Determining the maximal flow in a network by the method of preflows

Alexander Karzanov
Federal Research Center “Computer Science and Control” of the Russian Academy of Sciences (FRC CS…
101 PUBLICATIONS **1,533** CITATIONS

The preflow algorithm for the maximum flow problem

Karzanov's max-flow algorithm of 1974 given in [1] is based on a concept of *preflow* introduced there, which is a function on the arcs that may violate, in a certain way, the flow conservation condition in nonterminal nodes. The algorithm, called the *preflow algorithm (method)*, takes advantages of handling preflows on intermediate iterations (rather than handling flows, as in the previous algorithms), due to which the running time of the algorithm reduces to $O(n^3)$ (compared with $O(mn^2)$ for Dinitz' algorithm [2]); hereinafter n and m are the numbers of nodes and arcs in the input digraph $G = (V(G), A(G))$. Subsequently preflows and the idea of push operations have been widely used in other max-flow algorithms (having the same or smaller running time), in particular, in Cherkassky's algorithm [3] (which is slightly faster) and in Goldberg's push-relabel algorithm [4] of the same complexity $O(n^3)$.

Similar to Dinitz's algorithm, the preflow algorithm consists of $O(n)$ *stages* (big iterations), each solving the following *auxiliary problem*: find a *blocking flow* in a *layered* network (i.e. a network where all paths from the source to the sink have the same length; such a network is formed by the nodes and arcs contained in shortest source-to-sink paths of the residual network w.r.t. the current flow). The preflow algorithm solves the auxiliary problem in $O(n^2)$ time, thus yielding the time bound $O(n^3)$ for the whole algorithm.

In fact, the algorithm of finding a blocking flow given in [1] can be slightly modified so as to work with an arbitrary acyclic, not necessarily layered, network, and below we give a description just for this more general situation.

We start with specifying definitions and settings. Consider a network $N = (G, s, t, c)$, where $G = (V, A)$ is a directed graph, s and t are two distinguished nodes in G, the *source* and the *sink*, respectively, and $c : A \to \mathbb{R}_+$ is a nonnegative real function of arc *capacities*. For a function $f : A \to \mathbb{R}_+$, the *excess* of f at a node $v \in V$ is defined to be $\mathrm{ex}_f(v) := \sum_{a\in\delta^{\mathrm{in}}(v)} f(a) - \sum_{a\in\delta^{\mathrm{out}}(v)} f(a)$ (where $\delta^{\mathrm{in}}(v)$ ($\delta^{\mathrm{out}}(v)$) is the set of arcs in G entering (resp. leaving) a node v). Then f is a *flow* from s to t if $f \le c$ (i.e. $f(a) \le c(a)$ for each $a \in A$), and f satisfies $\mathrm{ex}_f(t) \ge 0$ and $\mathrm{ex}_f(v) = 0$ for all $v \in V - \{s, t\}$. We say that f is a *preflow* in N if $f \le c$ and $\mathrm{ex}_f(v) \ge 0$ for all $v \in V - \{s\}$. A flow or preflow f is called *blocking* if any (directed) path from s to t contains at least one *saturated* arc a, i.e. such that $f(a) = c(a)$.

Theorem. *Let $N = (G = (V, A), s, t, c)$ be an acyclic network. A blocking flow f from s to t in N can be found in $O(n^2)$ time.*

Proof. First of all we order the nodes of G topologically, i.e. label them as $v_1, \dots, v_n$ so that $(v_i, v_j) \in A$ imply $i < j$ (this takes $O(m)$ time). One may assume that $s = v_1$, $t = v_n$, and each arc lies on an s–t path and has nonzero capacity.

The algorithm iteratively handles a preflow $f : A \to \mathbb{R}_+$. For each node $v \in V$, the following data are explicitly maintained:

(i) The excess $\mathrm{ex}(v) = \mathrm{ex}_f(v)$.

(ii) A (double-linked) *list* $\mathrm{Out}(v)$. It is formed by the arcs of G leaving v (each arc occurs in the list exactly once). Each arc e can be either *scanned* or *unscanned*. If e

is unscanned, then $f(e) = 0$. One arc in this list is distinguished, called *active* and denoted by $\widetilde{e}_v$. The following condition holds:

(C1) all arcs of $\mathrm{Out}(v)$ before $\widetilde{e}_v$ are scanned, while all arcs after $\widetilde{e}_v$ are unscanned (the arc $\widetilde{e}_v$ itself may be either scanned or not).

Also some arcs in $\mathrm{Out}(v)$ can be labeled as "frozen" (the meaning will be clear later).

(iii) A *stack* $\mathrm{In}(v)$ (to work with on the "last come first serve" basis). Its elements are pairs (e, Δ), where e is an arc entering v and Δ is a nonnegative real. Each arc e entering v may occur in this stack once or several times or it may not occur there at all, and the sum of numbers Δ over the pairs (e, Δ) with the same e is equal to $f(e)$. In particular, if e does not occur in $\mathrm{In}(v)$, then $f(e) = 0$.

The algorithm starts with the function f such that $f(e) = c(e)$ for all arcs e leaving the source s, and $f(e) = 0$ otherwise (in the latter case e is unscanned). Accordingly, for each arc $e = (s, v)$, the pair $(e, f(e))$ is inserted (as a unique element) in the stack $\mathrm{In}(v)$. The initial stacks $\mathrm{In}(v')$ for the remaining nodes v' are empty. Clearly f is a blocking preflow.

The algorithm alternates "pushing" and "balancing" iterations. Although the first iteration is "pushing", it is more convenient for us (and more enlightening) to start with describing a "balancing" iteration.

Balancing. We assume that at the moment of beginning a "balancing" iteration, the following condition holds:

(C2) f is a blocking preflow; moreover, for each node v with $\mathrm{ex}_f(v) > 0$, any v–t path contains a saturated arc e.

Using the topological order on V, we find the node $v = v_i \neq t$ such that $\mathrm{ex}_f(v) > 0$ and $\mathrm{ex}_f(v_j) = 0$ for $j = i+1, \dots, n-1$. If such a node does not exist, then f is already a blocking flow, and the algorithm terminates.

We perform "balancing" at this v so as to reduce the excess at it to zero. To do so, we use the stack $\mathrm{In}(v)$ and decrease the numbers Δ there step by step in a natural way. More precisely, take the last (chronologically) member (e, Δ) of $\mathrm{In}(v)$ and let $\delta := \min\{\Delta, \mathrm{ex}_f(v)\}$. Update $f(e) := f(e) - \delta$, $\Delta(e) := \Delta(e) - \delta$, and $\mathrm{ex}(v) := \mathrm{ex}(v) - \delta$. If the new $\Delta(e)$ becomes 0, we handle the previous pair (e', Δ) in $\mathrm{In}(v)$ is a similar way (whenever $\mathrm{ex}(v)$ is still nonzero), and so on. Eventually, we obtain f with $\mathrm{ex}_f(v) = 0$.

All arcs of G entering v are labeled as "frozen" (which means that the function f on such arcs should not be changed on subsequent iterations).

Pushing. Note that after performing a balancing iteration, the current function f is a preflow and, moreover, a blocking preflow (which can be easily checked), but the second condition in (C2) need not hold. A "pushing" iteration increases f on certain arcs and restores validity of (C2). Recall that the first iteration in the algorithm is "pushing" as well.

We scan the nodes in the increasing order, starting with v_2 (where $v_1 = s$). Every time we meet a node $v = v_i$ with $\mathrm{ex}_f(v) > 0$ (for a current f), we try to reduce the excess at v as much as possible by increasing f on appropriate arcs leaving v. (Note that a growth of f at an arc (v, u) increases the excess at u (which may be zero before). However, since $u = v_j$ for some $j > i$, the node u will be scanned on a subsequent step of this iteration.)

More precisely, we take the active arc $\widetilde{e}_v$ and do the following:

(P) starting from $\widetilde{e}_v$, we scan step by step the (unscanned) arcs in $\mathrm{Out}(v)$ skipping the "frozen" arcs (if they exist); when scanning a current arc $e = (v, u)$ (which is made "active" at this moment), we increase $f(e)$ as much as possible, i.e. letting $\delta := \min\{c(e) - f(e), \mathrm{ex}_f(v)\}$, we update $f(e) := f(e) + \delta$ and $\mathrm{ex}(v) := \mathrm{ex}(v) - \delta$, and accordingly insert the pair (e, δ) in the stack $\mathrm{In}(u)$.

We stop as soon as either the list $\mathrm{Out}(v)$ terminates, or the excess at v becomes zero. In the former case, all arcs in $\mathrm{Out}(v)$ are saturated or "frozen" (and the active arc is formally the last arc), and in the latter case, the current arc $e = (v, u)$ becomes active, the arcs before e are saturated or "frozen", and the arcs after e remain unscanned.

After scanning $v = v_i$, we repeat the procedure with the next vertex $v' = v_j$ $(j > i)$ where the excess w.r.t. the current f is positive. And so on (until we reach the sink t). One can see that the number of operations during a "pushing" iteration is $O(n + q)$, where q is the number of (new) arcs that become saturated at the iteration plus the number of "frozen" arcs skipped when scanning the lists $\mathrm{Out}(v)$.

The following fact is easy.

Lemma 1. *After performing a "pushing" iteration, the obtained f satisfies (C2).*

This implies that the next "balancing" iteration has a correct input, and the whole process of alternating "pushing" and "balancing" iterations is well-defined. The key observation is as follows.

Lemma 2. *Suppose a node $v = v_i$ is handled at some "balancing" iteration. Then for any arc e incident to v, the value $f(e)$ is not changed on subsequent iterations.*

Proof. This relies on the fact that at the moment of balancing f at v, one has $\mathrm{ex}_f(v_j) = 0$ for $j = i+1, \dots, n-1$. Analyzing the algorithm and using this fact, one can realize that for any subsequent function f' and for any arc e' incident to v_j, the value $f'(e')$ cannot be less than $f(e)$. In particular, $f'(e) \geq f(e)$ for each e leaving v.

On the other hand, since all arcs entering v becomes "frozen", the value of a subsequent function f' on an arc e cannot be greater than $f(e)$.

This implies that f preserves on the arcs incident to v (in view of $\mathrm{ex}_f(v) = 0$). ▮

As a result, any node can be balanced at most once, implying that the number of iterations is $O(n)$. Also if an arc $e = (u, v)$ becomes saturated, the number of subsequent operations involving e is $O(1)$ (a possible operation is a decrease of $f(e)$ during balancing v, after which e becomes "frozen" and it can be scanned once during pushing from u).

Thus, using the above-mentioned bound $O(n + q)$ for one "pushing" iteration, we can conclude that to perform all "pushing" iterations takes $O(n^2 + m)$ time. A similar bound is valid for all "balancing" iterations taken together.

Thus, the algorithm runs in $O(n^2 + m)$ time, yielding the theorem. ▮▮

References

[1] A.V. Karzanov, Determining a maximal flow in a network by the method of preflows, *Doklady Akademii Nauk SSSR*, **215**, 1974, 49–52, in Russian. (English translation in *Soviet Math. Dokl.*, **15**, No.2, 1974, 434–437.)

[2] E.A. Dinic, Algorithm for solution of a problem of maximum flow with power estimation, *Doklady Akademii Nauk SSSR*, **194**, 1970, 754–757, in Russian. (English translation in *Soviet Math. Dokl.*, **11**, 1970, 1277–1280.)

[3] B.V. Cherkassky, Algorithm for construction of maximal flow in networks with complexity of $(O(n^2\sqrt{p})$ operations, In: *Mathematical Methods in Economical Research, issue* 7, Nauka, Moscow, 1977, pp. 117–126.

[4] A.V. Goldberg, A new max-flow algorithm, *Technical Report* MIT/LCS/TM-291, Cambridge, 1985.

Dokl. Akad. Nauk SSSR
Tom 215 (1974), No. 1

Soviet Math. Dokl.
Vol. 15 (1974), No. 2

DETERMINING THE MAXIMAL FLOW IN A NETWORK BY THE METHOD OF PREFLOWS

UDC 518.5

A. V. KARZANOV

An algorithm is presented for determining the maximal flow in a network with an upper bound $O(n^3)$ on the number of operations, where n is the number of vertices of the network.

1. A flow network is an oriented graph $G = [V; U]$ with a set of vertices V, $|V| = n$, a set of arcs U, $|U| = p$, a real function, the "transmission capacity" of an arc, $c(u) > 0$, $u \in U$, a "source" $s \in V$, a "sink" $t \in V$. A function $f(u)$, $u \in U$, is called a flow if

1) $0 \leq f(u) \leq c(u) \quad \forall u \in U$,

2) $\operatorname{Div}(x) = \sum_{(x,y)\in U} f((x, y)) - \sum_{(z,x)\in U} f((z, x)) = 0 \quad \forall x \in V/\{s, t\}$.

A flow with greatest output $v(f) = \operatorname{Div}(s) = -\operatorname{Div}(t)$ [1], is called maximal.

In [2] the problem is in effect reduced to the solution of not more than $n - 1$ problems of the following type:

A network is given (a manual of shortest paths) $S_k = [V_k; U_k]$, $|V_k| \leq n$, $|U_k| \leq p$, $1 \leq k \leq n - 1$, with "source" s, "sink" t, and "transmission capacity" $c_k(u)$, $u \in U_k$, such that any vertex, and also any arc, belong to some shortest (oriented) path from s to t (with a number k of arcs). Find a flow f_k such that for any (oriented) path ξ from s to t there is a saturated arc $u \in \xi$, i.e. $f_k(u) = c_k(u)$. f_k is called a *dead-end flow*.

An algorithm is presented for solving this problem in $O(n^2)$ operations.

2. In the manual $S = [V; U]$ (we shall omit the index k in what follows), by the lth layer, $l = 0, 1, \cdots, k$, is meant the set $O_l = \{x: x \in V, x$ being situated at distance l from $s\}$. In accordance with the definition $O_0 = \{s\}$, $O_k = \{t\}$.

Let $\rho(u)$, $u \in U$, be an admissible function, i.e. $0 \leq \rho(u) \leq c(u)$. Let ξ be called ρ-blocked if there is a saturated arc $u \in \xi$, i.e. $\rho(u) = c(u)$. An arc $u = (x, y)$ (respectively, vertex x') is called ρ-blocked if any path ξ of the form $x, u, y, \cdots, t$ (respectively, path ξ' of the form $x', \cdots, t$) is ρ-blocked. Thus a dead-end flow is a flow f for which the source s is f-blocked.

We shall employ the following notation: $\alpha(x)$, $x \in V$, is the set of arcs originating from x, and $\beta(x)$ is the set of incoming arcs at x;

AMS (MOS) subject classifications (1970). Primary 90B10.

$$\overline{V} \rightleftharpoons V/\{s,\ t\}; \qquad \rho(U') \rightleftharpoons \rho(u)|_{U'}, \qquad |\rho(U'(x))| \rightleftharpoons \sum_{u \in U} \rho(u), \qquad U' \subseteq U$$

(the symbol $\rightleftharpoons$ denotes equality by definition).

We call a function $g(u)$, $u \in U$, a *preflow* if the following properties are satisfied:

$P1$. $0 \leq g(u) \leq c(u)$.

$P2$. $|g(\beta(x))| \geq |g(\alpha(x))|$ $\forall x \in \overline{V}$.

$P3$. If $|g(\beta(x))| > |g(\alpha(x))|$, $x \in \overline{V}$ (x is called a *deficient vertex*), then the vertex x is g-blocked.

$P4$. The vertex s is g-blocked.

If the preflow g does not contain deficient vertices, then g is a dead-end flow.

3. Description of the algorithm. A dead-end flow in S is constructed iteratively. The ith iteration, $i = 1, 2, \cdots$, consists of two parts: a) completion to a preflow, b) balancing of the deficient vertices. The function obtained at the ith iteration as a result of completion we shall denote by g_i, and the function resulting from balancing by g_i^{bal}.

Each arc has a flow index: open or closed. If an arc becomes closed, then it remains so until the end of operation of the algorithm. Initially all arcs are open.

We shall consider that the set $M = \{m_1, m_2, \cdots, m_q\}$ is given in the form of a list if a certain ordering $m_{p_1}, m_{p_2}, \cdots, m_{p_q}$ is fixed by the designation of an initial element ($m_{\text{init}} = m_{p_1}$), a terminal element ($m_{\text{ter}} = m_{p_q}$), and for each element m_{p_j}, $j = 1, 2, \cdots, q$, a preceding element ($m_{p_{j-1}}$) and a succeeding element ($m_{p_{j+1}}$); we shall set $m_{p_0} = \varnothing$, $m_{p_{q+1}} = \varnothing$.

The following flow sets are given in the form of lists:

a) $\overline{\alpha}(x)$ $\forall x \in V$ is the set of open unsaturated arcs;

b) $\beta_\Delta(x)$ $\forall x \in \overline{V}$ is a set whose meaning can be made precise as follows. For each $z \in \beta_\Delta(x)$ a real number $\Delta(z) > 0$ is determined, called an *addition*. Initially, $\overline{\alpha}(x) = \alpha(x)$ $\forall x \in V$; $\beta_\Delta(x) = \varnothing$ $\forall x \in \overline{V}$.

Completion to a preflow. Let $i - 1$ iterations, $i \geq 2$, already have been effected, the function g_{i-1}^{bal} determined, and the assertions verified (for $r = i - 1$):

1°. For g_r^{bal} properties $P1$, $P2$, $P4$ are satisfied, and a layer $O_{s(r)}$ is determined such that $\forall x \in \bigcup_{i=1}^{s(r)-1} O_l$ $P3$ is satisfied and in the layers $O_{s(r)+1}, O_{s(r)+2}, \cdots, O_{k-1}$ there are no deficient vertices.

2°. All closed arcs of g_r^{bal} are blocked.

3°. For each arc $u \in \overline{\alpha}(x)$ $\forall x \in V$, except perhaps for the first, $g_r^{\text{bal}}(u) = 0$.

Completion at the ith iteration consists in the construction of a preflow g_i from g_{i-1}^{bal}. We shall say that on the arc $u \in U$ an *active assignment* is produced, if $g_i(u) > g_{i-1}^{\text{bal}}(u)$. We shall say that in all the remaining arcs a *passive assignment* is produced.

For all the arcs incident to vertices of the layers $O_0, O_1, \cdots, O_{s(i-1)-1}$, we assume $g_i = g_{i-1}^{\text{bal}}$ (passive assignment).

A *layered circuit* of the vertices of the layers $O_{s(i-1)}, O_{s(i-1)+1}, \cdots, O_{k-1}$ is produced: first the vertices of the layer $O_{s(i-1)}$ are sorted out, then those of

$O_{s(i-1)+1}$, and so forth up to O_{k-1}, inclusively. Suppose the vertices $x_1, x_2, \cdots, x_N$ have already been traversed in a layered circuit, and let $x_{N+1} \in O_j$ be the next vertex. $g_i(\beta(x))$ is already determined for any vertex $x \in O_j$, and

$$|g_i(\beta(x_{N+1}))| \geqslant |g_{i-1}^{\text{bal}}(\alpha(x_{N+1}))|.$$

a) If $|g_i(\beta(x_{N+1}))| = |g_{i-1}^{\text{bal}}(\alpha(x_{N+1}))|$, then we set $g_i(\alpha(x_{N+1})) = g_{i-1}^{\text{bal}}(\alpha(x_{N+1}))$ (passive assignment).

b) If $|g_i(\beta(x_{N+1}))| < |g_{i-1}^{\text{bal}}(\alpha(x_{N+1}))|$ and $\bar{\alpha}(x_{N+1}) = \varnothing$ then we set $g_i(\alpha(x_{N+1})) = g_{i-1}^{\text{bal}}(\alpha(x_{N+1}))$ (passive assignment).

c) Let $|g_i(\beta(x_{N+1}))| > |g_{i-1}^{\text{bal}}(\alpha(x_{N+1}))|$ and $\bar{\alpha}(x_{N+1}) \neq \varnothing$. Let $\bar{\alpha}(x) = \{u_1, u_2, \cdots, u_q\}$.

We set $g_i(\alpha(x_{N+1})/\bar{\alpha}(x_{N+1})) = g_{i-1}^{\text{bal}}(\alpha(x_{N+1})/\bar{\alpha}(x_{N+1}))$ (passive assignment). For the arcs $u_0 = \varnothing, u_1, u_2, \cdots, u_l$, $0 \leq 1 < q$, suppose g_i has already been determined and $|g_i(\hat{\alpha}(x_{N+1}))| < |g_i(\beta(x_{N+1}))|$, where $\hat{\alpha}(x_{N+1}) = \alpha(x_{N-1})/\bar{\alpha}(x_{N+1}) \cup \{u_0, u_1, \cdots, u_l\}$ is the set of arcs with already given g_i. For the next arc u_{l+1} of the list $\bar{\alpha}(x_{N+1})$ we set

$$g_i(u_{l+1}) = \min\{c(u_{l+1}), |g_i(\beta(x_{N+1}))| - |g_i(\hat{\alpha}(x_{N+1}))|\}$$

(active assignment). We make active assignments up to the arc u_m (inclusive) for which either: 1) $|g_i(\alpha(x_{N+1})/\bar{\alpha}(x_{N+1}) \cup \{u_1, u_2, \cdots, u_m\})| = |g_i(\beta(x_{N+1}))|$, or 2) $m = q$. For the arcs $u_{m+1}, u_{m+2}, \cdots, u_q$ (in case 1) we set $g_i = g_{i-1}^{\text{bal}} = 0$ (passive assignment). The new list $\bar{\alpha}(x_{N+1})$ takes the form $\{u_m, u_{m+1}, \cdots, u_q\}$ if $g_i(u_m) < c(u_m)$, and takes the form $\{u_{m+1}, u_{m+2}, \cdots, u_q\}$ if $m < q$ and $g_i(u_m) \doteq c(u_m)$.

If an active assignment is made on the arc $u = (x_{N+1}, y)$, then u is added to the end of the list $\beta_\Delta(y)$ (even if the arc u is already contained in $\beta_\Delta(y)$) and we define $\Delta(u)$ to be equal to $g_i(u) - g_{i-1}^{\text{bal}}(u)$.

In the construction of g_i we set $g_0^{\text{bal}} \equiv 0$, $s(0) = 0$, $g_i(\alpha(s)) = c(\alpha(s))$ (active assignment) and proceed further in accordance with the completion algorithm.

Balancing. Let $d(i)$ be the *maximal index* of a layer in which g_i has a deficient vertex (x). At the vertex x an operation of balancing is effected, i.e. determination of $g_i^{\text{bal}}(\beta(x))$ so that $|g_i^{\text{bal}}(\beta(x))| = |g_i(\alpha(x))|$ and $g_i^{\text{bal}}(u) \leq g_i(u)$ $\forall u \in \beta(x)$. We effect balancing at the vertex x in accordance with the list $\beta_\Delta(x)$, starting from its terminus and decreasing g_i at the expense of "additions". Let u_l be the next element in $\beta_\Delta(x)$ and suppose $|g_i^{\text{bal}}(\hat{\beta}(x))| + |g_i(\beta(x)/\hat{\beta}(x))| > |g_i(\alpha(x))|$, where $\hat{\beta}(x) \subset \beta(x)$ is the set of arcs for which g_i^{bal} is defined. We set

$$g_i^{\text{bal}}(u_l) = \max\{g_i(u_l) - \Delta(u_l), |g_i(\alpha(x)) - g_i^{\text{bal}}(\hat{\beta}(x))| - |g_i(\beta(x)/\hat{\beta}(x)/u_l)|\}$$

(active assignment) and proceed to u_{l-1}, and so on until $|g_i^{\text{bal}}(\beta(x))| + |g_i(\beta(x)/\hat{\beta}(x))| = |g_i(\alpha(x))|$, where $\hat{\beta}(x)$ is the (new) set with already determined g_i^{bal}. For all $u \in \beta(x)/\hat{\beta}(x)$ we put $g_i^{\text{bal}}(u) = g_i(u)$ (passive assignment). Each arc $u = (y, x) \in \beta(x)$ is "closed" and is stricken from the list $\alpha(y)$ (if it is present in it).

If there are deficient vertices in the layer $O_{d(i)}$, then we effect balancing independently for each one. For all arcs with not already designated g_i^{bal} we set $g_i^{\text{bal}} = g_i$ (passive assignment). We set $s(i) = d(i) - 1$.

Lemma 1. 1) *g_i is a preflow.* 2) *assertions* 1°–3° *are valid for $r = i$.* 3) *If a vertex $x \in V$ is g_i^{bal}-blocked, then it is g_i-blocked and g_i^{bal}-blocked.*

The lemma is proved by induction on i (for $i = 1$ the proof is immediate).

To each "addition" Δ we associate the index $e(\Delta)$ of that iteration at which it occurred as a result of completion. Let $k_{i,j}$ be the maximal index of the additions relative to all the arcs incoming to vertices of the layer O_j before commencement of balancing in the ith iteration.

Lemma 2. *In the balancing at the ith iteration an active assignment can be made only for those arcs $u \in \beta_\Delta(x)$, $x \in O_{d(i)}$, for which $e(\Delta(u)) = k_{i,d(i)} g_i^{\text{bal}}(u) \geq g_i(u) - \Delta(u)$.*

With the aid of Lemmas 1 and 2 one can prove the

Theorem. *At each vertex $x \in \overline{V}$ balancing is carried out not more than once.*

It follows from the theorem that the number of iterations is not greater than $n - 2$. The final function is a flow, and since s is g_i-blocked, it follows from Lemma 1 that this flow is dead-end.

4. Estimate of the number of operations. In a passive assignment no operations are effected. Owing to the utilization of lists, the number of operations in the construction of a dead-end flow is $\eta = O(n^2 + \overline{\eta})$, where η is the number of active assignments. From the Theorem it follows that $\overline{\eta} = O(p + \eta_a)$, where η_a is the number of active assignments in the completions. We say that as a result of an active assignment in completion on an arc u the event A occurs if the arc u is saturated, and the event B occurs if it is not saturated.

Lemma 3. 1) *The total number of events A is not greater than p.*
2) *In the course of one iteration in $\alpha(x)$, $\forall x \in V$ not more than one event B occurs.*

From Lemma 3 it follows that $\eta_a = p + \eta'$ and $\eta' = O(n^2)$, where η' is the total number of events B, whence it follows that $\eta = O(n^2 - \eta') = O(n^2)$.

There is a modification of the method presented for finding a dead-end flow for which $\eta = O(p + \eta')$. One can construct extremal examples of networks supporting the precision of the estimate $O(n^3)$ for finding the maximal flow by means of the given and modified methods.

Institute for Control Problems Received 16/APR/73

BIBLIOGRAPHY

1. L. R. Ford, Jr. and D. R. Fulkerson, *Flows in networks*, Princeton Univ. Press, Princeton, N. J., 1962. MR 28 #2917.

2. E. A. Dinic, *Algorithm for solution of a problem of maximum flow in a network with power estimation*, Dokl. Akad. Nauk SSSR 194 (1970), 754–757 = Soviet Math. Dokl. 11 (1970), 1277–1280. MR 44 #5178.

Translated by R. F. RINEHART