# A genetic algorithm framework for multilocal optimization

A. Ismael F. Vaz
Lino Costa

Production and Systems Department,
School of engineering,
University of Minho, Gualtar Campus,
4710 - 057 Braga, Portugal;
Email: {lac,aivaz}@dps.uminho.pt

EURO XXIII, Bonn, Germany

5-8 July 2009

# Outline

# Outline

# Outline

# Outline

# Outline

# Outline

# Outline

# Addressed problem

## The following problem is under consideration

$$\min_{x \in \Omega \subset R^n} f(x) \tag{1}$$

where $f(x)$ is the objective function, $x = (x_1, \dots, x_n)^T$ is an $n$ dimensional vector and $\Omega \subset R^n$ is the feasible set, herein assumed to be a cartesian product of intervals with finite bounds ($\Omega = [\alpha_1, \beta_1] \times \cdots \times [\alpha_n, \beta_n]$).

## Multi-local

We aim to compute approximations to all the local optima for problem (1).

## Assumptions

$f(x)$ is assumed to be twice continuous differentiable and cheap to evaluate (*i.e.* the number of objective function evaluations is not a concern).

# Addressed problem

## The following problem is under consideration

$$\min_{x \in \Omega \subset R^n} f(x) \qquad (1)$$

where $f(x)$ is the objective function, $x = (x_1, \ldots, x_n)^T$ is an $n$ dimensional vector and $\Omega \subset R^n$ is the feasible set, herein assumed to be a cartesian product of intervals with finite bounds ($\Omega = [\alpha_1, \beta_1] \times \cdots \times [\alpha_n, \beta_n]$).

## Multi-local

We aim to compute approximations to all the local optima for problem (1).

## Assumptions

$f(x)$ is assumed to be twice continuous differentiable and cheap to evaluate (*i.e.* the number of objective function evaluations is not a concern).

# Addressed problem

## The following problem is under consideration

$$\min_{x \in \Omega \subset R^n} f(x) \tag{1}$$

where $f(x)$ is the objective function, $x = (x_1, \ldots, x_n)^T$ is an $n$ dimensional vector and $\Omega \subset R^n$ is the feasible set, herein assumed to be a cartesian product of intervals with finite bounds ($\Omega = [\alpha_1, \beta_1] \times \cdots \times [\alpha_n, \beta_n]$).

## Multi-local

We aim to compute approximations to all the local optima for problem (1).

## Assumptions

$f(x)$ is assumed to be twice continuous differentiable and cheap to evaluate (*i.e.* the number of objective function evaluations is not a concern).

# Outline

# Algorithm framework

## Genetic algorithms

A genetic algorithm is a population based algorithm that uses techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover.

## A GA framework

1. Randomly initialize an initial population (real-parameter representation).
2. Compute a set of Parents from the elite population using a *fitness* function (tournament selection).
3. Compute a set of Offsprings obtained from the set of Parents using the *crossover* and *mutation* operators.
4. Verify the stopping criteria. If not met then goto step 2.

# Algorithm framework

## Genetic algorithms

A genetic algorithm is a population based algorithm that uses techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover.

## A GA framework

1. Randomly initialize an initial population (real-parameter representation).
2. Compute a set of Parents from the elite population using a *fitness* function (tournament selection).
3. Compute a set of Offsprings obtained from the set of Parents using the *crossover* and *mutation* operators.
4. Verify the stopping criteria. If not met then goto step 2.

# Algorithm framework

## Genetic algorithms

A genetic algorithm is a population based algorithm that uses techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover.

## A GA framework

1. Randomly initialize an initial population (real-parameter representation).
2. Compute a set of Parents from the elite population using a *fitness* function (tournament selection).
3. Compute a set of Offsprings obtained from the set of Parents using the *crossover* and *mutation* operators.
4. Verify the stopping criteria. If not met then goto step 2.

# Algorithm framework

## Genetic algorithms

A genetic algorithm is a population based algorithm that uses techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover.

## A GA framework

1. Randomly initialize an initial population (real-parameter representation).
2. Compute a set of Parents from the elite population using a *fitness* function (tournament selection).
3. Compute a set of Offsprings obtained from the set of Parents using the *crossover* and *mutation* operators.
4. Verify the stopping criteria. If not met then goto step 2.

# Algorithm framework

## Genetic algorithms

A genetic algorithm is a population based algorithm that uses techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover.

## A GA framework

1. Randomly initialize an initial population (real-parameter representation).
2. Compute a set of Parents from the elite population using a *fitness* function (tournament selection).
3. Compute a set of Offsprings obtained from the set of Parents using the *crossover* and *mutation* operators.
4. Verify the stopping criteria. If not met then goto step 2.

# Algorithm framework

## Genetic algorithms

A genetic algorithm is a population based algorithm that uses techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover.

## A GA framework

1. Randomly initialize an initial population (real-parameter representation).
2. Compute a set of Parents from the elite population using a *fitness* function (tournament selection).
3. Compute a set of Offsprings obtained from the set of Parents using the *crossover* and *mutation* operators.
4. Verify the stopping criteria. If not met then goto step 2.

# Typical Operators

### The *fitness* function

Usually the *fitness* function corresponds to the objective function. The points in the population are sorted by the objective function value.

### The Tournament Selection

Tournaments are played between points and the better solution is chosen as a Parent (survival of the fittest). The process is repeated until the set of Parents is fulfilled.

# Typical Operators

## The *fitness* function

Usually the *fitness* function corresponds to the objective function. The points in the population are sorted by the objective function value.

## The Tournament Selection

Tournaments are played between points and the better solution is chosen as a Parent (survival of the fittest). The process is repeated until the set of Parents is fulfilled.

# Typical Operators

**The *crossover* operator**

Simulated Binary Crossover (SBX) that simulates the working principle of single-point crossover operator for binary strings.

**The *mutation* operator**

Polynomial mutation that guarantees that the probability of creating a point closer to the parent is more than the probability of creating one away from it.

# Typical Operators

### The *crossover* operator

Simulated Binary Crossover (SBX) that simulates the working principle of single-point crossover operator for binary strings.

### The *mutation* operator

Polynomial mutation that guarantees that the probability of creating a point closer to the parent is more than the probability of creating one away from it.

# Outline

# The proposed *fitness* function

## The *fitness* function

The use of the objective function as the *fitness* function may no longer be appropriate.

## The proposed *fitness* function - Version 1

Points in the population that are nearby each other (e.g. using the Euclidean distance) are assigned a huge *fitness* value, since diversity near a local optima is not requested.

The objective function is used as the *fitness* function.

# The proposed *fitness* function

## The *fitness* function

The use of the objective function as the *fitness* function may no longer be appropriate.

## The proposed *fitness* function - Version 1

Points in the population that are nearby each other (e.g. using the Euclidean distance) are assigned a huge *fitness* value, since diversity near a local optima is not requested.

The objective function is used as the *fitness* function.

# The proposed *fitness* function

## The proposed *fitness* function - Version 2

The second version of the proposed *fitness* function uses some concepts from the multiobjective (biobjective) optimization.

## The dominance concept

Let $f_1(x)$ ($\equiv f(x)$) and $f_2(x)$ ($\equiv ||\nabla L(x,\lambda)||$[a]) be two objective functions. A point $x$ is said to dominate a point $y$ ($x \prec y$) if

$$f_i(x) \leq f_i(y), \quad i = 1,2 \quad \text{and} \quad \exists j \quad \text{such that} \quad f_j(x) < f_j(y). \qquad (2)$$

If the conditions in (2) are not verified we simply say that $x$ does not dominate $y$ ($x \nprec y$).

[a] $L(x,\lambda)$ is the Lagrangian function with Lagrange multipliers $\lambda$.

# The proposed *fitness* function

## The proposed *fitness* function - Version 2

The second version of the proposed *fitness* function uses some concepts from the multiobjective (biobjective) optimization.

## The dominance concept

Let $f_1(x)$ ($\equiv f(x)$) and $f_2(x)$ ($\equiv ||\nabla L(x,\lambda)||$[a]) be two objective functions. A point $x$ is said to dominate a point $y$ ($x \prec y$) if

$$f_i(x) \leq f_i(y), \quad i = 1,2 \quad \text{and} \quad \exists j \quad \text{such that} \quad f_j(x) < f_j(y). \tag{2}$$

If the conditions in (2) are not verified we simply say that $x$ does not dominate $y$ ($x \not\prec y$).

[a] $L(x,\lambda)$ is the Lagrangian function with Lagrange multipliers $\lambda$.

# The proposed *fitness* function

## Lagrangian computation

Since we are dealing with a simple bound constrained problem the norm of the Lagrangian gradient can be computed without explicitly knowing the Lagrange multipliers.

## The proposed *fitness* function - Version 2

Points in the population not dominated by any other point (in the population) is assigned rank 1 (*fitness* equal to 1) and removed from the *fitness* computation. Points thereafter not dominated by any other point (in the remaining point of the population) is assigned rank 2. The procedure is repeated until no point are left to assign a rank.

A point $x$ only dominates a point $y$ if it satisfies relations (2) and is within a specified Euclidian distance.

# The proposed *fitness* function

## Lagrangian computation

Since we are dealing with a simple bound constrained problem the norm of the Lagrangian gradient can be computed without explicitly knowing the Lagrange multipliers.

## The proposed *fitness* function - Version 2

Points in the population not dominated by any other point (in the population) is assigned rank 1 (*fitness* equal to 1) and removed from the *fitness* computation. Points thereafter not dominated by any other point (in the remaining point of the population) is assigned rank 2. The procedure is repeated until no point are left to assign a rank.

A point $x$ only dominates a point $y$ if it satisfies relations (2) and is within a specified Euclidian distance.

# Quasi-Newton iterations

A quasi-Newton iteration is performed for each point in the elite population:

- a BFGS update formula to approximate the inverse Lagrangian Hessian is used.
- The identity matrix is used whenever a new point enters the elite population.
- A point kept in the elite population performs a sequence of quasi-Newton iterations (independently of the fitness).
- a line search strategy is used together with an Armijo like rule in order to globalize the algorithm.
- A reset strategy (setting the approximation to the identity matrix) is performed each $n$ successive iterations and the inverse Lagrangian Hessian approximation is not updated if the denominator of the updating formula is, in absolute value, lower than a specified tolerance.

# Quasi-Newton iterations

A quasi-Newton iteration is performed for each point in the elite population:

- a BFGS update formula to approximate the inverse Lagrangian Hessian is used.
- The identity matrix is used whenever a new point enters the elite population.
- A point kept in the elite population performs a sequence of quasi-Newton iterations (independently of the fitness).
- a line search strategy is used together with an Armijo like rule in order to globalize the algorithm.
- A reset strategy (setting the approximation to the identity matrix) is performed each $n$ successive iterations and the inverse Lagrangian Hessian approximation is not updated if the denominator of the updating formula is, in absolute value, lower than a specified tolerance.

# Quasi-Newton iterations

A quasi-Newton iteration is performed for each point in the elite population:

- a BFGS update formula to approximate the inverse Lagrangian Hessian is used.
- The identity matrix is used whenever a new point enters the elite population.
- A point kept in the elite population performs a sequence of quasi-Newton iterations (independently of the fitness).
- a line search strategy is used together with an Armijo like rule in order to globalize the algorithm.
- A reset strategy (setting the approximation to the identity matrix) is performed each $n$ successive iterations and the inverse Lagrangian Hessian approximation is not updated if the denominator of the updating formula is, in absolute value, lower than a specified tolerance.

# Quasi-Newton iterations

A quasi-Newton iteration is performed for each point in the elite population:

- a BFGS update formula to approximate the inverse Lagrangian Hessian is used.
- The identity matrix is used whenever a new point enters the elite population.
- A point kept in the elite population performs a sequence of quasi-Newton iterations (independently of the fitness).
- a line search strategy is used together with an Armijo like rule in order to globalize the algorithm.
- A reset strategy (setting the approximation to the identity matrix) is performed each $n$ successive iterations and the inverse Lagrangian Hessian approximation is not updated if the denominator of the updating formula is, in absolute value, lower than a specified tolerance.

# Quasi-Newton iterations

A quasi-Newton iteration is performed for each point in the elite population:

- a BFGS update formula to approximate the inverse Lagrangian Hessian is used.
- The identity matrix is used whenever a new point enters the elite population.
- A point kept in the elite population performs a sequence of quasi-Newton iterations (independently of the fitness).
- a line search strategy is used together with an Armijo like rule in order to globalize the algorithm.
- A reset strategy (setting the approximation to the identity matrix) is performed each $n$ successive iterations and the inverse Lagrangian Hessian approximation is not updated if the denominator of the updating formula is, in absolute value, lower than a specified tolerance.

# The algorithm

The GA can be described as follows

## MLOCGAMO

1. Randomly initialize the initial population $\mathcal{E}^0$. Set $k = 0$ as the iteration counter.
2. While the number of iterations and objective function evaluations is below the given maxima then
   - Compute the set of parents $\mathcal{P}^k$ from the elite population $\mathcal{E}^k$ using a *fitness* function (selection).
   - Compute the Offspring $\mathcal{O}^k$ from the parent population $\mathcal{P}^k$ using the *crossover* and *mutation* operators.
   - Apply a quasi-Newton iteration for all the points in the elite set $\mathcal{E}^k$.
   - Compute the new elite set $\mathcal{E}^{k+1}$ by selecting points from $\mathcal{E}^k \bigcup \mathcal{O}^k$ with the best *fitness*. For points with the same *fitness* value the ones with lower objective function value are selected.
   - Set $k = k + 1$.

# The algorithm

The GA can be described as follows

**MLOCGAMO**

1. Randomly initialize the initial population $\mathcal{E}^0$. Set $k = 0$ as the iteration counter.
2. While the number of iterations and objective function evaluations is below the given maxima then
   - Compute the set of parents $\mathcal{P}^k$ from the elite population $\mathcal{E}^k$ using a *fitness* function (selection);
   - Compute the Offspring $\mathcal{O}^k$ from the parent population $\mathcal{P}^k$ using the *crossover* and *mutation* operators;
   - Apply a quasi-Newton iteration for all the points in the elite set $\mathcal{E}^k$.
   - Compute the new elite set $\mathcal{E}^{k+1}$ by selecting points from $\mathcal{E}^k \bigcup \mathcal{O}^k$ with the best *fitness*. For points with the same *fitness* value the ones with lower objective function value are selected.
   - Set $k = k + 1$

# The algorithm

The GA can be described as follows

**MLOCGAMO**

1. Randomly initialize the initial population $\mathcal{E}^0$. Set $k = 0$ as the iteration counter.
2. While the number of iterations and objective function evaluations is below the given maxima then
   - Compute the set of parents $\mathcal{P}^k$ from the elite population $\mathcal{E}^k$ using a *fitness* function (selection);
   - Compute the Offspring $\mathcal{O}^k$ from the parent population $\mathcal{P}^k$ using the *crossover* and *mutation* operators;
   - Apply a quasi-Newton iteration for all the points in the elite set $\mathcal{E}^k$.
   - Compute the new elite set $\mathcal{E}^{k+1}$ by selecting points from $\mathcal{E}^k \bigcup \mathcal{O}^k$ with the best *fitness*. For points with the same *fitness* value the ones with lower objective function value are selected.
   - Set $k = k + 1$.

# The algorithm

The GA can be described as follows

## MLOCGAMO

1. Randomly initialize the initial population $\mathcal{E}^0$. Set $k = 0$ as the iteration counter.
2. While the number of iterations and objective function evaluations is below the given maxima then
   - Compute the set of parents $\mathcal{P}^k$ from the elite population $\mathcal{E}^k$ using a *fitness* function (selection);
   - Compute the Offspring $\mathcal{O}^k$ from the parent population $\mathcal{P}^k$ using the *crossover* and *mutation* operators;
   - Apply a quasi-Newton iteration for all the points in the elite set $\mathcal{E}^k$.
   - Compute the new elite set $\mathcal{E}^{k+1}$ by selecting points from $\mathcal{E}^k \bigcup \mathcal{O}^k$ with the best *fitness*. For points with the same *fitness* value the ones with lower objective function value are selected.
   - Set $k = k + 1$.

# The algorithm

The GA can be described as follows

**MLOCGAMO**

1. Randomly initialize the initial population $\mathcal{E}^0$. Set $k = 0$ as the iteration counter.
2. While the number of iterations and objective function evaluations is below the given maxima then
   - Compute the set of parents $\mathcal{P}^k$ from the elite population $\mathcal{E}^k$ using a *fitness* function (selection);
   - Compute the Offspring $\mathcal{O}^k$ from the parent population $\mathcal{P}^k$ using the *crossover* and *mutation* operators;
   - Apply a quasi-Newton iteration for all the points in the elite set $\mathcal{E}^k$.
   - Compute the new elite set $\mathcal{E}^{k+1}$ by selecting points from $\mathcal{E}^k \bigcup \mathcal{O}^k$ with the best *fitness*. For points with the same *fitness* value the ones with lower objective function value are selected.
   - Set $k = k + 1$.

# The algorithm

The GA can be described as follows

**MLOCGAMO**

1. Randomly initialize the initial population $\mathcal{E}^0$. Set $k = 0$ as the iteration counter.
2. While the number of iterations and objective function evaluations is below the given maxima then
   - Compute the set of parents $\mathcal{P}^k$ from the elite population $\mathcal{E}^k$ using a *fitness* function (selection);
   - Compute the Offspring $\mathcal{O}^k$ from the parent population $\mathcal{P}^k$ using the *crossover* and *mutation* operators;
   - Apply a quasi-Newton iteration for all the points in the elite set $\mathcal{E}^k$.
   - Compute the new elite set $\mathcal{E}^{k+1}$ by selecting points from $\mathcal{E}^k \bigcup \mathcal{O}^k$ with the best *fitness*. For points with the same *fitness* value the ones with lower objective function value are selected.
   - Set $k = k + 1$.

# The algorithm

The GA can be described as follows

**MLOCGAMO**

1. Randomly initialize the initial population $\mathcal{E}^0$. Set $k = 0$ as the iteration counter.
2. While the number of iterations and objective function evaluations is below the given maxima then
   - Compute the set of parents $\mathcal{P}^k$ from the elite population $\mathcal{E}^k$ using a *fitness* function (selection);
   - Compute the Offspring $\mathcal{O}^k$ from the parent population $\mathcal{P}^k$ using the *crossover* and *mutation* operators;
   - Apply a quasi-Newton iteration for all the points in the elite set $\mathcal{E}^k$.
   - Compute the new elite set $\mathcal{E}^{k+1}$ by selecting points from $\mathcal{E}^k \bigcup \mathcal{O}^k$ with the best *fitness*. For points with the same *fitness* value the ones with lower objective function value are selected.
   - Set $k = k + 1$.

# The algorithm

The GA can be described as follows

**MLOCGAMO**

1. Randomly initialize the initial population $\mathcal{E}^0$. Set $k = 0$ as the iteration counter.
2. While the number of iterations and objective function evaluations is below the given maxima then
   - Compute the set of parents $\mathcal{P}^k$ from the elite population $\mathcal{E}^k$ using a *fitness* function (selection);
   - Compute the Offspring $\mathcal{O}^k$ from the parent population $\mathcal{P}^k$ using the *crossover* and *mutation* operators;
   - Apply a quasi-Newton iteration for all the points in the elite set $\mathcal{E}^k$.
   - Compute the new elite set $\mathcal{E}^{k+1}$ by selecting points from $\mathcal{E}^k \bigcup \mathcal{O}^k$ with the best *fitness*. For points with the same *fitness* value the ones with lower objective function value are selected.
   - Set $k = k + 1$.

# Outline

# Definition

A semi-infinite programming problem can be described in the following form:

## SIP

$$\begin{aligned} \min_{y \in R^m} \quad & g(y), \\ s.t. \quad & h(y,x) \leq 0 \\ & \forall x \in \Omega \subset R^n. \end{aligned}$$

## Why semi-infinite?

These problems are characterized to have a finite number of variables ($m$) to be determined subject to an infinite number of constraints (recall that $\Omega$ is an infinite set).

# Definition

A semi-infinite programming problem can be described in the following form:

### SIP

$$
\begin{array}{rl}
\displaystyle\min_{y \in R^m} & g(y), \\
s.t. & h(y, x) \leq 0 \\
 & \forall x \in \Omega \subset R^n.
\end{array}
$$

### Why semi-infinite?

These problems are characterized to have a finite number of variables ($m$) to be determined subject to an infinite number of constraints (recall that $\Omega$ is an infinite set).

# Multi-local

## Where multi-local plays a role?

One of the major difficulties is to deal with the infinite number of constraints. A simple check of the $y^*$ KKT conditions for optimality requests the computation of all the global optima (the set of global optima is the set of active constraints) for the problem

$$\min_{x \in \Omega} f(x) \equiv -h(y^*, x). \tag{3}$$

## For $\bar{y}$ to be feasible:

$$h(\bar{y}, x) \leq 0$$
$$\forall x \in \Omega$$

# Multi-local

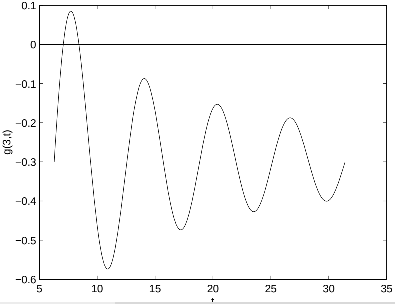
## Where multi-local plays a role?

One of the major difficulties is to deal with the infinite number of constraints. A simple check of the $y^*$ KKT conditions for optimality requests the computation of all the global optima (the set of global optima is the set of active constraints) for the problem

$$\min_{x \in \Omega} f(x) \equiv -h(y^*, x). \quad (3)$$

## For $\bar{y}$ to be feasible:

$$h(\bar{y}, x) \leq 0$$
$$\forall x \in \Omega$$



# Outline

# Test problems

A set of test SIP problems were obtained from the SIPAMPL problem database where problem (3) was considered.

## Multi-local problems considered

| SIP problem | Test problem | $n$ |
|---|---|---|
| watson2 | sip_wat2 | 1 |
| vaz1 | sip_vaz1 | 2 |
| vaz3 | sip_vaz3 | 2 |
| priceS6 | sip_S6 | 6 |
| priceU | sip_U | 6 |

## Multi-local problems considered

Since problems were obtained from SIP by replacing $y$ by $y^*$ the global optima is attained at $f^* = 0$.

# Test problems

A set of test SIP problems were obtained from the SIPAMPL problem database where problem (3) was considered.

## Multi-local problems considered

| SIP problem | Test problem | $n$ |
|---|---|---|
| `watson2` | `sip_wat2` | 1 |
| `vaz1` | `sip_vaz1` | 2 |
| `vaz3` | `sip_vaz3` | 2 |
| `priceS6` | `sip_S6` | 6 |
| `priceU` | `sip_U` | 6 |

## Multi-local problems considered

Since problems were obtained from SIP by replacing $y$ by $y^*$ the global optima is attained at $f^* = 0$.

# Test problems

## Additional problems

Other well known problems from global optimization were also considered.

## Some details

- 4000 function evaluations allowed (since we have a population algorithm this limit can be slightly exceeded).
- Since we are dealing with a stochastic algorithm we are reporting average values for 10 runs.
- We are considering that a global optima was obtained when the best obtained point is a Lagrangian stationary point.
- A local optima is obtained if it is a stationary point of the Lagrangian and its objective function value is far away from the best obtained point.

# Test problems

## Additional problems

Other well known problems from global optimization were also considered.

## Some details

- 4000 function evaluations allowed (since we have a population algorithm this limit can be slightly exceeded).
- Since we are dealing with a stochastic algorithm we are reporting average values for 10 runs.
- We are considering that a global optima was obtained when the best obtained point is a Lagrangian stationary point.
- A local optima is obtained if it is a stationary point of the Lagrangian and its objective function value is far away from the best obtained point.

# Test problems

## Additional problems

Other well known problems from global optimization were also considered.

## Some details

- 4000 function evaluations allowed (since we have a population algorithm this limit can be slightly exceeded).
- Since we are dealing with a stochastic algorithm we are reporting average values for 10 runs.
- We are considering that a global optima was obtained when the best obtained point is a Lagrangian stationary point.
- A local optima is obtained if it is a stationary point of the Lagrangian and its objective function value is far away from the best obtained point.

# Test problems

## Additional problems

Other well known problems from global optimization were also considered.

## Some details

- 4000 function evaluations allowed (since we have a population algorithm this limit can be slightly exceeded).
- Since we are dealing with a stochastic algorithm we are reporting average values for 10 runs.
- We are considering that a global optima was obtained when the best obtained point is a Lagrangian stationary point.
- A local optima is obtained if it is a stationary point of the Lagrangian and its objective function value is far away from the best obtained point.

# Test problems

## Additional problems

Other well known problems from global optimization were also considered.

## Some details

- 4000 function evaluations allowed (since we have a population algorithm this limit can be slightly exceeded).
- Since we are dealing with a stochastic algorithm we are reporting average values for 10 runs.
- We are considering that a global optima was obtained when the best obtained point is a Lagrangian stationary point.
- A local optima is obtained if it is a stationary point of the Lagrangian and its objective function value is far away from the best obtained point.

# Test problems

## Additional problems

Other well known problems from global optimization were also considered.

## Some details

- 4000 function evaluations allowed (since we have a population algorithm this limit can be slightly exceeded).
- Since we are dealing with a stochastic algorithm we are reporting average values for 10 runs.
- We are considering that a global optima was obtained when the best obtained point is a Lagrangian stationary point.
- A local optima is obtained if it is a stationary point of the Lagrangian and its objective function value is far away from the best obtained point.

# Numerical results - Version 1

| Test functions | $n$ | $N_{x^*}$ | $f^*$ | $f_{mean}(f_{sd})$ | $N_{glob}$ | $N_{loc}$ |
|---|---|---|---|---|---|---|
| b2 | 2 | 1 | 0 | 5.71E-14 ( 1.81E-13 ) | 1 | 1 |
| bohachevsky | 2 | 1 | 0 | 5.72E-14 ( 1.81E-13 ) | 1 | 1 |
| branin | 2 | 3 | 3.98E-01 | 3.98E-01 ( 5.85E-17 ) | 2.9 | 2.9 |
| dejoung | 3 | 1 | 0 | 0.00E+00 ( 0.00E+00 ) | 1 | 1 |
| easom | 2 | 1 | -1 | -1.00E+00 ( 0.00E+00 ) | 1 | 58.2 |
| f1 | 30 | 1 | -1.26E+04 | -1.43E+04 ( 1.04E+01 ) | 0.6 | 1 |
| goldprice | 2 | 1 | 3 | 3.00E+00 ( 5.11E-04 ) | 0 | 0 |
| griewank | 6 | 1 | 0 | 2.39E-02 ( 8.69E-03 ) | 1.7 | 23.2 |
| hartmann3 | 3 | 1 | -3.86E+00 | -3.86E+00 ( 9.70E-04 ) | 0 | 1.7 |
| hartmann6 | 6 | 1 | -3.32E+00 | -3.32E+00 ( 9.72E-07 ) | 0.4 | 0.9 |
| hump | 2 | 2 | 0 | 4.65E-08 ( 8.59E-13 ) | 1 | 1.2 |
| hump_camel | 2 | 2 | -1.0316285 | -1.03E+00 ( 2.34E-16 ) | 2 | 6 |
| levy3 | 2 | 18 | -1.77E+02 | -1.77E+02 ( 0.00E+00 ) | 3.8 | 11.9 |
| parsopoulos | 2 | 12 | 0 | 2.61E-04 ( 4.19E-04 ) | 0.6 | 7 |
| rosenbrock10 | 10 | 1 | 0 | 1.96E+02 ( 2.58E+02 ) | 0 | 0 |
| rosenbrock2 | 2 | 1 | 0 | 3.65E-02 ( 7.33E-02 ) | 0 | 0 |
| rosenbrock5 | 5 | 1 | 0 | 4.74E-01 ( 1.59E-01 ) | 0 | 0 |
| shekel10 | 4 | 1 | -1.05E+01 | -7.85E+00 ( 2.84E+00 ) | 1 | 4.9 |
| shekel5 | 4 | 1 | -1.02E+01 | -9.65E+00 ( 1.60E+00 ) | 1 | 3.2 |

# Numerical results - Version 1 (Cont.)

| Test functions | $n$ | $N_{x^*}$ | $f^*$ | $f_{mean}(f_{sd})$ | $N_{glob}$ | $N_{loc}$ |
|---|---|---|---|---|---|---|
| shekel7 | 4 | 1 | -1.04E+01 | -7.76E+00 ( 2.78E+00 ) | 1 | 4.2 |
| shubert | 2 | 18 | -1.87E+02 | -1.87E+02 ( 0.00E+00 ) | 8 | 15.9 |
| storn1 | 2 | 2 | -4.08E-01 | -4.07E-01 ( 0.00E+00 ) | 2 | 3 |
| storn2 | 2 | 2 | -1.81E+01 | -1.81E+01 ( 3.74E-15 ) | 2 | 2.2 |
| storn3 | 2 | 2 | -2.28E+02 | -2.28E+02 ( 6.39E-02 ) | 0 | 0 |
| storn4 | 2 | 2 | -2.43E+03 | -2.43E+03 ( 8.68E-03 ) | 0 | 0 |
| storn5 | 2 | 2 | -2.48E+04 | -2.48E+04 ( 1.85E+01 ) | 0 | 0 |
| storn6 | 2 | 2 | -2.49E+05 | -2.48E+04 ( 6.44E+00 ) | 0 | 0 |
| zakharov10 | 10 | 1 | 0 | 5.38E-11 ( 1.11E-10 ) | 1 | 1 |
| zakharov2 | 2 | 1 | 0 | 2.65E-12 ( 2.47E-12 ) | 1 | 1 |
| zakharov20 | 20 | 1 | 0 | 5.00E-02 ( 1.19E-01 ) | 0.1 | 0.1 |
| zakharov4 | 4 | 1 | 0 | 3.30E-12 ( 3.99E-12 ) | 1 | 1 |
| zakharov5 | 5 | 1 | 0 | 1.02E-12 ( 6.34E-13 ) | 1 | 1 |
| spherical | 6 | 2 | 1 | -3.00E+05 ( 0.00E+00 ) | 40 | 40 |
| sip_S6 | 6 | 1 | 0 | -9.25E+00 ( 0.00E+00 ) | 1 | 15.8 |
| sip_U | 6 | 1 | 0 | -8.00E+00 ( 9.53E-05 ) | 1.2 | 4.1 |
| sip_vaz1 | 2 | 1 | 0 | 0.00E+00 ( 0.00E+00 ) | 74 | 74 |
| sip_vaz3 | 2 | 1 | 0 | 0.00E+00 ( 0.00E+00 ) | 69 | 69 |
| sip_wat2 | 1 | 1 | 0 | -1.11E-01 ( 1.46E-17 ) | 1 | 2 |

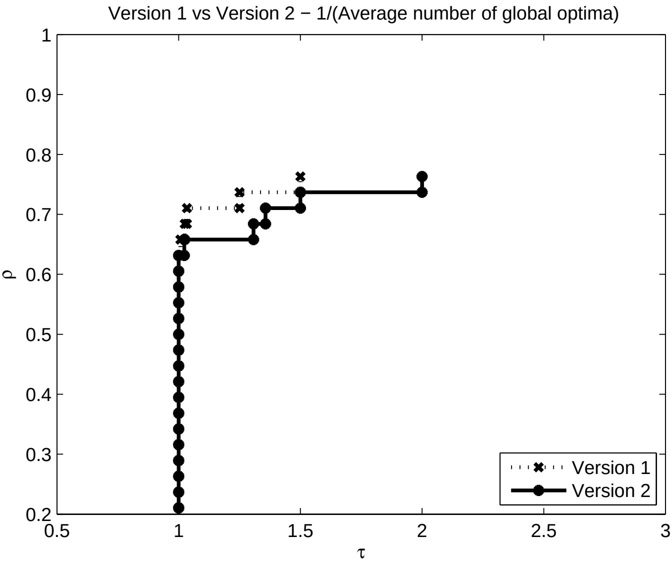
# Numerical results - Version 2

| Test functions | $n$ | $N_{x^*}$ | $f^*$ | $f_{mean}(f_{sd})$ | $N_{glob}$ | $N_{loc}$ |
|---|---|---|---|---|---|---|
| b2 | 2 | 1 | 0 | 1.55E-14 ( 4.91E-14 ) | 1 | 1 |
| bohachevsky | 2 | 1 | 0 | -5.55E-17 ( 6.50E-33 ) | 1 | 1 |
| branin | 2 | 3 | 3.98E-01 | 3.98E-01 ( 5.85E-17 ) | 3 | 3 |
| dejoung | 3 | 1 | 0 | 0.00E+00 ( 0.00E+00 ) | 1 | 1 |
| easom | 2 | 1 | -1 | -1.00E+00 ( 0.00E+00 ) | 1 | 58.4 |
| f1 | 30 | 1 | -1.26E+04 | -1.43E+04 ( 4.01E+01 ) | 0.4 | 0.5 |
| goldprice | 2 | 1 | 3 | 3.00E+00 ( 1.58E-04 ) | 0 | 0 |
| griewank | 6 | 1 | 0 | 1.70E-02 ( 1.27E-02 ) | 1.3 | 21.9 |
| hartmann3 | 3 | 1 | -3.86E+00 | -3.86E+00 ( 2.42E-03 ) | 0.1 | 2.1 |
| hartmann6 | 6 | 1 | -3.32E+00 | -3.32E+00 ( 1.27E-06 ) | 0.2 | 1.3 |
| hump | 2 | 2 | 0 | 4.65E-08 ( 1.65E-12 ) | 1 | 1.1 |
| hump_camel | 2 | 2 | -1.0316285 | -1.03E+00 ( 2.34E-16 ) | 2 | 6 |
| levy3 | 2 | 18 | -1.77E+02 | -1.77E+02 ( 0.00E+00 ) | 2.8 | 12 |
| parsopoulos | 2 | 12 | 0 | 1.14E-04 ( 2.01E-04 ) | 0.9 | 7.3 |
| rosenbrock10 | 10 | 1 | 0 | 3.22E+02 ( 3.95E+02 ) | 0 | 0 |
| rosenbrock2 | 2 | 1 | 0 | 8.74E-02 ( 8.74E-02 ) | 0 | 0 |
| rosenbrock5 | 5 | 1 | 0 | 4.41E-01 ( 2.75E-01 ) | 0 | 0 |
| shekel10 | 4 | 1 | -1.05E+01 | -8.39E+00 ( 2.77E+00 ) | 1 | 5 |
| shekel5 | 4 | 1 | -1.02E+01 | -1.02E+01 ( 0.00E+00 ) | 1 | 3.7 |

# Numerical results - Version 2 (cont.)

| Test functions | $n$ | $N_{x^*}$ | $f^*$ | $f_{mean}(f_{sd})$ | $N_{glob}$ | $N_{loc}$ |
|---|---|---|---|---|---|---|
| shekel7 | 4 | 1 | -1.04E+01 | -9.35E+00 ( 2.22E+00 ) | 1 | 4.2 |
| shubert | 2 | 18 | -1.87E+02 | -1.87E+02 ( 0.00E+00 ) | 8.2 | 16.7 |
| storn1 | 2 | 2 | -4.08E-01 | -4.07E-01 ( 0.00E+00 ) | 2 | 3 |
| storn2 | 2 | 2 | -1.81E+01 | -1.81E+01 ( 3.74E-15 ) | 2 | 2.4 |
| storn3 | 2 | 2 | -2.28E+02 | -2.28E+02 ( 1.14E-01 ) | 0 | 0 |
| storn4 | 2 | 2 | -2.43E+03 | -2.43E+03 ( 9.40E-02 ) | 0 | 0 |
| storn5 | 2 | 2 | -2.48E+04 | -2.48E+04 ( 7.63E+00 ) | 0 | 0 |
| storn6 | 2 | 2 | -2.49E+05 | -2.48E+04 ( 8.09E+00 ) | 0 | 0 |
| zakharov10 | 10 | 1 | 0 | 8.07E-11 ( 1.41E-10 ) | 1 | 1 |
| zakharov2 | 2 | 1 | 0 | 1.86E-12 ( 1.78E-12 ) | 1 | 1 |
| zakharov20 | 20 | 1 | 0 | 4.35E-02 ( 5.63E-02 ) | 0 | 0 |
| zakharov4 | 4 | 1 | 0 | 3.29E-11 ( 8.68E-11 ) | 1 | 1 |
| zakharov5 | 5 | 1 | 0 | 9.71E-13 ( 1.67E-12 ) | 1 | 1 |
| spherical | 6 | 2 | 1 | -3.00E+05 ( 0.00E+00 ) | 40 | 40 |
| sip_S6 | 6 | 1 | 0 | -9.25E+00 ( 0.00E+00 ) | 1 | 15.7 |
| sip_U | 6 | 1 | 0 | -8.00E+00 ( 3.91E-04 ) | 1.5 | 4 |
| sip_vaz1 | 2 | 1 | 0 | 0.00E+00 ( 0.00E+00 ) | 72.3 | 72.3 |
| sip_vaz3 | 2 | 1 | 0 | 0.00E+00 ( 0.00E+00 ) | 69.5 | 69.6 |
| sip_wat2 | 1 | 1 | 0 | -1.11E-01 ( 1.46E-17 ) | 1 | 2.1 |

# Version 1 *versus* Version 2



# Outline

# Conclusions

- We propose an algorithm for multi-local optimization (with two versions).
- The proposed algorithm uses a neighborhood concept for the *fitness* function. One version considered the dominance concept from multi-objective optimization.
- The algorithms use a quasi-Newton step in order to accelerate the convergence to local optima.
- We provide numerical results for the implemented algorithm and a compare between the two versions.
- Version 2 (with the dominance concept from multi-objective) proved to be (slightly) less efficient than version 1.

# Conclusions

- We propose an algorithm for multi-local optimization (with two versions).
- The proposed algorithm uses a neighborhood concept for the *fitness* function. One version considered the dominance concept from multi-objective optimization.
- The algorithms use a quasi-Newton step in order to accelerate the convergence to local optima.
- We provide numerical results for the implemented algorithm and a compare between the two versions.
- Version 2 (with the dominance concept from multi-objective) proved to be (slightly) less efficient than version 1.

# Conclusions

- We propose an algorithm for multi-local optimization (with two versions).
- The proposed algorithm uses a neighborhood concept for the *fitness* function. One version considered the dominance concept from multi-objective optimization.
- The algorithms use a quasi-Newton step in order to accelerate the convergence to local optima.
- We provide numerical results for the implemented algorithm and a compare between the two versions.
- Version 2 (with the dominance concept from multi-objective) proved to be (slightly) less efficient than version 1.

# Conclusions

- We propose an algorithm for multi-local optimization (with two versions).
- The proposed algorithm uses a neighborhood concept for the *fitness* function. One version considered the dominance concept from multi-objective optimization.
- The algorithms use a quasi-Newton step in order to accelerate the convergence to local optima.
- We provide numerical results for the implemented algorithm and a compare between the two versions.
- Version 2 (with the dominance concept from multi-objective) proved to be (slightly) less efficient than version 1.

# Conclusions

- We propose an algorithm for multi-local optimization (with two versions).
- The proposed algorithm uses a neighborhood concept for the *fitness* function. One version considered the dominance concept from multi-objective optimization.
- The algorithms use a quasi-Newton step in order to accelerate the convergence to local optima.
- We provide numerical results for the implemented algorithm and a compare between the two versions.
- Version 2 (with the dominance concept from multi-objective) proved to be (slightly) less efficient than version 1.

# END

email: aivaz@dps.uminho.pt
Web http://www.norg.uminho.pt/aivaz

email: lac@dps.uminho.pt
Web http://pessoais.dps.uminho.pt/lac

Supported by
FCT Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E ENSINO SUPERIOR Portugal


EURO XXIV LISBON
24th European Conference on Operational Research
July 11-14 www.euro2010lisbon.org
Welcome!