldots, k-1$. So $\psi(v) \leq \psi(w) + k$. Note that $k \leq n - 1$.

(c): follows from (b) as $\psi(s) = n$ and $\psi(t) = 0$. □

Part (c) helps us to prove the following:

Theorem 8.26. *When the algorithm terminates, f is a maximum s-t-flow.*

Proof: f is an s-t-flow because there are no active vertices. Lemma 8.25(c) implies that there is no augmenting path. Then by Theorem 8.5 we know that f is maximum. □

The question now is how many PUSH and RELABEL operations are performed.

Lemma 8.27.

(a) *For each $v \in V(G)$, $\psi(v)$ never decreases, and $\psi(v) \leq 2n - 1$ at any stage of the algorithm.*
(b) *No vertex is relabelled more than $2n - 1$ times. The total increase of $\sum_{v \in V(G)} \psi(v)$ during the algorithm is at most $2n^2 - n$.*

Proof: Recall from Proposition 8.24 that $\psi(v)$ is strictly increased by every RELABEL(v). Moreover, we only change $\psi(v)$ by RELABEL(v) if v is active. By Lemma 8.25(a) and (b), $\psi(v) \leq \psi(s) + n - 1 = 2n - 1$. This implies (a) and (b). □

We shall now analyse the number of PUSH operations. We distinguish between **saturating** pushes (where $u_f(e) = 0$ after the push) and **nonsaturating** pushes. As usual we denote $m := |E(G)|$ (and $n := |V(G)|$).

Lemma 8.28. *The number of saturating pushes is at most* $2mn$.

Proof: After each saturating push from v to w, another such push cannot occur until $\psi(w)$ increases by at least 2, a push from w to v occurs, and $\psi(v)$ increases by at least 2. Together with Lemma 8.27(a), this proves that there are at most n saturating pushes on each edge $(v, w) \in E(\overleftrightarrow{G})$. □

The number of nonsaturating pushes can be in the order of n^2m in general (Exercise 25). By choosing an active vertex v with $\psi(v)$ maximum in ③ we can prove a better bound. We may assume $n \leq m \leq n^2$.

Lemma 8.29. *If we always choose* v *to be an active vertex with* $\psi(v)$ *maximum in* ③ *of the* PUSH-RELABEL ALGORITHM*, the number of nonsaturating pushes is at most* $8n^2\sqrt{m}$.

Proof: Call a phase the time between two consecutive changes of $\psi^* := \max\{\psi(v) : v \text{ active}\}$. As ψ^* can increase only by relabeling, its total increase is less than $2n^2$. As $\psi^* = 0$ initially, it decreases less than $2n^2$ times, and the number of phases is less than $4n^2$.

Call a phase cheap if it contains at most $\sqrt{m}$ nonsaturating pushes and expensive otherwise. Clearly there are at most $4n^2\sqrt{m}$ nonsaturating pushes in cheap phases.

Let

$$\Phi := \sum_{v \in V(G): v \text{ active}} |\{w \in V(G) : \psi(w) \leq \psi(v)\}|.$$

Initially $\Phi \leq n^2$. A relabeling step may increase Φ by at most n. A saturating push may increase Φ by at most n. A nonsaturating push does not increase Φ. Since $\Phi = 0$ at termination, the total decrease of Φ is at most $n^2 + n(2n^2 - n) + n(2mn) \leq 4mn^2$.

Now consider the nonsaturating pushes in an expensive phase. Each of them pushes flow along an edge (v, w) with $\psi(v) = \psi^* = \psi(w) + 1$, deactivating v and possibly activating w.

As the phase ends by deactivating the last active vertex v with $\psi(v) = \psi^*$ or by relabeling, the set of vertices w with $\psi(w) = \psi^*$ remains constant during the phase, and it contains more than $\sqrt{m}$ vertices as the phase is expensive. Hence each nonsaturating push in an expensive phase decreases Φ by at least $\sqrt{m}$. Thus the total number of nonsaturating pushes in expensive phases is at most $\frac{4mn^2}{\sqrt{m}} = 4n^2\sqrt{m}$. □

This proof is due to Cheriyan and Mehlhorn [1999]. We finally get:

Theorem 8.30. (Goldberg and Tarjan [1988], Cheriyan and Maheshwari [1989], Tunçel [1994]) *The* PUSH-RELABEL ALGORITHM *solves the* MAXIMUM FLOW PROBLEM *correctly and can be implemented to run in* $O(n^2\sqrt{m})$ *time.*

Proof: The correctness follows from Theorem 8.26.

As in Lemma 8.29 we always choose v in ③ to be an active vertex with $\psi(v)$ maximum. To make this easy we keep track of doubly-linked lists $L_0, \dots, L_{2n-1}$, where L_i contains the active vertices v with $\psi(v) = i$. These lists can be updated during each PUSH and RELABEL operation in constant time.

We can then start by scanning L_i for $i = 0$. When a vertex is relabelled, we increase i accordingly. When we find a list L_i for the current i empty (after deactivating the last active vertex at that level), we decrease i until L_i is nonempty. As we increase i at most $2n^2$ times by Lemma 8.27(b), we also decrease i at most $2n^2$ times.

As a second data structure, we store a doubly-linked list A_v containing the admissible edges leaving v for each vertex v. They can also be updated in each PUSH operation in constant time, and in each RELABEL operation in time proportional to the total number of edges incident to the relabelled vertex.

So RELABEL(v) takes a total of $O(|\delta_G(v)|)$ time, and by Lemma 8.27(b) the overall time for relabelling is $O(mn)$. Each PUSH takes constant time, and by Lemma 8.28 and Lemma 8.29 the total number of pushes is $O(n^2\sqrt{m})$. □

8.6 Gomory-Hu Trees

Any algorithm for the MAXIMUM FLOW PROBLEM also implies a solution to the following problem:

MINIMUM CAPACITY CUT PROBLEM	
Instance:	A network (G, u, s, t).
Task:	An s-t-cut in G with minimum capacity.

Proposition 8.31. *The* MINIMUM CAPACITY CUT PROBLEM *can be solved in the same running time as the* MAXIMUM FLOW PROBLEM, *in particular in* $O(n^2\sqrt{m})$ *time.*

Proof: For a network (G, u, s, t) we compute a maximum s-t-flow f and define X to be the set of all vertices reachable from s in G_f. X can be computed with the GRAPH SCANNING ALGORITHM in linear time (Proposition 2.17). By Lemma 8.3 and Theorem 8.5, $\delta_G^+(X)$ constitutes a minimum capacity s-t-cut. The $O(n^2\sqrt{m})$ running time follows from Theorem 8.30 (and is not best possible). □

In this section we consider the problem of finding a minimum capacity s-t-cut for each pair of vertices s, t in an undirected graph G with capacities $u : E(G) \to \mathbb{R}_+$.

This problem can be reduced to the above one: For all pairs $s, t \in V(G)$ we solve the MINIMUM CAPACITY CUT PROBLEM for (G', u', s, t), where (G', u') arises from (G, u) by replacing each undirected edge $\{v, w\}$ by two oppositely directed edges (v, w) and (w, v) with $u'((v, w)) = u'((w, v)) = u(\{v, w\})$. In this way we obtain minimum s-t-cuts for all s, t after $\binom{n}{2}$ flow computations.

This section is devoted to the elegant method of Gomory and Hu [1961], which requires only $n - 1$ flow computations. We shall see some applications in Sections 12.3 and 20.3.

Definition 8.32. *Let G be an undirected graph and $u : E(G) \to \mathbb{R}_+$ a capacity function. For two vertices $s, t \in V(G)$ we denote by λ_{st} their* **local edge-connectivity***, i.e. the minimum capacity of a cut separating s and t.*

The edge-connectivity of a graph is obviously the minimum local edge-connectivity with respect to unit capacities.

Lemma 8.33. *For all vertices $i, j, k \in V(G)$ we have $\lambda_{ik} \geq \min\{\lambda_{ij}, \lambda_{jk}\}$.*

Proof: Let $\delta(A)$ be a cut with $i \in A$, $k \in V(G) \setminus A$ and $u(\delta(A)) = \lambda_{ik}$. If $j \in A$ then $\delta(A)$ separates j and k, so $u(\delta(A)) \geq \lambda_{jk}$. If $j \in V(G) \setminus A$ then $\delta(A)$ separates i and j, so $u(\delta(A)) \geq \lambda_{ij}$. We conclude that $\lambda_{ik} = u(\delta(A)) \geq \min\{\lambda_{ij}, \lambda_{jk}\}$. □

Indeed, this condition is not only necessary but also sufficient for numbers $(\lambda_{ij})_{1 \leq i, j \leq n}$ with $\lambda_{ij} = \lambda_{ji}$ to be local edge-connectivities of some graph (Exercise 31).

Definition 8.34. *Let G be an undirected graph and $u : E(G) \to \mathbb{R}_+$ a capacity function. A tree T is called a* **Gomory-Hu tree** *for (G, u) if $V(T) = V(G)$ and*

$$\lambda_{st} = \min_{e \in E(P_{st})} u(\delta_G(C_e)) \quad \text{for all } s, t \in V(G),$$

where P_{st} is the (unique) s-t-path in T and, for $e \in E(T)$, C_e and $V(G) \setminus C_e$ are the connected components of $T - e$ (i.e. $\delta_G(C_e)$ is the fundamental cut of e with respect to T).

We shall see that every undirected graph possesses a Gomory-Hu tree. This implies that for any undirected graph G there is a list of $n - 1$ cuts such that for each pair $s, t \in V(G)$ a minimum s-t-cut belongs to the list. This is not true for digraphs: for each $n \in \mathbb{N}$, Jelinek and Mayeda [1963] constructed a digraph G with n vertices and capacities $u : E(G) \to \mathbb{R}_+$ such that the set $\{\min\{u(\delta^+(X)) : s \in X \subseteq V(G) \setminus \{t\}\} : s, t \in V(G),\ s \neq t\}$ contains $(n+2)(n-1)/2$ different numbers.

In general, a Gomory-Hu tree cannot be chosen as a subgraph of G. For example, consider $G = K_{3,3}$ and $u \equiv 1$. Here $\lambda_{st} = 3$ for all $s, t \in V(G)$. It is easy to see that the Gomory-Hu trees for (G, u) are exactly the stars with five edges.

The main idea of the algorithm for constructing a Gomory-Hu tree is as follows. First we choose any $s, t \in V(G)$ and find some minimum s-t-cut, say $\delta(A)$. Let $B := V(G) \setminus A$. Then we contract A (or B) to a single vertex, choose any

$s', t' \in B$ (or $s', t' \in A$, respectively) and look for a minimum s'-t'-cut in the contracted graph G'. We continue this process, always choosing a pair s', t' of vertices not separated by any cut obtained so far. At each step, we contract – for each cut $E(A', B')$ obtained so far – A' or B', depending on which part does not contain s' and t'.

Eventually each pair of vertices is separated. We have obtained a total of $n - 1$ cuts. The crucial observation is that a minimum s'-t'-cut in the contracted graph G' is also a minimum s'-t'-cut in G. This is the subject of the following lemma. Note that when contracting a set A of vertices in (G, u), the capacity of each edge in G' is the capacity of the corresponding edge in G.

Lemma 8.35. *Let G be an undirected graph and $u : E(G) \to \mathbb{R}_+$ a capacity function. Let $s, t \in V(G)$, and let $\delta(A)$ be a minimum s-t-cut in (G, u). Let now $s', t' \in V(G) \setminus A$, and let (G', u') arise from (G, u) by contracting A to a single vertex. Then for any minimum s'-t'-cut $\delta(K \cup \{A\})$ in (G', u'), $\delta(K \cup A)$ is a minimum s'-t'-cut in (G, u).*

Proof: Let s, t, A, s', t', G', u' be as above. W.l.o.g. $s \in A$. It suffices to prove that there is a minimum s'-t'-cut $\delta(A')$ in (G, u) such that $A \subset A'$. So let $\delta(C)$ be any minimum s'-t'-cut in (G, u). W.l.o.g. $s \in C$.

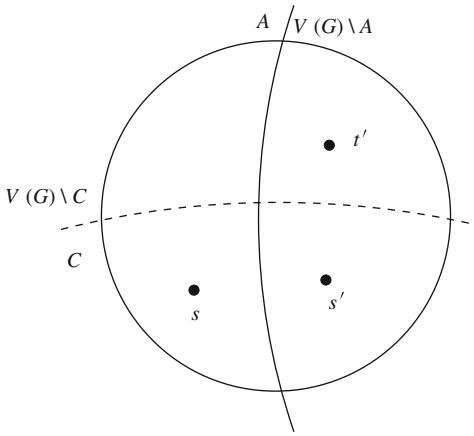

Fig. 8.3.

Since $u(\delta(\cdot))$ is submodular (cf. Lemma 2.1(c)), we have $u(\delta(A)) + u(\delta(C)) \geq u(\delta(A \cap C)) + u(\delta(A \cup C))$. But $\delta(A \cap C)$ is an s-t-cut, so $u(\delta(A \cap C)) \geq \lambda_{st} = u(\delta(A))$. Therefore $u(\delta(A \cup C)) \leq u(\delta(C)) = \lambda_{s't'}$ proving that $\delta(A \cup C)$ is a minimum s'-t'-cut. (See Figure 8.3.) □

Now we describe the algorithm which constructs a Gomory-Hu tree. Note that the vertices of the intermediate trees T will be vertex sets of the original graph;

indeed they form a partition of $V(G)$. At the beginning, the only vertex of T is $V(G)$. In each iteration, a vertex of T containing at least two vertices of G is chosen and split into two.

GOMORY-HU ALGORITHM

Input: An undirected graph G and a capacity function $u : E(G) \to \mathbb{R}_+$.

Output: A Gomory-Hu tree T for (G, u).

① Set $V(T) := \{V(G)\}$ and $E(T) := \emptyset$.

② Choose some $X \in V(T)$ with $|X| \geq 2$. **If** no such X exists **then go to** ⑥.

③ Choose $s, t \in X$ with $s \neq t$.
For each connected component C of $T - X$ **do**: Let $S_C := \bigcup_{Y \in V(C)} Y$.
Let (G', u') arise from (G, u) by contracting S_C to a single vertex v_C for each connected component C of $T - X$.
(So $V(G') = X \cup \{v_C : C$ is a connected component of $T - X\}$.)

④ Find a minimum s-t-cut $\delta(A')$ in (G', u'). Let $B' := V(G') \setminus A'$.
Set $A := \left(\bigcup_{v_C \in A' \setminus X} S_C \right) \cup (A' \cap X)$ and $B := \left(\bigcup_{v_C \in B' \setminus X} S_C \right) \cup (B' \cap X)$.

⑤ Set $V(T) := (V(T) \setminus \{X\}) \cup \{A \cap X, B \cap X\}$.
For each edge $e = \{X, Y\} \in E(T)$ incident to the vertex X **do**:
If $Y \subseteq A$ **then** set $e' := \{A \cap X, Y\}$ **else** set $e' := \{B \cap X, Y\}$.
Set $E(T) := (E(T) \setminus \{e\}) \cup \{e'\}$ and $w(e') := w(e)$.
Set $E(T) := E(T) \cup \{\{A \cap X, B \cap X\}\}$.
Set $w(\{A \cap X, B \cap X\}) := u'(\delta_{G'}(A'))$.
Go to ②.

⑥ Replace all $\{x\} \in V(T)$ by x and all $\{\{x\}, \{y\}\} \in E(T)$ by $\{x, y\}$. **Stop**.

Figure 8.4 illustrates the modification of T in ⑤. To prove the correctness of this algorithm, we first show the following lemma:

Lemma 8.36. *Each time at the end of ④ we have*

(a) $A \dot{\cup} B = V(G)$
(b) $E(A, B)$ *is a minimum* s-t-*cut in* (G, u).

Proof: The elements of $V(T)$ are always nonempty subsets of $V(G)$, indeed $V(T)$ constitutes a partition of $V(G)$. From this, (a) follows easily.

We now prove (b). The claim is trivial for the first iteration (since here $G' = G$). We show that the property is preserved in each iteration.

Let $C_1, \ldots, C_k$ be the connected components of $T - X$. Let us contract them one by one; for $i = 0, \ldots, k$ let (G_i, u_i) arise from (G, u) by contracting each of $S_{C_1}, \ldots, S_{C_i}$ to a single vertex. So (G_k, u_k) is the graph which is denoted by (G', u') in ③ of the algorithm.

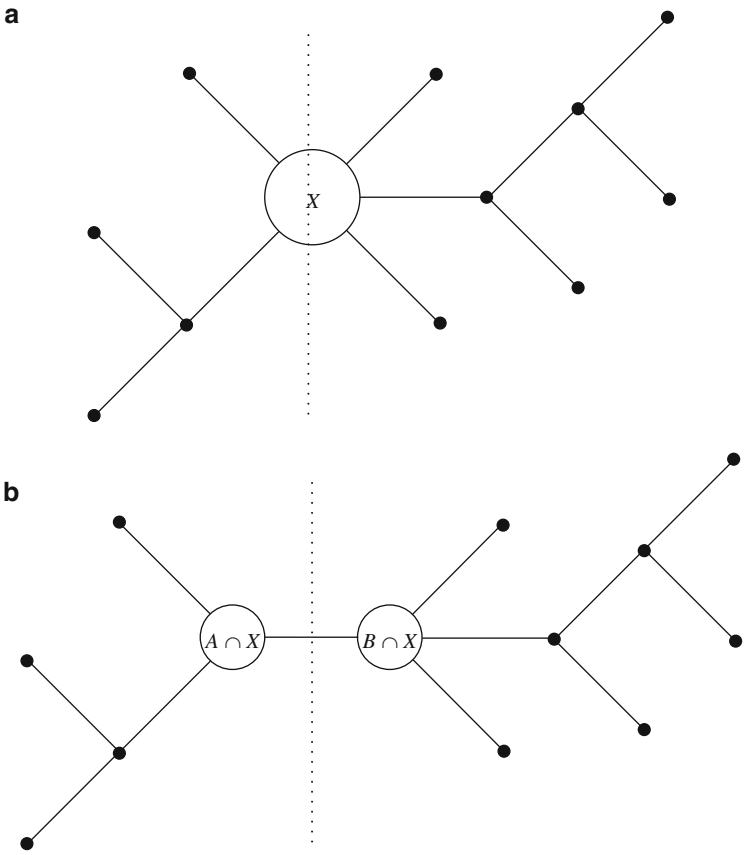

Fig. 8.4.

Claim: For any minimum s-t-cut $\delta(A_i)$ in (G_i, u_i), $\delta(A_{i-1})$ is a minimum s-t-cut in (G_{i-1}, u_{i-1}), where

$$A_{i-1} := \begin{cases} (A_i \setminus \{v_{C_i}\}) \cup S_{C_i} & \text{if } v_{C_i} \in A_i \\ A_i & \text{if } v_{C_i} \notin A_i \end{cases}.$$

Applying this claim successively for $k, k-1, \ldots, 1$ implies (b).

To prove the claim, let $\delta(A_i)$ be a minimum s-t-cut in (G_i, u_i). By our assumption that (b) is true for the previous iterations, $\delta(S_{C_i})$ is a minimum s_i-t_i-cut in (G, u) for some appropriate $s_i, t_i \in V(G)$. Furthermore, $s, t \in V(G) \setminus S_{C_i}$. So applying Lemma 8.35 completes the proof. □

Lemma 8.37. *At any stage of the algorithm (until ⑥ is reached) for all $e \in E(T)$*

$$w(e) = u\left(\delta_G\left(\bigcup_{Z \in C_e} Z\right)\right),$$

where C_e and $V(T) \setminus C_e$ are the connected components of $T - e$. Moreover for all $e = \{P, Q\} \in E(T)$ there are vertices $p \in P$ and $q \in Q$ with $\lambda_{pq} = w(e)$.

Proof: Both statements are trivial at the beginning of the algorithm when T contains no edges; we show that they are never violated. So let X be the vertex of T chosen in ② in some iteration of the algorithm. Let s, t, A', B', A, B be as determined in ③ and ④ next. W.l.o.g. assume $s \in A'$.

Edges of T not incident to X are not affected by ⑤. For the new edge $\{A \cap X, B \cap X\}$, $w(e)$ is clearly set correctly, and we have $\lambda_{st} = w(e)$, $s \in A \cap X$, $t \in B \cap X$.

So let us consider an edge $e = \{X, Y\}$ that is replaced by e' in ⑤. We assume w.l.o.g. $Y \subseteq A$, so $e' = \{A \cap X, Y\}$. Assuming that the assertions were true for e we claim that they remain true for e'. This is trivial for the first assertion, because $w(e) = w(e')$ and $u\left(\delta_G\left(\bigcup_{Z \in C_e} Z\right)\right)$ does not change.

To show the second statement, we assume that there are $p \in X, q \in Y$ with $\lambda_{pq} = w(e)$. If $p \in A \cap X$ then we are done. So henceforth assume that $p \in B \cap X$ (see Figure 8.5).

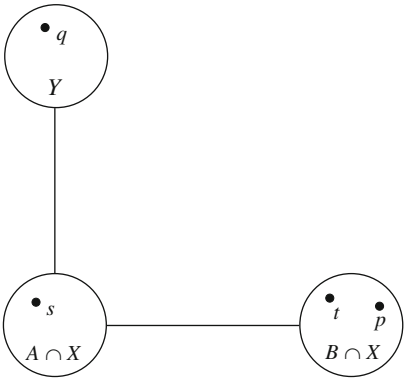

Fig. 8.5.

We claim that $\lambda_{sq} = \lambda_{pq}$. Since $\lambda_{pq} = w(e) = w(e')$ and $s \in A \cap X$, this will conclude the proof.

By Lemma 8.33,

$$\lambda_{sq} \geq \min\{\lambda_{st}, \lambda_{tp}, \lambda_{pq}\}.$$

Since by Lemma 8.36(b) $E(A, B)$ is a minimum s-t-cut, and since $s, q \in A$, we may conclude from Lemma 8.35 that λ_{sq} does not change if we contract B. Since $t, p \in B$, this means that adding an edge $\{t, p\}$ with arbitrary high capacity does not change λ_{sq}. Hence

$$\lambda_{sq} \geq \min\{\lambda_{st}, \lambda_{pq}\}.$$

Now observe that $\lambda_{st} \geq \lambda_{pq}$ because the minimum s-t-cut $E(A, B)$ also separates p and q. So we have

$$\lambda_{sq} \geq \lambda_{pq}.$$

To prove equality, observe that $w(e)$ is the capacity of a cut separating X and Y, and thus s and q. Hence

$$\lambda_{sq} \leq w(e) = \lambda_{pq}.$$

This completes the proof. □

Theorem 8.38. (Gomory and Hu [1961]) *The* GOMORY-HU ALGORITHM *works correctly. Every undirected graph possesses a Gomory-Hu tree, and such a tree is found in* $O(n^3\sqrt{m})$ *time.*

Proof: The complexity of the algorithm is clearly determined by $n-1$ times the complexity of finding a minimum s-t-cut, since everything else can be implemented in $O(n^3)$ time. By Proposition 8.31 we obtain the $O(n^3\sqrt{m})$ bound.

We prove that the output T of the algorithm is a Gomory-Hu tree for (G,u). It should be clear that T is a tree with $V(T) = V(G)$. Now let $s,t \in V(G)$. Let P_{st} be the (unique) s-t-path in T and, for $e \in E(T)$, let C_e and $V(G) \setminus C_e$ be the connected components of $T-e$.

Since $\delta(C_e)$ is an s-t-cut for each $e \in E(P_{st})$,

$$\lambda_{st} \leq \min_{e \in E(P_{st})} u(\delta(C_e)).$$

On the other hand, a repeated application of Lemma 8.33 yields

$$\lambda_{st} \geq \min_{\{v,w\} \in E(P_{st})} \lambda_{vw}.$$

Hence applying Lemma 8.37 to the situation before execution of ⑥ (where each vertex X of T is a singleton) yields

$$\lambda_{st} \geq \min_{e \in E(P_{st})} u(\delta(C_e)),$$

so equality holds. □

A similar algorithm for the same task (which might be easier to implement) was suggested by Gusfield [1990]. For digraphs, Cheung, Lau and Leung [2011] showed how to compute the minimum cardinality of an s-t-cut for all pairs $s,t \in V(G)$ in $O(m^{2.38})$ time.

8.7 The Minimum Capacity of a Cut in an Undirected Graph

If we are only interested in a minimum capacity cut in an undirected graph G with capacities $u : E(G) \to \mathbb{R}_+$, there is a simpler method using $n-1$ flow computations: just compute a minimum s-t-cut for some fixed vertex s and each $t \in V(G) \setminus \{s\}$. However, there are more efficient algorithms.

Hao and Orlin [1994] found an $O(nm \log \frac{n^2}{m})$-algorithm for determining a minimum capacity cut. They use a modified version of the PUSH-RELABEL ALGORITHM.

If we just want to compute the edge-connectivity of the graph (i.e. unit capacities), the currently fastest algorithm is due to Gabow [1995] with running time $O(m+\lambda^2 n \log \frac{n}{\lambda(G)})$, where $\lambda(G)$ is the edge-connectivity (observe that $2m \geq \lambda n$). Gabow's algorithm uses matroid intersection techniques. We remark that the MAXIMUM FLOW PROBLEM in undirected graphs with unit capacities can also be solved faster than in general (Karger and Levine [1998]).

Nagamochi and Ibaraki [1992] found a completely different algorithm to determine a minimum capacity cut in an undirected graph. Their algorithm does not use max-flow computations at all. In this section we present this algorithm in a simplified form due to Stoer and Wagner [1997] and independently to Frank [1994]. We start with an easy definition.

Definition 8.39. *Given an undirected graph G with capacities $u : E(G) \to \mathbb{R}_+$, we call an order $v_1, \ldots, v_n$ of the vertices an* **MA (maximum adjacency) order** *if for all $i \in \{2, \ldots, n\}$:*

$$\sum_{e \in E(\{v_1,\ldots,v_{i-1}\},\{v_i\})} u(e) \;=\; \max_{j \in \{i,\ldots,n\}} \sum_{e \in E(\{v_1,\ldots,v_{i-1}\},\{v_j\})} u(e).$$

Proposition 8.40. *Given an undirected graph G with capacities $u : E(G) \to \mathbb{R}_+$, an MA order can be found in $O(m + n \log n)$ time.*

Proof: Consider the following algorithm. First set $\alpha(v) := 0$ for all $v \in V(G)$. Then for $i := 1$ to n do the following: choose v_i from among $V(G) \setminus \{v_1, \ldots, v_{i-1}\}$ such that it has maximum α-value (breaking ties arbitrarily), and set $\alpha(v) := \alpha(v) + \sum_{e \in E(\{v_i\},\{v\})} u(e)$ for all $v \in V(G) \setminus \{v_1, \ldots, v_i\}$.

The correctness of this algorithm is obvious. By implementing it with a Fibonacci heap, storing each vertex v with key $-\alpha(v)$ until it is selected, we get a running time of $O(m + n \log n)$ by Theorem 6.7 as there are n INSERT-, n DELETEMIN- and (at most) m DECREASEKEY-operations. □

Lemma 8.41. (Stoer and Wagner [1997], Frank [1994]) *Let G be an undirected graph with $n := |V(G)| \geq 2$, capacities $u : E(G) \to \mathbb{R}_+$ and an MA order $v_1, \ldots, v_n$. Then*

$$\lambda_{v_{n-1}v_n} \;=\; \sum_{e \in \delta(v_n)} u(e).$$

Proof: Of course we only have to show "≥". We shall use induction on $|V(G)| + |E(G)|$. For $|V(G)| < 3$ the statement is trivial. We may assume that there is no edge $e = \{v_{n-1}, v_n\} \in E(G)$, because otherwise we would delete it (both left-hand side and right-hand side decrease by $u(e)$) and apply the induction hypothesis.

Denote the right-hand side by R. Of course $v_1, \ldots, v_{n-1}$ is an MA order in $G - v_n$. So by induction,

$$\lambda^{G-v_n}_{v_{n-2}v_{n-1}} = \sum_{e\in E(\{v_{n-1}\},\{v_1,\dots,v_{n-2}\})} u(e) \geq \sum_{e\in E(\{v_n\},\{v_1,\dots,v_{n-2}\})} u(e) = R.$$

Here the inequality holds because $v_1,\dots,v_n$ was an MA order for G. The last equality is true because $\{v_{n-1},v_n\}\notin E(G)$. So $\lambda^G_{v_{n-2}v_{n-1}} \geq \lambda^{G-v_n}_{v_{n-2}v_{n-1}} \geq R$.

On the other hand $v_1,\dots,v_{n-2},v_n$ is an MA order in $G-v_{n-1}$. So by induction,

$$\lambda^{G-v_{n-1}}_{v_{n-2}v_n} = \sum_{e\in E(\{v_n\},\{v_1,\dots,v_{n-2}\})} u(e) = R,$$

again because $\{v_{n-1},v_n\}\notin E(G)$. So $\lambda^G_{v_{n-2}v_n} \geq \lambda^{G-v_{n-1}}_{v_{n-2}v_n} = R$.

Now by Lemma 8.33 $\lambda_{v_{n-1}v_n} \geq \min\{\lambda_{v_{n-1}v_{n-2}},\lambda_{v_{n-2}v_n}\} \geq R$. □

Note that the existence of two vertices x,y with $\lambda_{xy} = \sum_{e\in\delta(x)} u(e)$ was already shown by Mader [1972], and follows easily from the existence of a Gomory-Hu tree (Exercise 33).

Theorem 8.42. (Nagamochi and Ibaraki [1992], Stoer and Wagner [1997]) *A minimum capacity cut in an undirected graph with nonnegative capacities can be found in $O(mn+n^2\log n)$ time.*

Proof: We may assume that the given graph G is simple since we can unite parallel edges. Denote by $\lambda(G)$ the minimum capacity of a cut in G. The algorithm proceeds as follows:

Let $G_0 := G$. In the i-th step ($i = 1,\dots,n-1$) choose vertices $x,y\in V(G_{i-1})$ with

$$\lambda^{G_{i-1}}_{xy} = \sum_{e\in\delta_{G_{i-1}}(x)} u(e).$$

By Proposition 8.40 and Lemma 8.41 this can be done in $O(m+n\log n)$ time. Set $\gamma_i := \lambda^{G_{i-1}}_{xy}$, $z_i := x$, and let G_i result from G_{i-1} by contracting $\{x,y\}$. Observe that

$$\lambda(G_{i-1}) = \min\{\lambda(G_i),\gamma_i\}, \tag{8.1}$$

because a minimum cut in G_{i-1} either separates x and y (in this case its capacity is γ_i) or does not (in this case contracting $\{x,y\}$ does not change anything).

After arriving at G_{n-1} which has only one vertex, we choose a $k\in\{1,\dots,n-1\}$ for which γ_k is minimum. We claim that $\delta(X)$ is a minimum capacity cut in G, where X is the vertex set in G whose contraction resulted in the vertex z_k of G_{k-1}. But this is easy to see, since by (8.1) $\lambda(G) = \min\{\gamma_1,\dots,\gamma_{n-1}\} = \gamma_k$ and γ_k is the capacity of the cut $\delta(X)$. □

A randomized contraction algorithm for finding a minimum cut (with high probability) is discussed in Exercise 37. Moreover, we mention that the vertex-connectivity of a graph can be computed by $O(n^2)$ flow computations in a graph with unit capacities (Exercise 38).

In this section we have shown how to minimize $f(X) := u(\delta(X))$ over $\emptyset \neq X \subset V(G)$. Note that this $f : 2^{V(G)} \to \mathbb{R}_+$ is submodular and symmetric (i.e. $f(A) = f(V(G) \setminus A)$ for all A). The algorithm presented here has been generalized by Queyranne [1998] to minimize general symmetric submodular functions; see Section 14.5.

The problem of finding a maximum cut is much harder and will be discussed in Section 16.2.

Exercises

1. Let (G, u, s, t) be a network, and let $\delta^+(X)$ and $\delta^+(Y)$ be minimum s-t-cuts in (G, u). Show that $\delta^+(X \cap Y)$ and $\delta^+(X \cup Y)$ are also minimum s-t-cuts in (G, u).
2. Show that in case of irrational capacities, the FORD-FULKERSON ALGORITHM may not terminate at all.
   Hint: Consider the following network (Figure 8.6):

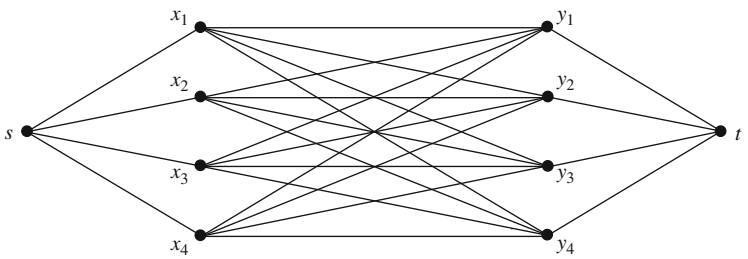

Fig. 8.6.

   All lines represent edges in both directions. All edges have capacity $S = \frac{1}{1-\sigma}$ except

$$u((x_1, y_1)) = 1, \quad u((x_2, y_2)) = \sigma, \quad u((x_3, y_3)) = u((x_4, y_4)) = \sigma^2$$

   where $\sigma = \frac{\sqrt{5}-1}{2}$. Note that $\sigma^n = \sigma^{n+1} + \sigma^{n+2}$.
   (Ford and Fulkerson [1962])

* 3. Let G be a digraph and M the incidence matrix of G. Prove that for all $c, l, u \in \mathbb{Z}^{E(G)}$ with $l \leq u$:

$$\max\left\{cx : x \in \mathbb{Z}^{E(G)},\ l \leq x \leq u,\ Mx = 0\right\} =$$

$$\min\left\{y'u - y''l : y', y'' \in \mathbb{Z}_+^{E(G)},\ zM + y' - y'' = c \text{ for some } z \in \mathbb{Z}^{V(G)}\right\}.$$

   Show how this implies Theorem 8.6 and Corollary 8.7.

4. Prove Hoffman's circulation theorem: Given a digraph G and lower and upper capacities $l, u : E(G) \to \mathbb{R}_+$ with $l(e) \leq u(e)$ for all $e \in E(G)$, there is circulation f with $l(e) \leq f(e) \leq u(e)$ for all $e \in E(G)$ if and only if

$$\sum_{e \in \delta^-(X)} l(e) \leq \sum_{e \in \delta^+(X)} u(e) \quad \text{for all } X \subseteq V(G).$$

Note: Hoffman's circulation theorem in turn quite easily implies the Max-Flow-Min-Cut Theorem.
(Hoffman [1960])

5. Consider a network (G, u, s, t), a maximum s-t-flow f and the residual graph G_f. Form a digraph H from G_f by contracting the set S of vertices reachable from s to a vertex v_S, contracting the set T of vertices from which t is reachable to a vertex v_T, and contracting each strongly connected component X of $G_f - (S \cup T)$ to a vertex v_X. Observe that H is acyclic. Prove that there is a one-to-one correspondence between the sets $X \subseteq V(G)$ for which $\delta^+_G(X)$ is a minimum s-t-cut in (G, u) and the sets $Y \subseteq V(H)$ for which $\delta^+_H(Y)$ is a directed v_T-v_S-cut in H (i.e. a directed cut in H separating v_T and v_S).
Note: This statement also holds for G_f without any contraction instead of H. However, we shall use the statement in the above form in Section 20.4.
(Picard and Queyranne [1980])

6. Let G be a digraph and $c, c' : E(G) \to \mathbb{R}$. We look for a set $X \subset V(G)$ with $s \in X$ and $t \notin X$ such that $\sum_{e \in \delta^+(X)} c(e) - \sum_{e \in \delta^-(X)} c'(e)$ is minimum.
   (a) Show how to reduce this problem to the MINIMUM CAPACITY CUT PROBLEM.
   (b) Now consider the special case where $c = c'$. Can you solve this problem in linear time?

* 7. Let G be an acyclic digraph with mappings $\sigma, \tau, c : E(G) \to \mathbb{R}_+$, and a number $C \in \mathbb{R}_+$. We look for a mapping $x : E(G) \to \mathbb{R}_+$ such that $\sigma(e) \leq x(e) \leq \tau(e)$ for all $e \in E(G)$ and $\sum_{e \in E(G)} (\tau(e) - x(e)) c(e) \leq C$. Among the feasible solutions we want to minimize the length (with respect to x) of the longest path in G.
The meaning behind the above is the following. The edges correspond to jobs, $\sigma(e)$ and $\tau(e)$ stand for the minimum and maximum completion time of job e, and $c(e)$ is the cost of reducing the completion time of job e by one unit. If there are two jobs $e = (i, j)$ and $e' = (j, k)$, job e has to be finished before job e' can be processed. We have a fixed budget C and want to minimize the total completion time.
Show how to solve this problem using network flow techniques. (This application is known as PERT, program evaluation and review technique, or CPM, critical path method. The problem is also known as the budget version of the time-cost tradeoff problem.)
Hint: Introduce one source s and one sink t. Start with $x = \tau$ and successively reduce the length of the longest s-t-path (with respect to x) at the minimum possible cost. Use Exercise 8 of Chapter 7, Exercise 9 of Chapter 3, and Exercise 6.
(Phillips and Dessouky [1977])

* 8. Let (G, c, s, t) be a network such that G is planar even when an edge $e = (s, t)$ is added. Consider the following algorithm. Start with the flow $f \equiv 0$ and let $G' := G_f$. At each step consider the boundary B of a face of $G' + e$ containing e (with respect to some fixed planar embedding). Augment f along $B - e$. Let G' consist of the forward edges of G_f only and iterate as long as t is reachable from s in G'.
Prove that this algorithm computes a maximum s-t-flow. Use Theorem 2.40 to show that it can be implemented to run in $O(n^2)$ time.
(Ford and Fulkerson [1956], Hu [1969])
Note: This problem can be solved in $O(n)$ time. For general planar networks an $O(n \log n)$-algorithm exists; see Weihe [1997] and Borradaile and Klein [2009].

9. Show that the directed edge-disjoint version of Menger's Theorem 8.9 also follows directly from Theorem 6.18.

10. Let G be an undirected graph. Prove that one can compute an orientation G' of G in linear time such that for each $v, w \in V(G)$ the following holds: if G has two edge-disjoint v-w-paths, then G' has a (directed) v-w-path.
Hint: Use DFS.
(Tarjan [1972])

11. Let G be a digraph with conservative weights $c : E(G) \to \mathbb{R}$ and two vertices $s, t \in V(G)$ such that t is reachable from s. Suppose that for every edge $e \in E(G)$ we have $\text{dist}_{(G-e,c)}(s, t) = \text{dist}_{(G,c)}(s, t)$. Prove that then there are two edge-disjoint shortest s-t-paths in (G, c).

12. Consider an undirected graph G with edge-connectivity $k \in \mathbb{N}$ and (not necessarily distinct) vertices $v_0, v_1, \ldots, v_k \in V(G)$. Prove that there are pairwise edge-disjoint paths $P_1, \ldots, P_k$ such that P_i is a v_0-v_i-path $(i = 1, \ldots, k)$.

13. Let G be a graph (directed or undirected), x, y, z three vertices, and $\alpha, \beta \in \mathbb{N}$ with $\alpha \leq \lambda_{xy}$, $\beta \leq \lambda_{xz}$ and $\alpha + \beta \leq \max\{\lambda_{xy}, \lambda_{xz}\}$. Prove that there are α x-y-paths and β x-z-paths such that these $\alpha + \beta$ paths are pairwise edge-disjoint.

14. Let G be a digraph that contains k edge-disjoint s-t-paths for any two vertices s and t (such a graph is called strongly k-edge-connected).
Let H be any digraph with $V(H) = V(G)$ and $|E(H)| = k$. Prove that the instance (G, H) of the DIRECTED EDGE-DISJOINT PATHS PROBLEM has a solution.
(Mader [1981] and Shiloach [1979])

15. Let G be a digraph with at least k edges. Prove: G contains k edge-disjoint s-t-paths for any two vertices s and t if and only if for any k distinct edges $e_1 = (x_1, y_1), \ldots, e_k = (x_k, y_k)$, $G - \{e_1, \ldots, e_k\}$ contains k edge-disjoint spanning arborescences $T_1, \ldots, T_k$ such that T_i is rooted at y_i $(i = 1, \ldots, k)$.
Note: This generalizes Exercise 14. *Hint:* Use Theorem 6.18.
(Su [1997])

16. Let G be a digraph with capacities $c : E(G) \to \mathbb{R}_+$ and $r \in V(G)$. Can one determine an r-cut with minimum capacity in polynomial time? Can one determine a directed cut with minimum capacity in polynomial time (or decide that G is strongly connected)?
    Note: The answer to the first question solves the SEPARATION PROBLEM for the MINIMUM WEIGHT ROOTED ARBORESCENCE PROBLEM; see Corollary 6.15.
17. An airline wants to conduct a given set of scheduled flights with as few airplanes as possible. All available airplanes are of the same type. For each flight we know the departure time and the flight time. We also know, for any pair of flights i and j, how much time an airplane needs after finishing flight i until it can begin flight j (this time will depend in particular on where i ends and where j begins). Show how to compute efficiently a feasible schedule with as few airplanes as possible.
18. Prove that the value of a blocking s-t-flow in a network (G, u, s, t) with an acyclic digraph G is at least $\frac{1}{|V(G)|}$ times the value of a maximum s-t-flow. Show that this bound is sharp up to a constant factor.
19. Show how to find a blocking flow in an acyclic network in $O(nm)$ time by successively augmenting along a path of non-saturated edges and using DEPTH-FIRST SEARCH to find such a path. Show how to obtain a running time of $O(m)$ if all edges that are not incident to s or t have capacity 1.

* 20. Let (G, u, s, t) be a network with $u(e) = 1$ for all edges $e \in E(G)$ that are not incident to s or t.
    (a) Show that then a maximum s-t-flow can be computed in $O(mn^{2/3})$ time.
    (b) Let in addition G have the property that for each $v \in V(G) \setminus \{s, t\}$ we have $|\delta^-(v)| = 1$ or $|\delta^+(v)| = 1$. Show that then a maximum s-t-flow can be computed in $O(m\sqrt{n})$ time.
    Hint: Consider DINIC'S ALGORITHM and the situation when no augmenting path has length less than $\lceil n^{2/3} \rceil$ in (a) and $\lceil \sqrt{n} \rceil$ in (b). Bound the number of remaining iterations and use the second part of Exercise 19.
    (Karzanov [1973], Even and Tarjan [1975])
21. An s-t-preflow f is called *maximum* if $\mathrm{ex}_f(t)$ is maximum.
    (a) Show that for any maximum preflow f there exists a maximum flow f' with $f'(e) \leq f(e)$ for all $e \in E(G)$.
    (b) Show how a maximum preflow can be converted into a maximum flow in $O(nm)$ time.
22. Let (G, u, s, t) be a network such that $G - t$ is an arborescence. Show how to find a maximum s-t-flow in linear time.
    Hint: Use DFS.

* 23. Let (G, u, s, t) be a network such that the underlying undirected graph of $G - \{s, t\}$ is a forest. Show how to find a maximum s-t-flow in linear time.
    (Vygen [2002])
24. Consider a modified version of FUJISHIGE'S ALGORITHM where in ⑤ we choose $v_i \in V(G) \setminus \{v_1, \ldots, v_{i-1}\}$ such that $b(v_i)$ is maximum, step ④ is

replaced by stopping if $b(v) = 0$ for all $v \in V(G) \setminus \{v_1, \dots, v_i\}$, and in the beginning of ⑥ we set $\beta(t) := \min_{j=2}^{i} b(j)$. Then X and α are not needed anymore.

(a) Show that this variant of the algorithm works correctly.

(b) Let α_k be the number $\min_{j=2}^{i} b(j)$ in iteration k (or zero if the algorithm stops before iteration k). Show that $\min_{l=k+1}^{k+2n} \alpha_l \leq \frac{1}{2}\alpha_k$ for all k. Conclude that the number of iterations is $O(n \log u_{\max})$.

(c) Show how to implement one iteration in $O(m + n \log n)$ time.

25. Prove that the PUSH-RELABEL ALGORITHM performs $O(n^2 m)$ nonsaturating pushes, independent of the choice of v in ③.

26. Let (G, u, s, t) be a network, f an s-t-preflow, and ψ a distance labeling with respect to f with $\psi(v) \leq 2n$ for $v \in V(G)$. Define $\psi'(v) := \min\{\text{dist}_{G_f}(v, t), n + \text{dist}_{G_f}(v, s), 2n\}$ for $v \in V(G)$. Show that ψ' is a distance labeling with respect to f, and $\psi \leq \psi'$.
Note: Replacing ψ by ψ' from time to time, e.g. after every n RELABEL operations, improves the performance of the PUSH-RELABEL ALGORITHM in practice.

27. Given an acyclic digraph G with weights $c : E(G) \to \mathbb{R}_+$, find a maximum weight directed cut in G. Show how this problem can be reduced to the MINIMUM CAPACITY CUT PROBLEM.
Hint: Use Exercise 6.

28. Let G be an acyclic digraph with weights $c : E(G) \to \mathbb{R}_+$. We look for the maximum weight edge set $F \subseteq E(G)$ such that no path in G contains more than one edge of F. Show that this problem is equivalent to looking for the maximum weight directed cut in G (and thus can be solved in $O(n^3)$ time by Exercise 27).

29. Let G be a digraph and $p : V(G) \to \mathbb{R}$. Show how to find a set $X \subseteq V(G)$ with $\delta^+(X) = \emptyset$ such that $p(X)$ is maximum.
Note: This was used to model open-pit mining, where $p(v)$ is the (possibly negative) profit of mining v, and an edge (v, w) models the constraint that we cannot mine v unless we mine w.

30. Given an undirected graph G with capacities $u : E(G) \to \mathbb{R}_+$ and a set $T \subseteq V(G)$ with $|T| \geq 2$. We look for a set $X \subset V(G)$ with $T \cap X \neq \emptyset$ and $T \setminus X \neq \emptyset$ such that $\sum_{e \in \delta(X)} u(e)$ is minimum. Show how to solve this problem in $O(n^4)$ time, where $n = |V(G)|$.

31. Let λ_{ij}, $1 \leq i, j \leq n$, be nonnegative numbers with $\lambda_{ij} = \lambda_{ji}$ and $\lambda_{ik} \geq \min\{\lambda_{ij}, \lambda_{jk}\}$ for any three distinct indices $i, j, k \in \{1, \dots, n\}$. Show that there exists a graph G with $V(G) = \{1, \dots, n\}$ and capacities $u : E(G) \to \mathbb{R}_+$ such that the local edge-connectivities are precisely the λ_{ij}.
Hint: Consider a maximum weight spanning tree in (K_n, c), where $c(\{i, j\}) := \lambda_{ij}$.
(Gomory and Hu [1961])

32. Let G be an undirected graph with capacities $u : E(G) \to \mathbb{R}_+$, and let $T \subseteq V(G)$ with $|T|$ even. A T-cut in G is a cut $\delta(X)$ with $|X \cap T|$ odd. Construct a

polynomial time algorithm for finding a T-cut of minimum capacity in (G, u).
Hint: Use a Gomory-Hu tree.
(A solution of this exercise can be found in Section 12.3.)

33. Let G be a simple undirected graph with at least two vertices. Suppose the degree of each vertex of G is at least k. Prove that there are two vertices s and t such that at least k edge-disjoint s-t-paths exist. What if there is exactly one vertex with degree less than k?
Hint: Consider a Gomory-Hu tree for G.
34. Consider the problem of determining the edge-connectivity $\lambda(G)$ of an undirected graph (with unit capacities). Section 8.7 shows how to solve this problem in $O(mn)$ time, provided that we can find an MA order of an undirected graph with unit capacities in $O(m + n)$ time. How can this be done?

∗ 35. Let G be an undirected graph with an MA order $v_1, \ldots, v_n$. Let κ^G_{uv} denote the maximum number of internally disjoint u-v-paths in G. Prove $\kappa^G_{v_{n-1}v_n} = |E(\{v_n\}, \{v_1, \ldots, v_{n-1}\})|$ (the vertex-disjoint counterpart of Lemma 8.41).
Hint: Prove by induction that $\kappa^{G_{ij}}_{v_j v_i} = |E(\{v_j\}, \{v_1, \ldots, v_i\})|$, where $G_{ij} = G[\{v_1, \ldots, v_i\} \cup \{v_j\}]$. To do this, assume w.l.o.g. that $\{v_j, v_i\} \notin E(G)$, choose a minimal set $Z \subseteq \{v_1, \ldots, v_{i-1}\}$ separating v_j and v_i (Menger's Theorem 8.10), and let $h \leq i$ be the maximum number such that $v_h \notin Z$ and v_h is adjacent to v_i or v_j.
(Frank [unpublished])

∗ 36. An undirected graph is called chordal if it has no circuit of length at least four as an induced subgraph. An order $v_1, \ldots, v_n$ of an undirected graph G is called simplicial if $\{v_i, v_j\}, \{v_i, v_k\} \in E(G)$ implies $\{v_j, v_k\} \in E(G)$ for $i < j < k$.
(a) Prove that a graph with a simplicial order must be chordal.
(b) Let G be a chordal graph, and let $v_1, \ldots, v_n$ be an MA order. Prove that $v_n, v_{n-1}, \ldots, v_1$ is a simplicial order.
Hint: Use Exercise 35 and Menger's Theorem 8.10.
Note: The fact that a graph is chordal if and only if it has a simplicial order is due to Rose [1970].

37. Let G be an undirected graph with capacities $u : E(G) \to \mathbb{R}_+$. Let $\emptyset \neq A \subset V(G)$ such that $\delta(A)$ is a minimum capacity cut in G.
(a) Show that $u(\delta(A)) \leq \frac{2}{n} u(E(G))$. (*Hint:* Consider the trivial cuts $\delta(x)$, $x \in V(G)$.)
(b) Assume w.l.o.g. that $u(\delta(A)) > 0$ and consider the following procedure. We randomly choose an edge and contract it; each edge e is chosen with probability $\frac{u(e)}{u(E(G))}$. We repeat this operation until there are only two vertices. Prove that the probability that we never contract an edge of $\delta(A)$ is at least $\frac{2}{(n-1)n}$.
(c) Conclude that running the randomized algorithm in (b) kn^2 times yields $\delta(A)$ with probability at least $1 - e^{-2k}$. (Such an algorithm with a positive probability of a correct answer is called a Monte Carlo algorithm.)
(Karger and Stein [1996]; see also Karger [2000])

38. Show how the vertex-connectivity of an undirected graph can be determined in $O(n^{2.5}m)$ time.
Hint: Recall the proof of Menger's Theorem and use Exercise 20.
Note: Faster algorithms were proposed by Henzinger, Rao and Gabow [2000], and by Gabow [2006].
39. Let G be a connected undirected graph with capacities $u : E(G) \to \mathbb{R}_+$. We are looking for a minimum capacity 3-cut, i.e. an edge set whose deletion splits G into at least three connected components.
Let $n := |V(G)| \geq 4$. Let $\delta(X_1), \delta(X_2), \ldots$ be a list of the cuts ordered by nondecreasing capacities: $u(\delta(X_1)) \leq u(\delta(X_2)) \leq \cdots$. Assume that we know the first $2n-2$ elements of this list (note: they can be computed in polynomial time by a method of Vazirani and Yannakakis [1992]).
    (a) Show that for some indices $i, j \in \{1, \ldots, 2n-2\}$ all sets $X_i \setminus X_j$, $X_j \setminus X_i$, $X_i \cap X_j$ and $V(G) \setminus (X_i \cup X_j)$ are nonempty.
    (b) Show that there is a 3-cut of capacity at most $\frac{3}{2}u(\delta(X_{2n-2})$.
    (c) For each $i = 1, \ldots, 2n-2$ consider $\delta(X_i)$ plus a minimum capacity cut of $G - X_i$, and also $\delta(X_i)$ plus a minimum capacity cut of $G[X_i]$. This yields a list of at most $4n-4$ 3-cuts. Prove that one of them is optimum.
(Nagamochi and Ibaraki [2000])
Note: This was generalized to k-cuts (for any fixed k) by Kamidoi, Yoshida and Nagamochi [2007]; see also Thorup [2008]. The problem of finding the optimum 3-cut separating three given vertices is much harder; see Dahlhaus et al. [1994] and Cheung, Cunningham and Tang [2006].
40. Let G be an undirected graph with capacities $u : E(G) \to \mathbb{Z}_+$.
    (a) Show that if $\delta(X)$ and $\delta(Y)$ are two minimum capacity cuts with $X \cap Y \neq \emptyset$ and $X \cup Y \neq V(G)$, then $\delta(X \setminus Y) \cap \delta(Y \setminus X) = \emptyset$.
    (b) Suppose that the minimum capacity of a cut is odd. Show that then the family of vertex sets X for which $\delta(X)$ is a minimum capacity cut is cross-free, and hence there are at most $n-1$ minimum capacity cuts.
Note: Dinitz, Karzanov and Lomonosov [1976] showed that there are at most $\binom{n}{2}$ minimum capacity cuts in general. They can be described by a so-called cactus representation, generalizing tree-representations. See also Frank [2011].

References

General Literature:

Ahuja, R.K., Magnanti, T.L., and Orlin, J.B. [1993]: Network Flows. Prentice-Hall, Englewood Cliffs 1993

Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., and Schrijver, A. [1998]: Combinatorial Optimization. Wiley, New York 1998, Chapter 3

Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C. [2001]: Introduction to Algorithms. Second Edition. MIT Press, Cambridge 2001, Chapter 26

Ford, L.R., and Fulkerson, D.R. [1962]: Flows in Networks. Princeton University Press, Princeton 1962

Frank, A. [1995]: Connectivity and network flows. In: Handbook of Combinatorics; Vol. 1 (R.L. Graham, M. Grötschel, L. Lovász, eds.), Elsevier, Amsterdam, 1995

Frank, A. [2011]: Connections in Combinatorial Optimization. Oxford University Press, Oxford 2011

Goldberg, A.V., Tardos, É., and Tarjan, R.E. [1990]: Network flow algorithms. In: Paths, Flows, and VLSI-Layout (B. Korte, L. Lovász, H.J. Prömel, A. Schrijver, eds.), Springer, Berlin 1990, pp. 101–164

Gondran, M., and Minoux, M. [1984]: Graphs and Algorithms. Wiley, Chichester 1984, Chapter 5

Jungnickel, D. [2007]: Graphs, Networks and Algorithms. Third Edition. Springer, Berlin 2007

Phillips, D.T., and Garcia-Diaz, A. [1981]: Fundamentals of Network Analysis. Prentice-Hall, Englewood Cliffs 1981

Ruhe, G. [1991]: Algorithmic Aspects of Flows in Networks. Kluwer Academic Publishers, Dordrecht 1991

Schrijver, A. [2003]: Combinatorial Optimization: Polyhedra and Efficiency. Springer, Berlin 2003, Chapters 9,10,13–15

Tarjan, R.E. [1983]: Data Structures and Network Algorithms. SIAM, Philadelphia 1983, Chapter 8

Thulasiraman, K., and Swamy, M.N.S. [1992]: Graphs: Theory and Algorithms. Wiley, New York 1992, Chapter 12

Cited References:

Ahuja, R.K., Orlin, J.B., and Tarjan, R.E. [1989]: Improved time bounds for the maximum flow problem. SIAM Journal on Computing 18 (1989), 939–954

Borradaile, G. and Klein, P. [2009]: An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. Journal of the ACM 56 (2009), Article 9

Cheriyan, J., and Maheshwari, S.N. [1989]: Analysis of preflow push algorithms for maximum network flow. SIAM Journal on Computing 18 (1989), 1057–1086

Cheriyan, J., and Mehlhorn, K. [1999]: An analysis of the highest-level selection rule in the preflow-push max-flow algorithm. Information Processing Letters 69 (1999), 239–242

Cherkassky, B.V. [1977]: Algorithm of construction of maximal flow in networks with complexity of $O(V^2\sqrt{E})$ operations. Mathematical Methods of Solution of Economical Problems 7 (1977), 112–125 [in Russian]

Cheung, K.K.H., Cunningham, W.H., and Tang, L. [2006]: Optimal 3-terminal cuts and linear programming. Mathematical Programming 106 (2006), 1–23

Cheung, H.Y., Lau, L.C., and Leung, K.M. [2011]: Graph connectivities, network coding, and expander graphs. Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (2011), 190–199

Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., and Yannakakis, M. [1994]: The complexity of multiterminal cuts. SIAM Journal on Computing 23 (1994), 864–894

Dantzig, G.B., and Fulkerson, D.R. [1956]: On the max-flow min-cut theorem of networks. In: Linear Inequalities and Related Systems (H.W. Kuhn, A.W. Tucker, eds.), Princeton University Press, Princeton 1956, pp. 215–221

Dinic, E.A. [1970]: Algorithm for solution of a problem of maximum flow in a network with power estimation. Soviet Mathematics Doklady 11 (1970), 1277–1280

Dinitz, E.A., Karzanov, A.V., and Lomonosov, M.V. [1976]: On the structure of the system of minimum edge cuts of a graph. Issledovaniya po Diskretnoĭ Optimizatsii (A.A. Fridman, ed.), Nauka, Moscow, 1976, pp. 290–306 [in Russian]

Edmonds, J., and Karp, R.M. [1972]: Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM 19 (1972), 248–264

Elias, P., Feinstein, A., and Shannon, C.E. [1956]: Note on maximum flow through a network. IRE Transactions on Information Theory, IT-2 (1956), 117–119

Even, S., and Tarjan, R.E. [1975]: Network flow and testing graph connectivity. SIAM Journal on Computing 4 (1975), 507–518

Ford, L.R., and Fulkerson, D.R. [1956]: Maximal Flow Through a Network. Canadian Journal of Mathematics 8 (1956), 399–404

Ford, L.R., and Fulkerson, D.R. [1957]: A simple algorithm for finding maximal network flows and an application to the Hitchcock problem. Canadian Journal of Mathematics 9 (1957), 210–218

Frank, A. [1994]: On the edge-connectivity algorithm of Nagamochi and Ibaraki. Laboratoire Artemis, IMAG, Université J. Fourier, Grenoble, 1994

Fujishige, S. [2003]: A maximum flow algorithm using MA ordering. Operations Research Letters 31 (2003), 176–178

Gabow, H.N. [1995]: A matroid approach to finding edge connectivity and packing arborescences. Journal of Computer and System Sciences 50 (1995), 259–273

Gabow, H.N. [2006]: Using expander graphs to find vertex-connectivity. Journal of the ACM 53 (2006), 800–844

Galil, Z. [1980]: An $O(V^{\frac{5}{3}}E^{\frac{2}{3}})$ algorithm for the maximal flow problem. Acta Informatica 14 (1980), 221–242

Galil, Z., and Namaad, A. [1980]: An $O(EV\log^2 V)$ algorithm for the maximal flow problem. Journal of Computer and System Sciences 21 (1980), 203–217

Gallai, T. [1958]: Maximum-minimum Sätze über Graphen. Acta Mathematica Academiae Scientiarum Hungaricae 9 (1958), 395–434

Goldberg, A.V., and Rao, S. [1998]: Beyond the flow decomposition barrier. Journal of the ACM 45 (1998), 783–797

Goldberg, A.V., and Tarjan, R.E. [1988]: A new approach to the maximum flow problem. Journal of the ACM 35 (1988), 921–940

Gomory, R.E., and Hu, T.C. [1961]: Multi-terminal network flows. Journal of SIAM 9 (1961), 551–570

Gusfield, D. [1990]: Very simple methods for all pairs network flow analysis. SIAM Journal on Computing 19 (1990), 143–155

Hao, J., and Orlin, J.B. [1994]: A faster algorithm for finding the minimum cut in a directed graph. Journal of Algorithms 17 (1994), 409–423

Henzinger, M.R., Rao, S., and Gabow, H.N. [2000]: Computing vertex connectivity: new bounds from old techniques. Journal of Algorithms 34 (2000), 222–250

Hoffman, A.J. [1960]: Some recent applications of the theory of linear inequalities to extremal combinatorial analysis. In: Combinatorial Analysis (R.E. Bellman, M. Hall, eds.), AMS, Providence 1960, pp. 113–128

Hu, T.C. [1969]: Integer Programming and Network Flows. Addison-Wesley, Reading 1969

Jelinek, F., and Mayeda, W. [1963]: On the maximum number of different entries in the terminal capacity matrix of oriented communication nets. IEEE Transactions on Circuit Theory 10 (1963), 307–308

Kamidoi, Y., Yoshida, N., and Nagamochi, H. [2007]: A deterministic algorithm for finding all minimum k-way cuts. SIAM Journal on Computing 36 (2007), 1329–1341

Karger, D.R. [2000]: Minimum cuts in near-linear time. Journal of the ACM 47 (2000), 46–76

Karger, D.R., and Levine, M.S. [1998]: Finding maximum flows in undirected graphs seems easier than bipartite matching. Proceedings of the 30th Annual ACM Symposium on Theory of Computing (1998), 69–78

Karger, D.R., and Stein, C. [1996]: A new approach to the minimum cut problem. Journal of the ACM 43 (1996), 601–640

Karzanov, A.V. [1973]: On finding a maximum flow in a network with special structure and some applications. In: Matematicheskie Voprosy Upravleniya Proizvodstvom 5 (L.A. Lyusternik, ed.), Moscow State University Press, Moscow, 1973, pp. 81–94 [in Russian]

Karzanov, A.V. [1974]: Determining a maximal flow in a network by the method of preflows. Soviet Mathematics Doklady 15 (1974), 434–437

King, V., Rao, S., and Tarjan, R.E. [1994]: A faster deterministic maximum flow algorithm. Journal of Algorithms 17 (1994), 447–474

Mader, W. [1972]: Über minimal n-fach zusammenhängende, unendliche Graphen und ein Extremalproblem. Arch. Math. 23 (1972), 553–560

Mader, W. [1981]: On a property of n edge-connected digraphs. Combinatorica 1 (1981), 385–386

Malhotra, V.M., Kumar, M.P., and Maheshwari, S.N. [1978]: An $O(|V|^3)$ algorithm for finding maximum flows in networks. Information Processing Letters 7 (1978), 277–278

Menger, K. [1927]: Zur allgemeinen Kurventheorie. Fundamenta Mathematicae 10 (1927), 96–115

Nagamochi, H., and Ibaraki, T. [1992]: Computing edge-connectivity in multigraphs and capacitated graphs. SIAM Journal on Discrete Mathematics 5 (1992), 54–66

Nagamochi, H., and Ibaraki, T. [2000]: A fast algorithm for computing minimum 3-way and 4-way cuts. Mathematical Programming 88 (2000), 507–520

Phillips, S., and Dessouky, M.I. [1977]: Solving the project time/cost tradeoff problem using the minimal cut concept. Management Science 24 (1977), 393–400

Picard, J., and Queyranne, M. [1980]: On the structure of all minimum cuts in a network and applications. Mathematical Programming Study 13 (1980), 8–16

Queyranne, M. [1998]: Minimizing symmetric submodular functions. Mathematical Programming B 82 (1998), 3–12

Rose, D.J. [1970]: Triangulated graphs and the elimination process. Journal of Mathematical Analysis and Applications 32 (1970), 597–609

Shiloach, Y. [1978]: An $O(nI \log^2 I)$ maximum-flow algorithm. Technical Report STAN-CS-78-802, Computer Science Department, Stanford University, 1978

Shiloach, Y. [1979]: Edge-disjoint branching in directed multigraphs. Information Processing Letters 8 (1979), 24–27

Shioura, A. [2004]: The MA ordering max-flow algorithm is not strongly polynomial for directed networks. Operations Research Letters 32 (2004), 31–35

Sleator, D.D. [1980]: An $O(nm \log n)$ algorithm for maximum network flow. Technical Report STAN-CS-80-831, Computer Science Department, Stanford University, 1978

Sleator, D.D., and Tarjan, R.E. [1983]: A data structure for dynamic trees. Journal of Computer and System Sciences 26 (1983), 362–391

Su, X.Y. [1997]: Some generalizations of Menger's theorem concerning arc-connected digraphs. Discrete Mathematics 175 (1997), 293–296

Stoer, M., and Wagner, F. [1997]: A simple min cut algorithm. Journal of the ACM 44 (1997), 585–591

Tarjan, R.E. [1972]: Depth first search and linear graph algorithms. SIAM Journal on Computing 1 (1972), 146–160

Thorup, M. [2008]: Minimum k-way cuts via deterministic greedy tree packing. Proceedings of the 40th Annual ACM Symposium on Theory of Computing (2008), 159–165

Tunçel, L. [1994]: On the complexity preflow-push algorithms for maximum flow problems. Algorithmica 11 (1994), 353–359

Vazirani, V.V., and Yannakakis, M. [1992]: Suboptimal cuts: their enumeration, weight, and number. In: Automata, Languages and Programming; Proceedings of the 19th ICALP conference; LNCS 623 (W. Kuich, ed.), Springer, Berlin 1992, pp. 366–377

Vygen, J. [2002]: On dual minimum cost flow algorithms. Mathematical Methods of Operations Research 56 (2002), 101–126

Weihe, K. [1997]: Maximum (s,t)-flows in planar networks in $O(|V|\log|V|)$ time. Journal of Computer and System Sciences 55 (1997), 454–475

Whitney, H. [1932]: Congruent graphs and the connectivity of graphs. American Journal of Mathematics 54 (1932), 150–168

http://www.springer.com/978-3-642-24487-2

Combinatorial Optimization
Theory and Algorithms
Korte, B.; Vygen, J.
2012, XX, 660 p., Hardcover
ISBN: 978-3-642-24487-2