Computer Science Journal of Moldova, vol.**9**, no.3(27), 2001

Algorithms for minimum flows *

Eleonor Ciurea Laura Ciupal

Abstract

We present a generic preflow algorithm and several implementations of it, that solve the minimum flow problem in $O(n^2m)$ time.

AMS Mathematics Subject Classification: 90B10, 90C35, 05C35, 68R10

Key words: Network flow, network algorithms, minimum flow, preflow, residual capacity, distance label.

1 Minimum Flow Problem

We consider a capacitated network $G = (N, A, l, c, s, t)$ with a nonnegative capacity $c(i, j)$ and with a nonnegative lower bound $l(i, j)$ associated with each arc $(i, j) \in A$. We distinguish two special nodes in the network G: a source node s and a sink node t. A *flow* is a function $f : A \to \Re_+$ satisfying the next conditions:

$$f(i, N) - f(N, i) = \begin{cases} v, & i = s \\ 0, & i \neq s, t \\ -v, & i = t \end{cases} \tag{1a}$$

$$l(i, j) \leq f(i, j) \leq c(i, j) \qquad \forall (i, j) \in A \tag{1b}$$

for some $v \geq 0$. We refer to v as the *value* of the flow f. The minimum flow problem is to determine a flow f for which v is minimized.

*The paper was prepared after the report presented at the I Conference of the Mathematical Society of Moldova

For solving the minimum flow problem we will use the following definitions: A *preflow* is a function $f : A \to \Re_+$ that satisfies (1b) and

$$f(i,N) - f(N,i) \leq 0, \qquad \forall i \in N \setminus \{s,t\} \tag{2}$$

The algorithm that we will describe next maintains a preflow at each intermediate stage. For any preflow f, we define the *deficit* of the node i as

$$e(i) = f(i,N) - f(N,i), \qquad \forall i \in N \tag{3}$$

We refer to a node i with $e(i) = 0$ as *balanced*. The *residual capacity* $r(i,j)$ of any arc $(i,j) \in A$, with respect to a given preflow f, is given by $r(i,j) = c(j,i) - f(j,i) + f(i,j) - l(i,j)$. The residual capacity of the arc (i,j) represents the maximum amount of flow from the node i to node j that can be cancelled. The network $G_f = (N, A_f)$ consisting only of the arcs with positive residual capacity is referred to as the *residual network* (with respect to preflow f).

A *cut* is a partition of the node set N into two subsets S and $\overline{S} = N - S$; we represent this cut using the notation $[S,\overline{S}]$. We refer to a cut $[S,\overline{S}]$ as a *$s-t$ cut* if $s \in S$ and $t \in \overline{S}$. We refer to an arc (i,j) with $i \in S$ and $j \in \overline{S}$ as a *forward arc* of the cut, and an arc (i,j) with $i \in \overline{S}$ and $j \in S$ as a *backward arc* of the cut. Let $(S,\overline{S})$ denote the set of forward arcs in the cut, and let $(\overline{S},S)$ denote the set of backward arcs.

We define the *capacity* $c[S,\overline{S}]$ of a $s-t$ cut $[S,\overline{S}]$ as the sum of the lower bounds of the forward arcs minus the sum of the capacities of the backward arcs. That is, $c[S,\overline{S}] = l(S,\overline{S}) - c(\overline{S},S)$ We refer to a $s-t$ cut whose capacity is maximum among all $s-t$ cuts as a *maximum cut.*

Theorem 1.1 (Min-Flow Max-Cut Theorem). *The minimum value of the flow from a source node s to a sink node t in a capacitated network with nonnegative lower bounds equals the maximum capacity among all $s-t$ cuts.*

This theorem can be proved in a manner similar to the proof of the Max-Flow Min-Cut Theorem.

We refer to a path in G from the source node s to the sink node t as a *decreasing path* if it consists only in arcs with positive residual capacity. Clearly, there is a one-to-one correspondence between decreasing paths in G and directed paths from s to t in G_f. Given a decreasing path P in G, we can decrease the current flow f in the following manner:

$$f(i,j) = \begin{cases} f(i,j) - r, & \text{if}(i,j) \text{ is a forward arc in } P \\ f(i,j) + r, & \text{if}(i,j) \text{ is a backward arc in } P \\ f(i,j), & \text{if}(i,j) \text{ is not an arc in } P \end{cases}$$

where $r = \min\{r(i,j) \mid (i,j) \in P\}$ is the residual capacity of the decreasing path P.

Theorem 1.2. (Decreasing Path Theorem) *A flow f^* is a minimum flow if and only if the residual network G_{f^*} contains no directed path from the source node to the sink node.*

This theorem can be proved in a manner similar to the proof of the Increasing Path Theorem.

The minimum flow problem in a network can be solved in two phases:
(1) establish a feasible flow
(2) from a given feasible flow, establish the minimum flow

Establishing a Feasible Flow

The problem of determining a feasible flow consists in finding a function $f : A \to \Re_+$ that satisfies conditions (1a) and (1b). First, we transform this problem into a circulation problem by adding an arc (t,s) of infinite capacity and zero lower bound. This arc carries the flow sent from node s to node t back to node s. Clearly, the minimum flow problem admits a feasible flow if and only if the circulation problem admits a feasible circulation. Because these two problems are equivalent, we focus on finding a feasible circulation if it exists. The feasible circulation problem is to identify a flow f satisfying these following constraints:

$$f(i,N) - f(N,i) = 0, \quad \text{for every node} \quad i \in N \tag{4a}$$

$$l(i,j) \leq f(i,j) \leq c(i,j), \quad \text{for every arc } (i,j) \in A \tag{4b}$$

By replacing $f(i,j) = f'(i,j) + l(i,j)$ in (4a) and (4b) we obtain the following transformed problem:

$$f'(i,N) - f'(N,i) = b(i), \quad \text{for every node } i \in N \tag{5a}$$

$$0 \leq f'(i,j) \leq c(i,j) - l(i,j), \quad \text{for every arc } (i,j) \in A \tag{5b}$$

with the supplies/demands $b(\cdot)$ at the nodes defined by $b(i) = l(N,i) - l(i,N)$.

Clearly, $\sum_{i \in N} b(i) = 0$. We can solve this feasible flow problem by solving a maximum flow problem defined in a transformed network. We introduce two new nodes: a source node s' and a sink node t'. For each node i with $b(i) > 0$ we add source arc (s', i) with capacity $b(i)$ and for each node i with $b(i) < 0$ we add sink arc (i, t') with capacity $-b(i)$. Than we solve a maximum flow problem in this transformed network. If the maximum flow saturates all the source and the sink arcs, then the initial problem has a feasible solution (which is the restriction of the maximum flow that saturates all the source and sink arcs to the initial set of arcs A); otherwise it is infeasible.

Establishing a Minimum Flow

There are two approaches for solving maximum flow problem: (1) using augmenting path algorithms and (2) using preflow-push algorithms. From the first algorithms we can easily obtain algorithms for minimum flow by replacing augmenting paths with decreasing paths. The second type of algorithms for maximum flow does not permit such an easy transformation. In the next section we present a generic preflow algorithm for minimum flow and we suggest several implementations of it.

2 Generic Preflow Algorithm for Minimum Flow

Before describing the generic preflow algorithm, we introduce some defi-nitions. In the residual network G_f, the *distance function* $d : N \to \mathcal{N}$ with respect to a given preflow f is a function from the set of nodes to the nonnegative integers. We say that a distance function is *valid* if it satisfies the following conditions:

$$d(s) = 0 \tag{6a}$$

$$d(j) \leq d(i) + 1, \quad \text{for every arc } (i,j) \in A_f \tag{6b}$$

We refer to $d(i)$ as the *distance label* of node i.

Lemma 2.1 *(a) If the distance labels are valid, the distance label $d(i)$ is a lower bound on the length of the shortest directed path from node s to node i in the residual network.*

(b) If $d(t) \geq n$, the residual network contains no directed path from the source node to the sink node.

Proof. (a) Let $P = (s = i_1, i_2, \ldots, i_k, i_{k+1} = i)$ be any path of length k from node s to node i in the residual network. The validity conditions imply that:

$$\begin{array}{ccccccccccc}
d(i_2) & \leq & d(i_1) & + & 1 & = & d(s) & + & 1 & = & 1 \\
d(i_3) & \leq & d(i_2) & + & 1 & \leq & 2 & & & & \\
d(i_4) & \leq & d(i_3) & + & 1 & \leq & 3 & & & & \\
\ldots & & & & & & & & & & \\
d(i_{k+1}) & \leq & d(i_k) & + & 1 & \leq & k & & & &
\end{array}$$

(b) Since $d(t)$ is a lower bound on the length of the shortest path from s to t in the residual network and no directed path can contain more than $(n-1)$ arcs, imply that if $d(t) \geq n$, then the residual network contains no directed path from s to t. ■

We say that the distance labels are *exact* if for each node $i, d(i)$ equals the length of the shortest path from node s to node i in the residual network. We refer to an arc (i,j) from the residual network as an *admissible* arc if $d(j) = d(i) + 1$; otherwise it is *inadmissible.* We refer to a path from node s to node t consisting entirely of admissible arcs as an *admissible path.*

Lemma 2.2 *An admissible path is a shortest decreasing path from the source to the sink.*

Proof. Since every arc (i,j) in an admissible path P is admissible, the residual capacity of this arc and the distance labels of its end nodes satisfy the following conditions:
(1) $r(i,j) > 0$
(2) $d(j) = d(i) + 1$
Condition (1) implies that P is a decreasing path and condition (2) implies that if P contains k arcs then $d(t) = k$. Since $d(t)$ is a lower bound on the length of any path from the source to the sink in the residual network (from Lemma 2.1 (a)), the path P must be a shortest decreasing path. ■

We refer to a node i with $e(i) < 0$ as an *active* node. We adopt the convention that the source node and the sink node are never active.

The generic algorithm for the minimum flow problem is the following:

```
Generic Algorithm;
Begin
    let $f$ be a feasible flow in network $G$;
    compute the exact distance labels $d(\cdot)$ in the residual network $G_f$;
    if $t$ is not labeled then
        $f$ is a minimum flow
    else
        begin
            for each arc $(i,t) \in A$ do
```

```
                $f(i,t) := l(i,t)$;
            $d(t) := n$;
            **while** the network contain an active node **do**
                **begin**
                    select an active node $j$;
                    **if** the network contain an admissible arc $(i,j)$ **then**
                        pull $g = \min(-e(j), r(i,j))$ units of flow from node
$j$
                        to node $i$;
                    **else** $d(j) = \min\{d(i) \mid (i,j) \in A_f\} + 1$
                **end**
        **end**
**end**.
```

A pull of g units of flow from node j to node i increases both $e(j)$ and $r(j,i)$ by g units and decreases both $e(i)$ and $r(i,j)$ by g units. We refer to the process of increasing the distance label of node j, $d(j) = \min\{d(j) \mid (i,j) \in A_f\} + 1$, as a *relabel operation.* This algorithm begins with a feasible flow and sends back as much flow, as it is possible, from the sink node to the source node. Because the algorithm decreases the flow on individual arcs, it does not satisfy the mass balance constraint (1a) at intermediate stages. In fact, it is possible that the flow entering in a node exceeds the flow leaving from it. The basic operation of this algorithm is to select an active node and to send the flow entering in it back, closer to the source. For measuring closeness, we will use the distance labels $d(\cdot)$. Let j be a node with strictly negative deficit. If it exists an admissible arc (i,j), we pull flow on this arc; otherwise we relabel the node j so that we create at least one admissible arc.

Theorem 2.3 *The generic algorithm computes correctly a minimum flow.*

Proof. The algorithm terminates when the network does not contain any active nodes, therefore the current preflow is a flow. Since $d(t) := n$, the residual network contains no directed path from the source node to the sink node and Theorem 1.2 implies that the obtained flow is a minimum flow. ∎

Actually, the algorithm terminates with optimal residual capacities. From these residual capacities we can determine a minimum flow in several ways. For example, we can make a variable change: For all arcs (i, j), let $c'(i, j) = c(i, j) - l(i, j)$, $r'(i, j) = r(i, j)$ and $f'(i, j) = f(i, j) - l(i, j)$. The residual capacity of arc (i, j) is

$$r(i, j) = c(j, i) - f(j, i) + f(i, j) - l(i, j).$$

Equivalently,

$$r'(i, j) = c'(j, i) - f'(j, i) + f'(i, j).$$

Similarly,

$$r'(j, i) = c'(i, j) - f'(i, j) + f'(j, i).$$

We can compute the value of f' in several ways; for example

$$f'(i, j) = \max(r'(i, j) - c'(j, i), 0)$$

and

$$f'(j, i) = \max(r'(j, i) - c'(i, j), 0)$$

Converting back into the original variables, we obtain the following expressions:

$$f(i, j) = l(i, j) + \max(r(i, j) - c(j, i) + l(j, i), 0)$$

and

$$f(j, i) = l(j, i) + \max(r(j, i) - c(i, j) + l(i, j), 0)$$

We refer to a pull of flow from node j to node i as a *cancelling pull* if it deletes the arc (i, j) from the residual network; otherwise it is a *noncancelling pull.* After the initializations the nodes adjacent to the sink have negative deficits, so the algorithm can select an active node.

Complexity of the Algorithm

Lemma 2.4 *For each node i, $d(i) < 2n$.*

Proof. Each node i with negative deficit is connected to node t by a directed path P from t to i in the residual network. Since $d(t) = n$ and $d(l) \leq d(k) + 1$, for each admissible arc (k, l), $d(i) \leq d(t) + |P| < 2n$. ■

Since each time the algorithm relabels a node i, $d(i)$, increases at least 1 unit, we have established the following result:

Lemma 2.5 *Each distance label increases at most n times. Consequently, the total number of relabel operations is at most $2n^2$.*

Lemma 2.6 *The algorithm performs at most nm cancelling pulls.*

Proof. We will show that between two consecutive cancelling of an arc (i, j) both $d(i)$ and $d(j)$ must increase by at least 2 units. Since the algorithm increases each distance label at most $2n$ times, this result would imply that the algorithm could drop any arc from the residual network at most n times. Consequently, the total number of cancelling pulls would be at most nm.

Suppose that a pull drops the arc (i, j) from the residual network. Since the arc (i, j) is admissible,

$$d(j) = d(i) + 1.$$

Before the algorithm cancels this arc again, it must pull the flow from node i to node j. At this time the distance labels $d'(i)$ and $d'(j)$ satisfy the equality

$$d'(i) = d'(j) + 1$$

In the next cancelling of arc (i, j) we must have

$$d''(j) = d''(i) + 1.$$

By using these relations, we obtain

$$d''(j) = d''(i) + 1 \geq d'(i) + 1 \geq d'(j) + 2 \geq d(j) + 2$$

and

$$d''(i) = d''(j) - 1 \geq d(j) + 2 - 1 = d(i) + 2. \quad ■$$

Lemma 2.7 *The generic algorithm performs $O(n^2m)$ noncancelling pulls.*

Proof. Let I be the set of active nodes. We consider the potential function $\Phi = \sum_{i \in I} d(i)$. Since $|I| < n$ and $d(i) < 2n$ for all $i \in I$, the initial value of Φ is at most $2n^2$. At the termination of the algorithm, $\Phi = 0$. During the execution of the algorithm, one of the following two cases must apply:
Case 1: The algorithm is unable to find an admissible arc, in which case the distance label of node j increases by $\epsilon \geq 1$ units. This operation increases Φ by ϵ units. Since the total increase in $d(j)$ for each node j throughout the execution of the algorithm is bounded by $2n$, the total increase in Φ due to increases in distance labels is bounded by $2n^2$.
Case 2. The algorithm is able to identify an admissible arc and it can pull flow from node j to node i. If the pull cancels the arc (i, j) then it might create a new deficit at node i, thereby increasing the number of active nodes by 1 and increasing Φ by $d(i)$, which could be as much as $2n$ per pull and so $2n^2m$ over all cancelling pulls. Note that a noncancelling pull from node j to node i does not increase $|I|$. The noncancelling pull decreases Φ by $d(j)$ since j becomes inactive, but it simultaneously increases Φ by $d(i) = d(j) - 1$ if the pull causes node i to become active, the total decrease in Φ being of value 1. If node i was active before the pull, Φ decreases by an amount equal to $d(j)$. Consequently, the net decrease in Φ is at least 1 unit per cancelling pull. Since the initial value of Φ is at most $2n^2$, the maximum possible increase in Φ is $2n^2 + 2n^2m$, each cancelling pull decreases Φ by at least 1 unit and Φ always remains nonnegative, the algorithm can perform at most $2n^2 + 2n^2 + 2n^2m = O(n^2m)$ noncancelling pulls. ∎

Theorem 2.8 *The generic algorithm runs in $O(n^2m)$ time.*

Proof. We can maintain the set L of active nodes organized as simply or doubly linked list, so that the algorithm can add, delete or select elements from it in $O(1)$ time. Consequently, in view of lemmas above, it is easy to implement the generic algorithm in $O(n^2m)$ time. ∎

We suggest a practical improvement. We define a *minimum preflow* as a preflow with the minimum possible flow outgoing from the source node.The generic algorithm for minimum flow performs pull operations and relabel operations at active node until all the deficit reaches the source node or returns to the sink node. Typically, the algorithm establishes a minimum preflow long before it establishes a minimum flow. After establishing a minimum preflow, the algorithm performs relabel operations at active nodes until their distance label are sufficiently higher than n so it can pull flow back to the sink node. We can modify the generic algorithm in the following manner: we maintain a set N' of nodes that satisfy the property that the residual network contains no path from the source node to a node in N'. Initially, $N' = \{t\}$. We add nodes to N' in the following way: let *numb* (k) be the number of nodes whose distance label is k. We can update *numb*$(\cdot)$ in $O(1)$ steps per relabel operation. Moreover, whenever *numb* $(k') = 0$ for some k' any node j with $d(j) > k'$ is disconnected from the set of nodes i with $d(i) < k'$ in the residual network. So, we can add any node j with $d(j) > k'$ to the set N'. We do not perform pull or relabel operations for nodes in N' and terminate the algorithm when all nodes in $N \setminus N'$ are inactive. At this point, the current preflow is a minimum preflow. By the flow decomposition theory, any preflow f can be decomposed into a sequence of at most $O(n+m)$ paths and cycles. Let S be such a set of augmenting paths and cycles. Let $S_0 \subseteq S$ be a set of paths which start at a deficit node and terminate at sink node; let f^0 be the flow contributed by these path flows. Then $f^* = f + f^0$ will be a minimum flow.

3 Several Implementations of the Generic Preflow Algorithm

The generic algorithm does not specify a rule for selecting active nodes. By specifying different rules we can develop many different algorithms, which can be better than the generic algorithm. For example, we could select active nodes in FIFO order, or we could always select the active node with the greatest distance label, or the active node with the minimum distance label, or the active node selected most recently or least recently, or the active node with the largest excess or we could select any of active nodes with a *sufficiently large* excess.

FIFO Preflow Algorithm for MinimumFlow

In an iteration, the generic algorithm for minimum flow selects a node, say node i, and performs a cancelling or a noncancelling pull, or relabels the node. If the algorithm performs a cancelling pull, then node i might still be active, but it is not mandatory for the algorithm to select this node again in the next iteration. We can incorporate the rule that whenever the algorithm selects an active node, it keeps pulling flow from that node until either its deficit becomes zero or the algorithm relabels the node. We refer to a sequence of cancelling pulls followed either by a noncancelling pull or a relabel operation as a *node examination.*

The FIFO preflow algorithm for minimum flow examines active nodes in FIFO order. The algorithm maintains the set L of the active nodes as a queue. It selects an active node i from the front of L, performs pulls from this node and adds newly active nodes to the rear of L. The algorithm terminates when the queue of active nodes is empty.

The FIFO preflow algorithm for the minimum flow problem is the following:

```
FIFO Preflow Algorithm;
Begin
  let $f$ be a feasible flow in network $G$;
  compute the exact distance labels $d(\cdot)$ in the residual network $G_f$;
  if $t$ is not labeled then
    $f$ is a minimum flow
  else
    begin
      $L := \emptyset$;
      for each arc $(i,t) \in A$ do
        begin
```

```
                $f(i,t) := l(i,t)$;
                **if** $e(i) < 0$ **and** $(i \neq s)$ **then**
                    add $i$ to the rear of $L$;
            **end;**
        $d(t) := n$;
        **while** $L \neq \emptyset$ **do**
            **begin**
                remove the node $j$ from the front of the queue $L$;
                pull / relabel $(j)$;
            **end**
        **end**
**end**.
**procedure** pull/relabel $(j)$;
**begin**
    select the first arc $(i,j)$ that enters in node $j$;
    $B := 1$;
    **repeat**
        **if** $(i,j)$ is an admissible arc **then**
            **begin**
                pull $g = \min(-e(j), r(i,j))$ units of flow from node $j$ to node $i$;
                **if** $i \notin L$ **and** $i \neq s$ **and** $i \neq t$ **then**
                    add $i$ to the rear of $L$;
            **end;**
        **if** $e(j) < 0$ **then**
            **if** $(i,j)$ is not the last arc that enters in node $j$ **then**
                select the next arc $(i,j)$ that enters in node $j$
            **else begin**
                    $d(j) = \min\{d(i) \mid (i,j) \in A_f\} + 1$;
                    $B := 0$;
                        **end;**
**until** $(e(j) < 0)$ **or** $(B = 0)$;
**if** $e(j) < 0$ **then**
    add $j$ to the rear of $L$;
**end**;
```

Theorem 3.1 *The FIFO preflow algorithm computes correctly a minimum flow.*

Proof. The correctness of the FIFO preflow algorithm follows from the correctness of the generic preflow algorithm (Theorem 2.3). ■

Theorem 3.2 *The FIFO preflow algorithm runs in $O(n^3)$ time.*

Proof. To analyze the worst-case complexity of this algorithm, we partition the total number of node examinations into different phases. The first phase consists of node examinations for those nodes that become active during the initialization at the beginning of the algorithm. The second phase consists of the node examinations of all the nodes that are in the queue after the algorithm has examined the nodes in the first phase. The third phase consists of the node examinations of all the nodes that are in the queue after the algorithm has examined the nodes in the second phase, and so on.

To bound the number of phases, we consider the difference between the initial and final value of the potential function $\phi = \max\{d(i) \mid i$ is active $\}$ during a phase. We consider two cases.
Case 1: The algorithm performs at least one relabel operation during a phase. Then ϕ might increase by as much as the maximum increase in any distance label. Lemma 2.5 implies that the total increase in ϕ is at most $2n^2$.
Case 2: The algorithm performs no relabel operation during a phase. In this case the deficit of every node that was active at the beginning of the phase moves to nodes with smaller distance labels. Consequently, ϕ decreases by at least 1 unit.

Since the initial value of ϕ is at most n, combining cases 1 and 2, we find that the total number of phases is at most $2n^2 + n$. In each phase, any node is examined at most once and, during each node examination, the algorithm performs at most one noncancelling pull. Since the number of phases is at most $2n^2 + n$, the number of noncancelling pulls is $O(n^3)$. Therefore, the FIFO preflow algorithm runs in $O(n^3)$ time because the bottleneck operation in the generic preflow algorithm is the number of noncancelling pulls. ■

Highest Label Preflow Algorithm for Minimum Flow

The highest-label preflow algorithm always pulls flow from an active node with the highest distance label. Let $h = \max\{d(i) \mid i$ is active $\}$. The algorithm first examines nodes with distance label equal to h and pulls the flow from these nodes to nodes with distance labels equal to $h-1$ and, from these nodes, to the nodes with distance labels equal to $h-2$ and so on until the algorithm relabels a node or it has exhausted all the active nodes. When it has relabeled a node, the algorithm repeats the same process. If the algorithm does not relabel any node during n consecutive node examinations, all the deficit reaches the source or the sink and the algorithm terminates. Since the algorithm performs at most $2n^2$ relabel operations, we obtain a bound of $O(n^3)$ on the number of node examinations. Since each node e-xamination entails at most one noncancelling pull, the highest-label preflow algorithm performs $O(n^3)$ noncancelling pulls. Therefore, it runs in $O(n^3)$ time.

The highest-label preflow algorithm is the same as the FIFO preflow algorithm, with the set L implemented not as a queue, but as a priority queue.

References

[1] Ahuja, R., Magnanti, T., Orlin, J.: *Network Flow. Theory, Algorithms and Applications.* New Jersey, Prentice Hall, 1993.

[2] Ahuja, R., Orlin, J., Tarjan, R.: *Improved Time Bounds for the Maximum Flow Problem.* SIAM Journal of Computing, 18 (1988), pp.939–954.

[3] Ahuja, R., Orlin, J.: *A Fast and Simple Algorithm for the Maximum Flow Problem.* Operation Research, 37 (1988), pp.748–759.

[4] Ahuja, R., Magnanti, T., Orlin, J.; *Some Recent Advances in Network Flows.* SIAM Review, 33 (1990), pp.175–219.

[5] Goldberg, A. V.: *A New Max-Flow Algorithm.* Cambridge, MIT, 1985.

[6] Goldberg, A. V., Tarjan, R.E.: *A New Approach to the Maximum Flow Problem.* Journal of ACM, 35 (1988), pp.921–940.

[7] Karzanov, A.V.: *Determining the Maximum Flow in a Network by the Method of Preflows.* Soviet. Math. Dokl., 15 (1974), pp.434–437.

[8] Tarjan, R. E.: *A Simple Version of Karzanov's Blocking Flow Algorithm*, 1984.

Eleonor Ciurea, Laura Ciupal,
"Transilvania" University,
Brasov, Romania

Received September 25, 2001