Pré-Publicações do Departamento de Matemática
Universidade de Coimbra
Preprint Number 06–08

A PARTICLE SWARM PATTERN SEARCH METHOD FOR BOUND CONSTRAINED NONLINEAR OPTIMIZATION

A. ISMAEL F. VAZ AND L. N. VICENTE

Abstract: In this paper we develop, analyze, and test a new algorithm for the global minimization of a function subject to simple bounds without the use of derivatives. The underlying algorithm is a pattern search method, more specifically a coordinate search method, which guarantees convergence to stationary points from arbitrary starting points.

In the optional search phase of pattern search we apply a particle swarm scheme to globally explore the possible nonconvexity of the objective function. Our extensive numerical experiments showed that the resulting algorithm is highly competitive with other global optimization methods also based on function values.

Keywords: Direct search, pattern search, particle swarm, derivative free optimization, global optimization, bound constrained nonlinear optimization.
AMS Subject Classification (2000): 90C26, 90C30, 90C56.

1. Introduction

Pattern and direct search methods are one of the most popular classes of methods to minimize functions without the use of derivatives or of approximations to derivatives [23]. They are based on generating search directions which positively span the search space. Direct search is conceptually simple and natural for parallelization. These methods can be designed to rigorously identify points satisfying stationarity for local minimization (from arbitrary starting points). Moreover, their flexibility can be used to incorporate algorithms or heuristics for global optimization, in a way that the resulting direct or pattern search method inherits some of the properties of the imported global optimization technique, without jeopardizing the convergence for local stationarity mentioned before.

The particle swarm optimization algorithm was firstly proposed in [11, 21] and has deserved some recent attention in the global optimization community. The particle swarm algorithm tries to simulate the social behaviour of

Received February 11, 2006.
Support for this work was provided by Centro de Matemática da Universidade de Coimbra, by Algoritmi Research Center, and by FCT under grants POCI/MAT/59442/2004 and POCI/MAT/58957/2004.

a population of agents or particles, in an attempt to optimally explore some given problem space. During time (iterations in the optimization context), each particle is associated with a stochastic velocity vector which indicates where the particle is moving to. The velocity vector for a given particle at a given time is a linear stochastic combination of the velocity in the previous time instant, of the direction to the particle's best ever position, and of the direction to the best ever swarm positions (for all particles). The particle swarm algorithm 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. Some of its advantages are being simple to implement and easy to parallelize. It depends, however, on a number of parameters which influence the rate of convergence in the vicinity of the global optimum. Particle swarm seems to outperform genetic algorithms on difficult problem classes, namely for unconstrained global optimization problems [6].

The goal of this paper is to show how particle swarm can be incorporated in the pattern search framework. The resulting particle swarm pattern search algorithm is still a pattern search algorithm, producing sequences of iterates along the traditional requirements for this class of methods (based on integer lattices and positive spanning sets). The new algorithm is better equipped for global optimization because it is more aggressive in the exploration of the search space. Our numerical experiences showed that a large percentage of the computational work is spent in the particle swarm component of pattern search.

Within the pattern search framework, the use of the search step for surrogate optimization [5, 27] or global optimization [1] is an active area of research. Hart has also used evolutionary programming to design evolutionary pattern search methods (see [14] and the references therein). There are some significative differences between his work and ours. First, we are exclusively focused on global optimization and our heuristic is based on particle swarm rather then on evolutionary algorithms. Further, our algorithm is deterministic in its pattern search component. As a result, we obtain that a subsequence of the mesh size parameters tends to zero is the deterministic sense rather than with probability one like in Hart's algorithms.

We are interested in solving mathematical problems of the form

$$\min_{z\in\mathbb{R}^n} f(z) \quad \text{s.t.} \quad \ell \;\leq\; z \;\leq u,$$

where the inequalities $\ell \leq z \leq u$ are posed componentwise. There is no need to assume any type on smoothness on the objective function $f(z)$ to apply particle swarm or pattern search. To study the convergence properties of pattern search, and thus of the particle swarm pattern search method, one has to impose some smoothness on $f(z)$, the necessary to characterize stationarity at local minimizers.

The next two sections are used to describe the particle swarm paradigm and the basic pattern search framework. We introduce the particle swarm pattern search method in Section 4. The convergence and termination properties of the proposed method are discussed in Section 5. A brief review about the optimization solvers used in the numerical comparisons and implementation details about our method are given in Section 6. The numerical results are presented in Section 7 for a large set of problems. We end the paper in Section 8 with conclusions and directions for future work.

2. Particle swarm

In this section, we briefly describe the particle swarm optimization algorithm. Our description follows the presentation of the algorithm tested in [6] and the reader is pointed to [6] for other algorithmic variants and details.

The particle swarm optimization algorithm is based on a population (swarm) of particles. Each particle is associated with a velocity which indicates where the particle is moving to. Let t be a time instant. Suppose that there are s particles (where s is known as the population size). 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 $v^i(t+1)$:

$$x^i(t+1) \;=\; x^i(t) + v^i(t+1), \tag{1}$$

for $i = 1, \ldots, s$.

The velocity vector associated to each particle i is updated by

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

for $j = 1, \ldots, n$, where $\iota(t)$ is a weighting factor (called *inertial*) and μ and ν are positive real parameters (called, in the particle swarm terminology, the *cognition* parameter and the *social* parameter, respectively). The numbers $\omega_{1j}(t)$ and $\omega_{2j}(t)$ are randomly drawn from the uniform $(0,1)$ distribution used for each dimension $j = 1, \ldots, n$. Finally, $y^i(t)$ 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. The position $\hat{y}(t)$ can be rigorously described as

$$\hat{y}(t) \;=\; \mathrm{argmin}_{z \in \{y^1(t),\ldots,y^s(t)\}} f(z).$$

The bound constraints in the variables are enforced by considering the projection onto $\Omega = \{x \in \mathbb{R}^n : \; \ell \leq x \leq u\}$, given for all particles $i = 1, \ldots, s$ by

$$proj_\Omega(x^i_j(t)) = \begin{cases} \ell_j & \text{if } \; x^i_j(t) < \ell_j, \\ u_j & \text{if } \; x^i_j(t) > u_j, \\ x^i_j(t) & \text{otherwise,} \end{cases} \tag{3}$$

for $j = 1, \ldots, n$. This projection must be applied to the new particles positions computed by equation (1).

The stopping criterion of the algorithm should be practical and has to ensure proper termination. One possibility is to stop when the norm of the velocities vector is small for all particles. It is possible to prove under some assumptions and for some algorithmic parameters that the expected value of the norm of the velocities vectors tends to zero for all particles (see also the analysis presented in the Section 5).

The particle swarm optimization algorithm is described in Algorithm 2.1.

Algorithm 2.1.

(1) Choose a population size s and a stopping tolerance $v_{tol} > 0$. Randomly initialize the initial swarm $\left\{x^1(0), \ldots, x^s(0)\right\}$ and the initial swarm velocities $v^1(0), \ldots, v^s(0)$.

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

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

For $i = 1, \ldots, s$ do (for every particle i):

- *Compute $\hat{x}^i(t) = proj_\Omega(x^i(t))$.*
- *If $f(\hat{x}^i(t)) < f(y^i(t))$ then*
  - *Set $y^i(t+1) = \hat{x}^i(t)$ (update the particle i best position).*
  - *If $f(y^i(t+1)) < f(\hat{y}(t+1))$ then $\hat{y}(t+1) = y^i(t+1)$ (update the particles best position).*
- *Otherwise set $y^i(t+1) = y^i(t)$.*

(4) Compute $v^i(t+1)$ and $x^i(t+1)$, $i = 1, \ldots, s$, using formulae (1) and (2).

(5) If $\|v^i(t+1)\| < v_{tol}$, for all $i = 1, \ldots, s$, then stop. Otherwise, increment t by one and go to Step 3.

3. Pattern search

Direct search methods are an important class of optimization algorithms which attempt to minimize a function by comparing, at each iteration, its value in a finite set of trial points (computed by simple mechanisms). Direct search methods not only do not use any derivative information but also do not try to implicitly build any type of derivative approximation. Pattern search methods can be seen as direct search methods for which the rules of generating the trial points follow stricter calculations and for which convergence for stationary points can be proved from arbitrary starting points. A comprehensive review of direct and pattern search can be found in [23], where the terminology 'generating set search' methods is adopted instead. In this paper, we prefer to describe pattern search methods using the search/poll step framework [3] since it suits better the incorporation of heuristic procedures.

The central notion in direct or pattern search is positive spanning. The definitions and properties of positive spanning sets and of positive bases are given, for instance, in [9, 23]. One of the simplest positive spanning sets is formed by the vectors of the canonical basis and its symmetrical counterparts:

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

(The set $D_\oplus$ is also a (maximal) positive basis.) The elementary direct search method based on this positive spanning set is known as coordinate or compass search and its structure is basically all we need from direct or pattern search in our paper.

Given a positive spanning set D and the current iterate* $y(t)$, we define two sets of points: the mesh M_t and the poll set P_t. The mesh M_t is given by

$$M_t \;=\; \left\{ y(t) + \alpha(t) D z,\; z \in \mathbb{N}_0^{|D|} \right\},$$

where $\alpha(t) > 0$ is the mesh size parameter (also known as the step-length control parameter). The mesh has to meet some integrality requirements for the method to achieve global convergence to stationary points, in other words, convergence to stationary points from arbitrary starting points. In particular, the matrix D has to be of the form $G\hat{Z}$, where $G \in \mathbb{R}^{n\times n}$ is

*We will use $y(t)$ to denote the current iterate, rather than x_k or y_k, to follow the notation of the particle swarm framework.

a nonsingular generating matrix and $\hat{Z} \in \mathbb{Z}^{n\times|D|}$. The positive basis $D_\oplus$ satisfies this requirement trivially.

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 point for which f is lower than $f(y(t))$. The poll step evaluates the function at the points in the poll set

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

trying to find a point where f is lower than $f(y(t))$. Note that P_t is a subset of M_t. If f is continuously differentiable at $y(t)$, the poll step is guaranteed to succeed if $\alpha(t)$ is sufficiently small, since the positive spanning set D contains at least one direction of descent (which makes an acute angle with $-\nabla f(y(t))$). Thus, if the poll step fails then the mesh size parameter must be reduced. It is the poll step that guarantees the global convergence of the pattern search method.

In order to generalize pattern search for bound constrained problems it is necessary to use a feasible initial guess $y(0) \in \Omega$ and to keep feasibility of the iterates by rejecting any trial point that is out of the feasible region. Rejecting infeasible trial points can be accomplished by applying a pattern search algorithm to the following penalty function

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

The iterates produced by a pattern search method applied to the unconstrained problem of minimizing $\hat{f}(x)$ coincide trivially with those generated by the same type of pattern search method, but applied to the minimization of $f(z)$ subject to simple bounds and to the rejection of infeasible trial points.

It is also necessary to include in the search directions D those directions that guarantee the presence of a descent direction at any nonstationary point of the bound constrained problem. One can achieve this goal in several ways, but, rather then entering into many details about this matter, we prefer to say that the set of directions in $D_\oplus$ is sufficiently rich to fulfill this goal in the case of bound constraints.

In order to completely describe the basic pattern search algorithm, we need to specify how to expand and contract the mesh size or step-length control parameter $\alpha(t)$. These expansions and contractions use the factors $\phi(t)$ and

$\theta(t)$, respectively, which must obey to the following rules:

$$\begin{array}{lll} \phi(t) = \tau^{\ell_t}, & \ell_t \in \mathbb{N}_0, & \text{if } t \text{ is successful}, \\ \theta(t) = \tau^{m_t}, & m_t \in \mathbb{Z}^-, & \text{if } t \text{ is unsuccessful}, \end{array}$$

where $\tau > 1$ is a positive integer chosen at the beginning of the method and unchanged with t. For instance, we can have $\theta(t) = 1/2$ for unsuccessful iterations and $\phi(t) = 1$ or $\phi(t) = 2$ for successful iterations.

The basic pattern search method for use in this paper is described in Algorithm 3.1.

Algorithm 3.1.

(1) Choose a positive integer τ and the stopping tolerance $\alpha_{tol} > 0$. Choose the positive spanning set $D = D_\oplus$.

(2) Let $t = 0$. Select an initial feasible guess $y(0)$. Choose $\alpha(0) > 0$.

*(3) **[Search Step]***

Evaluate f at a finite number of points in M_t. If a point $z(t) \in M_t$ is found for which $\hat{f}(z(t)) < \hat{f}(y(t))$ then set $y(t+1) = z(t)$, $\alpha(t+1) = \phi(t)\alpha(t)$ (optionally expanding the mesh size parameter), and declare successful both the search step and the current iteration.

*(4) **[Poll Step]***

Skip the poll step if the search step was successful.

- *If there exists $d(t) \in D$ such that $\hat{f}(y(t) + \alpha(t)d(t)) < \hat{f}(y(t))$ then*
  - *Set $y(t+1) = y(t) + \alpha(t)d(t)$ (poll step and iteration successful).*
  - *Set $\alpha(t+1) = \phi(t)\alpha(t)$ (optionally expand the mesh size parameter).*
- *Otherwise, $\hat{f}(y(t) + \alpha(t)d(t)) \geq \hat{f}(y(t))$ for all $d(t) \in D$, and*
  - *Set $y(t+1) = y(t)$ (iteration and poll step unsuccessful).*
  - *Set $\alpha(t+1) = \theta(t)\alpha(t)$ (contract the mesh size parameter).*

(5) If $\alpha(t+1) < \alpha_{tol}$ then stop. Otherwise, increment t by one and go to Step 3.

An example of the use of the search step is given in the next section. The poll step can be implemented in a number of different ways. The polling can be opportunistic (when it quits once the first decrease in the objective function is found) or complete (when the objective function is evaluated at all the points of the poll set). The order in which the points in P_t are evaluated can also differ [4, 8].

4. The particle swarm pattern search method

Pattern search methods are local methods in the sense that they are designed to achieve convergence (from arbitrary starting points) to points that satisfy necessary conditions for local optimality. These methods are known to identify well global minimizers for certain classes of problems (see, for instance, [1, 29]), especially when appropriate heuristic procedures are incorporated in the search step. In fact, the flexibility of the search step of a pattern search method can be used to improve the ability of the method to jump over local minimizers. On the other hand, the poll step can rigorously guarantee convergence to stationary points.

The hybrid method introduced in this paper 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 pattern search mesh M_t. This task can be done in several ways and, in particular, one can compute their 'projection' onto M_t

$$proj_{M_t}(x^i(t)) \;=\; \min_{u \in M_t} \|u - x^i(t)\|,$$

for $i = 1, \ldots, s$, or an approximation thereof.

There is no need then to project onto Ω since the use of the penalty function $\hat{f}$ in pattern search takes care of the bound constraints.

The stopping criterion of the particle swarm pattern search method is the conjunction of the stopping criteria for particle swarm and pattern search. The particle swarm pattern search method is described in Algorithm 4.1.

Algorithm 4.1.

(1) *Choose a positive integer τ and the stopping tolerance $\alpha_{tol} > 0$. Choose the positive spanning set $D = D_\oplus$.*

Choose a population size s and a stopping tolerance $v_{tol} > 0$. Randomly initialize the initial swarm $\{x^1(0), \ldots, x^s(0)\}$ and the initial swarm velocities $v^1(0), \ldots, v^s(0)$.

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

(3) ***[Search Step]***

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

For $i = 1, \ldots, s$ do (for every particle i):

- *Compute $\hat{x}^i(t) = proj_{M_t}(x^i(t))$.*
- *If $\hat{f}(\hat{x}^i(t)) < \hat{f}(y^i(t))$ then*
  - *Set $y^i(t+1) = \hat{x}^i(t)$ (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 $y^i(t+1) = y^i(t)$.*

(4) ***[Poll Step]***

Skip the poll step if the search step was successful.

- *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).*

(5) *Compute $v^i(t+1)$ and $x^i(t+1)$, $i = 1, \ldots, s$, using formulae (1) and (2).*

(6) *If $\alpha(t+1) < \alpha_{tol}$ and $\|v^i(t+1)\| < v_{tol}$, for all $i = 1, \ldots, s$, then stop. Otherwise, increment t by one and go to Step 3.*

5. Convergence

To analyze the sequence of iterates generated by Algorithm 4.1 we need to let it run for an infinite number of iterations. For this purpose, we consider

$\alpha_{tol} = 0$ and $v_{tol} = 0$, so that the algorithm never meets the termination criterion. Let $\{\hat{y}(t)\}$ be the sequence of iterates produced by Algorithm 4.1. Since all necessary pattern search ingredients are present, this method generates, under the appropriate assumptions, a sequence of iterates converging (independently of the starting point) to first-order critical points. A standard result for this class of methods tells us that there is a subsequence of unsuccessful iterations converging to a limit point and for which the mesh size parameter tends to zero [3, 23].

Theorem 5.1. *Let $L(\hat{y}(0)) = \{z \in \mathbb{R}^n : f(z) \leq f(\hat{y}(0))\}$ be a compact set. Then, there exists a subsequence $\{\hat{y}(t_k)\}$ of the iterates produced by Algorithm 4.1 (with $\alpha_{tol} = v_{tol} = 0$) such that*

$$\lim_{k \longrightarrow +\infty} \hat{y}(t_k) \;=\; \hat{y}_* \quad \textit{and} \quad \lim_{k \longrightarrow +\infty} \alpha(t_k) \;=\; 0,$$

for some $\hat{y}_* \in \Omega$. The subsequence $\{t_k\}$ consists of unsuccessful iterations.

The integrality assumptions imposed in the construction of the meshes M_t and on the update of the mesh size parameter are fundamental for the integer lattice type arguments required to prove this result. There are other ways to obtain such a result that circumvent the need for these integrality assumptions [23], for instance the imposition of a sufficient decrease condition on the step acceptance mechanism of the poll step.

Depending on the differentiability properties of the objective function, different type of stationarity can be proved for the point $\hat{y}_*$. For instance, if the function is continuously differentiable at this point, one can prove from the positive spanning properties of D that $\nabla f(\hat{y}_*) = 0$.

Theorem 5.1 tells us that a stopping criterion solely based on the size of the mesh size parameter (of the form $\alpha(t) < \alpha_{tol}$) will guarantee termination of the algorithm in a finite number of iterations. However, the stopping criterion of Algorithm 4.1 also requires $\|v^i(t)\| < v_{tol}$, $i = 1, \ldots, s$. Thus, it must be investigated whether or not the velocities in the particle swarm scheme satisfy a limit of the form

$$\lim_{k \longrightarrow +\infty} \|v^i(t_k)\| \;=\; 0 \quad i = 1, \ldots, s.$$

To do this we have to investigate the asymptotic behavior of the search step which is where the particle swarm strategy is applied. Rigorously speaking such a limit can only occur with probability one. To carry on the analysis we need to assume that $x^i(t)$, $y^i(t)$, $v^i(t)$, and $\hat{y}(t)$ are random variables of stochastic processes.

Theorem 5.2. *Suppose that for t sufficiently large one has that $\iota(t)$, $E(y^i(t))$, $i = 1, \ldots, s$, and $E(\hat{y}(t))$ are constant and that $E(proj_{M_t}(x^i(t-1)+v^i(t))) = E(x^i(t-1)+v^i(t))$, $i = 1, \ldots, s$. Then, for appropriate choices of the control parameters of the particle swarm,*

$$\lim_{t \longrightarrow +\infty} E(v^i_j(t)) \; = \; 0, \quad i = 1, \ldots, s, \; j = 1, \ldots, n.$$

and Algorithm 4.1 will stop (with 'probability one') in a finite number of iterations.

Proof: Given that there exists a subsequence driving the mesh size parameter to zero, it remains to investigate under what conditions do the velocities tend to zero.

Consider the velocity equation (2), repeated here for convenience, with the indices i for the particles and j for the vector components dropped for simplicity. To shorten notation we write the indices t as subscripts. Since $\omega_1(t)$ and $\omega_2(t)$ depend only on t, we get from (2) that

$$E(v_{t+1}) = \iota E(v_t) + \mu\omega_1 \left(E(y_t) - E(x_t)\right) + \nu\omega_2 \left(E(\hat{y}_t) - E(x_t)\right). \tag{4}$$

where $\omega_1 = E(\omega_1(t))$ and $\omega_2 = E(\omega_2(t))$. From equation (2), with $v(t)$ instead of $v(t+1)$, we also obtain that

$$\begin{aligned} E(v_t) \;&=\; \iota E(v_{t-1}) + \mu\omega_1 \left(E(y_{t-1}) - E(x_{t-1})\right) \\ &\quad + \nu\omega_2 \left(E(\hat{y}_{t-1}) - E(x_{t-1})\right). \end{aligned} \tag{5}$$

Subtracting (5) to (4) yields

$$\begin{aligned} E(v_{t+1}) - E(v_t) \;&=\; \iota(E(v_t) - E(v_{t-1})) - (\mu\omega_1 + \nu\omega_2)(E(x_t) - E(x_{t-1})) \\ &\quad + \mu\omega_1(E(y_t) - E(y_{t-1})) + \nu\omega_2(E(\hat{y}_t) - E(\hat{y}_{t-1})). \end{aligned}$$

Noting that $x_t = proj_{M_t}(x_{t-1} + v_t)$ we obtain the following inhomogeneous recurrence relation

$$E(v_{t+1}) - (1 + \iota - \mu\omega_1 - \nu\omega_2)E(v_t) + \iota E(v_{t-1}) \; = \; g_t, \tag{6}$$

where

$$\begin{aligned} g_t \;&=\; \mu\omega_1(E(y_t) - E(y_{t-1})) + \nu\omega_2(E(\hat{y}_t) - E(\hat{y}_{t-1})) \\ &\quad + (\mu\omega_1 + \nu\omega_2)E(proj_{M_t}(x_{t-1} + v_t) - (x_{t-1} + v_t)). \end{aligned}$$

From the assumptions of the theorem, we have, for sufficiently large t, that g_t is zero and therefore that the recurrence relation is homogeneous, with characteristic polynomial given by

$$t^2 - (1 + \iota - \mu\omega_1 - \nu\omega_2)t + \iota \;=\; 0. \tag{7}$$

Its solutions is then of the form

$$E(v_{t+1}) \;=\; c_1 a^t + c_2 b^t,$$

where c_1 and c_2 are constants and a and b are the two roots of the characteristic polynomial (7):

$$a \;=\; \frac{(1 + \iota - \mu\omega_1 - \nu\omega_2) + \sqrt{(1 + \iota - \mu\omega_1 - \nu\omega_2)^2 - 4\iota}}{2}$$

$$b \;=\; \frac{(1 + \iota - \mu\omega_1 - \nu\omega_2) - \sqrt{(1 + \iota - \mu\omega_1 - \nu\omega_2)^2 - 4\iota}}{2}$$

Thus, as long as $\max\{|a|, |b|\} < 1$ (which is achievable for certain choices of the control parameters ι, ω_1, ω_2, μ, and ν), we will get $E(v_{t+1}) \to 0$ when $t \to \infty$. ■

For instance, when $\omega_1 = \omega_2 = 0.5$ and $\mu = \nu = 0.5$, we obtain

$$a \;=\; \frac{(\iota + 0.5) + \sqrt{(\iota + 0.5)^2 - 4\iota}}{2}$$

and

$$b \;=\; \frac{(\iota + 0.5) - \sqrt{(\iota + 0.5)^2 - 4\iota}}{2}.$$

In this case, one has $\max\{|a|, |b|\} < 1$ for any $0 < \iota < 1$. One can clearly observe this fact in Figure 1. Zooming into the picture reveals that for $\iota < 0.0858$ we have $(\iota + 0.5)^2 - 4\iota \geq 0$, resulting in two real roots for the characteristic polynomial. For $\iota \geq 0.0858$ we have two complex conjugate roots whose modulus are equal.

It is difficult to show that the conditions of Theorem 5.2 can be rigorously satisfied but it can be given some indication of their practical reasonability.

Given that the function is bounded below in $L(y(0))$ it is known that the monotonically decreasing sequences $\{f(y^i(t))\}$, $i = 1, \ldots, s$, and $\{f(\hat{y}(t))\}$ converge. Thus, it is reasonable to suppose that the expected values of $y^i(t)$, $i = 1, \ldots, s$, and $\hat{y}(t)$ converge too.

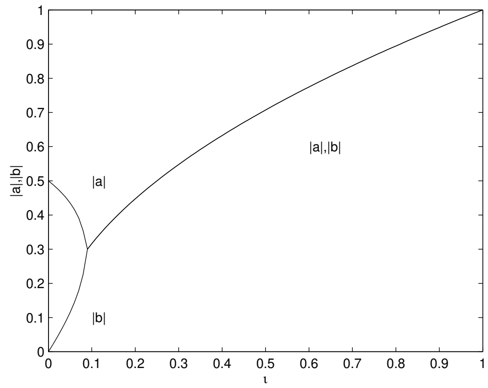

FIGURE 1. Plot of ι for $\mu\omega_1 = \nu\omega_2 = 0.25$.

On the other hand, the difference between $proj_{M_t}(x^i(t-1) + v^i(t))$ and $x^i(t-1) + v^i(t)$, — and thus between their expected values — is a multiple of $\alpha(t)$ for some type of meshes. This situation occurs in coordinate search, where $D = D_\oplus$. Since there is a subsequence of the mesh size parameters that converge to zero, there is at least the guarantee that the expected difference between $x^i(t-1) + v^i(t)$ and its projection onto M_t converges to zero along that subsequence.

So, there is at least some indication that the term g_t in (6) converges to zero for a subsequence of the iterates. Although this is not the same as saying that the assumptions of Theorem 5.2 are satisfied, it explains however the observed numerical termination of the algorithm.

6. Optimization solvers used for comparison

The particle swarm pattern search algorithm (Algorithm 4.1) was implemented in the C programming language. The solver is referred to as `PSwarm`.

In order to assess the performance of `PSwarm`, a set of 122 test problems from the literature ([2, 15, 18, 22, 25, 26, 30, 32]) was collected and coded in AMPL. All coded problems have simple bounds on the variables. The problems are available in `http://www.norg.uminho.pt/aivaz` (under software).

AMPL [13] is a mathematical modeling language which allows an easy and fast way to code optimization problems. AMPL provides also automatic differentiation (not used in the context of derivative free optimization) and interfaces to a number of optimization solvers. The AMPL mechanism to tune solvers was used to pass options to `PSwarm`.

6.1. Solvers. The optimization solver `PSwarm` was compared to a few solvers for global optimization, namely `Direct` [12], `ASA` [17], `MCS` [16], and `PGAPack` [24].

`Direct` is an implementation of the method described in [20]. `Direct` reads from *DIviding RECTangles* and it is implemented in MATLAB [28]. `Direct` solved the problems coded in AMPL by using the `amplfunc` external AMPL function for MATLAB together with a developed M-file to interface `Direct` with AMPL.

`ASA` is an implementation in C of the *Adaptative Simulated Annealing*. The user has to write its objective function as a C function and to compile it with the optimization code. Options are defined during compilation time. To use the AMPL coded problems, an interface for AMPL was also developed. Here, we have followed the ideas of the MATLAB interface to ASA provided by S. Sakata (see `http://www.econ.lsa.umich.edu/∼sakata/software`).

`MCS` stands for *Multilevel Coordinate Search* and it is inspired by the methods of Jones *et al.* [20]. `MCS` is implemented in MATLAB and, as in `Direct`, the AMPL interface to MATLAB and a developed M-file were used to obtain the numerical results for the AMPL coded problems.

`PGAPack` is an implementation of a genetic algorithm. The *Parallel Genetic Algorithm Pack* is written in C. As in `ASA`, the user defines a C function and compiles it along with the optimization code. As for `ASA`, an interface to AMPL was also developed. The population size selected for `PGAPAck` was changed to 200 (since it performed better with a higher population size).

`Direct` and `MCS` are deterministic codes. The other two, `PGAPack` and `ASA`, together with `PSwarm`, are stochastic ones. A relevant issue in stochastic codes is the choice of the underlying random numbers generator. It is well known that good random numbers generator are hard to find (see, for example, [31]). `ASA` and `Pswarm` use the number generator from [7, 31]. `PGAPack` uses a generator described in [19].

6.2. `PSwarm` details. The default values for `PSwarm` are $\alpha_{tol} = 10^{-5}$, $\nu = \mu = 0.5$, $\phi(t) = 2$, $\theta(t) = 0.5$, $\alpha(0) = \max_{j=1,..,n}(u_j - \ell_j)/c$ with $c = 5$, and $s = 20$. The projection onto the mesh ($\hat{x}^i(t) = proj_{M_t}(x^i(t))$) has not been implemented in the search step.

The inertial parameter ι 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. Bigger values for $\max\{|a|, |b|\}$ will result in slower convergence.

Using a linear interpolation for ι, we start with a slower rate and terminate faster.

The initial swarm is obtained by generating s random points following an uniform distribution $U(\ell, u)$, yielding $x_j^i(0) \sim U(\ell_j, u_j)$, $j = 1, \ldots, n$, for all particles $i = 1, \ldots, s$, and therefore providing a feasible initial swarm.

In particle swarm all particles will in principle converge to $\hat{y}$ and a high concentration of particles is needed in order to obtain a solution with some degree of precision. Thus, in the last iterations of particle swarm, a massive number of objective function evaluations is necessary to obtain some precision in the solution. Removing particles from the swarm that are nearby others seems a good idea, but a price in precision is paid in order to gain a decrease in the number of objective function evaluations.

In the proposed particle swarm pattern search algorithm the scenario is somehow different since the $\hat{y}$ particle position is improved by the poll steps of pattern search. Removing particles that do not drive the search to a global minimizer is highly desirable. A particle i is removed from the swarm in `PSwarm` when it is close to $\hat{y}$, *i.e.*, when $\|y^i(t) - \hat{y}(t)\| \leq \alpha(0)$. If a particle is close to $\hat{y}$ (compared in terms of $\alpha(0)$) it means that it is unlikely to further explore the search space for a global optimum.

6.3. Performance profiles. A fair comparison among different solvers should be based on the number of function evaluations, instead of based on the number of iterations or on the CPU time. The number of iterations is not a reliable measure because the amount of work done in each iteration is completely different among solvers, since some are population based and other are single point based. Also, while all the problems are coded in AMPL, the solvers are implemented in different programming languages and their accesses to the problems use different scripting languages, leading to significant differences in CPU times.

Besides the number of function evaluations, the objective function value at the solutions determined should also be taken into account, as one of the measures of quality of a global optimization solver. The approach taken in this paper consisted of imposing a maximum number of function evaluations and of comparing the objective function value at the solutions computed by the different solvers. A lower objective function computed means that the solver performed better.

The ASA solver does not support an option that limits the number of objective function evaluations. The interface developed for AMPL accounts for the number of objective function calls, and when the limit is reached exit is immediately forced by properly setting an ASA option. Solvers PSwarm, Direct, MCS, and PGaPack take control of the number of objective function evaluations in each iteration, and therefore the maximum number of objective function evaluations can be directly imposed. Algorithms that perform more than one objective function evaluation per iteration can exceed the requested maximum since the stopping criterion is checked at the beginning of each iteration. For instance, MCS for problem lj1_38 (a Lennard-Jones cluster problem of size 38) computes 14296 times the objective function value, when a maximum of 1000 function evaluations is requested. Tuning other solvers options could reduce this gap, but we decide to use, as much as possible, the solver default options (the exceptions were the maximum numbers of function evaluations and iterations and the population size in PGAPack).

We chose to present the numerical results in the form of performance profiles, as described in [10]. This procedure was developed to benchmark optimization software, *i.e.*, to compare different solvers on several (possibly many) test problems. One advantage of the performance profiles is the fact that they can be presented in one figure, by plotting for the different solvers a cumulative distribution function $\rho(\tau)$ representing a performance ratio.

The performance ratio is defined as $r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s}:s\in\mathcal{S}\}}$, where $\mathcal{P}$ is the test set, $p \in \mathcal{P}$, $\mathcal{S}$ is the set of solvers, $s \in \mathcal{S}$, and $t_{p,s}$ is the value obtained by solver s on test problem p. Define $\rho_s(\tau) = \frac{1}{n_p}\mathrm{size}\{p \in \mathcal{P} : r_{p,s} \leq \tau\}$, where n_p is the number of test problems.

The value of $\rho_s(1)$ is the probability that the solver will win over the remaining ones. If we are only interested in determining which solver is the best (in the sense that wins the most), we compare the values of $\rho_s(1)$ for all the solvers. At the other end, solvers with the largest probabilities $\rho_s(\tau)$ for large values of τ are the most robust ones.

The performance profile measure described in [10] was the computing time required to solve the problem, but other performance quantities can be used, namely the number of function evaluations. However, the objective function value achieved at the maximum number of function evaluations imposed cannot be used directly as a performance profile measure. For instance, a problem in the test set whose objective function value at the solution computed by one of the solvers is zero could lead to $\min\{t_{p,s} : s \in \mathcal{S}\} = 0$. If the objective function value at the solution determined by a solver is negative, then the value of $r_{p,s}$ could also be negative. In any of the situations, it is not possible to use the performance profiles.

Also, several runs for the same stochastic solver must be made for every problem, so that average, best, and worst behavior can be analyzed. In [2], the following scaled performance profile measure was introduced

$$t_{p,s} = \frac{\bar{f}_{p,s} - f_p^*}{f_{p,w} - f_p^*}, \tag{8}$$

where $\bar{f}_{p,s}$ is the average function value for the several runs of solver s on problem p, f_p^* is the best function value found among all the solvers (or the global minimum when known), and $f_{p,w}$ is the worst function value found among all the solvers. If we were interested in the best (resp. worst) behavior one would use, instead of $\bar{f}_{p,s}$, the best (resp. the worst) value among all runs of the stochastic solver s on problem p.

While using (8) could prevent $r_{p,s}$ from taking negative values, a division by zero can occur when $f_{p,w}$ is equal to f_p^*. To avoid division by zero, we suggest a shift to the positive axis for problems where a negative or zero $\min\{t_{p,s} : s \in \mathcal{S}\}$ is obtained. Our performance profile measure is defined as:

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

$$r_{p,s} = \begin{cases} 1 + t_{p,s} - \min\{t_{p,s} : s \in \mathcal{S}\} & \text{when } \min\{t_{p,s} : s \in \mathcal{S}\} < 0.001, \\ \frac{t_{p,s}}{\min\{t_{p,s} : s \in \mathcal{S}\}} & \text{otherwise.} \end{cases}$$

7. Numerical results

The stochastic solvers `PSwarm`, `ASA` and `PGAPack` were run 30 times. The performance profiles are plotted for the best, average, and worst objective value found. The deterministic solvers `Direct` and `MCS` were run only once. The tests were run in a Pentium IV (3.0GHz and 1Gb of RAM).

Figures 2, 3, and 4 present the plots for the best, average, and worst solutions found, respectively, when the maximum number of function evaluations was set to 1000 for problems with dimension lower than 100 and 7500 for the 13 remaining ones ($maxf = 1000(7500)$). Figures 5, 6, and 7 correspond to the case when $maxf = 10000(15000)$. Each figure includes two plots. One for better visibility around $\rho(1)$ and another to capture the tendency near $\rho(\infty)$.

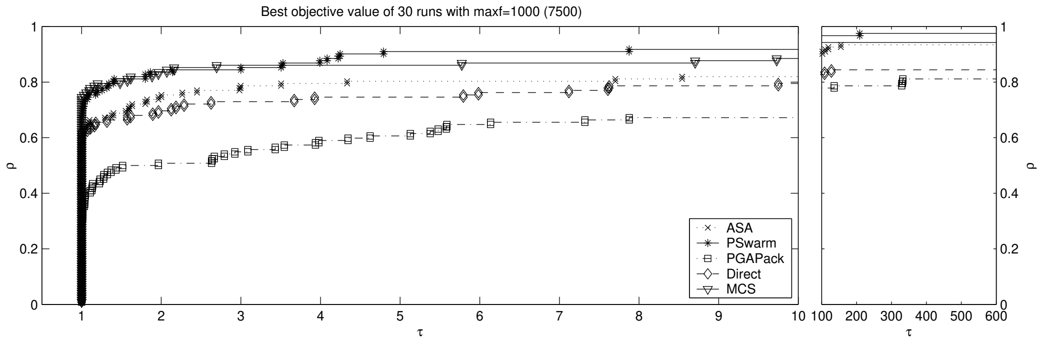

FIGURE 2. Best objective function value for 30 runs with $maxf = 1000(7500)$.

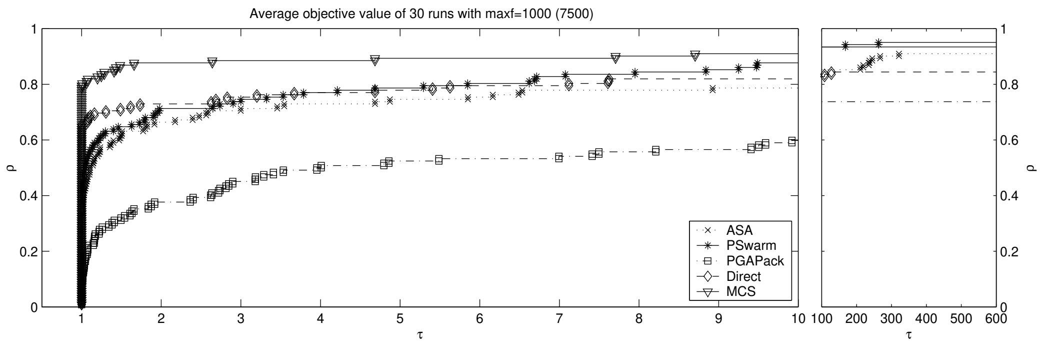

FIGURE 3. Average objective function value for 30 runs with $maxf = 1000(7500)$.

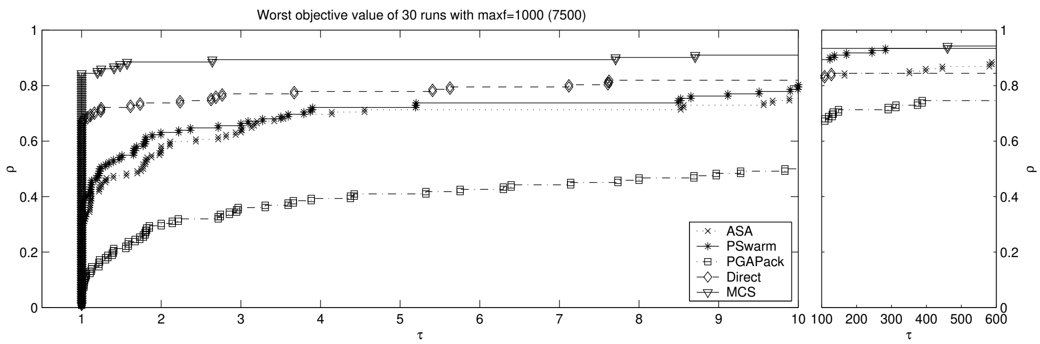

FIGURE 4. Worst objective function value for 30 runs with $maxf = 1000(7500)$.

From Figure 2, we can conclude that `PSwarm` has a slightly advantage over the other solvers in the best behavior case for $maxf = 1000(7500)$. In the average and worst behaviors, `PSwarm` loses in performance against `Direct` and `MCS`. In any case, it wins to all the other solvers in robustness.

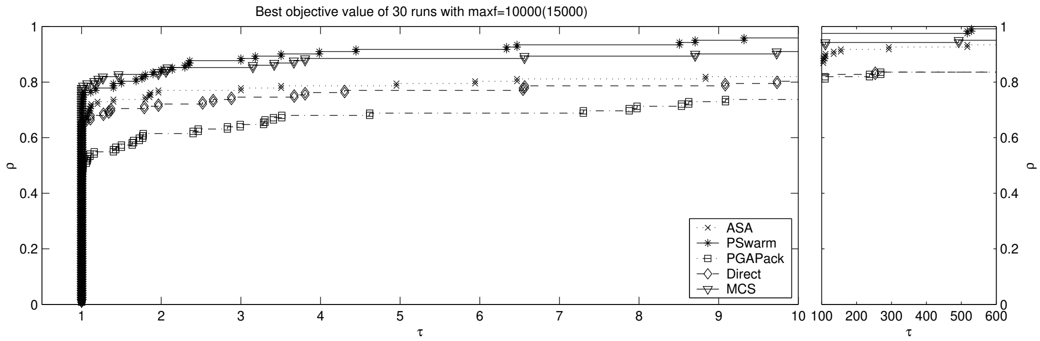

FIGURE 5. Best objective function value for 30 runs with $maxf = 10000(15000)$.

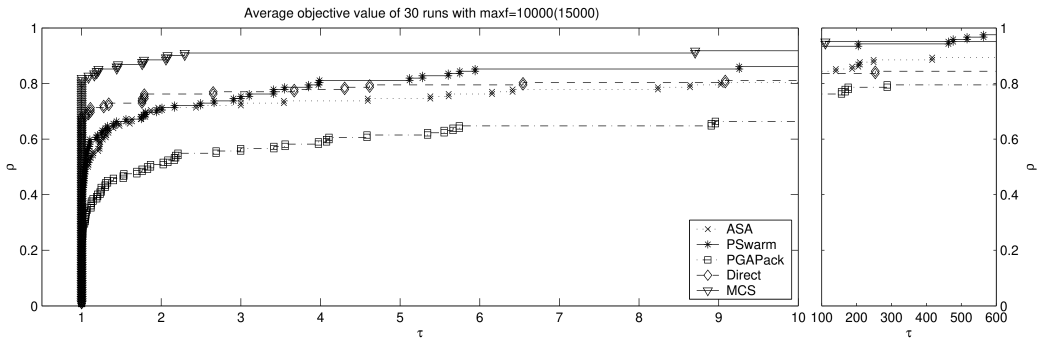

FIGURE 6. Average objective function value for 30 runs with $maxf = 10000(15000)$.

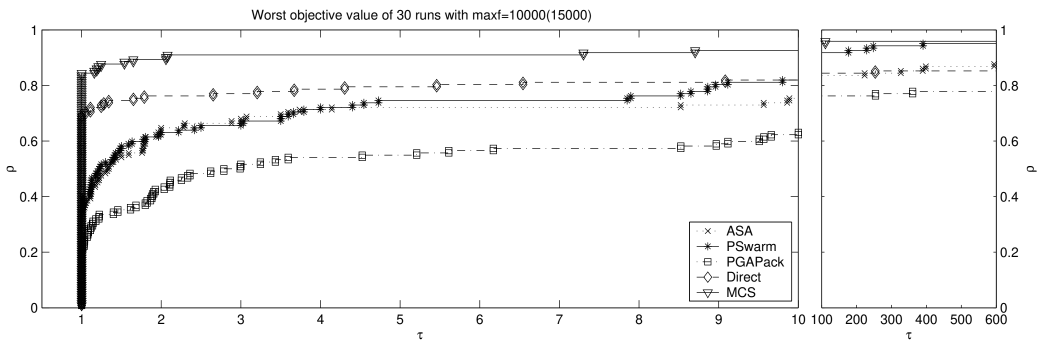

FIGURE 7. Worst objective function value for 30 runs with $maxf = 10000(15000)$.

When $maxf = 10000(15000)$ and for the best behavior, `PSwarm` together with `MCS` win over the remaining solvers, being the former one slightly more robust. In the average and worst scenarios, `PSwarm` loses against `Direct` and `MCS`, but again it wins on robustness overall.

In Figures 8 and 9 we plot the profiles for the number of function evaluations taken to solve the problems in our list (for the cases $maxf = 1000$ and $maxf = 10000$). The best solver is MCS, which is not surprising since it is based on interpolation models and most of the objective functions tested are smooth. PSwarm appears clearly in the second place in these profiles. Moreover, in Table 1 we report the average number of function evaluations taken to solve the problems in our list (also for the cases $maxf = 1000$ and $maxf = 10000$). One can see from these tables that PSwarm appears first and MCS performed apparently worse. This effect is due to some of the problems in our test set where the objective function exhibits steep oscillations. PSwarm is a direct search type method and thus better suited to deal with such type of functions, and thus it seemed to present the best balance (among all solvers) for smooth and less smooth types of objective functions.

It is important to point out that the performance of Direct is not necessarily better than the one of PSwarm as we could have inferred from the profiles for the quality of the final objective value. In fact, the stopping criterion for Direct (as well as for PGAPack) is based on the maximum number of function evaluations permitted. One can clearly see from Table 1 that Direct took many more evaluations than PSwarm (and that the latter one also performed significantly better than ASA).

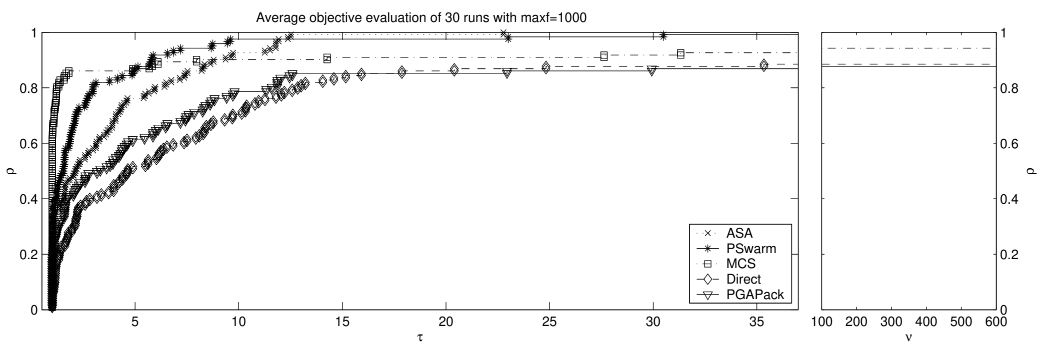

FIGURE 8. Number of objective function evaluations in the case $maxf = 1000$ (averages among the 30 runs for stochastic solvers).

Table 2 reports detailed numerical results obtained by the solver PSwarm for all problems in our test set. The maximum number of function evaluations was set to 10000. For each problem, we chose to report the best result (in terms of f) obtained among the 30 runs. The columns in the table refer to: 'problem' the problem name (AMPL model file); n the problem dimension (number of variables); 'nfevals' the number of objective function

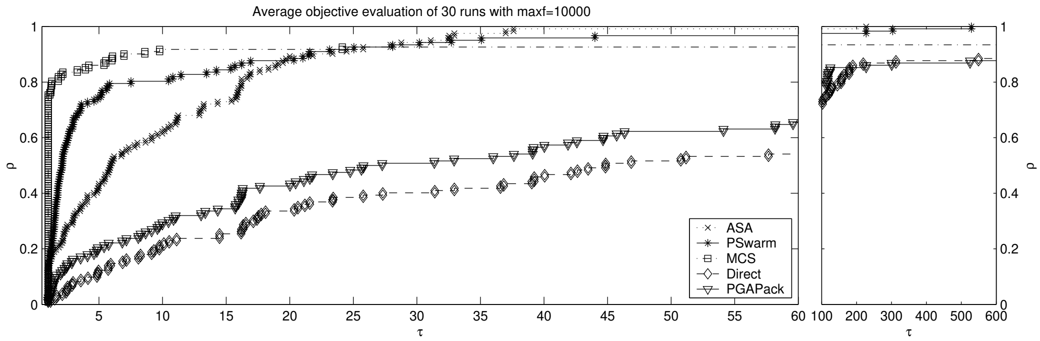

FIGURE 9. Number of objective function evaluations in the case $maxf = 10000$ (averages among the 30 runs for stochastic solvers).

$maxf$	ASA	PGAPack	PSwarm	Direct	MCS
1000	857	1009	686	1107	1837
10000	5047	10009	3603	11517	4469

TABLE 1. Average number of function evaluations for the test set in the cases $maxf = 1000$ and $maxf = 10000$ (averages among the 30 runs for stochastic solvers).

evaluations; 'niter' the number of iterations; 'npoll' the number of poll steps; '%spoll' the percentage of successful poll steps; 'gap' the optimal gap (when known, otherwise the value marked with $*$ is just the final objective function calculated). We did not report the final number of particles because this number is equal to one in the majority of the problems ran.

problem	n	nfevals	niter	npoll	%spoll	gap
ack	10	1797	121	117	81.2	2.171640E-01
ap	2	207	34	32	40.63	-8.600000E-05
bf1	2	204	36	33	33.33	0.000000E+00
bf2	2	208	37	35	37.14	0.000000E+00
bhs	2	218	29	28	39.29	-1.384940E-01
bl	2	217	36	34	41.18	0.000000E+00
bp	2	224	39	37	45.95	-3.577297E-07
cb3	2	190	29	27	29.63	0.000000E+00
cb6	2	211	37	35	48.57	-2.800000E-05
cm2	2	182	34	31	45.16	0.000000E+00
cm4	4	385	45	41	60.98	0.000000E+00
da	2	232	45	41	48.78	4.816600E-01

continues...

. . . continued

problem	n	nfevals	niter	npoll	%spoll	gap
em_10	10	4488	324	321	89.41	1.384700E+00
em_5	5	823	99	94	79.79	1.917650E-01
ep.mod	2	227	39	35	45.71	0.000000E+00
exp.mod	10	1434	84	80	80	0.000000E+00
fls.mod	2	227	28	22	27.27	3.000000E-06
fr.mod	2	337	71	67	52.24	0.000000E+00
fx_10	10	1773	125	108	78.7	8.077291E+00
fx_5	5	799	123	57	68.42	6.875980E+00
gp	2	190	28	26	30.77	0.000000E+00
grp	3	1339	263	28	28.57	0.000000E+00
gw	10	2296	152	146	82.19	0.000000E+00
h3	3	295	37	35	57.14	0.000000E+00
h6	6	655	59	51	68.63	0.000000E+00
hm	2	195	32	30	36.67	0.000000E+00
hm1	1	96	22	20	15	0.000000E+00
hm2	1	141	29	27	25.93	-1.447000E-02
hm3	1	110	22	21	19.05	2.456000E-03
hm4	2	198	31	28	35.71	0.000000E+00
hm5	3	255	34	30	50	0.000000E+00
hsk	2	204	28	26	34.62	-1.200000E-05
hv	3	343	44	42	54.76	0.000000E+00
ir0	4	671	84	80	66.25	0.000000E+00
ir1	3	292	41	37	51.35	0.000000E+00
ir2	2	522	131	119	61.34	1.000000E-06
ir3	5	342	25	20	10	0.000000E+00
ir4	30	8769	250	244	93.03	1.587200E-02
ir5	2	513	116	40	45	1.996000E-03
kl	4	1435	170	164	75.61	-4.800000E-07
ks	1	92	18	17	0	0.000000E+00
lj1_38	114	10072	146	127	95.28	1.409238E+02*
lj1_75	225	10063	137	127	96.85	3.512964E+04*
lj1_98	294	10072	129	119	98.32	1.939568E+05*
lj2_38	114	10109	153	139	95.68	3.727664E+02*
lj2_75	225	10090	116	90	98.89	3.245009E+04*
lj2_98	294	10036	125	114	98.25	1.700452E+05*
lj3_38	114	10033	157	127	93.7	1.729289E+03*
lj3_75	225	10257	124	112	98.21	1.036894E+06*
lj3_98	294	10050	113	107	99.07	1.518801E+07*
lm1	3	335	44	40	52.5	0.000000E+00
lm2_10	10	1562	93	86	77.91	0.000000E+00
lm2_5	5	625	59	56	67.86	0.000000E+00
lms1a	2	1600	172	123	55.28	-2.000000E-06

continues. . .

. . . continued

problem	n	nfevals	niter	npoll	%spoll	gap
lms1b	2	2387	452	55	36.36	1.078700E-02
lms2	3	1147	163	60	48.33	1.501300E-02
lms3	4	2455	262	109	53.21	6.233700E-02
lms5	6	5596	1631	366	59.84	7.384100E-02
lv8	3	310	42	39	48.72	0.000000E+00
mc	2	211	32	29	41.38	7.700000E-05
mcp	4	248	29	22	27.27	0.000000E+00
mgp	2	193	33	31	41.94	-2.593904E+00
mgw_10	10	10007	473	461	93.71	1.107800E-02
mgw_2	2	339	43	37	43.24	0.000000E+00
mgw_20	20	10005	306	299	93.98	5.390400E-02
ml_10	10	2113	129	118	75.42	0.000000E+00
ml_5	5	603	59	55	67.27	0.000000E+00
mr	3	886	179	171	62.57	1.860000E-03
mrp	2	217	44	43	55.81	0.000000E+00
ms1	20	3512	216	207	90.82	4.326540E-01
ms2	20	3927	238	225	91.56	-1.361000E-02
nf2	4	2162	205	198	64.65	2.700000E-05
nf3_10	10	4466	586	579	95.16	0.000000E+00
nf3_15	15	10008	800	792	96.46	7.000000E-06
nf3_20	20	10008	793	768	94.92	2.131690E-01
nf3_25	25	10025	535	508	95.67	5.490210E-01
nf3_30	30	10005	359	347	96.25	6.108021E+01
osp_10	10	1885	134	121	80.17	1.143724E+00
osp_20	20	5621	229	220	90.45	1.143833E+00
plj_38	114	10103	163	135	96.3	7.746385E+02*
plj_75	225	10028	127	109	98.17	3.728411E+04*
plj_98	294	10182	119	105	98.1	1.796150E+05*
pp	10	1578	104	100	81	-4.700000E-04
prd	2	400	66	34	44.12	0.000000E+00
ptm	9	10009	1186	618	73.46	3.908401E+00
pwq	4	439	57	53	60.38	0.000000E+00
rb	10	10003	793	712	76.12	1.114400E-02
rg_10	10	4364	672	158	71.52	0.000000E+00
rg_2	2	210	34	32	43.75	0.000000E+00
s10	4	431	51	48	62.5	-4.510000E-03
s5	4	395	46	43	58.14	-3.300000E-03
s7	4	415	52	49	63.27	-3.041000E-03
sal_10	10	1356	76	68	60.29	3.998730E-01
sal_5	5	452	39	37	40.54	1.998730E-01
sbt	2	305	39	37	45.95	-9.000000E-06
sf1	2	210	32	29	24.14	9.716000E-03

continues. . .

. . . continued

problem	n	nfevals	niter	npoll	%spoll	gap
sf2	2	266	45	41	43.9	5.383000E-03
shv1	1	101	20	19	21.05	-1.000000E-03
shv2	2	196	33	31	41.94	0.000000E+00
sin_10	10	1872	124	117	81.2	0.000000E+00
sin_20	20	5462	225	216	88.43	0.000000E+00
st_17	17	10011	1048	457	78.12	3.081935E+06
st_9	9	10001	1052	847	82.88	7.516622E+00
stg	1	113	26	23	17.39	0.000000E+00
swf	10	2311	161	158	82.91	1.184385E+02
sz	1	125	34	28	25	-2.561249E+00
szzs	1	112	29	27	33.33	-1.308000E-03
wf	4	10008	3505	1150	59.57	2.500000E-05
xor	9	887	73	60	68.33	8.678270E-01
zkv_10	10	10003	1405	752	75.8	1.393000E-03
zkv_2	2	212	39	35	45.71	0.000000E+00
zkv_20	20	10018	1031	422	77.01	3.632018E+01
zkv_5	5	1318	168	163	85.89	0.000000E+00
zlk1	1	119	27	25	20	4.039000E-03
zlk2a	1	130	26	22	22.73	-5.000000E-03
zlk2b	1	113	26	24	25	-5.000000E-03
zlk3a	1	138	32	29	24.14	0.000000E+00
zlk3b	1	132	32	29	24.14	0.000000E+00
zlk3c	1	132	27	25	24	0.000000E+00
zlk4	2	224	39	37	45.95	-2.112000E-03
zlk5	3	294	40	37	56.76	-2.782000E-03
zzs	1	120	29	26	23.08	-4.239000E-03

Table 2: Numerical results obtained by `PSwarm`.

8. Conclusions and future work

In this paper we developed a hybrid algorithm for global minimization with simple bounds that combines a heuristic for global optimization (particle swarm) with a rigorous method (pattern search) for local minimization. The particle swarm pattern search method proposed enjoys the global convergence properties of pattern search for stationary points (convergence regardless of the starting point).

We presented some analysis for the particle swarm pattern search method that indicates proper termination for an appropriate hybrid stopping criterion. The numerical results presented are particularly encouraging given that no fine tuning of algorithmic choices or parameters has been done yet for the new algorithm. A basic implementation of the particle swarm pattern search (`PSwarm` solver) has been shown to be the most robust among all global optimization solvers tested and to be highly competitive in efficiency with the most efficient of these solvers (`MCS`).

We plan to implement the particle swarm pattern search method in a parallel environment. Given that both techniques (particle swarm and pattern search) are easy to parallelize, so is the combined algorithm. In the search step of the method, where particle swarm is applied, one can distribute the evaluation of the objective function on the new swarm by the processors available. The same can be done in the poll step for the poll set. Another task for future research is to handle problems with more general type of constraints.

References

[1] P. Alberto, F. Nogueira, H. Rocha, and L. N. Vicente. Pattern search methods for user-provided points: Application to molecular geometry problems. *SIAM J. Optim.*, 14:1216–1236, 2004.

[2] M. M. Ali, C. Khompatraporn, and Z. B. Zabinsky. A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *J. Global Optim.*, 31:635–672, 2005.

[3] C. Audet and J. E. Dennis. Analysis of generalized pattern searches. *SIAM J. Optim.*, 13:889–903, 2003.

[4] C. Audet and J. E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM J. Optim.*, (to appear).

[5] C. Audet and D. Orban. Finding optimal algorithmic parameters using the mesh adaptive direct search algorithm. Technical Report Les Cahiers du GERAD, G-2004-96, 2004.

[6] F. van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, Faculty of Natural and Agricultural Science, University of Pretoria, November 2001.

[7] A. K. Binder and A. D. Stauffer. A simple introduction to Monte Carlo simulations and some specialized topics. In E. K. Binder, editor, *Applications of the Monte Carlo Method in Statistical Physics*, pages 1–36. Springer-Verlag, Berlin, 1985.

[8] A. L. Custódio and L. N. Vicente. Using sampling and simplex derivatives in pattern search methods. Technical Report 04-35, Departamento de Matemática, Universidade de Coimbra, Portugal, 2004.

[9] C. Davis. Theory of positive linear dependence. *Amer. J. Math.*, 76:733–746, 1954.

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

[11] R. Eberhart and J. Kennedy. New optimizers using particle swarm theory. In *Proceedings of the 1995 6th International Symposium on Micro Machine and Human Science*, pages 39–43.

[12] D. E. Finkel. DIRECT *Optimization Algorithm User Guide*. North Carolina State University, 2003. http://www4.ncsu.edu/∼definkel/research/index.html.

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

[14] W. E. Hart. Locally-adaptive and memetic evolutionary pattern search algorithms. *Evolutionary Computation*, 11:29–52, 2003.

[15] A.-R. Hedar and M. Fukushima. Heuristic pattern search and its hybridization with simulated annealing for nonlinear global optimization. *Optim. Methods Softw.*, 19:291–308, 2004.

[16] W. Huyer and A. Neumaier. Global optimization by multilevel coordinate search. *J. Global Optim.*, 14:331–355, 1999. http://solon.cma.univie.ac.at/∼neum/software/mcs.

[17] L. Ingber. Adaptative simulated annealing (ASA): Lessons learned. *Control and Cybernetics*, 25:33–54, 1996. http://www.ingber.com.

[18] L. Ingber and B. Rosen. Genetic algorithms and very fast simulated reannealing: A comparison. *Mathematical and Computer Modelling*, 16:87–100, 1992.

[19] F. James. A review of pseudorandom number generators. *Computer Physics Communication*, 60:329–344, 1990.

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

[21] 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.

[22] E. Kiseleva and T. Stepanchuk. On the efficiency of a global non-differentiable optimization algorithm based on the method of optimal set partitioning. *J. Global Optim.*, 25:209–235, 2003.

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

[24] D. Levine. Users guide to the PGAPack parallel genetic algorithm library. Technical Report ANL-95/18, Argonne National Laboratory, 1996. http://www.mcs.anl.gov/pgapack.html.

[25] M. Locatelli. A note on the Griewank test function. *J. Global Optim.*, 25:169–174, 2003.

[26] M. Locatelli and F. Schoen. Fast global optimization of difficult Lennard-Jones clusters. *Comput. Optim. and Appl.*, 21:55–70, 2002.

[27] A. L. Marsden. *Aerodynamic Noise Control by Optimal Shape Design*. PhD thesis, Stanford University, 2004.

[28] MathWorks. *MATLAB*. The MathWorks Inc., 1999. Version 5.4, Release 11.

[29] J. C. Meza and M. L. Martinez. On the use of direct search methods for the molecular conformation problem. *Journal of Computational Chemistry*, 15:627–632, 1994.

[30] M. Mongeau, H. Karsenty, V. Rouzé, and J.-B. Hiriart-Urruty. Comparison of public-domain software for black box global optimization. *Optim. Methods Softw.*, 13:203–226, 2000.

[31] S. K. Park and K. W. Miller. Random number generators: Good ones are hard to find. *Communications of the ACM*, 31:1192–1201, 1988.

[32] K. E. Parsopoulos, V. P. Plagianakos, G. D. Magoulas, and M. N. Vrahatis. Stretching technique for obtaining global minimizers through particle swarm optimization. In *Proc. Of the Particle Swarm Optimization Workshop*, pages 22–29, Indianapolis, USA, 2001.

A. Ismael F. Vaz
Departamento de Produção e Sistemas, Escola de Engenharia, Universidade do Minho, Campus de Gualtar, 4710-057 Braga, Portugal (aivaz@dps.uminho.pt).

L. N. Vicente
Departamento de Matemática, Universidade de Coimbra, 3001-454 Coimbra, Portugal (lnv@mat.uc.pt).