# An exterior Simplex type algorithm for the minimum cost network flow problem

Konstantinos Paparrizos, Nikolaos Samaras [*]
Angelo Sifaleras

*Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece, 156 Egnatia Str., Postal Code 54006*

**Abstract**

In this paper a new Network Exterior Point Simplex Algorithm (NEPSA) for the Minimum Cost Network Flow Problem (MCNFP) is analytically presented. NEPSA belongs to a special simplex type category and is a modification of the classical network simplex algorithm. The main idea of the algorithm is to compute two flows. One flow is basic but not always feasible and the other is feasible but not always basic. A complete proof of correctness for the proposed algorithm is also presented. Moreover, the computational behavior of NEPSA is shown by an empirical study carried out for randomly generated sparse instances created by the well known gridgen network problem generator.

*Key words:* Combinatorial Optimization, Minimum Cost Network Flow Problem, Simplex Algorithm, Exterior Point Algorithm.
*1991 MSC:* 90C27, 90C49, 90C35, 65K05, 91A90

## 1 Introduction

Network Optimization is a large part of Combinatorial Optimization. The Minimum Cost Network Flow Problem (MCNFP) constitutes a wide category of problems. A large number of real-world applications in communications, informatics and transportation can be modelled as network flow problems. Many of them can be found in [2] and [11]. Furthermore, other well known problems

[*] Corresponding author. Address: 156 Egnatia Str., 54006 Thessaloniki, Greece, Tel: +302310 891866, Fax: +302310 891879
*Email addresses:* `paparriz, samaras@uom.gr` (Konstantinos Paparrizos, Nikolaos Samaras), `sifalera@uom.gr` (Angelo Sifaleras).

like the shortest path problem and the assignment problem are special cases of MCNFP

Computational algorithms for the solution of network flow problems are of great practical significance. The first polynomial algorithm for MCNFP was developed by Edmonds and Karp [9]. They proved that the Out-of-Kilter method can be transformed into a polynomially bounded method, provided a special scaling technique is used. Tardos [26], proposed the first strongly polynomial algorithm for MCNFP. The existence of a strongly polynomial algorithm distinguishes MCNFP from the general linear programming problem. Since then, the operations research community have developed a variety of algorithms and data structures for the solution of MCNFP. There exist a large number of different polynomial time algorithms for MCNFP. The most efficient algorithms for MCNFP were developed by [1], [19] and [14]. However, the classical Network Primal simplex Algorithm (NPSA) remains the best choice for solving MCNFP. The first polynomial time NPSA for MCNFP was proposed by Orlin [20]. The number of pivots needed for this algorithm is $O(\min\{nm\text{logn}C, nm^2\text{logn}\})$, where $C$ denotes the maximum absolute arc cost if arc costs are integer and $\infty$ otherwise. Recently, Interior Point Methods have been proposed and applied to solve large-scale network flow problems. Survey on Interior Point Methods for network flow problems can be found in [24].

The algorithm presented in this paper belongs to a special "exterior simplex type" category and it is of the same type as the one described in [21]. The first exterior point algorithm was developed by Paparrizos [22], for the assignment problem. Computational results on randomly generated sparse and dense linear problems have shown that primal exterior point algorithm is up to ten times faster than classical simplex algorithm. This computational improvement is due to the essential reduction on the number of iterations. It is believed that these promising results could be also applied to MCNFP. Since then, exterior point approaches have been also used by other researchers, as in [25], for linear programming problems. NEPSA constructs two flows to the optimal solution. One flow is basic but not always feasible; so this is an "exterior flow". The other flow is feasible but not always basic. A common feature of almost all simplex type algorithms is that they can be interpreted as a method following simplex type paths that lead to the optimal solution. The main difference between NEPSA and NPSA is that the basic flow of the former does not always remain feasible. To this date, there are no specializations of exterior type simplex algorithms for MCNFP. Implementations of exterior type simplex algorithms exist only for the assignement problem [22]. In this paper we present the first exterior type network simplex algorithm for MCNFP.

The paper is organised as follows. Following this introduction, in Section 2 we give some necessary notations and definitions. An analytic description of the proposed algorithm is presented in Section 3. In the same Section we discuss the degeneracy problem and a method which was used in our implementation to avoid the *stalling* phenomenon. In Section 4 we show the proof of correctness. Specifically, we prove that NEPSA is finite on non degenerate problems. Finally, in Section 6 we conclude and possible future work is discussed.

## 2 Notations and definitions

Let $G = (N, A)$ be a directed network with $n$ nodes and $m$ arcs, where $N$ and $A$ are the sets of nodes and arcs respectively. Each arc $(i, j) \in A$ has a cost $c_{ij}$ which denotes the unit shipping cost along arc $(i, j)$. In this paper, NEPSA algorithm is applied to un-capacitated MCNFPs. This means that $l_{ij} \leq x_{ij} \leq u_{ij}$, where $l_{ij} = 0$ and $u_{ij} = +\infty$. Associated with each arc $(i, j)$ is also an amount $x_{ij}$ of flow on the arc (contrary to NPSA, $x_{ij}$ might become negative). We associate with each node $i \in N$ a number $b_i$ which indicates its available amount of supply or demand. Node $i$ will be called a source, sink or transshipment node, depending upon whether $b(i) > 0,\ b(i) < 0,\ b(i) = 0$ respectively. Source nodes produce more flow than they consume. Sink nodes consume more flow than they produce and transhipment nodes produce as much flow as they consume. We make the assumption that $G$ is a balanced network, that is $\sum\limits_{i=1}^{n} b(i) = 0$. Now, the minimum cost flow problem can be formulated as the following linear program:

$$\min \sum_{\mathrm{(i,j)\in A}} \mathrm{c_{ij}} x_{ij}$$

$$s.t. \sum_{\mathrm{i:(k,i)\in A}} x_{ki} - \sum_{\mathrm{j:(j,k)\in A}} x_{jk} = b_k, k \in N \tag{1}$$

$$0 \leq x_{ij} \leq +\infty, \forall (i, j) \in A$$

In the above formulation, constraints of type $\sum x_{ki} - \sum x_{jk} = b_k$ are known as the flow conservation equations, while constraints of type $0 \leq x_{ij} \leq +\infty$ are called the flow capacity constraints. In matrix notation MCNFP can be formulated as a linear program of the form $\min\Big\{c^T x : \ Ax = b,\ l \leq x \leq u\Big\}$ where $A \in \Re^{n\times m},\ c,\ x,\ l,\ u\ \in \Re^{m},\ b\ \in \Re^{n}$.

A vector of dual variables for the network $G$ is a vector $w$ with $n$ components, where $w_i$ is associated with node $i$. For each vector $w$, the reduced costs $s_{ij}$ are defined as $s_{ij} = c_{ij} - w_i + w_j$. Network Simplex type algorithms compute basic solutions $x$ and $(w, s)$ which are complementary.

Furthermore, in this paper, notations such as paths, directed paths, cycles, directed cycles and trees, are defined as in [4]. The basic solution for the MCNFP can be represented using a tree. Each basis tree of a MCNFP consists of a flow vector $x$, which is the unique solution of problem 1. Henceforth, the basis tree of a MCNFP will be denoted by $T$, while the basic flow of each $T$ will be denoted by $x(T)$. We say that $T$ is a feasible tree, iff $x(T) \geq 0$. Otherwise, it will be called an infeasible tree.

Contrary to NPSA, NEPSA first selects a leaving arc and afterward selects an entering arc. For simplicity reasons, from now on the leaving arc will be denoted by $(k, l)$ while the entering arc by $(g, h)$. We also denote by $C^{(t)}$ the cycle that would have been created at the $t^{th}$ iteration, if prior to the leaving arc, the entering arc would have joined the basic tree $T$. If an arc $(k, l)$ leaves the basis tree, then the current $T$ is divided into two sub-trees. The sub-tree which includes the $k$ node will be denoted by $T^+$, while the other by $T^-$. All the nonbasic arcs $(i, j)$ will either not join these two sub-trees, or they will join them with two ways. Either the node $i \in T^+$ and the node $j \in T^-$, or vice versa. The previous 3 cases, are illustrated in Figure 1. Moreover, the value of the $\psi$ variable at the $t^{th}$ iteration will be denoted by $\psi^{(t)}$, as also the inverse of a vector $c$, will be denoted by $c^T$.

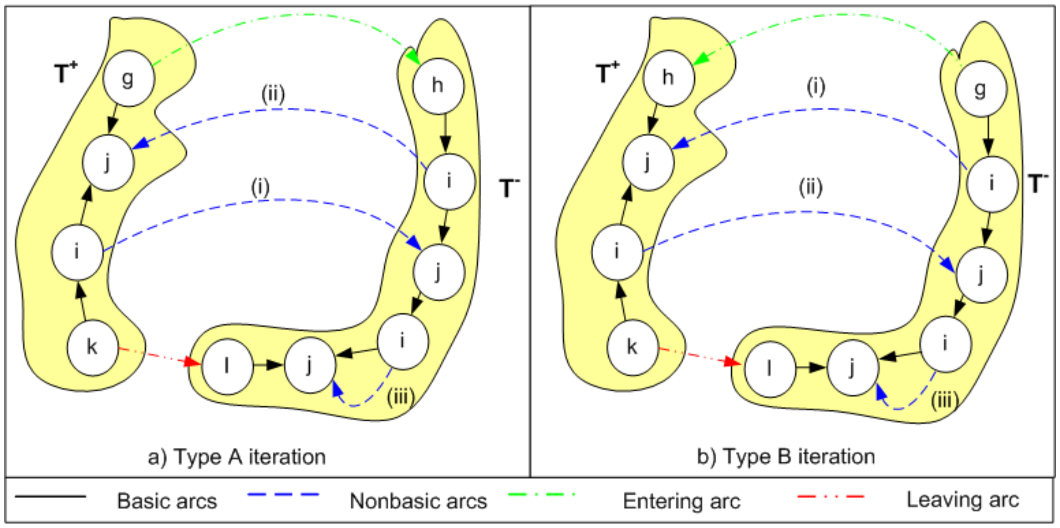


Fig. 1. Possible combinations for any nonbasic arc $(i, j)$

At the initialization of NEPSA, a starting feasible tree can be constructed using the well known big $M$ method. For a detailed review of the Big $M$ method see [4]. To continue with, the arc set $A$ is partitioned into three subsets. The first subset contains the arcs belonging in $T$, while the other two sets $P$ and $Q$ contain the remaining arcs:

$$P = \{(i, j) : s_{ij} < 0\}, Q = \{(i, j) : s_{ij} \geq 0\}, \forall \ (i, j) \ \notin T \tag{2}$$

In this way, the initial arc set $A$ is partitioned as $A = T \cup P \cup Q$. Also vector

$h_{gh}$ will denote the representation of each non basic arc, for example $(g, h)$, in terms of the basic arcs. Each non basic arc $(g, h)$ corresponds to a column vector $h_{gh}$. Each element of this column corresponds to a basic arc, for example $(k, l)$. The elements of $h_{gh}$ are calculated using the following rules.

$$\begin{aligned} &h_{gh}(k,l) = -1, \text{if}(k,l) \text{ has the same orientation with } (g,h) \text{ in } C^{(t)} \\ &h_{gh}(k,l) = 1, \text{if}(k,l) \text{ has the opposite orientation with } (g,h) \text{ in } C^{(t)} \\ &h_{gh}(k,l) = 0, \text{ if } (k,l) \notin C^{(t)} \end{aligned} \tag{3}$$

Detailed description of these calculations can be found in [4]. Furthermore, vector $d$ will denote a "complemental" flow which assists the algorithm to compute the second feasible but not always basic flow. According to the partition of $A$, vector $d$ is partitioned as $d = \{d(T), d(P), d(Q)\}$. The elements of $d$ are computed using the following relations, where $|P|$ denotes the cardinality of set $P$.

$$\begin{aligned} &d(T) = -\sum_{(i,j)\in P} h_{ij}, \\ &d(P) = (\lambda_1, \lambda_2, ..., \lambda_{|P|}), \\ &d(Q) = (0, 0, ..., 0). \end{aligned} \tag{4}$$

## 3 Description of NEPSA algorithm

### *3.1 Analytic Description*

At *Step 1* the optimality condition is examined. If $P = \emptyset$, then an optimal solution has been found and the algorithm terminates. Otherwise, the algorithm continues. If $\exists (i,j) \in T : d_{ij} < 0$, then the algorithm proceeds to the next step. Otherwise the problem 1 is unbounded and therefore negative cost cycles have been identified. At *Step* 2, the leaving arc is chosen using the following minimum ratio test:

$$a = \frac{x_{kl}}{-d_{kl}} = \min\left\{\frac{x_{ij}}{-d_{ij}} : (i,j) \in T, d_{ij} < 0\right\} \tag{5}$$

Afterward, the second flow $y$ is calculated as follows:

$$y = x + ad \tag{6}$$

This second flow remains feasible, but not always basic, since $y$ corresponds to a network and not to a tree. Elements of vector $\lambda = (\lambda_1, \lambda_2, ..., \lambda_{|P|})$ can have any value, so long as vector $d$ has appropriate values in order to compute a feasible flow $y$. In our implementation, vector $\lambda$ is initialized using the values $\lambda = (1, 1, ..., 1)$.

At *Step* 3, the entering arc is chosen using the following minimum ratio tests:

$$\theta_1 = -s_{p_1p_2} = \min\{-s_{ij} : h_{ij}(k,l) = 1 \textit{ and } (i,j) \in P\} \tag{7}$$

$$\theta_2 = s_{q_1q_2} \min\{s_{ij} : h_{ij}(k,l) = -1 \textit{ and } (i,j) \in Q\} \tag{8}$$

If $\theta_1 \leq \theta_2$, then the pivot will be called "*Type A iteration*". In this case the entering arc will be $(g,h) = (p_1,p_2)$ and $P = P \setminus (p_1,p_2)$. Otherwise, the pivot will be called "*Type B iteration*". In the latter case the entering arc will be $(g,h) = (q_1,q_2)$ and $Q = Q \setminus (q_1,q_2)$. Using the new partition $(T \;\; P \;\; Q)$, where $T = T \setminus (k,l) \cup (g,h)$ and $Q = Q \cup (k,l)$ the vectors $x_{ij}$, $s_{ij}$, $h_{ij}$ and $d_{ij}$ are updated.

There are two choices for the implementation of the $d$ vector. The first choice is to compute the feasible flow $y$ at each iteration and to update $d$ using Relation 9.

$$d^{(t+1)} = y^{(t)} - x^{(t+1)} \tag{9}$$

If $(g,h) \in P$, $d_{gh}^{(t+1)} = d_{gh}^{(t+1)} + \lambda_{gh}$. The second method doesn't calculate the feasible flow $y$ at each iteration. Instead to the first method of update, only the $d(T)$ subset is computed. At each iteration, the $d(T)$ is updated using the same method as the columns $h_{ij}$ (from the network simplex tableau). However, this difference in the values in $d_{ij}(T)$ between the two methods doesn't affect the minimum tests. Therefore, it is much more preferable to implement the NEPSA algorithm in PC using the second method, since it demands less data to be kept and accessed at each iteration. This method also will be used in the proofs of correctness of Section 4.

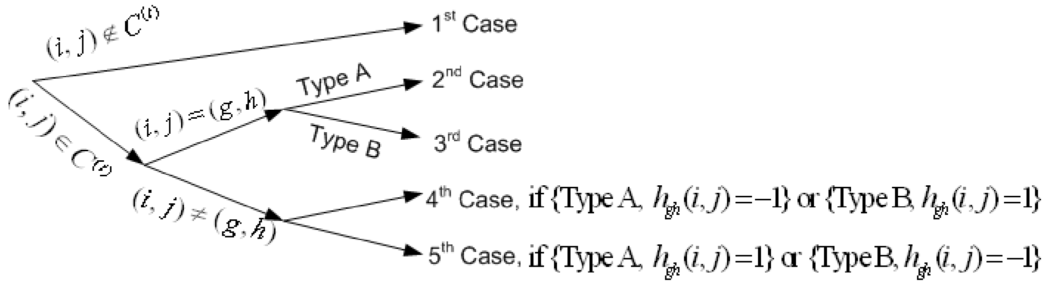
NEPSA is well defined under the assumption that all the problems, which NEPSA is applied to, are not degenerate. Consequently, $x_{kl} \neq 0$ and $s_{gh} \neq 0$. However, in the implementation of NEPSA a method to avoid the bad results due to degeneracy; stalling or cycling, was applied. More precisely, there is a possibility that even after the selection of the leaving arc using Relation 5 or the entering arc using Relations 7 and 8, NEPSA might have to choose between two or more, equally qualified, arcs. This phenomenon is usually named as *tie*. NEPSA was programmed in such a way that it breaks the *ties* using the following method. A numbering was given at each arc and always we selected between equally qualified, leaving or entering arcs, the one with the minimum index. This method is similar to the rule of Bland [5] for the general linear problem. Based on preliminary computational results on random generated sparse problems [23] using this method, there wasn't any basic tree recurrence in succesive iterations. Moreover, we neither observed any long sequence of arcs to leave from the basic tree and enter back again repeatedly. However, it is for sure that the experiments must continue in order to find out if NEPSA suffers from the cycling or stalling phenomenon.

### *3.2 Pseudo Code*

A formal description of NEPSA is given below.

### *3.3 Variables Updates*

For any basic arc $(i,j)$, the corresponding variables $x_{ij}, d_{ij}, s_{ij}$ and the $h_{ij}$ columns, are updated depending on the combination of two parameters. The first parameter is the type of iteration while the second is the orientation of arc $(i,j)$ in relation to the orientation of the leaving arc $(k,l)$. Some of these combinations can be grouped together as in Figure 2, because the corresponding variables of arcs, which belong in the same case, are updated similarily. Therefore, the following cases - combinations will be used to explain the updates of all the arc variables and also to the proofs of Section 4.



$(i,j) \notin C^{(t)}$
$(i,j) \in C^{(t)}$
$(i,j) = (g,h)$
$(i,j) \neq (g,h)$
Type A
Type B
1st Case
2nd Case
3rd Case
4th Case, if {Type A, $h_{gh}(i,j) = -1$} or {Type B, $h_{gh}(i,j) = 1$}
5th Case, if {Type A, $h_{gh}(i,j) = 1$} or {Type B, $h_{gh}(i,j) = -1$}

Fig. 2. Possible combinations for any basic arc $(i,j)$

In Table 1, one can see all the variables updates corresponding to basic arcs, using the classification of Figure 2. For the $h_{ij}$ columns, assume that $(n_1,n_2) \notin T$ and $(i,j) \in T$. For example, let $(i,j) \in T$ be an arc which has basic flow equal to $x_{ij}^{(t)}$. Assume at the $t^{th}$ iteration, which is of type $A$, the $(i,j)$ arc has the same orientation with the entering arc $(g,h)$ in $C^{(t)}$. According to Figure 2 this is the $4^{th}$ case. In this case, the updated basic flow of the $(i,j)$ arc at

**Algorithm 1** NEPSA Algorithm

```
Require: $G = (N, A), b, c, T$
 1: procedure NEPSA($G, T$)
    Step 0 (Initializations)
 2:     Compute $x$, $w$ and $s$
 3:     Partition of $A$ into subsets $P$ and $Q$
 4:     Compute vectors $h_{ij}$, and $d$, using Relation 3 and 4
    Step 1 (Test of optimality)
 5:     while $P \neq \emptyset$ do
 6:         if $d(T) \geq 0$ then
 7:             STOP. The problem 1 is unbounded.
 8:         else
    Step 2 (Choice of leaving arc)
 9:             Compute $a$, using Relation 5.
10:             Choose the leaving arc $(k, l)$ and compute $y$ using Relation 6
    Step 3 (Choice of entering arc)
11:             Compute $\theta_1$, $\theta_2$ using Relations 7 and 8.
12:             Choose the entering arc $(g, h)$
    Step 4 (Pivoting)
13:             if $\theta_1 \leq \theta_2$ then
14:                 Set the entering arc $(g, h) = (p_1, p_2)$
15:                 Set $P = P \setminus (g, h)$.
16:             else
17:                 Set the entering arc $(g, h) = (q_1, q_2)$
18:                 Set $Q = Q \setminus (g, h)$.
19:             end if
20:             $Q = Q \cup (k, l)$
21:             Set $T = T \setminus (k, l) \cup (g, h)$
22:             Update $x$, $s$, $h_{ij}$ and $d$
23:         end if
24:     end while
25:     STOP. The problem 1 is optimal.
26: end procedure
```

the $(t+1)^{st}$ iteration will derive according to the $4^{th}$ row of Table 1. Therefore we take $x_{ij}^{(t)} + x_{kl}^{(t)}$.

Finally, the reduced costs are updated, using different classification, as follows:

$$s_{ij}^{(t+1)} = \begin{cases} s_{ij}^{(t)} - s_{gh}^{(t)}, & (i, g \in T^+ \wedge j, h \in T^-) \vee (i, g \in T^- \wedge j, h \in T^+) \\ s_{ij}^{(t)} + s_{gh}^{(t)}, & (i, h \in T^+ \wedge j, g \in T^-) \vee (i, h \in T^- \wedge j, g \in T^+) \\ s_{ij}^{(t)}, & (i, j \in T^+) \vee (i, j \in T^-) \end{cases}$$

| | $x_{ij}^{(t+1)} =$ | $d_{ij}^{(t+1)} =$ | $h_{n_1 n_2}^{(t+1)}(i,j) =$ |
|---|---|---|---|
| 1st Case | $x_{ij}^{(t)}$ | $d_{ij}^{(t)}$ | $h_{n_1 n_2}^{(t)}(i,j)$ |
| 2nd Case | $x_{kl}^{(t)}$ | $d_{kl}^{(t)} + \lambda_{kl}$ | $h_{n_1 n_2}^{(t)}(k,l) + \lambda_{kl}$ |
| 3rd Case | $-x_{kl}^{(t)}$ | $-d_{kl}^{(t)}$ | $-h_{n_1 n_2}^{(t)}(k,l)$ |
| 4th Case | $x_{ij}^{(t)} + x_{kl}^{(t)}$ | $d_{ij}^{(t)} + d_{kl}^{(t)}$ | $h_{n_1 n_2}^{(t)}(i,j) + h_{n_1 n_2}^{(t)}(k,l)$ |
| 5th Case | $x_{ij}^{(t)} - x_{kl}^{(t)}$ | $d_{ij}^{(t)} - d_{kl}^{(t)}$ | $h_{n_1 n_2}^{(t)}(i,j) - h_{n_1 n_2}^{(t)}(k,l)$ |

Table 1

Basic arcs updates

## 4 Proofs of correctness

Proof of correctness of NEPSA, relies in the proofs of the following theorems.

**Theorem 1** *If the problem 1 is not degenerate, then the value of the objective function strictly decreases, from iteration to iteration.*

**Proof** Let $z^{(t)}$ be the value of the objective function at the $t^{th}$ iteration. If the iteration is of type A, then the difference $\Delta z = z^{(t+1)} - z^{(t)}$ is due to the flow of $x_{kl}^{(t)}$ units through $(g,h)$ in $C^{(t)}$. It only needs to prove that, this change to the flow corresponds also to a decrement of the total cost. If one unit $(x_{kl} = 1)$ flows through the cycle, then $\Delta z = \sum\limits_{(i,j)\in C} t_{ij}c_{ij}$, where $t_{ij} = 1$, if $h_{gh}(i,j) = -1$ and $t_{ij} = -1$, if $h_{gh}(i,j) = 1$. It is well known, that $\sum\limits_{(i,j)\in C} t_{ij}c_{ij} = s_{gh} \Rightarrow \Delta z < 0$, (since $s_{gh}^{(t)} < 0$ in type A iteration).

If the iteration is of type B, then $\Delta z$ is due to the flow of $-x_{kl}^{(t)}$ units through the entering arc in the $C^{(t)}$. Again, it only needs to prove that, this change to the flow corresponds also to a decrement of the total cost. If one negative unit flows through the cycle, then $\Delta z = \sum\limits_{(i,j)\in C} t'_{ij}c_{ij}$, where now $t'_{ij} = -1$, if $h_{gh}(i,j) = -1$ and $t'_{ij} = 1$, if $h_{gh}(i,j) = 1$. Accordingly, $\sum_{(i,j)\in C} t'_{ij}c_{ij} = -\sum_{(i,j)\in C} t_{ij}c_{ij} \Rightarrow \Delta z = -s_{gh} \Rightarrow \Delta z < 0$, (since $s^{(t)}_{gh} > 0$ in type B iteration). □

**Theorem 2** *If the problem is not degenerate, then the algorithm will perform a finite number of iterations.*

**Proof** From Theorem 1 we conclude that the value of the objective function stirctly decreases, from iteration to iteration. Therefore no tree will be created twice. It is well known that the number of trees of any network is finite. Hence, NEPSA will perform a finite number of iterations. □

**Theorem 3** *At each type A iteration, it holds that if $(i,j) \in P$ then $s_{ij} < 0$, and if $(i,j) \in Q$ then $s_{ij} \geq 0$*

**Proof** At the 1st iteration we defined that $s^{(1)}_{ij} < 0$ for $(i,j) \in P$. Assume that at the $t^{th}$ iteration $s^{(t)}_{ij} < 0$ for $(i,j) \in P$. We will show that it also holds at the *t+1* iteration. Since the $t^{th}$ iteration is of type $A$, then $s^{(t)}_{gh} < 0$. We examine the following three possible cases for arc $(i,j)$; the same classification as the reduced costs update in subsection 3.3:

- 1st *Case*: If $(i,j) \in P$ connect $T^+$ and $T^-$ with the same way as $(g,h)$, then it was selectable as entering. In this case we have $-s^{(t)}_{ij} > \theta^{(t)}_1$ or $-s^{(t)}_{ij} > -s^{(t)}_{gh} \Rightarrow s^{(t)}_{ij} - s^{(t)}_{gh} < 0 \Rightarrow s^{(t+1)}_{ij} < 0$, (see $a(i)$ in Figure 1).
  It must be mentioned that in case where $-s^{(t)}_{ij} = \theta^{(t)}_1$, since we had a type A iteration (therefore $\theta^{(t)}_1 \leq \theta^{(t)}_2$), then obviously $(i,j)$ was selected as entering arc, so $(i,j) \notin P$ in the *(t+1)* iteration.
- 2nd *Case*: If $(i,j) \in P$ connect $T^+$ and $T^-$ with the opposite way than $(g,h)$, then it was not selectable as entering. In this case we have $s^{(t+1)}_{ij} = s^{(t)}_{ij} + s^{(t)}_{gh} \Rightarrow s^{(t+1)}_{ij} < 0$, (see $a(ii)$ in Figure 1).
- 3rd *Case*: If $(i,j) \in P$ doesn't connect $T^+$ with $T^-$, then $s^{(t+1)}_{ij} = s^{(t)}_{ij} \Rightarrow s^{(t+1)}_{ij} < 0$, (see $a(iii)$ in Figure 1).

Therefore at each type $A$ iteration, it holds that if $(i,j) \in P$ then $s_{ij} < 0$. The proof for the case $(i,j) \in Q$ is analogous. □

**Theorem 4** *At each type B iteration, it holds that if $(i,j) \in P$ then $s_{ij} < 0$, and if $(i,j) \in Q$ then $s_{ij} \geq 0$*

**Proof** At the 1st iteration we defined that $s_{ij}^{(1)} < 0$ for $(i,j) \in P$. Assume that at the $t^{th}$ iteration it holds that $s_{ij}^{(t)} < 0$ for $(i,j) \in P$. We will show that it is also true at the *t+1* iteration. Since the $t$ iteration is of type $B$, it holds $s_{gh}^{(t)} > 0$. We examine the following three possible cases for arc $(i,j)$:

- 1st *Case*: If $(i,j) \in P$ connect $T^+$ and $T^-$ with the same way as $(g,h)$, then it was not selectable as entering. In this case we have $s_{ij}^{(t+1)} = s_{ij}^{(t)} - s_{gh}^{(t)} \Rightarrow s_{ij}^{(t+1)} < 0$, (see $b(i)$ in Figure 1).
- 2nd *Case*: If $(i,j) \in P$ connect $T^+$ and $T^-$ with the opposite way than $(g,h)$, then it was selectable as entering. In this case we have $-s_{ij}^{(t)} > \theta_1^{(t)} \Rightarrow -s_{ij}^{(t)} > \theta_2^{(t)}$, or $-s_{ij}^{(t)} > s_{gh}^{(t)} \Rightarrow s_{ij}^{(t)} + s_{gh}^{(t)} > 0 \Rightarrow s_{ij}^{(t+1)} < 0$, (see $b(ii)$ in Figure 1).
- 3rd *Case*: If $(i,j) \in P$ doesn't connect $T^+$ with $T^-$, then $s_{ij}^{(t+1)} = s_{ij}^{(t)} \Rightarrow s_{ij}^{(t+1)} < 0$, (see $b(iii)$ in Figure 1).

Therefore at each type $B$ iteration, it holds that if $(i,j) \in P$ then $s_{ij} < 0$. The proof for the case $(i,j) \in Q$ is analogous. □

It is crucial for exterior point Simplex type algorithms, always to keep contact with the feasible region. In terms of Linear Programming it could be said that the following Theorem 5 make sure that NEPSA will find a path which crosses inside the feasible region of the problem. Therefore, Theorem 5 is of great importance.

**Theorem 5** *If* $x_{ij} < 0$ *then* $\exists d_{ij} > 0$ *such that* $\frac{x_{ij}}{-d_{ij}} < \alpha$

**Proof** The induction method will be used in this proof. At the 1st iteration, $\nexists (i,j) : x_{ij}^{(1)} < 0$, thus the statement is true.

We will show that the first time that an arc $(i,j)$ will have $x_{ij} < 0$, then it will also be $d_{ij} > 0$. An arc might acquire negative flow, only in the 3rd and 5th case of Table 1. In the 3rd case, it holds $x_{ij}^{(t+1)} = -x_{kl}^{(t)} < 0$. However, it is also $d_{ij}^{(t+1)} = -d_{kl}^{(t)} > 0$.

In the 5th case if $x_{ij}^{(t+1)} < 0$, then we get

$$x_{ij}^{(t)} < x_{kl}^{(t)} \tag{10}$$

We only need to prove that $d_{ij}^{(t+1)}$ or according to Table 1

$$d_{ij}^{(t)} > d_{kl}^{(t)} \tag{11}$$

If we assume that this Relation is not true, or $d_{ij}^{(t)} < d_{kl}^{(t)}$ we get that $d_{ij}^{(t)} < 0$. Therefore, Relation 11 becomes

$$-d_{kl}^{(t)} < -d_{ij}^{(t)} \tag{12}$$

On multiplying Equations (positive left and right parts) 10 and 12, we come to contradiction. Therefore, in any case, the first time that an arc $(i, j)$ will have $x_{ij} < 0$, it will also have $d_{ij} > 0$.

Assume that for arc $(i, j)$, at the $t^{th}$ iteration, if $x_{ij}^{(t)} < 0$ then $\exists d_{ij}^{(t)} > 0$ such that $\frac{x_{ij}^{(t)}}{-d_{ij}^{(t)}} < \alpha^{(t)}$. We will show that if at the *t+1* iteration, arc $(i, j)$ maintains $x_{ij}^{(t+1)} < 0$ then $\exists d_{ij}^{(t+1)} > 0$ such that $\frac{x_{ij}^{t+1}}{-d_{ij}^{t+1}} < \alpha^{(t+1)}$.

We show first that for each arc $(i, j)$ such that $x_{ij}^{(t+1)} < 0$, $\exists d_{ij}^{(t+1)} > 0$, and afterward that it is such that $\frac{x_{ij}^{(t+1)}}{-d_{ij}^{(t+1)}} \leq \alpha^{(t+1)}$. In order to examine $d_{ij}^{(t+1)}$, all the possible variable updates are examined, corresponding to any type of iteration. The classification of the updates of the basic arcs as in Figure 2, will be used in this Theorem.

*Examination of the cases of $(i, j)$ from Table 1.*

- $1^{st}$ *Case*: $d_{ij}^{(t+1)} = d_{ij}^{(t)} > 0$.
- $2^{nd}$ *and* $3^{rd}$ *Case*: In these cases since $(i, j) \notin T^{(t)}$ (it is the next entering arc), it holds $x_{ij}^{(t)} = 0$. However, we assumed that $x_{ij}^{(t)} < 0$. Therefore, these cases come into contradiction with the assumption and will not be examined.
- $4^{th}$ *Case*: $x_{ij}^{(t+1)} < 0$ then $x_{ij}^{(t)} + x_{kl}^{(t)} < 0$, so we take that

  $$x_{kl}^{(t)} < -x_{ij}^{(t)} \tag{13}$$

  By the induction hypothesis,

  $$\frac{x_{ij}^{(t)}}{-d_{ij}^{(t)}} < \frac{x_{kl}^{(t)}}{-d_{kl}^{(t)}} \overset{\cdot(-d_{ij}^{(t)}d_{kl}^{(t)})>0}{\rightarrow} \underbrace{d_{kl}^{(t)}x_{ij}^{(t)}}_{>0} < \underbrace{x_{kl}^{(t)}d_{ij}^{(t)}}_{>0} \tag{14}$$

  On multiplying Equations 13 and 14, we take $x_{kl}^{(t)}d_{kl}^{(t)}x_{ij}^{(t)} < -x_{ij}^{(t)}x_{kl}^{(t)}d_{ij}^{(t)} \overset{\div(-x_{kl}^{(t)}x_{ij}^{(t)}>0)}{\rightarrow} -d_{kl}^{(t)} < d_{ij}^{(t)} \Rightarrow d_{ij}^{(t)} + d_{kl}^{(t)} > 0$, so we take that $d_{ij}^{(t+1)} > 0$.
- $5^{th}$ *Case*: It holds from our case that if $x_{ij}^{(t)} < 0$ then $d_{ij}^{(t)} > 0$. For arc $(k, l)$ at the $t^{th}$ iteration, it holds: $d_{kl}^{(t)} < 0$ or $-d_{kl}^{(t)} > 0$ and therefore $d_{ij}^{(t)} - d_{kl}^{(t)} > 0$. Hence $d_{ij}^{(t+1)} > 0$.

Therefore, in any case $\exists d_{ij}^{(t+1)} > 0$. It only needs to prove that $\frac{x_{ij}^{(t+1)}}{-d_{ij}^{(t+1)}} < \frac{x_{kl}^{(t+1)}}{-d_{kl}^{(t+1)}}$. This will be done in two steps. First, it will be proved that $\frac{x_{ij}^{(t+1)}}{-d_{ij}^{(t+1)}} < \frac{x_{kl}^{(t)}}{-d_{kl}^{(t)}}$.

*Examination of the cases of $(i, j)$ from Table 1.*

- $1^{st}$ *Case*: Due to the induction hypothesis, it holds: $\frac{x_{ij}^{(t+1)}}{-d_{ij}^{(t+1)}} = \frac{x_{ij}^{(t)}}{-d_{ij}^{(t)}} < \frac{x_{kl}^{(t)}}{-d_{kl}^{(t)}} = \alpha^{(t)}$.
- $2^{nd}$ *and* $3^{rd}$ *Case*: In these cases since $(i, j) \notin T^{(t)}$ (it is the next entering arc), it holds $x_{ij}^{(t)} = 0$. However, we assumed that $x_{ij}^{(t)} < 0$. Therefore, these cases come into contradiction with the assumption and will not be examined.
- $4^{th}$ *and* $5^{th}$ *Case*: By the induction hypothesis, it holds $\frac{x_{ij}^{(t)}}{-d_{ij}^{(t)}} < \frac{x_{kl}^{(t)}}{-d_{kl}^{(t)}}$ and performing some algebraic manipulations, we take $d_{ij}^{(t)} > d_{kl}^{(t)} \frac{x_{ij}^{(t)}}{x_{kl}^{(t)}}$. Also, it holds $\frac{x_{ij}^{(t+1)}}{-d_{ij}^{(t+1)}} = \frac{\overbrace{-(x_{ij}^{(t)} \pm x_{kl}^{(t)})}^{>0}}{\underbrace{d_{ij}^{(t)} \pm d_{kl}^{(t)}}_{>0}}$. On substituting the quantity $d_{ij}^{(t)}$ and performing some algebraic manipulations, we take $\frac{x_{ij}^{(t+1)}}{-d_{ij}^{(t+1)}} < \frac{x_{kl}^{(t)}}{-d_{kl}^{(t)}}$. Hence $\frac{x_{ij}^{(t+1)}}{-d_{ij}^{(t+1)}} < \alpha^{(t)}$.

To complete the proof, it remains to show that $\alpha^{(t)} < \alpha^{(t+1)}$. However, this proof is based again on a case study, which is very similar to the previous one. Thus, completes the proof. Therefore, at every iteration it holds that $\beta = max\left\{\frac{x_{ij}}{-d_{ij}}\right\} < \alpha$. $\square$

**Theorem 6** *If $d(T) \geq 0$ and $P \neq \emptyset$, then the problem 1 is unbounded.*

**Proof** In case $d(T) = 0 \Rightarrow \sum_{(i,j)\in P} h_{ij} = 0$. Therefore the non basic arcs, represented by these columns, are linear depended. Accordingly, these arcs form a directed cycle. It is well known that $\sum_{(i,j)\in C} t_{ij}c_{ij} = \sum_{(i,j)\in C} t_{ij}s_{ij}$. Since $C$ is a directed cycle, then $t_{ij} = 1, \forall (i, j) \in C$. So, $\sum_{(i,j)\in C} c_{ij} = \sum_{(i,j)\in C} s_{ij}$ or $\sum_{(i,j)\in C} c_{ij} = S_o$ and therefore $\sum_{(i,j)\in C} c_{ij} < 0$. Followingly, since cycle $C$ has been identified as negative cost directed, our problem is unbounded

In case $d(T) > 0$, assume that $\exists (t_1, t_2) \in T : d_{t_1 t_2} = k > 0$ and $d_{ij} = 0, \forall (i, j) \in T \setminus \{(t_1, t_2)\}$. Without loss of generality we examine the case in which there is only one element having positive $d$ value. Assume that for the $d$ vector, it holds $d(T) = [0, 0, ..., k, ..., 0, 0]^T$. Then:

$$\sum_{(i,j)\in P} h_{ij} = [0, 0, ..., -k, ..., 0, 0]^T > 0 \tag{15}$$

We only need to prove that, in this case a negative cost directed cycle is formed. From the classic network simplex tableau it is known that, each basic arc is represented in terms of the basic arcs, using only itself. Then, for the $(t_1, t_2) \in T$, the column $h_{t_1 t_2}$ will be:

$$h_{t_1 t_2} = [0, 0, ..., 1, ..., 0, 0]^T \tag{16}$$

Hence, $\sum_{(i,j)\in P} h_{ij} + k h_{t_1 t_2} = [0, 0, ..., 0, ..., 0, 0]^T$

Therefore, arcs $(i, j) \in P$ with the addition of $k$ times the arc $(t_1, t_2)$ are linear depended and therefore they form $k$ directed cycles. We also need to prove that, the sum of the costs of the arcs, which form the directed cycles, is negative. Since $(t_1, t_2) \in T$, it holds $s_{t_1 t_2} = 0$. So, $\sum_{(i,j)\in P} s_{ij} + k s_{t_1 t_2} = \sum_{(i,j)\in P} s_{ij} = S_o < 0$. To conclude with, there are $k$ cycles, different between them, which are formed. These cycles consists of arcs belonging to $P$ as also of the basic arc $(t_1, t_2)$ and constitute a larger expanded cycle. Arc $(t_1, t_2)$ participates in each of them $k$ cycles. Therefore, all the different combinations of arcs, which may form a cycle, are negative cost oriented. Therefore, our problem is unbounded.

However, in the particular algorithm it remains to prove that if $\exists (t_1, t_2) \in T : x_{t_1 t_2} < 0$, then $(t_1, t_2) \in C$. Otherwise, no matter how much flow we send through the cycle and minimize the value of the objective function, the problem would have been unfeasible, since it would hold that $x_{ij} < 0$. If $\exists (t_1, t_2) \in T : x_{t_1 t_2} < 0$, then according to Theorem 5 $d_{t_1 t_2} > 0$. Therefore arc $(t_1, t_2)$ participates in the, previously mentioned, negative cost directed cycles. $\square$

**Theorem 7** *The y flow remains always feasible*

**Proof** We need to show $y \geq 0 \Leftrightarrow x + ad \geq 0 \Leftrightarrow x \geq -ad$. Therefore this relation must be proved that it holds for any arc, or:

$$\forall (i, j) : x_{ij} \geq -a d_{ij} \tag{17}$$

*Examination of all posible cases for any arc.*

- 1st *Case*: If $d_{ij} < 0$, then according to Theorem 5, $x_{ij} \geq 0$. So, relation 17 becomes $x_{ij} \geq -ad_{ij} \overset{\div(-d_{ij})>0}{\Rightarrow} \frac{x_{ij}}{-d_{ij}} \geq a$ which is true by the definition of $a$.

- 2nd *Case*: If $d_{ij} > 0$, then the following sub-cases exist:
  - · If $x_{ij} \geq 0$, then since $a,\ d_{ij} > 0$, it holds that $y \geq 0$
  - · If $x_{ij} < 0$, then the relation 17 becomes $x_{ij} \geq -ad_{ij} \overset{\div(-d_{ij})<0}{\Rightarrow} \frac{x_{ij}}{-d_{ij}} \leq a$ which is true according to Theorem 5.
- 3rd *Case*: If $d_{ij} = 0$ then, according to Theorem 5, $x_{ij} \geq 0$. Therefore, $y = x + ad \overset{d_{ij}=0}{\Rightarrow} y = x \Rightarrow y \geq 0$.

Therefore, in any case the $y$ flow remains feasible. □

**Theorem 8** *If $P = \emptyset$, then the algorithm has reached an optimal solution.*

**Proof** At every iteration it holds:

$$d = \{d(T), d(P), d(Q)\} = \left\{ d(T), \overbrace{1, 1, \ldots, 1}^{d(P)}, \overbrace{0, 0, \ldots, 0}^{d(Q)} \right\} \tag{18}$$

According to the definition of $d(T)$, $\forall(i,j) \in T$ the following relation holds

$$-|P| \leq d_{ij} \leq |P| \tag{19}$$

If $P = \oslash \Rightarrow |P| = 0$, so relation 19 will become:

$$d_{ij} = 0, \forall(i,j) \in T \Rightarrow d(T) = 0 \tag{20}$$

Using relation 20 in relation 18, the later becomes $d = \{0, 0, ..., 0\}$. Therefore, since it holds $y = x + ad \overset{d=0}{\Rightarrow} y = x$. According to Theorem 7, the $y$ flow remains always feasible, so we take $x \geq 0$. This means that the $x(T)$ flow is primal feasible. Combining also the fact that $P = \emptyset \Rightarrow$ the basis tree $T$ is also dual feasible, we conclude that the basis tree $T$ is the optimal solution. □

## 5 NEPSA implementation and computational results

NEPSA was implemented using the Augmented Thread Index method, (ATI method), [13]. This method was chosen because it allows the fast update of the basic tree, as also it can easily identify the cycle that would have been created with the addition of the entering arc.

The computational study was carried out using a PC with an Intel Dual Core 2 Duo 6600 2,4GHz processor, 2 GB RAM DDR 3 800Mhz and Ubuntu 7.04 “Feisty Fawn” version. Furthermore, the Intel Fortran compiler 10.0.026 was used for the compiling of the source code with the -O3 compiler optimization option. There have been developed many network generators like [18] and many more. However, in this computational study, all the MCNF problem instances were created using the well known GRIDGEN generator [17].

Following in the Table 2 all the GRIDGEN parameters, used for producing the instances in this computational study, are presented analytically. More specifically, four classes of instances were developed. The first class have density 0.1%, the second 2.5%, the third 5.0% and finally the fourth have 10.0%. All classes of instances consists of six problem categories, with different dimensions. The number of the nodes in the first sparse class of instances, starts from 5000 and are up to 10,000, with step equal to 1000, (so this way the six, previously mentioned, categories are built). On the other hand for the other three classes of instances, the number of nodes starts from 500 and are up to 1,000, with step equal to 100. The number of the arcs depends on the class of the instance. Moreover, in each one category of the classes, ten instances have been created, in order to compute the average number of the iterations and also of the total time. To conclude with, 240 MCNFP instances have been created and solved.

The empirical behaviour of the NEPSA now will be presented using the following computational results, concerning the number of iterations and the CPU time for the solution of each instance.

In the following Figures 3, 4, 5 and 6 one can see the graphical plots of the average number of iterations as well as the average number of CPU time. All the figures have been created with gnuplot version 4.0.0.5. These average numbers come up from the instances which were solved in each category of the four classes.

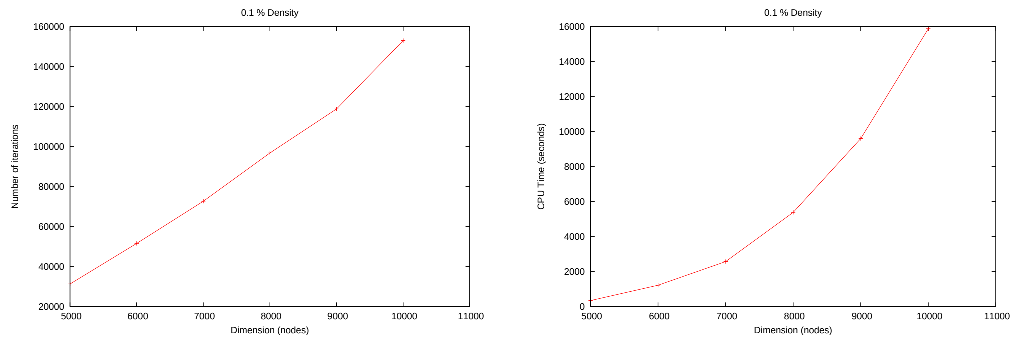


Fig. 3. Computational results NEPSA for problem instances of density 0.1%

It can be seen that in all problem classes, the number of iterations of NEPSA seems to grow in a semi-linear way. The same computational behaviour also

| Density | two-way arcs | nodes | grid width | source nodes | sink nodes | average degree | total flow | distribution of arc costs, (1 for UNIFORM) | cost lower bound | cost upper bound |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 5,000 | 100 | 2,500 | 1,500 | 5 | 100,000 | 1 | 1 | 100 |
| | 1 | 6,000 | 100 | 3,000 | 2,000 | 6 | 200,000 | 1 | 1 | 100 |
| 0.1% | 1 | 7,000 | 100 | 3,500 | 2,500 | 7 | 300,000 | 1 | 1 | 100 |
| | 1 | 8,000 | 100 | 4,000 | 3,000 | 8 | 400,000 | 1 | 1 | 100 |
| | 1 | 9,000 | 100 | 4,000 | 3,000 | 9 | 400,000 | 1 | 1 | 100 |
| | 1 | 10,000 | 100 | 5,000 | 4,000 | 10 | 400,000 | 1 | 1 | 100 |
| | 1 | 500 | 100 | 250 | 150 | 12 | 100,000 | 1 | 1 | 100 |
| | 1 | 600 | 100 | 300 | 200 | 15 | 200,000 | 1 | 1 | 100 |
| 2.5% | 1 | 700 | 100 | 350 | 250 | 17 | 300,000 | 1 | 1 | 100 |
| | 1 | 800 | 100 | 400 | 300 | 20 | 400,000 | 1 | 1 | 100 |
| | 1 | 900 | 100 | 400 | 300 | 22 | 400,000 | 1 | 1 | 100 |
| | 1 | 1,000 | 100 | 500 | 400 | 25 | 400,000 | 1 | 1 | 100 |
| | 1 | 500 | 100 | 250 | 150 | 25 | 100,000 | 1 | 1 | 100 |
| | 1 | 600 | 100 | 300 | 200 | 30 | 200,000 | 1 | 1 | 100 |
| 5% | 1 | 700 | 100 | 350 | 250 | 35 | 300,000 | 1 | 1 | 100 |
| | 1 | 800 | 100 | 400 | 300 | 40 | 400,000 | 1 | 1 | 100 |
| | 1 | 900 | 100 | 400 | 300 | 45 | 400,000 | 1 | 1 | 100 |
| | 1 | 1,000 | 100 | 500 | 400 | 50 | 400,000 | 1 | 1 | 100 |
| | 1 | 500 | 100 | 250 | 150 | 50 | 100,000 | 1 | 1 | 100 |
| | 1 | 600 | 100 | 300 | 200 | 60 | 200,000 | 1 | 1 | 100 |
| 10% | 1 | 700 | 100 | 350 | 250 | 70 | 300,000 | 1 | 1 | 100 |
| | 1 | 800 | 100 | 400 | 300 | 80 | 400,000 | 1 | 1 | 100 |
| | 1 | 900 | 100 | 400 | 300 | 90 | 400,000 | 1 | 1 | 100 |
| | 1 | 1,000 | 100 | 500 | 400 | 100 | 400,000 | 1 | 1 | 100 |

Table 2

GRIDGEN parameters

| Density | nodes | arcs | CPU Time | Number of Iterations |
|---|---|---|---|---|
| 0.1% | 5,001 | 25,005 | 345.250 | 31,348 |
| | 6,001 | 36,006 | 1,222.761 | 51,605 |
| | 7,001 | 49,007 | 2,574.591 | 72,688 |
| | 8,001 | 64,008 | 5,386.986 | 96,839 |
| | 9,001 | 81,009 | 9,603.110 | 118,849 |
| | 10,001 | 100,010 | 15,883.072 | 153,017 |
| 2.5% | 501 | 6,012 | 9.621 | 5,167 |
| | 601 | 9,015 | 19.834 | 7,002 |
| | 701 | 11,917 | 31.782 | 9,050 |
| | 801 | 16,020 | 61.493 | 11,529 |
| | 901 | 19,822 | 88.979 | 13,208 |
| | 1,001 | 25,025 | 176.626 | 16,649 |
| 5% | 501 | 12,525 | 30.702 | 7,423 |
| | 601 | 18,030 | 70.790 | 10,044 |
| | 701 | 24,535 | 102.182 | 13,282 |
| | 801 | 32,040 | 177.980 | 16,607 |
| | 901 | 40,545 | 251.108 | 19,669 |
| | 1,001 | 50,050 | 434.978 | 25,097 |
| 10% | 501 | 25,050 | 88.160 | 11,078 |
| | 601 | 36,060 | 170.834 | 15,420 |
| | 701 | 49,070 | 323.727 | 20,315 |
| | 801 | 64,080 | 631.264 | 25,991 |
| | 901 | 81,090 | 885.870 | 31,253 |
| | 1,001 | 100,100 | 1,669.825 | 39,402 |

Table 3

Computational results

holds for the cpu time, perhaps with the exception of the first sparse class. In the first sparse class of 0.1% density, the cpu time grow with an exponential behaviour. However, this could be due to the fact that the first class was the bigger class of instances solved by this first version of NEPSA.

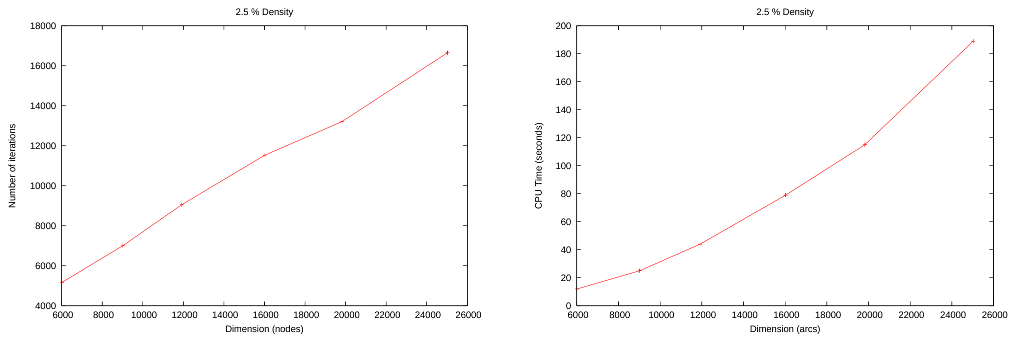


Fig. 4. Computational results NEPSA for problem instances of density 2.5%

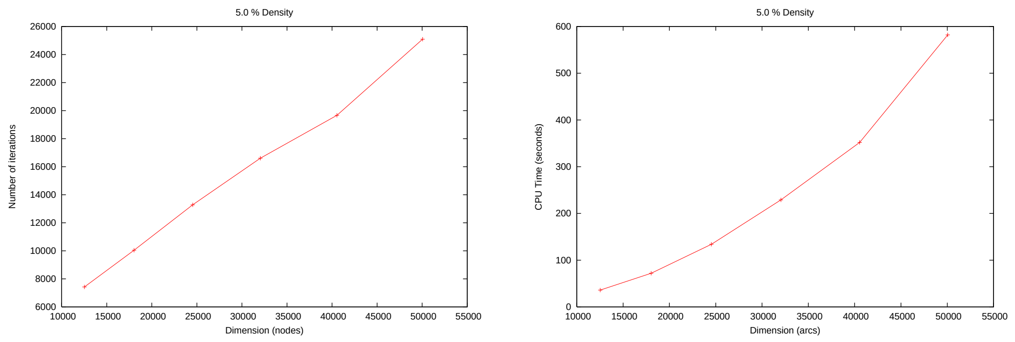


Fig. 5. Computational results NEPSA for problem instances of density 5.0%

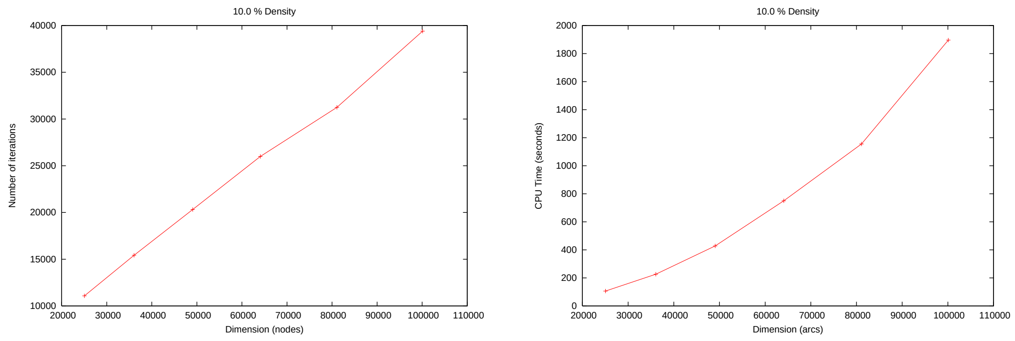


Fig. 6. Computational results NEPSA for problem instances of density 10.0%

## 6 Conclusions and future work

In this paper a new simplex type algorithm for MCNFP is analytically presented. An efficient implementation of NEPSA will be the subject of our future work. There have been developed many efficient data structures. Some of them are the dynamic trees as described in [12], Fibonacci heaps as described in [10] or other class of data structures using depth, parent, and preorder lists as described in [3]. The computational complexity of an algorithm depends greatly on the data structures used for updating all the necessary variables. Therefore, it is interesting to use such data structures, in the implementation of NEPSA.

If certain improvements will be made, regarding the data structures used and efficient programming techniques, then it is possible to test NEPSA against some of the state-of-the-art implementations, like for example RELAX-IV [6], NETPD [8] or RNET [15]. Numerical experiments will demonstrate the efficiency of NEPSA relative to existing algorithms. However, it would require first to develop appropriate anti-cycling rules, and examine the behavior of NEPSA to certain well known pathological instances (at least for the classical NPSA) [27] and [28]. Even more, it would be very interesting to develop a data structure similar to the well known strongly feasible tree [7], for use in NEPSA. Moreover, it is also interesting to develop similar type algorithms for other well known problems, like the shortest path problem.

Finally, NEPSA will be incorporated in the Network Optimization suite WebNetPro [16]. This way, WebNetPro will expand its capabilities with new algorithms for MCNFP.

## 7 Acknowledgments

Research supported by Operational Program for Educational and Vocational Training II (EPEAEK II), and particularly the program HRAKLEITOS.

## References

[1] R.K. Ahuja, A.V. Goldberg, J.B. Orlin and R.E. Tarjan. Finding minimum - cost flows by double scaling. Math. Program. 1992;53: 243-266.

[2] R.K. Ahuja, T.L. Magnanti, J.B. Orlin and M.R. Reddy, Applications of Network Optimization, in: M.O. Ball, T.L. Magnanti, C.L. Monma and G.L. Nemhauser (eds.), Handbooks of Operations Research and Management Science, 7: Network Models, Elsevier Publications, 1995. p. 1-83.

[3] A.I. Ali, R.V. Helgason, J.L. Kennington and H.S. Lall, Primal simplex network codes: state-of-the-art implementation technology, Networks 1978;8: 315-339.

[4] M.S. Bazaraa, J.J. Jarvis and H.D. Sherali, Linear Programming and Network Flows, (3rd ed.), John Wiley & Sons, Inc., Hoboken, New Jersey, 2005.

[5] R.G. Bland, New finite pivoting rules for the simplex method, Math. Oper. Res. 1977;2: 103-107.

[6] D.P. Bertsekas and P.Tseng, RELAX-IV: a faster version of the RELAX code for solving minimum cost flow problems, Technical report, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1994.

[7] W.H. Cunningham, A network simplex method, Math. Program. 1976;11: 105-116.

[8] N.D. Curet, Applying steepest-edge techniques to a network primal-dual algorithm, Comp. Op Res. 1997;24(7): 601-609.

[9] J. Edmonds and R.M. Karp, Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems, J. ACM 1972;19: 248-264.

[10] M.C. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. ACM 1987;34(3): 596-615.

[11] F. Glover, D. Klingman and N. Phillips, Network Models in Optimization and Their Applications in Practice, first ed., Wiley, 1992.

[12] A.V. Goldberg, M.D. Grigoriadis and R.E. Tarjan, Use of dynamic trees in a network simplex algorithm for the maximum flow problem, Math. Program. 1991;50: 277-290.

[13] F. Glover, D. Karney, and D. Klingman, The augmented predecessor index

method for locating stepping stone paths and assigning dual prices in distribution problems, Transportation Science 1972;6: 171-180.

[14] A.V. Goldberg and R. Tarjan, Solving minimum - cost flow problems by successive approximation, in: Proc. Nineteenth annual ACM Conference on Theory of computing 1987. p. 7-18.

[15] M. Grigoriadis, An Efficient Implementation of the Network Simplex Method, Math. Program. Study 1984;26: 83-111.

[16] P. Karagiannis, I. Markelis, K. Paparrizos, N. Samaras and A. Sifaleras, E-learning technologies: employing matlab web server to facilitate the education of mathematical programming, Internat. J. Math. Ed. Sci. Tech. 2006;37(7): 765-782.

[17] Y. Lee and J.B. Orlin, Computational testing of a network simplex algorithm, in: Proc. of the First DIMACS International Algorithm Implementation Challenge, 1991.

[18] C.M. Murphy and M.S. Hung, Network generation using the prufer code, Comp. Op Res. 1986;13(6): 693-705.

[19] J.B. Orlin, A Faster Strongly Polynomial Minimum Cost Flow Algorithm, Oper. Res. 1993;41: 338-350.

[20] J.B. Orlin, A polynomial time primal network simplex algorithm for minimum cost flows, Math. Program. 1997;78: 109-129.

[21] K. Paparrizos, N. Samaras, and K. Tsiplidis, Pivoting algorithms for (LP) generating two paths, in: P. Pardalos and C. Floudas (eds.), Encyclopedia of

Optimization, Kluwer Academic, 2001;4: 302-306.

[22] K. Paparrizos, An infeasible (exterior point) simplex algorithm for assignment problems, Math. Program. 1991;51: 45-54.

[23] K. Paparrizos, N. Samaras and A. Sifaleras, Some preliminary results on the cycling problem for the network exterior point simplex algorithm, in: Proc. Seventh International Conference on Operations and Quantitative Management, Jaipur, India, 2006. p. 536-540.

[24] M.G.C. Resende and P.M. Pardalos, Interior point algorithms for network flow problems, in: J. Beasley (ed.), Advances in linear and integer programming, 1996. p. 147-187.

[25] H.D. Sherali, B. Ozdaryal, W.P. Adams and N. Attia, On using exterior penalty approaches for solving linear programming problems, Comp. Op Res. 2001;28(11): 1049-1074.

[26] E. Tardos, A Strongly Polynomial Minimum Cost Circulation Algorithm, Combinatorica 1985;5: 247-255.

[27] N. Zadeh, More Pathological Examples for Network Flow Problems, Math. Program. 1973;5: 217-224.

[28] N. Zadeh, A bad network problem for the simplex method and other minimum cost flow algorithms, Math Program. 1973;5: 255-266.