Direct search for linearly constrained global optimization using different search steps

A. Ismael F. Vaz

University of Minho - Portugal
aivaz@dps.uminho.pt

Joint work with Luis Nunes Vicente and Le Thi Hoai An

ICCOPT 2010

July 26-29, 2010

Outline

1. Direct search
2. Using particle swarm in the search step
3. Using radial basis functions in the search step
4. Conclusions

Outline

Outline

1. Direct search
2. Using particle swarm in the search step
3. Using radial basis functions in the search step
4. Conclusions

Outline

Outline

Linear-constrained derivative-free optimization

Problem formulation

$$\min_{x\in\Omega} \; f(x)$$

where

$$\Omega = \{x \in \mathbb{R}^n : \quad Ax \; \leq \; b, \quad \ell \; \leq \; x \; \leq u\},$$

$A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$.

We aim at solving this problem without using derivatives of f.

Some definitions

Positive spanning set

Is a set of vectors that spans $\mathbb{R}^n$ with nonnegative coefficients.

Examples

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

$$D_{\otimes} = \{e_1, \ldots, e_n, -e_1, \ldots, -e_n, e, -e\}$$

Extreme barrier function

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

Some definitions

Positive spanning set

Is a set of vectors that spans $\mathbb{R}^n$ with nonnegative coefficients.

Examples

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

$$D_\otimes = \{e_1, \ldots, e_n, -e_1, \ldots, -e_n, e, -e\}$$

Extreme barrier function

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

Some definitions

Positive spanning set

Is a set of vectors that spans $\mathbb{R}^n$ with nonnegative coefficients.

Examples

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

$$D_{\otimes} = \{e_1, \ldots, e_n, -e_1, \ldots, -e_n, e, -e\}$$

Extreme barrier function

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

A direct-search method

(0) Initialization
Choose $x_0 \in \Omega$, $\alpha_0 > 0$.

For $k = 0, 1, 2, \ldots$

Let D_k be a positive spanning set (set of positive generators when there are linear constraints).

(1) Search step (Optional)
Try to compute a point x in the grid $M_k = \left\{x_k + \alpha_k D_k z,\ z \in \mathbb{N}_0^{|D_k|}\right\}$ with

$$f_\Omega(x) < f(x_k).$$

If $f_\Omega(x) < f(x_k)$ then set $x_{k+1} = x$, declare the iteration and the search step successful, and skip the poll step.

A direct-search method

(0) Initialization
Choose $x_0 \in \Omega$, $\alpha_0 > 0$.

For $k = 0, 1, 2, \ldots$

Let D_k be a positive spanning set (set of positive generators when there are linear constraints).

(1) Search step (Optional)
Try to compute a point x in the grid $M_k = \left\{ x_k + \alpha_k D_k z,\ z \in \mathbb{N}_0^{|D_k|} \right\}$ with

$$f_\Omega(x) < f(x_k).$$

If $f_\Omega(x) < f(x_k)$ then set $x_{k+1} = x$, declare the iteration and the search step successful, and skip the poll step.

A direct-search method

(0) Initialization
Choose $x_0 \in \Omega$, $\alpha_0 > 0$.

For $k = 0, 1, 2, \ldots$

Let D_k be a positive spanning set (set of positive generators when there are linear constraints).

(1) Search step (Optional)
Try to compute a point x in the grid $M_k = \left\{ x_k + \alpha_k D_k z,\ z \in \mathbb{N}_0^{|D_k|} \right\}$ with

$$f_\Omega(x) < f(x_k).$$

If $f_\Omega(x) < f(x_k)$ then set $x_{k+1} = x$, declare the iteration and the search step successful, and skip the poll step.

A direct-search method

(0) Initialization
Choose $x_0 \in \Omega$, $\alpha_0 > 0$.

For $k = 0, 1, 2, \ldots$

Let D_k be a positive spanning set (set of positive generators when there are linear constraints).

(1) Search step (Optional)
Try to compute a point x in the grid $M_k = \left\{ x_k + \alpha_k D_k z,\ z \in \mathbb{N}_0^{|D_k|} \right\}$ with

$$f_\Omega(x) < f(x_k).$$

If $f_\Omega(x) < f(x_k)$ then set $x_{k+1} = x$, declare the iteration and the search step successful, and skip the poll step.

A direct-search method

(2) Poll step: Optionally order the poll set $P_k = \{x_k + \alpha_k d : \ d \in D_k\}$.

If a poll point $x_k + \alpha_k d_k$ is found such that $f_\Omega(x_k + \alpha_k d_k) < f(x_k)$ then stop polling, set $x_{k+1} = x_k + \alpha_k d_k$, and declare the iteration and the poll step successful.

Otherwise declare the iteration (and the poll step) unsuccessful and set $x_{k+1} = x_k$.

A direct-search method

(2) Poll step: Optionally order the poll set $P_k = \{x_k + \alpha_k d : \; d \in D_k\}$.

If a poll point $x_k + \alpha_k d_k$ is found such that $f_\Omega(x_k + \alpha_k d_k) < f(x_k)$ then stop polling, set $x_{k+1} = x_k + \alpha_k d_k$, and declare the iteration and the poll step successful.

Otherwise declare the iteration (and the poll step) unsuccessful and set $x_{k+1} = x_k$.

A direct-search method

(2) Poll step: Optionally order the poll set $P_k = \{x_k + \alpha_k d : \; d \in D_k\}$.

If a poll point $x_k + \alpha_k d_k$ is found such that $f_\Omega(x_k + \alpha_k d_k) < f(x_k)$ then stop polling, set $x_{k+1} = x_k + \alpha_k d_k$, and declare the iteration and the poll step successful.

Otherwise declare the iteration (and the poll step) unsuccessful and set $x_{k+1} = x_k$.

A direct-search method

(3) Step size update: If the iteration was successful then maintain the step size parameter ($\alpha_{k+1} = \alpha_k$) or double it ($\alpha_{k+1} = 2\alpha_k$) after two consecutive poll successes along the same direction.

If the iteration was unsuccessful, halve the step size parameter ($\alpha_{k+1} = \alpha_k/2$).

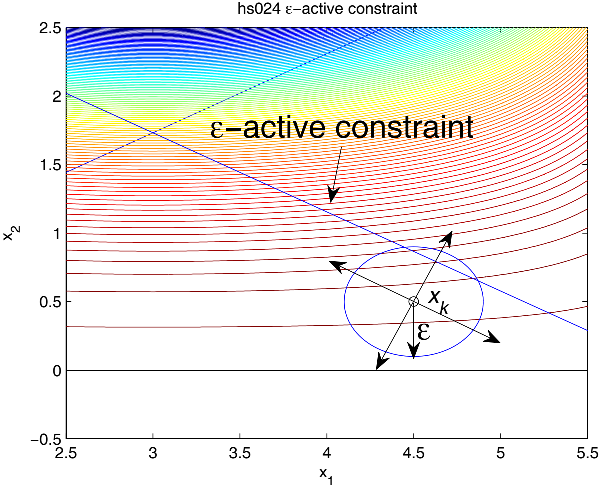
A direct-search method

(3) Step size update: If the iteration was successful then maintain the step size parameter ($\alpha_{k+1} = \alpha_k$) or double it ($\alpha_{k+1} = 2\alpha_k$) after two consecutive poll successes along the same direction.

If the iteration was unsuccessful, halve the step size parameter ($\alpha_{k+1} = \alpha_k/2$).

Poll step — linear constraints

The set of polling directions needs to conform with the geometry of the feasible set.



Positive generators for the tangent cone

No ϵ-active constraints

The set of polling directions D_k is the positive spanning set $D_\otimes$.

For ϵ-active constraint(s)

D_k is the set of positive generators for the tangent cone of the ϵ-active constraints (obtained by QR factorization).

Degeneracy

The ϵ parameter is dynamically adapted when degeneracy in the ϵ-active constraints is detected. If no success is attained $D_\otimes$ is used.

Positive generators for the tangent cone

No ϵ-active constraints

The set of polling directions D_k is the positive spanning set $D_\otimes$.

For ϵ-active constraint(s)

D_k is the set of positive generators for the tangent cone of the ϵ-active constraints (obtained by QR factorization).

Degeneracy

The ϵ parameter is dynamically adapted when degeneracy in the ϵ-active constraints is detected. If no success is attained $D_\otimes$ is used.

Positive generators for the tangent cone

No ϵ-active constraints

The set of polling directions D_k is the positive spanning set $D_\otimes$.

For ϵ-active constraint(s)

D_k is the set of positive generators for the tangent cone of the ϵ-active constraints (obtained by QR factorization).

Degeneracy

The ϵ parameter is dynamically adapted when degeneracy in the ϵ-active constraints is detected. If no success is attained $D_\otimes$ is used.

Outline

Motivation for using particle swarm

Central idea

A particle swarm iteration is performed in the search step (using several particles).

Key points

- In the first iterations the algorithm takes advantage of the particle swarm ability to find a global optimum (exploiting the search space), while in the last iterations the algorithm takes advantage of the direct-search robustness to find a stationary point.
- The number of particles in the swarm can be decreased along the iterations (no need to have a large number of particles around a local optimum).

Motivation for using particle swarm

Central idea

A particle swarm iteration is performed in the search step (using several particles).

Key points

- In the first iterations the algorithm takes advantage of the particle swarm ability to find a global optimum (exploiting the search space), while in the last iterations the algorithm takes advantage of the direct-search robustness to find a stationary point.
- The number of particles in the swarm can be decreased along the iterations (no need to have a large number of particles around a local optimum).

Motivation for using particle swarm

Central idea

A particle swarm iteration is performed in the search step (using several particles).

Key points

- In the first iterations the algorithm takes advantage of the particle swarm ability to find a global optimum (exploiting the search space), while in the last iterations the algorithm takes advantage of the direct-search robustness to find a stationary point.
- The number of particles in the swarm can be decreased along the iterations (no need to have a large number of particles around a local optimum).

Motivation for using particle swarm

Central idea

A particle swarm iteration is performed in the search step (using several particles).

Key points

- In the first iterations the algorithm takes advantage of the particle swarm ability to find a global optimum (exploiting the search space), while in the last iterations the algorithm takes advantage of the direct-search robustness to find a stationary point.
- The number of particles in the swarm can be decreased along the iterations (no need to have a large number of particles around a local optimum).

Particle Swarm (new position and velocity)

The new particle position is updated by

Update particle

$$x^p_{k+1} = x^p_k + v^p_{k+1}, \quad p = 1, \dots, s.$$

v^p_{k+1} is the new velocity given by

Update velocity

$$v^p_{k+1} = \iota_k v^p_k + \mu\omega_{1k} \bullet \left(\bar{x}^p_k - x^p_k\right) + \nu\omega_{2k} \bullet \left(x_k - x^p_k\right),$$

where ι_k, μ and ν are parameters and ω_{1k} and ω_{2k} are random vectors drawn from the uniform $(0,1)$ distribution.

$\bar{x}^p_k$ is the best particle p position and x_k is the best population position.

Poll step

It is performed on x_k, i.e., on the best population position (leader).

Particle Swarm (new position and velocity)

The new particle position is updated by

Update particle

$$x_{k+1}^p = x_k^p + v_{k+1}^p, \quad p = 1, \ldots, s.$$

v_{k+1}^p is the new velocity given by

Update velocity

$$v_{k+1}^p = \iota_k v_k^p + \mu\omega_{1k} \bullet \left(\bar{x}_k^p - x_k^p\right) + \nu\omega_{2k} \bullet \left(x_k - x_k^p\right),$$

where ι_k, μ and ν are parameters and ω_{1k} and ω_{2k} are random vectors drawn from the uniform $(0,1)$ distribution.
$\bar{x}_k^p$ is the best particle p position and x_k is the best population position.

Poll step

It is performed on x_k, i.e., on the best population position (leader).

Particle Swarm (new position and velocity)

The new particle position is updated by

Update particle

$$x^p_{k+1} = x^p_k + v^p_{k+1}, \quad p = 1, \dots, s.$$

v^p_{k+1} is the new velocity given by

Update velocity

$$v^p_{k+1} = \iota_k v^p_k + \mu\omega_{1k} \bullet \left(\bar{x}^p_k - x^p_k\right) + \nu\omega_{2k} \bullet \left(x_k - x^p_k\right),$$

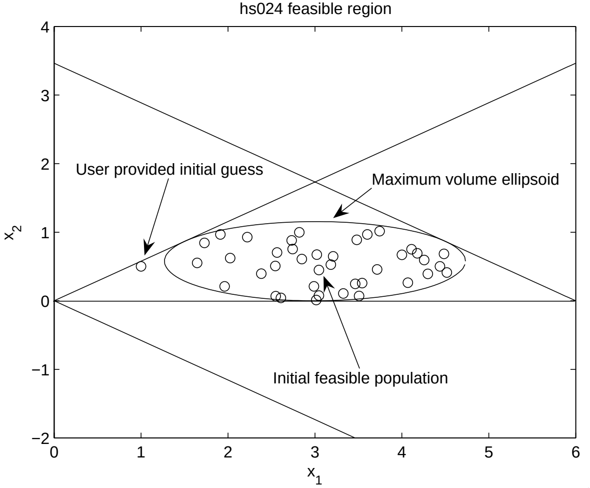
where ι_k, μ and ν are parameters and ω_{1k} and ω_{2k} are random vectors drawn from the uniform $(0, 1)$ distribution.
$\bar{x}^p_k$ is the best particle p position and x_k is the best population position.

Poll step

It is performed on x_k, i.e., on the best population position (leader).

Feasible initial population

Getting an initial feasible population allows a more efficient search for the global optimum.



Zhang and Gao interior-point code is being used to compute the maximum volume ellipsoid.

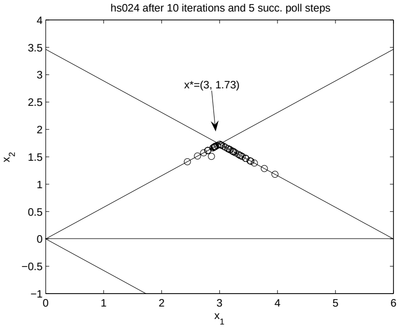
Search step (Particle Swarm)

Feasibility is kept during the optimization process for all particles. This is achieved by introducing a maximum allowed step in the *search direction*.

Maximum allowed step

$$x^p_{k+1} = x^p_k + \alpha_{max} v^p_{k+1},$$

where α_{max} is the maximum step allowed to keep x^p_{k+1} inside the feasible region.



Testing environment — bound constrained

Test problems

- 122 problems.
- Including 12 are of large dimension (100-300 variables).

Solvers used

- ASA – Adaptative Simulated Annealing
- PSwarm – (our approach: Pattern Search with Particle Swarm step)
- PGAPack – Parallel Genetic Algorithms Package
- Direct – Dividing Rectangles.
- MCS – Multilevel Coordinate Search.

Testing environment — bound constrained

Test problems

- 122 problems.
- Including 12 are of large dimension (100-300 variables).

Solvers used

- ASA – Adaptative Simulated Annealing
- PSwarm – (our approach: Pattern Search with Particle Swarm step)
- PGAPack – Parallel Genetic Algorithms Package
- Direct – Dividing Rectangles
- MCS – Multilevel Coordinate Search

Testing environment — bound constrained

Test problems

- 122 problems.
- Including 12 are of large dimension (100-300 variables).

Solvers used

- ASA – Adaptative Simulated Annealing.
- PSwarm – (our approach: Pattern Search with Particle Swarm step)
- PGAPack – Parallel Genetic Algorithms Package.
- Direct – Dividing Rectangles.
- MCS – Multilevel Coordinate Search.

Testing environment — bound constrained

Test problems

- 122 problems.
- Including 12 are of large dimension (100-300 variables).

Solvers used

- ASA – Adaptative Simulated Annealing.
- PSwarm – (our approach: Pattern Search with Particle Swarm step).
- PGAPack – Parallel Genetic Algorithms Package.
- Direct – Dividing Rectangles.
- MCS – Multilevel Coordinate Search.

Testing environment — bound constrained

Test problems

- 122 problems.
- Including 12 are of large dimension (100-300 variables).

Solvers used

- ASA – Adaptative Simulated Annealing.
- PSwarm – (our approach: Pattern Search with Particle Swarm step).
- PGAPack – Parallel Genetic Algorithms Package.
- Direct – Dividing Rectangles.
- MCS – Multilevel Coordinate Search.

Testing environment — bound constrained

Test problems

- 122 problems.
- Including 12 are of large dimension (100-300 variables).

Solvers used

- ASA – Adaptative Simulated Annealing.
- PSwarm – (our approach: Pattern Search with Particle Swarm step).
- PGAPack – Parallel Genetic Algorithms Package.
- Direct – Dividing Rectangles.
- MCS – Multilevel Coordinate Search.

Testing environment — bound constrained

Test problems

- 122 problems.
- Including 12 are of large dimension (100-300 variables).

Solvers used

- ASA – Adaptative Simulated Annealing.
- PSwarm – (our approach: Pattern Search with Particle Swarm step).
- PGAPack – Parallel Genetic Algorithms Package.
- Direct – Dividing Rectangles.
- MCS – Multilevel Coordinate Search.

Testing environment — bound constrained

Test problems

- 122 problems.
- Including 12 are of large dimension (100-300 variables).

Solvers used

- ASA – Adaptative Simulated Annealing.
- PSwarm – (our approach: Pattern Search with Particle Swarm step).
- PGAPack – Parallel Genetic Algorithms Package.
- Direct – Dividing Rectangles.
- MCS – Multilevel Coordinate Search.

Testing environment — bound constrained

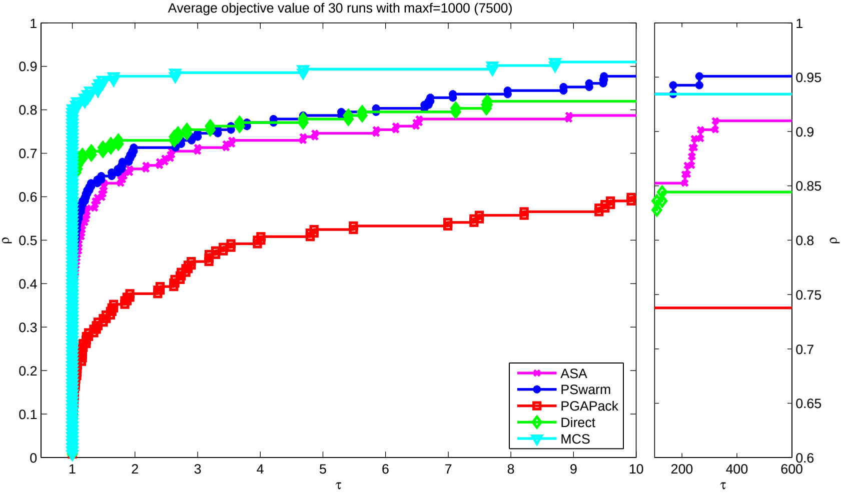
Test problems

- 122 problems.
- Including 12 are of large dimension (100-300 variables).

Solvers used

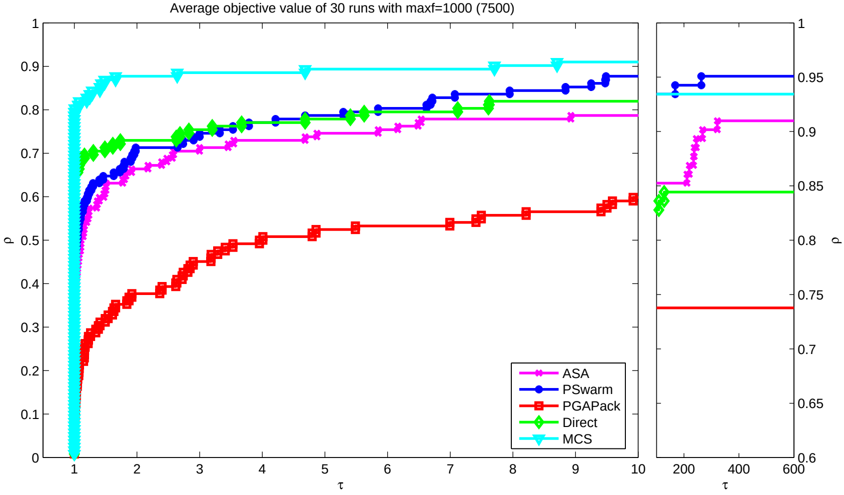
- ASA – Adaptative Simulated Annealing.
- PSwarm – (our approach: Pattern Search with Particle Swarm step).
- PGAPack – Parallel Genetic Algorithms Package.
- Direct – Dividing Rectangles.
- MCS – Multilevel Coordinate Search.

Numerical results (final value for f)



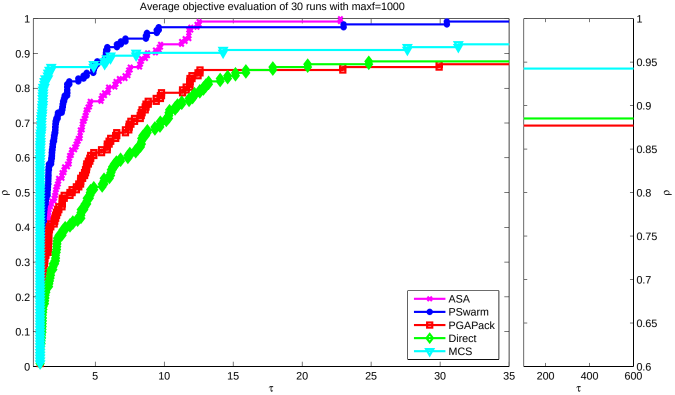
For further details see Vaz and Vicente, JOGO, 2007.

Numerical results (final value for f)



For further details see Vaz and Vicente, JOGO, 2007.

Numerical results (number of evaluations)



Average number of objective function evaluations.

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

Testing environment — linear constrained

Test problems

- 120 problems with linear constraints were collected from 1564 optimization problems (AMPL, CUTE, GAMS, NETLIB, etc.).
- 23 linear, 55 quadratic and 32 general nonlinear.
- 10 highly non-convex objective functions with random generated linear constraints (Pinter).

Solvers used

- PSwarm – (our approach: Pattern Search with Particle Swarm step).
- ASA – Adaptative Simulated Annealing.
- Direct – Dividing Rectangles.
- NOMAD – A blackbox optimization software (MATLAB version).

Testing environment — linear constrained

Test problems

- 120 problems with linear constraints were collected from 1564 optimization problems (AMPL, CUTE, GAMS, NETLIB, etc.).
- 23 linear, 55 quadratic and 32 general nonlinear.
- 10 highly non-convex objective functions with random generated linear constraints (Pinter).

Solvers used

- PSwarm – (our approach: Pattern Search with Particle Swarm step).
- ASA – Adaptative Simulated Annealing.
- Direct – Dividing Rectangles.
- NOMAD – A blackbox optimization software (MATLAB version).

Testing environment — linear constrained

Test problems

- 120 problems with linear constraints were collected from 1564 optimization problems (AMPL, CUTE, GAMS, NETLIB, etc.).
- 23 linear, 55 quadratic and 32 general nonlinear.
- 10 highly non-convex objective functions with random generated linear constraints (Pinter).

Solvers used

- PSwarm – (our approach: Pattern Search with Particle Swarm step)
- ASA – Adaptative Simulated Annealing
- Direct – Dividing Rectangles.
- NOMAD – A blackbox optimization software (MATLAB version)

Testing environment — linear constrained

Test problems

- 120 problems with linear constraints were collected from 1564 optimization problems (AMPL, CUTE, GAMS, NETLIB, etc.).
- 23 linear, 55 quadratic and 32 general nonlinear.
- 10 highly non-convex objective functions with random generated linear constraints (Pinter).

Solvers used

- PSwarm – (our approach: Pattern Search with Particle Swarm step).
- ASA – Adaptative Simulated Annealing.
- Direct – Dividing Rectangles.
- NOMAD – A blackbox optimization software (MATLAB version).

Testing environment — linear constrained

Test problems

- 120 problems with linear constraints were collected from 1564 optimization problems (AMPL, CUTE, GAMS, NETLIB, etc.).
- 23 linear, 55 quadratic and 32 general nonlinear.
- 10 highly non-convex objective functions with random generated linear constraints (Pinter).

Solvers used

- PSwarm – (our approach: Pattern Search with Particle Swarm step).
- ASA – Adaptative Simulated Annealing.
- Direct – Dividing Rectangles.
- NOMAD – A blackbox optimization software (MATLAB version).

Testing environment — linear constrained

Test problems

- 120 problems with linear constraints were collected from 1564 optimization problems (AMPL, CUTE, GAMS, NETLIB, etc.).
- 23 linear, 55 quadratic and 32 general nonlinear.
- 10 highly non-convex objective functions with random generated linear constraints (Pinter).

Solvers used

- PSwarm – (our approach: Pattern Search with Particle Swarm step).
- ASA – Adaptative Simulated Annealing.
- Direct – Dividing Rectangles.
- NOMAD – A blackbox optimization software (MATLAB version).

Testing environment — linear constrained

Test problems

- 120 problems with linear constraints were collected from 1564 optimization problems (AMPL, CUTE, GAMS, NETLIB, etc.).
- 23 linear, 55 quadratic and 32 general nonlinear.
- 10 highly non-convex objective functions with random generated linear constraints (Pinter).

Solvers used

- PSwarm – (our approach: Pattern Search with Particle Swarm step).
- ASA – Adaptative Simulated Annealing.
- Direct – Dividing Rectangles.
- NOMAD – A blackbox optimization software (MATLAB version).

Testing environment — linear constrained

Test problems

- 120 problems with linear constraints were collected from 1564 optimization problems (AMPL, CUTE, GAMS, NETLIB, etc.).
- 23 linear, 55 quadratic and 32 general nonlinear.
- 10 highly non-convex objective functions with random generated linear constraints (Pinter).

Solvers used

- PSwarm – (our approach: Pattern Search with Particle Swarm step).
- ASA – Adaptative Simulated Annealing.
- Direct – Dividing Rectangles.
- NOMAD – A blackbox optimization software (MATLAB version).

Testing environment — linear constrained

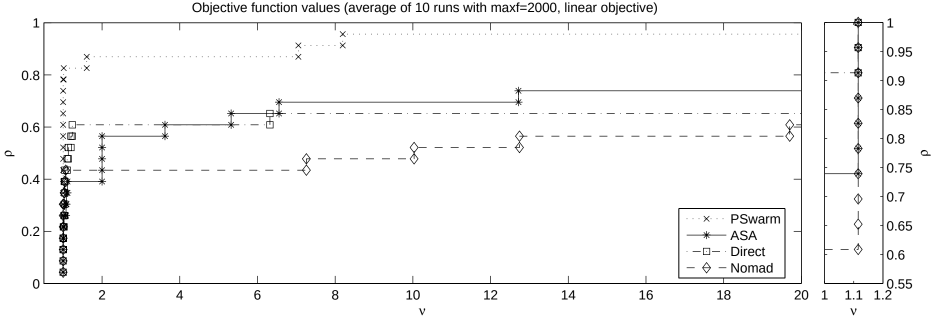
Test problems

- 120 problems with linear constraints were collected from 1564 optimization problems (AMPL, CUTE, GAMS, NETLIB, etc.).
- 23 linear, 55 quadratic and 32 general nonlinear.
- 10 highly non-convex objective functions with random generated linear constraints (Pinter).

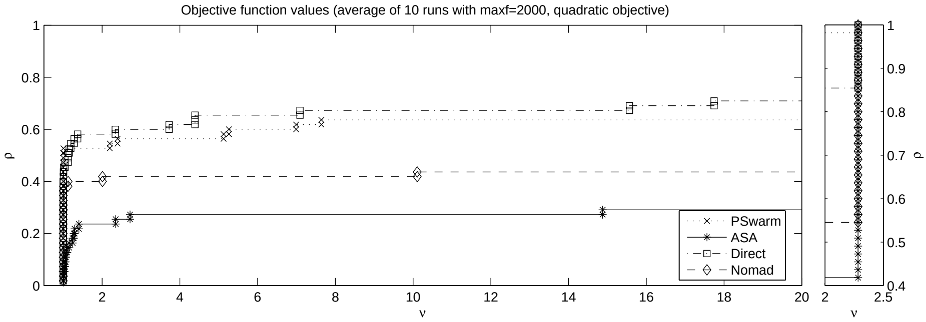
Solvers used

- PSwarm – (our approach: Pattern Search with Particle Swarm step).
- ASA – Adaptative Simulated Annealing.
- Direct – Dividing Rectangles.
- NOMAD – A blackbox optimization software (MATLAB version).

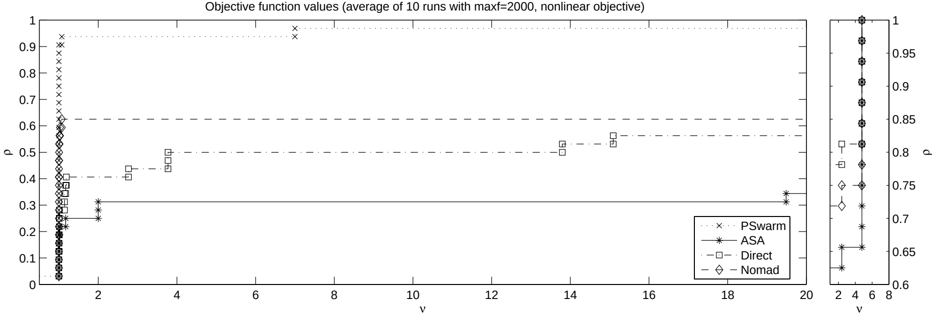
Linear objective functions



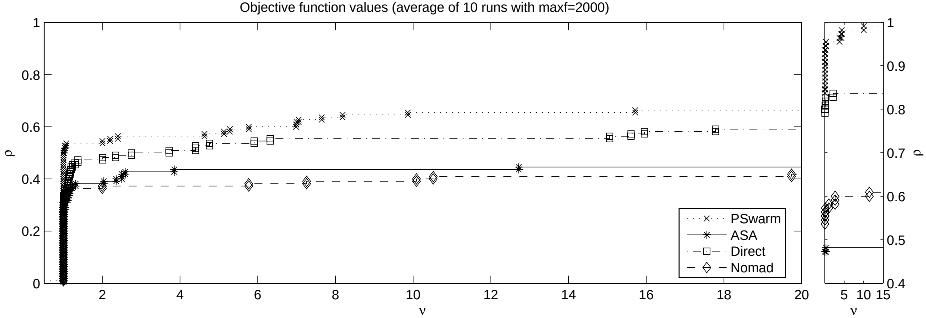
Quadratic objective functions



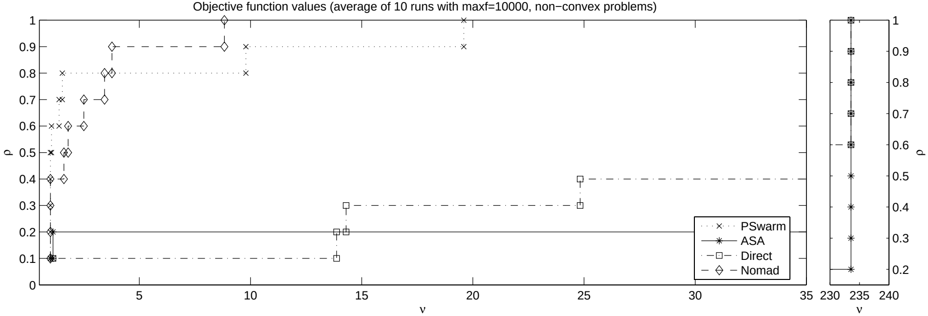
General nonlinear objective functions



All objective functions

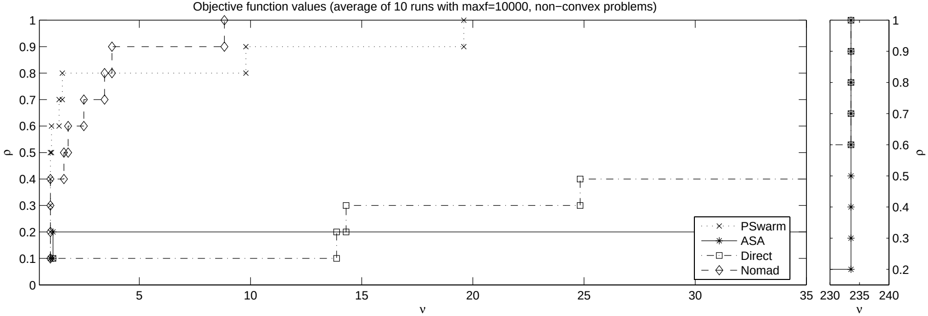


Highly non-convex objective functions



For further details see Vaz and Vicente, OMS, 2009.

Highly non-convex objective functions



For further details see Vaz and Vicente, OMS, 2009.

Outline

1. Direct search
2. Using particle swarm in the search step
3. Using radial basis functions in the search step
4. Conclusions

Motivation for using RBFs

Main idea

- To take advantage of direct-search methods of directional type where the iterations can be divided into two main steps (a search step and a poll step).
- Consists of forming and minimizing an Radial Basis Function (RBF) model in the search step.
- The RBF model can be used to order the poll set of directions.

Motivation for using RBFs

Main idea

- To take advantage of direct-search methods of directional type where the iterations can be divided into two main steps (a search step and a poll step).
- Consists of forming and minimizing an Radial Basis Function (RBF) model in the search step.
- The RBF model can be used to order the poll set of directions.

Motivation for using RBFs

Main idea

- To take advantage of direct-search methods of directional type where the iterations can be divided into two main steps (a search step and a poll step).
- Consists of forming and minimizing an Radial Basis Function (RBF) model in the search step.
- The RBF model can be used to order the poll set of directions.

Motivation for using RBFs

Main idea

- To take advantage of direct-search methods of directional type where the iterations can be divided into two main steps (a search step and a poll step).
- Consists of forming and minimizing an Radial Basis Function (RBF) model in the search step.
- The RBF model can be used to order the poll set of directions.

Radial Basis Functions (RBFs)

In order to interpolate a function f whose values on a set $Y = \{y^1, \ldots, y^{n_p}\} \subset \mathbb{R}^n$ are known, one can use a RBF model of the form

$$m(x) \; = \; \sum_{i=1}^{n_p} \lambda_i \phi(\|x - y^i\|),$$

where $\phi(\| \cdot \|)$, with $\phi : \mathbb{R}_+ \to \mathbb{R}$, is a radial basis function and $\lambda_1, \ldots, \lambda_{n_p} \in \mathbb{R}$ are parameters to be determined.

Property

For $m(x)$ to be C^2, the function $\phi(x)$ must be both C^2 and $\phi'(0) = 0$.

Radial Basis Functions (RBFs)

In order to interpolate a function f whose values on a set $Y = \{y^1, \ldots, y^{n_p}\} \subset \mathbb{R}^n$ are known, one can use a RBF model of the form

$$m(x) = \sum_{i=1}^{n_p} \lambda_i \phi(\|x - y^i\|),$$

where $\phi(\|\cdot\|)$, with $\phi : \mathbb{R}_+ \rightarrow \mathbb{R}$, is a radial basis function and $\lambda_1, \ldots, \lambda_{n_p} \in \mathbb{R}$ are parameters to be determined.

Property

For $m(x)$ to be C^2, the function $\phi(x)$ must be both C^2 and $\phi'(0) = 0$.

Radial Basis Functions (RBFs)

In many applications, it is desirable that the linear space spanned by the basis functions includes constant or linear functions.

One can augment RBF model by allowing a low-order *polynomial tail*. The new model is now of the form

$$m(x) \ = \sum_{i=1}^{n_p} \lambda_i \phi(\|x - y^i\|) + \sum_{j=0}^{q} \gamma_j p_j(x),$$

where $p_j(x)$, $j = 0, \ldots, q$, are some basis functions for the polynomial and $\gamma_0, \ldots, \gamma_q \in \mathbb{R}$.

Radial Basis Functions (RBFs)

The coefficients λ's are required to satisfy

$$\sum_{i=1}^{n_p} \lambda_i p_j(y^i) \;=\; 0, \quad j = 0, \ldots, q.$$

These, in conjunction with the interpolation conditions $m(y^i) = f(y^i)$, $i = 1, \ldots, n_p$, give the linear system

$$\left[\begin{array}{cc} \Phi & P \\ P^\top & 0 \end{array} \right] \left[\begin{array}{c} \lambda \\ \gamma \end{array} \right] = \left[\begin{array}{c} f(Y) \\ 0 \end{array} \right],$$

where $\Phi_{ij} = \phi(\|y^i - y^j\|)$ for $i, j \in \{1, \ldots, n_p\}$, $P_{ij} = p_j(y^i)$ for $i \in \{1, \ldots, n_p\}$, $j \in \{0, \ldots, q\}$, and $f(Y)$ is the vector formed by the values $f(y^1), \ldots, f(y^{n_p})$.

Radial Basis Functions (RBFs)

The polynomial tails most frequently used in the context of RBF are linear, and we will write $t(x) = c + g^\top x$ and

$$m(x) \ = \ \sum_{i=1}^{n_p} \lambda_i \phi(\|x - y^i\|) + t(x).$$

This model has $n_p + n + 1$ parameters, n_p for the radial basis terms and $n + 1$ for the linear polynomial terms.

Radial Basis Functions (RBFs)

Common/Used approach

Common approaches for derivative-free optimization use cubic RBFs and linear polynomial tails

$$m(x) = \sum_{i=1}^{n_p} \lambda_i \|x - y^i\|^3 + t(x).$$

Constraints handling

Bound constraints

They can simply be considered in the minimization of the RBF model.

Linear constraints

- They are temporarily removed from the RBF model minimization and then we project the minimizer onto Ω.
- For a feasible initial guess and the set of poll directions we use the same strategies already shown.

Constraints handling

Bound constraints

They can simply be considered in the minimization of the RBF model.

Linear constraints

- They are temporarily removed from the RBF model minimization and then we project the minimizer onto Ω.
- For a feasible initial guess and the set of poll directions we use the same strategies already shown.

Constraints handling

Bound constraints

They can simply be considered in the minimization of the RBF model.

Linear constraints

- They are temporarily removed from the RBF model minimization and then we project the minimizer onto Ω.
- For a feasible initial guess and the set of poll directions we use the same strategies already shown.

Constraints handling

Bound constraints

They can simply be considered in the minimization of the RBF model.

Linear constraints

- They are temporarily removed from the RBF model minimization and then we project the minimizer onto Ω.
- For a feasible initial guess and the set of poll directions we use the same strategies already shown.

RBF model subproblem

Thus, the RBF model subproblem we are addressing is

$$\min \ \ m(x) \ \ \text{s.t.} \ \ x \in \bar{\Omega},$$

where $\bar{\Omega}$ is the feasible region defined by upper and lower bounds on the variables, *i.e.*, $\bar{\Omega} = [\ell, u] \cap B_\infty(x_k; \sigma\alpha_k)$.

Solvers used for subproblems

RBF model subproblem

Thus, the RBF model subproblem we are addressing is

$$\min \quad m(x) \quad \text{s.t.} \quad x \in \bar{\Omega},$$

where $\bar{\Omega}$ is the feasible region defined by upper and lower bounds on the variables, *i.e.*, $\bar{\Omega} = [\ell, u] \cap B_\infty(x_k; \sigma\alpha_k)$.

Solvers used for subproblems

- Difference of Convex (D.C.) algorithm, in order to take advantage of the RBF structure.
- `fmincon` from the MATLAB optimization toolbox.

RBF model subproblem

Thus, the RBF model subproblem we are addressing is

$$\min \quad m(x) \;\text{ s.t. }\; x \in \bar{\Omega},$$

where $\bar{\Omega}$ is the feasible region defined by upper and lower bounds on the variables, *i.e.*, $\bar{\Omega} = [\ell, u] \cap B_\infty(x_k; \sigma\alpha_k)$.

Solvers used for subproblems

- Difference of Convex (D.C.) algorithm, in order to take advantage of the RBF structure.
- `fmincon` from the MATLAB optimization toolbox.

RBF model subproblem

Thus, the RBF model subproblem we are addressing is

$$\min \quad m(x) \ \text{ s.t. } \ x \in \bar{\Omega},$$

where $\bar{\Omega}$ is the feasible region defined by upper and lower bounds on the variables, *i.e.*, $\bar{\Omega} = [\ell, u] \cap B_\infty(x_k; \sigma\alpha_k)$.

Solvers used for subproblems

- Difference of Convex (D.C.) algorithm, in order to take advantage of the RBF structure.
- `fmincon` from the MATLAB optimization toolbox.

Testing environment

Test problems

- The test set used in the numerical results includes 119 bound constrained problems and 109 linearly constrained problems coded in the AMPL format.
- In all cases, the stopping criteria consisted of reaching a maximum budget of 1000 function evaluations or driving the step size parameter α_k below 10^{-5}.

Testing environment

Test problems

- The test set used in the numerical results includes 119 bound constrained problems and 109 linearly constrained problems coded in the AMPL format.
- In all cases, the stopping criteria consisted of reaching a maximum budget of 1000 function evaluations or driving the step size parameter α_k below 10^{-5}.

Testing environment

Test problems

- The test set used in the numerical results includes 119 bound constrained problems and 109 linearly constrained problems coded in the AMPL format.
- In all cases, the stopping criteria consisted of reaching a maximum budget of 1000 function evaluations or driving the step size parameter α_k below 10^{-5}.

Testing environment

Solvers

- Pattern – Simple coordinate search with an empty search step.
- PSwarm – (our previous approach: Pattern Search with Particle Swarm step).
- RBF – (our new approach: RBF model in the search step), using for subproblem minimization:
  - DCA – D.C. algorithm.
  - fmincon.
  - DCA Sort – D.C. algorithm (polling order according to RBF model values).

Testing environment

Solvers

- Pattern – Simple coordinate search with an empty search step.
- PSwarm – (our previous approach: Pattern Search with Particle Swarm step).
- RBF – (our new approach: RBF model in the search step), using for subproblem minimization:
  - DCA – D.C. algorithm
  - fmincon
  - DCA Sort – D.C. algorithm (polling order according to RBF model values)

Testing environment

Solvers

- Pattern – Simple coordinate search with an empty search step.
- PSwarm – (our previous approach: Pattern Search with Particle Swarm step).
- RBF – (our new approach: RBF model in the search step), using for subproblem minimization:
  - DCA – D.C. algorithm.
  - `fmincon`
  - DCA Sort – D.C. algorithm (polling order according to RBF model values)

Testing environment

Solvers

- Pattern – Simple coordinate search with an empty search step.
- PSwarm – (our previous approach: Pattern Search with Particle Swarm step).
- RBF – (our new approach: RBF model in the search step), using for subproblem minimization:
  - DCA – D.C. algorithm.
  - `fmincon`.
  - DCA Sort – D.C. algorithm (polling order according to RBF model values).

Testing environment

Solvers

- Pattern – Simple coordinate search with an empty search step.
- PSwarm – (our previous approach: Pattern Search with Particle Swarm step).
- RBF – (our new approach: RBF model in the search step), using for subproblem minimization:
  - DCA – D.C. algorithm.
  - `fmincon`.
  - DCA Sort – D.C. algorithm (polling order according to RBF model values).

Testing environment

Solvers

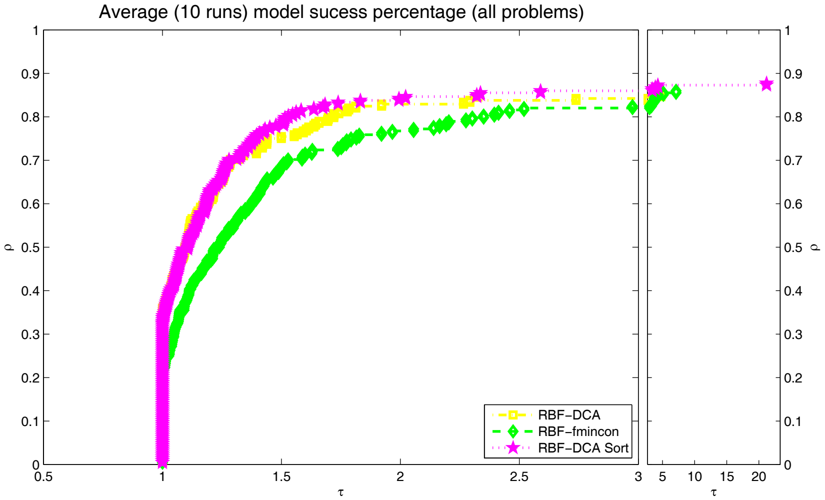
- Pattern – Simple coordinate search with an empty search step.
- PSwarm – (our previous approach: Pattern Search with Particle Swarm step).
- RBF – (our new approach: RBF model in the search step), using for subproblem minimization:
  - DCA – D.C. algorithm.
  - `fmincon`.
  - DCA Sort – D.C. algorithm (polling order according to RBF model values).

Testing environment

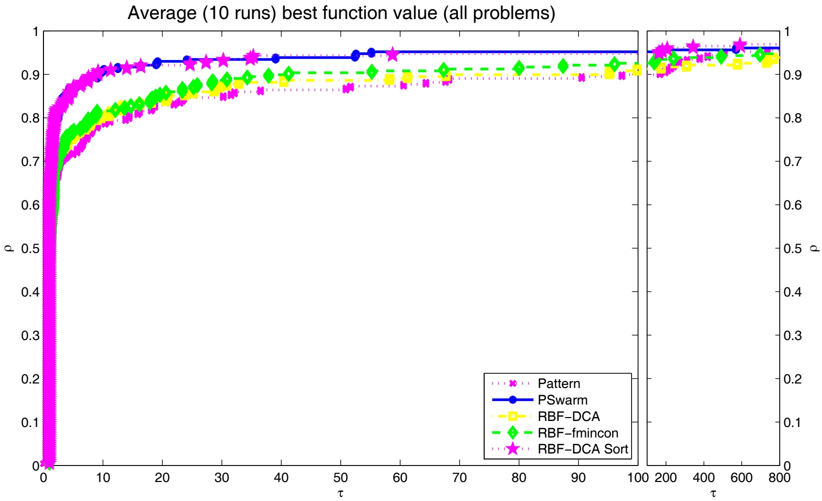
Solvers

- Pattern – Simple coordinate search with an empty search step.
- PSwarm – (our previous approach: Pattern Search with Particle Swarm step).
- RBF – (our new approach: RBF model in the search step), using for subproblem minimization:
  - DCA – D.C. algorithm.
  - `fmincon`.
  - DCA Sort – D.C. algorithm (polling order according to RBF model values).

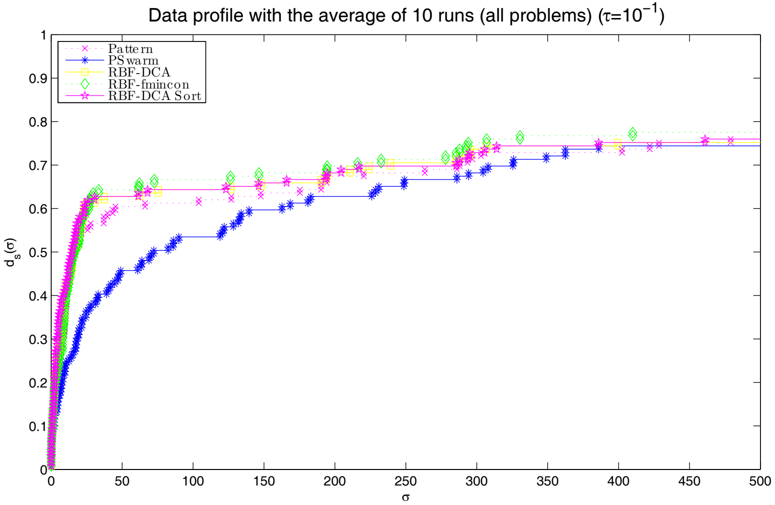
Numerical results



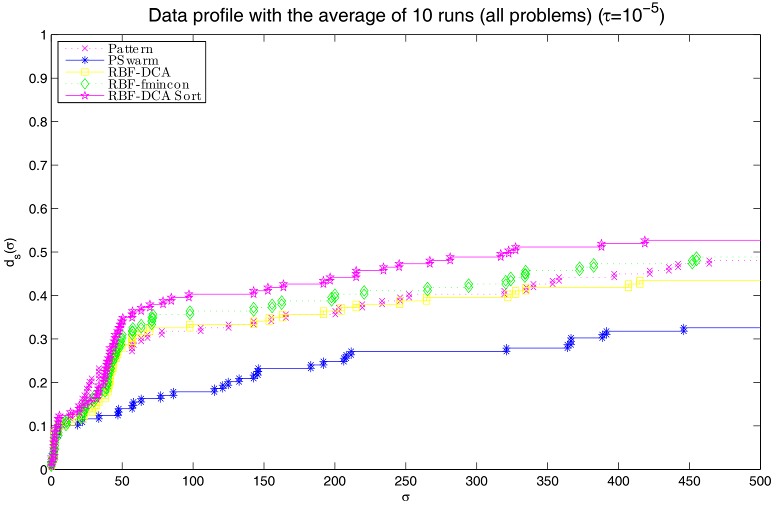
Numerical results



Numerical results



Numerical results



Outline

Conclusions

- Description of two strategies for enhancing the search step of direct-search algorithms.
- Using the search step of direct-search algorithms is advantageous.
- Numerical results confirm the improvement in solvers efficiency and robustness.

Conclusions

- Description of two strategies for enhancing the search step of direct-search algorithms.
- Using the search step of direct-search algorithms is advantageous.
- Numerical results confirm the improvement in solvers efficiency and robustness.

Conclusions

- Description of two strategies for enhancing the search step of direct-search algorithms.
- Using the search step of direct-search algorithms is advantageous.
- Numerical results confirm the improvement in solvers efficiency and robustness.

References

Le Thi Hoai An, A.I.F. Vaz, and L.N. Vicente.
Optimizing radial basis functions by d.c. programming and its use in direct search for global derivative-free optimization.
Technical Report 09-37, Univ. Coimbra, 2009.

A.I.F. Vaz and L. N. Vicente.
PSwarm: A hybrid solver for linearly constrained global derivative-free optimization.
Optimization Methods and Software, 24:669–685, 2009.

A.I.F. Vaz and L.N. Vicente.
A particle swarm pattern search method for bound constrained global optimization.
Journal of Global Optimization, 39:197–219, 2007.

Yin Zhang and Liyan Gao.
On numerical solution of the maximum volume ellipsoid problem.
SIAM Journal on Optimization, 14:53–76, 2003.

Optimization 2011 (July 24–27, Portugal)

2011
optimization
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
Caparica *Campus*
July 24-27

plenary speakers

Gilbert Laporte | HEC Montréal
New trends in vehicle routing

Jean Bernard Lasserre | LAAS-CNRS, Toulouse
Moments and semidefinite relaxations for parametric optimization

José Mario Martínez | State University of Campinas
Unifying inexact restoration, SQP, and augmented Lagrangian methods

Mauricio G.C. Resende | AT&T Labs - Research
Using metaheuristics to solve real optimization problems in telecommunications

Nick Sahinidis | Carnegie Mellon University
Recent advances in nonconvex optimization

Stephen J. Wright | University of Wisconsin
Algorithms and applications in sparse optimization