Combination of interval analysis and PSO for optimization

Clément Solau[1,2], Bruno Marhic[3], Laurent Delahoche[3], Arnaud Clérentin[3], Anne-Marie Jolly-Desodt[1] and David Menga[1]

[1] EDF R&D – Clamart - France
[2] PRISME laboratory of University Orleans - France
[3] LTI laboratory of University of Picardie Jules Vernes – France)

Abstract

In this paper, a constrained optimization method for various functions such as non differentiable ones based on interval analysis is proposed. The process can be broken down into two distinct parts: the first one that approximates the space satisfying the constraints and the second one that is exclusively in charge of optimizing a given function. The first phase relies on successive bisections of the initial space while discarding the boxes that cannot verify the constraints. The last part of the algorithm is in charge of the unconstrained optimization using a Particle Swarm Optimization algorithm (commonly known as PSO).

Keywords: Constrained optimization, interval analysis, particle swarm optimization.

1. Introduction

Optimizing a constrained function can be carried out by various methods. These methods differ from each other with regards to the constrained function to be optimized. Some methods are able to only deal with both linear functions and constraints (i.e. Simplex algorithm) while others have a larger field of application (i.e. Newton's methods…). In addition, metaheuristic algorithms also became as they do not require the function to be differentiable.

1.1. Preliminary

In past decades, researchers tried to combine optimization properties with interval analysis methods. First, [1] introduced a method composed of interval arithmetic and optimization properties for minimizing a C^2 unconstrained function of one variable. In a subsequent paper [2], the same mechanism has been extended to C^2 unconstrained functions from R^n to R. This method is based on subdivision of intervals where their convexity is tested. Then, a Newton's interval method [3] is applied to find the solution of $\nabla f(x) = 0$ while an upper bound on the optimal solution is continuously assessed and modified if required. This method offers relevant results but only for unconstrained and differentiable functions. Subsequently, [4] presented an interval arithmetic method for constrained optimization by bisecting successively an original domain (i.e. a box or an interval) into two smaller ones. By means of computing a lower and an upper bound of the interval extension of the function (i.e. inclusion function), each domain is assessed and then discarded if the optimal solution cannot belong to it. Afterwards, when the initial search space is reduced, a Newton's method [3] is used to work out the zeros of the derivative. The algorithm presented in [4] is able to deal with equality and inequality constraints by adding slack variables and finding the 0 of the Lagrange function derivative. The importance of inclusion functions is a major topic since it is used as a bounding procedure. Some methods use only the natural extension [5] while others are based on the Hessian matrix calculation. Comparative results are provided in [6]. Furthermore, [7] introduced an accelerated method by means of modified termination and box selection criteria. [7] also stated the importance of the relaxation of the equality constraints since interval arithmetic is not a general framework that can rigorously deal with it: it recommends to relax each of them. Last, [8] proposed a reformulated method that transforms the non-convex objective terms into constraints. All the methods presented above work for various classes of problems except when the function is not differentiable.

1.2. Overview

This paper will address the problem of optimizing a various class of functions (linear, non-linear, non-differentiable) with heterogeneous constraints (inequalities, equalities, linear constraints…) based on set estimation and particle based optimization. First, the constrained optimization problem will be stated. Section 2 will provide an overview of the interval analysis tools. Section 3 will describe the PSO algorithm for optimizing a large range of functions (non-differentiable...). In section 4, our algorithm will be explained and results are provided in the following part of this paper.

1.3. Problem Statement

Optimization problems are now well known and can be applied to many fields such as physics, biology and economy. These optimization problems can also be classified into different classes but this paper is only focused on mono-objective functions. Obviously, these problems can be subject to equality and/or inequality constraints or not. In this paper, the inequality constraint satisfaction problem will be ensured by a set

approximation algorithm whilst the equality constraint satisfaction problem will be guaranteed by a penalty function directly inserted in the optimization criteria. However, the penalty function design will not be mentioned in this paper since it can be an open subject in itself. On the one hand, let us consider the following minimization problem defined as below:

$$f:\begin{cases} R^n \to R \\ (x_1,x_2,...,x_n) \to y = f(x_1,x_2,...,x_n) \end{cases} \quad (1)$$

On the other hand, let us consider that the objective function will be subject to the *m* inequality constraints and to the *k* equality constraints defined as written below:

$$\begin{aligned} & h_j(x_1,x_2,...,x_n) \le 0 \\ & g_i(x_1,x_2,...,x_n) = 0 \\ & j = 1,2,....,m \text{ and } i = 1,2,...,k \end{aligned} \quad (2)$$

An optimal solution of the previously defined problem will be written $(x_1^*,x_2^*,...,x_n^*)$ such that $\forall (x_1,x_2,...,x_n) \in R^n$ verifying $h_j(x_1^*,x_2^*,...,x_n^*) \le 0$ and $g_i(x_1^*,x_2^*,...,x_n^*) = 0$.

we get $f(x_1^*,x_2^*,...,x_n^*) \le f(x_1,x_2,...,x_n)$ .Here, the proposed method computes an approximated solution in which the distance with the optimal solution $(x_1^*,x_2^*,...,x_n^*)$ is not too important. Furthermore all the constraints are verified. This algorithm solves this sort of problems with a variable number of constraints and for objective functions (differentiable or not) from R^n to *R* (with *R* the set of the real numbers). As a conclusion, the proposed algorithm will be able to solve this type of problems (1)-(2) but also unconstrained problems (or only problems with inequality constraints or equality constraints).

2. Interval Analysis

2.1. Basic definitions and operations on intervals

Interval analysis entails a set of method dedicated to the inner or outer approximation of sets. The principle is based on a simple idea: to bound the real numbers and vectors respectively in intervals and boxes. The fundamental contribution has been provided by [5]. In this part, the basic concepts of intervals will be provided as well as the operators. An interval is a set that is also a subset of *R*. In the sequel, intervals will be written as $[x]$. Let us define its lower bound written as $lb([x]) = \underline{x}$ and defined as below:

$$\underline{x} = lb([x]) = \sup\left\{a \in R \cup \{-\infty,+\infty\} \,\middle|\, \forall x \in [x], a \le x\right\} \quad (3)$$

In other words, the lower bound of an interval $[x]$ can be perceived as the greatest low delimiter of $[x]$. In a similar way, the upper bound of an interval $[x]$ and written $ub([x]) = \underline{x}$ can be defined as described below:

$$\overline{x} = ub([x]) = Inf\left\{b \in R \cup \{-\infty,+\infty\} \,\middle|\, \forall x \in [x], x \le b\right\} \quad (4)$$

The upper bound of an interval $[x]$ can be viewed as the smallest upper delimiter of $[x]$. Hence, in accordance with these definitions, it is possible to define an interval $[x]$ with its two bounds:

$$[x] = [\underline{x},\overline{x}] = \left\{x \in R \,\middle|\, \underline{x} \le x \le \overline{x}\right\} \quad (5)$$

In this case, the interval is said to be closed but other forms of intervals exist such as open intervals $[x] = (\underline{x},\overline{x})$, half-open intervals $[x] = [\underline{x},\overline{x})$. However, this paper will exclusively deal with closed-intervals. An interval can also be described by its width (if not empty) that will be written $w([x])$:

$$w([x]) = \overline{x} - \underline{x} \quad (6)$$

Last, the center of an interval $[x]$ (if not empty) is written $mid([x])$ and will be defined as follows:

$$mid([x]) = \frac{\overline{x} + \underline{x}}{2} \quad (7)$$

As mentioned earlier in this paper, an interval is a set and it is therefore logic that arithmetic operators and operations on sets can be extended to intervals. Therefore, the intersection of two intervals $[x]$ and $[x]$ will be defined as follows:

$$\begin{aligned} & [x] \cap [y] = \left\{z \in R \,\middle|\, z \in [x] \text{ and } z \in [y]\right\} \\ & = \begin{cases} [\max\{\underline{x},\underline{y}\}, \min\{\overline{x},\overline{y}\}] \text{ if } \max\{\underline{x},\underline{y}\} \le \min\{\overline{x},\overline{y}\} \\ \varnothing \text{ otherwise} \end{cases} \end{aligned} \quad (8)$$

In the same way, the union can also be extended to intervals:

$$\begin{aligned} & [x] \cup [y] = \left\{z \in R \,\middle|\, z \in [x] \text{ or } z \in [y]\right\} \\ & = [\min\{\underline{x},\underline{y}\}, \max\{\overline{x},\overline{y}\}] \end{aligned} \quad (9)$$

Furthermore, the four arithmetic operators on real numbers $\Diamond \in \{+,-,*,/\}$ can be extended to intervals. The result of a binary operation $\Diamond \in \{+,-,*,/\}$ of two intervals $[x]$ and $[y]$ will be defined as follows:

$$[x] \Diamond [y] = \left[\left\{x \Diamond y \in R \,\middle|\, x \in [x], y \in [y]\right\}\right] \quad (10)$$

This rule has been introduced first by [9] for the closed intervals and then extended to open-intervals by [10]. Since this paper is focuses on closed-intervals, these definitions will only be provided for closed-intervals:

$$[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \quad (11.1)$$

$$[x] - [y] = [\underline{x} - \underline{y}, \overline{x} - \overline{y}] \quad (11.2)$$

$$\begin{aligned} & [x] \times [y] \\ & = \left[\min\left\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\right\}, \max\left\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\right\}\right] \end{aligned} \quad (11.3)$$

$$1/[y] = \begin{cases} \varnothing \text{ if } [y] = [0,0] \\ [1/\overline{y}, 1/\underline{y}] \text{ if } 0 \notin [y] \\ [1/\overline{y}, \infty] \text{ if } \underline{y} = 0 \text{ and } \overline{y} > 0 \\ ]-\infty, 1/\underline{y}] \text{ if } \underline{y} < 0 \text{ and } \overline{y} = 0 \\ ]-\infty, \infty[\text{ if } \underline{y} < 0 \text{ and } \overline{y} > 0 \end{cases} \quad (11.4)$$

$$\text{with } [x]/[y] = [x] \times (1/[y])$$

It is obvious that when applied to punctual intervals ($[x]=x$ and $[y]=y$), the previous rules becomes the usual arithmetic rule.

2.2. Interval Vector

In interval analysis, the notion of interval vectors is a crucial point since they allow the generalization to problems in n-dimensions and thus to bound vectors in entities named boxes. Indeed, an interval vector is a subset of R^n and is defined as the Cartesian product of the n intervals composing the interval vector. These interval vectors will be written $[x]$ and described as written below:

$$[x] = [x_1] \times [x_2] \times \ldots \times [x_n] \quad with [x_i] = \left\lfloor \underline{x_i}, \overline{x_i} \right\rfloor and\ i \in \{1,2,\ldots,n\} \tag{12}$$

Furthermore, the i-th component of $[x]$written $[x_i]$ is the projection of $[x]$ onto the i-th axis. Many notions already defined in this paper can be applied to the case where $n>1$. Logically, the lower and upper bounds of a box $[x]$ will consist respectively in computing the lower and upper bound of its interval components:

$$\underline{x} = lb([x]) = (lb([x_1]), lb([x_1]), \ldots, lb([x_n]))^T = (\underline{x_1}, \underline{x_2}, \ldots, \underline{x_n})^T \tag{13.1}$$

$$\overline{x} = ub([x]) = (ub([x_1]), ub([x_1]), \ldots, ub([x_n]))^T = (\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n})^T \tag{13.2}$$

Similarly, the width of a box $[x]$ will be equal to the width of one of its interval components whose width is maximal while the centre of a box $[x]$will be a vector composed of the centre of all its interval components:

$$w([x]) = \max_{1 \le i \le n} w([x_i]) \qquad mid([x]) = (mid([x_1]), mid([x_2]), \ldots, mid([x_n]))^T \tag{14}$$

The intersection and the union of two boxes $[x]$ and $[y]$will be defined respectively as:

$$[x] \cap [y] = ([x_1] \cap [y_1]) \times ([x_2] \cap [y_2]) \times \ldots \times ([x_n] \cap [y_n]) \tag{15.1}$$

$$[x] \cup [y] = ([x_1] \cup [y_1]) \times ([x_2] \cup [y_2]) \times \ldots \times ([x_n] \cup [y_n]) \tag{15.2}$$

Hence, the binary operators already mentioned in this paper can also be extended to boxes in a very similar way.

2.3. Inclusion function

Let f be a function from R^n to R^m . Its interval extension written $[f]$from IR^n to IR^m (with IR^n and IR^m the sets of boxes of n and m dimensions) is an inclusion function of f if it verifies:

$$\forall [x] \in IR^n, f([x]) \subset [f]([x]) \tag{16}$$

The problem is to find a method for producing such a function (except the trivial case where $[f]([x]) = R^m$). Let us consider a function $f: R^n \to R^m$ and let $|f_j|$ with j=1,2,…, m be m inclusion functions defined on $IR^n \to IR$. In this case the inclusion function of f will be written as illustrated below:

$$[f]([x]) = [f_1]([x]) \times \ldots \times [f_m]([x]) \tag{17}$$

Hence, the formulation of an inclusion function from R^n to R^m can be broken down into m sequential formulations that relates to its respective components. This is the reason why the following examples will only deal with inclusion functions from R^n to R (one component among the m components). Moreover, as already mentioned the binary operators could be extended to intervals. Let us consider that for elementary functions (cos, sin, tan, exp,…..) the same consideration can be ascertained. A mathematical function can always be written as a combination of variables, binary operators and elementary functions. Let $f: R^n \to R$ be such a function, it is thus possible to find a natural inclusion function written $[f]$ from IR^n to IR by simply both replacing each of the n real variables x_i and operators by theirs interval counterparts. This type of inclusion functions will be called natural inclusion function of f and the following instance illustrates this case:

Example: Let f be a function from R^2 to R^2 such that f: $(x_1,x_2) \to x_1^2 + \cos(x_2), x_2 \times \exp(x_1))$.
Its natural inclusion function from IR^2 to IR^2 is then defined as:
$[f](x_1,x_2) \to [x_1]^2 + \cos([x_2]), [x_2] \times \exp([x_1]))$.

Many sorts [6] of inclusion functions exist that are only the interval counterpart of function approximations (Taylor's series expansion, centred inclusion function…) of f. In our algorithm, only natural inclusion functions will be manipulated in order to compute the interval counterpart of the different constraints included in the problem.

2.4. Pavings, subpavings et SIVIA algorithm

Intervals form an interesting family of sets with some specific properties (they can be easily manipulated). In this part, we will focus on the different ways to use these sets in order to approximate more complex sets. The aim is to provide inner and outer approximations of the set X to be approximated.

Two subpavings $\underline{X}$ and $\overline{X}$ will be computed so that $\underline{X} \subset X \subset \overline{X}$. These two subpavings provide valuable information about X such as its volume since $vol(\underline{X}) \le vol(X) \le vol(\overline{X})$. In our case, these two subpavings $\underline{X}$ and $\overline{X}$ calculations will be based on successive bisections of an initial box $[x]$ stored in a binary tree (since each bisection will generate two children, the left and the right child) and so that $\underline{X} \subset [x]$and $\overline{X} \subset [x]$. Let us consider a box defined as $[x] = [x_1] \times [x_2] \times \ldots \times [x_n]$ and let j be the index of the maximum width component so that:

$$j = \min\{i \mid w([x_i]) = w([x])\} \tag{18}$$

It is thus possible to define its left and right child respectively written $L[x]$ and $R[x]$ and resulting from one step of bisection:

$$L[x] = [x_1] \times \ldots \times \left[\underline{x_j}, (\underline{x_j} + \overline{x_j})/2\right] \times \ldots \times [x_n] \quad (19.1)$$

$$R[x] = [x_1] \times \ldots \times \left[(\underline{x_j} + \overline{x_j})/2, \overline{x_j}\right] \times \ldots \times [x_n] \quad (19.2)$$

In our case, the set X to be approximated will be the set verifying the constraints of our optimization problem. This approximation process will be based on the use of the SIVIA (standing for Set Inversion Via Interval Analysis) algorithm first introduced in [11]. Let f be a function from R^n to R^n denoting the m constraints of the optimization problem and let Y be a subset of R^m . The set inversion will be defined as the following set:

$$X = \left\{x \in R^n \,\middle|\, f(x) \in Y\right\} = f^{-1}(Y) \quad (20)$$

The SIVIA algorithm allows to obtain two subpavings so that $\underline{X} \subset X \subset \overline{X}$ and thus to approximate the set X that satisfies the constraints (X would be a polytope if all the constraints are linear which is not always the case). In our optimization problem, the set $\underline{X}$ represents a subset of X and thus a subset of the set verifying the constraints. This approximation will need to be as accurate as possible in order to not exclude solutions that could be potentially optimal. The SIVIA algorithm is a recursive algorithm described as follows:

```
I: [x] : initial box, ε : algorithm
accuracy, f : a function denoting the
m constraints, Y : a subset of R^m.
I/O: X et X̄
1 : If f([x])∩Y=∅ then return; EndIf;
2 : If f([x])⊂Y then
3 :          X = X ∪ [x] ;
4 :          X̄ = X̄ ∪ [x] ;
5 :          return;
6 : EndIf ;
7 : If w([x])<ε then
8 :          X̄ = X̄ ∪ [x] ;
9 :          return ;
10 :EndIf ;
11 : SIVIA(I : L[x], ε, f, Y, IO : X, X̄ );
12 : SIVIA(I : R[x], ε, f, Y, IO : X, X̄ );
```

Line 1 states that if $[f]([x])$ has an empty intersection with Y, the current box $[x]$ can be erased from the search tree. Conversely (lines 2 to 4), if $[x]$ is strictly included in Y then $[x]$belongs to X and is then stored in $\underline{X}$ and $\overline{X}$. Last (lines 7 to 10), if $[x] \cap Y \neq \varnothing$ then $[x]$ is undetermined but if its width is inferior to the accuracy ε, $[x]$ will be considered as a part of the outer approximation. The recursive aspect of the algorithm is mentioned in lines 11 and 12 since the same function take as inputs the left and right child of the bisected box. Let us illustrate it by a toy example with a set composed of two constraints:

$$\begin{cases} 2 \le x_1 \le 8 \\ 0 \le x_1 - x_2 \le 2 \end{cases} \quad (21)$$

When applying SIVIA to this example, we will approximate the set verifying the two previously defined constraints and thus the possible values of the pair (x_1, x_2). This set is represented by the white boxes as depicted below:

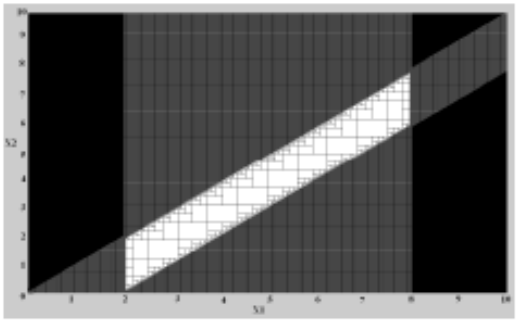

Fig. 1: Representation of the possible values of (x_1, x_2) on the search space [0,10]× [0,10] with different levels of accuracy.

The union of all the white boxes denotes the set satisfying the constraints and verifying $f([x_1]\ [x_2]) \subset Y$, with $y = \begin{pmatrix} [2,8] \\ [0,2] \end{pmatrix}$ and $f : ([x_1],[x_2]) \to \begin{pmatrix} [x_1] \\ [x_1]-[x_2] \end{pmatrix}$. The union of all the grey boxes represents the set for which $f([x_1],[x_2]) \cap Y \neq \varnothing$ and $w([x_1] \times [x_2]) < \varepsilon$. In other words, this set verifies at least one constraint without verifying all of them. Last, the union of all black boxes is a set that does not satisfy any constraints. Mathematically speaking, it satisfies $f([x_1],[x_2]) \cap Y \neq \varnothing$. Once SIVIA has been applied, the union of all the white boxes is used as an input of an unconstrained optimization algorithm based on PSO and in charge of minimizing (or maximizing) an objective function. The union of all the white boxes represents the search space of the optimization algorithm.

3. Optimization using Particle based algorithm

The particle swarm optimization is a very recent optimization method introduced by [12]. The principle is fairly basic and is based on iterative optimization trying to improve a current solution with regard to a certain optimization criteria. This method can be classified in the family of the metaheuristics providing a correct approximation of the optimal solution. Furthermore, the growing calculation capacities of computers tend to extend the use of such metaheuristics. However, plenty other methods exist such as the simulated annealing [13], the tabu search [14], the ant colony algorithm [15] or the genetic algorithms [16]. The main advantage of these methods relies on the fact that they are capable of dealing with a large range of functions (linear, nonlinear,…). Indeed, particle swarm optimization does not require the gradient calculation which proves that this method does not require the function to be differentiable as it could be the case with other classical methods such as the gradient descent method or Newton's methods. The particle swarm optimization uses a chosen population (i.e. the swarm) that is composed of candidates that are

potentially a solution to the problem in a given search space. These candidate solutions are called the particles and their union is what composes the swarm. Along the optimization process, the particles are subject to movement in the search space according to a specific process. Indeed, the movement of each particle is weighted by both their own best experience and also by the current best solution of the swarm. In accordance with this process, all the particles are attracted in the best regions of the search space where the optimal solution may be located. This process is repeated until a maximal number of iterations is reached. Let f be the function to be minimized defined in a mono-objective framework:

$$f:\begin{cases} x \to f(x) \\ R^n \to R \end{cases} \qquad (22)$$

The aim is to compute an approximation of x^* for which $\forall x \in S \subset R^n$ we get $f(x^*) \le f(x)$ with S being the search space. In our case, the set S will be denoted by the union of all the white boxes. Let N be the number of particles defined by both their own current position $x_i \in R^n$ included in the search space S and by a velocity vector written $v_i \in R^n$. p_i represents the best known position of the particle i and g denotes the best current solution of the entire swarm. The particle swarm optimization algorithm is then defined as follows:

```
I: N: number of particles, M: number
of iterations, S: search space.
O: g  : global solution.
1 : For each particle i=1 to i=N do,
2 :    Initialize the particle posi-
tion with a uniformly distributed
random vector  x_i ← U(b_lo, b_up) (with b_lo and
b_up the lower and upper bounds of the
search space S);
3 :    Initialize the best particle
position i to its actual position
p_i ← x_i ;
4 :    If f(p_i) < f(g) then g ← p_i  EndIf ;
5 :    Initialize the velocity vector
of particle i,
          v_i ← U(-|b_up - b_lo|, |b_up - b_lo|) ;
6 : EndFor ;

7 : While the maximum number of ite-
rations (M) is not reached do,
8 :     For each particle i=1 to i=N
do,
9 :Generate random numbers r_p, r_g ← U(0,1) ;
10 :   Update the velocity of i
    v_i ← w v_i + φ_p r_p (p_i - x_i) + φ_g r_g (g - x_i) ;
11 :  Update the position  x_i ← x_i + v_i ;
12 :If f(x_i) < f(p_i) then update p_i ← x_i ;
13 :If f(p_i) < f(g) then g ← p_i  EndIf;
14 :        EndIf.
15 :    EndFor
16 : M=M+1 ;
17 : EndWhile
```

The output of this algorithm is the best solution found so far in a given number of iteration (i.e. M) with a given number of particles. The higher the number of particles and iterations, the better the solution quality is.

It has to be noted that several parameters occur in the optimization algorithm. Many papers devoted to this parameter tuning have been published such [17], [18] and [19]. In our case, the chosen values are respectively $w = 0.729$, $\varphi_P = 1.49445$, $\varphi_G = 1.49445$ and offers correct convergence criteria.

4. Proposed method

This part will be dedicated to our algorithm description. This algorithm has the main advantage of being capable of dealing with a large range of optimization problems as well as offering good approximation of the optimal solution. On the one hand, the constrained function to be minimized (or maximized) does not require to be differentiable nor linear. This consideration is due to the PSO mechanism. On the other hand, the algorithm is able to work with many types of constraints (linear or non-linear). The inequality constraints $h_i(x) \le 0$ will be treated by the SIVIA algorithm while the $g_j(x) = 0$ are directly included in the objective function through penalty function design. Let us define the proposed algorithm as follows:

```
I: f: function to be minimized, C:
set of inequality constraints and S:
intial search space.
O: g  : global solution.
1 : If C ≠ ∅ do
2 :       X ← ∅ ;
3 :       ε ← GetAccuracy(S) ;
4 :       X̄ ← ∅ ;
5 :    SIVIA(I : S, ε, C, IO: X, X̄ );
6 :     For i=1 to |X| do
7 :   (NBPart,NBIt)=GetNB(X(i)) ;
8 :    min=PSO(NBPArt, NBIt, f, X(i));
9 :             If min<g then
10 :              g ← min ; EndIf ;
11 :    EndFor ;
12 : Else do
13 :   (NBPart,NBIt)=GetNB(S) ;
14 :   g=PSO(NBPArt, NBIt, f,S);
15 :EndIf;
```

This algorithm can be split into two parts linked to the problem characterization. The first part (lines 1 to 11) only treat constrained problems while the second part (lines 12 to 15) deal with unconstrained optimization problems. The SIVIA algorithm is first applied to approximate the set defined by the m inequality constraints. This set is stored in a data structure list containing boxes. Moreover, the accuracy parameter tuning ε of the SIVIA algorithm is commensurate with the size of the search space in such a manner that the larger the size of the search space is, the higher the accuracy will be. Then (lines 6 to 11), the PSO algorithm is applied to each of the boxes. The number of particles and iterations is defined (line 7) commensurately with the size of the input box (i.e. $\underline{X}(i)$). Thus, the larger the size of the input box (i.e. $X(i)$) will be, the more important the number of particles and iterations is. Then, when the current global solution is improved (lines 9 to 11), its value is updated. As an output of this algorithm part, we will get an

approximation of the optimal solution. Furthermore, the second part of the algorithm (lines 12 to 15) works as a basic PSO for unconstrained problem. The output of this part will be an approximation of the optimal solution. As a remark, the parameter (N, M and ε) tuning can strongly influence the quality of the estimated solution.

5. Results

The algorithm has been developed using MatLab but performances could be improved by using a lower level programming language. Last, this part will only present examples of functions from R^2 to R for graphical representation reasons. This example will illustrate the optimization of a non-linear and not-differentiable constrained function from R^2 to R. Let f be the function to be minimized and (P) the following problem:

$$(P)\quad \begin{aligned} &\min_{(x,y)} f(x,y) = \min_{(x,y)}(x^2+y^2)\times|x| \\ &s.t.\quad 2 \le x^2+y^2 \le 12 \\ &\quad x+y-\sqrt{2}=0 \\ &with\quad -10 \le x \le 10 \text{ and } 0 \le y \le 10 \end{aligned} \qquad (23)$$

As a preliminary remark, the constraint $g(x, y) = 0$ will be inserted into the objective function and weighted with a penalty factor while the inequality constraint $12 \ge h(x,y) \ge 2$ will be kept as it is. The theoretical optimal solution minimizing the function is the pair $(x^*, y^*) = (0, \sqrt{2})$ associated with the function value $f(x^*, y^*) = f(0, \sqrt{2}) = 0$. With our approach and in moderate elapsed time (about 20 seconds), we obtain the following approximated solution (- 0.15625 , 1.5625) which is fairly far away from the optimal one. The function value of this pair is f (- 0.15625 , 1.5625)= 0.38528442382813. The resulting subpaving is as depicted below:

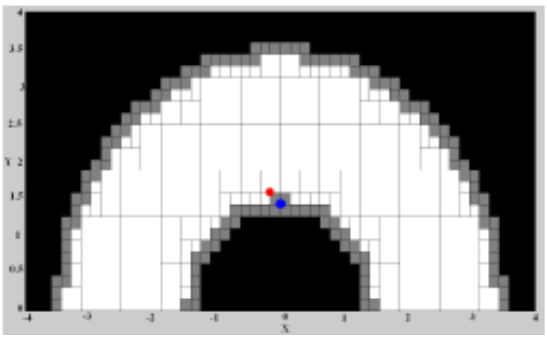

Fig. 2: Subpaving resulting from SIVIA.

As a remark, only a subpart (i.e. $[-4,4]\times[0,4]$) of the entire search space is represented. The blue spot denotes the theoretical optimal solution while the red one indicates our approximated solution. However, it is still possible to improve the algorithm output by increasing the accuracy level in order to converge in a thinner way to a correct estimation:

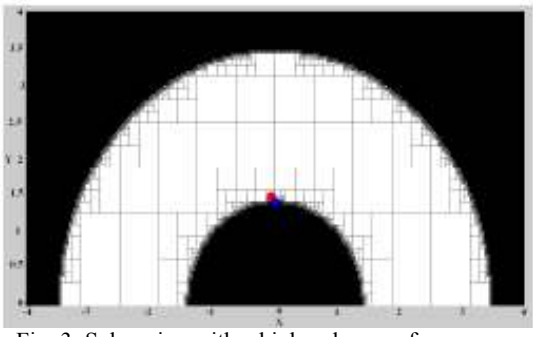

Fig. 3: Subpaving with a higher degree of accuracy.

As a result, the estimated solution (i.e. the red spot) is far better than the one obtained in the previous case but the elapsed time is a bit higher (about 70 seconds). The estimated solution is:
(- 0.06041220037076, 1.47500489534406) = (α,β),
with f (α,β) = 0.13165564827936. With this approximation the algorithm output is getting closer to the optimal solution $(x^*, y^*) = (0, \sqrt{2})$ associated with $f(x^*, y^*) = f(0, \sqrt{2}) = 0$. In this case, the accuracy factor influences the solution quality as well as the elapsed time.
However, this is not always the case and the following example will confirm this consideration. Let (P') be an optimization problem with the following linear constraints:

$$(P')\quad \begin{aligned} &\min_{(x,y)} f(x,y) = \min_{(x,y)}(x^2+y^2) \\ &s.t.\quad -10 \le x+y \le 10 \\ &\quad -10 \le x-y \le 10 \\ &with\quad 10 \ge x \ge -10 \text{ and } 10 \ge y \ge -10 \end{aligned} \qquad (24)$$

In this example, the theoretical optimal solution is the point (x^*, y^*)= (0, 0). We tested our algorithm with different degrees of accuracy; our results are illustrated on the following curves:

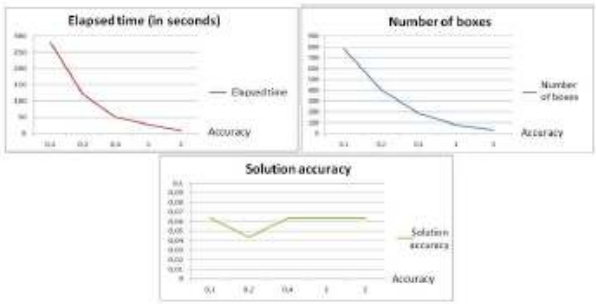

Fig. 4: Influence of the SIVIA accuracy on the elapsed time, the number of boxes and the result quality.

As a commentary, the x-axis of each subplot represents the accuracy of SIVIA in terms of the size of the box under which the box will never be bisected again. It is more generally the upper bound on the minimal size of a box. The lower the value, the lower the size of the box will be and the higher the accuracy of the algorithm. It is thus logical that this factor influences the number of boxes to be treated and as a consequence the elapsed time. However, this factor does not influence the quality of the solution (i.e. the green curve that

illustrates the distance between our solution and the optimal one). It is due to the fact that the solution is located in an area far away from the constraints frontiers. As a consequence, the accuracy required to get a good optimal solution bounding is not so important. The two following graphs illustrate this consideration:

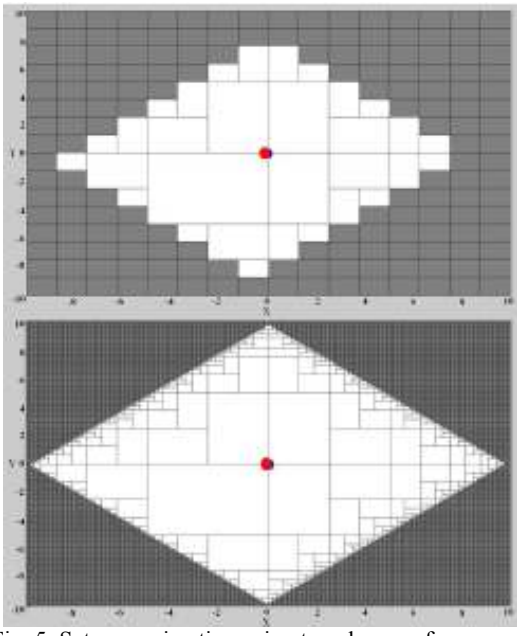

Fig. 5: Set approximation using two degree of accuracy.

It is clearly visible that the two solutions are overlapping with each other which means that they are really close to each other (approximation: ¸ 0.05938, 0.01672). In this case the accuracy of the constraint satisfaction problem is not a crucial point since the optimal solution is located in a zone which is far away from the constraints frontiers. Conversely, let us illustrate a case where the accuracy will strongly affect the quality of the estimated solution. Let (P'') be the following non-linear optimization problem:

$$(P'') \quad \min_{(x,y)} f(x,y) = \min_{(x,y)}(x^2 + y^2) \quad s.t. \quad x + y \le 0 \quad (25) \quad with \quad 10 \ge x \ge -10 \; and \; 10 \ge y \ge -10$$

The theoretical optimal solution is the pair (x^*, y^*)= (0,0) with $f(x^*, y^*)$= 0 . Similarly to the previous example, we will assess the influence of the same factor on different aspects. The influence is shown on the following curves:

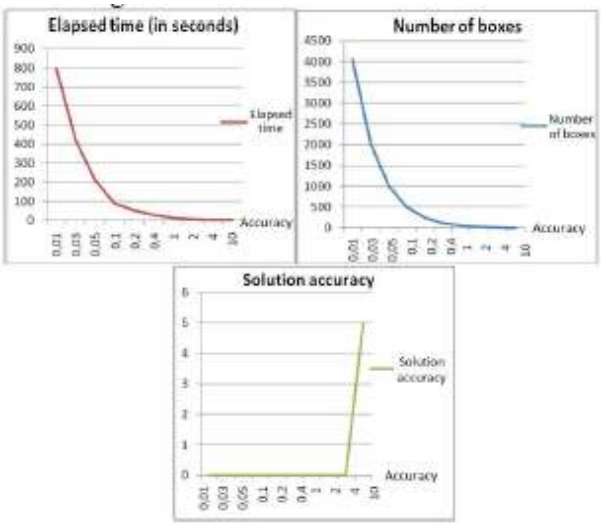

Fig. 6: Influence of the SIVIA accuracy on the elapsed time, the number of boxes and the result quality.

On the two first curves, the same phenomenon occurs for exactly the same reasons as in the previous case. Indeed, a higher accuracy level increases both the number of boxes to be dealt with and also the elapsed time. However, on the third curve from an accuracy ranging from 0.01 (highest accuracy level) to 2, the error (i.e. distance between the estimated solution and the optimal one) can be neglected and is closed to 0. Moreover, this error becomes important when the accuracy level is decreasing (or the limit size of the boxes is increasing). Similarly to Fig.5, we present the same experiment with different levels of accuracy:

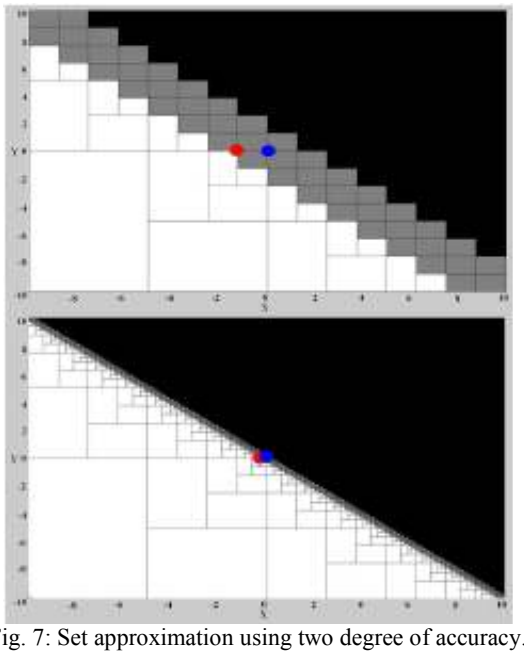

Fig. 7: Set approximation using two degree of accuracy.

In this example, it is easy to notice that the granularity of the subpaving strongly influences the quality of the solution. On the one hand, when the level of precision is fairly poor (on the left figure) the approximated solution is (-1.25,0)with f (-1.25,0) =1.5625which is fairly different from the optimal solution $(x^*, y^*) = (0, 0)$ with $f(x^*, y^*) = 0$. On the other hand, when the level of precision is increased (bottom figure), the solution

quality is improved. Indeed, in this case, the estimated solution is (- 0.15625, 0) with f (- 0.15625, 0) = 0.0244140625 which is closer to the optimal solution than the previous case. The reason of that influence lies in the fact that the optimal solution is located close to the constraint boundary. Indeed, $(x^*, y^*) = (0, 0)$ and $x^* + y^* = 0$. This is a classical topic in optimization and especially in «interior point method». With such a method, the objective function is penalized when the current solution is getting close to the constraint boundary. It allows the current solution to stay inside the set verifying the constraint by moving the current solution far away from the constraint boundary. Thus, it also moves the current solution away from the optimal point (since it is close to the constraint frontier). This method proposes to deal with the constraints first in order to avoid this type of problems (if the accuracy level is enough). However the recursive aspect of the SIVIA algorithm can be time consuming depending on the form of the problem. Last, the dimension of the problem strongly affects the execution speed of this algorithm.

6. Conclusion

We proposed a general method for solving various optimization problems. Indeed, constraints can range from inequality constraints to equality constraints with a linear or non-linear objective function. Furthermore, the function to be optimized does not require being differentiable as is the case with a lot of penalty design methods. However, our algorithm will only provide one solution and will not be able to compute the minimum of a function having many global solutions. A solution to remediate to this problem could be envisaged. Moreover, the problem of the parameter tuning (i.e. accuracy) is a crucial point since the quality of the solution could depend on it. A solution may be to set for every single problem a high accuracy level but the problem would become time consuming. A method for evaluating a compromise between the quality of the solution and the elapsed time could be added to the algorithm.

References

[1] E. Hansen, Global Optimization Using Interval Analysis : The One Dimensional Case, 1979.

[2] E. Hansen, Global Optimization Using Interval Analysis : The multidimensional Case, 1980.

[3] E. Hansen, Interval Forms of Newtons Method, 1978.

[4] K. Ichida, An Interval Arithmetic Method For Global Optimization, 1979.

[5] R.E. Moore, Interval Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1966.

[6] H. Ratschek & J. Rockne, New computer methods for global optimization, Chichester, West Sussex, England: Ellis Horwood, 1988.

[7] R.P. Byrne & I.D.L. Bogle, An Accelerated Interval Method For Global Optimisation, Journal of Computer and Chemical Engineering Volume 20, 1996.

[8] R.P. Byrne & I.D.L. Bogle, Global Optimization of Constrained Non-Convex Programs Using Reformulation And Interval Analysis, Journal of Computer and Chemical Enginnering, 1999.

[9] R.E. Moore, Automatic error analysis in digital computation, technical report, 1959.

[10] R.J. Hanson, Interval arithmetic as a closed arithmetic system on a computer, 1968.

[11] L. Jaulin and E. Walter. Set inversion via interval analysis for nonlinear bounded-error estimation, Automatica, 29(4), 1053-1064, 1993.

[12] J. Kennedy & R. Eberhart, Particle Swarm Optimization, Proceedings of IEEE International Conference on Neural Networks. **IV**. pp. 1942–1948, 1995.

[13] Kirkpatrick S., Gelatt C.D., Vecchi M.P., Optimization by Simulated Annealing, Science volume 220 number 4598, 1983.

[14] Glover F., Tabu Search — Part I, ORSA Journal on Computing 1: 3, 190-206, 1989.

[15] Dorigo M., Optimization, Learning and Natural Algorithms. Ph.D.Thesis, Politecnico di Milano, Italy, in Italian, 1992.

[16] Mitchell M., An Introduction to Genetic Algorithm, MIT Press, 1996.

[17] Y. Shi & R.C. Eberhart, Parameter selection in particle swarm optimization, Proceedings of Evolutionary Programming VII (EP98) pp. 591–600, 1998.

[18] M. Clerc & J. Kennedy, The particle swarm - explosion, stability, and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation **6** (1): 58–73, 2002.

[19] A. Carlisle& G. Dozier, An Off-The-Shelf PSO, Proceedings of the Particle Swarm Optimization Workshop. pp. 1–6, 2001.