J Glob Optim (2007) 38:315–331
DOI 10.1007/s10898-006-9132-y

ORIGINAL PAPER

Obtaining an outer approximation of the efficient set of nonlinear biobjective problems

José Fernández · Boglárka Tóth

Received: 19 December 2005 / Accepted: 14 December 2006 / Published online: 1 February 2007

Abstract A new method for obtaining an outer approximation of the efficient set of nonlinear biobjective optimization problems is presented. It is based on the well known 'constraint method', and obtains a superset of the efficient set by computing the regions of δ-optimality of a finite number of single objective constraint problems. An actual implementation, which makes use of interval tools, shows the applicability of the method and the computational studies on a set of competitive location problems demonstrate its efficiency.

Keywords Nonlinear biobjective optimization · Efficient set · Outer approximation · Constraint method · Interval analysis · Competitive location

1 Introduction

Multiobjective optimization problems are ubiquitous. Many real-life problems require taking several conflicting points of view into account. In this paper, we restrict ourselves to the biobjective case, that is, to the problem

$$\begin{aligned} &\min\ \{f_1(x), f_2(x)\}, \\ &\text{s.t.}\quad x \in S \subseteq \mathbb{R}^n, \end{aligned} \tag{1}$$

An extended version of this paper, with more comments, details, examples, and references, can be found in Fernández and Tóth [5]. This paper has been supported by the Ministry of Education and Science of Spain under the research project SEJ2005-06273/ECON, in part financed by the European Regional Development Fund (ERDF).
Boglárka Tóth—On leave from the Research Group on Artificial Intelligence of the Hungarian Academy of Sciences and the University of Szeged, H-6720 Szeged, Aradi vértanúk tere 1., Hungary.

J. Fernández (✉) · B. Tóth
Department of Statistics and Operations Research, University of Murcia, Murcia, Spain
e-mail: josefdez@um.es

where $f_1, f_2\colon \mathbb{R}^n \longrightarrow \mathbb{R}$ are two real-valued functions. Let us denote by $f(x) = (f_1(x), f_2(x))$ the vector of objective functions, and by $Z = f(S)$ the image of the feasible region.

When dealing with multiobjective problems we need to clarify what 'solving' a problem means. Some widely known definitions to explain the concept of solution of (1) follow.

Definition 1 A feasible vector $x^* \in S$ is said to be *efficient* iff there does not exist another feasible vector $x \in S$ such that $f_i(x) \leq f_i(x^*)$ for all $i = 1, 2$, and $f_j(x) < f_j(x^*)$ for at least one index j. The set S_E of all the efficient points is called the *efficient set*.

Efficiency is defined in the decision space. The corresponding definition in the criterion space is as follows.

Definition 2 An objective vector $z^* = f(x^*) \in Z$ is said to be *nondominated* iff x^* is efficient. The set Z_N of all nondominated vectors is called the *nondominated set*.

In this paper, we assume that both S_E and Z_N are bounded. Another related concept widely used is weak efficiency.

Definition 3 A feasible vector $x^* \in S$ is said to be *weakly efficient* iff there does not exist another feasible vector $x \in S$ such that $f_i(x) < f_i(x^*)$ for $i = 1, 2$.

Ideally, solving (1) means obtaining the whole efficient set, that is, all the points which are efficient. That set might be described analytically as a closed formula, numerically as a set of points, or in mixed form as a parameterized set of points. Unfortunately, as pointed out in Ruzika and Wiecek [19], for the majority of multi-objective optimization problems, it is not easy to obtain such a description, since the efficient set includes typically a very large number or infinite number of points. The methods proposed in the literature with that purpose are specialized either for particular problems (for instance, in Nickel and Puerto [16] it is shown how to obtain the whole efficient set of some location problems) or for a particular class of multiobjective problems (for instance, the multiobjective simplex method for the linear case [7]). To the extent of our knowledge, only one general method (see [6]) has been proposed in the literature with that purpose for the general nonlinear biobjective problem (1). The reason for this lack of methods is that even obtaining a single efficient point of a nonlinear biobjective problem can be a difficult task. That is why some authors have proposed to present to the decision-maker a 'representative set' of efficient points which suitably represent the whole efficient set (either by modifying the definition of efficiency [1] or by selecting a finite set of efficient points with the criteria of coverage, uniformity, and cardinality as quality measures [20]) or an 'approximation' of the efficient set by means of sets with a simpler structure (see [19] for a survey of methods with that aim).

The approach in this paper is similar to that in Fernández et al. [6]: we offer a *superset* which tightly contains the complete efficient set. However, the method presented in this paper is completely different from that in Fernández et al. [6], and what is more important, it is much faster (see Sect. 5.2). By drawing in the image space that superset the decision-maker can easily see the trade-off between the two objectives, i.e., how one objective improves as the other gets worse. Something similar can be done in the decision space, by drawing the superset in a color scale depending on the objective value of one of the objectives.

The paper is organized as follows. In the following section, we present the tools used to derive our method. It is in Sect. 3, where we introduce our constraint-like method, which provides a superset of the efficient set of (1). In Sect. 4, we detail how the constraint-like method can be carried out in practice using interval tools. Some computational studies are reported in Sect. 5. The paper ends with conclusions and points for future research.

2 Preliminaries

2.1 The constraint method

There is a variety of methods for finding efficient points of nonlinear multiobjective optimization problems (see [2,15] and the references therein), e.g., weighting method, lexicographic method, ..., but among them, only a few (e.g., the constraint method, reference point methods,...) are able to detect *all* nondominated points. Probably, the constraint method is the most famous among them. The rationale behind the constraint method is rather simple. One of the objective functions, say f_1, is selected to be minimized, whereas the other one, f_2, is converted into a constraint by setting an upper bound f_2^{ub} to it. The single objective problem to be solved, called *constraint problem*, is then

$$\begin{aligned} &\min f_1(x),\\ &\text{s.t.}\;\; f_2(x) \le f_2^{\text{ub}},\\ &\qquad y \in S. \end{aligned} \tag{2}$$

The goodness of the constraint method can be seen in the following theorems (for a proof, see for instance [15]).

Theorem 1 The solution of the constraint problem (2) is weakly efficient.

Theorem 2 A feasible vector $x^* \in S$ is efficient if and only if it is a solution of the constraint problems

$$\begin{aligned} &\min f_1(x), \\ &\text{s.t.}\;\; f_2(x) \le f_2(x^*), \\ &\qquad x \in S \end{aligned} \quad \text{and} \quad \begin{aligned} &\min f_2(x), \\ &\text{s.t.}\;\; f_1(x) \le f_1(x^*), \\ &\qquad x \in S. \end{aligned}$$

From the previous theorems it follows that it is possible to find *all* the efficient solutions of *any* biobjective optimization problem by the constraint method. However, we need to solve one or two problems to find one nondominated point, which means that if the nondominated set is not a discrete set (as it is usually the case in continuous multiobjective optimization) then the method is not practical for finding the *complete* efficient set.

2.2 Obtaining a region of δ-optimality

Consider a single-objective problem

$$\begin{aligned} &\min h(x),\\ &\text{s.t.}\;\; x \in Y \subseteq \mathbb{R}^n, \end{aligned} \tag{3}$$

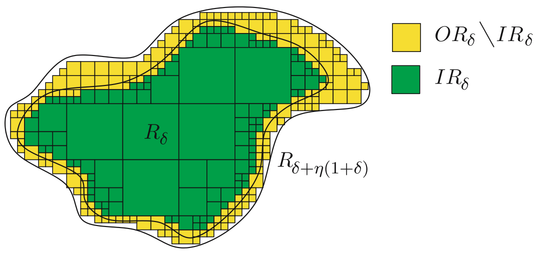


Fig. 1 Inner and outer approximation of R_δ

where h: $\mathbb{R}^n \longrightarrow \mathbb{R}$ is a real-valued function and Y is a general set defined by any kind of constraints. If we denote by h^* the optimal value of (3), the region of δ-optimality of (3) is the set

$$R_\delta = \{x \in Y : h(x) - h^* \leq \delta \cdot |h^*|\}.$$

The other tool that we need to derive the method of the next section is a procedure for obtaining the *region of* δ*-optimality* of a given problem. In Plastria [17], a branch-and-bound method for obtaining the region of δ-optimality of (3) is presented. It consists of two phases. The first one entails the determination of the optimal objective value of (3) up to a prespecified relative precision ϵ. The second phase consists of the determination of R_δ, up to a prespecified precision η. The output of the algorithm is two lists of subsets, LIR_δ and LOR_δ. The union of the subsets in the first list, $IR_\delta = \cup_{Q\in LIR_\delta} Q$, intersected with Y gives an *inner approximation* of R_δ (a subset of R_δ). The union of the subsets in both lists, $OR_\delta = \cup_{Q\in LIR_\delta \cup LOR_\delta} Q$, intersected with Y forms an *outer approximation* of R_δ (a superset of R_δ), guaranteed to lie entirely within $R_{\delta+\eta(1+\delta)}$ (see Fig. 1), i.e.,

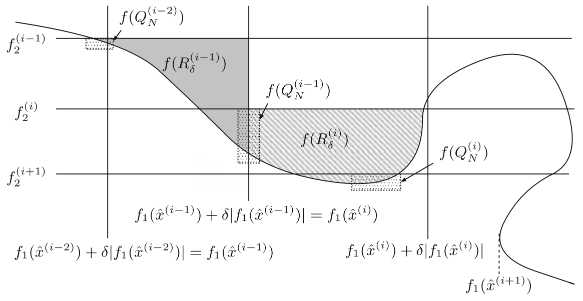
$$IR_\delta \cap Y \subseteq R_\delta \subseteq OR_\delta \cap Y \subseteq R_{\delta+\eta(1+\delta)}.$$

3 The constraint-like method

As explained above, with the classical constraint method we have to solve one (or two) constraint problems of the form (2) in order to be able to obtain a single non-dominated point, z^*. However, if in addition to solving (2) to optimality we compute its region of δ-optimality, then R_δ contains a 'portion' of the efficient set whose values in the criterion space are close to z^*. The idea of the constraint-like method to obtain a superset containing the whole efficient set is simply this: considering that Z_N is plotted in the criterion space, we sweep the nondominated set from (say) top to bottom by obtaining the regions of δ-optimality of a finite sequence of constraint problems, whose type is a modification of (2), by choosing appropriate upper bounds f_2^{ub} for the f_2 function. Next, we give the details.

The first constraint problem that we will consider, $(\bar{P}_0)$, is

$$\begin{aligned} &\min f_1(x),\\ &\text{s.t.}\quad x \in S. \end{aligned}$$



$f(Q_N^{(i-2)})$
$f_2^{(i-1)}$
$f(R_\delta^{(i-1)})$
$f(Q_N^{(i-1)})$
$f_2^{(i)}$
$f(R_\delta^{(i)})$
$f(Q_N^{(i)})$
$f_2^{(i+1)}$
$f_1(\hat{x}^{(i-1)}) + \delta|f_1(\hat{x}^{(i-1)})| = f_1(\hat{x}^{(i)})$
$f_1(\hat{x}^{(i-2)}) + \delta|f_1(\hat{x}^{(i-2)})| = f_1(\hat{x}^{(i-1)})$
$f_1(\hat{x}^{(i)}) + \delta|f_1(\hat{x}^{(i)})|$
$f_1(\hat{x}^{(i+1)})$

Fig. 2 Image space of a biobjective problem using MCLM. The gray region is the image of $R_\delta^{(i-1)}, f(R_\delta^{(i-1)})$, and the striped region is the image of $R_\delta^{(i)}, f(R_\delta^{(i)})$. $f(Q_N^{(i)})$ denotes the image of a subset $Q_N^{(i)}$ at which the minimum (4) or (5) is attained, i.e., a subset such that $f_2^{(i+1)} = \overline{F}_2(Q_N^{(i)})$

The remaining constraint problems that we will use are of the form

$$(\bar{P}_i) \qquad \begin{array}{ll} \min & f_1(x), \\ \text{s.t.} & f_2(x) \le f_2^{(i)}, \\ & f_1(x) \ge f_1(\hat{x}^{(i-1)}) + \delta|f_1(\hat{x}^{(i-1)})|, \\ & x \in S, \end{array}$$

where $f_2^{(i)}$ is a given constant defined below and $\hat{x}^{(i-1)}$ denotes an optimal solution of the previous problem $(\bar{P}_{i-1})$, $i \ge 1$. The feasible set of $(\bar{P}_i)$ is $Y^{(i)} = \{x \in S : f_2(x) \le f_2^{(i)}, f_1(x) \ge f_1(\hat{x}^{(i-1)}) + \delta|f_1(\hat{x}^{(i-1)})|\}$.

The constant $f_2^{(i)}$ is defined as follows. For $i = 0$ we set $f_2^{(0)} = f_2(\hat{x}^{(0)})$. Let $R_\delta^{(i)}$ be the region of δ-optimality of $(\bar{P}_i), i \ge 0$. Then, once we have solved problem $(\bar{P}_i)$ and have obtained an outer approximation $OR_\delta^{(i)}$ of $R_\delta^{(i)}$ with the help of the procedure mentioned in Sect. 2.2, the constant $f_2^{(i+1)}$ for the next problem $(\bar{P}_{i+1})$, is given by

$$f_2^{(i+1)} = \min\{f_2^{(i)}, \min\{\overline{F}_2(Q) : Q \in LIR_\delta^{(i)} \cup LOR_\delta^{(i)}, Q \cap R_\delta^{(i)} \neq \emptyset\}\}, \tag{4}$$

where Q denotes a subset of the output lists of the procedure described in Sect. 2.2, and $\overline{F}_2(Q)$ is an upper bound on all objective values of f_2 found within the subset Q (see Fig. 2). However, from a computational point of view, it can be better to set

$$f_2^{(i+1)} = \min\{f_2^{(i)}, \min\{\overline{F}_2(Q) : Q \in LIR_\delta^{(i)}, Q \cap Y^{(i)} \neq \emptyset\}\} \tag{5}$$

although this is a worse (higher) value than the one obtained with (4). Using (5) we only have to check whether a subset Q in $LIR_\delta^{(i)}$ contains at least one feasible point of $(\bar{P}_i)$. If so, we take that subset into account for calculating the minimum in (5).

Notice that whereas the constraint on f_2 in problem $(\bar{P}_i)$ forces the image of the feasible set of the problems to go down in the criterion space as i increases, the constraint on f_1 forces it to go to the right (see Fig. 2).

The method that we propose to obtain an outer approximation of the efficient set of (1) is the following (see Fig. 2):

Constraint-like method (MCLM)

```
1. $i \leftarrow 0$.
2. While $(\bar{P}_i)$ is feasible
   (a) Obtain an outer approximation $OR_\delta^{(i)}$ of the region $R_\delta^{(i)}$ of $\delta$-optimality of
       problem $(\bar{P}_i)$ using the procedure mentioned in Sect. 2.2.
```

   (b) Calculate $f_2^{(i+1)}$ as given by (4) or (5).
   (c) $i \leftarrow i+1$.
3. $\bigcup_{j=0}^{i-1} OR_\delta^{(j)}$ contains the efficient set of (1).

Theorem 3 Suppose that both the efficient set and the nondominated set of (1) are bounded. Suppose also that $f_1(\hat{x}^{(0)}) > 0$. Then MCLM obtains the complete efficient set of (1) in a finite number of steps.

Proof From Theorem 2, a necessary condition for a feasible vector to be efficient is that it must be a solution of a problem of the form

$$(P_{\text{aux}}) \quad \begin{array}{ll} \min & f_1(x), \\ \text{s.t.} & f_2(x) \leq f_2^{(\text{aux})}, \\ & x \in S. \end{array}$$

We shall proof that all the optimal solutions of problems of that type which are efficient lie in a region of δ-optimality of one of the problems of type $(\bar{P}_i)$ solved by MCLM. Let $\hat{x}^{(\text{aux})}$ be an optimal solution of (P_{aux}).

Notice that the sequence $\{f_1(\hat{x}^{(i)})\}$ is nondecreasing. Thus, one of the following three cases must happen:

A: $f_1(\hat{x}^{(0)}) \leq f_1(\hat{x}^{(\text{aux})}) \leq f_1(\hat{x}^{(0)}) + \delta|f_1(\hat{x}^{(0)})|$. It means that $\hat{x}^{(\text{aux})}$ is in the region of δ-optimality of $(\bar{P}_0)$.

B: There is an index i, $1 \leq i \leq last-1$, such that $f_1(\hat{x}^{(i-1)}) + \delta|f_1(\hat{x}^{(i-1)})| \leq f_1(\hat{x}^{(\text{aux})}) \leq f_1(\hat{x}^{(i)}) + \delta|f_1(\hat{x}^{(i)})|$ (last-1 is the index of the last feasible problem considered by the algorithm). Two subcases can happen:

   B.1: $f_2(\hat{x}^{(\text{aux})}) \leq f_2^{(i)}$. Then the point $\hat{x}^{(\text{aux})}$ is a feasible point of $(\bar{P}_i)$. In particular, $f_1(\hat{x}^{(\text{aux})}) \geq f_1(\hat{x}^{(i)})$. Furthermore, by assumption, $f_1(\hat{x}^{(\text{aux})}) \leq f_1(\hat{x}^{(i)}) + \delta|f_1(\hat{x}^{(i)})|$. Thus, $\hat{x}^{(\text{aux})}$ is in the region of δ-optimality of $(\bar{P}_i)$.

   B.2: $f_2(\hat{x}^{(\text{aux})}) > f_2^{(i)}$. Then, there must exist a point $\check{x} \in R_\delta^{(i-1)}$ such that $f_2(\check{x}) \leq f_2^{(i)} < f_2(\hat{x}^{(\text{aux})})$ and $f_1(\check{x}) \leq f_1(\hat{x}^{(i-1)}) + \delta|f_1(\hat{x}^{(i-1)})| \leq f_1(\hat{x}^{(\text{aux})})$. That is, $\check{x}$ dominates $\hat{x}^{(\text{aux})}$, i.e., $\hat{x}^{(\text{aux})}$ is not an efficient point for problem (1).

C: $f_1(\hat{x}^{(\text{aux})}) > f_1(\hat{x}^{(\text{last}-1)}) + \delta|f_1(\hat{x}^{(\text{last}-1)})|$. Two cases can happen:

   C.1: If $f_2(\hat{x}^{(\text{aux})}) > f_2^{(\text{last})}$ then $\hat{x}^{(\text{aux})}$ will not be an efficient point (similarly to subcase B.2).

   C.2: If $f_2(\hat{x}^{(\text{aux})}) \leq f_2^{(\text{last})}$ then $\hat{x}^{(\text{aux})}$ will be a feasible point of $(\bar{P}_{\text{last}})$, but this is a contradiction, since $(\bar{P}_{\text{last}})$ is infeasible (it was the problem provoking the termination of the algorithm).

Notice that $f_1(\hat{x}^{(\text{aux})}) < f_1(\hat{x}^{(0)})$ cannot happen, since the feasible set of $(\bar{P}_0)$ contains the feasible set of (P_{aux}).

Thus, in any case, if $\hat{x}^{(\text{aux})}$ is an efficient point, then it lies in the region of δ-optimality of one of the problems solved by the algorithm.

To prove that the algorithm stops after a finite number of steps, just notice that with problem $(\bar{P}_i)$ we sweep $\delta|f_1(\hat{x}^{(i)})|$ units length along the f_1 axis on Z_{N} in the criterion space. Thus, if we denote by $\hat{y}^{(0)}$ an optimal solution of problem

$$\begin{array}{ll} \min & f_2(x), \\ \text{s.t.} & x \in S \end{array} \tag{6}$$

we need at most

$$\frac{f_1(\hat{y}^{(0)}) - f_1(\hat{x}^{(0)})}{\delta f_1(\hat{x}^{(0)})}$$

problems to sweep to whole nondominated set. □

Remark 1 Notice that in addition to the constraint on f_2 employed in the classical constraint method, we have a second constraint on f_1. We need to add this second constraint, because otherwise, the algorithm may get stuck when $f_2^{(i+1)} = f_2^{(i)}$. This may happen when the nondominated set is not connected and the 'jump' (along the abscissa) is greater than $\delta|f_1(\hat{x}^{(i)})|$ or when there is a continuum of weakly efficient points with the same f_2-value (i.e., in the image space the weakly efficient points form a segment parallel to the axis, the length of that segment being greater than $\delta|f_1(\hat{x}^{(i)})|$).

Remark 2 The constraint-like method obtains a superset of the efficient set S_{E} which maps into a superset of the nondominated set Z_{N}, which may be made as tight as required by reducing the value of δ (and the tolerances used in the procedure obtaining the regions of δ-optimality): the smaller the value of δ, the better the quality of the approximation, but also the higher the number of subproblems to be solved.

Remark 3 Problem $(\bar{P}_i)$ can be rewritten as

$$(\tilde{P}_i) \quad \begin{array}{ll} \min & f_1(x), \\ \text{s.t.} & f_2(x) \le f_2^{(i)}, \\ & x \in S \backslash \cup_{j=0}^{i-1} OR_\delta^{(j)}, \end{array}$$

where instead of the constraint on f_1, we remove from S the outer approximations of the regions of δ-optimality of the problems already solved.

Remark 4 The condition $f_1(\hat{x}^{(0)}) > 0$ in Theorem 3 is not restrictive. If a function f_1 does not satisfy it, we can use instead, for instance, the function $\hat{f}_1(x) = f_1(x) + |f_1(\hat{x}^{(0)})| + 1$.

Remark 5 Notice that although the method that we have used to obtain the whole set R_δ when deriving MCLM is inspired by the method in Plastria [17], any other method which obtains the *complete* set R_δ (or an outer approximation of it) could serve. If we denote by NOR_δ the outer approximation (or exact representation) of R_δ obtained by any other method, the only thing that has to be changed when using it in MCLM is the computation of the bounds $f_2^{(i)}$. If we denote by q any point in NOR_δ, the bounds should be computed as

$$f_2^{(i+1)} = \min\{f_2^{(i)}, \min\{f_2(q) : q \in NOR_\delta^{(i)} \cap R_\delta^{(i)}\}.$$

In fact, (4) and (5) are surrogates for the previous formula.

4 Carrying out the method: an interval implementation

The constraint problems that we have to solve may be global optimization ones (they may have many local optima). Thus, we need to use global optimization techniques to cope with them. Furthermore, instead of merely solving the constraint problems, we must also obtain their region of δ-optimality. Among the global optimization techniques only a branch-and-bound scheme seems to be appropriate for our purposes, although the computation of bounds is a difficult task, too. In this paper, we present such a method, which has some similarities with the two-phase method mentioned in Sect. 2.2 and described in Plastria [17], although modified for our purposes and in a more general framework: *Interval Analysis* (for details on the topic, the interested reader is referred to Hansen and Walster, Kearfott, and Ratschek and Rokne [9,13,18]).

In what follows, boldface will denote intervals, lower case will be used for scalar quantities or vectors (vectors are then distinguished from components by use of subscripts), and upper case for matrices. Brackets $[\cdot]$ will delimit intervals, while parentheses $(\cdot)$ indicate vectors and matrices. Underlines will denote lower bounds of intervals and overlines give upper bounds of intervals. For example, we may have the interval vector $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^T$, where $\boldsymbol{x}_i = [\underline{x_i}, \overline{x_i}]$. The *width* of an interval $\boldsymbol{x}_i$ is denoted by $w(\boldsymbol{x}_i) = \overline{x_i} - \underline{x_i}$ whereas the width of an interval vector $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^T$ is to be understood as $w(\boldsymbol{x}) = \max\{w(\boldsymbol{x}_i) : i = 1, \ldots, n\}$. The midpoint of $\boldsymbol{x}$ will be denoted by $\mathrm{mid}\,\boldsymbol{x}$. The set of intervals will be denoted by $\mathbb{IR}$, and the set of n-dimensional interval vectors, also called *boxes*, by $\mathbb{IR}^n$.

We recall that a function $\boldsymbol{h} : \mathbb{IR}^n \to \mathbb{IR}$ is said to be an *inclusion function* of $h : \mathbb{R}^n \to \mathbb{R}$ provided $\{h(y) : y \in \boldsymbol{y}\} \subseteq \boldsymbol{h}(\boldsymbol{y})$ for all boxes $\boldsymbol{y} \subset \mathbb{IR}^n$ within the domain of h. For a function h predeclared in some programming language (like sin, exp, etc.,), it is not too difficult to obtain a *predeclared* inclusion function $\boldsymbol{h}$, since the monotonicity intervals of predeclared functions are well known and then we can take $\boldsymbol{h}(\boldsymbol{y}) = \{h(y) : y \in \boldsymbol{y}\}$ for any $\boldsymbol{y} \in \mathbb{IR}$ in the domain of h. For a general function $f(y), y \in \mathbb{R}^n$, several methods can be employed to obtain inclusion functions although how to find an inclusion function as good as possible, that is, producing bounds as tight as possible, is still an open question (see [21,22]). The easiest method to obtain an inclusion function is the *natural interval extension*, which is obtained by replacing each occurrence of the variable y with a box including it, $\boldsymbol{y}$, each occurrence of a predeclared function h by its corresponding inclusion function $\boldsymbol{h}$, and the real arithmetic operators by the corresponding interval operators. Another method to evaluate inclusion functions when $h(y)$ is differentiable is the *centered form*, given by $\boldsymbol{h}(\boldsymbol{y}) = \boldsymbol{h}(c) + (\boldsymbol{y} - c)^T \nabla \boldsymbol{h}(\boldsymbol{y})$ where c is any point of $\boldsymbol{y}$ (usually its midpoint) and $\nabla \boldsymbol{h}(\boldsymbol{y})$ an inclusion function of the gradient ∇h of h at $\boldsymbol{y}$ (usually obtained as the natural interval extension of ∇h). The centered form usually gives over small boxes tighter bounds as compared to the natural interval extension. However, the natural interval extension is often very useful because of its computational simplicity.

4.1 The algorithm

The implementation of MCLM is described in pseudo-code form in Algorithm 1 (the code is available upon request from the authors). We have considered the constraint problems to be written in the form of $(\tilde{P}_i)$. In this way, we only have to deal with the constraint on f_2, and not with the one on f_1 (the removal of the regions of δ-optimality of the previous problems is done easily in our implementation, see Step 35 of Phase 2 in Algorithm 2).

In Algorithm 1, $\boldsymbol{s}$ denotes a box containing the feasible set S and $\max f_j$ is the maximum f_j-value that any efficient point can take, $j = 1, 2$, that is, $(\max f_1, \max f_2)$ is the *nadir point* of (1). As we can see, to calculate $\max f_1$ we first solve the problem

(6) to optimality by calling Phase 1 of Algorithm 2 (see Step 3) with $\delta = 0$ and using f_2 instead of f_1, and then we compute $\max f_1 = \max\{\overline{f_1}(\boldsymbol{y}) : \boldsymbol{y} \in L_2\}$, where L_2 is the solution list of Phase 1. Something similar is done to calculate $\max f_2$ (Step 5), although this time we do not modify the value of δ, since $(\bar{P}_0)$ is the first problem for which we have to compute its region of δ-optimality. After that, the algorithm (Steps 6–9) keeps calling Algorithm 2, the main procedure which obtains the regions of δ-optimality of the constraint problems, until one of the problems is infeasible.

Algorithm 1 Constraint-Like Method

```
Input: $\boldsymbol{f}_1, \boldsymbol{f}_2, \boldsymbol{h}_l (l = 1, \dots, r), \boldsymbol{s}, \delta, \epsilon, \eta, \mu$,
Output: $L_{\rm sol}$,
1 $\max f_1 = \max f_2 = \infty$;
2 $\boldsymbol{s} \to L_1$;
3 Call Phase 1 using $f_2$ as $f_1$, and $\delta = 0$; $\max f_1 = \max\{\overline{f_1}(\boldsymbol{y}) : \boldsymbol{y} \in L_2\}$;
4 $\boldsymbol{s} \to L_1$;
5 Call Phase 1; $\max f_2 = \max\{\overline{f_2}(\boldsymbol{y}) : \boldsymbol{y} \in L_2\}$;
6 repeat
7     Phase 2,
8     Phase 1,
9 until Phase 1 terminates with no solution.
```

Algorithm 2 is the core of Algorithm 1. It allows to obtain the outer approximation of the region of δ-optimality of the constraint problems. The algorithm is inspired by the method in Plastria [17]. It consist of two phases. The aim of the first phase is to obtain the optimal value of the constraint problem (within a tolerance ϵ), whereas the aim of the second is to obtain its region of δ-optimality $R_\delta^{(i)}$ (within a tolerance η).

However, Algorithm 2 differs from the method in Plastria [17] in several points. First, the bounds are computed here with the help of interval analysis. Second, in addition to the 'feasibility test' (with two implementations, Feasible and Infeasible, in the code) and the 'δ-cut-off test' (CutOffTest in the code, a modification of the classical cut-off test [18], also used in Plastria [17]), we also use a variant of the 'δ-monotonicity test' for strictly feasible and undetermined boxes (MonoTest) [4], a 'pruning test' [4] (Prune in the code) applied to f_2 or to both f_1 and f_2, and a multiobjective cut-off test (which discards dominated boxes) [6].

A brief description of the discarding tests used follows.

Feasibility test: Let us suppose that S is given by $S = \{y \in \mathbb{R}^n : h_l(y) \leq 0, l = 1, \dots, r\}$. We say that a box $\boldsymbol{y}$ *certainly satisfies* the constraint $h_l(y) \leq 0$ if $\overline{\boldsymbol{h}_l}(\boldsymbol{y}) \leq 0$ and that $\boldsymbol{y}$ *does certainly not satisfy* it if $\underline{\boldsymbol{h}_l}(\boldsymbol{y}) > 0$. A box $\boldsymbol{y} \subseteq \boldsymbol{s}$ is said *certainly feasible* if it certainly satisfies all the constraints, *certainly infeasible* if it does certainly not satisfy at least one of the constraints, and *undetermined* otherwise. The 'Infeasible($\boldsymbol{y}$)' test is true when the box $\boldsymbol{y}$ is certainly infeasible, whereas the 'Feasible($\boldsymbol{y}$)' test is true when the box $\boldsymbol{y}$ is certainly feasible.

δ-Cut-off test: Every time a box $\boldsymbol{y}$ is chosen from the list L_1, and provided that its midpoint c is certainly feasible, we use $\overline{\boldsymbol{f}}_1(c)$ (previously computed to evaluate the centered form) to update (if possible, i.e., when $\overline{\boldsymbol{f}}_1(c) < \tilde{f}$) the best upper bound $\tilde{f}$ of the global minimum of the constraint problem (but see also the multi-objective cut-off test). If updated, then the 'CutOffTest($c, L_1 \cup L_2, L_3, \delta$)' sends to L_3 all the boxes $\boldsymbol{y}$ in L_1 and L_2 such that $\underline{\boldsymbol{f}_1}(\boldsymbol{y}) > \tilde{f}$ (since they cannot contain the optimal

Algorithm 2 Main procedure

Phase 1

```
Input: $f_1, f_2, \boldsymbol{h}_l, L_1, L_{PES}, \delta, \epsilon, \max f_1, \max f_2$,
Output: $L_2, L_3, L_4, L_{PES}, \tilde{f}$,
1  while ( $L_1 \neq \emptyset$ ) do
2    $L_1 \to \boldsymbol{y}$; $c \to \mathrm{mid}\,(\boldsymbol{y})$
3    Eval CenteredForm($f_1, \boldsymbol{y}, c$)
4    Eval CenteredForm($f_2, \boldsymbol{y}, c$)
5    if ($\underline{f_1}(\boldsymbol{y}) > \max f_1 \,||\, \underline{f_2}(\boldsymbol{y}) > \max f_2$)
6      continue
7    if Infeasible($\boldsymbol{y}$)
8      continue
9    if MultiCutOffTest($\boldsymbol{y}$))
10     continue
11   CutOffTest($c, L_1 \cup L_2, L_3, \tilde{f}, \delta$);
12   if ($\underline{f_1}(\boldsymbol{y}) > \tilde{f}$)
13     if ($\underline{f_1}(\boldsymbol{y}) \leq \tilde{f} * (1+\delta)$)
14       $\boldsymbol{y} \to L_3$; continue
15     else
16       $\boldsymbol{y} \to L_4$; continue
17   if MonoTest($\boldsymbol{y}, \boldsymbol{z}$)
18     $\boldsymbol{y} \to L_3$
19     $\boldsymbol{z} \to \boldsymbol{y}$
20   if ($\underline{f_1}(\boldsymbol{y}) + \epsilon|\underline{f_1}(\boldsymbol{y})| \geq \tilde{f} \,||\, w(\boldsymbol{y}) < \epsilon$)
21     $\boldsymbol{y} \to L_2$
22   else
23     Prune($\boldsymbol{y}, \tilde{f}, \epsilon, \delta$) $\to \boldsymbol{y}_1, \boldsymbol{y}_2$
24   for $i = 1, 2$ do
25     $\boldsymbol{f}_2(\boldsymbol{y}_i) \cap = \boldsymbol{f}_2(c) + (\boldsymbol{y}_i - c)^T \nabla \boldsymbol{f}_2(\boldsymbol{y})$
26     if Infeasible($\boldsymbol{y}_i$)
27       continue
28     Eval $\boldsymbol{f}_1(\boldsymbol{y}_i)$
29     $\boldsymbol{f}_1(\boldsymbol{y}_i) \cap = \boldsymbol{f}_1(c) + (\boldsymbol{y}_i - c)^T \nabla \boldsymbol{f}_1(\boldsymbol{y})$
30     if MultiCutOffTest($\boldsymbol{y}_i$)
31       continue
32     if ($\underline{f_1}(\boldsymbol{y}_i) > \tilde{f}$)
33       if ($\underline{f_1}(\boldsymbol{y}_i) \leq \tilde{f} * (1+\delta)$ )
34         $\boldsymbol{y}_i \to L_3$; continue
35       else
36         $\boldsymbol{y}_i \to L_4$; continue
37     if ($\underline{f_1}(\boldsymbol{y}_i) > \max f_1 \,||\, \underline{f_2}(\boldsymbol{y}_i) > \max f_2$)
38       continue
39     $\boldsymbol{y}_i \to L_1$
40   endfor
41 endwhile
```

Phase 2

```
Input: $f_1, f_2, \boldsymbol{h}_l, L_2, L_3, L_4, L_{PES}, \tilde{f}, \delta, \epsilon$,
    $\eta, \mu, \max f_1, \max f_2$,
Output: $L_1, L_{\mathrm{sol}}, L_{PES}$,
1  while ( $L_3 \neq \emptyset$) do
2    $L_3 \to \boldsymbol{y}$; $c \to \mathrm{mid}\,(\boldsymbol{y})$
3    Eval CenteredForm($f_1, \boldsymbol{y}, c$)
4    Eval CenteredForm($f_2, \boldsymbol{y}, c$)
5    if ($\underline{f_1}(\boldsymbol{y}) > \max f_1 \,||\, \underline{f_2}(\boldsymbol{y}) > \max f_2$)
6      continue
7    if Infeasible($\boldsymbol{y}$)
8      continue
9    if MultiCutOffTest($\boldsymbol{y}$)
10     continue
11   if ( $\underline{f_1}(\boldsymbol{y}) > \tilde{f} * (1+\delta)$ )
12     $\boldsymbol{y} \to L_4$; continue
13   if (Feasible($c$) and $\overline{f_2}(c) < f_2^{(i)}$ and
       $\overline{f_1}(c) < \tilde{f} * (1+\delta)$)
14     $f_2^{(i)} = \overline{f_2}(c)$
15   if ($\overline{f}_1(\boldsymbol{y}) < \tilde{f} * (1+\delta)(1+\eta)$ and
       (Feasible($\boldsymbol{y}$) $||w(\boldsymbol{f}_2(\boldsymbol{y})) < \mu||\, w(\boldsymbol{y}) < \epsilon$) )
16     $\boldsymbol{y}_i \to L_2$; continue
17   Prune($\boldsymbol{y}, \tilde{f}, \epsilon, \delta$) $\to \boldsymbol{y}_1, \boldsymbol{y}_2$
18   for $i = 1, 2$ do
19     $\boldsymbol{f}_2(\boldsymbol{y}_i) \cap = \boldsymbol{f}_2(c) + (\boldsymbol{y}_i - c)^T \nabla \boldsymbol{f}_2(\boldsymbol{y})$
20     if Infeasible($\boldsymbol{y}_i$)
21       continue
22     Eval $\boldsymbol{f}_1(\boldsymbol{y}_i)$
23     $\boldsymbol{f}_1(\boldsymbol{y}_i) \cap = \boldsymbol{f}_1(c) + (\boldsymbol{y}_i - c)^T \nabla \boldsymbol{f}_1(\boldsymbol{y})$
24     if MultiCutOffTest($\boldsymbol{y}_i$)
25       continue
26     if ( $\underline{f_1}(\boldsymbol{y}_i) \leq \tilde{f} * (1+\delta)$)
27       $\boldsymbol{y}_i \to L_3$; continue
28     else
29       $\boldsymbol{y}_i \to L_4$; continue
30     if ($\underline{f_1}(\boldsymbol{y}_i) > \max f_1 \,||\, \underline{f_2}(\boldsymbol{y}_i) > \max f_2$)
31       continue
32     $\boldsymbol{y}_i \to L_3$
33   endfor
34 endwhile
35 $L_2 \to L_{\mathrm{sol}}$; $L_4 \to L_1$
36 for $z^* \in L_{PES}$ do
37   if $z_2^* > f_2^{(i)}$
38     Remove $z^*$ from $L_{PES}$
```

value of the constraint problem), and then sends to L_4 all the boxes $\boldsymbol{y}$ in L_3 such that $\underline{f_1}(\boldsymbol{y}) > \tilde{f}(1+\delta)$ (since they cannot be in $R_\delta^{(i)}$).

Pruning test applied to $\boldsymbol{f_2}$: The pruning test was recently proposed in Martínez et al. [14], and modified in Fernández et al. [3]. It uses gradient information to determine regions in the actual box which cannot contain global optimizers. We briefly explain it using, not to complicate matters, a two-dimensional minimization problem, and we describe how to apply it along the y_1-direction. Consider a box $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2)$, and suppose that we know a lower bound for the value of the objective function h at $(\mathrm{mid}\,\boldsymbol{y}_1, \boldsymbol{y}_2)$, and also bounds for the gradient $\nabla \boldsymbol{h}(\boldsymbol{y})$ (see Fig. 3). Then a lower bounding function of h can be constructed as the planes in Fig. 3 (similarly to what

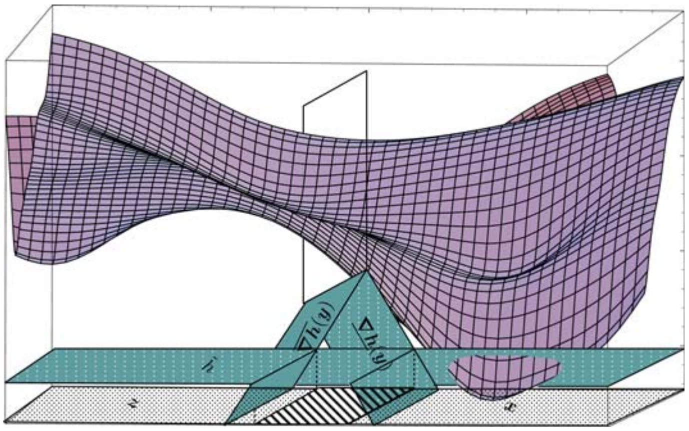
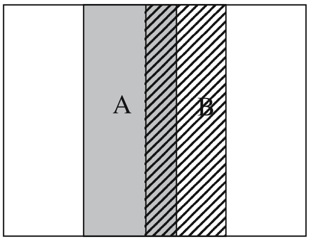


Fig. 3 Pruning method using the gradient

Fig. 4 Pruning applied to f_1 and f_2



is commonly done in Lipschitz optimization [10]). Then, using an upper bounding value $\tilde{h}$, the minimizer points in $\boldsymbol{y}$ can lie only in $\boldsymbol{x}$ and $\boldsymbol{z}$.

In particular, when applied to f_2 using $f_2^{(i)}$ as upper bounding value, it means that the feasible points in $\boldsymbol{y}$ can only lie in $\boldsymbol{x}$ or $\boldsymbol{z}$. Therefore, the other parts of $\boldsymbol{y}$ are not of interest neither for the current constraint problem nor for the rest of constraint problems (since in the following constraint problems the upper bound on f_2 will be smaller than or equal to $f_2^{(i)}$) and can be deleted.

This pruning test can be done for any coordinate direction of the box, generating one or two new subboxes. In this sense, it can be seen as a bisection method along the chosen coordinate. However, a coordinate j is selected only if $w(\boldsymbol{y}_j) > \epsilon$. In particular, we choose the coordinate direction j such that the corresponding part removed from $\boldsymbol{y}$ is the largest one.

Pruning test applied to f_1 and f_2: A similar process to the one described in the above pruning test can be done applying it to function f_1 considering $\tilde{f}(1+\delta)$ as upper bounding value. However, now, the parts of $\boldsymbol{y}$ which are not of interest (because their f_1 value is greater than $\tilde{f}(1+\delta)$) have to be sent to L_4, because they may be of interest for the next constraint problems.

In order to apply the pruning technique to both f_1 and f_2 within the same algorithm without generating too many boxes, we have used the strategy which we explain with the following example (see Fig. 4). Let us suppose that the gray region A can be deleted by pruning with f_2, while the striped area B can be cut by pruning with f_1. It is easy to see that cutting the region A is always a good decision, because we generate at most two new subboxes, and the deleted regions do not have to be taken into consideration anymore (they are not of interest for any of the remaining problems). However the region B\A has to be sent to L_4, if we decide to cut it. That is why we only cut it if its area is greater that 10% of the area of the original box. Furthermore, when B\A is not connected, only its greater part is sent to L_4 (provided that the previous condition holds). The selection of the coordinate direction to apply the test is done as in the previous test.

δ-monotonicity test (for strictly feasible and undetermined boxes): In Fernández et al. [6] a monotonicity test (for strictly feasible and undetermined boxes) is described to decide whether the objective function f_1 is strictly monotonous in a box, which allows either to discard the box or to reduce it to one of its facets. Since we are now computing $R_\delta^{(i)}$ and not solving the constraint problem till optimality, we cannot discard monotonous boxes. Instead, when MonoTest($\boldsymbol{y}, \boldsymbol{z}$) is true, i.e., when the objective function is monotonous at $\boldsymbol{y}$ ($0 \notin \nabla \boldsymbol{f}_1(\boldsymbol{y})$), the δ-monotonicity test sends the box $\boldsymbol{y}$ to L_3 (since it cannot contain the optimal value of the constraint problem), and follows the process with the facet $\boldsymbol{z}$ of $\boldsymbol{y}$ containing the best points: $\boldsymbol{z}$ cannot be discarded since it can lie on the border of the feasible set of the constraint problem). Note that this test is only useful in *Phase 1*.

Multiobjective cut-off test: This test was introduced in Fernández et al. [6]. Every time MultiCutOffTest($\boldsymbol{y}$) is applied to a box $\boldsymbol{y}$, and provided that its midpoint c (as a point interval) is certainly feasible, we compute $\overline{\boldsymbol{f}}(c) = (\overline{\boldsymbol{f}}_1(c), \overline{\boldsymbol{f}}_2(c))$ and update (if possible, i.e., if $\overline{\boldsymbol{f}}(c)$ is not dominated by any point in L_{PES}) the list L_{PES} of 'provisional' efficient solutions available so far. The test discards boxes whose points are not efficient, i.e., a box $\boldsymbol{y}$ is removed (and then the test is true) if $\underline{\boldsymbol{f}}(\boldsymbol{y}) > z^*$ for some $z^* \in L_{PES}$. When this test and the δ-cut-off test are used together in the same algorithm, then $\tilde{f}$ is updated in the δ-cut-off test only if $\overline{\boldsymbol{f}}(c)$ is not dominated by any point in L_{PES}. This is to avoid the possibility that the list L_1 of a constraint problem becomes empty, what may happen in some cases in which all its boxes are deleted by the multiobjective cut-off test or the δ-cut-off test.

A few comments on the Algorithm 2 are in order. The box $\boldsymbol{y}$ to be chosen from L_1 (Step 2 of Phase 1) and L_3 (Step 2 of Phase 2) is the one with the lowest $\underline{\boldsymbol{f}}_1(\boldsymbol{y})$. In Step 20 of Phase 1 (see also Step 15 of Phase 2) we send a box to L_2 if its width is less than a given tolerance. We have added this stopping rule because if the tolerances are chosen too small then, due to the overestimation of the inclusion functions, it may happen that the other stopping criteria are never fulfilled. On the other hand, in Step 15 of Phase 2, for sending a box $\boldsymbol{y}$ to L_2, in addition to the condition $\overline{\boldsymbol{f}}_1(\boldsymbol{y}) < \tilde{f}(1+\delta)(1+\eta)$ we also require the box $\boldsymbol{y}$ either to be certainly feasible or to satisfy either $w(\boldsymbol{f}_2(\boldsymbol{y})) < \mu$ (μ is a given tolerance) or $w(\boldsymbol{y}) < \epsilon$. This is to avoid to send to L_2 big undetermined boxes.

The resulting solution list L_{sol} of Algorithm 1 is the desired list of boxes containing the complete efficient set of (1).

5 Computational experiments

5.1 Competitive location problems

In our computational studies, we have used test problems which correspond to the biobjective competitive facility location problem introduced in Fernández et al. [6].

A franchise which wants to enlarge its presence in a given geographical region (where other competing facilities offering the same service are already present) by opening one new facility. Both the franchisor (the owner of the franchise) and the franchisee (the actual owner of the new facility to be opened) have the same objective: maximize their own profit. However, the maximization of the profit obtained by the franchisor is in conflict with the maximization of the profit obtained by the franchisee. In the model the *demand* is supposed to be *inelastic* and concentrated at n demand points, whose locations and *buying power* are known. The location and quality of the existing facilities are also known. In the spirit of Huff [11] we consider that the demand points split their buying power among all the facilities proportionally to the *attraction* they feel for them. The attraction (or utility) function of a customer towards a given facility depends on the distance between the customer and the facility, as well as on other characteristics of the facility which determine its *quality*. The location (x_1, x_2) and the quality α of the new facility are the variables of the problem. For a mathematical formulation of the problem see [5] or [6].

To clarify the biobjective nature of the problem and the way the constraint-like method works, consider Fig. 5. In the picture on the left, light gray circles with numbers 1–5 denote forbidden regions around the existing demand points, supposed to be at the center of the forbidden regions and all with demand 1, the cross $\times$ denotes the location of an existing facility owned by the chain and the dotted box $\boxdot$ the location of a competitor's facility. The franchisor would like the new facility to be located close to demand point 5 (he/she already captures most of the market of demand points 1–4, and in this way he/she can win a part of the market of demand point 5), whereas the franchisee would like the facility to be located close to the existing chain-owned facility (in this way, he can capture nearly half of the market of demand points 1–4, which is much more than he/she can gets by locating close to demand point 5). In different colors, we can see the part of the outer approximation of the efficient set (projected in the location space) obtained with the solution of the different constraint problems considered in the execution of MCLM. In the picture on the right we can see the corresponding outer approximation of the nondominated set offered by the algorithm. Notice that the biobjective problem considered is rather difficult (each objective alone leads to a global optimization problem), and its efficient and nondominated sets may have a very general shape (they can be even nonconnected, as in the example).

5.2 Results

To investigate the performance of Algorithm 1, as well as the efficiency of the different discarding tests, we have generated different problems, varying the number of demand points ($n = 50, 100$), the number of existing facilities ($m = 2, 5, 10$) and the number of those facilities belonging to the chain ($k = 0, 1$ for $m = 2$, $k = 0, 1, 2$ for $m = 5$, and $k = 0, 2, 4$ for $m = 10$). For each of the 16 settings considered one instance was generated, by randomly choosing the parameters of the model uniformly within some intervals. The searching space for every problem was $x \in [0, 10]^2$, $\alpha \in [0.5, 5]$. We set $\delta = 0.01$ and the tolerances used in the algorithm were $\epsilon = 0.01$ and $\eta = 0.005$.

All the computational results presented in this paper have been obtained on a PC with an Intel Pentium IV 2.33 GHz processor and with 1 Gbyte RAM running under Linux operating system. For the implementation we have used the interval arithmetic modules provided in the PROFIL/BIAS library [12] and the automatic differentiation of the C++ toolbox library described in Hammer et al. [8].

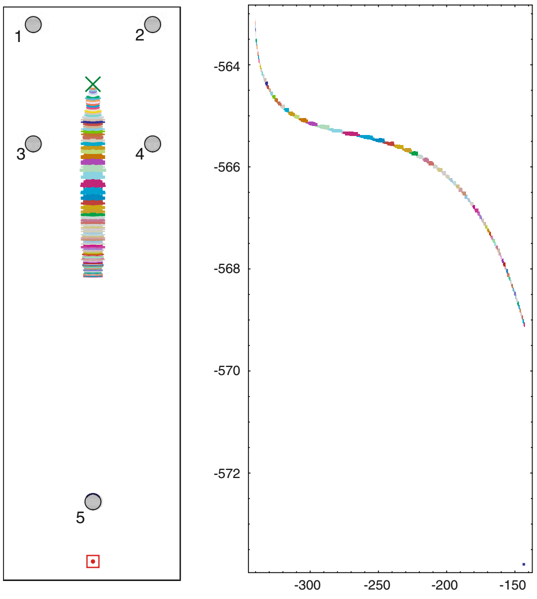


Fig. 5 Conflicting objectives

First, we have studied the usefulness of the different discarding tests. To this end, we have solved all the problems with the following algorithms:

simple: in this algorithm we only use the δ-cut-off and the feasibility test. In the later, we use the corresponding natural interval extensions as the inclusion functions of the constraints to check the feasibility. The pruning test is substituted by a simple bisection of the box under consideration perpendicular to the direction of maximum width.
basic: we use the tests in ‘simple,’ but now, in the feasibility test, for the constraint on f_2 we use the centered form as inclusion function.
basic + mono: we use the tests in ‘basic’ and the δ-monotonicity test.
basic + mco: we use the tests in ‘basic’ and the multiobjective cut-off test.
basic + prunf_2: we use the tests in ‘basic’ and the pruning test applied to f_2.
basic + prunf_1f_2: we use the test in ‘basic’ and the pruning test applied to f_1 and f_2.
basic + mco + prunf_1f_2: we use the tests in ‘basic + prunf_1f_2’ and the multiobjective cut-off test.
basic + all: in which we use all the discarding tests, that is, we use as main procedure the one given in Algorithm 2.

The results obtained are given in Table 1. For the ‘basic’ algorithm we computed the CPU time in seconds (Time), the effort of the algorithm [to be understood as the number of function evaluations plus three times the number of gradient evaluations (recall that our problem is a three-dimensional one), Effort], the maximum number of boxes stored in the lists ($L_1 + L_2 + L_3 + L_4$) at any time during the execution of the algorithm (ML), the number of boxes in L_{sol} (FB), and the volume of the boxes in L_{sol} (Vol). For the rest of the algorithms we computed the relative values of each of those indices as compared to the values for ‘basic,’ in percentage. The values in Table 1 summarize those results, and give the corresponding values (in average) when we

Table 1 Comparison of the different discarding tests

Algorithm		Time	Effort	ML	FB	Vol
basic	Average:	336.6	17,10,024	13,791	62,043	2.4e-2
simple	Average:	–	–	–	–	–
	Av. of%:	461%	755%	1,563%	7,360%	4,307%
basic+	Average:	101.8%	99.1%	85.1%	101.9%	104.2%
mono	Av. of%:	100.8%	99.3%	87.9%	98.0%	93.6%
basic+	Average:	86.0%	61.8%	66.1%	44.8%	50.0%
mco	Av. of%:	82.6%	64.1%	66.2%	38.3%	49.9%
basic+	Average:	94.5%	93.4%	55.2%	65.4%	100.0%
prunf_2	Av. of%:	92.8%	86.9%	54.7%	64.4%	103.9%
basic+	Average:	79.3%	70.9%	67.6%	52.2%	83.3%
prunf_1f_2	Av. of%:	90.9%	85.0%	72.1%	65.3%	97.3%
basic+mco	Average:	75.7%	55.7%	43.6%	34.0%	54.2%
+prunf_1f_2	Av. of%:	71.9%	53.0%	44.3%	28.3%	61.5%
basic+	Average:	80.4%	60.4%	44.9%	34.7%	58.3%
all	Av. of%:	75.5%	59.1%	46.9%	27.6%	53.9%

consider the sixteen problems altogether (Average), and the average of the relative values of each problem (Av. of %), respectively.

Whereas ‘basic’ is able to solve all the problems, ‘simple’ runs out of memory in 13 of the 16 problems (that is why we have not written any value in the average line; on the other hand, in the other line we compute the mean of the averages of the three problems for which ‘simple’ finished). This clearly shows that the overestimation produced by the natural interval extension used in the feasibility test causes serious troubles to the algorithm, which may be overcome with the use of the centered form. ‘basic’ solves all the problems in a reasonable amount of time (less than 6 min) and the volume of the outer approximating set of the efficient set is, in average, 0.024, that is, 0.0053% of the volume of the searching region.

The δ-monotonicity test does not seem to be useful for the type of problems under consideration, since it does not significantly reduce any of the parameters under study. On the contrary, the multiobjective cut-off test is very effective: it reduces the CPU time more than 15%, the corresponding reductions in effort and maximum number of boxes stored are around 35%, and the number of boxes is L_{sol} and their volume is reduced by almost half.

The pruning test applied to f_2 is also quite effective, specially in the reduction of storage (ML and FB). However, it is better to apply the pruning to both f_1 and f_2, since all the parameters under study (except ML) improve with regard to ‘basic + prunf_2.’

When the multiobjective cut-off test and the pruning test applied to f_1 and f_2 are used together, the algorithm obtains the best results. The CPU time is reduced more than 25%, the effort and the volume of the solution boxes to nearly the half, the maximum number of boxes stored is reduced more than 55% and the number of final boxes more than 70%. Although the reductions obtained when used jointly are not the sum of the individual reductions obtained with each discarding test alone, the results clearly show that when they are used together the performance of the algorithm

Table 2 Comparison with the direct B&B method

Algorithm		Time	Effort	ML	FB	Vol
biobjective algorithm	Average:	783.5	494,528	20,267	21,882	9.2e-3
basic + mco + prunf_1f_2	Average:	41.5%	274.4%	23.9%	148.7%	92.4%
	Av. of%:	68.5%	227.0%	34.6%	140.5%	95.0%

is much better. This is so because the type of information that they use is different, and thus, they discard different types of boxes.

If in addition to the previous tests we also use the δ-monotonicity test (i.e., the algorithm 'basic + all') then all the parameters under study either remain in similar values or slightly worsen, confirming again that the δ-monotonicity test is not useful for the type of problems under consideration.

Second, we have compared the best implementation of MCLM (basic + mco + prunf_1f_2) with the interval branch-and-bound method presented in the companion paper [6], which is also aimed at the explicit construction of the full efficient set. The method in Fernández et al. [6] deals with the biobjective problem directly, without reducing it to a sequence of single-objective problems, and uses as discarding tests the feasibility and multiobjective cut-off tests. The results obtained are summarized in Table 2. For the biobjective interval B&B algorithm in Fernández et al. [6] we give the values obtained when the tolerances used are $\epsilon_1 = \epsilon_2 = \epsilon_3 = 0.01$, whereas for the 'basic + mco + prun f_1f_2' we give the relative values of each of those indices as compare to the other algorithm, using $\delta = 0.01$, $\epsilon = 0.008$, and $\eta = 0.0004$. With those tolerances both methods produce outer approximations of similar quality (on average, the volumes of the boxes in the corresponding L_{sol} lists are very similar).

As we can see, the effort required by our constraint-like method is much greater than for the biobjective interval B&B algorithm (due to the use of gradient information). However, our implementation of MCLM is considerably faster (on average it needs 60% less time) and needs much less space (in average, ML is reduced around 75%). As for the number of boxes in L_{sol}, it is greater for our constraint-like method, although the area covered by them is smaller. Thus, we can say that our constraint-like method clearly outperforms the biobjective interval B&B algorithm in Fernández et al. [6].

6 Conclusions and future research

We have presented a general method for obtaining an outer approximation of the efficient set (and the nondominated set) of nonlinear biobjective optimization problems. It is based on the *constraint method*, and transforms the problem to the computation of the regions of δ-optimality of a finite number of single objective constraint problems.

A unified interval branch-and-bound implementation is developed, for which specific discarding tests are proposed. Although, the δ-cut-off test and the feasibility test (using the centered form as inclusion function for the constraint on f_2) are enough to guarantee the convergence of the method, the use of other discarding tests, namely, the pruning test applied to f_1 and f_2 and the multiobjective cut-off test, makes the implementation much faster and reduces its storage requirements. In fact, when compared to another general method recently proposed in Fernández et al. [6] with the same aim, our method has shown to be clearly superior.

The design of additional discarding tests, as well as other accelerating devices, will be the subject of future research. The extension of the method and the discarding tests to more than two objectives, and the study of its efficiency, should also be investigated.

References

1. Carrizosa, E., Conde, E., Romero-Morales, D.: Location of a semiobnoxious facility. A biobjective approach, In: Advances in Multiple Objective and Goal Programming 1996.Lecture Notes in Economics and Mathematical Systems 455, pp. 338–346, Springer, Berlin Heidelberg Newyork (1997)
2. Ehrgott, M., Wiecek, M.M. : Multiobjective programming. In: Figueira, J., Greco, S., Ehrgott, M. (eds.) Multiple criteria Decision Analysis: State of the art surveys., pp. 667–722. Kluwer, Dordrecht, MA (2005)
3. Fernández, J., Pelegrín, B., Plastria, F., Tóth, B.: Solving a Huff-like competitive location and design model for profit maximization in the plane. Eur. J. Oper. Res. **179**, 1274–1287 (2007)
4. Fernández, J., Pelegrín, B., Plastria, F., Tóth, B.: Planar location and design of a new facility with inner and outer competition: an interval lexicographical-like solution procedure, Netw. Spat. Econ, to appear (DOI: 10.1007/s11067-006-9005-4) (2007)
5. Fernández, J., Tóth, B.: Obtaining an outer approximation of the efficient set of nonlinear biobjective problems (extended version of this paper), Available at http://www.um.es/geloca/gio/CLMextended.pdf (2005)
6. Fernández, J., Tóth, B., Plastria, F., Pelegrín, B.: Reconciling franchisor and franchisee: a planar biobjective competitive location and design model. In: Recent Advances in Optimization. Lectures Notes in Economics and Mathematical Systems 563, pp. 375-398. Springer, Berlin Heidelberg Newyork (2006)
7. Figueira, J., Greco, S., Ehrgott, M. (eds): Multiple Criteria Decision Analysis: State of the Art Surveys, Kluwer, Dordrecht, MA (2004)
8. Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: C++ Toolbox for Verified Computing. Springer, Berlin Heidelberg New York (1995)
9. Hansen, E., Walster, G.W.: Global Optimization Using Interval Analysis, Second Edition, Revised and Expanded. Marcel Dekker, New York (2004)
10. Hansen, P., Jaumard, B.: Lipschitz optimization. In: Handbook of Global Optimization. pp. 407–494. Kluwer, Dordrecht (1995)
11. Huff, D.L.: Defining and estimating a trading area. J. Mark. **28**, 34–38 (1964)
12. Knüppel, O.: PROFIL/BIAS—a fast interval library. Computing **53**, 277–287 (1994)
13. Kearfott, R.B.: Rigorous Global Search: Continuous Problems. Kluwer, Dordrecht (1996)
14. Martínez, J.A., Casado, L.G., García, I., Tóth, B.: AMIGO: advanced multidimensional interval analysis global optimization algorithm, In: Frontiers in Global Optimization. Nonconvex Optimization and Its Applications, vol. 74, pp. 313–326. Kluwer, Dordrecht (2004)
15. Miettinen, K.S.: Nonlinear Multiobjective Optimization. Kluwer, Boston (1998)
16. Nickel, S., Puerto, J.: Location Theory - a Unified Approach. Springer, Berlin (2005)
17. Plastria, F.: GBSSS: the generalized big square small square method for planar single-facility location. Eur. J. Oper. Res. **62**, 163–174 (1992)
18. Ratschek, H., Rokne, J.: New Computer Methods for Global Optimization. Ellis Horwood, Chichester (1988)
19. Ruzika, S., Wiecek, M.M.: Approximation methods in multiobjective programming. J. Optim. Theory Appl. **126**, 473–501 (2005)
20. Sayin, S.: Measuring the quality of discrete representations of efficient sets in multiple objective mathematical programming. Math. Program. **87**, 543–560 (2000)
21. Tóth, B., Csendes, T.: Empirical investigation of the convergence speed of inclusion functions. Reliable Comput. **11**, 253–273 (2005)
22. Tóth, B., Fernández, J., Csendes, T.: Empirical convergence speed of inclusion functions for facility location problems. J. Comput. Appl. Math. **199**, 384–389 (2007)