Chapter 10

Maximum flow

An $s-t$ flow is defined as a nonnegative real-valued function on the arcs of a digraph satisfying the 'flow conservation law' at each vertex $\neq s,t$. In this chapter we consider the problem of finding a maximum-value flow subject to a given capacity function. Basic results are Ford and Fulkerson's max-flow min-cut theorem and their augmenting path algorithm to find a maximum flow.

Each $s-t$ flow is a nonnegative linear combination of incidence vectors of $s-t$ paths and of directed circuits. Moreover, an integer flow is an integer such combination. This makes flows tightly connected to disjoint paths. Thus, maximum integer flow corresponds to a capacitated version of a maximum packing of disjoint paths, and the max-flow min-cut theorem is equivalent to Menger's theorem on disjoint paths.

Distinguishing characteristic of flow is however that it is not described by a combination of paths but by a function on the arcs. This promotes the algorithmic tractability.

In this chapter, graphs can be assumed to be simple.

10.1. Flows: concepts

Let $D=(V,A)$ be a digraph and let $s,t\in V$. A function $f:A\to\mathbb{R}$ is called a *flow from s to t*, or an *$s-t$ flow*, if:

$$\begin{array}{lll} \text{(i)} & f(a)\geq 0 & \text{for each } a\in A,\\ \text{(ii)} & f(\delta^{\text{out}}(v))=f(\delta^{\text{in}}(v)) & \text{for each } v\in V\setminus\{s,t\}. \end{array} \tag{10.1}$$

Condition (10.1)(ii) is called the *flow conservation law*: the amount of flow entering a vertex $v\neq s,t$ should be equal to the amount of flow leaving v.

The *value* of an $s-t$ flow f is, by definition:

$$\text{value}(f):=f(\delta^{\text{out}}(s))-f(\delta^{\text{in}}(s)). \tag{10.2}$$

So the value is the net amount of flow leaving s. This is equal to the net amount of flow entering t (this follows from (10.5) below).

Let $c:A\to\mathbb{R}_+$ be a *capacity* function. We say that a flow f is *under c* (or *subject to c*) if

$$f(a)\leq c(a) \text{ for each } a\in A. \tag{10.3}$$

A *maximum $s-t$ flow*, or just a *maximum flow*, is an $s-t$ flow under c, of maximum value. The *maximum flow problem* is to find a maximum flow.

By compactness and continuity, a maximum flow exists. It will follow from the results in this chapter (in particular, Theorem 10.4), that if the capacities are rational, then there exists a rational-valued maximum flow.

It will be convenient to make an observation on general functions $f : A \to \mathbb{R}$. For any $f : A \to \mathbb{R}$, the *excess function* is the function $\text{excess}_f : \mathcal{P}(V) \to \mathbb{R}$ defined by

$$\text{excess}_f(U) := f(\delta^{\text{in}}(U)) - f(\delta^{\text{out}}(U)) \tag{10.4}$$

for $U \subseteq V$. Set $\text{excess}_f(v) := \text{excess}_f(\{v\})$ for $v \in V$. Then:

Theorem 10.1. *Let $D = (V, A)$ be a digraph, let $f : A \to \mathbb{R}$, and let $U \subseteq V$. Then:*

$$\text{excess}_f(U) = \sum_{v \in U} \text{excess}_f(v). \tag{10.5}$$

Proof. This follows directly by counting, for each $a \in A$, the multiplicity of $f(a)$ at both sides of (10.5). ▮

To formulate a min-max relation, define the *capacity* of a cut $\delta^{\text{out}}(U)$ by $c(\delta^{\text{out}}(U))$. Then:

Theorem 10.2. *Let $D = (V, A)$ be a digraph, $s, t \in V$, and $c : A \to \mathbb{R}_+$. Then*

$$\text{value}(f) \leq c(\delta^{\text{out}}(U)), \tag{10.6}$$

for each $s-t$ flow $f \leq c$ and each $s-t$ cut $\delta^{\text{out}}(U)$. Equality holds in (10.6) if and only if $f(a) = c(a)$ for each $a \in \delta^{\text{out}}(U)$ and $f(a) = 0$ for each $a \in \delta^{\text{in}}(U)$.

Proof. Using (10.5) we have

$$\begin{aligned} \text{value}(f) &= -\text{excess}_f(s) = -\text{excess}_f(U) = f(\delta^{\text{out}}(U)) - f(\delta^{\text{in}}(U)) \\ &\leq c(\delta^{\text{out}}(U)), \end{aligned} \tag{10.7}$$

with equality if and only if $f(\delta^{\text{out}}(U)) = c(\delta^{\text{out}}(U))$ and $f(\delta^{\text{in}}(U)) = 0$. ▮

Finally, we consider a concept that turns out to be important in studying flows. Let $D = (V, A)$ be a digraph. For each $a = (u, v) \in A$, let $a^{-1} := (v, u)$. Define

$$A^{-1} := \{a^{-1} \mid a \in A\}. \tag{10.8}$$

Fix a lower bound function $d : A \to \mathbb{R}$ and an upper bound function $c : A \to \mathbb{R}$. Then for any $f : A \to \mathbb{R}$ satisfying $d \leq f \leq c$ we define

$$A_f := \{a \mid a \in A, f(a) < c(a)\} \cup \{a^{-1} \mid a \in A, f(a) > d(a)\}. \tag{10.9}$$

Clearly, A_f depends not only on f, but also on D, d, and c, but in the applications below D, d, and c are fixed, while f is variable. The digraph

$$D_f = (V, A_f) \tag{10.10}$$

is called the *residual graph* of f. So D_f is a subgraph of the directed graph $(V, A \cup A^{-1})$. As we shall see, the residual graph is very useful in studying flows and circulations, both theoretically and algorithmically.

In the context of flows we take $d = \mathbf{0}$. We observe:

Corollary 10.2a. *Let f be an $s-t$ flow in D with $f \leq c$. Suppose that D_f has no $s-t$ path. Define U as the set of vertices reachable in D_f from s. Then* value$(f) = c(\delta_A^{\rm out}(U))$. *In particular, f has maximum value.*

Proof. We apply Theorem 10.2. For each $a \in \delta_A^{\rm out}(U)$, one has $a \notin A_f$, and hence $f(a) = c(a)$. Similarly, for each $a \in \delta_A^{\rm in}(U)$ one has $a^{-1} \notin A_f$, and hence $f(a) = 0$. So value$(f) = c(\delta_A^{\rm out}(U))$ and f has maximum value by Theorem 10.2. ∎

Any directed path P in D_f gives an undirected path in $D = (V, A)$. Define $\chi^P \in \mathbb{R}^A$ by:

$$\chi^P(a) := \begin{cases} 1 & \text{if } P \text{ traverses } a, \\ -1 & \text{if } P \text{ traverses } a^{-1}, \\ 0 & \text{if } P \text{ traverses neither } a \text{ nor } a^{-1}, \end{cases} \tag{10.11}$$

for $a \in A$.

10.2. The max-flow min-cut theorem

The following theorem was proved by Ford and Fulkerson [1954,1956b] for the undirected case and by Dantzig and Fulkerson [1955,1956] for the directed case. (According to Robacker [1955a], the max-flow min-cut theorem was conjectured first by D.R. Fulkerson.)

Theorem 10.3 (max-flow min-cut theorem)**.** *Let $D = (V, A)$ be a digraph, let $s, t \in V$, and let $c : A \to \mathbb{R}_+$. Then the maximum value of an $s-t$ flow subject to c is equal to the minimum capacity of an $s-t$ cut.*

Proof. Let f be an $s-t$ flow subject to c, of maximum value. By Theorem 10.2, it suffices to show that there is an $s-t$ cut $\delta^{\rm out}(U)$ with capacity equal to value(f).

Consider the residual graph D_f (for lower bound $d := \mathbf{0}$). Suppose that it contains an $s-t$ path P. Then $f' := f + \varepsilon\chi^P$ is again an $s-t$ flow subject to c, for $\varepsilon > 0$ small enough, with value(f') = value$(f) + \varepsilon$. This contradicts the maximality of value(f).

So D_f contains no $s-t$ path. Let U be the set of vertices reachable in D_f from s. Then value$(f) = c(\delta^{\text{out}}(U))$ by Corollary 10.2a. ∎

This 'constructive' proof method is implied by the algorithm of Ford and Fulkerson [1955,1957b], to be discussed below.

Moreover, one has (Dantzig and Fulkerson [1955,1956])[16]:

Corollary 10.3a (integrity theorem)**.** *If c is integer, there exists an integer maximum flow.*

Proof. Directly from the proof of the max-flow min-cut theorem, where we can take $\varepsilon = 1$. ∎

10.3. Paths and flows

The following observation gives an important link between flows at one side and paths at the other side.

Let $D = (V, A)$ be a digraph, let $s, t \in V$, and let $f : A \to \mathbb{R}_+$ be an $s-t$ flow. Then f is a nonnegative linear combination of at most $|A|$ vectors χ^P, where P is a directed $s-t$ path or a directed circuit. If f is integer, we can take the linear combination integer-scalared.

Conversely, if $P_1, \ldots, P_k$ are $s-t$ paths in D, then $f := \chi^{AP_1} + \cdots + \chi^{AP_k}$ is an integer $s-t$ flow of value k.

With this observation, Corollary 10.3a implies the arc-disjoint version of Menger's theorem (Corollary 9.1b). Conversely, Corollary 10.3a (the integrity theorem) can be derived from the arc-disjoint version of Menger's theorem by replacing each arc a by $c(a)$ parallel arcs.

10.4. Finding a maximum flow

The proof idea of the max-flow min-cut theorem can also be used algorithmically to find a maximum $s-t$ flow, as was shown by Ford and Fulkerson [1955,1957b]. Let $D = (V, A)$ be a digraph and $s, t \in V$ and let $c : A \to \mathbb{Q}_+$ be a 'capacity' function.

Initially set $f := \mathbf{0}$. Next apply the following *flow-augmenting algorithm* iteratively:

(10.12) let P be a directed $s-t$ path in D_f and reset $f := f + \varepsilon\chi^P$, where ε is as large as possible so as to maintain $\mathbf{0} \le f \le c$.

If no such path exists, the flow f is maximum, by Corollary 10.2a.

The path P is called a *flow-augmenting path* or an *f-augmenting path*, or just an *augmenting path*.

[16] The name 'integrity theorem' was used by Ford and Fulkerson [1962].

As for termination, we have:

Theorem 10.4. *If all capacities $c(a)$ are rational, the algorithm terminates.*

Proof. If all capacities are rational, there exists a natural number K such that $Kc(a)$ is an integer for each $a \in A$. (We can take for K the l.c.m. of the denominators of the $c(a)$.)

Then in the flow-augmenting iterations, every $f_i(a)$ and every ε is a multiple of $1/K$. So at each iteration, the flow value increases by at least $1/K$. Since the flow value cannot exceed $c(\delta^{\text{out}}(\{s\}))$, there are only finitely many iterations. ∎

If we delete the rationality condition, this theorem is not maintained — see Section 10.4a. On the other hand, in Section 10.5 we shall see that if we always choose a *shortest possible* flow-augmenting path, then the algorithm terminates in a polynomially bounded number of iterations, regardless whether the capacities are rational or not.

10.4a. Nontermination for irrational capacities

Ford and Fulkerson [1962] showed that Theorem 10.4 is not maintained if we allow arbitrary real-valued capacities. The example is as follows.

Let $D = (V, A)$ be the complete directed graph on 8 vertices, with $s, t \in V$. Let $A_0 = \{a_1, a_2, a_3\}$ consist of three disjoint arcs of D, each disjoint from s and t. Let r be the positive root of $r^2 + r - 1 = 0$; that is, $r = (-1 + \sqrt{5})/2 < 1$. Define a capacity function c on A by

$$c(a_1) := 1,\ c(a_2) := 1,\ c(a_3) := r, \tag{10.13}$$

and $c(a)$ at least

$$q := \frac{1}{1-r} = 1 + r + r^2 + \cdots \tag{10.14}$$

for each $a \in A \setminus A_0$. Apply the flow-augmenting algorithm iteratively as follows.

In step 0, choose, as flow-augmenting path, the $s-t$ path of length 3 traversing a_1. After this step, the flow f satisfies, for $k = 1$:

(10.15)
- (i) f has value $1 + r + r^2 + \cdots + r^{k-1}$,
- (ii) $\{c(a) - f(a) \mid a \in A_0\} = \{0, r^{k-1}, r^k\}$,
- (iii) $f(a) \le 1 + r + r^2 + \cdots + r^{k-1}$ for each $a \in A$.

We describe the further steps. In each step k, for $k \ge 1$, the input flow f satisfies (10.15). Choose a flow-augmenting path P in D_f that contains the arc $a \in A_0$ satisfying $c(a) - f(a) = 0$ in backward direction, and the other two arcs in A_0 in forward direction; all other arcs of P are arcs of D in forward direction. Since $r^k < r^{k-1}$, and since $(1 + r + \cdots + r^{k-1}) + r^k < q$, the flow augmentation increases the flow value by r^k. Since $r^{k-1} - r^k = r^{k+1}$, the new flow satisfies (10.15) with k replaced by $k + 1$.

We can keep iterating this, making the flow value converge to $1+r+r^2+r^3+\cdots = q$. So the algorithm does not terminate, and the flow value does not converge to the optimum value, since, trivially, the maximum flow value is more than q.

(Zwick [1995] gave the smallest directed graph (with 6 vertices and 8 arcs) for which the algorithm (with irrational capacities) need not terminate.)

10.5. A strongly polynomial bound on the number of iterations

We saw in Theorem 10.4 that the number of iterations in the maximum flow algorithm is finite, if all capacities are rational. But if we choose as our flow-augmenting path P in the auxiliary graph D_f an *arbitrary* $s-t$ path, the number of iterations yet can get quite large. For instance, in the graph in Figure 10.1 the number of iterations, at an unfavourable choice of paths, can become $2 \cdot 10^k$, so exponential in the size of the input data (which is $O(k)$).

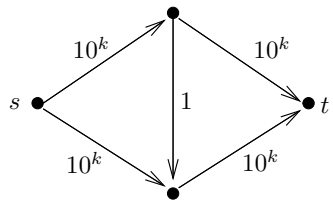

Figure 10.1

However, if we choose always a *shortest* $s-t$ path in D_f as our flow-augmenting path P (that is, with a minimum number of arcs), then the number of iterations is at most $|V| \cdot |A|$ (also if capacities are irrational). This was shown by Dinits [1970] and Edmonds and Karp [1972]. (The latter remark that this refinement 'is so simple that it is likely to be incorporated innocently into a computer implementation.')

To see this bound on the number of iterations, let again, for any digraph $D = (V, A)$ and $s, t \in V$, $\mu(D)$ denote the minimum length of an $s-t$ path. Moreover, let $\alpha(D)$ denote the set of arcs contained in at least one shortest $s-t$ path. Recall that by Theorem 9.5:

(10.16) for $D' := (V, A \cup \alpha(D)^{-1})$, one has $\mu(D') = \mu(D)$ and $\alpha(D') = \alpha(D)$.

This implies the result of Dinits [1970] and Edmonds and Karp [1972]:

Theorem 10.5. *If we choose in each iteration a shortest $s-t$ path in D_f as flow-augmenting path, the number of iterations is at most $|V| \cdot |A|$.*

Proof. If we augment flow f along a shortest $s-t$ path P in D_f, obtaining flow f', then $D_{f'}$ is a subgraph of $D' := (V, A_f \cup \alpha(D_f)^{-1})$. Hence $\mu(D_{f'}) \geq \mu(D') = \mu(D_f)$ (by (10.16)). Moreover, if $\mu(D_{f'}) = \mu(D_f)$, then $\alpha(D_{f'}) \subseteq \alpha(D') = \alpha(D_f)$ (again by (10.16)). As at least one arc in P belongs to D_f but not to $D_{f'}$, we have a strict inclusion. Since $\mu(D_f)$ increases at most $|V|$ times and, as long as $\mu(D_f)$ does not change, $\alpha(D_f)$ decreases at most $|A|$ times, we have the theorem. ∎

Since a shortest path can be found in time $O(m)$ (Theorem 6.3), this gives:

Corollary 10.5a. *A maximum flow can be found in time $O(nm^2)$.*

Proof. Directly from Theorem 10.5. ∎

10.6. Dinits' $O(n^2m)$ algorithm

Dinits [1970] observed that one can speed up the maximum flow algorithm, by not augmenting simply along *paths* in D_f, but along *flows* in D_f. The approach is similar to that of Section 9.3 for path packing.

To describe this, define a capacity function c_f on A_f by, for each $a \in A$:

$$\begin{array}{ll} c_f(a) := c(a) - f(a) & \text{if } a \in A_f \text{ and} \\ c_f(a^{-1}) := f(a) & \text{if } a^{-1} \in A_f. \end{array} \tag{10.17}$$

Then for any flow g in D_f subject to c_f,

$$f'(a) := f(a) + g(a) - g(a^{-1}) \tag{10.18}$$

gives a flow f' in D subject to c. (We define $g(a)$ or $g(a^{-1})$ to be 0 if a or a^{-1} does not belong to A_f.)

Now we shall see that, given a flow f in D, one can find in time $O(m)$ a flow g in D_f such that the flow f' arising by (10.18) satisfies $\mu(D_{f'}) > \mu(D_f)$. It implies that there are at most n iterations.

The basis of the method is the concept of 'blocking flow'. An $s-t$ flow f is called *blocking* if for each $s-t$ flow f' with $f \leq f' \leq c$ one has $f' = f$.

Theorem 10.6. *Given an acyclic graph $D = (V, A)$, $s, t \in V$, and a capacity function $c : A \to \mathbb{Q}_+$, a blocking $s-t$ flow can be found in time $O(nm)$.*

Proof. By depth-first search we can find, in time $O(|A'|)$, a subset A' of A and an $s-t$ path P in A' such that no arc in $A' \setminus AP$ is contained in any $s-t$ path: just scan s (cf. (6.2)) until t is reached; then A' is the set of arcs considered so far.

Let f be the maximum flow that can be sent along P, and reset $c := c - f$. Delete all arcs in $A' \setminus AP$ and all arcs a with $c(a) = 0$, and recursively find

a blocking $s-t$ flow f' in the new network. Then $f'+f$ is a blocking $s-t$ flow for the original data, as is easily checked.

The running time of the iteration is $O(n+t)$, where t is the number of arcs deleted. Since there are at most $|A|$ iterations and since at most $|A|$ arcs can be deleted, we have the required running time bound. ∎

Hence we have an improvement on the running time for finding a maximum flow:

Corollary 10.6a. *A maximum flow can be found in time $O(n^2m)$.*

Proof. It suffices to describe an $O(nm)$ method to find, for given flow f, a flow f' with $\mu(D_{f'}) > \mu(D_f)$.

Find a blocking flow g in $(V, \alpha(D_f))$. (Note that $\alpha(D_f)$ can be determined in $O(m)$ time.) Let $f'(a) := f(a)+g(a)-g(a^{-1})$, taking values 0 if undefined. Then $D_{f'}$ is a subgraph of $D' := (V, A_f \cup \alpha(D_f)^{-1})$, and hence by (10.16), $\mu(D_{f'}) \geq \mu(D') = \mu(D_f)$. If $\mu(D_{f'}) = \mu(D_f)$, $D_{f'}$ has a path P of length $\mu(D_f)$, which (again (10.16)) should also be a path in $\alpha(D_f)$. But then g could have been increased along this path, contradicting the fact that g is blocking in D_f. ∎

10.6a. Karzanov's $O(n^3)$ algorithm

Karzanov [1974] gave a faster algorithm to find a blocking flow, thus speeding up the maximum flow algorithm. We give the short proof of Malhotra, Kumar, and Maheshwari [1978] (see also Cherkasskiĭ [1979] and Tarjan [1984]).

Theorem 10.7. *Given an acyclic digraph $D = (V, A)$, $s, t \in V$, and a capacity function $c : A \to \mathbb{Q}_+$, a blocking $s-t$ flow can be found in time $O(n^2)$.*

Proof. First order the vertices reachable from s as $s = v_1, v_2, \ldots, v_{n-1}, v_n$ *topologically*; that is, if $(v_i, v_j) \in A$, then $i < j$. This can be done in time $O(m)$ (see Corollary 6.5b).

We describe the algorithm recursively. Consider the minimum of the values $c(\delta^{\text{in}}(v))$ for all $v \in V \setminus \{s\}$ and $c(\delta^{\text{out}}(v))$ for all $v \in V \setminus \{t\}$. Let the minimum be attained by v_i and $c(\delta^{\text{out}}(v_i))$ (without loss of generality). Define $f(a) := c(a)$ for each $a \in \delta^{\text{out}}(v_i)$ and $f(a) := 0$ for all other a.

Next for $j = i+1, \ldots, n-1$, redefine $f(a)$ for each $a \in \delta^{\text{out}}(v_j)$ such that $f(a) \leq c(a)$ and such that $f(\delta^{\text{out}}(v_j)) = f(\delta^{\text{in}}(v_j))$. By the minimality of $c(\delta^{\text{out}}(v_i))$, we can always do this, as initially $f(\delta^{\text{in}}(v_j)) \leq c(\delta^{\text{out}}(v_i)) \leq c(\delta^{\text{out}}(v_j))$. We do this in such a way that finally $f(a) \in \{0, c(a)\}$ for all but at most one a in $\delta^{\text{out}}(v_j)$.

After that, for $j = i, i-1, \ldots, 2$, redefine similarly $f(a)$ for $a \in \delta^{\text{in}}(v_j)$ such that $f(a) \leq c(a)$, $f(\delta^{\text{in}}(v_j)) = f(\delta^{\text{out}}(v_j))$, and $f(a) \in \{0, c(a)\}$ for all but at most one a in $\delta^{\text{in}}(v_j)$.

If $v_i \in \{s, t\}$ we stop, and f is a blocking flow. If $v_i \notin \{s, t\}$, set $c'(a) := c(a) - f(a)$ for each $a \in A$, and delete all arcs a with $c'(a) = 0$ and delete v_i and all arcs incident with v_i, thus obtaining the directed graph $D' = (V', A')$. Obtain

(recursively) a blocking flow f' in D' subject to the capacity function c'. Define $f''(a) := f(a) + f'(a)$ for $a \in A'$ and $f''(a) = f(a)$ for $a \in A \setminus A'$. Then f'' is a blocking flow in D.

This describes the algorithm. The correctness can be seen as follows. If $v_i \in \{s,t\}$ the correctness is immediate. If $v_i \notin \{s,t\}$, suppose that f'' is not a blocking flow in D, and let P be an $s-t$ path in D with $f''(a) < c(a)$ for each arc a in P. Then each arc of P belongs to A', since $f''(a) = f(a) = c(a)$ for each $a \in A \setminus (A' \cup \delta^{\text{in}}(v_i))$. So for each arc a of P one has $c'(a) = c(a) - f(a) > f''(a) - f(a) = f'(a)$. This contradicts the fact that f' is a blocking flow in D'.

The running time of the algorithm is $O(n^2)$, since the running time of the iteration is $O(n + |A \setminus A'|)$, and since there are at most $|V|$ iterations. ∎

Theorem 10.7 improves the running time for finding a maximum flow as follows:

Corollary 10.7a. *A maximum flow can be found in time $O(n^3)$.*

Proof. Similar to the proof of Corollary 10.6a. ∎

Sharper blocking flow algorithms were found by Cherkasskiĭ [1977a] ($O(n\sqrt{m})$), Galil [1978,1980a] ($O((nm)^{2/3})$), Shiloach [1978] and Galil and Naamad [1979,1980] ($O(m \log^2 n)$), Sleator [1980] and Sleator and Tarjan [1981,1983a] ($O(m \log n)$), and Goldberg and Tarjan [1990] ($O(m \log(n^2/m))$), each yielding a maximum flow algorithm with running time bound a factor of n higher.

An alternative approach finding a maximum flow in time $O(nm \log(n^2/m))$, based on the 'push-relabel' method, was developed by Goldberg [1985,1987] and Goldberg and Tarjan [1986,1988a], and is described in the following section.

10.7. Goldberg's push-relabel method

The algorithms for the maximum flow problem described above are all based on flow augmentation. The basis is updating a flow f until D_f has no $s-t$ path. Goldberg [1985,1987] and Goldberg and Tarjan [1986,1988a] proposed a different, in a sense dual, method, the 'push-relabel' method: update a 'preflow' f, maintaining the property that D_f has no $s-t$ path, until f is a flow. (Augmenting flow methods are 'primal' as they maintain feasibility of the primal linear program, while the push-relabel method maintains feasibility of the dual linear program.)

Let $D = (V, A)$ be a digraph, $s, t \in V$, and $c : A \to \mathbb{Q}_+$. A function $f : A \to \mathbb{Q}$ is called an $s-t$ *preflow*, or just a *preflow*, if

(10.19) (i) $0 \le f(a) \le c(a)$ for each $a \in A$,
(ii) $\text{excess}_f(v) \ge 0$ for each vertex $v \ne s$.

(Preflows were introduced by Karzanov [1974]. excess_f was defined in Section 10.1.)

Condition (ii) says that at each vertex $v \ne s$, the outgoing preflow does not exceed the ingoing preflow. For any preflow f, call a vertex v *active* if

$v \neq t$ and $\text{excess}_f(v) > 0$. So f is an $s-t$ flow if and only if there are no active vertices.

The *push-relabel method* consists of keeping a pair f, p, where f is a preflow and $p : V \to \mathbb{Z}_+$ such that

(10.20) (i) if $(u,v) \in A_f$, then $p(v) \geq p(u) - 1$,
(ii) $p(s) = n$ and $p(t) = 0$.

Note that for any given f, such a function p exists if and only if D_f has no $s-t$ path. Hence, if a function p satisfying (10.20) exists and f is an $s-t$ flow, then f is an $s-t$ flow of maximum value (Corollary 10.2a).

Initially, f and p are set by:

(10.21) $f(a) := c(a)$ if $a \in \delta^{\text{out}}(s)$ and $f(a) := 0$ otherwise;
$p(v) := n$ if $v = s$ and $p(v) := 0$ otherwise.

Next, while there exist active vertices, choose an active vertex u maximizing $p(u)$, and apply the following iteratively, until u is inactive:

(10.22) choose an arc $(u,v) \in A_f$ with $p(v) = p(u) - 1$ and *push* over (u,v); if no such arc exists, *relabel* u.

Here to *push* over $(u,v) \in A_f$ means:

(10.23) if $(u,v) \in A$, reset $f(u,v) := f(u,v) + \varepsilon$, where $\varepsilon := \min\{c(u,v) - f(u,v), \text{excess}_f(u)\}$;
if $(v,u) \in A$, reset $f(v,u) := f(v,u) - \varepsilon$, where $\varepsilon := \min\{f(v,u), \text{excess}_f(u)\}$.

To *relabel* u means:

(10.24) reset $p(u) := p(u) + 1$.

Note that if A_f has no arc (u,v) with $p(v) = p(u) - 1$, then we can relabel u without violating (10.20).

This method terminates, since:

Theorem 10.8. *The number of pushes is $O(n^3)$ and the number of relabels is $O(n^2)$.*

Proof. First we show:

(10.25) throughout the process, $p(v) < 2n$ for each $v \in V$.

Indeed, if v is active, then D_f contains a $v-s$ path (since f can be decomposed as a sum of incidence vectors of $s-v$ paths, for $v \in V$, and of directed circuits). So by (10.20)(i), $p(v) - p(s) \leq \text{dist}_{D_f}(v,s) < n$. As $p(s) = n$, we have $p(v) < 2n$. This gives (10.25), which directly implies:

(10.26) the number of relabels is at most $2n^2$.

To estimate the number of pushes, call a push (10.23) *saturating* if after the push one has $f(u,v) = c(u,v)$ (if $(u,v) \in A$) or $f(v,u) = 0$ (if $(v,u) \in A$). Then:

(10.27) the number of saturating pushes is $O(nm)$.

For consider any arc $a = (u,v) \in A$. If we increase $f(a)$, then $p(v) = p(u) - 1$, while if we decrease $f(a)$, then $p(u) = p(v) - 1$. So meantime $p(v)$ should have been relabeled at least twice. As p is nondecreasing (in time), by (10.25) we have (10.27).

Finally:

(10.28) the number of nonsaturating pushes is $O(n^3)$.

Between any two relabels the function p does not change. Hence there are $O(n)$ nonsaturating pushes, as each of them makes an active vertex v maximizing $p(v)$ inactive (while possibly a vertex v' with $p(v') < p(v)$ is activated). With (10.26) this gives (10.28). ∎

There is an efficient implementation of the method:

Theorem 10.9. *The push-relabel method finds a maximum flow in time* $O(n^3)$.

Proof. We order the vertex set V as a doubly linked list, in order of increasing value $p(v)$. Moreover, for each $u \in V$ we keep the set L_u of arcs (u,v) in A_f with $p(v) = p(u) - 1$, ordered as a doubly linked list. We also keep with each vertex v the value $\mathrm{excess}_f(v)$, and we keep linked lists of arcs of D incident with v.

Throughout the iterations, we choose an active vertex u maximizing $p(u)$, and we process u, until u becomes inactive. Between any two relabelings, this searching takes $O(n)$ time, since as long as we do not relabel, we can continue searching the list V in order. As we relabel $O(n^2)$ times, we can do the searching in $O(n^3)$ time.

Suppose that we have found an active vertex u maximizing $p(u)$. We next push over each of the arcs in L_u. So finding an arc $a = (u,v)$ for pushing takes time $O(1)$. If it is a saturating push, we can delete (u,v) from L_u in time $O(1)$. Moreover, we can update $\mathrm{excess}_f(u)$ and $\mathrm{excess}_f(v)$ in time $O(1)$. Therefore, as there are $O(n^3)$ pushes, they can be done in $O(n^3)$ time.

We decide to relabel u if $L_u = \emptyset$. When relabeling, updating the lists takes $O(n)$ time: When we reset $p(u)$ from i to $i+1$, then for each arc (u,v) or (v,u) of D, we add (u,v) to L_u if $p(v) = i$ and $(u,v) \in A_f$, and we remove (v,u) from L_v if $p(v) = i+1$ and $(v,u) \in A_f$; moreover, we move u to its new rank in the list V. This all takes $O(n)$ time. Therefore, as there are $O(n^2)$ relabels, they can be done in $O(n^3)$ time. ∎

Further notes on the push-relabel method. If we allow any active vertex u to be chosen for (10.22) (not requiring maximality of $p(u)$), then the bounds of

$O(n^2)$ on the number of relabels and $O(nm)$ on the number of saturating pushes are maintained, while the number of nonsaturating pushes is $O(n^2m)$.

A first-in first-out selection rule was studied by Goldberg [1985], also yielding an $O(n^3)$ algorithm. Theorem 10.9 (using the largest-label selection) is due to Goldberg and Tarjan [1986,1988a], who also showed an implementation of the push-relabel method with dynamic trees, taking $O(nm\log(n^2/m))$ time. Cheriyan and Maheshwari [1989] and Tunçel [1994] showed that the bound on the number of pushes in Theorem 10.8 can be improved to $O(n^2\sqrt{m})$, yielding an $O(n^2\sqrt{m})$ running time bound. Further improvements are given in Ahuja and Orlin [1989] and Ahuja, Orlin, and Tarjan [1989]. The worst-case behaviour of the push-relabel method was studied by Cheriyan and Maheshwari [1989].

10.8. Further results and notes

10.8a. A weakly polynomial bound

Edmonds and Karp [1972] considered the following *fattest augmenting path* rule: choose a flow-augmenting path for which the flow value increase is maximal. They showed that, if all capacities are integer, it terminates in at most $1 + m'\log\phi$ iterations, where ϕ is the maximum flow value and where m' is the maximum number of arcs in any $s-t$ cut. This gives a maximum flow algorithm of running time $O(n^2m\log nC)$, where C is the maximum capacity (assuming all capacities are integer). (For irrational capacities, Queyranne [1980] showed that the method need not terminate.)

Edmonds and Karp [1970,1972] and Dinits [1973a] introduced the idea of *capacity-scaling*, which gives the following stronger running time bound:

Theorem 10.10. *For integer capacities, a maximum flow can be found in time* $O(m^2\log C)$.

Proof. Let $L := \lceil\log_2 C\rceil + 1$. For $i = L, L-1, \dots, 0$, we can obtain a maximum flow f' for capacity function $c' := \lfloor c/2^i\rfloor$, from a maximum flow f'' for capacity function $c'' := \lfloor c/2^{i+1}\rfloor$ as follows. Observe that the maximum flow value for c' differs by at most m from that of the maximum flow value ϕ for $2c''$. For let $\delta^{\mathrm{out}}(U)$ be a cut with $2c''(\delta^{\mathrm{out}}(U)) = \phi$. Then $c'(\delta^{\mathrm{out}}(U)) - \phi \le |\delta^{\mathrm{out}}(U)| \le m$. So a maximum flow with respect to c' can be obtained from $2f''$ by at most m augmenting path iterations. As each augmenting path iteration can be done in $O(m)$ time, and as $\lfloor c/2^L\rfloor = 0$, we have the running time bound given. ∎

With methods similar to those used in Corollary 10.6a, the bound in Theorem 10.10 can be improved to $O(nm\log C)$, a result of Dinits [1973a] and Gabow [1985b]. To see this, observe that the proof of Theorem 10.6 also yields:

Theorem 10.11. *Given an acyclic graph* $D = (V, A)$, $s, t \in V$, *and a capacity function* $c : A \to \mathbb{Z}_+$, *an integer blocking flow* f *can be found in time* $O(n\phi + m)$, *where* ϕ *is the value of* f.

Proof. Consider the proof of Theorem 10.6. We do at most ϕ iterations, while each iteration takes $O(n+t)$ time, where t is the number of arcs deleted. ∎

Hence, similarly to Corollary 10.6a one has:

Corollary 10.11a. *For integer capacities, a maximum flow can be found in time $O(n(\phi+m))$, where ϕ is the maximum flow value.*

Proof. Similar to the proof of Corollary 10.6a. ∎

Therefore,

Corollary 10.11b. *For integer capacities, a maximum flow can be found in time $O(nm \log C)$.*

Proof. In the proof of Theorem 10.10, a maximum flow with respect to c' can be obtained from $2f''$ in time $O(nm)$ (by Corollary 10.11a), since the maximum flow value in the residual graph $D_{f''}$ is at most m. ∎

10.8b. Complexity survey for the maximum flow problem

Complexity survey ($*$ indicates an asymptotically best bound in the table):

$O(n^2mC)$	Dantzig [1951a] simplex method
$O(nmC)$	Ford and Fulkerson [1955,1957b] augmenting path
$O(nm^2)$	Dinits [1970], Edmonds and Karp [1972] shortest augmenting path
$O(n^2m \log nC)$	Edmonds and Karp [1972] fattest augmenting path
$O(n^2m)$	Dinits [1970] shortest augmenting path, layered network
$O(m^2 \log C)$	Edmonds and Karp [1970,1972] capacity-scaling
$O(nm \log C)$	Dinits [1973a], Gabow [1983b,1985b] capacity-scaling
$O(n^3)$	Karzanov [1974] (preflow push); cf. Malhotra, Kumar, and Maheshwari [1978], Tarjan [1984]
$O(n^2\sqrt{m})$	Cherkasskiĭ [1977a] blocking preflow with long pushes
$O(nm \log^2 n)$	Shiloach [1978], Galil and Naamad [1979,1980]
$O(n^{5/3}m^{2/3})$	Galil [1978,1980a]

≫

continued

	$O(nm \log n)$	Sleator [1980], Sleator and Tarjan [1981,1983a] dynamic trees
*	$O(nm \log(n^2/m))$	Goldberg and Tarjan [1986,1988a] push-relabel+dynamic trees
	$O(nm + n^2 \log C)$	Ahuja and Orlin [1989] push-relabel + excess scaling
	$O(nm + n^2\sqrt{\log C})$	Ahuja, Orlin, and Tarjan [1989] Ahuja-Orlin improved
*	$O(nm \log((n/m)\sqrt{\log C} + 2))$	Ahuja, Orlin, and Tarjan [1989] Ahuja-Orlin improved + dynamic trees
*	$O(n^3/\log n)$	Cheriyan, Hagerup, and Mehlhorn [1990,1996]
	$O(n(m + n^{5/3} \log n))$	Alon [1990] (derandomization of Cheriyan and Hagerup [1989,1995])
	$O(nm + n^{2+\varepsilon})$	(for each $\varepsilon > 0$) King, Rao, and Tarjan [1992]
*	$O(nm \log_{m/n} n + n^2 \log^{2+\varepsilon} n)$	(for each $\varepsilon > 0$) Phillips and Westbrook [1993,1998]
*	$O(nm \log_{\frac{m}{n \log n}} n)$	King, Rao, and Tarjan [1994]
*	$O(m^{3/2} \log(n^2/m) \log C)$	Goldberg and Rao [1997a,1998]
*	$O(n^{2/3} m \log(n^2/m) \log C)$	Goldberg and Rao [1997a,1998]

Here $C := \|c\|_\infty$ for integer capacity function c. For a complexity survey for unit capacities, see Section 9.6a.

Research problem: Is there an $O(nm)$-time maximum flow algorithm?

For the special case of *planar* undirected graphs:

	$O(n^2 \log n)$	Itai and Shiloach [1979]
	$O(n \log^2 n)$	Reif [1983] (minimum cut), Hassin and Johnson [1985] (maximum flow)
	$O(n \log n \log^* n)$	Frederickson [1983b]
*	$O(n \log n)$	Frederickson [1987b]

For *directed* planar graphs:

	$O(n^{3/2} \log n)$	Johnson and Venkatesan [1982]
	$O(n^{4/3} \log^2 n \log C)$	Klein, Rao, Rauch, and Subramanian [1994], Henzinger, Klein, Rao, and Subramanian [1997]
*	$O(n \log n)$	Weihe [1994b,1997b]