Optimization
Vol. 00, No. 00, Month 200x, 1–15

RESEARCH ARTICLE

On the minimization of a class of generalized linear functions on a flow polytope

Riccardo Cambini and Claudio Sodini

(*Received 00 Month 200x; in final form 00 Month 200x*)

The aim of this paper is to propose a solution method for the minimization of a class of generalized linear functions on a flow polytope. The problems will be solved by means of a network algorithm, based on graph operations, which lies within the class of the so called "optimal level solutions" parametric methods. The use of the network structure of flow polytopes, allows to obtain good algorithm performances and small numerical errors. Results of a computational test are also provided.

Keywords: network flows, nonconvex programming, global optimization, optimal level solutions.

AMS Subject Classification: 90C35, 90C26, 90C31.

1. Introduction

In this paper the following class of generalized linear problems is considered:

$$P: \begin{cases} \inf f(x) = \phi(c^T x + c_0, d^T x + d_0) \\ x \in X \end{cases}$$

where $c, d \in \Re^m$, with c and d linearly independent, $c_0, d_0 \in \Re$ and $X \neq \emptyset$ is a *flow polytope*. The scalar function $\phi(y_1, y_2)$ is assumed to be continuous and to be strictly increasing with respect to variable y_1. This allows to solve in an unifying framework a large class of nonconvex flow problems, including (for example) d.c., fractional and multiplicative ones [8–12, 14, 16], which are known to be useful from an applicative point of view [1, 6, 13].

In the case of linear problems on flow polytopes the simplex algorithm can be adapted to the particular structure of the feasible region, thus obtaining the so called "network simplex algorithm" which computes directly on the graph all the data needed for the iterations. The aim of this paper is to propose a network algorithm, based on graph operations, which can solve the general class of problems P.

The solution method proposed in this paper lies within the class of the so called "optimal level solutions" methods (see [3–5, 7, 15]), parametric algorithms which find the optimum of the problem by determining the minima of particular subproblems. In particular, the optimal solutions of these subproblems are obtained by means of a sensitivity analysis which maintains the optimality conditions. The linear subproblems turn out to be independent of function $\phi(y_1, y_2)$, hence a unify-

Department of Economics and Management, University of Pisa, Via Cosimo Ridolfi 10, 56124 Pisa, ITALY, e-mail: cambric@ec.unipi.it, csodini@ec.unipi.it

ing method to solve all of the problems belonging to the general class of problems P can be stated.

In Section 2 some preliminary definitions are recalled; in Sections 3 and 4 it is shown how the optimal level solutions can be determined by means of network operations in the primal and the dual approaches, respectively; in Section 5 a solution algorithm is proposed and fully described, as well as some methods to improve its overall performance; finally, in Section 6 the results of an extensive computational test are provided and discussed.

2. Definitions and Preliminary results

Consider the following class of generalized linear programs:

$$\begin{cases} \inf f(x) = \phi(c^T x + c_0, d^T x + d_0) \\ x \in X \end{cases}$$

where $c, d \in \Re^m$, with c and d linearly independent, $c_0, d_0 \in \Re$ and $X \neq \emptyset$ is a flow polytope. The scalar function $\phi(y_1, y_2)$ is assumed to be continuous, strictly increasing with respect to variable y_1, and defined for all the values in Ω where:

$$\Omega = \{(y_1, y_2) \in \Re^2 : \ y_1 = c^T x + c_0, \ y_2 = d^T x + d_0, \ x \in X\}$$

We aim to determine a solution algorithm for problem P based on graph operations and to study its efficiency. This will allow to solve large dimensional problems with good time performances and small numerical errors.

Let $G = (N, A, l, u)$ be a weighted graph where N ($|N| = n$) is the set of *nodes* and A ($|A| = m$) is the set of *arcs*; with respect to the k-th arc $(i, j) \in A$ we have the lower and upper *capacities* $l_k \in \Re$ and $u_k \in \Re$, respectively. Hence, problem P results to be:

$$P : \begin{cases} \min f(x) = \phi(c^T x + c_0, d^T x + d_0) \\ x \in X = \{x \in \Re^m : \ Ex = r \ , \ l \leq x \leq u\} \end{cases}$$

where E is the incidence matrix of the graph G, $x = (x_k)$, $x_k \in \Re \ \forall k = 1, \ldots, m$, is the vector of the flows, $r = (r_i)$, $r \in \Re^n$ ($\sum_{i=1}^n r_i = 0$), is the vector of the requirements of the nodes. Since matrix $[E|r]$ has rank $n - 1$ it is useful to insert an auxiliary variable/arc x_0 corresponding to one of the nodes of the graph. In this light, we choose to add an auxiliary variable/arc corresponding to the first node, so that region X can be rewritten as:

$$X = \{x \in \Re^m : \ Ex + e^1 x_0 = r \ , \ l \leq x \leq u \ , \ 0 \leq x_0 \leq 0\}$$

where $e^1 = (1, 0, ..., 0)^T \in \Re^n$. Finally, notice that the k-th column of E, denoted with $E^k = (E_h^k)$, corresponding to the k-th arc $(i, j) \in A$ is given by:

$$E_h^k = \begin{cases} 1 & \text{if } h = i \\ -1 & \text{if } h = j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The flow polytope X results to be a compact convex set. The following notations

can then be introduced:

$$\xi_{min} = d_0 + \min_{x \in X} d^T x \quad \text{and} \quad \xi_{max} = d_0 + \max_{x \in X} d^T x$$

so that the set of all *feasible levels* can be defined as:

$$\Lambda = \left\{ \xi \in \Re : \ \xi = d^T x + d_0, \ x \in X \right\} = [\xi_{min}, \xi_{max}]$$

Given a feasible level $\xi \in \Lambda$ the following subset of X can be introduced:

$$X_\xi = \{ x \in X : \ d^T x + d_0 = \xi \}$$

which allows to introduce the following parametrical subproblem:

$$\begin{cases} \min f(x) = \phi(c^T x + c_0, \xi) \\ x \in X_\xi \end{cases}$$

Taking into account of the strict increaseness of the scalar function $\phi(y_1, y_2)$ with respect to variable y_1, this parametrical subproblem is equivalent to the following one:

$$P_\xi : \begin{cases} \min c^T x \\ x \in X_\xi \end{cases}$$

An optimal solution of problem P_ξ is called an *optimal level solution*. Given a feasible level $\xi \in \Re$, the set of optimal solutions of P_ξ is denoted with $S_\xi \subset X_\xi$, while the set of all the optimal level solutions is denoted with $S = \cup_{\xi \in \Lambda} S_\xi \subset X$. Notice that for each feasible level $\xi \in \Lambda$ there exists at least one optimal level solution in S_ξ which is a vertex or belongs to an edge of the polytope X. Obviously, an optimal solution of problem P is also an optimal level solution and, in particular, it is the optimal level solution with the smallest value; the idea of this approach is then to scan all the feasible levels, studying the corresponding optimal level solutions, until the minimizer of the problem is reached. Starting from an incumbent optimal level solution, this can be done by means of a sensitivity analysis on the parameter ξ, which allows us to move in the various steps through several optimal level solutions until the optimal solution is found (see [4]).

3. Network simplex primal approach

Problem P_ξ is nothing but a linear minimum cost flow problem with the additional linear constraint $d^T x + d_0 = \xi$. There is no need to recall the wide literature on the classical network simplex algorithm, where problems are solved by using the particular structure of graphs and trees (see for all [2]). Our aim is to propose a solution method which uses the network structure of the problem and implicitly manages the additional constraint $d^T x + d_0 = \xi$.

In this light, let us now introduce the following notations:

$$\tilde{E} = [e^1 | E] \ , \ \tilde{x}^T = [x_0, x^T] \ , \ \tilde{c}^T = [0, c^T] \ , \tilde{d}^T = [0, d^T] \ , \tilde{l}^T = [0, l^T] \ , \tilde{u}^T = [0, u^T].$$

Subproblem P_ξ can be rewritten as:

$$P_\xi : \begin{cases} \min \tilde{c}^T \tilde{x} \\ \tilde{x} \in \tilde{X}_\xi = \{\tilde{x} \in \Re^{n+1} : \ \tilde{E}\tilde{x} = r \ , \ \tilde{d}^T\tilde{x} = \xi - d_0 \ , \ \tilde{l} \leq \tilde{x} \leq \tilde{u}\} \end{cases}$$

The structure of a basic matrix of problem P_ξ is:

$$T_{B,k} = \begin{bmatrix} \tilde{E}_B & E^k \\ \tilde{d}_B^T & d_k \end{bmatrix}$$

where $\tilde{E}_B$ is a basic matrix of $\tilde{E}$, $\tilde{d}_B$ is a vector whose elements correspond to the columns of $\tilde{E}_B$. Clearly, for the sake of the nonsingularity of $\tilde{E}_B$ and $T_{B,k}$, the vector e^1 corresponding to the auxiliary variable x_0 is always the first column of $\tilde{E}_B$. It is worth noticing that $\tilde{E}_B$ is nothing but the matrix associated to a spanning tree. B is the ordered list of the n indexes of the basic variables corresponding to the spanning tree; as it has been already explained, the first element of B is always the index 0 corresponding to the auxiliary variable x_0. k is the index of the basic variable not included in the basic spanning tree, hence E^k is the k-th column of E and d_k is the k-th component of d. The subgraph associated to $T_{B,k}$ results to be a spanning tree (rooted in the first node) plus an additional arc; such a kind of structures are known as 1-*trees*.

It is known that in order to solve linear min cost problems by means of a simplex primal approach, it is necessary to compute the inverse of the basis matrix and the reduced costs. This can be implemented by means of network operations as described below.

1) the inverse of $\tilde{E}_B$ can be computed by means of a proper visit of the corresponding spanning tree:

   all the elements of the first row of $\tilde{E}_B^{-1}$ are equal to 1; the first column of $\tilde{E}_B^{-1}$ is e^1; the h-th column of $\tilde{E}_B^{-1}$, $h > 1$, can be easily computed by considering the unique path of the spanning tree connecting node h to the root (node 1); such a column contains values in the set $\{-1, 0, 1\}$, in particular the value 0 is assigned to the rows corresponding to the arcs not belonging to the considered path, the other values are assigned in order to obtain a flow balance equal to 1 in node h and equal to 0 in the others (such assignement can be done by visiting the path from the node h towards the root);

2) once the inverse of $\tilde{E}_B$ is known, then the inverse of $T_{B,k}$ is given by:

$$T_{B,k}^{-1} = \begin{bmatrix} \tilde{E}_B^{-1} + (1/\bar{d}_k) y^k \lambda_{Bd}^T & -(1/\bar{d}_k) y^k \\ -(1/\bar{d}_k)\lambda_{Bd}^T & 1/\bar{d}_k \end{bmatrix} \tag{2}$$

   where $\lambda_{Bd}^T = \tilde{d}_B^T \tilde{E}_B^{-1}$, $\bar{d}_k = d_k - \lambda_{Bd}^T E^k$, $y^k = \tilde{E}_B^{-1} E^k$;

3) being E^k the k-th column of E corresponding to the k-th arc $(i, j) \in A$, then for (1) y^k can be computed as the difference between the i-th column of $\tilde{E}_B^{-1}$ and its j-th column.

Let $L = \{h \notin B, \ h \neq k : x_h = l_h\}$ and $U = \{h \notin B, \ h \neq k : x_h = u_h\}$. Given:

$$\tilde{r} = r - \sum_{h \in L} E^h l_h - \sum_{h \in U} E^h u_h \ \ , \ \ \tilde{\xi} = \xi - d_0 - \sum_{h \in L} d_h l_h - \sum_{h \in U} d_h u_h$$

the solution $\tilde{x}$ associated to the basic matrix $T_{B,k}$ has $\tilde{x}_h = l_h \ \forall h \in L$, $\tilde{x}_h = u_h$

$\forall h \in U$, while:

$$\tilde{x}_{B,k} = T_{B,k}^{-1} \begin{bmatrix} \tilde{r} \\ \tilde{\xi} \end{bmatrix} = \begin{bmatrix} \tilde{E}_B^{-1}\tilde{r} + (1/\bar{d}_k)(\lambda_{Bd}^T\tilde{r} - \tilde{\xi})y^k \\ -(1/\bar{d}_k)(\lambda_{Bd}^T\tilde{r} - \tilde{\xi}) \end{bmatrix}$$

$$\tilde{y}^h = T_{B,k}^{-1} \begin{bmatrix} E^h \\ d_h \end{bmatrix} = \begin{bmatrix} y^h - (\bar{d}_h/\bar{d}_k)y^k \\ \bar{d}_h/\bar{d}_k \end{bmatrix}$$

$$c'_h = c_h - (\tilde{c}_B^T, c_k)\tilde{y}^h = \bar{c}_h - (\bar{c}_k/\bar{d}_k)\bar{d}_h \tag{3}$$

where $\lambda_{Bc}^T = \tilde{c}_B^T\tilde{E}_B^{-1}$, $y^h = \tilde{E}_B^{-1}E^h$, $\bar{c}_k = c_k - \lambda_{Bd}^T E^k$, $\bar{c}_h = c_h - \lambda_{Bd}^T E^h$.

The vectors $\tilde{y}^h$ and the reduced costs c'_h can be computed by means of $\tilde{E}_B^{-1}$. If $c'_h \geq 0\ \forall h \in L$ and $c'_h \leq 0\ \forall h \in U$ then the basic solution $\tilde{x}$ is an optimal solution.

Let $s \in L$ such that $c'_s < 0$; the variable x_s must be increased (if possible) of a positive value δ from l_s to u_s. The following conditions must be verified:

$$\tilde{l}_{B,k} \leq \tilde{x}_{B,k} - \tilde{y}^s\delta \leq \tilde{u}_{B,k}, x_s = l_s + \delta \leq u_s, 0 \leq \delta \leq u_s - l_s \tag{4}$$

The maximun value $\bar{\delta}$ for δ is obtained from (4) as follows:

$$\begin{cases} \delta_1 = \frac{\tilde{x}_{(B,k)r} - \tilde{u}_{(B,k)r}}{\tilde{y}_r^s} = \min\limits_{h \in (B,k)\ :\ \tilde{y}_h^s < 0} \left\{ \frac{\tilde{x}_{(B,k)h} - \tilde{u}_{(B,k)h}}{\tilde{y}_h^s} \right\} \\ \delta_2 = \frac{\tilde{x}_{(B,k)p} - \tilde{l}_{(B,k)p}}{\tilde{y}_p^s} = \min\limits_{h \in (B,k)\ :\ \tilde{y}_h^s > 0} \left\{ \frac{\tilde{x}_{(B,k)h} - \tilde{l}_{(B,k)h}}{\tilde{y}_h^s} \right\} \\ \bar{\delta} = min\{\delta_1, \delta_2, u_s - l_s\} \end{cases} \tag{5}$$

Let $s \in U$ such that $c'_s > 0$; the variable x_s must be decreased (if possible) of a positive value δ from u_s to l_s. The following conditions must be verified

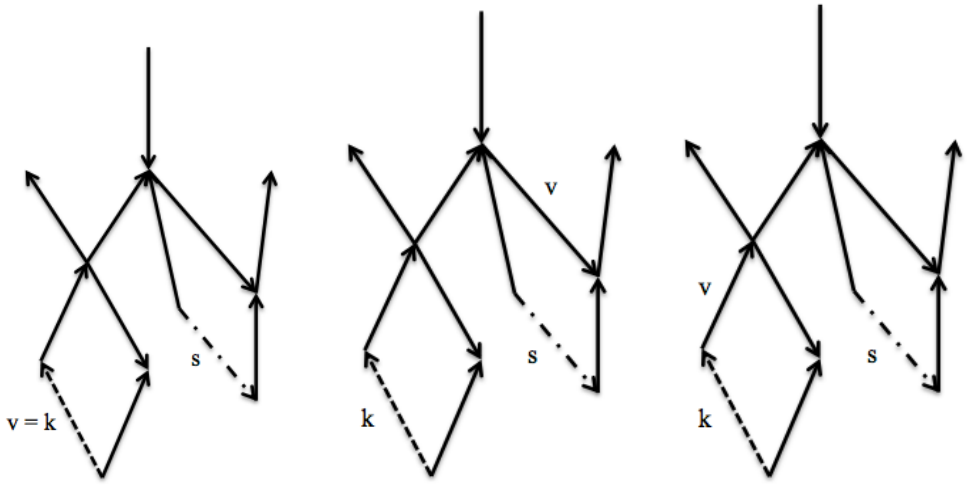
$$\tilde{l}_{B,k} \leq \tilde{x}_{B,k} + \tilde{y}^s\delta \leq \tilde{u}_{B,k}, x_s = u_s - \delta \geq l_s, 0 \leq \delta \leq u_s - l_s \tag{6}$$

The maximun value $\bar{\delta}$ for δ is obtained from (6) as follows:

$$\begin{cases} \delta_1 = \frac{\tilde{u}_{(B,k)r} - \tilde{x}_{(B,k)r}}{\tilde{y}_r^s} = \min\limits_{h \in (B,k)\ :\ \tilde{y}_h^s > 0} \left\{ \frac{\tilde{u}_{(B,k)h} - \tilde{x}_{(B,k)h}}{\tilde{y}_h^s} \right\} \\ \delta_2 = \frac{\tilde{l}_{(B,k)p} - \tilde{x}_{(B,k)p}}{\tilde{y}_p^s} = \min\limits_{h \in (B,k)\ :\ \tilde{y}_h^s < 0} \left\{ \frac{\tilde{l}_{(B,k)h} - \tilde{x}_{(B,k)h}}{\tilde{y}_h^s} \right\} \\ \bar{\delta} = min\{\delta_1, \delta_2, u_s - l_s\} \end{cases} \tag{7}$$

If $\bar{\delta} = u_s - l_s$ the variable x_s goes from L to U (or from U to L) and the basic matrix $T_{B,k}$ is unchanged. Otherwise, the non basic variable x_s enters into the basis and by (5) or (7) the basic variable x_v leaving the basis is detected, where:

$$v = \begin{cases} B_r & \text{if } \bar{\delta} = \delta_1 \\ B_p & \text{if } \bar{\delta} = \delta_2 \end{cases}$$



(a) Case $v = k$ (b) Case $v \neq k$ with s providing a spanning tree (c) Case $v \neq k$ with s not providing a spanning tree

Figure 1. How the one-tree structure can be maintained

Three cases can occur:

- **case** $v = k$. The additional arc k leaves the 1-tree, the spanning tree is untouched and the entering arc s becomes the new additional arc (see Figure 1.a). The basic matrix results:

$$T_{B,s} = \begin{bmatrix} \tilde{E}_B & E^s \\ \tilde{d}_B^T & d_s \end{bmatrix}$$

- **case** $v \neq k$ and vectors E^h with $h \in \{s\} \cup (B \setminus \{v\})$ are linearly independent. The entering arc s provides a spanning tree, arc k remains the additional one and s substitutes the leaving arc v in the spanning tree (see Figure 1.b). The basic matrix results:

$$T_{B',k} = \begin{bmatrix} \tilde{E}_{B'} & E^k \\ \tilde{d}_{B'}^T & d_k \end{bmatrix}$$

  where B' is obtained from B by substituting v with s, $\tilde{E}_{B'}$ is the new basic matrix of $\tilde{E}$ where column s substitutes column v, $\tilde{d}_{B'}$ is a subvector of $\tilde{d}$ whose elements correspond to the columns of $\tilde{E}_{B'}$;
- **case** $v \neq k$ and vectors E^h with $h \in \{s\} \cup (B \setminus \{v\})$ are linearly dependent. The entering arc s does not provide a spanning tree, s becomes the new additional arc and arc k substitutes v in the spanning tree (see Figure 1.c). The basic matrix results:

$$T_{B',s} = \begin{bmatrix} \tilde{E}_{B'} & E^s \\ \tilde{d}_{B'}^T & d_s \end{bmatrix}$$

  where B' is obtained from B by substituting v with k, $\tilde{E}_{B'}$ is the new basic matrix of $\tilde{E}$ where column k substitutes column v, $\tilde{d}_{B'}$ is a subvector of $\tilde{d}$ whose elements correspond to the columns of $\tilde{E}_{B'}$.

If $\tilde{E}_B$ is changed, then the inverse of $\tilde{E}_{B'}$ must be computed. It is worth noticing that the inverse $\tilde{E}_{B'}^{-1}$ can be obtained by just updating $\tilde{E}_B^{-1}$.

Procedure **Primal**(*inputs*: a feasible basis (B,k,L,U), ξ, $\tilde{E}_B^{-1}$; *outputs*: *Opt*, *OptVal*)
Compute $\tilde{x}$, $c'_h\ \forall h \in L \cup U$ as in (3);
Let $N_L = \{h \in L : c'_h < 0\}$, $P_U = \{h \in U : c'_h > 0\}$;
while $N_L \cup P_U \neq \emptyset$;
Let $c'_s = min\{min_{h\in N_L}\{c'_h\}; min_{h\in P_U}\{-c'_h\}\}$;
Compute $\tilde{y}^s$ and determine $\bar{\delta}$ by (5) or (7) ;
Update B,k,L,U and compute (if necessary) $\tilde{E}_B^{-1}$ by (8) ;
Compute $\tilde{x}$, $c'_h\ \forall h \in L \cup U$ as in (3);
Update $N_L = \{h \in L : c'_h < 0\}$, $P_U = \{h \in U : c'_h > 0\}$;
end while;
$Opt := \tilde{x}$; $OptVal := \tilde{c}^T\tilde{x}$;
end proc.

Let us define the vector $y_{inv} = -y^s/y^s_r$, $y_{inv_r} = 1/y^s_r$ (case $\bar{\delta} = \delta_1$) or the vector $y_{inv} = -y^s/y^s_p$, $y_{inv_r} = 1/y^s_p$ (case $\bar{\delta} = \delta_2$), and let El^r (El^p) an identity matrix except for the column r-th (p-th) given by y_{inv}. The inverse of $\tilde{E}_{B'}$ is then:

$$\tilde{E}^{-1}_{B'} = \begin{cases} El^r\tilde{E}^{-1}_B & \text{if } \bar{\delta} = \delta_1 \\ El^p\tilde{E}^{-1}_B & \text{if } \bar{\delta} = \delta_2 \end{cases} \tag{8}$$

Procedure *"Primal()"* shows how the optimal solution of P_ξ can be stated starting from the feasible basic solution represented by the index partition B,k,L,U.

4. Network simplex dual approach

Let $\tilde{x}$ the basic solution associated to the basic matrix $T_{B,k}$. If $c'_h \geq 0\ \forall h \in L$ and $c'_h \leq 0\ \forall h \in U$ then the basic solution $\tilde{x}$ is a dual feasible solution. In particular, if $\tilde{x}$ is also a feasible solution then it is an optimal solution. If $\tilde{x}$ is unfeasible then the dual simplex algorithm can be used to find an optimal solution. In this section we propose a dual solution method which uses, as in the previous section, the network structure of the problem and implicitly manages the additional constraint $d^Tx + d_0 = \xi$. As it is known, in the dual simplex algorithm, first the leaving variable (from the basis) is chosen and then the entering variable (into the basis) is detected in order to maintain the optimality of the reduced costs. The leaving variable is chosen among the basic variables out of the bounds (i.e variables having a value smaller than the lower bound or higher than the upper bound).

Let $\tilde{x}_{(B,k)_v}$ be the v-th component of $\tilde{x}_{B,k}$ and suppose $\tilde{x}_{(B,k)_v} < \tilde{l}_{(B,k)_v}$. Our aim is to increase variable $x_{(B,k)_v}$ up to its lower bound $\tilde{l}_{(B,k)_v}$. In this way, variable $x_{(B,k)_v}$ leaves the basis and enters in L. In order to detect the entering variable, the v-th component of vector $\tilde{y}^h$ must be computed for all $h \in L \cup U$. Let α_v be the v-th row of $T^{-1}_{B,k}$, then the v-th component of vector $\tilde{y}^h$, $h \in L \cup U$, is obtained by

$$\tilde{y}^h_v = \alpha_v \begin{bmatrix} E^h \\ d_h \end{bmatrix} \tag{9}$$

If $\tilde{y}^h_v \geq 0\ \forall h \in L$ and $\tilde{y}^h_v \leq 0\ \forall h \in U$ then region X_ξ is empty; otherwise, the entering variable x_s is detected by

$$\frac{c'_s}{\tilde{y}^s_v} = \max\left\{ \max_{h\in L\,:\,\tilde{y}^h_v<0}\left\{\frac{c'_h}{\tilde{y}^h_v}\right\}, \max_{h\in U\,:\,\tilde{y}^h_v>0}\left\{\frac{c'_h}{\tilde{y}^h_v}\right\}\right\} \tag{10}$$

Let $\tilde{x}_{(B,k)_v}$ be the v-th component of $\tilde{x}_{B,k}$ and suppose $\tilde{x}_{(B,k)_v} > \tilde{u}_{(B,k)_v}$. Our aim is to decrease variable $x_{(B,k)_v}$ down to its upper bound $\tilde{u}_{(B,k)_v}$. In this way, variable $x_{(B,k)_v}$ leaves the basis and enters in U. Just as in the previous case, the v-th component of vector $\tilde{y}^h$ must be computed for all $h \in L \cup U$. If $\tilde{y}^h_v \leq 0\ \forall h \in L$ and $\tilde{y}^h_v \geq 0\ \forall h \in U$ then region X_ξ is empty; otherwise the entering variable x_s is detected by

$$\frac{c'_s}{\tilde{y}^s_v} = \min\left\{ \min_{h\in L\,:\,\tilde{y}^h_v>0}\left\{\frac{c'_h}{\tilde{y}^h_v}\right\}, \min_{h\in U\,:\,\tilde{y}^h_v<0}\left\{\frac{c'_h}{\tilde{y}^h_v}\right\}\right\} \tag{11}$$

The basic variable $x_{(B,k)_v}$ leaves the basis and by (10) or (11) the non basic variable x_s enters into the basis.

Three cases can occur:

- **case** $v = n + 1$. The spanning tree is untouched while the additional arc is substituted by the entering arc. The basic matrix results:

$$T_{B,s} = \begin{bmatrix} \tilde{E}_B & E^s \\ \tilde{d}^T_B & d_s \end{bmatrix}$$

- **case** $v \leq n$ and vectors E^h with $h \in \{s\} \cup (B \setminus \{B_v\})$ are linearly independent. The basic matrix results:

$$T_{B',k} = \begin{bmatrix} \tilde{E}_{B'} & E^k \\ \tilde{d}^T_{B'} & d_k \end{bmatrix}$$

  where B' is obtained from B by substituting B_v with s, $\tilde{E}_{B'}$ is the new basic matrix of $\tilde{E}$ where column s substitutes column B_v, $\tilde{d}_{B'}$ is a subvector of $\tilde{d}$ whose elements correspond to the columns of $\tilde{E}_{B'}$;
- **case** $v \leq n$ and vectors E^h with $h \in \{s\} \cup (B \setminus \{B_v\})$ are linearly dependent. The basic matrix results:

$$T_{B',s} = \begin{bmatrix} \tilde{E}_{B'} & E^s \\ \tilde{d}^T_{B'} & d_s \end{bmatrix}$$

  where B' is obtained from B by substituting B_v with k, $\tilde{E}_{B'}$ is the new basic matrix of $\tilde{E}$ where column k substitutes column B_v, $\tilde{d}_{B'}$ is a subvector of $\tilde{d}$ whose elements correspond to the columns of $\tilde{E}_{B'}$.

Procedure **Dual**(*inputs*: a dual basis (B,k,L,U), ξ, $\tilde{E}_B^{-1}$;
outputs: *Opt*, *OptVal*, (B,k,L,U), $\tilde{E}_B^{-1}$)
Compute $\tilde{x}$, c'_h $\forall h \in L \cup U$ as in (3);
Let $LT_L = \{r \in (B,k) : \tilde{x}_r < \tilde{l}_r\}$, $GT_U = \{r \in (B,k) : \tilde{x}_r > \tilde{u}_r\}$;
while $LT_L \cup GT_U \neq \emptyset$;
Let v the index of (B,k) such that:
$min\{min_{r \in LT_L}\{\tilde{x}_r - \tilde{l}_r\}; min_{r \in GT_U}\{\tilde{u}_r - \tilde{x}_r\}\}$;
Compute $\tilde{y}_v^h, \forall h \in L \cup U$ by (9);
if $v \in LT_L$ *then*
if $\tilde{y}_v^h \geq 0$ $\forall h \in L$ *and* $\tilde{y}_v^h \leq 0$ $\forall h \in U$
then $X_\xi = \emptyset$; $Opt := []$; $OptVal := []$; *stop*;
else Detect the entering variable x_s by (10);
end if ;
else
if $\tilde{y}_v^h \leq 0$ $\forall h \in L$ *and* $\tilde{y}_v^h \geq 0$ $\forall h \in U$
then $X_\xi = \emptyset$; $Opt := []$; $OptVal := []$; *stop*;
else Detect the entering variable x_s by (11);
end if ;
end if ;
Update B, k, L, U and compute (if necessary) $\tilde{E}_B^{-1}$ by (8) ;
Compute $\tilde{x}$, c'_h $\forall h \in L \cup U$ as in (3);
Update $LT_L = \{r \in (B,k) : \tilde{x}_r < \tilde{l}_r\}$, $GT_U = \{r \in (B,k) : \tilde{x}_r > \tilde{u}_r\}$;
end while;
$Opt := \tilde{x}$; $OptVal := \tilde{c}^T\tilde{x}$;
end proc.

5. Solution algorithm

As it has been introduced in Section 2, in order to find the global minimum we just have to visit the optimal level solutions for all of the feasible levels in $\Lambda = [\xi_{min}, \xi_{max}]$. In this light, we will show that it is possible to scan the feasible levels from ξ_{min} to ξ_{max} by means of a finite number of iterations corresponding to simplex pivot operations executed over the flow polytope.

In Section 3 it has been already described the structure of the optimal level solution corresponding to a certain feasible level ξ. Assuming problem $P_{\xi'}$ has been already solved and that its optimal solution is x' with optimal basis (B,k,L,U), our aim is to solve the parametric problem $P_{\xi'+\theta}$, $\theta \geq 0$, such that the corresponding optimal level solution have the same basis (B,k,L,U).

Subproblem $P_{\xi'+\theta}$ can be rewritten as:

$$P_{\xi'+\theta} : \begin{cases} \min \tilde{c}^T\tilde{x} \\ \tilde{E}\tilde{x} = r \\ \tilde{d}^T\tilde{x} = \xi' + \theta - d_0 \\ \tilde{l} \leq \tilde{x} \leq \tilde{u} \\ \tilde{x} \in \Re^{n+1} \end{cases}$$

It can be easily verified that the reduced costs for $P_{\xi'+\theta}$ can still be computed with (3) and does not depend on θ, just like x'_h for $h \in L \cup U$ (it is $x'_h(\theta) = l_h$ $\forall h \in L$ and $x'_h(\theta) = u_h$ $\forall h \in U$). Moreover, as described in Section 3, it is:

$$\tilde{r} = r - \sum_{h \in L} E^h l_h - \sum_{h \in U} E^h u_h \quad , \quad \tilde{\xi} = \xi' - d_0 - \sum_{h \in L} d_h l_h - \sum_{h \in U} d_h u_h$$

$$x'_{B,k}(\theta) = T_{B,k}^{-1} \begin{bmatrix} \tilde{r} \\ \tilde{\xi} + \theta \end{bmatrix} = T_{B,k}^{-1} \begin{bmatrix} \tilde{r} \\ \tilde{\xi} \end{bmatrix} + \theta T_{B,k}^{-1} e^{n+1}$$

where $e^{n+1} = (0, \ldots, 0, 1)^T \in \Re^{n+1}$. Hence, the optimal solution of $P_{\xi'+\theta}$, $\theta \geq 0$, results to be of the kind $x' + \theta\Delta_x$, where the nonbasic components of Δ_x are equal to zero while the basic ones are given by $\Delta_{x_{B,k}} = T_{B,k}^{-1} e^{n+1}$.

The halfline $x'(\theta) = x' + \theta\Delta_x$, $\theta \geq 0$, verifies the equality constraints $\tilde{E}x'(\theta) = r$ and $\tilde{d}^T x'(\theta) = \xi' + \theta - d_0$, so that these points results to be optimal level solutions whenever $\tilde{l} \leq x'(\theta) \leq \tilde{u}$, that is to say whenever they are feasible (see Section 3). In this light, let $F_R = \max\{\theta \geq 0 : \tilde{l} \leqq x' + \theta\Delta_x \leqq \tilde{u}\}$. It is worth pointing out that $x'(\theta)$ is a vertex of the feasible region of $P_{\xi'+\theta}$, hence it belongs to an edge of the flow polytope X. As a consequence, the segment $[x', x'(F_R)]$ belongs to an edge of X; moreover, in the case x' is a vertex of X then $x'(F_R)$ is one of the vertices of X adjacent to x'.

The following procedures "*Main*()" can then be proposed. This procedure initializes the algorithm by first determining the set of feasible levels. This can be done by means of any simplex algorithm for minimum cost flows. Then, the optimal level solution corresponding to the starting level ξ_{min} is computed as well as its basis. The starting incumbent optimal solution is then initialized before the while cycle which will allow to visit the feasible levels from ξ_{min} to ξ_{max}, thus obtaining the optimal solution.

Notice that in procedure "*Main*()" there are two optional subprocedures. The first one, named "*ImplicitVisit*()", skips some of those feasible levels which cannot improve the incumbent optimal solution, thus reducing the number of while iterations needed to solve the problem and hence improving the performance of the algorithm itself. Notice that as smaller is the value UB of the incumbent optimal solution as more effective is the use of "*ImplicitVisit*()" subprocedure. In this very light the second optional subprocedure, named "*ImproveStartingValues*()", is aimed just to initialize the algorithm with a good incumbent optimal solution.

At the beginning of every while iterations, an optimal level solution x' is known. Starting from x' we need to determine a basis (B, k, L, U) which allow to obtain a segment of optimal level solutions $x'(\theta) = x' + \theta\Delta_x$, with $\theta \in [0, F_R]$ and $F_R > 0$. This could be done by analyzing all of the possible basis corresponding to x'. This leads to computational troubles, especially in the case of degeneracy. For this very reason a more efficient numerical approach is used. By means of a real step parameter $\delta > 0$ small enough and any optimal basis (B, k, L, U) of x', the "*Dual*()" procedure is invoked to solve problem $P_{\xi'+\delta}$ thus obtaining the optimal level solution $x'(\delta)$ and its basis which guarantees a value $F_R > 0$.

The objective function is then evaluated over the segment of optimal level solutions $x'(\theta)$, $\theta \in [0, F_R]$, in order to improve the incumbent optimal solution. Notice that $\overline{\theta} := \arg \min_{\theta \in [0, F_R]} z(\theta)$, where $z(\theta) = f(x'(\theta))$, can be implemented numerically, and eventually improved for specific functions $f(x)$ (see [3, 4, 15]).

Finally, the while iteration is closed by the updating of the current feasible level and its corresponding optimal level solution.

From a geometrical point of view, the single while iteration allow us to move from a vertex of the flow polytope X to another adjacent one corresponding to an higher feasible level by means of a simplex like operation.

The correctness and convergence (finiteness) of the algorithm follow the lines already proved in the literature of "optimal level solutions" parametric methods (see for all [3, 5]).

In particular, the correctness of the proposed algorithm yields since all the fea-

Procedure **Main**(*inputs*: P; *outputs*: Opt, $OptVal$)

Solve the problem $\min_{x\in X} d^T x$ and let $x_{min} \in X$ be its optimal solution with index partition B, L, U and reduced costs $\bar{d}$, let also be $\xi_{min} := d^T x_{min} + d_0$;

Let $k \in L \cup U$ be such that $\bar{d}_k \neq 0$ and let $\tilde{E}_B$ be the basic matrix of x_{min};

Solve $P_{\xi_{min}}$ by means of "Procedure Primal()" starting from the basis (B, k) and let x' be the optimal solution of $P_{\xi_{min}}$;

Let $\bar{x} := x'$, $UB := f(\bar{x})$, $\xi' := \xi_{min}$ and let $\delta > 0$ be the step parameter;

Compute the value $\xi_{max} := d_0 + \max_{x\in X} d^T x$;

\# Optional : $[\bar{x}, UB] := ImproveStartingValues()$;

while $\xi' < \xi_{max}$ *do*

let $[(B, k, L, U), \tilde{E}_B^{-1}] := Dual((B, k, L, U), \xi' + \delta, \tilde{E}_B^{-1})$;

define vector Δ_x having the nonbasic components equal to zero and the basic ones given by:

$$\Delta_{x_{B,k}} := \frac{1}{d_k - \tilde{d}_B^T \tilde{E}_B^{-1} E^k} \begin{bmatrix} -\tilde{E}_B^{-1} E^k \\ 1 \end{bmatrix}$$

set $F_R := \max\{\theta \geq 0 : \tilde{l} \leqq x' + \theta\Delta_x \leqq \tilde{u}\}$;

let $z(\theta) = \phi\left(\theta c^T \Delta_x + \overline{c^T} x' + c_0, \xi' + \theta\right)$;

let $\overline{\theta} := \arg\min_{\theta\in[0,F_R]} z(\theta)$;

if $z(\overline{\theta}) < UB$ *then*

$UB := z(\overline{\theta})$ and $\overline{x} := x' + \overline{\theta}\Delta_x$;

end if;

set $\xi' := \xi' + F_R$ and $x' := x' + F_R\Delta_x$;

\# Optional : $[\xi', x', (B, k, L, U), \tilde{E}_B^{-1}] := ImplicitVisit()$;

end while;

$Opt := \bar{x}$ and $OptVal := UB$;

end proc.

sible levels are scanned and the optimal solution is also an optimal level solution. As regards to the convergence, first note that at every iterative step of the proposed algorithm an edge of the feasible region is fully visited; note also that the level is increased from ξ' to $\xi' + F_R > \xi'$, so that it is not possible to consider an already visited edge; the convergence then follows since in a polytope there is a finite number of possible edges.

Remark 1 : Let us point out that problems P_ξ are independent of the function ϕ. This means that problems having the same feasible region, the same c, c_0, d and d_0, but different function ϕ, they share the same set of optimal level solutions. As a consequence, when procedure "*Main*()" explicitly visits all the feasible levels, these different problems are solved by means of the same number of iterations of the while cycle.

Let us now describe in details the optional subprocedures "*ImplicitVisit*()" and "*ImproveStartingValues*()". First of all, notice that function $z(\theta)$ evaluates the optimal level solutions for $\theta \in [0, F_R]$ while it represents an underestimation function for $\theta > F_R$, that is to say that:

$$\min_{x\in X_{\xi'+\theta},\ \theta>F_R} f(x) \geq z(\theta)$$

since $x'(\theta)$ is a dual feasible solution being the reduced costs independent to θ. For

this very reason, there is no need to visit the feasible levels such that $z(\theta) \geq UB$ since they cannot improve the incumbent optimal solution. The optional subprocedure "*ImplicitVisit*()" verifies the presence of feasible levels which cannot improve the incumbent optimal solution and hence which can be skipped. In the case some levels are implicitly visited a primal feasible optimal level solution is determined by means of procedure "*Dual*()".

Procedure **ImplicitVisit**(*outputs*: ξ', x', (B,k,L,U), $\tilde{E}_B^{-1}$)
if $\xi' < \xi_{max}$ *then*
let $\mathcal{L} = \{\theta \in (0, \xi_{max} - \xi'] : \ \phi\left(\theta c^T \Delta_x + c^T x' + c_0, \xi' + \theta\right) < UB\}$;
if $\mathcal{L} = \emptyset$ *then* $\xi' := \xi_{max}$
else if $\inf\{\mathcal{L}\} > 0$ *then*
$\xi' := \xi' + \inf\{\mathcal{L}\}$;
$[x', (B,k,L,U), \tilde{E}_B^{-1}] := Dual((B,k,L,U), \xi', \tilde{E}_B^{-1})$
end if;
end if;
end if;
end proc.

Subprocedure "*ImproveStartingValues*()" is just aimed to initialize the algorithm with a good starting incumbent optimal solution, in order to make the use of "*ImplicitVisit*()" more effective. Clearly, subprocedure "*ImproveStartingValues*()" results to be useless in the case "*ImplicitVisit*()" is not invoked.

6. Computational results

The previously described procedures have been fully implemented with the software MatLab 7.10 R2011b and tested on a computer having 6 Gb RAM and two Xeon dual core processors at 2.66 GHz. The following three different objective functions have been used in the computational test:

	$\phi(y_1, y_2)$	$f(x)$
P_1	$y_1 - y_2^2$	$\left(c^T x + c_0\right) - \left(d^T x + d_0\right)^2$
P_2	$y_1 y_2^3$	$\left(c^T x + c_0\right)\left(d^T x + d_0\right)^3$
P_3	y_1^3 / y_2^2	$\left(c^T x + c_0\right)^3 / \left(d^T x + d_0\right)^2$

Notice that the functions considered in these problems are not quasiconvex and hence allow the presence of many local minima. We considered problems with a number of nodes from $n = 10$ to $n = 100$. The performance of the algorithm has been tested with respect to the graph density, expressed by means of the variable *degree* which gives the average number of leaving arcs from the nodes over the number of nodes themselves. Clearly, $degree \in [0,1]$ and as smaller is the value of *degree* as more sparse is the graph. In this light, we considered graphs with a number of leaving arcs from the nodes equal to $n * degree$, where *degree* is chosen

Procedure **ImproveStartingValues**(*outputs*: $\bar{x}$, UB)
Let $x_{max} \in X$ be the optimal solution of $\max_{x \in X} d^T x$;
Solve $P_{\xi_{max}}$ by means of "Procedure Primal()" starting from x_{max} and let x'' be the obtained optimal solution;
if $f(x'') < UB$ *then* $\bar{x} := x''$, $UB := f(\bar{x})$ *end if*;
end proc.

to be 30% (Tables 1 and 2) and 70% (Tables 3 and 4). This means that the number of arcs in the various graphs is $m \approx n^2 * degree$.

The problems have been randomly created; in particular, arcs are randomly generated with the chosen degree and avoiding repetitions, vectors $c, d, l, u \in \Re^m$ have been generated by using the "randi()" MatLab function (c, d with components in the interval [-10,10], l with components in the interval [0,2] and $u - l$ with components in the interval [5,10]). In order to guarantee the existence of a feasible flow, it is $r = \frac{1}{2}E(l + u)$. Value $c_0 \in \Re$ is chosen equal to 0; $d_0 \in \Re$ is equal to 0 in P_1 while in P_2 and P_3 it has been chosen in order to have function $d^T x + d_0$ positive over the feasible region.

Various instances have been randomly generated and solved. In order to check the correctedness of the optimal global solution found, some of the instances have been solved also with the general solution algorithm proposed in [5]. The average number of iterations of the while cycle in procedure "*Main*" and the average CPU times spent by the algorithm to solve the instances are given as the results of the test (see Tables 1-4). These results point out the effectiveness of the improvements proposed in Section 5.

Regarding to Tables 1-4 notice that:

- "**n**" represents the number of nodes in the considered problems;
- "num" represents the number of instances randomly generated and solved within problems P_1, P_2 and P_3, with or without the use of the improving procedures;
- "Complete Visit" represents the average Iterations or CPU time spent to solve the instances within problem P_1 when neither "*ImproveStartingValues*" nor "*ImplicitVisit*" are used (hence all the feasible levels are explicitly scanned); we provide only the results related to P_1 since all the problems are solved in the same number of iterations (see Remark **1**);
- P_1, P_2 and P_3, represent the average iterations or CPU times spent to solve the instances within problems P_1, P_2 and P_3, respectively, when just "*ImplicitVisit*" is used or when both "*ImproveStartingValues*" and "*ImplicitVisit*" are used.

n	num	Complete Visit	With "*ImplicitVisit*" only			With both "*ImplicitVisit*" *and*"*ImproveStartingValues*"		
			P_1	P_2	P_3	P_1	P_2	P_3
10	1000	24.696	6.585	18.399	5.764	1.835	12.096	5.015
20	1000	134.24	16.447	102.07	15.411	1.998	68.145	12.98
30	1000	330.24	30.518	253.07	25.074	2.051	165.89	21.279
40	800	612.46	42.785	470.55	36.27	1.9863	331.85	30.824
50	500	984.02	67.88	758.29	42.686	2.098	517.75	36.776
60	300	1443.6	86.127	1115	51.257	2.1033	779.1	44.33
70	200	1989.4	102.48	1541.7	63.245	2.025	1068.6	54.01
80	100	2633.3	125.33	2039.9	75.52	2.06	1402.1	65.03
90	100	3373.5	162.47	2614.6	74.8	2.18	1861.3	64.67
100	100	4202	204.05	3266.3	116.29	2.2	2324.6	100.52

Table 1. Average Iterations - 30% arcs

The improvement criteria suggested in Section 3 results to be extremely effective in making the algorithm efficient; as expected, as bigger is the graph (number of nodes or density of arcs) as more expensive is the resolution of the problem. It is also worth pointing out that the effectiveness of the improvements criteria depends on the chosen objective function:

- as regards to the average number of iterations, the use of "*ImplicitVisit*" only provides better results in P_1 and P_3 than in P_2, while the additional use of "*ImproveStartingValues*" provides the best further improvement in P_1 and the worse improvement in P_3;

n	num	Complete Visit	With "*ImplicitVisit*" only			With both "*ImplicitVisit*" and"*ImproveStartingValues*"		
			P_1	P_2	P_3	P_1	P_2	P_3
10	1000	0.43618	0.12375	0.33882	0.11173	0.04269	0.23142	0.10123
20	1000	2.4148	0.33409	1.9156	0.31196	0.08883	1.3136	0.27501
30	1000	6.2514	0.71993	4.9911	0.57574	0.22269	3.3823	0.51657
40	800	12.35	1.2371	9.8578	0.99246	0.49555	7.1843	0.90332
50	500	21.268	2.4122	16.996	1.5144	1.1558	12.149	1.4135
60	300	33.622	3.9388	26.882	2.3921	2.262	19.816	2.2721
70	200	50.533	6.1371	40.476	3.9943	4.0184	30.005	3.824
80	100	72.996	9.793	58.321	6.2367	7.0307	43.445	6.0313
90	100	102.07	16.782	81.499	8.4984	13.004	63.238	8.2681
100	100	139.5	25.787	111.66	13.142	20.726	87.573	12.775

Table 2. Average CPU time (secs) - 30% arcs

n	num	Complete Visit	With "*ImplicitVisit*" only			With both "*ImplicitVisit*" and"*ImproveStartingValues*"		
			P_1	P_2	P_3	P_1	P_2	P_3
10	1000	59.536	9.555	45.093	10.098	1.951	29.448	8.605
20	1000	283.16	24.427	216.41	24.564	1.971	146.28	21.052
30	1000	676.87	47.814	520.75	37.164	2.047	358.13	31.685
40	800	1249.9	74.177	965.5	47.788	2.0438	666.77	41.219
50	500	2005.7	98.04	1555.7	55.176	2.026	1088.6	47.896
60	300	2955.3	131.62	2297.4	72.873	1.98	1634.4	62.78
70	200	4084.8	175.19	3179.6	87.42	2.055	2243.8	75.485
80	100	5445.8	182.33	4257.5	92.16	1.92	2901.4	81.19
90	100	6986.5	248.26	5460.3	80.2	2.06	3944.3	70.97
100	100	8726.6	282.37	6843	116.99	2.03	4731.9	103.25

Table 3. Average Iterations - 70% arcs

n	num	Complete Visit	With "*ImplicitVisit*" only			With both "*ImplicitVisit*" and"*ImproveStartingValues*"		
			P_1	P_2	P_3	P_1	P_2	P_3
10	1000	1.0372	0.18035	0.82091	0.19186	0.05271	0.55104	0.16984
20	1000	5.1037	0.52945	4.0656	0.51315	0.15298	2.8211	0.45965
30	1000	12.936	1.2777	10.359	0.94961	0.48605	7.3623	0.86337
40	800	25.543	2.5674	20.488	1.6473	1.2733	14.77	1.5416
50	500	44.384	4.7172	35.674	2.8196	2.9184	26.342	2.7014
60	300	71.258	8.2767	57.319	5.2393	5.7376	43.443	5.0677
70	200	108.54	15.313	87.219	8.6882	11.727	66.847	8.4796
80	100	159.59	22.351	128.61	14.049	18.469	97.488	13.812
90	100	226.23	40.993	182.06	23.029	35.467	146.31	22.846
100	100	312.64	62.096	252.25	34.492	55.534	200.69	34.292

Table 4. Average CPU time (secs) - 70% arcs

- as regards to the average cpu times, the use of "*ImplicitVisit*" only provides better results in P_1 and P_3 than in P_2, while the additional use of "*ImproveStartingValues*" provides good further improvements just in P_2;
- when both "*ImplicitVisit*" and "*ImproveStartingValues*" are used for problem P_1, almost all of the feasible levels are implicitly visited and very few iterations of the while cycle are needed to solve the problems, unfortunately the same improvements do not occur with respect to the cpu times since the "*ImplicitVisit*" procedure becomes extremely heavy from a computational point of view.

Finally, it is worth noticing that the proposed algorithm allows to solve in a reasonable time (about 0.5-3.5 minutes) nonconvex problems on graphs having 100 nodes and about 7000 arcs (see Table 4). For the sake of completeness, the procedure proposed in this paper has been compared with the "*fmincon*()" matlab command which resulted to be far slower (up to 50 times with respect to the spent CPU time) even for reaching a just local minimum (not in general global); in this light just the cases $n = 20, 30, 40$, with a 70% degree, have been considered since for $n \geq 50$ the "*fmincon*()" command did not produce any local minimum in a reasonable CPU

time.

7. Conclusions

The proposed algorithm allows to solve a wide range of nonconvex flow problems. In particular, the improvement criteria suggested in Section 5 resulted to be extremely effective in making the algorithm efficient. The correctness of the method guarantees that the global minimum is found. The use of the graph structure allows to solve large dimension problems in a reasonable time and with calculations affected by limited numerical errors given by the use of machine numbers.

References

[1] Barros A.I. (1998) Discrete and Fractional Programming Techniques for Location Models. Combinatorial Optimization, vol.3, Kluwer Academic Publishers, Dordrecht.

[2] Bazaraa M.S., Jarvis J.J., and H.D. Sherali (2009) Linear Programming and Network Flows. 4th edition, Wiley.

[3] Cambini R. and C. Sodini (2003) A finite algorithm for a class of nonlinear multiplicative programs. Journal of Global Optimization, 26:279-296.

[4] Cambini R. and C. Sodini (2007) An unifying approach to solve a class of parametrically-convexifiable problems. In: Generalized Convexity and Related Topics, I.V. Konnov, D.T. Luc and A.M. Rubinov (eds), Lecture Notes in Economics and Mathematical Systems, Springer, Berlin, 583:149-166.

[5] Cambini R. and C. Sodini (2010) Global optimization of a rank-two nonconvex program. Mathematical Methods of Operations Research, 71:165-180.

[6] Cooper W.W., Seiford L.M. and J. Zhu (eds) (2004) Handbook on Data Envelopment Analysis. International Series in Operations Research & Management Science, vol.71, Kluwer Academic Publishers, Boston.

[7] Ellero A. (1996) The optimal level solutions method. Journal of Information & Optimization Sciences, 17:355-372.

[8] Frenk J.B.G. and S. Schaible (2005) Fractional programming. In: Handbook of Generalized Convexity and Generalized Monotonicity, Nonconvex Optimization and Its Applications, 76:335-386, Springer, New York.

[9] Horst R. and H. Tuy (1996) Global Optimization: Deterministic Approaches. 3rd rev., Springer, Berlin.

[10] Horst R., Pardalos P.M. and N.V. Thoai (2001) Introduction to Global Optimization. 2nd ed., Nonconvex Optimization and Its Applications, vol.48, Kluwer Academic Publishers, Dordrecht.

[11] Konno H. and T. Kuno (1995) Multiplicative programming problems. In: Handbook of Global Optimization, R. Horst and P.M. Pardalos (eds), Nonconvex Optimization and Its Applications, Kluwer Academic Publishers, Dordrecht, 2:369-405.

[12] Konno H., Thach P.T. and H. Tuy (1997), *Optimization on low rank nonconvex structures*, Nonconvex Optimization and Its Applications, vol.15, Kluwer Academic Publishers, Dordrecht.

[13] Mjelde K.M. (1983) Methods of the Allocation of Limited Resources. Wiley, New York.

[14] Schaible S. (1995) Fractional programming. In: Handbook of Global Optimization, Horst R. and P.M. Pardalos (eds), Nonconvex Optimization and Its Applications, 2:495-608, Kluwer Academic Publishers, Dordrecht.

[15] Schaible S. and C. Sodini (1995) A finite algorithm for generalized linear multiplicative programming. Journal of Optimization Theory and Applications, 87:441-455.

[16] Tuy H. (1998) Convex Analysis and Global Optimization. Nonconvex Optimization and Its Applications, vol.22, Kluwer Academic Publishers, Dordrecht.