Computers & Operations Research 36 (2009) 2554 – 2564

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

Finding non-dominated solutions in bi-objective integer network flow problems

Augusto Eusébio[a,b,*], José Rui Figueira[b,c]

[a]Escola Superior de Tecnologia e Gestão, Instituto Politécnico de Leiria, Morro do Lena, Alto Vieiro, 2411-901 Leiria, Portugal
[b]CEG-IST, Centre for Management Studies, Instituto Superior Técnico, Technical University of Lisbon, TagusPark, Av. Cavaco Silva, 2780-990 Porto Salvo, Portugal
[c]LAMSADE, Université Paris-Dauphine, Place du Maréchal De Lattre de Tassigny, 75 775 Paris Cedex 16, France

ARTICLE INFO

Available online 17 November 2008

Keywords:
Bi-objective integer network flows
Network simplex algorithm
Efficient solutions
Non-dominated solutions

ABSTRACT

This paper deals with an algorithm for finding all the non-dominated solutions and corresponding efficient solutions for bi-objective integer network flow problems. The algorithm solves a sequence of ε-constraint problems and computes all the non-dominated solutions by decreasing order of one of the objective functions. The optimal integer solutions for the ε-constraint problems are determined by exploring a branch-and-bound tree. The algorithm makes use of the network structure to perform the computations, i.e., the network structure of the problem is not destroyed with the inclusion of an ε-constraint. This paper presents the main features of the algorithm, the theoretical bases of the proposed approach and some computational issues. Experiments were done and the results are also reported in the paper.

0. Introduction

Network flow problems can be found in a large range of real-world problems. Moreover, this class of problems also appears as subproblems in other complex models (see, for example, [1]). It is also well known that the very nature of real-world problems is in general multi-dimensional and thus it makes sense to use multi-objective based models to deal with such real-world decision-making problems. Therefore, the study of multi-objective network flow problems is of general interest. The reasons underlying the application of these models and techniques differ widely across the application areas. Some of the techniques can just be suitable for linear multi-objective network flow problems while others are suitable for its integer counterpart, i.e., the multi-objective integer network flow.

In this paper we propose an exact algorithm for the integer bi-objective network flow (BOINF) problem. Papers on exact methods for bi-objective network flow problem are rather scarce. Approaches dealing with the linear bi-objective network flow problems (BONF) can be found in [2–6], while for the integer case we can cite [7–13]. A very recent work dealing with bi-objective integer flows can be found in Raith and Ehrgott [33].

Computing the whole set of non-dominated (ND) solutions for integer multi-objective network flow problems is still a challenging problem. Despite the fact the problem has been proved to be intractable, even for the case of two objectives, there are many questions which remain open and more research is needed in the field (see [14,15] for a comprehensive review). One of the most interesting open questions is to check whether it is possible or not to devise an effective algorithm for the implicit generation of the whole set of ND solutions without destroying the network structure of the problem. A first answer to this question was provided in an algorithm presented in [7]. This paper improves and implements such an algorithm for BOINF . The proposed algorithm can also be used for generating all the efficient solutions in a particular region of the feasible objective set. This is an important feature that can be incorporated in interactive procedures (see, for example, [16, chapter 13]).

The main difficulty associated with integer multi-objective network flow problems comes from the fact that the set of ND solutions in the objective space (and the corresponding efficient set in the decision space) is composed of two types of solutions: supported and unsupported. While the former solutions can be computed through linear convex combinations of the objectives, the latter ones seem to be much more difficult to find. They are located in the dual gaps between ND supported solutions and finding them without destroying the structure of networks is a rather difficult task. This aspect along with the large size of the whole set of ND solutions renders the problem even more difficult to solve. As for the size of the ND set it is well known that the problem is intractable, i.e., the number of ND solutions grows exponentially with the size of the networks. Based on the pathological graphs by Zadeh [17] and Ruhe [18] proved that the number of supported ND solutions is intractable for BONF problems. As for the issue of determining the ND set by keeping the network structure the three works published in the literature [10,11,19] were proved to be wrong [10,12,20].

* Corresponding author at: Escola Superior de Tecnologia e Gestão, Instituto Politécnico de Leiria, Morro do Lena, Alto Vieiro, 2411-901 Leiria, Portugal.
Tel.: +351 244 820 300; fax: +351 244 820 310.
E-mail addresses: aeusebio@estg.ipleiria.pt (A. Eusébio), figueira@ist.utl.pt (J.R. Figueira).

0305-0548/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2008.11.001

In this paper a method for identifying the set of ND solutions (and corresponding efficient solutions in the decision space) for BOINF is presented. The method solves a sequence of ε-constraint problems by a branch-and-bound algorithm. An interesting property of network flows allows to compute non-integer solutions by preserving always the network structures. This is one of the main features of the proposed method. To our best knowledge this is the first algorithm able to compute all the ND solutions by implicit enumeration, without destroying the structure of networks. The algorithm can also be used for identifying all the efficient solutions in the decision space by exploring the branch-and-bound tree until all the solutions corresponding to the same ND solution are obtained.

The proposed approach makes use of parametric programming and a network simplex algorithm for computing non-integer solutions. The network simplex algorithm takes advantage of the special properties of network flow problems to make enormous computational gains. Graph structures are used instead of coefficient matrices allowing thus a more efficiently storing and accessing and the replacing of many arithmetic operations by more efficient logical operations. The network simplex algorithm enables one to solve problems faster than with a standard simplex approach (see [21]). Because the proposed approach makes use of parametric programming it can be seen as a kind of two-phase based method [22,23], but only some supported ND solutions are obtained in a sequential way during the optimization process of solving the ε-constraint problems.

The method has been implemented and experiments were made; the results are also provided in this paper. The experiments were mainly designed to know the maximum size of networks we can deal with, regarding CPU and memory requirements, and the difference between the number of ND solutions and efficient solutions. It is well known that, in general, the number of ND solutions is significantly smaller than the number of efficient solutions (see [24]); the results from the application of proposed algorithm can corroborate this observation.

The paper is organized as follows. Section 1 presents concepts, definitions and notation. Section 2 contains a brief presentation of the network simplex method. Section 3 presents the proposed method. Section 4 illustrates the method. Section 5 reports some computational results. Finally, conclusions and avenues for future research are provided.

1. Concepts: definitions and notation

Let $\mathscr{G}=(\mathscr{S},\mathscr{A})$ denote a *directed* and *connected graph*, where $\mathscr{S}$ is a finite set of *nodes* or *vertices* with cardinality $|\mathscr{S}|=m$, and $\mathscr{A}$ is a collection of ordered pairs of elements of $\mathscr{S}$ called *arcs*, with cardinality $|\mathscr{A}|=n$.

A graph $\mathscr{G}'=(\mathscr{S}',\mathscr{A}')$ is called a *subgraph* of $\mathscr{G}=(\mathscr{S},\mathscr{A})$ if $\mathscr{S}'\subseteq\mathscr{S}$ and $\mathscr{A}'\subseteq\mathscr{A}$. It is a spanning subgraph of $\mathscr{G}$ if $\mathscr{S}'=\mathscr{S}$. A *path* $\mathscr{P}$ is a sequence of vertices and arcs, $i_1-a_1-i_2-a_2-\cdots-i_{s-1}-a_{s-1}-i_s$, without repetition of vertices and where for which $1\leqslant k\leqslant s-1$ either $a_k=(i_k,i_{k+1})\in\mathscr{A}$, or $a_k=(i_{k+1},i_k)\in\mathscr{A}$. A *directed path* is a path without backwards arcs. A *cycle* $\mathscr{C}$ is a closed path where the only repeated vertex is the starting and the end point that coincide. A *directed cycle* is a closed directed path. When in a given graph $\mathscr{G}$ there is always a path linking any two different vertices of $\mathscr{G}$, the graph is called *connected*. A *tree* $\mathscr{T}=(\mathscr{V},\mathscr{E})$ is a connected subgraph that contains no cycle where $\mathscr{V}\subseteq\mathscr{S}$ and $\mathscr{E}\subseteq\mathscr{A}$. A tree $\mathscr{T}$ is called a *spanning tree* when it spans the set of vertices $\mathscr{S}$ of $\mathscr{G}$, that is $\mathscr{V}=\mathscr{S}$. A spanning tree is denoted by $\mathscr{T}=(\mathscr{S},\mathscr{E})$. Consider (k,l) a given arc belonging to the set $\mathscr{A}$ but not in $\mathscr{E}$. Then, there is a unique cycle $\mathscr{C}$ when the arc (k,l) is added to $\mathscr{E}$, if it exists. The direction of $\mathscr{C}$ is the same as (k,l). In a cycle $\mathscr{C}$ a partition of its vertices can be made by separating the arcs having the same direction as $\mathscr{C}$ from the arcs in the opposite direction. The collection of all possible cycles of this type is called *fundamental cycle basis* (for more details about network optimization, see [1,21]).

A directed graph with numerical values assigned to its vertices and/or arcs is called a *network*. Let $\mathscr{G}=(\mathscr{S},\mathscr{A})$ denote a network with two "*costs*" c_{ij}^1 and c_{ij}^2, a *lower bound* l_{ij} and an *upper bound* or *capacity* u_{ij} associated with every arc $(i,j)\in\mathscr{A}$. The numerical values l_{ij} and u_{ij}, respectively, denote the minimum and the maximum amounts that must flow on the arc (i,j). Finally, let x_{ij} denote the *amount of flow* on the arc (i,j). A numerical value b_i is also associated with each vertex $i\in\mathscr{S}$ denoting its *supply* (if $b_i>0$) or its *demand* (if $b_i<0$). A vertex with $b_i=0$ is called a *transshipment vertex*. The BONF problem can be stated as follows:

$$\begin{aligned}
\min\quad & f_1(x)=\sum_{(i,j)\in\mathscr{A}} c_{ij}^1 x_{ij}\\
\min\quad & f_2(x)=\sum_{(i,j)\in\mathscr{A}} c_{ij}^2 x_{ij}\\
\text{s.t.}\quad & \sum_{j|(i,j)\in\mathscr{A}} x_{ij}-\sum_{k|(k,i)\in\mathscr{A}} x_{ki}=b_i,\quad \forall i\in\mathscr{S}\\
& l_{ij}\leqslant x_{ij}\leqslant u_{ij},\quad \forall(i,j)\in\mathscr{A}
\end{aligned}\tag{1}$$

In what follows, the assumptions below must be taken into account: the graph is directed and connected; all the numerical values for the costs, lower and upper bounds on the arcs and supplies/demands on the vertices are integral and finite; the condition $\sum_{i\in\mathscr{S}} b_i=0$ must be fulfilled; the BONF problem has at least two feasible solutions and the minimum values for the individual objective functions are different. It can be assumed without lost of generality that $l_{ij}=0,\forall(i,j)\in\mathscr{A}$ (see [21]).

Problem (1) can be stated in a more dense form as follows:

$$\begin{aligned}
\min\quad & F(x)=(f_1(x),f_2(x))\\
\text{s.t.}\quad & x\in X\leftarrow\{x\in\mathbb{R}^n|Ax=b,l\leqslant x\leqslant u\}
\end{aligned}\tag{2}$$

where $x=(x_{i_1j_1},\ldots,x_{i_nj_n})\in\mathbb{R}^n$ is the vector of decision variables; X is the set of feasible solutions to (1); A is the $m\times n$ *node–arc incidence matrix*; l is the vector of lower bounds; u is the vector of upper bounds; b is the vector of supplies/demands on vertices; $F(x)=(f_1(x),f_2(x))$ is the vector of objectives to be "minimized" and $Y=F(X)$ is the set of all feasible solutions y in $\mathbb{R}^2$, where $y=(y_1,y_2)$ with $y_q=f_q(x)$ for $q=1,2$. By adding integrality requirements to the variables in X we obtain the BOINF. It can then be written as follows:

$$\begin{aligned}
\min\quad & F(x)\\
\text{s.t.}\quad & x\in X^I
\end{aligned}\tag{3}$$

where $X^I=\{x\in\mathbb{Z}^n|Ax=b,l\leqslant x\leqslant u\}$.

It is well known that for minimum cost network flow problems defined on a directed graph $\mathscr{G}$ with a root arc, every basis corresponds to a rooted spanning tree $\mathscr{T}$ and every extreme point is integer valued (see [21]). Therefore, the set X and the convex hull of X^I have the same extreme points.

Some concepts of multi-objective linear and integer programming must also be reviewed (see [16,25] and [22]). *Dominance* is a key concept. Let us define this concept for the general multiple objective case with $p\geqslant 2$ objectives. Let y', $y''\in\mathbb{R}^p$, the notation $y''\leqq y''$ means that $y'_q\leqslant y''_q$, $\forall q=1,2,\ldots,p$.

Definition 1.1 (*Dominance*). Consider y' and y'' two objective solutions. Then, y' dominates y'' iff $y'\leqq y''$ and $y'\neq y''$, that is, $y'_q\leqslant y''_q$ for all $q=1,\ldots,p$ with at least one strict inequality.

Definition 1.2 (*ND solution*). A solution $y' \in Y$ is called ND iff there is no other solution $y \in Y$ such that $y \leqq y'$ and $y \neq y'$. Otherwise, y' is a dominated solution.

A distinction between efficient solutions in the decision variable space and ND solutions in the objective space can be made. Efficient solutions are crucial for the usefulness of multiple objective methods. This concept was first introduced by Pareto [26]. Thus, these solutions are called *Pareto optimal*, and also *non-inferior* or *functional efficient* solutions.

Definition 1.3 (*Efficient solution*). A solution $x' \in X$ is said to be efficient iff it is impossible to find another solution $x \in X$ with a better evaluation of a given objective without deteriorating the evaluations of at least another objective, i.e., a solution $x' \in X$ is said to be efficient iff it is impossible to find another solution $x \in X$ such that $y = f(x) \leqq y' = f(x')$ and $y \neq y'$.

Theorem 1.1. *$x \in X$ is efficient if and only if there exists a $\lambda \in \Lambda = \{\lambda \in \mathbb{R}^p : \lambda_i > 0, \sum_{i=1}^{p} \lambda_i = 1\}$ such that x minimizes the weighted-sum linear programming problem* $\min\{\lambda^{\mathrm{T}} Cx : x \in X\}$ (*see* [16]).

In multiple objective linear integer programming, two types of ND solutions can be distinguished: supported and unsupported ND solutions. Let,

$$Y^{\geqq} = Conv(ND(Y^I) + \mathbb{R}^p_{\geqq})$$

where $\mathbb{R}^p_{\geqq} = \{y \in \mathbb{R}^p | y \geqq 0\}$ and $ND(Y^I) + \mathbb{R}^p_{\geqq} = \{y \in \mathbb{R}^p : y = y' + y'', y' \in ND(Y^I)$ and $y'' \in \mathbb{R}^p_{\geqq}\}$, *Conv* stands for convex hull and $Y^I = F(X^I)$.

Definition 1.4 (*Supported ND solution*). Let y denote an ND objective solution. Then, if y is on the boundary of $Y^{\geqq}$, y is a supported ND solution. Otherwise, y is an unsupported ND objective solution.

Supported ND solutions can be obtained by weighted sum based techniques [16].

Definition 1.5 (*Supported-extreme ND solution*). Let y denote a supported ND solution. Then, y is a supported-extreme solution if it is an extreme point of $Y^{\geqq}$. Otherwise, y is a supported non-extreme solution.

Inverse images of *supported* ND solutions are said to be supported efficient solutions and inverse images of unsupported ND objective solutions are said to be *unsupported efficient solutions*.

Let $ND(Y^I)$ denote the set of all the ND solutions to Y^I; $NDS(Y^I)$ denotes the set of all the supported ND solutions to Y^I; $NDU(Y^I)$ denotes the set of all the unsupported ND solutions to Y^I, that is, $NDU(Y^I) = ND(Y^I) \backslash NDS(Y^I)$; $NDE(Y^I)$ denotes the set of all the supported-extreme ND solutions; $EF(X^I)$ denotes the set of all the efficient solutions to X^I; $EFS(X^I)$ denotes the set of all the supported efficient solutions and $EFU(X^I)$ denotes the set of all the unsupported efficient solutions, that is, $EFU(X^I) = EF(X^I) \backslash EFS(X^I)$.

The ε-constraint approach [16] can be easily used in multiple objective integer problems without any additional restrictions. Efficient solutions can be characterized as optimal solutions for the ε-constraint problem.

The ε-constraint problem associated with problem (3) can be stated as follows:

$$\begin{aligned} \min \quad & f_1(x) \\ \text{s.t.} \quad & x \in X^I \\ & f_2(x) \leqslant \varepsilon \end{aligned} \tag{4}$$

where ε is a scalar. It varies among all the values for which (4) remains feasible. In order to identify a set of efficient solutions, a sequence of problems (4) is solved for each different value of ε [27]. Thus, for BOINF problems the entire set of ND solutions, $ND(Y^I)$, can be determined by solving a sequence of problems (4).

Theorem 1.2. *Let y_1^* be an optimal value for problem* (4), *$X^* = \{x \in X^I : y_1^* = f_1(x)\}$ and $\bar{y}_2 = \min_{x \in X^*} f_2(x)$. Then $y' = (y_1^*, \bar{y}_2)$ is an ND solution for* (3) (*see* [27]).

Problem (4) will be used in the algorithm outlined in Section 3 to determine all the ND solutions and all the efficient solutions for problem (3).

2. A brief overview of the network simplex method

Let us now succinctly recall the network simplex method on minimum cost network flow problems (see Algorithm 1). Consider the minimum cost flow problem defined on a directed graph $\mathscr{G}$ with a *root arc*. The basic idea for any variant of the network simplex method is a spanning tree structure (STS), of the form $(\mathscr{T}, L, U)$. Such a structure (or solution) is obtained when, for any arc not belonging to this tree, the flow value is fixed at its lower bound level or at its upper bound level. All the arcs fixed at their lower bound level belong to the set L, while all the arcs fixed at their upper bound level belong to the set U. The remaining arcs are those belonging to the spanning tree $\mathscr{T}$. A minimum cost network flow problem has always at least one optimal STS (see [1]). It is possible to find an optimal STS by shifting from one STS to another, successively. At each iteration, we exchange a pair of arcs (one arc entering in $\mathscr{T}$ and one arc coming out of $\mathscr{T}$). Any STS corresponds to one feasible basic solution in linear programming, and each shift from one STS to another coincides with one pivoting operation in the standard simplex method. The initialization of the algorithm consists of finding one feasible STS (or equivalently, a feasible basic solution in the standard simplex method). Two vectors are associated with this STS, the flow x (primal solution) and the potential π (dual solution). Each iteration of the method consists of: (1) identifying one eligible arc (k,l) with $(k,l) \notin \mathscr{T}$; (2) adding the arc (k,l) to $\mathscr{T}$ and finding an arc (p,q) coming out of $\mathscr{T}$ and, updating STS and the primal and dual solution (x, π).

An arc, (i,j), not belonging to $\mathscr{T}$ is said to be eligible if:

(i) Its reduced cost, $\bar{c}_{ij}$, is strictly negative and its flow is at its lower bound, that is, $\bar{c}_{ij} < 0$ and $(i,j) \in L$.
(ii) Its reduced cost is strictly positive and its flow is at its upper bound, that is, $\bar{c}_{ij} > 0$ and $(i,j) \in U$.

The reduced cost of a given arc (i,j) is defined as follows:

$$\bar{c}_{ij} = c_{ij} - \pi_i + \pi_j,$$

where π_i and π_j are the dual variables associated with the vertices i and j, respectively. It should be remarked that for all the arcs $(i,j) \in \mathscr{T}$ the reduced cost $\bar{c}_{ij} = 0$.

Algorithm 1. Network simplex algorithm {*Computing a minimum cost flow*}.

Step 1: *Let $k = 0$. Find a starting feasible $STS^{(k)}$.*
Step 2: *If $STS^{(k)}$ is optimal STOP; otherwise find the adjacent STS, $STS^{(k+1)}$:*
  (*a*) *Let $(\mathscr{T}, L, U)$ denote the spanning tree structure, x be the flow and π the dual vectors associated with $STS^{(k)}$;*
  (*b*) *Select an entering arc (k,l) not in $\mathscr{T}$;*

(c) *Add* (k,l) *to* $\mathscr{T}$ *and remove* (p,q) *from* $\mathscr{T}$;
(d) *Update the STS and the solutions* x *and* π.
Step 3: *Let* $k=k+1$, $STS^{(k)}=STS^{(k+1)}$ *and go to Step 2.*

At each iteration, the network simplex method shown in Algorithm 1 always gives an integer solution for the minimum cost network flow problem. But, an important property is that it is possible to obtain non-integer solutions between two adjacent STSs. Let us recall that when moving from one STS to an adjacent one, an amount of flow, $\triangle$, must be sent along the direction of cycle $\mathscr{C}$. This quantity $\triangle$ is integer. But, what happens if a non-integer amount of flow is sent along $\mathscr{C}$? It is obvious that a non-integer solution will be obtained. This solution has exactly $|\mathscr{C}|$ non-integer variables, but it does not define a STS. This idea is very important if we wish to obtain non-integer solutions for the LP-relaxation of problem (4).

Proposition 2.1. *Let* STS' *and* STS'' *be two adjacent STSs and* $\mathscr{C}$ *the cycle allowing to move from* STS' *to* STS'', *when an amount of flow* Δ *is sent along this cycle. Then, when sending a non-integer amount along* $\mathscr{C}$, *a non-integer solution is obtained and it contains exactly* $|\mathscr{C}|$ *non-integer variables.*

3. Outline of the method

This section presents the algorithm for searching the whole set of ND solutions, $ND(Y^I)$, for BOINF problems. The method solves a sequence of ε-constraint problems (4) and makes use of a branch-and-bound algorithm to determine integer optimal solutions. ND solutions are determined by decreasing order of the values for the second objective function. The algorithm can also compute all the efficient solutions associated to a given ND solution. It suffices to explore the branch-and-bound tree to get all the efficient solutions or alternative optima for problem (4). In this case the code computes all the alternative efficient optimal basis to be sure that the pivoting operations lead to the next adjacent efficient basis (see [16, p. 122]).

This method strongly depends on the computation of the STSs in the network simplex algorithm. Furthermore, the network simplex algorithm is also enough powerful to avoid the following major issues in network simplex algorithms (see also [21]), mainly due to degeneracy. (It should be noticed that in the absence of degeneracy, the network simplex algorithm converges in a finite number of iterations; by degeneracy, we mean that there exists at least one arc (i,j) in $\mathscr{T}$, other than the root arc, having a flow value equal to zero or equal to the upper bound, u_{ij}.):

1. *Cycling*. It is well known that the network simplex algorithm can *cycle*, that is, an infinite loop through a sequence of degenerate pivots can occur. However, cycling prevention rules are easy to implement and computationally beneficial for high degenerate problems. The cycling prevention rule is based on maintaining a *strongly STS* that is a feasible rooted spanning tree in which all degenerate arcs (i,j) (if any) are pointing toward the root, that is, j lies on the path from i to the root node. For more details on how to obtain and maintain strongly feasible tree structures (see [21]).
2. *Stalling*. Beyond the use of strongly feasible tree structures guarantees finite convergence it is still possible that the number of consecutive degenerate pivots becomes exponential in m and n, what is denominated *stalling*. The *least recently considered* (LRC) variable choice rule guarantees that the number of pivots in a sequence of degenerate pivots in which the set of arcs in $\mathscr{T}$ remains the same is no more than n (see [21]).

The network simplex algorithm that has been implemented is mainly based on the procedures proposed in [28], which means that it avoids cycling and stalling.

A brief overview of the algorithm can be presented as follows:

1. The algorithm starts by computing the lexicographically minimum solution $lex\min\{f_1(x),f_2(x) : x \in X\}$.
2. Then, the algorithm computes all the ND solutions until reaching an optimal solution to objective f_2. These ND solutions are obtained by solving a sequence of ε-constraint problems. The fractional optimal solutions to an ε-constraint problem is firstly obtained and then a branch-and-bound procedure is applied to get the integer optimal solution. An integer optimal solution to the ε-constraint problem is an ND solution to the bi-objective problem if it is unique. Otherwise, the algorithm finds the one corresponding to the minimum of $f_2(x)$ which is also ND.

The general procedure will be presented next. Then we will show how integer solutions can be obtained from non-integer optimal ones. Finally, the correctness of the algorithm is addressed.

3.1. The general procedure

This section presents the general procedure to find all ND solutions for the BOINF problem. Algorithm 2 computes all the ND solutions starting from the ND solution $y^{(0)}=(y_1^*,\bar{y}_2)$ associated with the lexicographically minimum of $f_1(x)$ ($y_1^*=\min_{x\in X} f_1(x)$ and $\bar{y}_2=\min_{x^*\in X^*} f_2(x^*)$, where $X^*=\{x^* : f_1(x^*)=y_1^*\}$) and ending with the minimum of $f_2(x)$. The remaining ND solutions are determined by decreasing order of the values for the second objective function and making use of the ε-constraint problem. The algorithm can be presented as follows.

Algorithm 2. *Computing all ND solutions.*

Step 1: *Let NDc denote the current set of ND solutions. Set* $NDc=\{\}$.
Step 2: *Find the solution* $y^{(0)}$ *and the value* y_2^* *such that*

$$y^{(0)}=(y_1^*,\bar{y}_2)=lex\min_{x\in X}\begin{pmatrix} f_1(x)\\ f_2(x)\end{pmatrix} \tag{5}$$

$$y_2^*=\begin{array}{ll}\min & f_2(x)\\ \text{s.t.} & x\in X\end{array} \tag{6}$$

Set $k=0$, $NDc=NDc\cup\{y^{(0)}\}$.
Step 3: *Let* $\varepsilon=y_2^{(k)}-\rho$, *with* ρ *being a positive real number sufficiently small so that there is no ND solution* $y=(y_1,y_2)$ *for which* y_2 *is between* $y_2^{(k)}$ *and* $y_2^{(k)}-\rho$. *Find the solution* $y^{(k+1)}$ *such that*

$$y_1^{(k+1)}=\begin{array}{ll}\min & f_1(x)\\ \text{s.t.} & x\in X^I\\ & f_2(x)\leqslant\varepsilon\end{array} \qquad (\varepsilon\text{-const})$$

and $y_2^{(k+1)}=\min_{x\in X^{(k+1)}} f_2(x)$, *where* $X^{(k+1)}=\{x\in X^I : y_1^{(k+1)}=f_1(x)\}$. *Set* $NDc=NDc\cup\{y^{k+1}\}$.
Step 4: *If* $y_2^{(k+1)}=y_2^*$ *STOP*; *otherwise set* $k=k+1$ *and go to Step 3.*

Step 2 of the algorithm finds the lexicographical optimal solution $y^{(0)}$ for BOINF problem. This can be done by making use of the network simplex algorithm, solving the network parametric problem $\min_{x\in X}\lambda f_1(x)+(1-\lambda)f_2(x)$, with $\lambda\in]0,1[$ being sufficiently close to 1, in such a way that the optimal solution is also optimal for the problem $\min_{x\in X} f_1(x)$, that is, the reduced cost $\lambda\bar{c}_{ij}^1+(1-\lambda)\bar{c}_{ij}^2$ related to the arcs $(i,j)\in L$ are non-negative and the arcs $(i,j)\in U$ are non-positive for $\lambda\in[\lambda_1,\lambda_2]$ such that $\lambda_1<1$ and $\lambda_2\geqslant 1$. The existence of

such a λ is guaranteed by the properties of the linear programming (see [25]).

It is well known that in general, in Step 3, $\rho \in]0,1]$, since there is no integer solution $y=(y_1,y_2)$ such that $y_2 \in]y_2^{(k)}, y_2^{(k)}-1[$. However, it must be possible for particular BOINF problems to have a ρ value greater than 1, which decreases the region where solutions are searched and may increase the speed of the computations.

3.2. Computing an integer optimal solution

Integer optimal solutions to problem (ε-const) are obtained by a branch-and-bound method where the main components: branching, bounding and fathoming are described in the algorithm below.

Let W denote a list of subproblems (K) to be analyzed and Incumbent denote the solution with the best value for problem (ε-const) among all the subproblems already analyzed.

Step 1: Consider the initial problem (K) with $X^{(K)}=X^I$. This problem represents the root of the branch-and-bound tree.
Set $W=\{\}$ and $Incumbent=(+\infty,+\infty)$

$$\begin{aligned} \min \quad & F(x) \\ \text{s.t.} \quad & x \in X^{(K)} \end{aligned} \qquad (K)$$

Step 2: Consider the following cases according to the set $X^{(K)}$.

(a) If there exist two supported efficient adjacent STSs, $STS^{(k_1)}$ and $STS^{(k_2)}$, such that $y_2^{(k_1)} < \varepsilon \leqslant y_2^{(k_2)}$, set $x^{(k')}=x^{(k_1)}+\Delta_1 d$, where $\Delta_1 = \Delta \times (\varepsilon - y_2^{(k_1)})/(y_2^{(k_2)} - y_2^{(k_1)})$, and $d=(x^{(k_2)}-x^{(k_1)})/\Delta$. The amount Δ is the quantity of flow sent along $\mathscr{C}$ in a simplex iteration, when the arc $(k,l) \notin \mathscr{T}$ is inserted into $\mathscr{T}$, allowing to move from $STS^{(k_1)}$ to $STS^{(k_2)}$.

(i) If all the components of $x^{(k')}$ are integer or $y_1^{(k')}$ is greater or equal than the first component of *Incumbent*, update *Incumbent* (fathoming).

(ii) Otherwise, consider the two new problems ($K1$) and ($K2$) (branching).

$$\begin{aligned} \min \quad & F(x) \\ \text{s.t.} \quad & x \in X^{(K)} \\ & x_{kl} \leqslant \lfloor x_{kl}^{(k')} \rfloor \end{aligned} \qquad (K1)$$

and

$$\begin{aligned} \min \quad & F(x) \\ \text{s.t.} \quad & x \in X^{(K)} \\ & x_{kl} \geqslant \lceil x_{kl}^{(k')} \rceil \end{aligned} \qquad (K2)$$

Set $W = W \cup \{K1, K2\}$.

(b) Otherwise, if there exists a supported efficient $STS^{(k')}$ such that $y_2^{(k')} < \varepsilon$ and there is no another supported efficient STS'' such that $y_2'' > y_2^{(k')}$ then update *Incumbent*.

Step 3: If $W=\{\}$, STOP; otherwise, consider another problem (K) from W, set $W = W \backslash \{(K)\}$ and repeat Step 2.

In Step 2 the update of the *Incumbent* is done setting $Incumbent = y^{(k')}$ when $y_1^{(k')} < Incumbent_1$ or $y_1^{(k')} = Incumbent_1$ and $y_2^{(k')} < Incumbent_2$. The notation $\lfloor x_{kl}^{(k')} \rfloor$ stands for the greatest integer less than or equal to $x_{kl}^{(k')}$ and $\lceil x_{kl}^{(k')} \rceil$ for the smallest integer greater than or equal to $x_{kl}^{(k')}$.

In Step 2(a)(ii), when the current problem is split out into two subproblems, $STS^{(k1)}$ is efficient for one of them and $STS^{(k2)}$ for the other. This allows for search of the two adjacent solutions in Step 2(a) through a path that links adjacent efficient solutions. It is well known that given an efficient STS' for the BOINF and an efficient extreme point $\bar{x}$, starting from STS' and performing only efficient pivots, i.e., moving from one efficient solution to another adjacent efficient solution, we can reach some STS corresponding to $\bar{x}$ (see [29]). Consider an efficient solution x' and move to an adjacent efficient solution x''. Solutions x' and x'' are optimal for the parametric problem $\min_{x \in X} \lambda f_1(x) + (1-\lambda) f_2(x)$ for $\lambda \in [\lambda_1, \lambda_2]$, where $\lambda_1 = \min \Lambda_{12}$ and $\lambda_2 = \max \Lambda_{12}$,

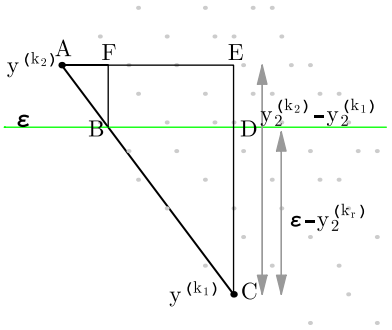

Fig. 1. Computing Δ_1.

$$\Lambda_{12} = \{\lambda \in [0,1] : \lambda(\bar{c}_{ij}^1 - \bar{c}_{ij}^2) + \bar{c}_{ij}^2 \geqslant 0 \quad \text{if} \quad (i,j) \in L$$

$$\text{and} \quad \lambda(\bar{c}_{ij}^1 - \bar{c}_{ij}^2) + \bar{c}_{ij}^2 \leqslant 0 \quad \text{if} \quad (i,j) \in U\} \qquad (7)$$

λ_1 is either 0 or $-\bar{c}_{k_1 l_1}^2/(\bar{c}_{k_1 l_1}^1 - \bar{c}_{k_1 l_1}^2)$ for some arc (k_1,l_1) not in $\mathscr{T}$ and λ_2 is either 1 or $-\bar{c}_{k_2 l_2}^2/(\bar{c}_{k_2 l_2}^1 - \bar{c}_{k_2 l_2}^2)$ for some arc (k_2,l_2) not in $\mathscr{T}$. Whenever $\lambda_1 \in]0,1[$, there exists $\lambda_1' \in [0,1]$, $\lambda_1' \leqslant \lambda_1$ such that a new optimal solution is obtained for the parametric problem with $\lambda \in [\lambda_1', \lambda_1]$, when the entering arc is (k_1,l_1). In the same way, whenever $\lambda_2 \in]0,1[$, there exists a $\lambda_2' \in [0,1]$, $\lambda_2' \geqslant \lambda_2$, such that a new optimal solution is obtained for that problem with $\lambda \in [\lambda_2, \lambda_2']$, when the entering arc is (k_2,l_2).

The procedure ends with an ND solution $Incumbent \neq (+\infty,+\infty)$, since for each ($\varepsilon$-const) problem there is at least one ND solution such that $f_2(x) \leqslant \varepsilon$.

3.3. Computing non-integer solutions

The main feature of this branch-and-bound algorithm is related to the branching variable x_{kl} which is the entering variable that allows a shift from $STS^{(k1)}$ to $STS^{(k2)}$ (Step 2). The optimal solution for problem (ε-const) is computed in Step 2 as $x^{(k')}=x^{(k_1)}+\Delta_1 d$, where d is the solution with the direction of the line $x^{(k_1)}x^{(k_2)}$. By construction, $x^{(k_1)}$ and $x^{(k_2)}$, are such that $\varepsilon \in [f_1(x^{(k_1)}), f_2(x^{(k_2)})]$. Since f_2 is a linear function it is known that there is a solution $x^{(k')}$ in the line segment $[x^{(k_1)}x^{(k_2)}]$ such that $f_2(x^{(k')})=\varepsilon$. Each point of $[x^{(k_1)}x^{(k_2)}]$ can be written as $x^{(k_1)}+\Delta_1 d$, $\Delta_1 \in [0,\Delta]$ being Δ the value in step 2. The value of Δ_1 corresponding to the point $x^{(k')}$ can be obtained considering the similarity between the two triangles $[ACE]$ and $[BCD]$ in Fig. 1 and the proportionality between the length of their sides: $\overline{EC}/\overline{AC} = \overline{DC}/\overline{BC}$. Note that the point $x^{(k')}$ could also be $x^{(k_2)}+\Delta_1 d'$, where $d'=(x^{(k_1)}-x^{(k_2)})/\Delta$. The quantity Δ_1 is now obtained considering the similarity between the two triangles $[ACE]$ and $[ABF]$.

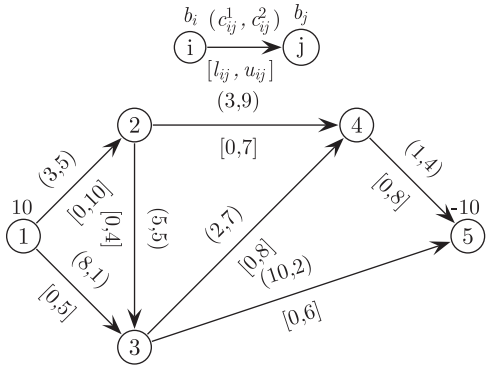

Fig. 2. A bi-objective example.

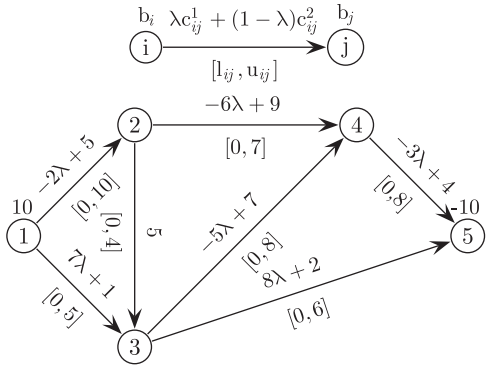

Fig. 3. Parametric problem.

3.4. The correctness of the algorithm

To prove the correctness of Algorithm 2 we have to show that all the ND solutions will be found, and that none of such solutions is a dominated one.

Proposition 3.1. *At termination of Algorithm* 2, *the solutions to the set NDc are all ND and any feasible solution not in NDc is dominated.*

Proof. All the solutions in *NDc* are non-dominated, since the first solution found is a lexicographically minimum solution and the remaining solutions in *NDc* are obtained according to Theorem 1.2, therefore they are ND solutions.

Suppose that there is an ND solution y such that $y \notin NDc$. Then there is no solution y' such that $y'_1 \leqslant y_1$ and $y'_2 < y_2$ or $y'_1 < y_1$ and $y'_2 \leqslant y_2$. Let $y^{(k+1)}$ denote the solution obtained from $y^{(k)}$ in Step 3 of Algorithm 2 such that $y_2^{(k+1)} \leqslant y_2 < y_2^{(k)}$. Since $y_1^{(k+1)} \leqslant y_1$ and $y_2^{(k+1)} \leqslant y_2$, then $y \geqq y^{(k+1)}$ and $y \neq y^{(k+1)}$, i.e., y is not an ND solution, thus leading to a contradiction. □

4. An illustrative example

This section illustrates the way how the proposed algorithm works. Consider the BOINF example in Fig. 2. This example has 93 feasible solutions (see [7]). Among the 93 solutions only 10 are efficient. These 10 efficient solutions correspond one to one to 10 ND solutions. Fig. 8 presents all the feasible solutions; ND solutions are represented by a circle.

When running Algorithm 2, the first current set of ND solutions is $NDc = \{\}$. Then, $y^{(0)}$ is found by solving the problem of Fig. 3 with $\lambda \in]0,1[$ close to 1. The solution $x^{(0)} = (7,3,0,7,1,2,8)$ is obtained with $y^{(0)} = F(x^{(0)}) = (96,144) = y^{48}$. The optimal value for $\min_{x \in X} f_2(x)$ is $y_2^* = 99$ (see Fig. 4). The current set of ND solutions is now $NDc = \{\} \cup \{y^{48}\} = \{y^{48}\}$.

Problem (ε-const) is solved by using the procedure described above and $\varepsilon = 143.5$. The choice of ε might have been different. We know that any real number between 143 and 144 will do, since all the components of any feasible solution in Y^I are integer. The ND solution $y^{27} = (100,138)$ is obtained. We have now $NDc = \{y^{27}, y^{48}\}$.

Next, we need to solve problem (ε-const). Let us set $\varepsilon = 137.5$. The branch-and-bound tree in Fig. 7 contains all the solutions visited before reaching the optimal one. The iterations are as follows.

It 1 1. Solve problem (ε-const) by first considering problem (K).

$$\begin{aligned} \min \quad & F(x) = (f_1(x), f_2(x)) \\ \text{s.t.} \quad & x \in X^I \end{aligned}$$

where $f_1(x) = 3x_{12} + 8x_{13} + 5x_{23} + 3x_{24} + 2x_{34} + 10x_{35} + x_{45}$, $f_2(x) = 5x_{12} + x_{13} + 5x_{23} + 9x_{24} + 7x_{34} + 2x_{35} + 4x_{45}$ and $X^I = \{(x_{12}, x_{13}, x_{23}, x_{24}, x_{34}, x_{35}, x_{45}) \in \mathbb{N}_0^7 : x_{12} + x_{13} = 10, -x_{12} + x_{23} + x_{24} = 0, -x_{13} - x_{23} + x_{34} + x_{35} = 0, -x_{24} - x_{34} + x_{45} = 0, -x_{35} - x_{45} = -10, x_{12} \leqslant 10, x_{13} \leqslant 5, x_{23} \leqslant 4, x_{24} \leqslant 7, x_{34} \leqslant 8, x_{35} \leqslant 6, x_{45} \leqslant 8\}$.
Set $W = \{\}$, *Incumbent* $= (+\infty, +\infty)$.

2. STS^{48} and STS^4 are supported efficient adjacent STSs such that $y_2^4 < 137.5 \leqslant y_2^{48}$. Next we describe how to get STS^4.
Solution $x^{(0)} = x^{48}$ is an optimal one for the parametric problem with $\frac{3}{5} \leqslant \lambda \leqslant 1$. Consider the STS, STS^{48}, associated with this solution (see Fig. 4(a), the arcs in $\mathcal{T}$ are in bold). When $\lambda = \frac{3}{5}$, a new alternative solution is obtained inserting arc $(2,4)$ into the tree $\mathcal{T}$ through a simplex iteration (Fig. 5). This leads to a new solution $x^4 = (5,5,0,5,3,2,8)$ and a new STS, STS^4 and $y^4 = F(x^4) = (104,132)$. STS^{48} and STS^4 are adjacent STSs such that $y_2^4 < 137.5 \leqslant y_2^{48}$.
The arc $(1,3)$ is the entering arc that permits to move from STS^4 to the adjacent STS^{48}. The point $y^{(k')}$ is the intersection of lines $y_2 = \varepsilon$ and $y^4 y^{48}$. We have $y_1^{(k')} = f_1(x^{(k')}) = 100.33333$, where $x^{(k')} = x^4 + \Delta_1 d = (5.91667, 4.08333, 0, 5.91667, 2.08333, 2, 8)$ with $\Delta_1 = 2 \times (137.5 - 132)/(144 - 132) = 0.916667$ and $d = (1,-1,0,1,-1,0,0)$.
The feasible set $X^{(K)}$ is partitioned into two subsets: $X^{(k_1)} = \{x \in X^{(k)} : x_{13} \leqslant 4\}$ and $X^{(k_2)} = \{x \in X^{(k)} : x_{13} \geqslant 5\}$.
Consider the two new problems:

$$\begin{aligned} \min \quad & F(x) \\ \text{s.t.} \quad & x \in X^{(K_1)} \end{aligned} \qquad (B)$$

$$\begin{aligned} \min \quad & F(x) \\ \text{s.t.} \quad & x \in X^{(K_2)} \end{aligned} \qquad (C)$$

Set $W = W \cup \{B, C\} = \{B, C\}$. STS^{48} is a supported efficient STS for problem B and STS^4 is a supported efficient STS for problem C. This information will be used later.

3. Problem $(K) = B$ is now considered (see Fig. 6 (B)). $W = \{C\}$.

It 2 2. STS^{48} is a supported efficient solution for problem B such that $y_2^{48} > \varepsilon = 137.5$. A simplex iteration leads to the supported efficient adjacent STS^{27} with $y_2^{27} > 137.5$. This solution has STS^{25} as supported efficient adjacent and $y_2^{25} < 137.5$. Thus the $STS^{(k_1)} = STS^{25}$ and $STS^{(k_2)} = STS^{27}$ are considered.
We have $\Delta_1 = 0.055556$, $d = (0,0,0,0,-1,1,-1)$ with $x^{(k')} = (6,4,0,6,1.94444,2.05556,7.94444)$, and $(k,l) = (4,5)$

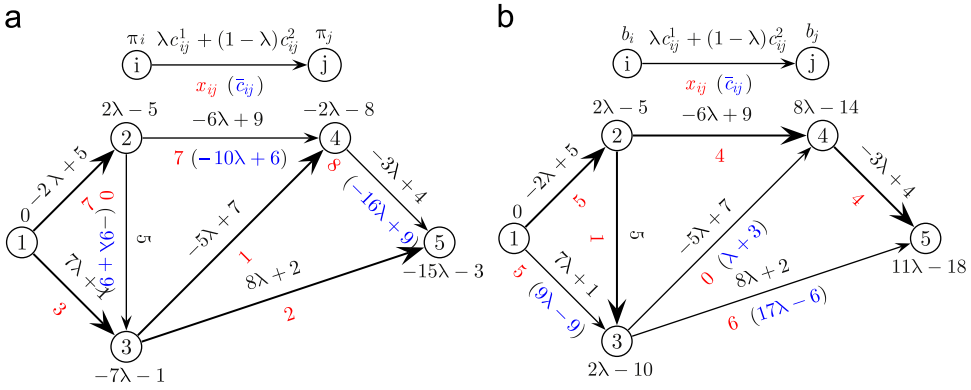

Fig. 4. Solutions: (a) $x^{(0)} = x^{48}$; (b) x^5.

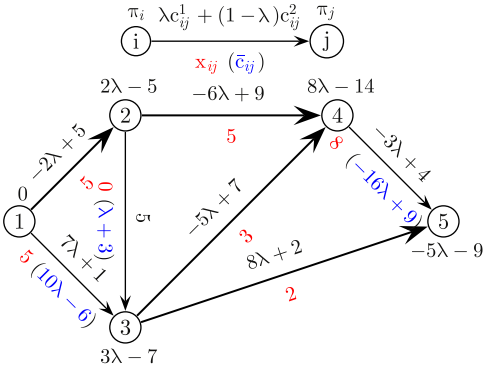

Fig. 5. Solution x^4.

is the entering arc allowing to move from STS^{27} to the adjacent STS^{25}. Problem (K) is partitioned into,

$$\begin{aligned} \min\quad & F(x) \\ \text{s.t.}\quad & x \in X^{(K)} \\ & x_{45} \leqslant 7 \end{aligned} \qquad (D)$$

$$\begin{aligned} \min\quad & F(x) \\ \text{s.t.}\quad & x \in X^{(K)} \\ & x_{45} = 8 \end{aligned} \qquad (E)$$

$W = \{C, D, E\}$

3. $W \neq \{\}$, problem D is considered (see Fig. 6(D)). $W = \{C, E\}$.

It 3 2. $STS^{(k_2)}$ does not exist and $STS^{(k_1)} = STS^{47}$. The value $y_2^{47} = 135 < \varepsilon$ and the *Incumbent* $= (+\infty, +\infty)$ thus we set *Incumbent* $= y^{47} = (103, 135)$.

3. $W \neq \{\}$, problem E is considered (see Fig. 6(E)). $W = \{C\}$.

It 4 2. $STS^{(k_2)}$ does not exist and $STS^{(k_1)} = STS^{27}$. The value $y_2^{27} = 138 > \varepsilon$.

3. $W \neq \{\}$, problem C is considered (see Fig. 6(C)). $W = \{\}$.

It 5 2. $STS^{(k_2)}$ does not exist and $STS^{(k_1)} = STS^4$. The value $y_2^4 = 132 < \varepsilon$. The *Incumbent* $= y^{47}$ and $y_1^4 = 104 > y_1^{47} = 103$.

3. $W = \{\}$, STOP. The ND solution y^{47} is the solution to the current problem (see the branch and bound tree of Fig. 7).

The set of ND solutions is updated $NDc = \{y^{27}, y^{47}, y^{48}\}$.

Since $y_2^{47} \neq y_2^*$ the algorithm continues. It stops when the ND solution y^5 is obtained (see Fig. 8).

5. Computational experiments

The algorithm presented in this paper was implemented and run for a large number of BOINF instances (more than 2160 instances). The aim of this section is to report the results for a better understanding of the algorithm behavior.

The algorithm has been implemented using the C programming language. The experiments were done in a Personal Computer equipped with an Intel Pentium processor 2.5 GHz with 3 GB of RAM, and runs under OS X operating system.

The computational experiments were designed on the basis of a set of 30 instances for each problem type. The choice of 30 instances is due to the practice adopted by statisticians that considered, in general, that a sample of a size lower than 30 is rather small for deriving statistical meaningful conclusions. Each instance was generated by using the NETGEN network generator [30] after some changes for generating BOINF instances. This aspect as well the use of appropriate based techniques are pointed out by Coffin and Saltzman [31] for the analysis of the data sets used in the computational experiments of algorithms. These authors also stated that the value of statistical analysis is important for gaining insight and increasing the explanation power of the behavior of the algorithms. As for the sample size it is very important to use enough data. Otherwise, the hypothesis test will not explain in a meaningful way the results. In our statistical analysis we apply a multiple linear regression model that was solved by the SPSS software version 17.0 [32].

First, we solved two sets of 30 instances, one set with 40 nodes and 500 arcs, and another set with 50 nodes and 870 arcs. The average number of ND solutions was 757.7 and 1031, respectively, and the average CPU time was 202.4 and 14 047.7, respectively.

Then, we solved a set of 72 types of instances built from the systematic scheme (Fig. 9) that was suggested by one of the referees.

The parameters that allowed the generation of each instance are the number of nodes (m), the number of arcs (n), the greatest cost value (C) (being the integer coefficients of the two objectives randomly generated within the range $[0, C]$), the greatest capacity of each arc (U) (being the upper bound values of each arc (i,j) randomly generated within the range $[0, U]$), and the total supply available in the network ($\sum$). We considered instances with 20 nodes and 40, 80, 120 and 156 arcs (156 was the maximum number of arcs allowed by NETGEN when the number of nodes was equal to 20), and instances with 30 nodes and 60, 120, 180, 240 and 306 arcs (306 was the maximum number of arcs allowed by NETGEN when the number of nodes was fixed at 30). The combinations of U, C and $\sum$ were chosen according to the scheme of Fig. 9 (5% of the arcs have the maximum cost). For each instance, two seed numbers (required by the modified NETGEN generator) were randomly generated. The first

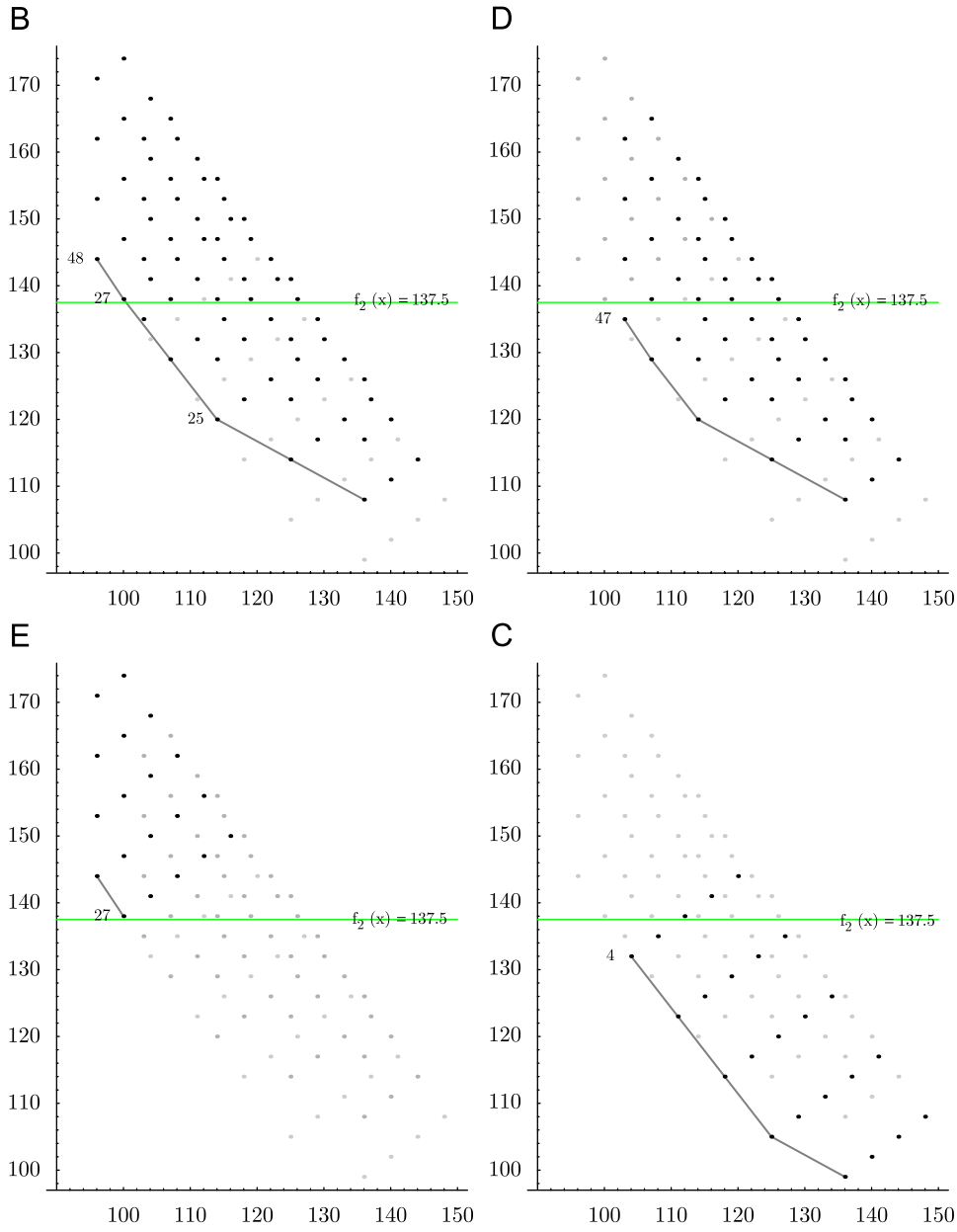

Fig. 6. Feasible set for problems *B*, *D*, *E* and *C*.

seed number was generated within the range [12345678, 15345678] and the second one within the range [56789123, 59789123]. There was no special reason for the selection of these ranges; others can of course be considered. The remaining parameters of NETGEN have been left constant. All the arcs have a lower capacity equal to zero and a finite upper capacity. Notice that despite the number of arcs considered, sometimes NETGEN adds one or more arcs so that the BOINF instances generated are always feasible.

The 72 types of instances were numbered according to Fig. 9. Thus instances of type 1 have $m = 20$, $n = 40$, $C = 100$, $\sum$=100 and $U = 20$. Instances of type 2 have the same parameters with the exception of $U = 60$. Instances of type 3 differs from $\sum$=500 and U=20. The enumeration continues in this way until instances of type 72 with $m = 30$, $n = 306$, $C = 1000$, $\sum$=500 and $U = 60$ are reached. The average (Av), the minimum (Min), the maximum (Max) and the median values of the 30 instances of each type are summarized in Table 1, for both the number of ND solutions and the CPU time.

The 30 hardest instances have $\sum$=500; a number of 30 nodes; only one instance has 240 arcs, the remaining ones have the greatest number of arcs, 306; 18 instances have $U = 20$ and 12 have $U = 60$; only one instance has $C = 100$, the remaining ones have $C = 1000$. The algorithm depends on the number of ND solutions since each one implies the resolution to a new ε-constraint problem. The determination of the optimal solution for each ε-constraint problem depends on the tree in the branch-and-bound method. These are the two main reasons to have a great or a small CPU time when running the algorithm to find all the ND solutions.

We have done some statistical analysis on the results (data) and the following comments can be presented. The statistical analysis were performed with SPSS [32]. The average number of ND solutions and the CPU time to find all these solutions grow when the upper limit of the cost values moves from 100 to 1000. The same happens when the total supply is changed from 100 to 500. When comparing instances with $C = 1000$ to instances with $C = 100$, the first ones

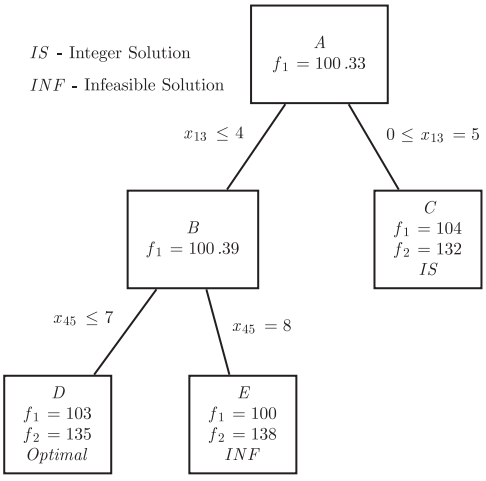

Fig. 7. Branch-and-bound iterations.

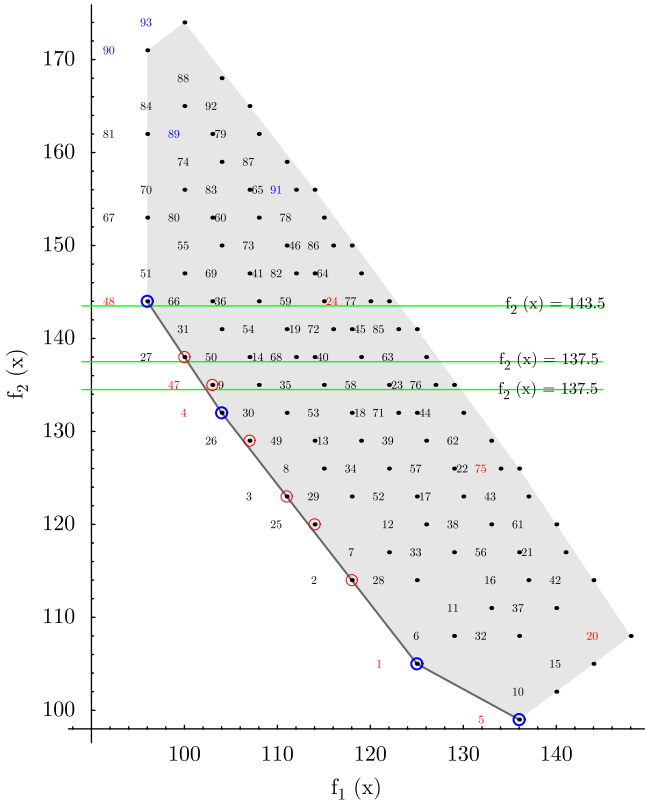

Fig. 8. Non-dominated solutions.

have, on average, 733.07 more ND solutions than the second ones. And the CPU time required to solve these instances is higher (more 368.32 seconds on average) than instances with $C = 100$. On average, instances with $\sum$ =500 have 995.42 more ND solutions than instances with $\sum$ =100. And the CPU time required to solve these instances is higher (more 1395.74 seconds, on average) than the CPU time for instances with $\sum$ =100. Instances with $U = 60$ have more 415.95 and spend less 42.01 seconds than instances with $U = 20$. On average, the instances considered have 69.62 more efficient solutions than ND solutions. The minimum value for this difference is 0 and the maximum is 6092.

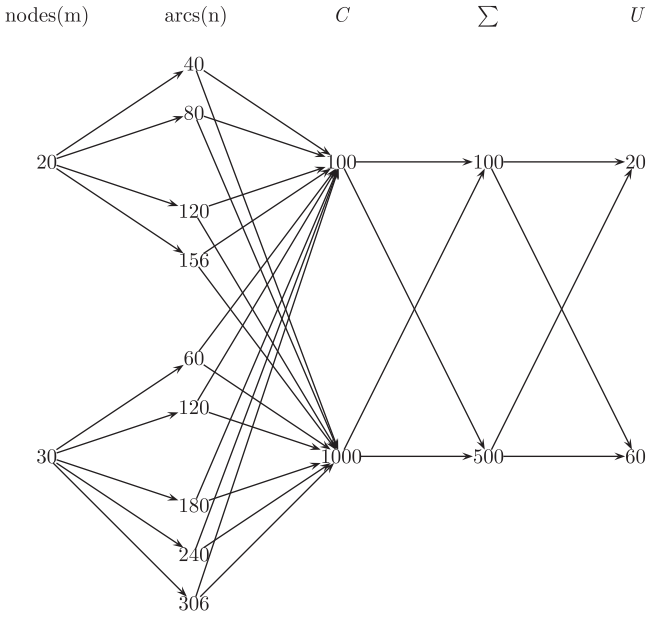

Fig. 9. Parameters problem scheme.

Now, consider the multiple regression model (one of the techniques presented in [31]),

$$nds = \beta_0 + \beta_1 \sum + \beta_2 n + \beta_3 C + \beta_4 m + \beta_5 U + \xi \quad (8)$$

where the variables *nds*, $\sum$, *n*, *C*, *m* and *U* represent the number of ND solutions, the total supply available in the network, the number of arcs, the greatest cost value, the number of nodes, and the greatest capacity of each arc in the network, respectively, and ξ is a random error term. The fitted regression model with the regression coefficients rounded to three decimal places is as follows:

$$nds \approx -2042.176 + 7.120 \sum + 16.400n + 0.815C - 54.066m + 10.401U$$

The standardized coefficients are 0.580, 0.543, 0.149, −0.109 and 0.085 for β_1, β_2, β_3, β_4 and β_5, respectively. The adjusted coefficient of multiple determination (adjusted R^2) is 0.613. We can say that 61.3% of the variability of *nds* is explained by the variables $\sum$, *n*, *C*, *m* and *U*. Moreover, $\sum$ and *n* are the variables that give rise to the biggest change of *nds*. The corresponding model only with two independent variables $\sum$ and *n* explains 57.4% of the variability of *nds*. We can see that the weak contribution of *U* for the variability of the number of ND solutions is negative, i.e., in our data (results) a larger value of *U* leads to a small number of ND solutions.

Now, consider the multiple regression model for the CPU time.

$$time = \beta_0 + \beta_1 \sum + \beta_2 n + \beta_3 C + \beta_4 m + \beta_5 U + \xi \quad (9)$$

where *time* is the CPU time. The statistical procedure stepwise deletes the terms $\beta_4 m$ and $\beta_5 U$ from the fitted regression model. This means that there is a statistical evidence to consider β_4 and β_5 equal to zero. The fitted regression model is then

$$time \approx -2011.512 + 2.489 \sum + 11.447n + 0.410C$$

The adjusted coefficient of multiple determination is 0.406 and the standardized coefficients are 0.296, 0.554 and 0.110 for β_1, β_2 and β_3, respectively. Therefore, we can say that 40.6% of the variability of the variable time is explained by the variables $\sum$, *n* and *C*. Moreover, the variable that gives rise to a greater variability of time is the number of arcs *n*.

Table 1
Computational results

N	ND solutions				CPU time (s)			
	Av	Min	Max	Median	Av	Min	Max	Median
1	113.57	5	391	87.50	0.29	0.011	1.178	0.18
2	97.80	19	256	85.00	0.21	0.044	0.642	0.16
3	575.43	150	1789	459.50	1.65	0.226	6.424	1.11
4	604.57	38	1942	559.00	1.65	0.062	4.9	1.33
5	115.17	12	437	90.00	0.67	0.024	1.764	0.56
6	101.33	2	244	88.50	0.58	0.013	2.082	0.46
7	576.87	132	2318	394.50	3.39	0.65	14.343	2.42
8	870.30	295	2775	706.50	4.66	1.287	23.146	3.19
9	294.47	52	646	268.00	3.64	0.277	13.892	3.08
10	336.83	104	791	304.50	4.18	0.532	22.523	3.17
11	1300.23	227	2687	1108.00	18.53	0.971	47.354	15.49
12	1763.80	190	4698	1834.50	29.03	1.352	86.254	28.96
13	351.60	92	1611	269.00	8.66	1.444	63.709	4.45
14	448.93	167	973	398.00	8.61	1.166	24.749	6.64
15	1707.73	600	7355	1373.00	37.34	5.539	120.171	24.95
16	3580.73	911	17 197	2425.50	82.16	11.66	422.525	52.53
17	523.03	133	975	506.00	18.18	2.161	34.621	18.20
18	532.07	228	910	519.50	17.01	3.26	43.568	16.86
19	1767.83	649	3413	1785.00	80.10	11.513	354.267	75.31
20	3179.37	549	7793	2873.00	134.40	6.033	286.355	126.93
21	615.03	241	1347	576.00	30.36	7.503	72.652	27.17
22	702.53	292	1693	611.50	31.18	6.913	80.043	24.83
23	3024.07	882	6687	2740.50	175.28	29.786	428.957	147.74
24	4178.90	1512	9231	3695.00	239.18	43.543	713.566	215.07
25	764.70	423	1544	735.00	61.93	20.242	127.17	55.47
26	754.50	397	1554	704.50	50.35	20.617	104.27	46.34
27	2983.47	1240	5914	3077.00	333.05	65.309	1161.936	283.10
28	4205.43	2104	6853	4189.50	480.41	125.621	1186.836	440.43
29	855.23	322	2061	813.00	80.20	23.48	205.406	73.54
30	885.70	279	1774	875.00	80.83	21.375	182.449	75.88
31	4617.73	1874	10 112	4181.50	625.66	153.453	2456.594	503.18
32	5817.00	2336	13 229	5475.00	673.65	144.211	2312.055	630.70
33	91.27	25	325	78.50	0.39	0.053	1.861	0.31
34	93.10	19	248	80.50	0.44	0.042	1.663	0.37
35	502.70	63	1679	386.50	2.98	0.121	28.739	1.63
36	747.37	92	3485	656.00	4.02	0.204	23.831	2.71
37	107.63	9	331	81.50	0.89	0.036	3.83	0.69
38	147.83	42	565	113.50	1.27	0.423	3.648	0.97
39	573.90	32	1870	485.50	4.59	0.242	14.244	4.07
40	614.27	24	1480	521.00	4.84	0.208	16.537	3.85
41	36.00	13	102	27.00	0.66	0.119	4.38	0.45
42	338.30	27	780	329.00	11.65	0.22	38.808	9.25
43	1675.30	362	3401	1651.50	71.90	6.869	162.427	55.18
44	2207.47	925	4910	1940.00	98.76	12.037	494.366	69.69
45	403.70	150	976	342.00	20.20	3.309	54.819	17.75
46	386.77	150	696	371.00	19.55	4.258	44.236	17.73
47	2297.20	681	5100	2066.00	130.22	18.566	326.761	103.84
48	3597.90	1024	8888	3300.50	202.30	36.978	707.646	139.90
49	542.17	206	953	541.50	64.49	11.656	140.511	58.76
50	571.13	221	992	588.00	61.32	15.391	129.027	62.16
51	3234.20	1617	6183	2899.00	539.51	170.234	1386.575	468.07
52	3836.90	2048	8772	3608.50	550.64	181.945	1805.395	439.72
53	613.43	185	1294	573.00	87.67	16.441	174.184	81.46
54	679.43	373	1287	648.50	108.48	34.969	269.078	98.78
55	3929.47	2158	7510	3423.00	799.14	284.926	2000.449	615.69
56	4858.10	1852	9425	4483.00	878.01	163.641	2547.984	798.18
57	719.60	307	1126	677.50	193.94	74.234	358.977	173.66
58	733.07	403	1104	695.50	194.02	80.484	353.578	186.43
59	4563.30	2486	6643	4695.00	1892.74	505.879	3388.99	1911.86
60	4643.60	2310	9170	4859.00	1587.67	589.191	3375.016	1537.61
61	867.97	445	1502	887.50	287.46	85.836	549.68	278.50
62	897.50	416	1544	862.00	298.17	116.031	726.563	274.47
63	6386.53	3797	10 619	6304.50	3092.06	1287.125	5970.688	2762.59
64	7568.33	4506	13 667	7083.50	3046.85	948.422	9036.875	2625.93
65	928.70	503	1348	977.50	513.91	195.721	1263.803	549.12
66	974.50	523	1618	979.00	509.94	248.315	1127.118	485.96
67	5354.77	3120	8007	5502.50	4528.82	2135.723	9163.605	4441.91
68	5560.97	3324	8349	5262.50	3629.21	1369.25	7638.859	3412.84
69	1216.00	663	1798	1189.00	801.09	362.969	1205.906	807.34
70	1246.93	565	1973	1172.00	831.46	260.125	1551.906	683.16
71	8629.67	3819	16 374	8370.50	8566.01	2786.063	20 199.67	8127.72
72	10 070.43	5117	17 127	9983.00	7688.51	2994.73	15 212.34	6925.04

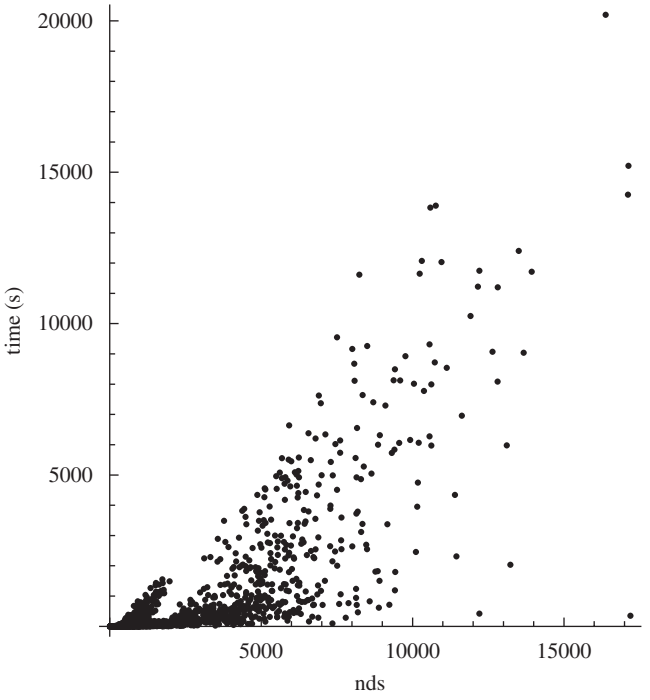

Fig. 10. Scatter diagram of the number of non-dominated solutions versus CPU time.

Considering the dependent variable as the CPU time and the independent variable as *nds* (see Fig. 10), we can see that 60.1% of the variability of the CPU time is due to the number of ND solutions.

Using the assembled data we can say that the variability of the CPU time spent to find all the ND solutions is mainly due to the amount of ND solutions. The larger part of the variability of this number is explained by the total supply available in the network, the number of arcs and the greatest cost value.

6. Conclusions

This paper presented the main features, implementation issues, theoretical bases and computational experiments of a new method for finding all the ND solutions for BOINF problems. The main characteristic of the algorithm is related to the way the feasible space is exploited without destroying the network structure of the problems. The results presented show that the algorithm cannot be applied to large size instances. Even for the instances solved, the number of ND solutions is rather big. However, the proposed method appears to be able to find both ND solutions and efficient solutions for small and medium size instances in a small amount of CPU time. An improved version of the network simplex algorithm will certainly make possible to improve the quality of the results in terms of CPU time and the size of the problems. This should be considered as an avenue for future development. The method is also of great interest, since the initial searching region of ND solutions can be broken into small regions, exploring only the desirable regions, enable its use with interactive methods helping the choice of the best solution according to the preferences of a decision maker. Another important issue is to extend the method to more than two objectives.

Acknowledgments

The authors acknowledge the Editor and the two anonymous referees for their valuable comments, remarks and suggestions. The authors research also partially benefited from the Cost Action 0602 on Algorithmic Decision Theory.

References

[1] Ahuja R, Magnanti TL, Orlin J. Network flows: theory, algorithms, and applications. New Jersey: Prentice-Hall; 1993.
[2] Lee H, Pulat PS. Bicriteria network flow problems: continuous case. European Journal of Operational Research 1991;51:119–26.
[3] Malhotra R, Puri MC. Bi-criteria network problem. Cahiers du CERO 1984;26(1):95–102.
[4] Pulat PS, Huarng F, Lee H. Efficient solutions for the bicriteria network flow problem. Computers & Operations Research 1992;19(7):649–55.
[5] Sedeño-Noda A, González-Martín C. The biobjective minimum cost flow problem. European Journal of Operational Research 2000;124:591–600.
[6] Sedeño-Noda A, González-Martín C. An alternative method to solve the biobjective minimum cost flow problem. Asia-Pacific Journal of Operational Research 2003;20:241–60.
[7] Figueira JR. On the integer bi-criteria network flow problem: a branch-and-bound approach. Research Report No. 3/2000 MSG, Faculdade de Economia da Universidade de Coimbra, Coimbra, Portugal [also in Cahiers du LAMSADE, 191, 2002].
[8] Figueira JR. Filtering algorithm for integer bi-criteria network flows. Manuscript, 2001.
[9] Gouveia MC. The bi-criteria network flow problem: a study of an algorithm for computing all the efficient solutions. Master's thesis, Faculdade de Economia da Universidade de Coimbra, Coimbra, Portugal, 2002 [in Portuguese].
[10] Huarng F, Pulat P, Ravindran A. An algorithm for bicriteria integer network flow problem. In: Proceedings of the 10th international conference on multiple criteria decision making, vol. 3, Taipei, Taiwan, 1992. Preprint, National Chiao Tung University, p. 305–18.
[11] Lee H, Pulat PS. Bicriteria network flow problems: integer case. European Journal of Operational Research 1993;66:148–57.
[12] Przybylski A, Gandibleux X, Ehrgott M. The biobjective integer minimum cost flow problem—incorrectness of Sedeño-Noda and González-Martín's algorithm. Computers & Operations Research 2006;33(5):1459–63.
[13] Sedeño-Noda A, González-Martín C. An algorithm for the biobjective integer minimum cost flow problem. Computers & Operations Research 2001;28:139–56.
[14] Hamacher HW, Pedersen CR, Ruzika S. Multiple objective minimum cost flow problems: a review. European Journal of Operational Research 2007;176:1404–22.
[15] Lee H. Integer solutions of bicriteria network flow problems. PhD thesis, University of Oklahoma, Norman, OK, 1989.
[16] Steuer RE. Multiple criteria optimization: theory, computation, and application. New York: Wiley; 1986.
[17] Zadeh N. A bad network for the simplex method and other minimum flow algorithms. Mathematical Programming 1973;5:255–66.
[18] Ruhe G. Complexity results for multicriterial and parametric network flows using a pathological graph of Zadeh. Zeitschrift für Operations Research 1988;32:9–27.
[19] Sedeño-Noda A, González-Martín C. An algorithm for the biobjective integer minimum cost flow problem. Computers & Operations Research 2001;28:139–56.
[20] Eusébio A, Figueira JR. On the computation of all supported efficient solutions in multi-objective. European Journal of Operational Research 2008; in press, doi:10.1016/j.ejor.2008.10.031.
[21] Bazaraa MS, Jarvis JJ, Sherali HD. Linear programming and network flows. New York: Wiley; 1977.
[22] Ehrgott M. Multicriteria optimization. 2nd ed., Berlin: Springer; 2005.
[23] Ulungu EL, Teghem J. The two phases method: an efficient procedure to solve bi-objective combinatorial optimization problems. Foundations of Computing & Decision Sciences 1995;20(2):149–65.
[24] Dauer JP. Analysis of the objective space in multiple objective linear programming. Journal of Optimization Theory and Applications 1987;126:579–93.
[25] Zeleny M. Linear multiobjective programming. In: Lecture notes in economics and mathematical systems, vol. 95. New York: Springer; 1972.
[26] Pareto V. Cours d'economie politique. Lausanne: Rouge; 1896.
[27] Chankong V, Haimes YY. Multiobjective decision maker: theory and methodology. New York: North-Holland; 1983.
[28] Grigoriadis MD. An efficient implementation of the network simplex method. Mathematical Programming Study 1986;26:83–111.
[29] Schechter M, Steuer RE. A correction to the connectedness of the Evans–Steuer algorithm of multiple objective linear programming. Foundations of Computing & Decision Sciences 2005;30(4):351–9.
[30] Klingman D, Napier A, Stutz J. NETGEN: a program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. Management Science 1974;20(5):814–21.
[31] Coffin M, Saltzman MJ. Statistical analysis of computational tests of algorithms and heuristics. INFORMS Journal on Computing 2000;12(1):24–44.
[32] SPSS. SPSS Statistics 17.0 for Microsoft Windows.
[33] Raith A, Ehrgott M. A two-phase algorithm for the biobjective integer minimum cost flow problem. Computers and Operations Research (2008), in press, doi:10.1016/j.cor.2008.06.008.