PSwarm: A Hybrid Solver for Linearly Constrained Global Derivative-Free Optimization

A. I. F. Vaz* L. N. Vicente†

March 16, 2009

Abstract

PSwarm was developed originally for the global optimization of functions without derivatives and where the variables are within upper and lower bounds. The underlying algorithm used is a pattern search method, or more specifically, a coordinate search method, which guarantees convergence to stationary points from arbitrary starting points. In the (optional) search step of coordinate search, the algorithm incorporates a particle swarm scheme for dissemination of points in the feasible region, equipping the overall method with the capability of finding a global minimizer. Our extensive numerical experiments showed that the resulting algorithm is highly competitive with other global optimization methods based only on function values.

PSwarm is extended is this paper to handle general linear constraints. The poll step now incorporates positive generators for the tangent cone of the approximated active constraints, including a provision for the degenerate case. The search step has also been adapted accordingly. In particular, the initial population for particle swarm used in the search step is computed by first inscribing an ellipsoid of maximum volume to the feasible set. We have again compared PSwarm to other solvers (including some designed for global optimization) and the results confirm its competitiveness in terms of efficiency and robustness.

Keywords: Direct search, linear constraints, bound constraints, pattern search, coordinate search, particle swarm, derivative-free optimization, global optimization.

*Department of Systems and Production, University of Minho, Campus de Gualtar, 4710-057, Portugal (aivaz@dps.uminho.pt). Support for this author was provided by Algoritmi Research Center and by FCT under grants POCI/MAT/59442/2004 and POCI/MAT/58957/2004.

†CMUC, Department of Mathematics, University of Coimbra, 3001-454 Coimbra, Portugal (lnv@mat.uc.pt). Support for this author was provided by FCT under grants POCI/MAT/59442/2004 and PTDC/MAT/64838/2006.

1 Introduction

A significant number of applied optimization problems involve functions whose derivatives are unknown. In some practical instances those derivatives can be computed but then either the cost of the calculation is prohibitive or the functions are noisy and the derivatives meaningless. Computing stationary points of optimization problems without using the derivatives of the problem defining functions is a challenging task, particularly when the functions evaluations are expensive. However, there are state-of-the-art derivative-free methods and software which can handle problems with many dozen (or even more than one hundred) optimization variables, in serial computation, using a reasonable number of functions evaluations. A comprehensive review on derivative-free optimization is given in the book [10].

In many of the abovementioned problems the objective functions are nonconvex, a situation which typically occurs when one tries to fit or adjust observable data by regression using nonlinear models (see, for instance, the recent study [13] on the estimation of stellar parameters from observable measurements). When the goal is to find a global optimizer, the overall computation is significantly more complicated. The contribution of the mathematical programming community to the solution of these problems has been limited and mostly directed to the application of heuristic techniques. In our view, not enough testing and benchmarking have been reported to help us find the most efficient and robust techniques.

The authors developed in [34] a derivative-free algorithm for the minimization of a function specifically for the case where the variables have upper and lower bounds. The underlying method is based on coordinate search which is known to be one of the simplest (directional) direct search methods. Such a choice is particularly well suited for problems with simple bounds, since the coordinate directions conform naturally to the local geometry of the constraints. We considered each iteration of our algorithm divided into two steps (a poll step and a search step). The poll step is where the coordinate search was applied. The search step was used to incorporate a dissemination scheme for generating points in the feasible region, in an attempt to equip the overall method with the capability of finding a global minimizer. We selected particle swarm for this purpose because it is a simple population-based scheme that can be easily parallelized. We took advantage of having used a population in the search step to then poll around the best particle, which improved the overall robustness of the algorithm. In the vicinity of a global minimizer, the application of the poll step allows the use of a reduced number of particles, which is trivially achieved by dropping particles once they become too close to each other. This procedure improves the efficiency of the overall scheme. One is able to prove (see [34]) that the algorithm is globally convergent to first-order stationary points and, under some additional conditions, that it can eventually meet the stopping criterion used in both search and poll steps. Our extensive numerical experiments reported in [34] showed that the resulting algorithm (PSwarm) is highly competitive with other global optimization methods also based on function values.

In this paper we extend PSwarm to solve optimization problems defined by general

linear constraints (without using the derivatives of the objective function, which might be nonsmooth and/or noisy). We treat only the case of inequality constraints. (Equality constraints can be converted into inequalities, although the process is known to introduce degeneracy.) The poll step now incorporates positive generators for the tangent cone of the approximated active constraints, including a provision for the degenerate case. The search step has also been adapted for general linear constraints. In particular, the initial population for particle swarm (needed for the search step) is computed by first inscribing an ellipsoid of maximum volume to the feasible set. Feasibility is maintained during the search step by judiciously controlling the displacement of the particles. We have again compared PSwarm to other global solvers and the results confirm its competitiveness in terms of efficiency and robustness.

The paper is organized as follows. We start by reviewing in Section 2 the material related to the PSwarm algorithm for bound constraints. In Section 3 we provide a description of the changes introduced to the PSwarm algorithm to deal with general linear constraints. Numerical results for a wide test set of problems are presented in Section 4 (where we also introduce a new type of profiles for benchmarking of derivative-free methods). We conclude the paper in Section 5 with some conclusions and prospects of future work.

In this paper, we address linearly constrained problems written in the form

$$\min_{z\in\Omega} \quad f(z) \tag{1}$$

$$\text{s.t.} \quad Az \leq b \tag{2}$$

where

$$\Omega \;=\; \{z \in \mathbb{R}^n :\; \ell \;\leq\; z \;\leq u\}\,,$$

$A \in \mathbb{R}^{m\times n}$, and $b \in \mathbb{R}^m$. The inequalities $\ell \leq z \leq u$ are posed componentwise and $\ell \in (-\infty, \mathbb{R})^n$, $u \in (\mathbb{R}, +\infty)^n$, $\ell < u$. We explicitly separate the simple bound constraints from the remaining linear ones since we are interested in exploiting such distinction whenever possible.

2 PSwarm for bound constraints

The particle swarm algorithm simulates the behavior of a population of particles, in an attempt to widely (and, in some sense, optimally) explore the feasible region. It is a stochastic algorithm in the sense that it relies on parameters drawn from random variables, and thus, different runs for the same starting swarm may produce different outputs. It was first proposed in [12, 19] and recently used for global optimization [32, 8]. Particle swarm is based on a population (swarm) of s particles, where s is known as the population size. Each particle is associated with a velocity, which indicates the direction the particle is moving. Let t be a time instant (an iteration in the optimization context). The new position $x^i(t+1)$ of the i-th particle at time $t+1$ is computed by adding to the old position $x^i(t)$ at time t a velocity vector $v^i(t+1)$:

$$x^i(t+1) \;=\; x^i(t) + v^i(t+1), \quad i = 1, \ldots, s. \tag{3}$$

The velocity vector for a given particle at a given time is a stochastic linear combination of the velocity in the previous time instant, the direction to the particle's best position, and the direction to the best swarm position (for all particles). In fact, the velocity vector associated with each particle i is updated by

$$v_j^i(t+1) \;=\; \iota(t)v_j^i(t) + \mu\omega_{1j}(t)\left(y_j^i(t) - x_j^i(t)\right) + \nu\omega_{2j}(t)\left(\hat{y}_j(t) - x_j^i(t)\right), \tag{4}$$

for $j = 1, \ldots, n$, where $\iota(t)$ is the weighting 'inertia' factor, $\mu > 0$ is the 'cognition' parameter, and $\nu > 0$ is the 'social' parameter. The numbers $\omega_{1j}(t)$ and $\omega_{2j}(t)$, $j = 1, \ldots, n$, are randomly drawn from the uniform $(0, 1)$ distribution. In our notation, $y^i(t)$ is the position of the i-th particle with the best objective function value so far calculated, and $\hat{y}(t)$ is the particle position with the best (among all particles) objective function value found so far. Thus, the update (4) adds to the previous velocity vector a stochastic combination of the directions to the best position of the i-th particle and to the best (among all) particles position. All the coefficients in (4) can be selected by the user.

The bound constraints in the variables can be trivially enforced by (orthogonally) projecting onto Ω the new particles positions computed by (3).

Direct search methods attempt to minimize a function by comparing its value in a finite number of trial points at each iteration. This class of methods does not use or try to approximate any type of derivative information and can be subdivided into directional methods and simplicial methods (see [10]). Direct search of directional type is based on the concept of positive spanning sets and relies on the fact that a positive spanning set for $\mathbb{R}^n$ contains at least one direction of descent at a nonstationary point if the objective function is continuously differentiable there. A simple positive spanning set is the maximal positive basis formed by the coordinate vectors and the negative coordinate vectors:

$$D_{\oplus} \;=\; \{e_1, \ldots, e_n, -e_1, \ldots, -e_n\}.$$

The elementary directional direct search method based on $D_{\oplus}$ is known as coordinate search. For linearly constrained problems, it is also necessary to include in the set D of search directions those that guarantee the presence of a feasible descent direction at nonstationary points. However, when the constraints are only simple bounds, the set $D = D_{\oplus}$ includes all such feasible descent directions (see [10, 20]).

When directional direct search is applied to constrained problems where the derivatives of the constraints are available (which is clearly the case of the problems studied in this paper), the iterates are typically kept feasible. This requires an initial feasible starting point and the maintenance of feasibility throughout the iterations. In the simple bounds case both can be enforced easily. In general, one way of rejecting infeasible trial points can be accomplished by using the extreme barrier function which, in the case of simple bounds, assigns $f(z)$ to a point $z \in \Omega$ and $+\infty$ to $z \notin \Omega$.

To follow the notation of the particle swarm framework, we will use $\hat{y}(t)$ to denote the current iterate. Given a positive spanning set D and the current iterate $\hat{y}(t)$, one defines the mesh M_t and the poll set P_t. The mesh M_t is given by

$$M_t \;=\; \left\{\hat{y}(t) + \alpha(t)Dz,\; z \in \mathbb{Z}_+^{|D|}\right\}, \tag{5}$$

where $\alpha(t) > 0$ is the mesh or step size parameter, $\mathbb{Z}_+$ is the set of nonnegative integers, and $|D|$ is the cardinality of the set D. (In (5) the directions of D are represented as columns of a matrix.) The definition of the mesh (in other words, the choices of D and $\alpha(t)$) has to meet some integrality requirements [10, 20] for the method to achieve global convergence to stationary points, in other words, convergence to stationary points from arbitrary starting points. For unconstrained problems or problems with simple bounds these requirements can be trivially satisfied for the choice $D = D_{\oplus}$.

The search step conducts a finite search in the mesh M_t. The poll step is executed only if the search step fails to find a feasible point at which f is less than $f(\hat{y}(t))$. The search step is optional and it is the poll step that essentially guarantees the global convergence properties of the directional direct search methods to stationary points. The poll step evaluates the extreme barrier function at the points in the poll set

$$P_t \;=\; \{\hat{y}(t) + \alpha(t)d,\; d \in D\} \;\subset\; M_t,$$

trying to find a feasible point where f is lower than $f(\hat{y}(t))$. If the poll step fails, then the mesh size parameter must be reduced. Otherwise, the mesh size parameter is kept constant or increased. The subclass of these methods where D is kept finite across all iterations (like coordinate search) is known as (generalized) pattern search.

The hybrid method implemented in the PSwarm solver for simple bounds constrained optimization is a pattern search method that incorporates a particle swarm search in the search step. The idea is to start with an initial population and to apply one step of particle swarm at each search step. Consecutive iterations where the search steps succeed reduce to consecutive iterations of particle swarm, in an attempt to identify a neighborhood of a global minimizer. Whenever the search step fails, the poll step is applied to the best position over all particles, performing a local search in the poll set centered at this point. The points calculated in the search step by the particle swarm scheme must belong to the mesh M_t. This task can be done in several ways and, in particular, one can compute their 'projection' onto M_t. The stopping criterion of PSwarm is the conjunction of the stopping criteria for particle swarm and pattern search and can be proved to be eventually achieved under appropriate conditions. PSwarm is based on coordinate search, which guarantees global convergence to stationary points in the simple bounds case.

3 PSwarm for general linear constraints

The extension of PSwarm to general linear constraints of the form (2) must take into account both the search step (particle swarm) and the poll step. We point out first that our goal is to design an algorithm which maintains feasibility, since, in many practical applications, linear constraints are typically unrelaxable (meaning that the objective function can only be evaluated when the constraints are satisfied [10]). Also, when dealing with costly function evaluations, a feasible algorithm always provides a feasible estimate once stopped prematurely.

We describe below the main structure of the PSwarm algorithm for linearly constrained problems of the form (1)–(2) indicating in bold the differences from the pure simple bounds version (minimize $f(z)$ s.t. $z \in \Omega$). In the poll step no mechanism is explicitly used to control the displacement along the polling directions in terms of feasibility. Rather, it is directly applied the extreme barrier function

$$\hat{f}(z) = \begin{cases} f(z) & \text{if } z \in \Omega \text{ and } Az \leq b, \\ +\infty & \text{otherwise.} \end{cases}$$

The search step, as we will later see, incorporates explicit procedures to enforce feasibility before the objective function is evaluated and, therefore, there is no need here to make use of the extreme barrier function.

Algorithm 3.1

1. *Choose the stopping tolerances $\alpha_{tol} > 0$ and $v_{tol} > 0$. Choose the initial population size s. Set $\mathcal{I} = \{1, \ldots, s\}$.*

2. *Calculate (randomly) the initial feasible swarm positions $x^1(0), \ldots, x^s(0)$* **(when general linear constraints (2) are present use, e.g., the technique of the maximum volume inscribed ellipsoid)**. *Calculate (randomly) the initial swarm velocities $v^1(0), \ldots, v^s(0)$.*

3. *Set $y^i(0) = x^i(0)$, $i = 1, \ldots, s$, and $\hat{y}(0) \in \arg\min_{z \in \{y^1(0), \ldots, y^s(0)\}} f(z)$. Choose $\alpha(0) > 0$. Let $t = 0$.*

4. `[Search Step]`

   Set $\hat{y}(t+1) = \hat{y}(t)$.

   For all $i \in \mathcal{I}$ (for all particles) do:

   - *If $f(x^i(t)) < f(y^i(t))$ then*
     - *Set $y^i(t+1) = x^i(t)$ (update the particle i best position).*
     - *If $f(y^i(t+1)) < f(\hat{y}(t+1))$ then*
       - *Set $\hat{y}(t+1) = y^i(t+1)$ (update the particles best position; search step and iteration successful).*
       - *Set $\alpha(t+1) = \phi(t)\alpha(t)$ (optionally expand the mesh size parameter).*
   - *Otherwise set $y^i(t+1) = y^i(t)$.*

5. `[Poll Step]`

   Skip the poll step if the search step was successful. Compute a set of polling directions D **(either use $D_\oplus$ or compute a set of positive generators for the tangent cone of the approximated active constraints when general linear constraints (2) are present)**.

- *If there exists $d(t) \in D$ such that $\hat{f}(\hat{y}(t) + \alpha(t)d(t)) < \hat{f}(\hat{y}(t))$ then*
  - *Set $\hat{y}(t+1) = \hat{y}(t) + \alpha(t)d(t)$ (update the leader particle position; poll step and iteration successful).*
  - *Set $\alpha(t+1) = \phi(t)\alpha(t)$ (optionally expand the mesh size parameter).*
- *Otherwise, $\hat{f}(\hat{y}(t) + \alpha(t)d(t)) \geq \hat{f}(\hat{y}(t))$ for all $d(t) \in D$, and*
  - *Set $\hat{y}(t+1) = \hat{y}(t)$ (no change in the leader particle position; poll step and iteration unsuccessful).*
  - *Set $\alpha(t+1) = \theta(t)\alpha(t)$ (contract the mesh size parameter).*

6. *Compute $v^i(t+1)$, $i \in \mathcal{I}$, using (4). Compute $x^i(t+1)$, $i \in \mathcal{I}$, using equation (7)* **below**.

7. *If $\alpha(t+1) < \alpha_{tol}$ and $\|v^i(t+1)\| < v_{tol}$, for all $i \in \mathcal{I}$, then stop. Otherwise, increment t by one, drop particles in the search step if too close to each other and update $\mathcal{I}$ accordingly, and go to Step 4.*

In our implementations we typically choose $\phi(t) = 1$ or $\phi(t) = 2$ after two consecutive poll successes along the same direction and $\theta(t) = 1/2$. A particle $x^i(t)$ is dropped when there exists another one, say $x^j(t)$, such that $\|x^i(t) - x^j(t)\| \leq \alpha(t)$ and $f(x^j(t)) \leq f(x^i(t))$. Note also that we omit the projection of $x^i(t)$ onto the mesh M_t.

3.1 Generating an initial feasible population

The first issue that arises is how to generate an initial feasible population for the search step. When only simple bounds are present, an initial feasible population can be trivially calculated in Ω following an uniform distribution. Thus, one possibility would be to ignore first the linear constraints different from simple bounds and then randomly generate points in Ω. However, such a strategy may not generate a sufficiently diverse feasible population for global optimization purposes (or even fail in the sense that no feasible point is generated). See, for example, Figure 1, where an initial randomly generated population using only the simple bounds led to a population with only three feasible particles.

There are techniques to randomly generate points in a polytope (see [30] and the references therein) but require the calculation of extreme points which seemed to us too expensive and hard to code. We wanted to use something simple and efficient. The idea we explored consisted of first computing the maximum volume ellipsoid inscribed in the feasible region and then using this ellipsoid to randomly generate the points (see Figure 2). Our motivation resulted partially from the fact that there exists good state-of-the-art optimization software to calculate this type of ellipsoid [35].

Let us write the maximum volume inscribed ellipsoid using a center q and a nonsingular scaling matrix E:

$$\mathcal{E}(q, E) \;=\; \{w \in \mathbb{R}^n : \; w = q + Es, \; \|s\| \leq 1\}\,.$$

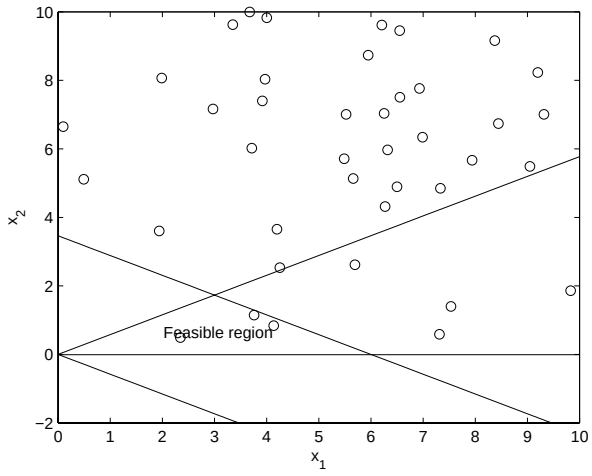

Figure 1: Feasible region for problem `hs024`. An example of an initial randomly generated population in Ω.

The initial population can then be easily generated using

$$x^i(0) \;=\; q + \varrho^{1/n} E\varsigma, \quad i = 1, \ldots, s,$$

where ϱ is a scalar drawn from the uniform distribution in $(0, 1)$ and ς is an n-dimensional vector drawn from the uniform distribution in $(-1, 1)^n$ (normalized afterwards using the ℓ_2-norm). User-provided feasible initial guesses (see Figure 2) can be easily included in the population.

The well-posedness of the problem of inscribing an ellipsoid of maximum volume into the feasible region is only guaranteed if the feasible region is bounded and, in addition, if A is full rank and there exists a point z such that $Az < b$.

In an attempt to regularize this ellipsoid calculation, one adds to the problem formulation fictitious bounds whenever the feasible region is unbounded (which is detected by first trying to inscribe an ellipsoid of maximum volume). Such fictitious bounds are used in the algorithm only for this purpose. In our implementation we made the following choices:

$$\begin{aligned}
-z_i &\leq -\min(-100, (u_i - 3|u_i|)), && \text{if} \quad \ell_i = -\infty \quad \text{and} \quad u_i \neq +\infty, \\
z_i &\leq \max(100, (\ell_i + 3|\ell_i|)), && \text{if} \quad \ell_i \neq -\infty \quad \text{and} \quad u_i = +\infty, \\
\min\left(-100, -10 \min_{j=1,\ldots,n,\, \ell_j \neq -\infty} \ell_j\right) &\leq z_i \leq \max\left(100, 10 \max_{j=1,\ldots,n,\, u_j \neq +\infty} u_j\right) \\
&\text{if} \quad \ell_i = -\infty \quad \text{and} \quad u_i = +\infty.
\end{aligned} \tag{6}$$

The computation of the ellipsoid with maximum volume inscribed into the resulting polytope is carried out in PSwarm by the interior point code[1] developed by Zhang and Gao [35].

[1]The code in [35] is originally implemented in MATLAB. We rewrote it in C using the BLAS [9] and LAPACK [7] linear algebra packages, for our own usage in the C version of PSwarm.

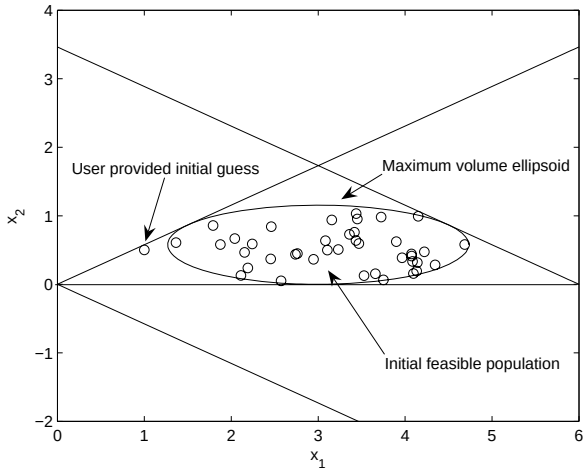

Figure 2: The maximum volume ellipsoid inscribed into the feasible region of problem `hs024`.

3.2 Imposing feasibility in the search step

To maintain feasibility of the new generated particles in the search step, we damp the displacement in (3) introducing step size parameters $\alpha^i_{j,max} > 0$, which depend on each component of each particle:

$$x^i_j(t+1) \;=\; x^i_j(t) + \alpha^i_{j,max} v^i_j(t+1), \quad j = 1, \ldots, n, \quad i \in \mathcal{I}, \tag{7}$$

where $\mathcal{I} \subseteq \{1, \ldots, s\}$ is the set of particles still in action.

The computation of $\alpha^i_{j,max} = \alpha^i_{max}\alpha^i_{j,\Omega}$ is done in two phases taking into account the structure of the constraints. The value of α^i_{max} is defined later in (9). The step size $\alpha^i_{j,\Omega}$ is the maximum step length allowed by the bound constraints:

$$\alpha^i_{j,\Omega} = \begin{cases} \min\left(1, \frac{\ell_j - x^i_j(t)}{v^i_j(t+1)}\right) & \text{if} \quad v^i_j(t+1) < 0, \\ \min\left(1, \frac{u_j - x^i_j(t)}{v^i_j(t+1)}\right) & \text{if} \quad v^i_j(t+1) > 0, \\ 1 & \text{if} \quad v^i_j(t+1) = 0. \end{cases} \tag{8}$$

To simplify the notation we now write $\bar{v}^i_j(t+1) = \alpha^i_{j,\Omega} v^i_j(t+1)$.

Let K^i be the set of indices corresponding to constraints which can lead to infeasibility by following the search direction $\bar{v}^i(t+1)$:

$$K^i \;=\; \left\{k \in \{1, \ldots, m\} : \; a_k \bar{v}^i(t+1) > 0\right\},$$

where a_k is the k-th row of the matrix A. The maximum step length along the velocity $\bar{v}^i(t+1)$ is given by

$$\alpha^i_{max} \;=\; \min_{k \in K^i} \left(1, \frac{b_k - a_k x^i(t)}{a_k \bar{v}^i(t+1)}\right). \tag{9}$$

This step length calculation allows larger steps and therefore a greater flexibility in the search phase.

Finally, we point out that the generation of an initial feasible population for the search step and the imposition of feasibility during this step automatically guarantee an initial feasible polling point $\hat{y}(0)$.

3.3 Calculating the positive generators for the poll step

As we said before, in the presence of general linear constraints, the set $D_\oplus$ of polling directions does not guarantee global convergence for generalized pattern search algorithms. The set D of directions used in the poll step must now contain positive generators for the tangent cone of the constraints which are ϵ-active ($\epsilon > 0$) at the current point. The ϵ-active are the constraints for which the residual at the current point is no larger than ϵ in absolute value:

$$\{i \in \{1, \ldots, m + 2n\} : \; c_i z - d_i \; \geq \; -\epsilon\}\,, \tag{10}$$

where we rewrote all the constraints $Az \leq b$ and $\ell \leq z \leq u$ as $Cz \leq d$ with

$$C \; = \; \begin{bmatrix} A \\ -I \\ I \end{bmatrix} \quad \text{and} \quad d \; = \; \begin{bmatrix} b \\ -\ell \\ u \end{bmatrix}.$$

(The rows of C are denoted by c_i, $i = 1, \ldots, m + 2n$.) The set D can be computed in a number of ways (see [10, 20]). One possibility is to include in D all positive generators for all the tangent cones for all $\epsilon \in [0, \epsilon_*]$, where $\epsilon_* > 0$ is independent of the iteration counter (see Lewis and Torczon [22]). Other alternatives only ask D to include the positive generators of the tangent cones of the ϵ-active constraints for the current value of ϵ, but require further provisions like the acceptability of new iterates based on a sufficient decrease condition. The approach by Lucidi, Sciandrone, and Tseng [23] requires the parameter ϵ to be reduced at unsuccessful iterations and a projection onto the feasible set during polling. Kolda, Lewis, and Torczon [21] adjust the parameter ϵ so that $\epsilon = \epsilon(t) = \mathcal{O}(\alpha(t))$. Our choice in PSwarm follows $\epsilon = \epsilon(t) = \mathcal{O}(\alpha(t))$ to avoid calculating all positive generators but ignores the sufficient decrease requirement to avoid rejecting points which yield a (simple) decrease in the function.

The set D is thus computed each time a poll step is executed by identifying first the ϵ-active constraints. At poll steps where no ϵ-active constraints are identified we set $D = D_\oplus$ as in [34]. When the matrix $\bar{C}$, associated with the ϵ-active constraints in (10), is rank deficient, it is not possible to calculate the positive generators of the tangent cone from an unique matrix factorization of $\bar{C}$. Following some of the original ideas of Lewis and Torczon [22] (also used by Abramson et al. [6] in the degenerate case), the algorithm given below (used in PSwarm) tries to dynamically decrease the parameter ϵ in order to obtain a set of ϵ-active constraints corresponding to a full row rank matrix $\bar{C}$.

When no small enough ϵ is found for which $\bar{C}$ is full row rank, the algorithm reverts to the simple mode $D = D_\oplus$. One could think that such a procedure is inappropriate and not

aligned to the basic requirements needed for global convergence of the overall algorithm. However, our numerical experience has shown us that this is a robust and efficient way of handling degeneracy. In part, this is due to the randomness features of the search step. We summarize below the algorithm used to compute the polling directions.

Algorithm 3.2

1. *Let* $\epsilon = \min(\epsilon_{init}, 10\alpha(t))$ *and* $\epsilon_{limit} = \min(0.1, \epsilon^2)$.
2. *While* $\epsilon > \epsilon_{limit}$ *do*
   (a) *Let* $\bar{C}$ *be a matrix formed by:*
      - *the rows of the matrix* A *such that* $a_k z - b_k \geq -\epsilon$, $k = 1, \ldots, m$,
      - *the rows of the matrix* I *such that* $z_j \geq u_j - \epsilon$, $j = 1, \ldots, n$.
      - *the rows of the matrix* $-I$ *such that* $z_j \leq \ell_j + \epsilon$, $j = 1, \ldots, n$.

   (b) *If* $0 < dim(\bar{C}) < n$ *and* $rank(\bar{C}) = dim(\bar{C})$ *then:*
      - *Compute a QR factorization of the matrix* $\bar{C}^\top$.
      - *Let* $B = QR^{-\top}$, $N = I - B\bar{C}$, *and stop with* $D = [B \;\; -B \;\; N \;\; -N]$.

   (c) *If* $dim(\bar{C}) = 0$ *then stop and consider* $D = D_\oplus$, *else* $\epsilon = \epsilon/2$.
3. *If no* D *has been computed (the condition of the while loop has become false) consider also* $D = D_\oplus$.

In our tests we set $\epsilon_{init} = 0.1$.

4 Numerical results

We have numerically compared PSwarm to other existing solvers for the derivative-free optimization of functions subject to linear constraints, having in mind the goal of global optimization.

4.1 Test problems

To obtain a sufficiently large set of test problems we searched all known databases of non-linear programming problems. We were able to gather 110 linearly constrained problems from a total of 1564 problems, collected from the following sources: Vanderbei [33] (given in AMPL, which includes the CUTE [15] collection), GLOBALlib [2] (available in AMPL format at [27]), one problem from [31], three problems from [17], one from [24], and four from [25].

The 110 problems collected were all written in AMPL [14]. They include 23 problems with a linear objective function, 55 with a quadratic objective function, and 32 with a non-quadratic objective function.

n	3	10	15	20	25	30	35	40	45	50
m	2	5	10	15	20	25	30	35	40	45
m_a	1	2	5	7	10	12	15	17	20	22

Table 1: Dimensions of the 10 highly nonconvex problems.

Ten additional highly nonconvex problems were obtained by random generation of the linear constraints, following the scheme reported in [28]. For these additional problems, the objective function (see Pinter [29]) is given by

$$r\, n \sum_{i=1}^{n} (x_i - x_i^*)^2 + \left(\sin\left(g_1\, P_1(x)\right)\right)^2 + \left(\sin\left(g_2\, P_2(x)\right)\right)^2,$$

where

$$P_1(x) = \sum_{i=1}^{n} (x_i - x_i^*)^2 + \sum_{i=1}^{n} (x_i - x_i^*)^2,$$
$$P_2(x) = \sum_{i=1}^{n} (x_i - x_i^*),$$

$r = 0.025$, $g_1 = 1$, and $g_2 = 1$. These problems have simple bound constraints on all variables ($x \in [-10, 10]^n$) and the linear constraints are randomly generated using the following procedure (m_a is the number of active linear constraints at the global minimizer):

Algorithm 4.1

1. *Randomly generate the solution x^* from an uniform distribution (in the simple bound domain Ω).*
2. *Randomly generate the elements of the matrix A from the uniform distribution in $(-10, 10)$. Denote the rows of A by a_k, $k = 1, \ldots, m$.*
3. *Let $b_k = a_k x^*$, $k = 1, \ldots, m_a$.*
4. *Let $b_k = a_k x^* + y$, where y is a random number drawn from an uniform distribution in $(1, 10)$ and $k = m_a + 1, \ldots, m$.*

The 10 problems selected resulted from the combination of the parameters n, m, and m_a, reported in Table 1.

4.2 Solvers tested

The solvers used in our numerical comparisons were ASA, NOMADm, and DIRECT.

ASA [16] stands for *Adaptative Simulated Annealing* and is written in C. We used the ASA-AMPL interface previous developed for [34]. Note that ASA uses the extreme barrier function to reject infeasible trial points.

NOMADm [4] is a MATLAB [3] version of the NOMAD solver [5] for black-box optimization. We were not able to use the particle swarm option that NOMADm incorporates in the search step because it is only available for problems with simple bounds. We have selected a maximal positive basis as in PSwarm. Note that PSwarm follows a simplified version of the way in which NOMADm handles the computation of the positive generators of the ϵ-active constraints.

DIRECT (*DIviding RECTangles*) is a MATLAB implementation [1] of the method described in [18]. DIRECT uses a penalty strategy to deal with constraints. The penalty parameters are fixed for each constraint and kept constant during all the iterations. In our testing we used 10^6 for all constraints. We did some additional testing to see if the numerical results could be improved by perturbing the values of the penalty parameters but no significant differences were observed.

To test PSwarm, NOMADm, and DIRECT, we considered the problems directly coded in AMPL and used the AMPL-MATLAB interface developed for this purpose.

A critical issue that relates all the solvers is the choice of the initial guess. PSwarm allows the user to specify an initial guess (in fact, the user can provide an initial population) which is included in the initial population if shown to be feasible. For NOMADm it is mandatory to provide an initial guess. When the provided guess is not feasible, NOMADm tries to project the provided point onto the feasible region. ASA also expects an initial guess, but it does not ask this initial guess to be feasible. (When the initial guess is infeasible, it tries to compute a feasible initial point as infeasible points are automatically rejected.) No initial guesses can be given to DIRECT. We also point out that some of the problems coded by us in AMPL do not include an initial guess. Thus, in order to be as fair as possible to all solvers, no initial guess is considered and, when requested, it is randomly generated within the bound constraints following an uniform distribution. We used the fictitious bounds (6) for this purpose. While PSwarm, NOMADm, and ASA use these bounds solely for the calculation of an initial guess or population, DIRECT uses them during the optimization phase.

No pre-processing of bound or linear constraints is done before calling the solvers. PSwarm performs no pre-processing of constraints, and, as far as we know, neither ASA or DIRECT do. NOMADm performs an internal scaling of variables and constraints based on the ℓ_∞-norm.

Finally, we point out that the number of particles chosen for PSwarm was $s = 40$. For the remaining swarm parameters, we used the values from [34]: $\nu = \mu = 0.5$ and $\iota(t)$ is linearly interpolated between 0.9 and 0.4, *i.e.*, $\iota(t) = 0.9 - (0.5/t_{max})t$, where t_{max} is the maximum number of iterations permitted.

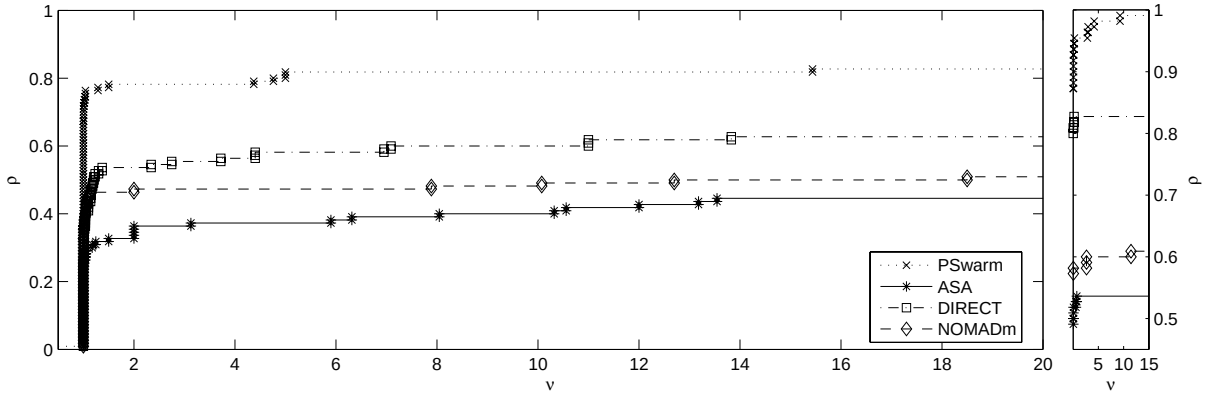

Figure 3: Performance profiles for the 110 problems (minimum objective function value for 10 runs with $maxf = 2000$).

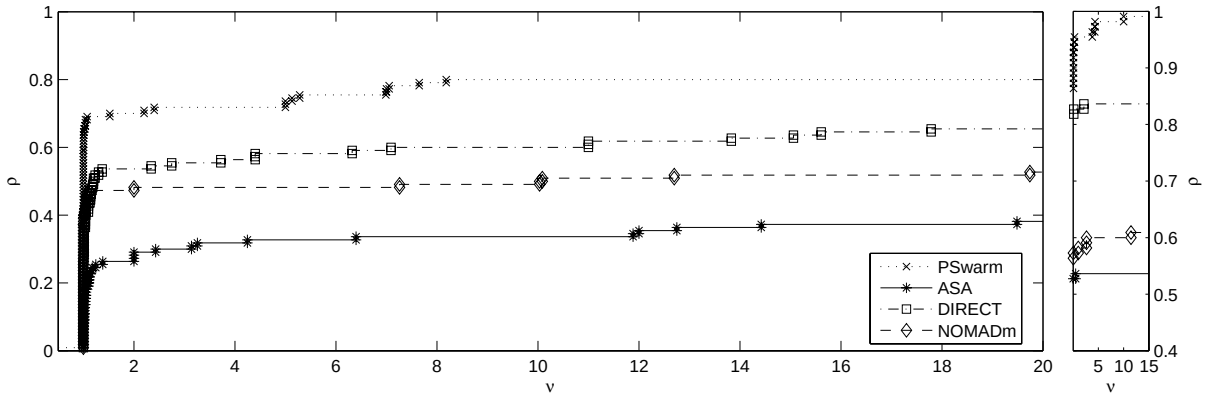

Figure 4: Performance profiles for the 110 problems (average objective function value for 10 runs with $maxf = 2000$).

4.3 Numerical results (performance profiles)

Figures 3–5 depict performance profiles obtained by using the procedure described in [34] (a modification of the performance profiles from [11]) for the 4 solvers and the 110 test problems (imposing a maximum of 2000 total function evaluations). See the appendix for information about these profiles. The stochastic solvers (ASA and PSwarm) were run 10 times for each of the problems. Then, from the 10 runs, we computed the final minimum, maximum, and average objective function values.

PSwarm attempted a poll step, in average, in approximately 70% of the iterations. Thus, 70% of the iterations corresponded to unsuccessful search steps. The percentage of successful poll steps was approximately 56%.

For the 10 highly nonconvex problems we imposed a maximum of 10000 total function

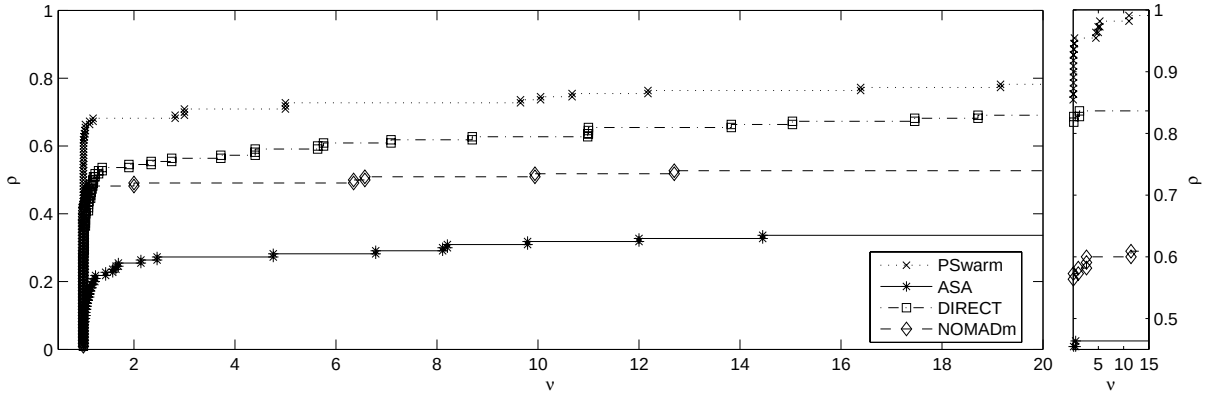

Figure 5: Performance profiles for the 110 problems (maximum objective function value for 10 runs with $maxf = 2000$).

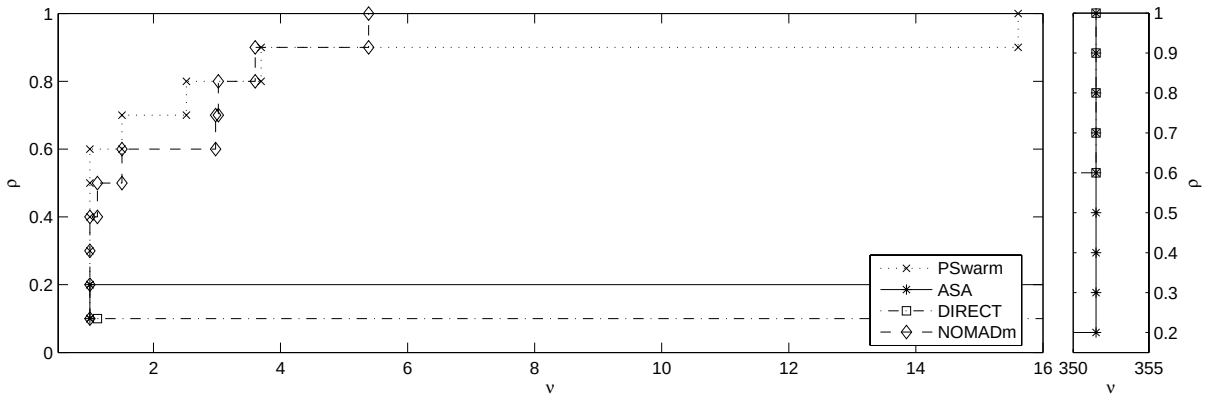

Figure 6: Performance profiles for the 10 highly nonconvex problems (minimum objective function value for 10 runs with $maxf = 10000$).

evaluations. The performance profiles are shown in Figures 6–8. Since NOMADm is not designed for global optimization we ran it 10 times for different randomly generated initial guesses, a procedure we only applied for these test problems. Note that NOMADm does not appear in Figure 8 since it fails for at least one run for all problems (and therefore the worst performance is always a failure). The NOMADm failures occur when no feasible starting point is provided. In that case, NOMADm attempts to compute a feasible point by solving an LP problem using the MATLAB linprog function; however, this is not always successful at generating a feasible point.

Since the 110 test problems considered include linear, quadratic, and non-quadratic objective functions, we also looked at the performance profiles for each type of problems. To shorten the presentation, we present here only the performance profiles for the non-quadratic objective functions using average objective function values (see Figure 9). The

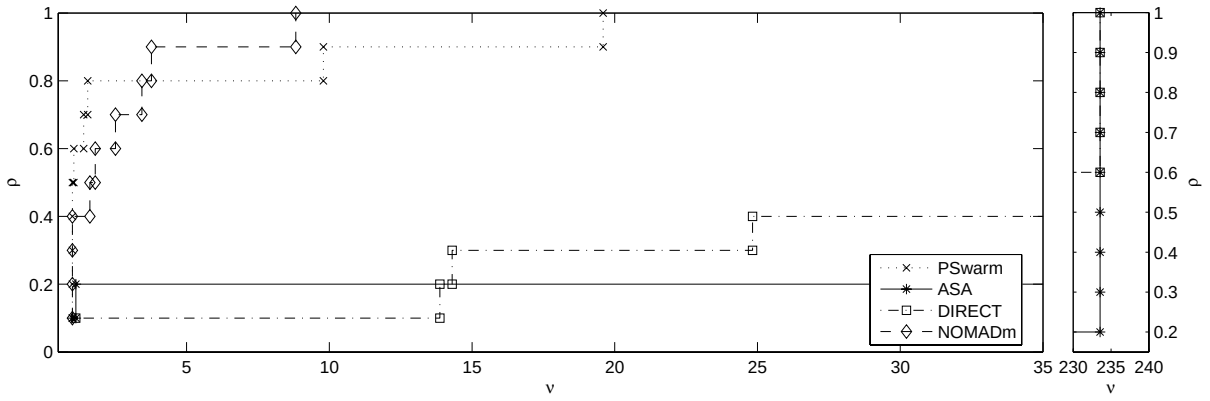

Figure 7: Performance profiles for the 10 highly nonconvex problems (average objective function value for 10 runs with $maxf = 10000$).

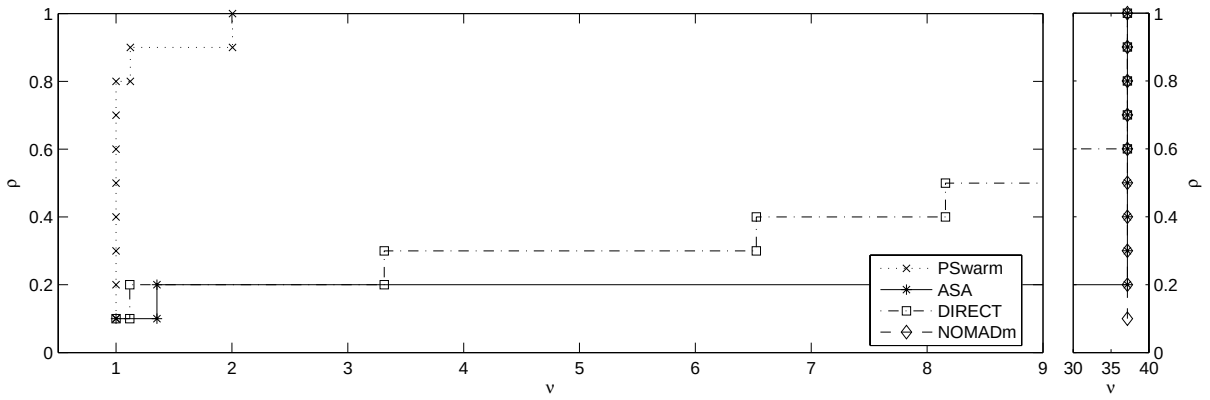

Figure 8: Performance profiles for the 10 highly nonconvex problems (maximum objective function value for 10 runs with $maxf = 10000$).

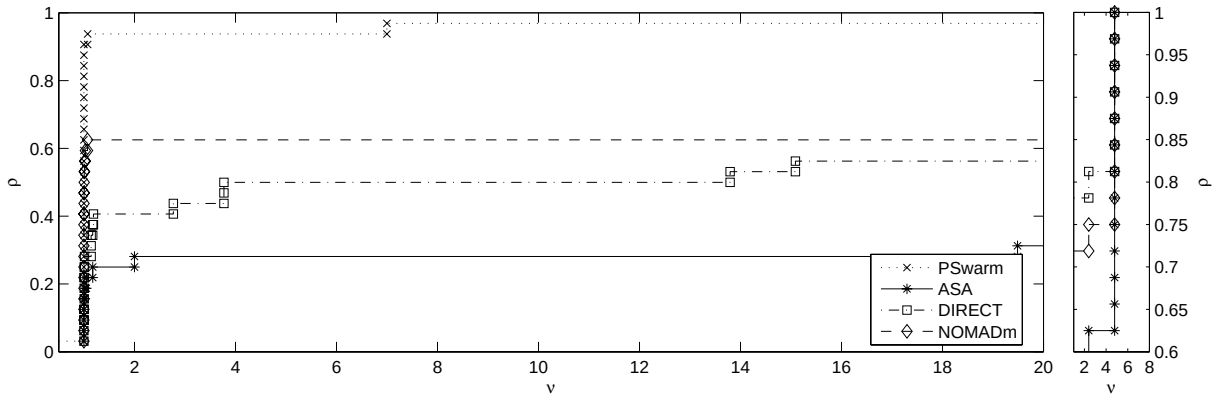

Figure 9: Performance profiles for the subset of 110 problems with non-quadratic objective functions (average objective function value for 10 runs with $maxf = 2000$).

remaining performance profiles can be seen at the PSwarm web page `http://www.norg.uminho.pt/aivaz/pswarm`. For the linear objective function problems, PSwarm was the most effective and robust solver. In the quadratic objectives case, we observed a small advantage of DIRECT, and for the non-quadratic objective function problems PSwarm was again the most effective and robust solver.

4.4 Numerical results (function profiles)

The performance profiles presented before measure the efficiency and robustness of the solvers when a maximum number of function evaluations is imposed in terms of the quality of the final value of the objective function. These profiles do not show how effective and robust each solver is in terms of the number of objective function evaluations necessary to compute a global minima (or to achieve some reduction in the objective function value).

Our first attempt to measure performance differently was to use the recently proposed *data profiles* [26] for derivative-free optimization. These profiles measure how well a solver does when asked to achieve a certain level of accuracy within some computational budget (CPU time or total number of function evaluations). However, these data profiles are not so practical in our case because some solvers are stochastic, and more importantly, do not necessarily produce a monotone decreasing sequence of best objective function values. Furthermore, since the goal of this paper is global optimization, the information contained in data profiles for smaller values of the budget is not so relevant.

In this paper we propose what we call *function profiles* to measure the efficiency and robustness of global derivative-free solvers in terms of function evaluations required to achieve some level of global optimality. To explain how our profiles are calculated, let $\mathcal{P}$ be a set of test problems and $\mathcal{S}$ a set of solvers. Define $r_{p,s}$ as the number of objective function evaluations taken by solver s to solve problem p, for $p \in \mathcal{P}$ and $s \in \mathcal{S}$. The value of $r_{p,s}$ is set to $+\infty$ whenever a failure occurs, i.e., when solver s is unable to provide a

feasible point for problem p. A failure is also declared when solver s is unable to produce a feasible point for problem p within a specified relative error τ, i.e., $r_{p,s}$ is set to $+\infty$ when $f_{p,s} - f_{p,L} > \tau|f_{p,L}|$, where $f_{p,s}$ is the objective function value obtained by the solver s on problem p and $f_{p,L}$ is the best objective function obtained by all the solvers for problem p. We define the *function profile* $\rho_s(\nu)$ of a solver $s \in \mathcal{S}$ as the fraction of problems where the number of objective function evaluations is lower than ν

$$\rho_s(\nu) \;=\; \frac{1}{|\mathcal{P}|} size\{p \in \mathcal{P} : r_{p,s} \;<\; \nu\}.$$

The values of $\rho_s(\nu)$ are calculated setting a limit for the number of function evaluations and letting the solvers stop when their stopping criteria are met. We used 2000 for the 110 problems test set and 10000 for the 10 highly nonconvex problems.

Figures 10 and 11 depict the function profiles for, respectively, the 110 test set and the set of 10 highly nonlinear test problems. Due to the stochasticity of some of the solvers, the quantity $r_{p,s}$ represents here the average number of function evaluations (for the 10 runs), and $f_{p,s}$ and $f_{p,L}$ are the average function values. We report results for $\tau = 0.1$ but no major differences were observed with different values.

By looking at function profiles, one can obtain useful information on the solvers performance in term of function evaluations needed for global optimization. For example from Figure 10, we observe that NOMADm solved about 40% of the problems using less than 1000 function evaluations, while PSwarm solved about 20%. Considering $\nu = 2000$, we infer that PSwarm is able to solve about 70% of the problems, and thus that it is the most robust among all.

DIRECT never uses less than the provided budget defined in terms of the total number of function evaluations (and for some problems it significantly exceeds the imposed budget). Figure 11 does not include DIRECT because this code was unable to solve any of the problems up to the requested accuracy ($\tau = 0.1$).

5 Conclusions and future work

The main goal of this paper was to extend PSwarm [34] to general linear constraints. We were mainly motivated from the fact that PSwarm yielded encouraging results for problems with simple bounds. In some applications the constraints assume a more general linear form which prohibits the application of this older version of PSwarm, in particular when such constraints are unrelaxable. We studied various possibilities to extend PSwarm to linear constraints and the presentation of this paper is the result of intensive testing.

This paper also contributes to the field of global derivative-free optimization by reporting a comprehensive numerical comparison of different solvers. For this purpose we collected a vast collection of linearly constrained optimization problems, a number of them nonconvex, which can be used by others researchers to perform their testing. Finally, we introduced new function profiles (different from the data profiles [26]) for the assessment

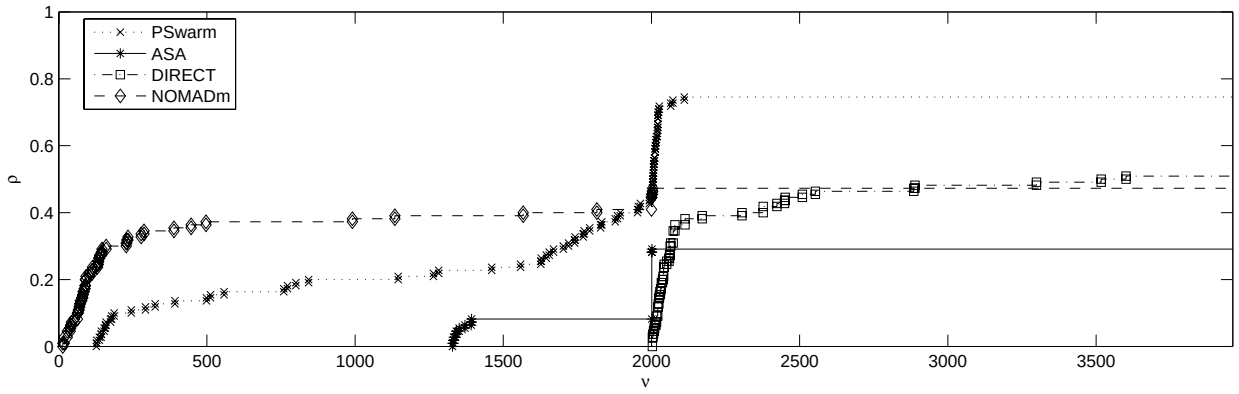

Figure 10: Function profiles for the 110 problems (average objective function value for 10 runs).

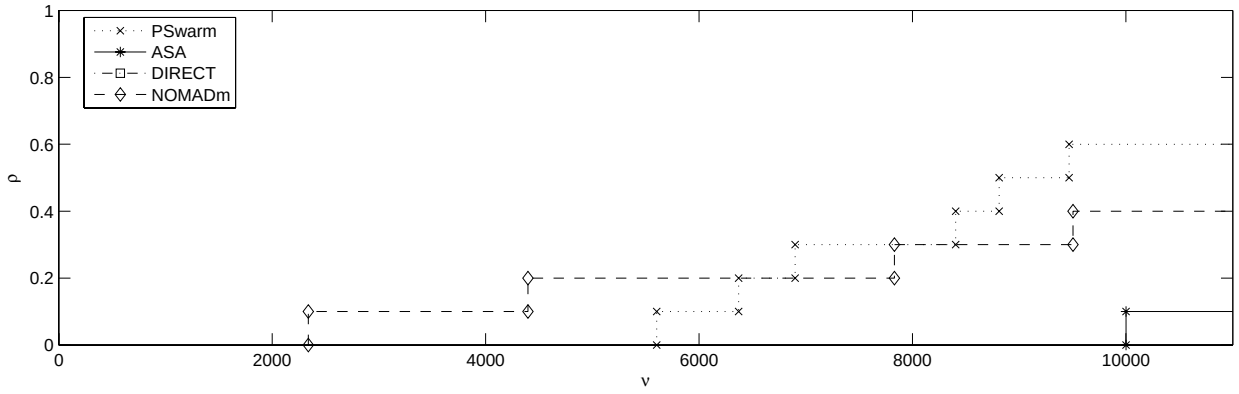

Figure 11: Function profiles for the 10 highly nonconvex problems (average objective function value for 10 runs).

of the efficiency and robustness of solvers in terms of the number of function evaluations needed to achieve a certain level of global optimality.

The natural next step is to try to handle nonlinear constraints. It is not clear to us how to proceed toward this goal. We plan to have a beta version soon which embeds PSwarm for linear constraints into some penalty or augmented Lagrangian scheme, but this might not be the way to proceed. Our experience has shown us that global derivative-free optimization is an extremely difficult field where good performance is the result of intensive research — and thus the definite extension of PSwarm to nonlinear constraints is expected to take significant effort.

Appendix

The performance ratio $\rho_s(\tau)$ used in the performance profiles is defined by setting $\rho_s(\tau) = \frac{1}{|\mathcal{P}|}\text{size}\{p \in \mathcal{P} : r_{p,s} \leq \tau\}$, where $r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s}:s\in\mathcal{S}\}}$, $\mathcal{P}$ is the test set, $\mathcal{S}$ is the set of solvers, and $t_{p,s}$ is the value obtained by solver $s \in \mathcal{S}$ on test problem $p \in \mathcal{P}$. The value of $\rho_s(1)$ is the probability that the solver will win over the remaining ones. The limit of $\rho_s(\tau)$ when τ tends to $+\infty$ gives the percentage of problems solved by the solver. Our performance profile measure for global optimization is defined as [34]:

$$t_{p,s} = \begin{array}{l}\text{(best/average/worst) objective function value obtained for} \\ \text{problem } p \text{ by solver } s \text{ (for all runs if solver } s \text{ is stochastic)},\end{array}$$

$$r_{p,s} = \begin{cases} 1 + t_{p,s} - \min\{t_{p,s} : s \in \mathcal{S}\} & \text{when } \min\{t_{p,s} : s \in \mathcal{S}\} < \epsilon, \\ \frac{t_{p,s}}{\min\{t_{p,s}:s\in\mathcal{S}\}} & \text{otherwise.} \end{cases}$$

We set $\epsilon = 0.001$.

References

[1] DIRECT — A Global Optimization Algorithm. `http://www4.ncsu.edu/~ctk/Finkel_Direct`.

[2] GLOBAL Library. `http://www.gamsworld.org/global/globallib.htm`.

[3] MATLAB, The MathWorks Inc. `http://www.mathworks.com`.

[4] NOMADm Optimization Software. `http://www.gerad.ca/NOMAD/nomadm.html`.

[5] The NOMAD Project. `http://www.gerad.ca/NOMAD`.

[6] M. A. Abramson, O. A. Brezhneva, J. E. Dennis, and R. L. Pingel. Pattern search in the presence of degenerate linear constraints. *Optim. Methods Softw.*, 23:297–319, 2008.

[7] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, third edition, 1999.

[8] F. van den Bergh and A. P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Inform. Sci.*, 176:937–971, 2006.

[9] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An updated set of basic linear algebra subprograms (BLAS). *ACM Trans. Math. Software*, 28:135–151, 2002.

[10] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization*. MPS-SIAM Series on Optimization. SIAM, Philadelphia, 2009.

[11] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91:201–213, 2002.

[12] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, Nagoya, Japan, 1995. IEEE Service Center, Piscataway, NJ.

[13] J. M. Fernandes, A. I. F. Vaz, and L. N. Vicente. Modelling nearby FGK Population I stars: A new form of estimating stellar parameters using an optimization approach. Technical Report 08-50, Department of Mathematics, University of Coimbra, 2008.

[14] R. Fourer, D. M. Gay, and B. W. Kernighan. A modeling language for mathematical programming. *Management Sci.*, 36:519–554, 1990.

[15] N. I. N. Gould, D. Orban, and Ph. L. Toint. CUTEr (and SifDec), a Constrained and Unconstrained Testing Environment, revisited. *ACM Trans. Math. Software*, 29:373–394, 2003. `http://cuter.rl.ac.uk/cuter-www`.

[16] L. Ingber. Adaptative simulated annealing (ASA): Lessons learned. *Control Cybernet.*, 25:33–54, 1996.

[17] Y. Ji, K.-C. Zhang, and S.-J. Qu. A deterministic global optimization algorithm. *Appl. Math. Comput.*, 185:382–387, 2007.

[18] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the Lipschitz constant. *J. Optim. Theory Appl.*, 79:157–181, 1993.

[19] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, pages 1942–1948, Perth, Australia. IEEE Service Center, Piscataway, NJ.

[20] T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Rev.*, 45:385–482, 2003.

[21] T. G. Kolda, R. M. Lewis, and V. Torczon. Stationarity results for generating set search for linearly constrained optimization. *SIAM J. Optim.*, 17:943–968, 2006.

[22] R. M. Lewis and V. Torczon. Pattern search methods for linearly constrained minimization. *SIAM J. Optim.*, 10:917–941, 2000.

[23] S. Lucidi, M. Sciandrone, and P. Tseng. Objective-derivative-free methods for constrained optimization. *Math. Program.*, 92:37–59, 2002.

[24] Z. Michalewicz. Evolutionary computation techniques for nonlinear programming problems. *International Transactions in Operational Research*, 1:223–240, 1994.

[25] Z. Michalewicz. *Genetic Algorithms+ Data Structures= Evolution Programs*. Springer, Berlin, third edition, 1996.

[26] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optim.*, 2009, to appear. `http://www.mcs.anl.gov/~more/dfo`.

[27] A. Neumaier. The COCONUT benchmark. `http://www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.htm%l`.

[28] P. Parpas, B. Rustem, and E. N. Pistikopoulos. Linearly constrained global optimization and stochastic differential equations. *J. Global Optim.*, 36:191–217, 2006.

[29] J. Pintér. *Global Optimization: Software, Test Problems and Applications*, volume 62 of *Nonconvex Optimization and Applications*, chapter 15, pages 515–569. Kluwer Academic Publishers, Dordrecht, 2002.

[30] P. A. Rubin. Generating random points in a polytope. *Comm. Statist. Simulation Comput.*, 13:375–396, 1984.

[31] T. P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4:284–294, 2000.

[32] J. F. Schutte and A. A. Groenwold. A study of global optimization using particle swarms. *J. Global Optim.*, 31:93–108, 2005.

[33] R. J. Vanderbei. Benchmarks for Nonlinear Optimization. `http://www.princeton.edu/~rvdb/bench.html`.

[34] A. I. F. Vaz and L. N. Vicente. A particle swarm pattern search method for bound constrained global optimization. *J. Global Optim.*, 39:197–219, 2007.

[35] Y. Zhang and L. Gao. On numerical solution of the maximum volume ellipsoid problem. *SIAM J. Optim.*, 14:53–76, 2003.