RAIRO-Oper. Res. 46 (2012) 211–234
DOI: 10.1051/ro/2012015

RAIRO Operations Research
www.rairo-ro.org

ON A DUAL NETWORK EXTERIOR POINT SIMPLEX TYPE ALGORITHM AND ITS COMPUTATIONAL BEHAVIOR *

George Geranis[1], Konstantinos Paparrizos[1] and Angelo Sifaleras[2]

Abstract. The minimum cost network flow problem, (MCNFP) constitutes a wide category of network flow problems. Recently a new dual network exterior point simplex algorithm (DNEPSA) for the MCNFP has been developed. This algorithm belongs to a special "exterior point simplex type" category. Similar to the classical dual network simplex algorithm (DNSA), this algorithm starts with a dual feasible tree-solution and after a number of iterations, it produces a solution that is both primal and dual feasible, *i.e.* it is optimal. However, contrary to the DNSA, the new algorithm does not always maintain a dual feasible solution. Instead, it produces tree-solutions that can be infeasible for the dual problem and at the same time infeasible for the primal problem. In this paper, we present for the first time, the mathematical proof of correctness of DNEPSA, a detailed comparative computational study of DNEPSA and DNSA on sparse and dense random problem instances, a statistical analysis of the experimental results, and finally some new results on the empirical complexity of DNEPSA. The analysis proves the superiority of DNEPSA compared to DNSA in terms of cpu time and iterations.

Keywords. Network flows, minimum cost network flow problem, dual network exterior point simplex algorithm.

Mathematics Subject Classification. 90C27, 65K05, 90B10, 91A90.

Received November 16, 2010. Accepted July 11, 2012.

* *This paper is dedicated to the late Professor Paparrizos Konstantinos.*

[1] Department of Applied Informatics, University of Macedonia, 156 Egnatia Str., 54006 Thessaloniki, Greece. geranis@uom.gr; paparriz@uom.gr

[2] Department of Technology Management, University of Macedonia, Loggou-Tourpali, 59200 Naoussa, Greece. sifalera@uom.gr

Article published by EDP Sciences

1. Introduction

The MCNFP is the problem of finding a minimum cost flow of product units, through a number of supply nodes (sources), demand nodes (sinks), and transshipment nodes. Other common problems, such the shortest path problem, the transportation problem, the transshipment problem, the assignment problem, etc., are special cases of the MCNFP. The MCNFP appears very frequently in different sectors of technology, like informatics, telecommunications, transportation, etc. Numerous real life problems can be solved, by applying network flow models as described in [2, 16].

The MCNFP can be easily transformed into a linear programming problem, and well-known general linear programming algorithms could be applied in order to find an optimal solution. Such algorithms do not take advantage of some special features met in the MCNFP. Therefore, other special Simplex-type algorithms have been developed, such as the primal network simplex algorithm and the dual network simplex algorithm. There are also other non Simplex-type algorithms that can be used for solving the same problem as presented in [9, 26].

This paper presents, an exterior point dual simplex-type algorithm for the MCNFP. The algorithm is named dual network exterior point simplex algorithm (DNEPSA for short) for the MCNFP. DNEPSA starts from a dual feasible tree-solution and, iteration by iteration, it produces new tree-solutions closer to an optimal solution, reducing the problem's infeasibility. Contrary to the dual network simplex algorithm (DNSA for short), the tree-solution at every iteration is not necessarily dual feasible. The algorithm computes the direction towards to the dual feasible area by maintaining a direction vector d. After a number of iterations, vector d becomes equal to zero. This happens because the current tree-solution is both primal and dual feasible and therefore it is optimal. It is worth mentioning that, DNEPSA is quite different from other pivot algorithms such as the criss-cross Simplex method. This is due to the fact that, the trace of a criss-cross method is not monotonic with respect to the objective function, [11]. A primal exterior point simplex-type algorithm for the MCNFP has been recently reported in [27]. A preliminary geometrical interpretation of DNEPSA was described in [12], while in this paper we show for the first time (i) the algorithm's mathematical proof of correctness, (ii) an encouraging comparative computational study of DNEPSA and DNSA, (iii) a statistical analysis of the experimental results, and finally (iv) some new results on the empirical complexity of DNEPSA.

Section 2 gives the notation that will be used in this paper and a short description of the MCNFP. In Section 3, the steps followed by DNEPSA are analytically described, while Section 4 shows the algorithm's mathematical proof of correctness. Section 5 presents the results of the comparative experimental analysis between DNEPSA and DNSA, and a statistical analysis of the performance evaluation follows in Section 6. Section 7 presents some new experimental results on the empirical complexity of DNEPSA. Finally, Section 8 provides some conclusions and plans for future work.

2. Problem statement and notation

Let $G = (N, A)$ be a directed network that consists of a finite set of nodes N and a finite set of directed arcs A. Let n and m be the number of nodes and arcs, respectively. For each node $i \in N$, there is an associated variable b_i representing the available supply or demand at that node. Node i is a *supply node* (source), if $b_i > 0$. On the other hand it is a *demand node* (sink), if $b_i < 0$. Finally, the node i is a *transshipment node* in the case that $b_i = 0$. Moreover, we consider that the total supply is equal to the total demand, *i.e.*, it holds $\sum_{i \in N} b_i = 0$ (balanced network).

For every arc $(i, j) \in A$ we have an associated flow x_{ij} that shows the amount of product units transferred from node i to node j and an associated cost per unit value c_{ij}. Therefore, the total cost is equal to $\sum_{(i,j)\in A} c_{ij}x_{ij}$, and the MCNFP is the problem of finding a flow that minimizes that total cost. We can have for the flow x_{ij} a lower and an upper bound, l_{ij} and u_{ij}, respectively. This gives an additional constraint $l_{ij} \leq x_{ij} \leq u_{ij}$ for every arc $(i, j) \in A$. In our case, we consider that $l_{ij} = 0$ and $u_{ij} = +\infty, \forall (i, j) \in A$. In other words, our algorithm is applied to the *uncapacitated* MCNFP. For every node $i \in N$, it has to be:

$$\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = b_i,$$

because the outgoing flow must be equal to the incoming flow plus the node's supply. Therefore, the mathematical formulation of the MCNFP is as follows:

$$\text{minimize} \quad z = \sum_{(i,j)\in A} c_{ij}x_{ij}, \tag{2.1}$$

subject to

$$\begin{aligned} &\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = b_i, \forall i \in N, \\ &x_{ij} \geq 0, \forall (i, j) \in A. \end{aligned} \tag{2.2}$$

Since $\sum_{i \in N} b_i = 0$, by using formulas (2.2), it comes out that:

$$\sum_{i\in N} \left(\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} \right) = 0.$$

That means that constraints (2.2) are linearly dependent and we can arbitrarily drop out one of them. In matrix notation format the problem can be expressed as follows:

$$\begin{aligned} &\text{minimize} \quad z = c^T x, \\ &\text{s.t.} \quad Ax = b, \\ &x \geq 0, \end{aligned} \tag{2.3}$$

where $A \in \Re^{n\times m}$, $c \in \Re^m$, $x \in \Re^m$ and $b \in \Re^n$. Notation c^T denotes the transpose of vector c. There is a set of dual variables w_i, one for every node, and a number of reduced cost variables s_{ij}, one for every directed arc. These are the variables used for the formulation of the dual problem. In matrix notation format, the dual problem has the following form:

$$\begin{aligned} &\text{maximize } z = b^T w, \\ &s.t.\ A^T w + I_m s = c, \\ &s \geq 0, \end{aligned} \tag{2.4}$$

where $A \in \Re^{n\times m}$, $c \in \Re^m$, $w \in \Re^n, s \in \Re^m$, $b \in \Re^n$ and I_m is the unit matrix of size m. Network simplex-type algorithm starts from a basic tree-solution and compute vectors (*i.e.*, x, w, s). If for a tree-solution T, $x_{ij} \geq 0$ for every arc $(i, j) \in T$, then that solution is said to be *primal feasible*. If for a tree-solution T, $s_{ij} \geq 0$ for every arc $(i, j) \notin T$ then it is said to be *dual feasible*. A solution being both primal and dual feasible is an optimal solution. Primal network simplex-type algorithms start from a primal feasible tree-solution and they move, at every iteration, to a new primal feasible solution, until they find an optimal solution. On the other hand, dual network simplex-type algorithms start from a dual feasible tree-solution and they reach an optimal solution, by following successive dual feasible solutions.

3. Algorithm description

The DNEPSA, starts from a dual feasible basic tree-solution T, and after a number of iterations, it comes to a tree-solution that is both primal and dual feasible. The main difference between DNEPSA and the existing dual network simplex-type algorithms, is the fact that the tree-solutions formed during the iterations of DNEPSA are not necessarily always dual feasible. In other words, DNEPSA starts from a dual solution and reaches an optimal solution by following a route consisting of solutions that do not always belong to the feasible area of the dual solution. Furthermore, DNEPSA contrary to the classical DNSA first selects the entering arc and afterwards selects the leaving arc. Finally, both entering and leaving arcs are selected using different rules for the DNEPSA and DNSA algorithms respectively.

Step 0 (Initializations). Various methods can be used to find a starting dual feasible tree-solution. An algorithm that can construct a dual feasible tree-solution for the generalized network problem (and also for pure networks) is described in [14], and an improved version of the algorithm is presented in [20], which gives a dual feasible solution that is closer to an optimal solution.

The starting dual feasible solution T consists of $n - 1$ directed arcs that form a tree. These arcs and the corresponding flows are called *basic arcs* and *basic variables*, respectively. For the non basic arcs $(i, j) \notin T$ it is $x_{ij} = 0$ and $s_{ij} \geq 0$,

while for the basic arcs $(i,j) \in T$ it is $s_{ij} = 0$. The values of the dual variables w_i, $1 \leq i \leq n$, can be easily computed from the following equations:

$$w_i - w_j = c_{ij},\ \forall (i,j) \in T. \tag{3.1}$$

In equation (3.1) we have $n-1$ equations and n variables, so we can choose one of the dual variables (*e.g.*, w_1) and set it equal to an arbitrary value (*e.g.*, 0). Then, it is easy to compute the values for the rest of the dual variables. In order to compute the reduced costs s_{ij} for the non-basic arcs (i,j), we can use the following equation:

$$s_{ij} = c_{ij} - w_i + w_j,\ \forall (i,j) \notin T, \tag{3.2}$$

while $s_{ij} = 0$ for all the basic arcs. Next, the algorithm creates a set named I_- that contains the basic arcs (i,j) having negative flow and a set I_+ containing the rest of the arcs:

$$I_- = \{(i,j) \in T : x_{ij} < 0\}, \quad I_+ = \{(i,j) \in T : x_{ij} \geq 0\}. \tag{3.3}$$

If a non-basic arc (i,j) is added into the basic tree T, then a cycle C is created. In that case, let h be the vector of orientations of all basic arcs relative to the entering arc (i,j). If an arc (u,v) in C has the same orientation as (i,j), then it is $h_{uv} = -1$, otherwise it is $h_{uv} = +1$. For an arc (u,v) not belonging to C, it is $h_{uv} = 0$. In every iteration, DNEPSA finds out a new basic tree-solution which is probably neither primal nor dual feasible. The algorithm computes the direction towards to the feasible region of the dual problem by maintaining a direction vector d. Vector d is computed by using the following formula:

$$\begin{aligned} &d_{ij} = 1,\ \ if\,(i,j) \in I_-, \\ &d_{ij} = 0,\ if\,(i,j) \in I_+, \\ &d_{ij} = \sum_{(u,v)\in I_-} h_{uv},\ \ if\,(i,j) \notin T. \end{aligned} \tag{3.4}$$

Step 1 (Test of optimality). If $I_- = \emptyset$, this means that the current tree-solution is optimal. Otherwise, DNEPSA creates a set named J_- defined as:

$$J_- = \{(i,j) \notin T : s_{ij} \geq 0 \text{ and } d_{ij} < 0\}. \tag{3.5}$$

If $(J_- = \emptyset) \wedge (I_- \neq \emptyset)$, then the problem is infeasible.

Step 2 (Choice of entering arc). DNEPSA uses J_-, to compute the following minimal ratio:

$$\alpha = \frac{s_{gh}}{-d_{gh}} = \min\left\{\frac{s_{ij}}{-d_{ij}} : (i,j) \in J_-\right\}. \tag{3.6}$$

This ratio, as it is seen in (3.6), is used in order to choose the *entering arc* (g,h). After (g,h) is added into the basic tree T, then a cycle C is created. When a unit

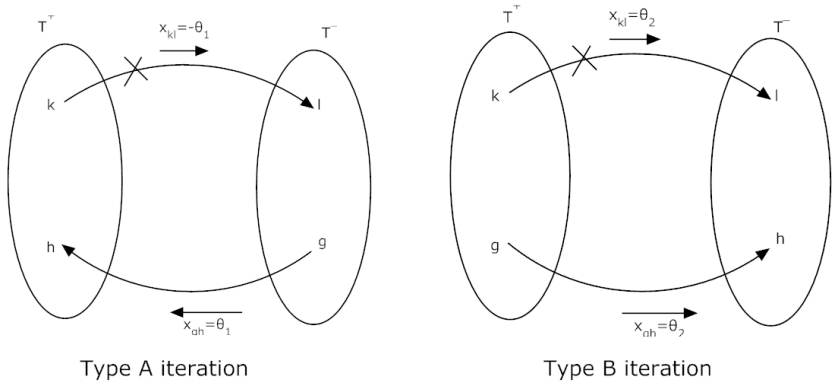

FIGURE 1. Type A and type B iterations.

of product flows through C, then the value of the objective function is changed by the following amount:

$$\Delta z = \sum_{(i,j)\in C} t_{ij} c_{ij},$$

where t_{ij} equals 1 if the arcs (i,j) and (g,h) have the same orientation in cycle C, otherwise $t_{ij} = -1$. By using equations (3.1) for arcs $(i,j) \neq (g,h)$ and equation (3.2) we take:

$$\Delta z = \sum_{(i,j)\in C} t_{ij} c_{ij} = c_{gh} - w_g + w_h = s_{gh}. \tag{3.7}$$

Step 3 (Choice of leaving arc). In order to find the *leaving arc* (k,l), DNEPSA calculates the following values:

$$\begin{aligned} \theta_1 &= -x_{k_1 l_1} = \min\{-x_{ij} : (i,j) \in I_- \text{ and } (i,j) \uparrow\uparrow (g,h)\}, \\ \theta_2 &= \ x_{k_2 l_2} = \min\{\ x_{ij} : (i,j) \in I_+ \text{ and } (i,j) \uparrow\downarrow (g,h)\}, \end{aligned} \tag{3.8}$$

where notation $\uparrow\uparrow$ is used for arcs that have the same orientation, while notation $\uparrow\downarrow$ stands for arcs of opposite orientation to each other. The algorithm compares the values of θ_1 and θ_2. If $\theta_1 \leq \theta_2$, then arc (k_1, l_1) is the leaving arc. In that case, we say we have a *type A iteration*. An arc of negative flow $x_{kl} = -\theta_1$ is leaving and an arc with flow $x_{gh} = \theta_1$ is entering the basic solution, as it is seen in Figure 1. For the leaving arc (k,l), the subtree containing node k is denoted by T^+, while the other subtree is denoted by T^-. If, on the other hand, $\theta_1 > \theta_2$, then arc (k_2, l_2) is the leaving arc. In that case, we say we have a *type B iteration*. An arc of positive flow $x_{kl} = \theta_2$ is leaving and an arc with flow $x_{gh} = \theta_2$ is entering the basic solution, as it is seen in Figure 1.

Step 4 (Pivoting). After finding the entering and the leaving arc, the algorithm comes to a new tree-solution, closer to an optimal solution, and iteration by iteration it finds an optimal solution. The formal description of DNEPSA, in pseudocode, follows in Algorithm 1.

Algorithm 1. DNEPSA

Require: $G = (N, A), b, c, T$
procedure DNEPSA(G, T)
Step 0 (Initializations)
    Compute x, w, and s, using relations (2.1), (3.1), and (3.2) respectively
    Find sets I_- and I_+, using relations (3.3)
    Compute vector d, using relation (3.4)
Step 1 (Test of optimality)
    while $I_- \neq \emptyset$ **do**
        Find set J_-, using relation (3.5)
        if $J_- = \emptyset$ **then**
            STOP. The problem 2.1 is infeasible
        else
Step 2 (Choice of entering arc)
            Compute α, using relation (3.6)
            Choose the entering arc (g, h)
Step 3 (Choice of leaving arc)
            Compute θ_1, θ_2, using relations (3.8)
            Choose the leaving arc (k, l)
Step 4 (Pivoting)
            Set $T = T \setminus (k, l) \cup (g, h)$
            Update x, s, and d
            if $\theta_1 \leq \theta_2$ **then**
                Set $I_- = I_- \setminus (k, l)$ and $I_+ = I_+ \cup (g, h)$
            else
                Set $I_+ = I_+ \cup (g, h) \setminus (k, l)$
            end if
        end if
    end while
    STOP. The problem 2.1 is optimal.
end procedure

It is not necessary for the algorithm in every iteration to compute the values of variables x_{ij}, s_{ij}, and d_{ij} or to create sets I_- and I_+ from scratch. These variables and sets can be efficiently updated from iteration to iteration. Notation $x_{ij}^{(t)}$ means the flow on arc (i, j) during iteration t of the algorithm. Similar notation is used for variables s_{ij} and d_{ij}. After adding the entering arc into the basic tree during the t iteration, a cycle, denoted $C^{(t)}$ is created. For the basic arcs $(i, j) \in T$ we can have the different cases, shown in Figure 2.

In iteration $t + 1$, for every basic arc (i, j), flow $x_{ij}^{(t+1)}$ depends on the flow of the arc in the previous iteration $x_{ij}^{(t)}$ and the flow of the leaving arc $x_{kl}^{(t)}$, as it

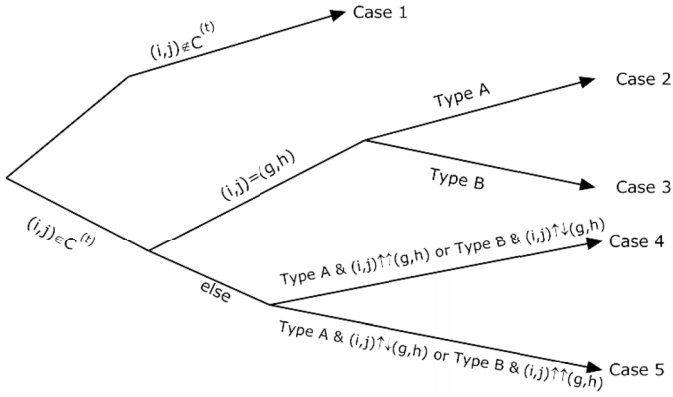

FIGURE 2. Possible cases for the basic arcs (i, j).

TABLE 1. Update of x_{ij} (Fig. 2), s_{ij} and d_{ij} (Fig. 3).

Cases	$x_{ij}^{(t+1)}$	$s_{ij}^{(t+1)}$	$d_{ij}^{(t+1)}$
Case 1	$x_{ij}^{(t)}$	$s_{ij}^{(t)}$	$d_{ij}^{(t)}$
Case 2	$-x_{kl}^{(t)} = \theta_1$	$s_{ij}^{(t)} + s_{gh}^{(t)}$	$d_{ij}^{(t)} + d_{gh}^{(t)}$
Case 3	$x_{kl}^{(t)} = \theta_2$	$s_{ij}^{(t)} - s_{gh}^{(t)}$	$d_{ij}^{(t)} - d_{gh}^{(t)}$
Case 4	$x_{ij}^{(t)} - x_{kl}^{(t)}$	$s_{ij}^{(t)} - s_{gh}^{(t)}$	$d_{ij}^{(t)} - d_{gh}^{(t)}$
Case 5	$x_{ij}^{(t)} + x_{kl}^{(t)}$	$s_{ij}^{(t)} + s_{gh}^{(t)}$	$d_{ij}^{(t)} + d_{gh}^{(t)}$

is shown in Table 1. For the non basic arcs $(i, j) \notin T$ we can also have different cases, depending on the type of iteration and the arc's position, as it is shown in Figure 3. In iteration $t + 1$, the reduced costs $s_{ij}^{(t+1)}$ for the non basic arcs (i, j) depend on the reduced costs in the previous iteration $s_{ij}^{(t)}$ and the reduced cost for the entering arc $s_{gh}^{(t)}$, as it is seen in Table 1. In a similar way, d_{ij} values are updated, according again to Table 1. In order to update the sets I_- and I_+, there are two cases to be considered depending on the type of the iteration: (1) type A iteration, (2) type B iteration.

- **Case 1.** for a type A iteration, both sets change according to the following formulas:

$$I_+ = I_+ \cup \{(g, h)\}, \qquad I_- = I_- - \{(k, l)\}. \tag{3.9}$$

- **Case 2.** for a type B iteration, only set I_+ changes according to the following formula:

$$I_+ = I_+ \cup \{(g, h)\} - \{(k, l)\}. \tag{3.10}$$

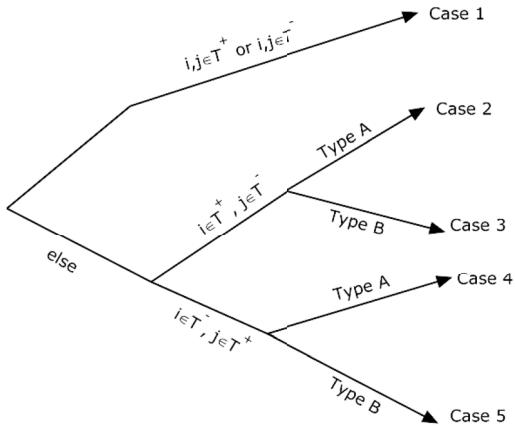

FIGURE 3. Possible cases for the non basic arcs (i, j).

4. Mathematical proof of correctness

In this section analytical proof of correctness for DNEPSA algorithm, will be presented. Although, we make the assumption for all theorems that the problem is not degenerate, a practical method to avoid the bad results due to degeneracy; stalling or cycling, was applied in the implementation of DNEPSA. More precisely, DNEPSA might have to choose between two or more equally qualified arcs, at the selection of the leaving or the entering arc. This phenomenon is usually denoted as a tie. DNEPSA was implemented in such a way that it breaks the ties using the following method. A numbering was given at each arc and always the arc with the minimum index was selected, between equally qualified leaving or entering arcs. This technique is similar to the rule of Bland for the general linear programming problem. Based on our computational experience on random generated sparse and dense problems, there was not any basic tree recurrence in successive iterations using this method. Moreover, we neither observed any long sequence of arcs leaving the basic tree and entering back again repeatedly. Other practical anti-degeneracy techniques in network linear programming, can be found in [23].

In Theorem 4.1 we prove that, for non-degerate pivots, the value of the objective function z increases strictly from iteration to iteration. This conclusion is used in Theorem 4.2, in order to prove that the algorithm terminates after a finite number of iterations.

Theorem 4.1. *The value of the objective function increases strictly from iteration to iteration.*

Proof. Let $z^{(t)}$ be the value of the objective function in iteration t. We will prove that $\Delta z = z^{(t+1)} - z^{(t)} > 0$. As it is shown in equation (3.7), for a product unit flow the objective function value changes by $\Delta z = s_{gh}$. For a type A iteration, the flow

on the entering arc (g, h) is equal to $\theta_1 = -x_{kl} > 0$. By taking also into account the non-degeneracy assumption, the total change equals $\theta_1 s_{gh} > 0$. Similarly, for a type B iteration, the total difference is equal to $\theta_2 s_{gh} > 0$. □

Theorem 4.2. *The algorithm terminates after a finite number of iterations.*

Proof. In Theorem 4.1 we proved that the value of the objective function strictly increases from iteration to iteration. That is, no tree-solution will be created twice. The number of trees that can be created for a given network is finite. Therefore, DNEPSA will perform a finite number of iterations before it terminates. □

In the next theorem, we prove that for every iteration, set I_- contains the basic arcs (i, j) of negative flow while set I_+ contains those of non-negative flow.

Theorem 4.3. *For all iterations, if* $(i, j) \in I_-$*, then* $x_{ij} < 0$ *and if* $(i, j) \in I_+$ *then* $x_{ij} \geq 0$.

Proof. We are going to use mathematical induction. Let's assume we have a type A iteration. For the first iteration, because of their definition, I_- contains the arcs having negative flow, while I_+ contains the arcs of the tree of non negative flow. We assume that it is true for iteration t. We'll show that it is also true for iteration $t + 1$. The elements of I_- are updated according to formula (3.9). Therefore, an arc that belongs to I_- during the $(t+1)$th iteration, also belongs to I_- during the tth iteration and, because of the assumption, it is $x_{ij}^{(t)} < 0$. We need to show that $x_{ij}^{(t+1)} < 0$. By examining the cases shown in Figure 2 and the variable updates of Table 1, we have

- *Case 1*: it is $x_{ij}^{(t+1)} = x_{ij}^{(t)} < 0$ because of the assumption.
- *Cases 2 and 3*: cannot hold because $(g, h) \in I_+$.
- *Case 4*: it is $\theta_1 = -x_{kl}^{(t)} < -x_{ij}^{(t)}$ because of (3.8).
  Therefore, $x_{ij}^{(t+1)} = x_{ij}^{(t)} - x_{kl}^{(t)} < 0$.
- *Case 5*: it is $x_{ij}^{(t+1)} = x_{ij}^{(t)} + x_{kl}^{(t)} < 0$ since $x_{ij}^{(t)} < 0$ and $x_{kl}^{(t)} < 0$.

The contents of set I_+ are also updated according to formula (3.9). Arc (g, h) is the only new arc in I_+ and, in a similar way, we have the following cases:

- *Case 1*: it is $x_{ij}^{(t+1)} = x_{ij}^{(t)} \geq 0$ because of the assumption.
- *Case 2*: it is $x_{gh} = \theta_1 = -x_{kl} \geq 0$.
- *Case 3*: cannot hold obviously.
- *Case 4*: it is $x_{ij}^{(t+1)} = x_{ij}^{(t)} - x_{kl}^{(t)} \geq 0$ because $x_{ij}^{(t)} \geq 0$ and $x_{kl}^{(t)} < 0$.
- *Case 5*: it is $x_{ij}^{(t+1)} = x_{ij}^{(t)} + x_{kl}^{(t)}$. It is also $\theta_2 \leq x_{ij}^{(t)}$ because of formulas (3.8), and $\theta_1 \leq \theta_2$ because we have iteration of type A.
  Therefore, $\theta_1 = -x_{kl}^{(t)} \leq x_{ij}^{(t)} \Rightarrow x_{ij}^{(t+1)} \geq 0$.

In a similar way, we can prove for type B iterations that, I_- contains the arcs having negative flow and I_+ contains those of non negative flow. □

The next theorem gives a property for the non basic arcs (i, j) that have negative reduced cost value. This property will be very important for the proof of the theorems that will follow.

Theorem 4.4. *If for a non basic arc* (i, j) *it is* $s_{ij} < 0$*, then* $d_{ij} > 0$ *and* $\frac{s_{ij}}{-d_{ij}} < \alpha = \frac{s_{gh}}{-d_{gh}}$.

Proof. We'll first prove, by using mathematical induction, the first part of the theorem, *i.e.*, that if $s_{ij} < 0$ then $d_{ij} > 0$. During the first iteration, we don't have non basic arcs having negative reduced cost, since the algorithm starts from a dual feasible solution. Let $k + 1$ be the first iteration for which we have $s_{ij}^{(k+1)} < 0$ for an arc (i, j), while $s_{ij}^{(k)} \geq 0$. According to Table 1, in order to have a negative value, it has to be $s_{ij}^{(k+1)} = s_{ij}^{(k)} - s_{gh}^{(k)}$. It is as follows:

$$s_{ij}^{(k+1)} < 0 \Rightarrow s_{ij}^{(k)} < s_{gh}^{(k)}. \tag{4.1}$$

In this case, from relation (4.1), it follows that $s_{gh}^{(k)} > 0$ (since $s_{ij}^{(k)} \geq 0$). If $d_{ij}^{(k)} \geq 0$, then it is obviously $d_{ij}^{(k+1)} = d_{ij}^{(k)} - d_{gh}^{(k)} > 0$ since $d_{gh}^{(k)} < 0$. If, on the other hand, it is $d_{ij}^{(k)} < 0$ then from Relation (3.6) we have:

$$\alpha = \frac{s_{gh}^{(k)}}{-d_{gh}^{(k)}} \leq \frac{s_{ij}^{(k)}}{-d_{ij}^{(k)}}. \tag{4.2}$$

Relation (4.2) by using (4.1) becomes:

$$\frac{s_{gh}^{(k)}}{-d_{gh}^{(k)}} < \frac{s_{gh}^{(k)}}{-d_{ij}^{(k)}} \overset{\times \frac{d_{gh}^{(k)} d_{ij}^{(k)}}{s_{gh}^{(k)}} > 0}{\Rightarrow} -d_{ij}^{(k)} < -d_{gh}^{(k)} \Rightarrow d_{ij}^{(k)} - d_{gh}^{(k)} > 0 \Rightarrow d_{ij}^{(k+1)} > 0.$$

Assume that the theorem holds for iteration t. For iteration $t + 1$, according to Figure 3 and Table 1, we have the following cases:

- *Case 1*: obviously the theorem holds since $d_{ij}^{(t+1)} = d_{ij}^{(t)}$ and $s_{ij}^{(t+1)} = s_{ij}^{(t)}$.
- *Cases 2 and 5*: It is $s_{ij}^{(t+1)} = s_{ij}^{(t)} + s_{gh}^{(t)} < 0 \Rightarrow s_{ij}^{(t)} < -s_{gh}^{(t)} \overset{s_{gh}^{(t)} \geq 0}{\Rightarrow} s_{ij}^{(t)} < 0$. Thus, due to the assumption, it is $d_{ij}^{(t)} > 0$ and:

$$\frac{s_{ij}^{(t)}}{-d_{ij}^{(t)}} < \frac{s_{gh}^{(t)}}{-d_{gh}^{(t)}}, \tag{4.3}$$

it is also:

$$s_{ij}^{(t+1)} = s_{ij}^{(t)} + s_{gh}^{(t)} < 0 \Rightarrow s_{gh}^{(t)} < -s_{ij}^{(t)}. \tag{4.4}$$

Relation (4.3) using (4.4) becomes:

$$\frac{s_{ij}^{(t)}}{-d_{ij}^{(t)}} < \frac{-s_{ij}^{(t)}}{-d_{gh}^{(t)}} \overset{\times \frac{d_{gh}^{(t)} d_{ij}^{(t)}}{s_{ij}^{(t)}} > 0}{\Rightarrow} -d_{gh}^{(t)} < d_{ij}^{(t)} \Rightarrow d_{gh}^{(t)} + d_{ij}^{(t)} > 0 \Rightarrow d_{ij}^{(t+1)} > 0.$$

- *Cases 3 and 4*:
  - if $d_{ij}^{(t)} \geq 0$, then it is $d_{ij}^{(t+1)} = d_{ij}^{(t)} - d_{gh}^{(t)} > 0$, (since $d_{gh}^{(t)} < 0$);
  - if $d_{ij}^{(t)} < 0$, then due to the assumption, it holds $s_{ij}^{(t)} \geq 0$. It is $s_{ij}^{(t+1)} < 0 \Rightarrow s_{ij}^{(t)} - s_{gh}^{(t)} < 0 \Rightarrow s_{gh}^{(t)} > s_{ij}^{(t)} \Rightarrow s_{gh}^{(t)} > 0$. Thus:

$$\frac{s_{gh}^{(t)}}{-d_{gh}^{(t)}} \leq \frac{s_{ij}^{(t)}}{-d_{ij}^{(t)}} \overset{s_{ij}^{(t)} < s_{gh}^{(t)}}{\Rightarrow} \frac{s_{gh}^{(t)}}{-d_{gh}^{(t)}} < \frac{s_{gh}^{(t)}}{-d_{ij}^{(t)}} \overset{\times \frac{d_{gh}^{(t)} d_{ij}^{(t)}}{s_{gh}^{(t)}} > 0}{\Rightarrow} -d_{ij}^{(t)} < -d_{gh}^{(t)} \Rightarrow d_{ij}^{(t)} - d_{gh}^{(t)} > 0 \Rightarrow d_{ij}^{(t+1)} > 0.$$

We'll prove now the second part of the theorem by using again mathematical induction. We first prove that it is $\frac{s_{ij}^{(t+1)}}{-d_{ij}^{(t+1)}} < \alpha^{(t)}$ and after that it is $\alpha^{(t)} \leq \alpha^{(t+1)}$. Assume that this is the case for iteration t and we will show the same for iteration $t+1$. For the first case of Figure 3 it is obviously $\frac{s_{ij}^{(t+1)}}{-d_{ij}^{(t+1)}} < \alpha^{(t)}$. For the rest of the cases it is:

$$\frac{s_{ij}^{(t)}}{-d_{ij}^{(t)}} < \frac{s_{gh}^{(t)}}{-d_{gh}^{(t)}} \overset{\times d_{ij}^{(t)} > 0}{\Rightarrow} -s_{ij}^{(t)} < \frac{-s_{gh}^{(t)} d_{ij}^{(t)}}{d_{gh}^{(t)}}. \tag{4.5}$$

We also have:

$$\frac{s_{ij}^{(t+1)}}{-d_{ij}^{(t+1)}} = \frac{s_{ij}^{(t)} \pm s_{gh}^{(t)}}{-(d_{ij}^{(t)} \pm d_{gh}^{(t)})} = \frac{-s_{ij}^{(t)}}{d_{ij}^{(t)} \pm d_{gh}^{(t)}} \pm \frac{s_{gh}^{(t)}}{-(d_{ij}^{(t)} \pm d_{gh}^{(t)})}. \tag{4.6}$$

Relation (4.6) using (4.5) becomes:

$$\frac{s_{ij}^{(t+1)}}{-d_{ij}^{(t+1)}} < \frac{\frac{-s_{gh}^{(t)} d_{ij}^{(t)}}{d_{gh}^{(t)}}}{d_{ij}^{(t)} \pm d_{gh}^{(t)}} \pm \frac{s_{gh}^{(t)}}{-(d_{ij}^{(t)} \pm d_{gh}^{(t)})} = \frac{s_{gh}^{(t)} d_{ij}^{(t)}}{-d_{gh}^{(t)}(d_{ij}^{(t)} \pm d_{gh}^{(t)})} \pm \frac{s_{gh}^{(t)} d_{gh}^{(t)}}{-d_{gh}^{(t)}(d_{ij}^{(t)} \pm d_{gh}^{(t)})}$$

$$= \frac{s_{gh}^{(t)}(d_{ij}^{(t)} \pm d_{gh}^{(t)})}{-d_{gh}^{(t)}(d_{ij}^{(t)} \pm d_{gh}^{(t)})} = -\frac{s_{gh}^{(t)}}{d_{gh}^{(t)}} = \alpha^{(t)}.$$

We will now show that $\alpha^{(t)} \leq \alpha^{(t+1)}$. Assume this is true for iteration t. For the first case of Figure 3, it is obviously true. For the rest of the cases, it is as follows:

$$\frac{s_{ij}^{(t)}}{-d_{ij}^{(t)}} \geq \alpha^{(t)} = \frac{s_{gh}^{(t)}}{-d_{gh}^{(t)}} \overset{\times(-d_{ij}^{(t)}) > 0}{\Longrightarrow} s_{ij}^{(t)} \geq \frac{d_{ij}^{(t)} s_{gh}^{(t)}}{d_{gh}^{(t)}}. \tag{4.7}$$

It is also:

$$\frac{s_{ij}^{(t+1)}}{-d_{ij}^{(t+1)}} = \frac{s_{ij}^{(t)} \pm s_{gh}^{(t)}}{-(d_{ij}^{(t)} \pm d_{gh}^{(t)})}. \tag{4.8}$$

From (4.7) and (4.8) we take:

$$\frac{s_{ij}^{(t+1)}}{-d_{ij}^{(t+1)}} \geq \frac{d_{ij}^{(t)} \frac{s_{gh}^{(t)}}{d_{gh}^{(t)}} \pm s_{gh}^{(t)}}{-(d_{ij}^{(t)} \pm d_{gh}^{(t)})} = \frac{s_{gh}^{(t)}(d_{ij}^{(t)} \pm d_{gh}^{(t)})}{-d_{gh}^{(t)}(d_{ij}^{(t)} \pm d_{gh}^{(t)})} = \frac{s_{gh}^{(t)}}{-d_{gh}^{(t)}} = \alpha^{(t)}.$$

We proved that, for every case, $\alpha^{(t)} \leq \alpha^{(t+1)}$ and therefore, $\frac{s_{ij}^{(t+1)}}{-d_{ij}^{(t+1)}} < \alpha^{(t+1)}$. □

The next theorem proves that the algorithm keeps in touch with the dual feasible region by maintaining direction vector d. We prove that $s_{ij} + \alpha\, d_{ij} \geq 0$, for every arc (i, j).

Theorem 4.5. *Solution $y = s + \alpha d$ is dual feasible during all iterations of the algorithm.*

Proof. If $s_{ij} < 0$ then according to Theorem 4.4 it is $d_{ij} > 0$ and $\frac{s_{ij}}{-d_{ij}} < \alpha$. Therefore, $s_{ij} + \alpha d_{ij} > 0$. If $s_{ij} \geq 0$ and $d_{ij} < 0$ then $\frac{s_{ij}}{-d_{ij}} \geq \alpha$ (relation (3.6)). Therefore, it is again $s_{ij} + \alpha d_{ij} \geq 0$. Finally, if $s_{ij} \geq 0$ and $d_{ij} \geq 0$ then obviously $s_{ij} + \alpha d_{ij} \geq 0$. □

The next theorem examines the case where the problem is infeasible.

Theorem 4.6. *If $J_- = \emptyset$ and $I_- \neq \emptyset$, then the problem is infeasible.*

Proof. As it was shown in Theorem 4.5, $y = s + \alpha d$ is always dual feasible. Therefore, it satisfies the restrictions of the dual problem as it is described in matrix format in formula (2.4). So, we have:

$$A^T w + I_m(s + \alpha d) = c.$$

We denote as A_B^T the matrix formed by the rows of matrix A^T that correspond to the basic variables. Vectors c_B, s_B, and d_B are formed in a similar way. It is as follows:

$$A_B^T w + (s_B + \alpha d_B) = c_B.$$

By multiplying both parts of the above equation by $b^T(A_B^T)^{-1}$ we take:

$$b^T w = b^T(A_B^T)^{-1} c_B - b^T(A_B^T)^{-1}(s_B + \alpha d_B). \tag{4.9}$$

For the basic solution x_B it is:

$$A_B x_B = b \Rightarrow x_B^T = b^T (A_B^T)^{-1},$$

so, by formula (4.9) we have:

$$b^T w = x_B^T c_B - x_B^T (s_B + \alpha d_B) = x_B^T c_B - \alpha x_B^T d_B,$$

because $s_B = 0$. If we denote z and z' the value of the objective function of the primal and the dual problem respectively, the last equation becomes:

$$z' = z - \alpha x_B^T d_B. \tag{4.10}$$

We have $d_{ij} = 1$ for the negative flows and $d_{ij} = 0$ for the non-negative flows (formula (3.4)). There is at least one negative flow because $I_- \neq \emptyset$. Therefore, $x_B d_B < 0$ and it can be seen in formula (4.10) that the objective function of the dual problem is unbounded because it increases as far as the value of α increases. The fact that the dual problem is unbounded, means that the primal problem is infeasible. □

The last theorem proves that the algorithm has reached an optimal solution when $I_- = \emptyset$.

Theorem 4.7. *If $I_- = \emptyset$ then the current solution is optimal.*

Proof. It is obvious from (3.4) that $-|I_-| \leq d_{ij} \leq |I_-|$, where notation $|I_-|$ means the cardinality of set I_-. If $I_- = \emptyset$, then $|I_-| = 0$ and therefore $d_{ij} = 0, \forall (i,j) \in A$. According to Theorem 4.5 it is $y = s + \alpha d \geq 0$, so $s \geq 0$. In tandem, $x \geq 0$ since $I_- = \emptyset$. The current tree-solution is both primal and dual feasible and therefore, it is optimal. □

5. Implementation of DNEPSA and computational results

In order to evaluate the performance of DNEPSA, we performed an experimental comparison of DNEPSA against the classic DNSA. In this section we report the numerical tests for both algorithms. These tests demonstrate the exterior point algorithms efficiency on randomly generated MCNFP instances. The MCNF problem instances were created using the well-known NETGEN generator [22]. We ran the experiments on an Intel Pentium 4, running Ubuntu 9.10 "Karmic Koala" version at 3.6 GHz processor, and 2 GB RAM DDR 2 400Mhz with the -O3 (fully optimized for speed) option. The competitive algorithms were implemented in C and compiled with the gcc compiler. The functions used for the implementation of the algorithms have been written following the same programming techniques adjusted to the special characteristics of each algorithm. We implemented the augmented thread index method (ATI method), due to Glover *et al.* [15], for both

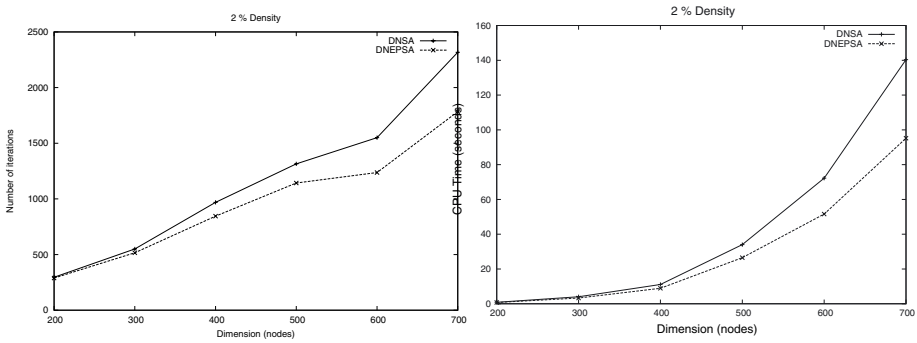

FIGURE 4. Comparative computational results of DNEPSA and DNSA for problem instances of density 2%.

algorithms. This method was chosen because it allows the fast update of the basic tree and it can also easily identify the cycle created with the addition of the entering arc. The two algorithms need an initial dual feasible solution to start from. At every execution, the same starting point was used for both algorithms. The time needed in order to find that initial solution, was included in the measurements.

Based on preliminary computational results on random generated problems [13], DNEPSA proved to be superior to DNSA for networks of density 20%. This superiority was shown on both, the number of the iterations and on the time needed in order to find an optimal solution. This section presents a more detailed computational study, in order to estimate the efficieny of DNEPSA to both dense and sparse problem instances. More specifically five classes of instances were developed; one sparse class and four dense classes. The densities are 2%, 10%, 20%, 30%, and 40% respectively for each class. Each class consists of six problem categories, with varying dimensions. The number of the nodes in each class, starts from 200 and is up to 700, with step equal to 100, (so this way the six, previously mentioned, categories are built). 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 cpu time. To conclude with, 300 MCNFP instances have been created and solved. The comparative computational results and the normalized comparative computational results, of DNEPSA and DNSA, on a number of random problem instances produced by NETGEN are presented in Tables 2 and 3 respectively.

Figures 4–8 demonstrate the performance of DNEPSA compared to the performance of the DNSA. The average numbers of iterations (*niter*) and the average numbers of cpu time (*cpu*) are depicted at the left and right part of each figure respectively. All the figures have been created with gnuplot 4.2. These average numbers come up from the instances which were solved in each category of the five classes.

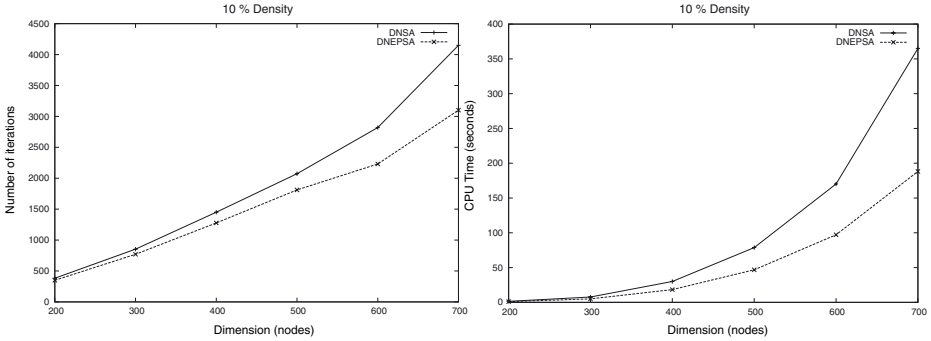

FIGURE 5. Comparative computational results of DNEPSA and DNSA for problem instances of density 10%.

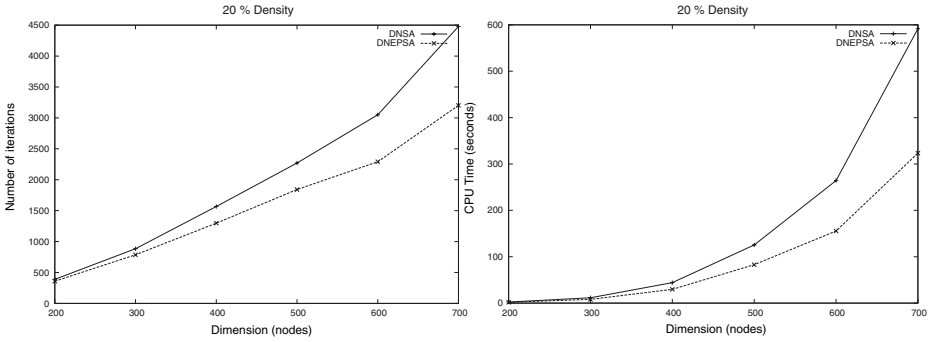

FIGURE 6. Comparative computational results of DNEPSA and DNSA for problem instances of density 20%.

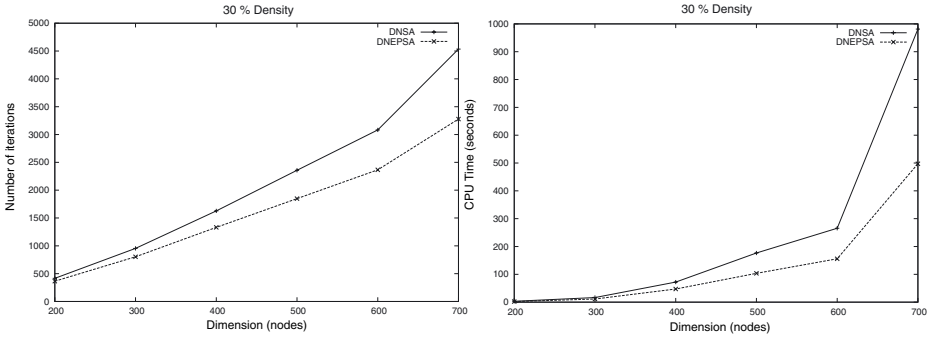

FIGURE 7. Comparative computational results of DNEPSA and DNSA for problem instances of density 30%.

TABLE 2. Iterations and *cpu* time (s) averages for random generated instances.

		DNSA		DNEPSA	
Density	Nodes × Arcs	*niter*	*cpu*	*niter*	*cpu*
2%	200 × 796	296	0.85	289	0.71
	300 × 1794	550	4.02	516	3.31
	400 × 3192	969	11.10	845	8.90
	500 × 4990	1315	34.02	1143	26.50
	600 × 7188	1550	72.20	1237	51.60
	700 × 9786	2317	140.38	1784	95.20
10%	200 × 3980	383	1.43	350	1.01
	300 × 8970	853	7.70	770	5.01
	400 × 15 960	1451	30.02	1277	18.30
	500 × 24 950	2073	78.69	1811	46.81
	600 × 35 940	2819	170.20	2231	97.10
	700 × 48 930	4149	365.00	3101	188.21
20%	200 × 7960	389	2.34	359	1.71
	300 × 17 940	884	11.50	786	8.02
	400 × 31 920	1567	44.20	1298	29.60
	500 × 49 900	2271	125.47	1841	82.90
	600 × 71 880	3050	264.20	2291	155.40
	700 × 97 860	4478	591.64	3202	323.23
30%	200 × 11 940	416	3.13	364	2.16
	300 × 26 910	954	16.50	803	11.20
	400 × 47 880	1628	72.15	1331	47.32
	500 × 74 850	2357	176.60	1848	103.63
	600 × 107 820	3083	265.52	2366	155.91
	700 × 146 790	4535	981.35	3278	497.33
40%	200 × 15 920	454	4.59	377	3.06
	300 × 35 880	1049	18.15	864	11.76
	400 × 63 840	1791	79.37	1455	51.05
	500 × 99 800	2512	199.26	1913	110.10
	600 × 143 760	3398	312.25	2580	167.50
	700 × 195 720	4943	1069.67	3473	552.08

A theoretical explanation of DNEPSA's superiority against the classic DNSA, is the fact that DNEPSA can cross over the infeasible region of the dual problem and return back to it by finding an optimal solution. This behavior can lead to an essential reduction on the number of iterations. In terms of linear programming, DNEPSA computes the direction towards to the dual feasible region and, iteration by iteration, it gets closer to the primal feasible region by reducing the solution's infeasibility. The comparative study of DNEPSA and DNSA algorithms shows

TABLE 3. Normalized iterations and *cpu* time (s) averages for random generated instances.

		DNSA		DNEPSA	
Density	Nodes × Arcs	*niter*	*cpu*	*niter*	*cpu*
2%	200 × 796	1.02	1.20	1	1
	300 × 1794	1.07	1.21	1	1
	400 × 3192	1.15	1.25	1	1
	500 × 4990	1.15	1.28	1	1
	600 × 7188	1.25	1.40	1	1
	700 × 9786	1.30	1.47	1	1
10%	200 × 3980	1.09	1.42	1	1
	300 × 8970	1.11	1.54	1	1
	400 × 15 960	1.14	1.64	1	1
	500 × 24 950	1.14	1.68	1	1
	600 × 35 940	1.26	1.75	1	1
	700 × 48 930	1.34	1.94	1	1
20%	200 × 7960	1.08	1.37	1	1
	300 × 17 940	1.13	1.43	1	1
	400 × 31 920	1.21	1.49	1	1
	500 × 49 900	1.23	1.51	1	1
	600 × 71 880	1.33	1.70	1	1
	700 × 97 860	1.40	1.83	1	1
30%	200 × 11 940	1.14	1.45	1	1
	300 × 26 910	1.19	1.47	1	1
	400 × 47 880	1.22	1.52	1	1
	500 × 74 850	1.28	1.70	1	1
	600 × 107 820	1.30	1.70	1	1
	700 × 146 790	1.38	1.97	1	1
40%	200 × 15 920	1.20	1.50	1	1
	300 × 35 880	1.21	1.54	1	1
	400 × 63 840	1.23	1.55	1	1
	500 × 99 800	1.31	1.81	1	1
	600 × 143 760	1.32	1.86	1	1
	700 × 195 720	1.42	1.94	1	1

that, for the instances considered, the cpu time and the number of iterations for DNEPSA algorithm are lesser than the same numbers for the DNSA.

6. Statistical analysis of the performance evaluation

In order to gain insight into the performance evaluation a statistical analysis is needed, as also shown in [7,24,25]. The results of this statistical analysis were based

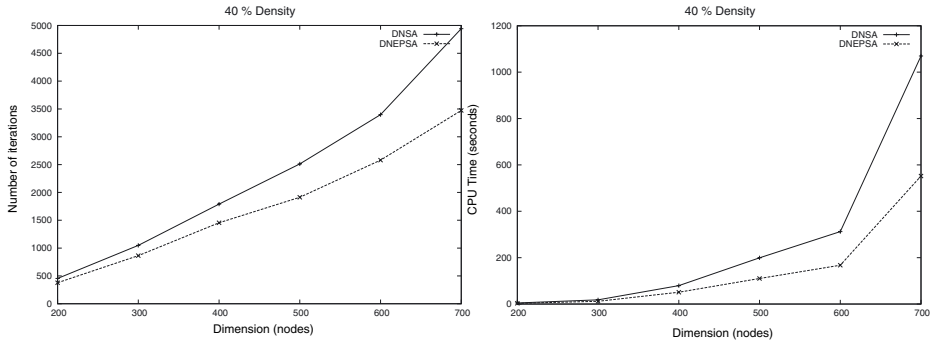

FIGURE 8. Comparative computational results of DNEPSA and DNSA for problem instances of density 40%.

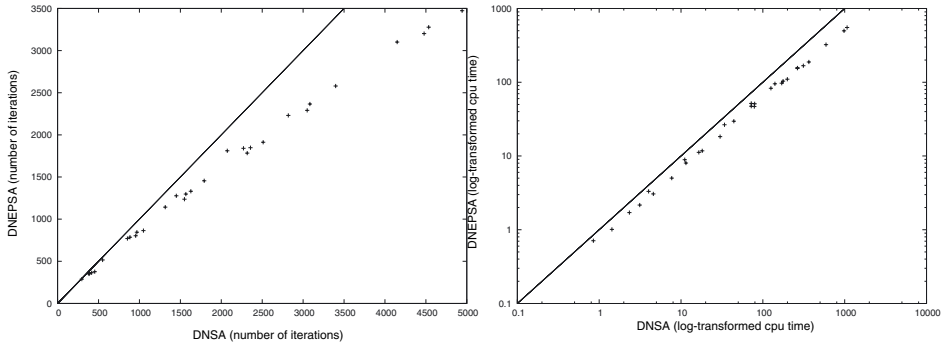

FIGURE 9. Scatterplots of DNEPSA *vs.* DNSA.

on IBM PASW Statistics v.19, and are presented in this Section in order to improve and strengthen our experimental results. Typically, an increase or decrease in the running time leads to an increase or decrease in the variance respectively. This can be attributed to the fact that, the running times are bounded below at zero. Therefore, a log transformation of the running times usually is preferred. Figure 9 depicts two scatterplots with a 45° line for reference; the left one showing the number of iterations while the right one showing the solution times on a double-logarithmic scale. Both of them exhibit a linear trend. The majority of the points lie below the 45° line, thus indicating that DNSA is generally slower.

However, in order to draw a valid statistical conclusion about the differences between the solution times or the number of iterations, a hypothesis testing is needed. To accomplish this, a decision must be made regarding the use of parametric or non-parametric statistical hypothesis test. We denote by d_{niter} the vector of pairwise differences in number of iterations between DNSA and DNEPSA.

Thus, $d_{\text{niter}} = niter_{\text{DNSA}} - niter_{\text{DNEPSA}}$, where $niter_{\text{DNSA}}$ and $niter_{\text{DNEPSA}}$ correspond to the vectors of DNSA and DNEPSA number of iterations respectively (30 values taken from Tab. 2). In a similar way, we denote by d_{cpu} the vector of pairwise differences in solution times between DNSA and DNEPSA. Thus, $d_{\text{cpu}} = t_{\text{DNSA}} - t_{\text{DNEPSA}}$, where t_{DNSA} and t_{DNEPSA} correspond to the vectors of DNSA and DNEPSA solution times respectively (30 values taken from Tab. 2).

By applying a one sample Kolmogorov-Smirnov test to the sample of d_{niter}, we take a p-value equal to 0.260 which is greater than 0.05. Therefore, there is no reason to doubt the distribution of d_{niter} is normal and we can safely proceed to a paired-sample t-test. On the contrary, by applying again a one sample Kolmogorov-Smirnov test to the sample of d_{cpu}, we take a p-value equal to 0.018 which is less than 0.05. Therefore, there is sufficient evidence to reject the normality assumption of the distribution of d_{cpu} and thus we should proceed to a Wilcoxon matched-pairs signed-ranks test.

In the first case, the paired-sample t-test is actually a test on the differences between the number of iterations between DNSA and DNEPSA. If we denote by M the population mean of pairwise differences, then $M = 0$ indicates that on randomly generated problem instances the experimental performance of DNSA and DNEPSA is about the same. However, $M > 0$ implies that DNSA is likely to need more iterations whereas $M < 0$ implies that DNEPSA needs more iterations. Since, we have no a priori reason to consider either algorithm is doing less iterations, we will test the hypothesis that $H_0 : M = 0$ *versus* $H_1 : M \neq 0$. The mean difference in number of iterations (Mean = 423.27, Standard Error = 74.90, $N = 30$) was significantly greater than zero, $t = 5.65$, two-tail $p < 0.05$, verifying the conclusion that the two algorithms perform differently. A 95% confidence interval about mean difference in number of iterations is (270, 576), indicating that the mean difference in number of iterations is between 270 iterations and 576 iterations. Therefore, we reject H_0 and conclude that the algorithms perform differently. Since the values of the pairwise differences are all positive, we conclude that DNEPSA needs a lesser number of iterations than DNSA.

In the second case, the Wilcoxon matched-pairs signed-ranks test is a distribution-free hypothesis test for the population median. The Wilcoxon signed rank statistic W_+ is based on the sizes of the absolute values of the differences between observations of solution times. If we denote by M the population median of pairwise differences, then $M = 0$ indicates that on a randomly generated problem instances the experimental performance of DNSA and DNEPSA is about the same. Since we have no a priori reason to consider either algorithm is faster, we will test the hypothesis that $H_0 : M = 0$ *versus* $H_1 : M \neq 0$. The median differences in solution times is significantly different from zero ($W_+ = 465$, $p < 0.05$) providing evidence that the two algorithms perform differently. Therefore, we reject H_0 and conclude that the algorithms perform differently. Since the values of the pairwise differences are all positive, we conclude that DNEPSA performs more quickly than DNSA. It should be noted that the differences between the the null hypothesis of the Wilcoxon matched-pairs signed-ranks test and the paired-sample t-test, is that

the median difference between pairs of observations, instead the mean difference between pairs, is zero.

7. Empirical complexity of DNEPSA using statistical analysis

Assuming an algorithm with expected running time $T(n,m) = O(g(n,m))$, where $g(n,m)$ is a function which the input length is parameterized by n, m, the estimated function $O(g(n,m))$ is usually referred to as the empirical complexity of the algorithm [7]. The empirical complexity of DNEPSA presented in this Section, was based on a statistical analysis carried out using IBM PASW Statistics v.19. The research approach includes a stepwise multiple regression analysis. The response variables are (i) the number of iterations and (ii) the cpu time, while the predictor variables are a large number of variables that are combinations of n and m (*e.g.*, log$n$, log$m$, n^2, n^3, m^2, nm, nm^2, etc.). A similar approach for the evaluation of the experimental performance of the classical Simplex algorithm for the linear programming problem was presented by Vanderbei in [30] as also in the papers [6, 8].

R-squared (R^2), is usually called the coefficient of determination and equals to the ratio of the sum of squares explained by our regression model and the total sum of squares around the mean. Furthermore, adjusted R-squared ($\bar{R}^2$) is a modification of R-squared that adjusts for the number of explanatory variables in a model. Unlike R-squared, the adjusted R-squared increases only if the new variable clearly improves the regression model (and not by chance).

The regression analysis, regarding the number of iterations, indicated the following regression equation (7.1), with an adjusted R-Squared value equal to $\bar{R}^2 = 97.5\%$ that is very nearly unity:

$$niter = a_1 + b_1 n\log m + c_1\sqrt{n} \tag{7.1}$$

where $a_1 = 775.991$, $b_1 = 1.45$, and $c_1 = -105.336$. Hence, about 97.5% of variation in the number of iterations can be explained by the variables $nlogm$ and $\sqrt{n}$. However, the function $g_1(n,m) = nlogm$ is the one having the larger order of growth. In a similar way, the regression analysis regarding the cpu time needed to solve each problem instance based on the problem dimension, indicated the following regression equation (7.2), with an adjusted R-Squared value $\bar{R}^2 = 97.4\%$ that is almost unity:

$$cpu = a_2 + b_2 mn^2 + c_2 nm + d_2\sqrt{nm} \tag{7.2}$$

where $a_2 = -16.871$, $b_2 = 1.959\times10^{-8}$, $c_2 = -1.208\times10^{-5}$, and $d_2 = 0.29$. Hence, about 97.4% of variation in the cpu time can be explained by the variables mn^2, nm, and $\sqrt{nm}$. In this case, the function $g_2(n,m) = mn^2$ have the larger order of growth.

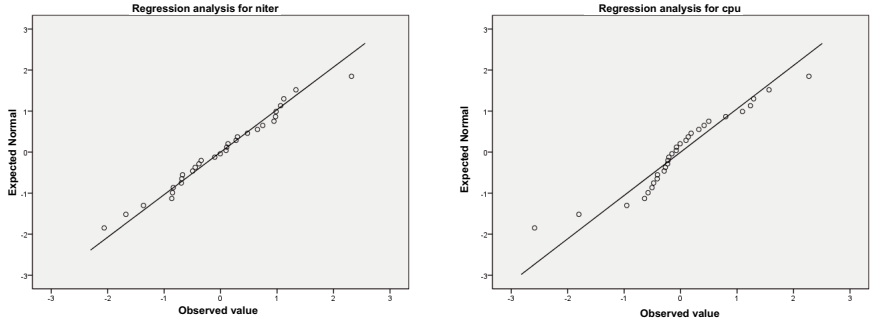

FIGURE 10. Normal Q-Q Plot of standardized residuals.

Moreover, the analysis of variance (ANOVA) resulted $p < 0.001$, showing an absolute linear correlation between the variables of each regression equation. The fit of both polynomials (7.1) and (7.2) were quite good at 5% level of significance. The left and right plot of Figure 10 depicts the normal probability plots of the standardized residuals regarding the number of iterations and the cpu time respectively. The standardized residuals of both regressions are normally distributed (Kolmogorov-Smirnov normality tests $p > 0.05$ in both cases).

Therefore, the regression analysis indicates that the DNEPSA algorithm requires $O(n\log m)$ number of iterations and $O(mn^2)$ cpu time. Thus, DNEPSA has a polynomial empirical computational behavior regarding the required number of iterations and the cpu time, observed statistically.

8. Conclusions and future work

The mathematical proof of correctness, a detailed comparative computational study of DNEPSA and DNSA on sparse and dense random problem instances, a statistical analysis of the experimental results, and finally some new results on the empirical complexity of DNEPSA were presented in this paper. A subject for future work is the improvement of the performance of the algorithm by using special data structures for storing and updating the necessary variables. Such data structures include Fibonacci heaps [10] and dynamic trees [18, 29]. The data structures used in an algorithm, can greatly affect its performance. It would be very interesting to use such data structures and compare DNEPSA's performance against some state-of-the-art algorithms. Such state-of-the-art algorithms include RELAX IV [5], combinatorial code CS2 [17], interior-point code DLNET [28], RNET [19], and NETFLO [21]. The algorithm's behavior has also to be examined in some well-known pathological instances, as described in [31, 32].

Furthermore, it would be interesting to develop a capacitated version of DNEPSA, although it is possible for any capacitated network to be transformed

into an uncapacitated equivalent one by removing arc capacities. This technique is analytically described in [1]. The only drawback of this transformation is that, it increases the number of nodes in the network. However, in most cases, the original and transformed networks can be solved by algorithms of the same complexity. This is due to the reason that the transformed network possesses a special structure that permits us to design more efficient algorithms.

Moreover, statistical techniques were used in order to present for the first time some new experimental results on the empirical complexity of DNEPSA. The fit of both polynomials (7.1) and (7.2) were quite good at 5% level of significance. Furthermore, high adjusted R-Squared values equal to 97.5% and 97.4% for the estimation of the number of iterations and the total cpu time respectively, provide the required validity of our experimental results. However, it is well known that the statistical measures of an algorithm's complexity do not always tally with the mathematical counterpart. Thus, it is very interesting to also derive the computational complexity of DNEPSA with rigorous theoretical proofs.

Finally, it would be also interesting to develop a visualization software for the teaching of DNEPSA to students. Similar educational tools have been already developed for other network optimization algorithms, as in [3, 4].

Acknowledgements. The authors gratefully acknowledge the helpful suggestions of two anonymous reviewers. This work was partially supported by the Research Committee of the University of Macedonia, Economic and Social Sciences, Greece, under Grant 80217 for the advance of Basic Research.

References

[1] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network flows: theory, algorithms and applications*. Prentice Hall, Englewood Cliffs, NJ (1993).

[2] R.K. Ahuja, T.L. Magnanti, J.B. Orlin and M. Reddy, Applications of network optimization. *Handbooks of Operations Research and Management Science* **7** (1995) 1–83.

[3] D. Andreou, K. Paparrizos, N. Samaras and A. Sifaleras, Visualization of the network exterior primal simplex algorithm for the minimum cost network flow problem. *Oper. Res.* **7** (2007) 449–464.

[4] Th. Baloukas, K. Paparrizos and A. Sifaleras, An animated demonstration of the uncapacitated network simplex algorithm. *ITE* **10** (2009) 34–40.

[5] D.P. Bertsekas and P. Tseng, *RELAX-IV: A faster version of the RELAX code for solving minimum cost flow problems*. Technical Report, Massachusetts Institute of Technology, Laboratory for Information and Decision Systems (1994).

[6] S. Chakraborty and P.P. Choudhury, A statistical analysis of an algorithm's complexity. *Appl. Math. Lett.* **13** (2000) 121–126.

[7] M. Coffin and M.J. Saltzman, Statistical Analysis of computational tests of algorithms and heuristics. *INFORMS J. Comput.* **12** (2000) 24–44.

[8] C. Cotta and P. Moscato, A mixed evolutionary-statistical analysis of an algorithm's complexity. *Appl. Math. Lett.* **16** (2003) 41–47.

[9] T.R. Ervolina and S.T. McCormick, Two strongly polynomial cut canceling algorithms for minimum cost network flow. *Discr. Appl. Math.* **46** (1993) 133–165.

[10] M. Fredman and R. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM* **34** (1987) 596–615.

[11] K. Fukuda and T. Terlaky, Criss-cross methods: a fresh view on pivot algorithms. *Math. Program.* **79** (1997), 369–395.
[12] G. Geranis, K. Paparrizos and A. Sifaleras, A dual exterior point simplex type algorithm for the minimum cost network flow problem. *Yugosl. J. Oper. Res.* **19** (2009) 157–170.
[13] G. Geranis, K. Paparrizos and A. Sifaleras, On the Computational Behavior of a Dual Network Exterior Point Simplex Algorithm for the Minimum Cost Network Flow Problem, in *Proceedings of the International Network Optimization Conference (INOC 2009)*. Pisa, Italy (2009).
[14] F. Glover, D. Klingman and A. Napier, Basic dual feasible solutions for a class of generalized networks. *Oper. Res.* **20** (1972) 126–136.
[15] F. Glover, D. Karney and D. Klingman, The augmented predecessor index method for locating stepping stone paths and assigning dual prices in distribution problems. *Transp. Sci.* **6** (1972) 171–180.
[16] F. Glover, D. Klingman and N. Phillips, *Network models in optimization and their applications in practice*. Wiley Publications (1992).
[17] A.V. Goldberg, An Efficient Implementation of a scaling minimum-cost flow algorithm. *J. Algorithms* **22** (1997) 1–29.
[18] A. Goldberg, M. Grigoriadis and R.E. Tarjan, Use of dynamic trees in a network simplex algorithm for the maximum flow problem, *Math. Program.* **50** (1991) 277–290.
[19] M. Grigoriadis, An efficient implementation of the network simplex method. *Math. Program. Stud.* **26** (1984) 83–111.
[20] J. Hultz and D. Klingman, An advanced dual basic feasible solution for a class of capacitated generalized networks. *Oper. Res.* **24** (1976) 301–313.
[21] J. Kennington and R. Helgason, *Algorithms for network programming*. Wiley, New York (1980).
[22] D. Klingman, A. Napier and J. Stutz, NETGEN: a program for generating large scale capacitated assignment, transportation, and minimum cost flow networks. *Manag. Sci.* **20** (1974) 814–821.
[23] I. Maros, A practical anti-degeneracy row selection technique in network linear programming. *Ann. Oper. Res.* **47** (1993) 431–442.
[24] C.C. McGeoch, Toward an experimental method for algorithm simulation. *INFORMS J. Comput.* **8** (1996) 1–15.
[25] R. Nance, R. Moose and R. Foutz, A statistical technique for comparing heuristics: an example from capacity assignment strategies in computer network design. *Commun. ACM* **30** (1987) 430–442.
[26] J.B. Orlin, *Genuinely polynomial simplex and non-simplex algorithms for the minimum cost flow problem*. Technical Report No. 1615-84, Sloan School of Management, M.I.T., Cambridge, MA (1984).
[27] K. Paparrizos, N. Samaras and A. Sifaleras, An exterior simplex type algorithm for the minimum cost network flow problem. *Comput. Oper. Res.* **36** (2009) 1176–1190.
[28] M. Resende and G. Veiga, An efficient implementation of a network interior point method, in *Network flows and matching: first DIMACS implementation challenge* **12**, edited by D.S. Johnson and C.C. McGeoch. DIMACS series in discrete mathematics and theoretical computer science, American Mathematical Society, Providence, Rhode Island (1993) 299–348.
[29] R.E. Tarjan, Dynamic trees as search trees *via* Euler tours, applied to the network simplex algorithm. *Math. Program.* **78** (1997) 169–177.
[30] R. Vanderbei, *Linear programming: foundations and extensions*, 3rd edition. Springer, New York (2007).
[31] N. Zadeh, More pathological examples for network flow problems. *Math. Program.* **5** (1973) 217–224.
[32] N. Zadeh, A bad network problem for the simplex method and other minimum cost flow algorithms. *Math. Programm.* **5** (1973) 255–266.