About preflow algorithms for the minimum flow problem

LAURA CIUPALĂ
Department of Computer Science
University Transilvania of Braşov
Iuliu Maniu Street 50, Braşov
ROMANIA
laura_ciupala@yahoo.com

ELEONOR CIUREA
Department of Computer Science
University Transilvania of Braşov
Iuliu Maniu Street 50, Braşov
ROMANIA
e.ciurea@unitbv.ro

Abstract: - In this paper, we describe the highest-label preflow algorithm for minimum flow. This algorithm is a special implementation of the generic preflow algorithm developed by Ciurea and Ciupală in [8], obtained by imposing in the generic preflow algorithm the rule that the algorithm must always select an active node with the highest distance label. Our new algorithm runs in $O(n^2m^{1/2})$ time, which is substantially better than the running time of the generic preflow algorithm, that is $O(n^2m)$. Moreover, the highest-label preflow algorithm is the fastest polynomial algorithm for minimum flow problem.

Key-Words: - Network flow; Network algorithms; Minimum flow problem; Scaling technique

1 Introduction

The literature on network flow problem is extensive. Over the past 50 years researchers have made continuous improvements to algorithms for solving several classes of problems. From the late 1940s through the 1950s, researchers designed many of the fundamental algorithms for network flow, including methods for maximum flow and minimum cost flow problems. In the next decades, there are many research contributions concerning improving the computational complexity of network flow algorithms by using enhanced data structures, techniques of scaling the problem data etc.

Although it has its own applications, the minimum flow problem was not dealt so often as the maximum flow ([1], [2], [10], [11], [12], [13], [14]) and the minimum cost flow problem ([1], [4], [15]). There are many problems that occur in economy that can be reduced to minimum flow problems.

For instance, we present the machine setup problem. A job shop needs to perform p tasks on a particular day. It is known the start time $\pi(i)$ and the end time $\pi'(i)$ for each task i, i=1,...,p. The workers must perform these tasks according to this schedule so that exactly one worker performs each task. A worker cannot work on two jobs at the same time. It is known the setup time $\pi_2(i, j)$ required for a worker to go from task i to task j. We wish to find the minimum number of workers to perform the tasks.

We can formulate this problem as a minimum flow problem in the network $G=(N, A, l, c, s, t)$, where $N=N_1\cup N_2\cup N_3\cup N_4$, $N_1=\{s\}$, $N_2=\{i \mid i=1,...,p\}$, $N_3=\{i' \mid i'=1,...,p\}$, $N_4=\{t\}$, $A=A_1\cup A_2\cup A_3\cup A_4$, $A_1=\{(s, i) \mid i\in N_2\}$, $A_2=\{(i, i') \mid i, i'=1,...,p\}$, $A_3=\{(i', j) \mid \pi'(i')+\pi_2(i', j)\le\pi(j)\}$, $A_4=\{(i', t) \mid i'\in N_3\}$, $l(s, i)=0$, $c(s, i)=1$, for any $(s, i)\in A_1$, $l(i, i')=1$, $c(i, i')=1$, for any $(i, i')\in A_2$, $l(i', j)=0$, $c(i', j)=1$, for any $(i', j)\in A_3$, $l(i', t)=0$, $c(i', t)=1$, for any $(i', t)\in A_4$.

We solve the minimum flow problem in the network $G=(N, A, l, c, s, t)$ and the value of the minimum flow is the minimum number of workers that can perform the tasks.

The minimum flow problem in a network can be solved in two phases:

(1) establishing a feasible flow, if there is one

(2) from a given feasible flow, establish the minimum flow.

The problem of determining a feasible flow can be reduced to a maximum flow problem (for details see [1]).

For the second phase of the minimum flow problem there are three approaches:

1. using decreasing path algorithms (see [8], [9])
2. using preflow algorithms (see [3], [5], [8], [9])
3. finding a maximum flow from the sink node to the source node in the residual network (see [2], [6]).

The preflow algorithms for the minimum flow are more efficient than the decreasing path algorithms. In [8], Ciurea and Ciupală presented a generic preflow algorithm that runs in $O(n^2m)$ time and a special implementation of it: FIFO preflow algorithm that runs in $O(n^3)$. In [3], Ciupală developed a deficit scaling which is a special implementation of the generic preflow algorithm for minimum flow that pulls flow from active nodes with sufficiently large deficits. This algorithm runs in $O(nm+n^2 \log C)$ time.

In this paper, first we describe the known algorithms for the minimum flow problem and after this we introduce the highest-label preflow algorithm for the minimum flow problem. This algorithm is a special implementation of the generic preflow algorithm for minimum flow (described in [8], [9]). As its name suggests, it always pulls flow from the active node with the highest distance label and it runs in $O(n^2m^{1/2})$ time. Consequently, the highest-label preflow algorithm is the fastest polynomial algorithm for minimum flow problem.

2 Notation and definition

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.

Let $n=|N|$, $m = |A|$ and $C = \max \{ c(i, j) \mid (i, j) \in A\}$.

A *flow* is a function $f : A \rightarrow \mathbf{R}_+$ satisfying the next conditions:

$$f(s, N) - f(N, s) = v \qquad (1)$$

$$f(i, N) - f(N, i) = 0,\ i \neq s,t \qquad (2)$$

$$f(t, N) - f(N, t) = -v \qquad (3)$$

$$l(i, j) \leq f(i, j) \leq c(i, j),\ (i, j)\in A \qquad (4)$$

for some $v \geq 0$, where

$$f(i, N) = \Sigma_j f(i, j),\ i\in N$$

and

$$f(N, i) = \Sigma_j f(j, i),\ i\in N.$$

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.

For the minimum flow problem, a *preflow* is a function $f : A \rightarrow \mathbf{R}_+$ satisfying the next conditions:

$$f(i, N) - f(N, i) \leq 0,\ i \neq s,t \qquad (5)$$

$$l(i, j) \leq f(i, j) \leq c(i, j),\ (i, j)\in A \qquad (6)$$

Let f be a preflow. We define the *deficit* of a node $i\in N$ in the following manner:

$$e(i) = f(i, N) - f(N, i) \qquad (7)$$

Thus, for the minimum flow problem, for any preflow f, we have $e(i) \leq 0$, $i\in N \setminus\{s, t\}$.

We say that a node $i\in N \setminus\{s, t\}$ is *active* if $e(i) < 0$ and *balanced* if $e(i) = 0$.

A preflow f for which

$$e(i) = 0,\ i\in N \setminus\{s, t\}$$

is a flow. Consequently, a flow is a particular case of preflow.

For the minimum flow problem, the *residual capacity* $r(i, j)$ of any arc $(i, j)\in A$, with respect to a given preflow f, is given by

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

By convention, if $(j, i)\notin A$ then we add arc (j, i) to the set of arcs A and we set $l(j, i) = 0$ and $c(j, i) = 0$. The residual capacity $r(i, j)$ of the arc (i, j) represents the maximum amount of flow from the node i to node j that can be canceled. 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).

In the residual network $G_f = (N, A_f)$ the *distance function* $d : N \rightarrow \mathbf{N}$ 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:

$$d(s) = 0 \qquad (8)$$

$$d(j) \leq d(i) + 1, \text{ for every arc } (i, j) \in A_f. \qquad (9)$$

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

Theorem 1[3](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.

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 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.

3 Generic preflow algorithm

This algorithm, developed by Ciurea and Ciupală in [8], 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 constraints (1), (2) and (3) 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 the flow on this arc; otherwise we relabel the node j so that we create at least one admissible arc.

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

```
Generic 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
            for each arc $(i, t) \in A$ do
                $f(i, t) := l(i, t)$;
            $d(t) := n$;
            while the network contains an active node
            do begin
                select an active node $j$;
                if the network contains 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, $d(j) := \min\{ d(i) \mid (i, j) \in A_f \}+1$, as a *relabel operation*.

Theorem 2 *If there is a feasible flow in the network $G = (N, A, l, c, s, t)$, the generic preflow 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(b) 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

$c'(i, j) = c(i, j) - l(i, j)$,

$r'(i, j) = r(i, j)$,

$f'(i, j) = f(i, j) - l(i, j)$.

The residual capacity of arc (i, j) is

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

Equivalently,

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

We can compute the value of f' in the following way:

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

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

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

We refer to a pull of flow from node j to node i as a *canceling pull* if it deletes the arc (i, j) from the residual network; otherwise it is a *noncanceling pull*.

Theorem 3 *The generic preflow algorithm runs in $O(n^2m)$ time.*

Proof. To analyze the complexity of the generic preflow algorithm, we begin by establishing three important results:

(1) the total number of relabel operations is at most $2n^2$.

(2) the algorithm performs at most nm canceling pulls.

(3) the algorithm performs $O(n^2m)$ noncanceling pulls.

This results can be proved in a manner similar to the proof of complexity of the generic preflow algorithm for the maximum flow problem.

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, it is easy to implement the generic algorithm in $O(n^2m)$ time.

In [8], Ciurea and Ciupală suggested a practical improvement. We defined 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 nodes 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 s to a node in

N’. Initially, *N’* = {*t*}. We add nodes to *N’* in the following way: let $nb(k)$ be the number of nodes whose distance label is *k*. We can update $nb(\cdot)$ in $O(1)$ steps per relabel operation. Moreover, whenever $nb(k') = 0$ 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 active nodes are in *N’*. 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.

The generic preflow algorithm does not specify a rule for selecting active nodes. By specifying different rules we can develop many different algorithms, which can be better then the generic algorithm. For example, we could select active nodes in FIFO order or we could always select the active node with the highest distance label or we could select an active node with a sufficiently large excess.

4 FIFO Preflow Algorithm for Minimum Flow

In an iteration, the generic preflow algorithm for minimum flow selects a node, say node *j*, and performs a canceling or a noncanceling pull, or relabels the node. If the algorithm performs a canceling pull, then node *j* might still be active, but, in the next iteration, the algorithm may select another active node for performing a pull or a relabel operation. We can establish 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 canceling pulls followed either by a noncanceling pull or a relabel operation as a *node examination*.

The FIFO preflow algorithm for minimum flow developed by Ciurea and Ciupală in [8], examines active nodes in FIFO order. The algorithm maintains the set *L* of the active nodes as a queue. It selects an active node *j* 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 resi-
   dual 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 entering 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 4 *If there is a feasible flow in the network $G = (N, A, l, c, s, t)$, 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).

Theorem 5 *The FIFO preflow algorithm runs in $O(n^3)$ time.*

Proof. This theorem can be proved in a manner similar to the proof of the complexity of the FIFO preflow algorithm for the maximum flow.

5 Deficit Scaling Algorithm

This algorithm is also a special implementation of the generic preflow algorithm for minimum flow and, like all preflow algorithms for minimum flow, it maintains a preflow at every step and proceeds by pulling the deficits of the active nodes closer to the source node. For measuring closeness it uses the exact distance labels. Consequently, pulling the deficits from the active nodes closer to the source node means decreasing flow on admissible arcs.

Let $e_{\max} = \max\ \{-e(i) \mid i \text{ is an active node}\}$.

The *deficit dominator* is the smaller integer $\bar{r}$ that is a power of 2 and satisfies $e_{\max} \le \bar{r}$. We refer to a node i with $e(i) \le -\bar{r}/2$ as a node with *large deficit* and as a node with *small deficit* otherwise.

The scaling deficit algorithm for the minimum flow always pulls flow from active nodes with sufficiently large deficits to nodes with sufficiently small deficits in order to not allow that a deficit becomes too large.

The deficit scaling algorithm for the minimum flow is the following:

```
Deficit scaling 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 $f(i, t) := l(i, t)$;
      $d(t) := n$;
      $\bar{r} := 2^{\lceil \log C \rceil}$;
      while $\bar{r} \ge 1$ do
      begin
          while the network contains an active
          node with a large deficit do
          begin
              among active nodes with large
              deficits, select a node $j$ with the
              smallest distance label;
              if the network contains an
              admissible arc $(i, j)$ then
                  if $i \ne t$ then
                      pull $g = \min\{-e(j), r(i, j),$
                      $\bar{r} + e(i)\}$ units of flow from
                      node $j$ to node $i$;
                  else
                      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
          $\bar{r} := \bar{r}/2$;
      end
  end
end.
```

Let us refer to a phase of the algorithm during which $\bar{r}$ remains constant as a scaling phase and a scaling phase with a specific value of $\bar{r}$ as a $\bar{r}$-*scaling phase.*

Theorem 6 *If there exists a feasible flow in the network $G = (N, A, l, c, s, t)$, then the deficit scaling algorithm determines a minimum flow.*

Proof. The algorithm starts with $\bar{r} := 2^{\lceil \log C \rceil}$, $C \le \bar{r} \le 2C$. During the $\bar{r}$-scaling phase, $e_{\max}$ might increase or decrease but it must meet the condition $\bar{r}/2 < e_{\max} \le \bar{r}$. When $e_{\max} \le \bar{r}/2$, the algorithm halves the value of $\bar{r}$ and begins a new scaling phase. After $1+\lceil \log C \rceil$ scaling phases, $e_{\max}$ becomes 0 and we obtain a minimum flow.

Actually, the algorithm terminates with optimal residual capacities. From these optimal residual capacities, we can determine a minimum flow as we did in Section 3.

Theorem 7 *During each $\bar{r}$ - scaling phase, the algorithm satisfies the following two conditions:*
(a) each noncanceling pull decreases the flow by at least $\bar{r}/2$ units
(b) $e_{\max} \le \bar{r}$.

Proof. (*a*) We consider a noncanceling pull on arc (i, j). Since (i, j) is an admissible arc, $d(j) = d(i) + 1 > d(i)$. But, j is a node with a smallest distance label among all nodes with a large deficit. Thus, $e(j) \le -\bar{r}/2$ and $e(i) > -\bar{r}/2$. Since this pull is a

noncanceling pull, it decreases the flow by $\min \{-e(j), \bar{r}+e(i)\} \geq \bar{r}/2$.
(*b*) A pull on arc (i, j) increases only the absolute value of the deficit of node *i*. The new deficit of node *i* is $e'(i) = e(i) - \min \{-e(j), r(i, j), \bar{r}+e(i)\} \geq e(i) - (\bar{r}+e(i)) = -\bar{r}$. Thus, $e'(i) \geq -\bar{r}$ and $e_{\max} \leq \bar{r}$.

Theorem 8 *During each scaling phase, the algorithm performs $O(n^2)$ noncanceling pulls.*

Proof. We consider the potential function $F = -\Sigma_{i\in N}\, e(i)d(i)/\ \bar{r}$. The initial value of *F* at the beginning of the $\bar{r}$-scaling phase is bounded by $2n^2$ because $e(i) \geq -\bar{r}$ and $d(i) \leq 2n$ for all $i \in N$. After the algorithm has selected node *j*, one of the following two cases must apply:
Case 1. The algorithm is unable to find an admissible arc along which it can pull flow. In this case the distance label of node *j* increases by $q \geq 1$ units. This increases *F* by at most *q* units because $e(i) \geq -\bar{r}$. Since for each *i* the total increase in $d(i)$ throughout the running of the algorithm is bounded by $2n$, the total increase in *F* due to relabelings of nodes is bounded by $2n^2$.
Case 2. The algorithm is able to find an admissible arc along which it can pull flow, so it performs either a canceling or a noncanceling pull. In either case, *F* decreases. After a noncanceling pull on arc (i, j), the flow from node *i* to node *j* decreases by at least $\bar{r}/2$ units and *F* decreases by at least 1/2 units because $d(j) = d(i) + 1$. As the initial value of *F* at the beginning of the scaling phase plus the increase in *F* sum to at most $4n^2$, this case cannot occur more than $8n^2$ times. Thus, the algorithm performs $O(n^2)$ noncanceling pulls per scaling phase.

Theorem 9 *The deficit scaling algorithm runs in $O(nm+n^2 \log C)$ time.*

Proof. Since the algorithm performs $O(\log C)$ scaling phases, from Theorem 8 it follows that the algorithm performs $O(n^2 \log C)$ noncanceling pulls in total. The other operations (canceling pulls, relabel operations and finding admissible arcs) require $O(nm)$ time (this can be proved in a similar way as Ciurea and Ciupală proved the complexity of the generic preflow algorithm in [9]). Consequently, the deficit scaling algorithm runs in $O(nm+n^2 \log C)$ time.

6 Highest-Label Preflow Algorithm

The highest-label preflow algorithm for minimum flow examines always an active node with the highest distance label. The algorithm maintains the set *L* of the active nodes as a priority queue, with priority *d*. It selects an active node *j* with the highest priority from the priority queue *L*, performs pulls from this node and adds newly active nodes to *L*. The algorithm terminates when the priority queue of active nodes is empty.

The highest-label preflow algorithm always pulls flow from an active node with the highest distance label. Let $h = \max \{d(i) \mid i \text{ is active}\}$. 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 analyzed 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.

The highest-label preflow algorithm for the minimum flow problem is the following:

```
Highest-Label Preflow Algorithm;
Begin
   let $f$ be a feasible flow in network $G$;
   compute the exact distance labels $d(\cdot)$ in the resi-
     dual 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 $i$ with priority $d(i)$ to the
                  priority queue $L$;
        end;
        $d(t) := n$;
        while $L \neq \emptyset$ do
        begin
             remove the node $j$ with the highest
             priority from the priority 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 $i$ with priority $d(i)$ to the
                priority queue $L$;
        end;
        if $e(j) < 0$ then
            if $(i, j)$ is not the last arc entering 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 $j$ with priority $d(j)$ to the priority queue $L$;
end;
```

Theorem 10 *If there is a feasible flow in the network $G = (N, A, l, c, s, t)$, the highest-label preflow algorithm computes correctly a minimum flow.*

Proof: The correctness of the highest-label preflow algorithm follows from the correctness of the generic preflow algorithm.

Actually, the algorithm terminates with optimal residual capacities. From these optimal residual capacities, we can determine a minimum flow as we did in Section 3.

Theorem 11 *The highest-label preflow algorithm runs in $O(n^2m^{1/2})$ time.*

Proof. This theorem can be proved in a manner similar to the proof of the complexity of the highest-label preflow algorithm for the maximum flow. (see [1]).

Consequently, by using the priority queues instead of queues in the FIFO preflow algorithm, we obtained an algorithm which is substantially faster than the FIFO preflow algorithm and the generic preflow algorithm for minimum flow problem.

7 Conclusion

In this paper, first we described the known algorithms for the minimum flow problem and after this we introduced the highest-label preflow algorithm for the minimum flow problem. This algorithm is a special implementation of the generic preflow algorithm for minimum flow (described in [8]), obtained by imposing in the generic preflow algorithm the rule that the algorithm must always select an active node with the highest distance label. Our new algorithm runs in $O(n^2m^{1/2})$ time, which is substantially better than the running time of the generic preflow algorithm, that is $O(n^2m)$. Moreover, the highest-label preflow algorithm is the fastest polynomial algorithm for minimum flow problem. Table 1 summarizes the preflow algorithms for determining minimum flow and their complexities.

Table 1

Preflow algorithm for minimum flow	Running time
Generic preflow algorithm	$O(n^2m)$
FIFO preflow algorithm	$O(n^3)$
Deficit scaling algorithm	$O(nm+n^2 \log C)$
Highest-label preflow algorithm	$O(n^2m^{1/2})$

References:

[1] R. Ahuja, T. Magnanti and J. Orlin, *Network flows. Theory, algorithms and applications*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1993.

[2] J. Bang-Jensen and G. Gutin, *Digraphs: Theory, Algorithms and Applications*, Springer-Verlag, London, 2001.

[3] L. Ciupală, A deficit scaling algorithm for the minimum flow problem, *Sadhana* Vol.31, No. 3, 2006, pp.1169-1174.

[4] L. Ciupală, A scaling out-of-kilter algorithm for minimum cost flow, *Control and Cybernetics* Vol.34, No.4, 2005, pp. 1169-1174.

[5] L. Ciupală and E. Ciurea, A highest-label preflow algorithm for the minimum flow problem, *Proceedings of the 11th WSEAS International Conference on Computers*, 2007, pp. 565-569.

[6] L. Ciupală and E. Ciurea, An algorithm for the minimum flow problem, *The Sixth International Conference of Economic Informatics*, 2003, pp. 167-170.

[7] L. Ciupală and E. Ciurea, An approach of the minimum flow problem, *The Fifth*

International Symposium of Economic Informatics, 2001, pp. 786-790.

[8] E. Ciurea and L. Ciupală, Sequential and parallel algorithms for minimum flows, *Journal of Applied Mathematics and Computing* Vol.15, No.1-2, 2004, pp. 53-78.

[9] E. Ciurea and L. Ciupală, Algorithms for minimum flows, *Computer Science Journal of Moldova* Vol.9, No.3(27), 2001, pp. 275-290.

[10] S. Fujishige, A maximum flow algorithm using MA ordering, *Oper. Res. Lett.* 31, No. 3, 176-178, 2003.

[11] S. Fujishige and S. Isotani, New maximum flow algorithms by MA orderings and scaling, *J. Oper. Res. Soc. Japan* 46, No. 3, 243-250, 2003.

[12] A. V. Goldberg and R. E. Tarjan, A New Approach to the Maximum Flow Problem, *Journal of ACM* Vol.35, 1988, pp. 921-940.

[13] S. Kumar and P. Gupta, An incremental algorithm for the maximum flow problem, *J. Math. Model. Algorithms* 2, No.1, 1-16, 2003.

[14] A. Schrijver, On the history of the transportation and maximum flow problems, *Math. Program.* 91, No.3, 437-445, 2002.

[15] K.D. Wayne, A polynomial Combinatorial Algorithm for Generalized Minimum Cost Flow, *Math. Oper. Res.*, 445-459, 2002.