BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **7**, No 3

Sofia • 2007

A Fast Algorithm for Solving the Optimum Circulation Problem

Dimiter Ivanchev

New Bulgarian University, Department of Telecommunications, 1618 Sofia
E-mail: divanchev@nbu.bg

Abstract: *A modification of an algorithm for solving the well-known optimum circulation problem in directed networks is presented here. It uses the Prim's algorithm in the initialization step for finding an optimum spanning tree according to one or two criteria in a lexicographical sense on a graph with special arc weights. The modification makes it much faster. The running time is* $O\big((m-n+1)mn\big)+O(Kmn^2)$ *where* m *and* n *are the arc and node numbers, respectively, and* K *is a constant, depending on the input data. A computational experiment is accomplished with series of randomly generated networks up to 10 000 arcs and the average time is compared with that of the Out-of-Kilter algorithm.*

Keywords: *Network programming, optimum circulation, complexity, computer experiments.*

1. Introduction and problem statement

The optimum circulation problem is one of the most important ones in mathematical programming and especially in network optimization. This is due to the fact that many other problems can stem from this one as a special case. On the other hand the problem is a polynomial one and there are efficient polynomial and pseudo polynomial time running algorithms for its solution.

About 70 real applications of this problem and variety of algorithms are mentioned in [1]. They can be summarized as three types: basic algorithms [1, Ch. 9], polynomial algorithms [1,Ch. 10] and network simplex algorithms [1, Ch. 11], [7, 8]. A short overview is given at the end of this section.

The problem statement is the following. Let $G = (V, A)$ be a directed graph with vertices $V_1, V_2, ..., V_n$ and arcs $A_1, A_2, ..., A_m$ with three arc weights a_k, b_k and c_k for each arc A_k, where: a_k is a lower bound of the arc capacity, b_k – an upper bound of the arc capacity, and c_k – transportations costs. We assume that $a_k \le b_k$ for all arcs.

Without any loss of generality we assume that a_k, b_k and c_k are integers. We denote by CC_i^+ and CC_i^- the arc subsets of the Co-Cycle CC_i, containing all arcs leaving and entering vertex V_i, respectively.

We have to find an integer $x = (x_1, x_2, ..., x_m)$, which minimizes

(1) $$F_C(x) = \sum_{j=1}^{m} c_j x_j$$

subject to

(2) $$\sum_{A_k \in \mathrm{CC}_i^+} x_k - \sum_{A_k \in \mathrm{CC}_i^-} x_k = 0, \quad i = 1, 2, ..., n$$

and

(3) $$a_k \le x_k \le b_k, \quad k = 1, 2, ..., m.$$

The known basic algorithms for solving this problem are given in [1]: out-of-kilter algorithm, cycle canceling algorithm, successive shortest path algorithm, primal-dual algorithm and relaxation algorithm. As a main feature they repeatedly find the shortest paths (or cycles) and the time complexity depends on a constant connected with the input data, i.e. they are pseudo polynomial.

The theoretical advantage of the strongly polynomial algorithms is that they can solve problems with irrational data. This is not the case in our consideration. Six algorithms are given below [1]. The first three ones are weakly polynomial, i.e. the time-complexity depends logarithmically on the input data and the last three ones are strongly polynomial: capacity scaling algorithm, cost scaling algorithm, double scaling algorithm, minimum mean cycle cancelling algorithm, repeated capacity scaling algorithm and enhanced capacity scaling algorithm. The last one seems to be the fastest known polynomial one – running time $O((m \log n)(m + n \log n))$. It is a hybrid version of the first and the fifth one above mentioned.

The network simplex algorithms rely on the fact that there always exists a spanning tree solution, i.e. for which there is a spanning tree containing all the arcs for which (3) is fulfilled as a strong inequality. This is the reason to restrict the search among all spanning tree solutions.

A network simplex algorithm with some related variants is presented in [1]. Some special cases are discussed for solving the shortest path problem, the maximum flow problem and sensitivity analysis.

Two simplex type algorithms are presented in [7] and [8] for non-flow capacitate networks.

In [5] the authors have proposed a simplex type algorithm dealing with basic feasible solutions.

The algorithm which is described below, is a pseudo polynomial one and not of simplex type. It has a relatively good complexity and a good real running time.

2. The algorithm

An integer node number t_i (dual variable, potential) is assigned to each node, to each arc $A_k = (i, j)$ an arc number is assigned

(*) $$d_k := t_j - t_i - c_k,\ k = 1, 2, ..., m\ .$$

If c_k in (1) is substituted by (*), it will be easy to see that

$$F_C(x) = -F_D(x) := -\sum_{j=1}^{m} d_j x_j$$

is satisfied. In this way we reformulate the problem as

(1*) $$F_D(x) := \sum_{j=1}^{m} d_j x_j \rightarrow \max$$

subject to (2) and (3).

Let $x = (x_1, x_2, ..., x_m)$ be a feasible circulation, i.e. for which (2) and (3) holds and $t = (t_1, t_2, ..., t_n)$ – a potential vector. The condition

(4) $$\begin{cases} d_k > 0 \Rightarrow x_k = b_k \\ \qquad\qquad\quad , \quad k = 1, 2, ..., m, \\ d_k < 0 \Rightarrow x_k = a_k \end{cases}$$

is called an optimality condition for the problem since the following theorem is true.

Theorem 1. If x is a feasible circulation and t is a potential vector such that (4) is fulfilled then x is an optimum circulation. If (2)and (3) are not contradictable then there exists an optimum circulation x and a potential vector t such that (4) holds.

P r o o f. Sufficiency – see [4, p. 233].

For the second part of the statement we start the out-of-kilter procedure with a feasible circulation and an arbitrary potential vector. It finishes with an optimum circulation x and a potential vector t, which satisfy (4).

Thus, to each arc $A_k = (i, j)$, a point (x_k, d_k), is assigned, which can be in nine states – three of them are optimal ones ((2), (3) and (4) are fulfilled), the others are not (see [4]). The optimum states are called In-Kilter States, the other ones are Out-of-Kilter States.

The curve

(5) $$KC: \{(x, y) \mid x = a_k \text{ if } y < 0; x = b_k \text{ if } y > 0; a_k < x < b_k \text{ if } d_k = 0\}$$

is called a *kilter-curve* of A_k. Its integer points can be moved to the kilter-curve horizontally if we change the flow on this arc and vertically, if we change the potentials of its nodes.

The algorithm Out-of-Kilter (OKA) keeps condition (2) fulfilled at each iteration and tries to fulfill (3) and (4) [4]. The algorithm given below, called Modified External Flow Algorithm (MEFA) keeps (3) and (4) and tries to fulfill (2) [2, 3].

For any arbitrary $x = (x_1, x_2, ..., x_m)$ and V_i

$$F_i := \sum_{A_k \in CC_i^+} x_k - \sum_{A_k \in CC_i^-} x_k, \quad i = 1, 2, ..., n,$$

and will call it external flow of V_i. The node is called a source node if F_i is negative. If it is positive the node is a sink one, otherwise it is a transit node.

To each node V_i we attach labels S_i and E_i – integers with the meaning:

$S_i = 0$ means that the node is not labeled; otherwise, $S_i > 0$ means that there exists a chain from a special starting node to V_i and the last arc is a forward one. If it is negative, the last arc is a backward one. E_i is the maximum quantity of the flow changed on the last arc of the chain.

Further, for each arc $A_k = (i, j)$ are denoted $\mathrm{SV}_k := i$ and $\mathrm{EV}_k := j$ (Start Vertex and End Vertex of A_k). For an arc A_k with $d_k = 0, S_i \neq 0, S_j = 0$ and $E_i \neq F_i$ three logical variables are defined as follows:

$\mathrm{CondF}(A_k, i, j) :=$ true iff

1) $\mathrm{SV}_k := i$ and $\mathrm{EV}_k := j$,

2) if $E_i > F_i$ then $x_k < b_k$, else $x_k > a_k$;

$\mathrm{CondB}(A_k, i, j) :=$ true iff

1) $\mathrm{SV}_k := j$ and $\mathrm{EV}_k := i$,

2) if $E_i > F_i$ then $x_k > a_k$ else $x_k < b_k$;

$\mathrm{Cond}(A_k, i, j) := \mathrm{CondF}(A_k, i, j) \vee \mathrm{CondB}(A_k, i, j)$.

The idea of the algorithm is the following. At the initialization step a maximum spanning tree $\hat{G}$ is found with arc weights $b_k - a_k$, which allows to have at least one flow augmenting chain between every two nodes for the defined flow x. Further (3) and (4) are preserved fulfilled and an attempt is made to reduce all external flows to zero, i.e. to fulfill (2) as well. In the Main loop we start with a source node V_l and using the strategy Depth First Search (DFS) all nodes reachable from it with one flow augmenting chain are found. If V_j is such one and it is not possible to continue the DFS procedure with the forward step, the algorithm either goes backward if it is not possible to reduce the external flow of V_j, or this

vertex is dislabeled, its external flow is reduced and the flow of the last arc leading to it changed and the DFS procedure continued. In this way we get a tree with labeled nodes which "pulsates" (gets larger or smaller). At the end of DFS execution we stop at V_l. If this node becomes a transit one, we simply jump to another source node and start the next iteration. Otherwise we change potentials of the labeled nodes and continue the procedure with a growing tree.

2. 1. Modified External Flow Algorithm (MEFA)

(*Input: Directed and connected graph $G=(V,\ A)$ and integers a_k, b_k, c_k with $a_k \le b_k$;

Output: Optimum circulation $x=(x_1, x_2, ..., x_m)$.*)

1^0. (*Initialization – finding a good non – feasible starting solution, which satisfies* (3) *and* (4)*)

Set $\hat{G} := (\hat{V}, \hat{A})$ with $\hat{V} := \{V_1\}$, $\hat{A} := \{\ \}$; set $t_1 := 0$;

for $l := 1$ to $n-1$ do

begin

find $A_k = (i, j)$ with largest $b_k - a_k$ such that

$(V_i \in \hat{V}$ and $V_j \notin \hat{V})$ or $(V_i \notin \hat{V}$ and $V_j \in \hat{V})$;

in first case set $t_j := t_i + c_k$ and $\hat{V} := \hat{V} \cup \{V_j\}$,

in second case set $t_i := t_j - c_k$ and $\hat{V} := \hat{V} \cup \{V_i\}$;

set $x_k := \left[\frac{a_k + b_k}{2}\right]$, $d_k := 0$ and $\hat{A} := \hat{A} \cup \{A_k\}$

end ;

for all $A_k \notin \hat{A}$ set $d_k := t_j - t_i - c_k$ and

if $d_k > 0$ then $x_k := b_k$ else if $d_k < 0$ then $x_k := a_k$

else $x_k := \left[\frac{a_k + b_k}{2}\right]$;

for $i := 1$ to n do set $F_i := 0$ and $E_i := 0$;

for $k := 1$ to m do

if $A_k = (i, j)$ then set $F_i := F_i + x_k$ and $F_j := F_j - x_k$;

2^0. (* *Main loop* *)

for $l := 1$ to n do

if $F_l \neq 0$ then

begin

$E_l := 0$; GoOn := true, $i := j$;

repeat $(*\, until\ F_l = 0\, *)$

if $i = l$ and GoOn then

begin

for $j = 1$ to n do $S_j = 0$;

GoOn := false, $S_l := m + 1$

end ;

find an arc A_k with $Cond(A_k, i, j)$ = true ;

if A_k is such an arc then $(*\, Forward\ step\, *)$

if $CondF(A_k, i, j)$ then

begin $S_j := k$;

if $E_i > F_i$ then $E_{j_i} := \min\{b_k - a_k, E_i - F_i\}$

else $E_j := \max\{a_k - x_k, E_i - F_i\}$; $i := j$

end

else $(*\, CondB(A_k i, j) = true\, *)$

begin $S_j := -k$;

if $E_i > F_i$ then $E_j := \min\{x_k - a_k, E_i - F_i\}$

else $E_j := \max\{x_k - b_k, E_i - F_i\}$; $i := j$

end

else $(*\, Cond(A_k i, j) = \ false\ for\ all\ A_k \in CC_i\, *)$

begin $k := S_i$;

If $i \neq l$ then

If $E_i F_i > 0$ then

begin (*Backward step : flow change and dislabeling*)

$\varepsilon := \text{sign}(F_i) \min\{|E_i|, |F_i|\}$;

$F_i := F_i - \varepsilon$; $GoOn := true$;

$S_i := 0$;

if $k > 0$ then

begin $x_k := x_k + \varepsilon$; $i := SV_k$ end

else begin $x_k := x_k - \varepsilon$; $i := EV_k$ end;

$F_i := F_i + \varepsilon$

end

```
        else (*E_i F_i ≤ 0*)
        if k > 0 then i := SV_k else i := EV_k
      else (*i = l *)
    if not GoOn then    (*Change potentials*)
    begin   δ := ∞ ;
    for k := 1 to m do
    begin
    p := SV_k ; q := EV_k ;
    if S_p ≠ 0 and S_q = 0 and d_k < 0 and − d_k < ∞
      then begin δ := −d_k ; i := p end
      else
      if S_p = 0 and S_q ≠ 0 and d_k > 0 and d_k < δ
        then begin δ := d_k ; i := q end
    end ;
    if δ = ∞ then STOP (*There exists no feasible circulations*)
    else
        begin
          for j := 1 to n do
          if S_j = 0 then t_j := t_j + δ ;
          for k := 1 to m do
            begin
                i := SV_k ; j := EV_k ;
                d_k := t_j − t_i − c_k
            end
        end
    end (*Change potentials*)
    end
  until F_l = 0
end .
```

Theorem 2. MEFA solves the problem correctly with time complexity $O\left(Kmn^2\right)$.

P r o o f. The algorithm finds at the initialization step a maximum spanning tree $\hat{G}$ with arc weights $b_k - a_k$. It needs $O(mn)$ operations (Prim). For each arc A_k of $\hat{G}$ $d_k = 0$ and $x_k \in [a_k, b_k]$ holds, i.e. (3) and (4) are fulfilled. For all other arcs x_k and d_k are defined in order to fulfill (3) and (4).

The complexity of the Main loop is $O\left(Kmn^2\right)$ [2], which is the total complexity.

The procedure terminates either if every node becomes a transit one or if no potential changes are possible. In the first case condition (2) is fulfilled, hence the problem is solved. In the second case a co-cycle $\mathrm{CC}(V')$ has been found for which the condition for existence of a feasible solution is not fulfilled. It is really the fact since: if $A_k \in \mathrm{CC}^+(V')$ is a forward arc then the flow on it cannot be increased since the point (x_k, d_k) is on one of the vertical branches of the Kilter Curve KC. If no potential changes are possible this point is on the right branch, i.e. $x_k = b_k$. Analogously, $x_k = a_k$ for all backward arcs from the co-cycle. Hence, if $\sum\{F_i \mid V_i \in V'\} < 0$ then

$$\begin{aligned} 0 > & \sum\{x_k \mid A_k \in \mathrm{CC}^+(V')\} - \sum\{x_k \mid A_k \in \mathrm{CC}^-(V')\} \\ & = \sum\{b_k \mid A_k \in \mathrm{CC}^+(V')\} - \sum\{a_k \mid A_k \in \mathrm{CC}^-(V')\}. \end{aligned}$$

It follows from the circulation existing theorem of Hoffmann [4, p.79], that there exists no feasible circulation in the network.

Comment. For an arbitrary pair of source-node V_i and sink-node V_j one wants to find a chain of arcs connecting them in order to send some flow. If it is not possible, we have to change potentials – $O(n)$ operations. But if we start with x and d from the initialization step, a change of the flow is possible from V_i to V_j at least on the only chain from the spanning tree $\hat{G}$. The optimality of $\hat{G}$ allows a larger change since this chain belongs to the spanning tree.

Sometimes it is possible to start with much better non-feasible x, i.e. which allows larger flow changes. It is a solution to the following problem P.

Let $\hat{G} = (\hat{V}, \hat{A})$ be a sub graph of G. We denote by

(5) $$\mathrm{ab}(\hat{G}) := \sum\{b_k - a_k \mid A_k \in \hat{A}\}$$

and

(6) $$\mathrm{AB}(\hat{G}) := \min\{b_k - a_k \mid A_k \in \hat{A}\}.$$

Now we define an optimization problem in lexicographical sense:

(*P*) $$\mathrm{lex\,max}\{(\mathrm{ab}(\hat{G}), \mathrm{AB}(\hat{G})) \mid \hat{G} - \text{a spanning tree of } G\},$$

i.e. find an optimum spanning tree according to (6) among all optimum solutions according to (5). It can be solved with the procedure given below. The idea is the same as that in [5] and [6] for solving the bicriterial optimum path problem.

Procedure **(*P*)**

Let $\hat{G}$ *be a* maximum *spannig tree of* G *with arc weights* $b_k - a_k$*;*
repeat

delete all arcs A_k *of* G *with the smallest arc weights*
and denote the new graph by G';
let $\hat{G}'$ *be a* maximum *spanning tree of* $\hat{G}'$;
if $\text{ab}(\hat{G}) = \text{ab}(\hat{G}')$ *then set* $G := G'$ *and* $\hat{G} := \hat{G}'$
until $\text{ab}(\hat{G}) > \text{ab}(\hat{G}')$;
$\hat{G}$ *is a solution of* (P).

This procedure needs no more than $m-n+1$ iterations, i.e. $O((m-n+1)mn)$ operations at all. Hence, if we modify the initialization step of MEFA in this way, the total complexity of the algorithm will be $O((m-n+1)mn) + O(Kmn^2)$. Note that $m-n+1>0$ since G is connected and contains cycles. We cannot say which term is larger since everything depends on K as well as the relation between m and n. If the graph is a dense one, the first term dominates and the complexity is polynomial one. If, on the other hand, it is sparse K plays a crucial role in the amplitude of the latter term. In this case the complexity of MEFA will be pseudo polynomial and the same as EFA – $O\left(Kmn^2\right)$. Finally, if we start MEFA with a bicriterial maximum spanning tree, mentioned above, the total complexity will be $O((m-n+1)mn) + O(Kmn^2)$.

3. Computational experiment [3]

Both algorithms OKA and MEFA and the old version EFA are pseudo polynomial ones, since the complexity $O(Km^2n)$, $O((m-n+1)mn) + O(Kmn^2)$ and $O(Kmn^2)$, respectively, depend on the input data. We again point out that we use OKA only as a measure unit since it is well-known and popular.

For "small" networks we have arranged a computational experiment and have compared the executing time of both algorithms OKA and EFA. For the purpose we have developed a computer code in Turbo Pascal 6.0 and have used a PC AT with microprocessor I80286 and coprocessor 180287. The regression model for the ratio

$$R = \text{time(OKA)/time(EFA)}$$

of type $Y = aX^b$ has been obtained with the program package STATGRAPHICS. We made three kinds of experiments.

We fix $n = 100$ (number of nodes) and let m (number of arcs) vary from 1000 to 10000 in 1000 interval. For each case we generate randomly five networks and solve the problem with both algorithms. The ratio of both times is $R(m) = 0.2772(m)^{0.4425}$. For 10000 arcs EFA is about 7 times faster than OKA – Fig. 1.

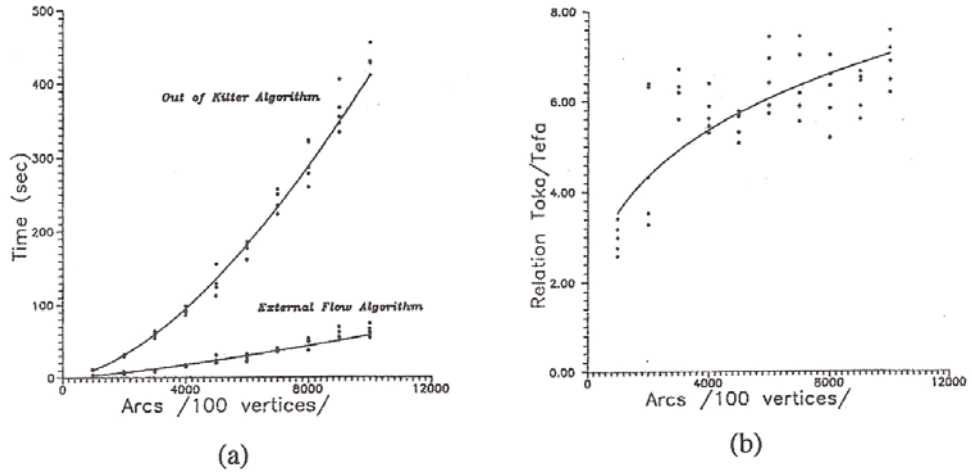

Fig. 1

Now we fix $m = 5000$ and vary n from 10 to 100 at 10 interval. Again five networks for each case have been generated. The regression function is $R(n) = 73.472(n)^{-0.6868}$. EFA is about 10 times faster for 100 nodes – Fig. 2.

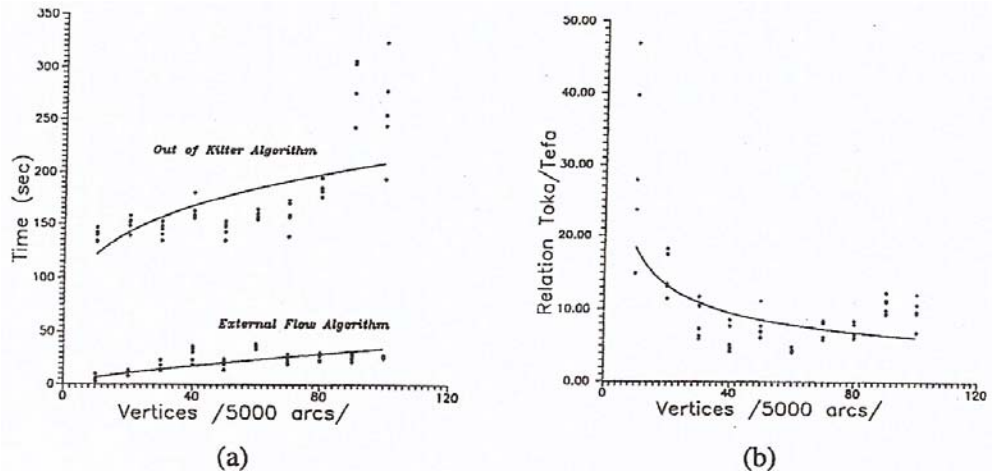

Fig. 2

Finally we fix $n = 50$ and $m = 500$ and vary the difference $b_k - a_k$ from 10 to 100 at 10 interval. Again a series of five networks have been generated. The result is $R(b-a) = 0.6504(b-a)^{0.651}$. For $b - a = 100$ EFA is more than 15 times faster (Fig. 3).

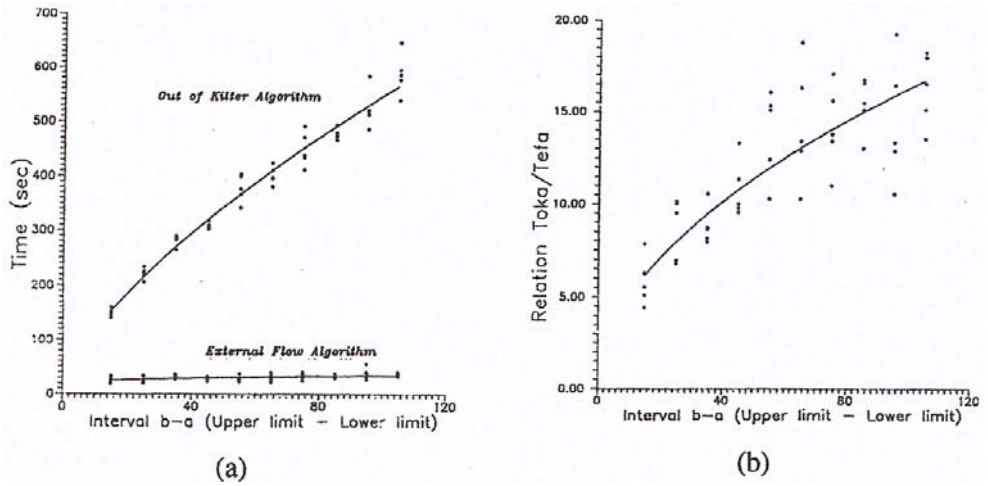

Fig. 3

Further we compare the number of iterations for both algorithms. At each iteration of EFA, the actual source node becomes a transit one. For OKA the actual arc, which is in an out-of-kilter state, gets in an in-kilter-state.

The results are given in Table 1, where ANI(EFA) and ANI(OKA) means "Average Numbers of Iterations" of the algorithm for the series of five randomly generated networks.

Table 1

n	*m*	ANI(EFA)	ANI(OKA)
20	30	6.8	12.6
20	55	8.4	19.4
20	80	6.4	23.8
20	105	5.8	32.0
20	130	6.4	32.2
20	155	6.0	36.8
20	180	6.2	40.8
20	205	5.6	43.6
20	230	5.8	41.0
40	1000	7.2	122.4
55	1000	13.0	127.0
70	1000	16.6	132.6
85	1000	23.0	116.8
100	1000	23.2	122.8
115	1000	33.8	115.2
130	1000	36.8	109.0
145	1000	40.2	121.8
160	1000	43.6	119.8
10	90	3.2	21.8
15	210	5.4	39.8
20	380	5.0	61.4
25	600	4.4	71.6
30	870	4.4	91.2
35	1190	5.4	107.4
40	1560	5.2	144.4
45	1980	5.4	147.2
50	2450	4.2	185.6

4. Conclusions

The difference between MEFA and EFA is in the initialization step which does not depend on the input data a, b and c.

Since MEFA starts with a better, non-feasible solution as EFA, it should be faster. In this case the ratio time(OKA)/time(MEFA) should be larger than time(OKA)/time(EFA) .

MEFA is much faster than OKA for dense graphs. It is not slower for sparse graphs.

If $b-a$ and c increase then the ratio of both times decreases but it is greater than one, i.e. MEFA is faster.

The memory we need for both computer codes is nearly the same.

The constant K increases for both algorithms but the speed for MEFA is much faster.

MEFA needs definitely fewer iterations.

The PASCAL-like description of MEFA allows to develop a computer code very easily. We recommend a Neighbour-Vertex-List networks presentation in order to speed up the procedure.

Acknowledgments. The author is grateful to Professor Vangelis Paschos from the University of Paris-Dauphine and the anonymous referees for their remarks and suggestions which improved the quality of this paper.

References

1. Ahuja, R. K., T. L. Magnanti, J. B. Orlin. Network Flows. Englewood Cliffs, Prentice-Hall, 1993. 943 p.
2. Ivanchev, D. An Algorithm for Solving the Minimum Cost Flow Problem with Complexity O(Kmn^2). – Compt. Rend. Acad. Bulg. Sci., **44 (7)** , 1991, 13-16.
3. Ivanchev, D. A Computational Comparison of Two Algorithms for Solving the Optimum Circulation Problem. Applications of Mathematics in Engineering and Economics. Sofia, Heron Press, 2002, 359-367.
4. Ford, L. R., D. R. Fulkerson. Flows in Networks. New Jersey, Princeton University Press, 1961. 276 p.
5. Ivanchev, D. Network Optimization. Sofia, Heron Press MathBooks, 2001. 204 p.
6. Ivanchev, D., D. Kydros. Multicriteria Optimum Path Problems. YUJOR, 5, 1995, No 1, 79-93.
7. Karagiannis, P., K. Paparrizos, N. Samaras, A. Sifarelas. A New Simplex Type Algorithm for the Minimum Cost Network Flow Problem. – In: Proc. of the BALCOR 2005, Konstanta, Romania, 2007, 133-139.
8. Geranis. G., K. Paparrizos, A. Sifarelas. A Dual Exterior Point Simplex Type Algorithm for the Minimum Cost Network Flow Problem. Abstracts. – In: Proc. of the BALCOR 2007, Belgrade–Zlatibor, Serbia, 2007. 53 p.