Heuristics for Continuous Optimisation

Design, Replicability & Behaviour

Dr Fabio Caraffini



Zienkiewicz Institute for Modelling, Data and AI

✉ **fabio.caraffini@swansea.ac.uk**

www.swansea.ac.uk/staff/fabio.caraffini/



Summer School on *Data Science, Learning and Optimization* - Norcia (Italy), 23-27 June 2025

Optimisation Problems: a generalised formulation

Any time we pick a decision/make a choice, we face an optimisation problem

$$
\begin{array}{lcl}
\textit{Maximise/Minimise} & f_m(\mathbf{x}) & m = 1, 2, \ldots, M \\
\textit{subject} - \textit{to} & g_j(\mathbf{x}) \leqslant 0 & j = 1, 2, \ldots, J \\
& h_k(\mathbf{x}) = 0 & k = 1, 2, \ldots, K \\
& x_i^L \leqslant x_i \leqslant x_i^U & i = 1, 2, \ldots, n
\end{array}
$$

Regardless of their nature, all optimisation problems share the same mathematical formulation, see [Caraffini, 2016] for details.

Optimisation Problems: a generalised formulation

Any time we pick a decision/make a choice, we face an optimisation problem

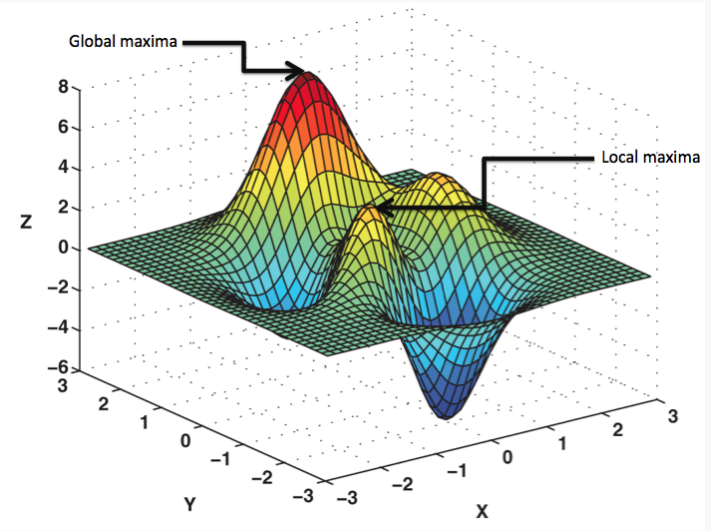
$$\begin{array}{lcl} \textit{Maximise/Minimise} & f_m(\mathbf{x}) & m = 1, 2, \ldots, M \\ \textit{subject} - \textit{to} & g_j(\mathbf{x}) \leqslant 0 & j = 1, 2, \ldots, J \\ & h_k(\mathbf{x}) = 0 & k = 1, 2, \ldots, K \\ & x_i^L \leqslant x_i \leqslant x_i^U & i = 1, 2, \ldots, n \end{array}$$

We will focus on real-valued, single-objective and box constrained optimisation:

$$\mathbf{x}^* \equiv \arg\{\min_{\mathbf{x} \in \mathbf{D}} f(\mathbf{x})\} = \{\mathbf{x}^* \in \mathbf{D} \,|\, f(\mathbf{x}^*) \leqslant f(\mathbf{x}) \; \forall \, \mathbf{x} \in \mathbf{D}\}$$

The Mathematical Approach: exploiting the gradient info

There is a number of exact methods that:



return global maxima & minima 👍

if numerical, always converge 👍

may require $f(\mathbf{x}) = \ldots$ 👎

need assumptions (smoothness) 👎

are computationally expensive 👎

straggle on multimodal problems 👎

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

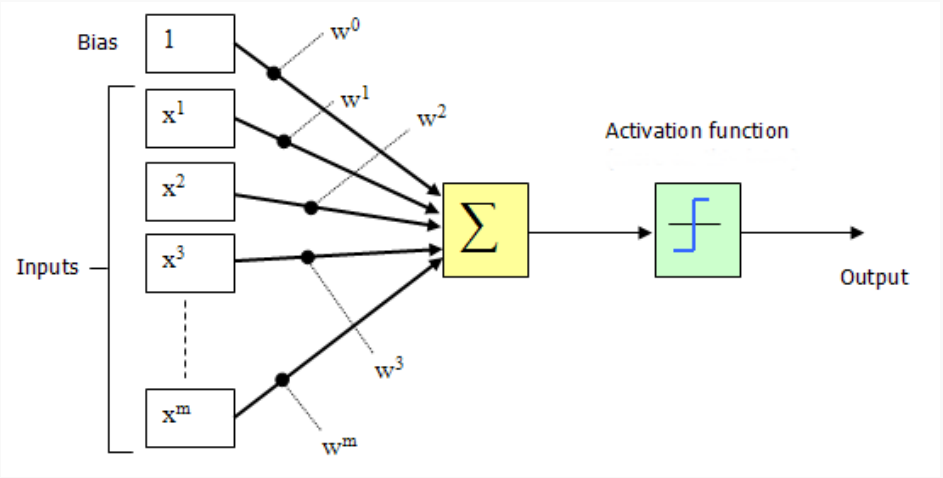
$$\begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \frac{\partial^2 f}{\partial x_3 \partial x_1} & \frac{\partial^2 f}{\partial x_3 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

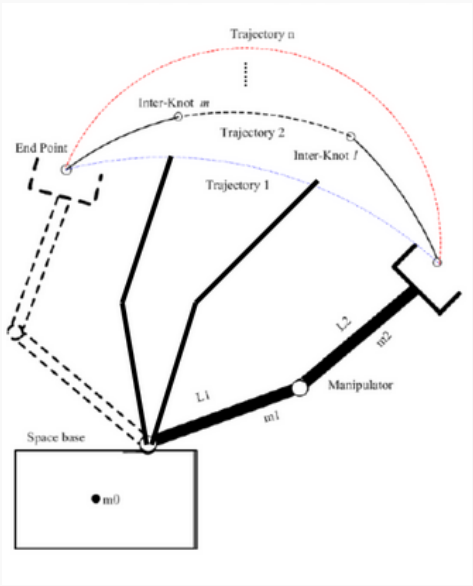
Features of real-world optimisation problems

- When possible the exact approach is preferable but:
  - in Engineering an analytical expression $f(x)$ is rarely available;
  - and it is rarely differentiable.

- Moreover, some features of the problem can be very challenging...

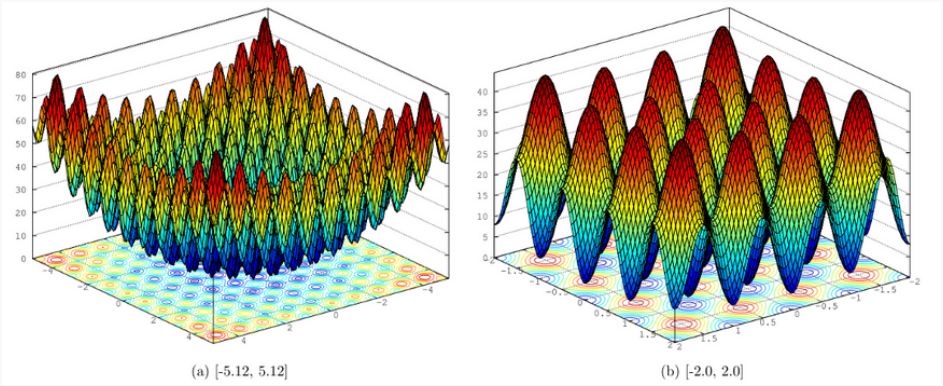
Hard real-life:
HIGH NON-LINEARITY.





- Usually optimisation problems are characterised by nonlinear function. **Optima are not on the bounds**!
- In real world optimisation problems, the physical phenomenon, due to its nature (e.g. in the case of saturation phenomenon or for systems which employ electronic components), cannot be approximated by a linear function neither in some areas of the decision space.

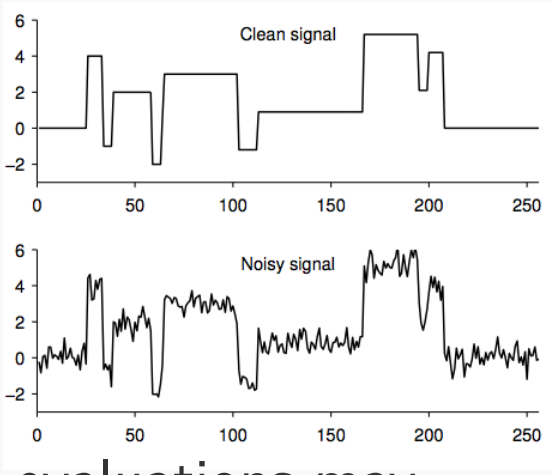
Hard real-life:
HIGH MULTI-MODALITY.


(a) [-5.12, 5.12] (b) [-2.0, 2.0]

- It often happens that the fitness landscape contains many local optima and that many of these have an unsatisfactory performance (fitness value).
- These fitness landscapes are usually rather difficult to be handled since **the optimisation algorithms which employ gradient** based information in detecting the search direction **could easily converge to a suboptimal basin of attraction**.[1]

[1] Basin of attraction: set of points of the decision space such that ,initial conditions chosen, dynamically evolve to a particular attractor.

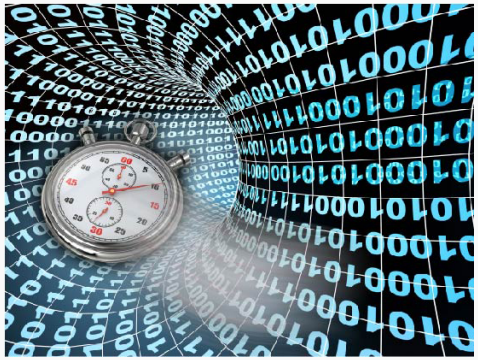
Hard real-life:
OPTIMISATION IN UNCERTAIN ENVIRONMENTS.



- **Noisy fitness function**: noise in fitness evaluations may come from many different sources such as sensory measurement errors or randomized simulations.
- **Approximated fitness function**: when the fitness function is very expensive to evaluate, or an analytical fitness function is not available, approximated fitness functions are often used instead. These approximated models implicitly introduce a noise which is the difference between the approximated value and real fitness value, which is unknown.
- **Robustness**: often, when a solution is implemented, the design variables or the environmental parameters are subject to perturbations or changes (e.g. control problems).

Hard real-life:
COMPUTATIONAL EXPENSIVE PROBLEMS.

Optimisation problems can be computationally expensive because of two reasons:



- large scale problems (needle in a haystack).
- computationally expensive fitness function (e.g. design of aircraft, control on an on-line electric drive).

Hard real-life:
MEMORY/TIME CONSTRAINTS

Many engineering problems are plagued by a modest hardware and stringent time constraints.



This can happen:

- due to cost limitations (e.g. vacuum cleaner robot);
- due to space limitations (e.g. use of minimalistic embedded systems as wearable devices, wireless sensors networks, etc.);
- in real-time systems (e.g. Telecommunication, Video-games, etc.).

Light (and simple) algorithms can be used in a modular way to tackle complex problems: if the hardware is limited an intelligent solution must be found!

Metaheuristic optimisation

*To overcome these issues, a logical choice of tools are **Evolutionary Algorithms**:*

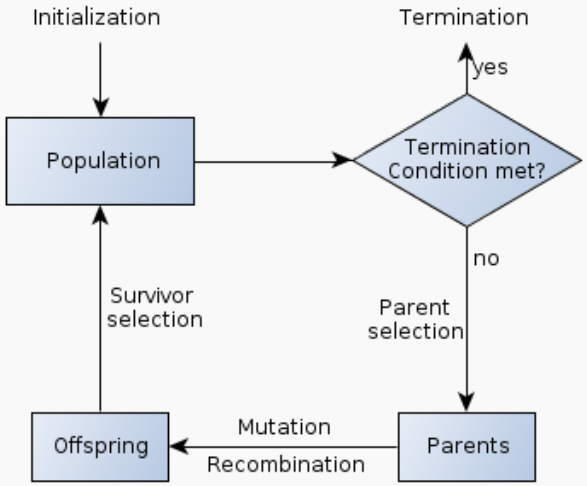
- **Fogel Owens (USA, 1964)**: *Evolutionary Programming (EP)*
  see [Fogel et al., 1964, Fogel et al., 1966];
- **Holland (USA, 1975)**: *Genetic Algorithm (GA)*
  see [Holland, 1975];
- **Rechenberg Schwefel (Germany, 1971)**: *Evolution Strategies (ES)*
  see [Rechenberg, 1971];
- **Koza (USA, 1990)**: *Genetic Programming (GP)*
  see [Koza, 1990, Koza, 1992b, Koza, 1992a];
- **Moscato (Argentina-USA 1989)**: *Memetic Algorithms (MA)*
  see [Moscato, 1989].

They all exploit the concept of Darwinian evolution to generate new solutions in the search space

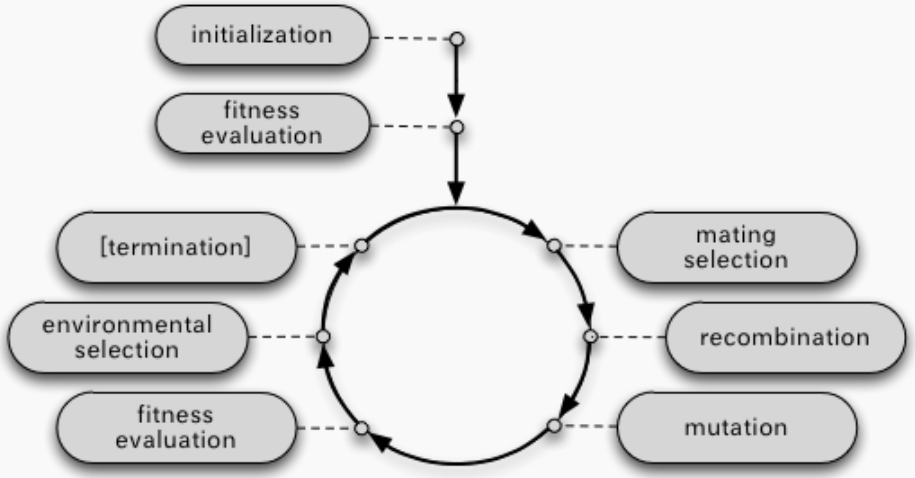
Evolutionary algorithms: general scheme.

Evolutionary Algorithms pseudo-code

INTITIALISE population with random candidate solutions
EVALUATE each solution
while *condition on budget* **do** ▷ or fitness value!
  SELECT parents ▷ 2 or multiple solutions
  RECOMBINE parents ▷ 2 or some of them
  MUTATE the resulting offspring
  EVALUATE new candidates
  SELECT individual for next generation
end while
Output best individual of the final population



Solutions cycle



Selection and variation operators cycle

Swarm intelligence

general metaphor:

- Some animals live and operate in groups;
- when these animals (e.g. swarms of birds, school of fish, ants, bees, monkeys, etc.) are all together can make things that would not be able to do when they are alone:
  - "collective intelligence".
- Swarm Intelligence algorithms are inspired by the collective behaviour, i.e. solutions share information with other close solutions (neighbourhood).

Swarm intelligence optimisation

generalities:

- population-based;
- each individual "move" thanks to a moving operator that makes use of the other individuals;
- One-to-one replacement (one-to-one spawning) right after the move.

Swarm Intelligence optimization

Initialisation ▷ Randomly sample initial swarm
while *Condition on budget* **do**
  for each $\mathbf{x} \in$ *Swarm* **do**
    Update moving parameters
    Perform move
    Evaluate $f(\mathbf{x})$ *and eventually perform replacement* ▷ one-to-one spawning
  end for
end while
Output *Best Individual*

References I



Caraffini, F. (2016).
Algorithmic issues in computational intelligence optimization: from design to implementation, from implementation to design.
Number 243 in Jyväskylä studies in computing. University of Jyväskylä.



Fogel, L. J., Owens, A., and Walsh, M. (1964).
On the evolution of artificial intelligence(artificial intelligence generated by natural evolution process).
In *National Symposium on Human Factors in Electronics, 5th, San Diego, California*, pages 63–76.



Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966).
Artificial Intelligence through Simulated Evolution.
John Wiley.



Holland, J. (1975).
Adaptation in Natural and Artificial Systems.
University of Michigan Press.

References II



Koza, J. R. (1990).
Concept formation and decision tree induction using the genetic programming paradigm.
In *Parallel Problem Solving from Nature*, pages 124–128. Springer.



Koza, J. R. (1992a).
Genetic programming: on the programming of computers by means of natural selection, volume 1.
MIT press.



Koza, J. R. (1992b).
Genetic Programming: vol. 1, On the programming of computers by means of natural selection, volume 1.
MIT press.



Moscato, P. (1989).
On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms.
Technical Report 826.

References III



Rechenberg, I. (1971).
Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.
PhD thesis, Technical University of Berlin.

General DE scheme

Differential Evolution pseudo-code

$g \leftarrow 1$ ▷ first generation
$\mathbf{Pop^g} \leftarrow$ *randomly sample M n-dimensional individuals within* $\mathbf{D}$
$\mathbf{x_{best}} \leftarrow$ *fittest individual* $\in \mathbf{Pop^g}$
while *Condition on budget* **do**
  for each $\mathbf{x_j} \in \mathbf{Pop^g}$ **do** ▷ $j = 0, 1, 2, ..., M$
    $\mathbf{x_m} \leftarrow$*Mutation* ▷ mutation first! (to create the mutant vector)
    $\mathbf{x_{off}} \leftarrow$*CrossOver*$\left(\mathbf{x_j}, \mathbf{x_m}\right)$ ▷ cross-over after (to generate an offspring)
    if $f\left(\mathbf{x_{off}}\right) \leq f\left(\mathbf{x_j}\right)$ **then** ▷ 1-to-1 spawning (survivor selection)
      $\mathbf{Pop^{g+1}}[j] \leftarrow \mathbf{x_{off}}$ ▷ N.B. we are not modifying $\mathbf{Pop^g}$!
    else
      $\mathbf{Pop^{g+1}}[j] \leftarrow \mathbf{x_j}$
    end if
  end for
  $g \leftarrow g + 1$ ▷ now we can get rid of the previous population
  $\mathbf{x_{best}} \leftarrow$ *fittest individual* $\in \mathbf{Pop^g}$ ▷ update best individual
end while
Output *Best Individual* $\mathbf{x_{best}}$

Individuals selection

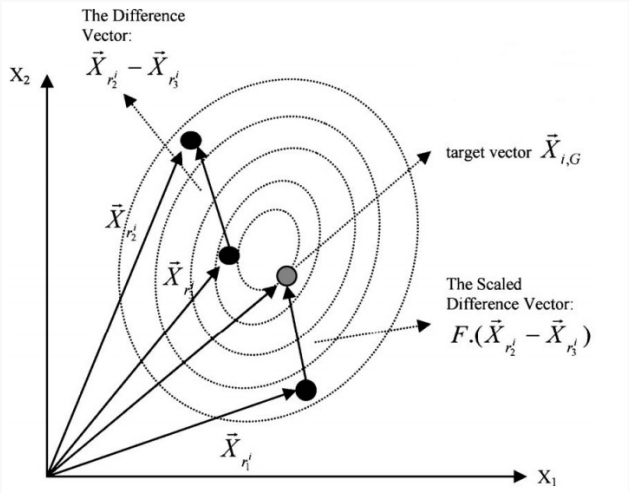
- individual are perturbed one at one;
- the mutation operator selects some individuals at random from the population by means of a uniformly distributed function to operate.
- Like in ES, there is no selection pressure for the choice of the parents undergoing variation operators (recombination and mutation).

Original mutation
(DE/rand/1)

- a mutant vector $\mathbf{x_m}$ is produced to cross-over with the current j^{th} point:
  - three points $\mathbf{x_{r_1}}$, $\mathbf{x_{r_2}}$ and $\mathbf{x_{r_3}}$ (with $r_3 \neq r_2 \neq r_1 \neq j$) are randomly picked up from the population and linearly combined

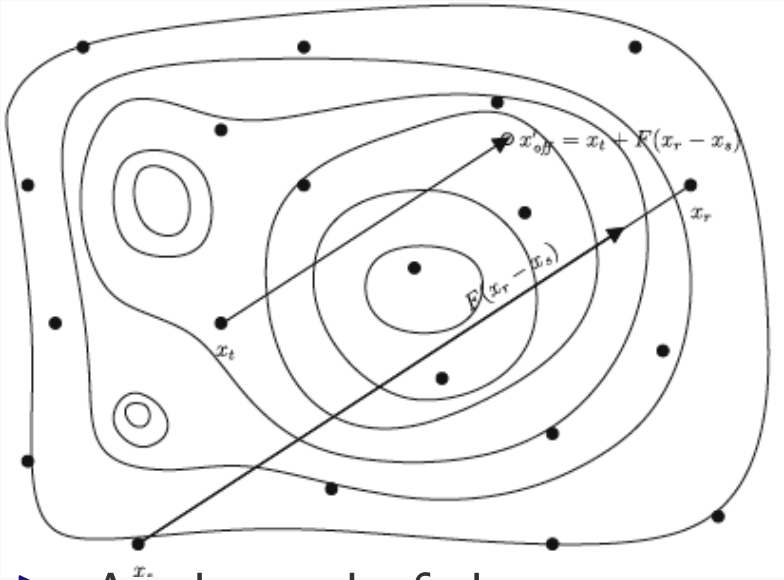
$$\mathbf{x_m} = \mathbf{x_{r_1}} + F\left(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}\right)$$

- the scale factor F has to be properly chosen in $[0, 2]$;
- $\mathbf{x_m}$ can be seen as the summation of two component:
  - $\mathbf{x_{r_1}}$, and a "difference vector" $\left(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}\right)$ moving it towards a new point.



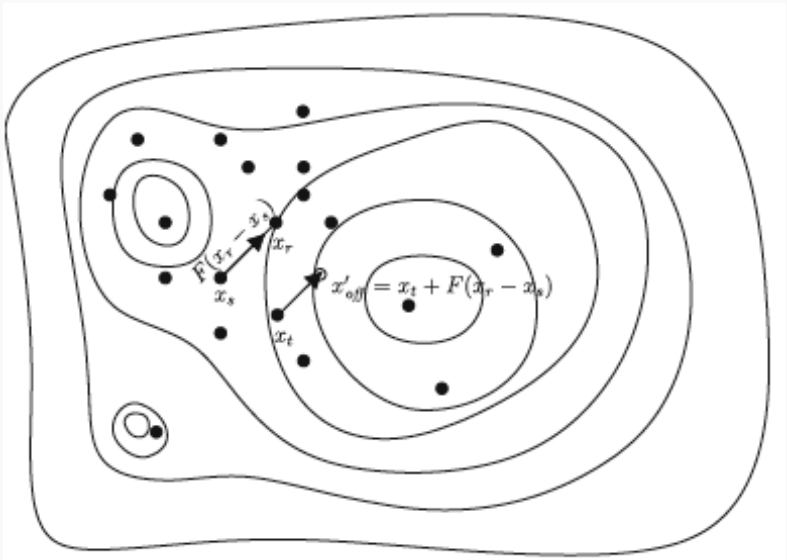
Original mutation

The idea:



- At the beginning of the optimization process the solutions are spread out in the decision space:
  - the difference vector is on average large! (*exploration*)

- At the end of the optimization process the solutions should be focused in an area of the decision space:
  - the search radius is on average small! (*exploitation*)



- DE has an implicit adaption: the "exploratory radius" progressively shrinks

Other classical mutation schemes
(to better handle different scenarios)

- best/1: $\mathbf{x_m} = \mathbf{x_{best}} + F\left(\mathbf{x_{r_1}} - \mathbf{x_{r_2}}\right)$
- rand/2: $\mathbf{x_m} = \mathbf{x_{r_1}} + F\left(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}\right) + F\left(\mathbf{x_{r_4}} - \mathbf{x_{r_5}}\right)$
- best/2: $\mathbf{x_m} = \mathbf{x_{best}} + F\left(\mathbf{x_{r_1}} - \mathbf{x_{r_2}}\right) + F\left(\mathbf{x}_{r_3} - \mathbf{x_{r_4}}\right)$
- current-to-best/1: $\mathbf{x_m} = \mathbf{x_j} + F\left(\mathbf{x_{best}} - \mathbf{x_j}\right) + F\left(\mathbf{x_{r_1}} - \mathbf{x_{r_2}}\right)$
- current-to-rand/1[a]: $\mathbf{x_m} = \mathbf{x_j} + K\left(\mathbf{x_{r_1}} - \mathbf{x_j}\right) + F^{'}\left(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}\right)$
- rand-to-best/1: $\mathbf{x_m} = \mathbf{x_{r_1}} + F\left(\mathbf{x_{best}} - \mathbf{x_j}\right) + F\left(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}\right)$
- rand-to-best/2: $\mathbf{x_m} = \mathbf{x_{r_1}} + F\left(\mathbf{x_{best}} - \mathbf{x_j}\right) + F\left(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}\right) + F\left(\mathbf{x_{r_4}} - \mathbf{x_{r_5}}\right)$

[a] $K = \mathcal{U}(0,1)$ and $F^{'} = K \cdot F$.

Crossover
Binomial and Exponential

- After mutation the current individual undergoes crossover with the mutant (i.e genes of $\mathbf{x_m}$ move to $\mathbf{x_j}$ to have an offspring);
- The most important options (equally popular) are:
  - binomial ($\forall x_i \in \mathbf{x_j} \exists$ CR[2] probability to crossover);
  - exponential (a burst is copied with exp-decreasing probability).

procedure BINOMIAL-XO($\mathbf{x_1}, \mathbf{x_2}$)
  Index $\leftarrow$ rand int $[1, n]$
  for $i = 1 : n$ **do**
    if $\mathcal{U}(0,1) \leq CR$ || $i ==$ *Index* **then**
      $\mathbf{x_{off}}[i] \leftarrow \mathbf{x_1}[i]$
    else
      $\mathbf{x_{off}}[i] \leftarrow \mathbf{x_2}[i]$
    end if
  end for
  Output $\mathbf{x_{off}}$
end procedure

procedure EXPONENTIAL-XO($\mathbf{x_1}, \mathbf{x_2}$)
  $\mathbf{x_{off}} \leftarrow \mathbf{x_2}$
  Index $\leftarrow$ rand int $[1, n]$
  $\mathbf{x_{off}}[$*Index*$] \leftarrow \mathbf{x_2}[$*Index*$]$
  $i \leftarrow$ *Index* $+ 1$
  while $\mathcal{U}(0,1) \leq CR$ || $i \neq$ *Index* **do**
    $\mathbf{x_{off}}[i] \leftarrow \mathbf{x_1}[i]$
    $i \leftarrow i + 1$
    if $i > n$ **then**
      $i \leftarrow 1$
    end if
  end while
  Output $\mathbf{x_{off}}$
end procedure

[2] CR: Crossover Rate.

Survivor selection

- When all the offspring population is generated for each individual, the 1-to-1 spawning is applied:
  - if the j^{th} offspring is fitter than the i^{th} individual, then replace it in the next population.
- Unlike PSO, this is done at the end of the generation:
  - generally produce similar results than instantaneous replacement,
  - and allow parallel implementations!

DE notation

- Some mutations are more exploitative (e.g. those moving towards a random solution) while other insist towards the direction of the current best point.
- To concisely describe a DE implementation the following notation is used: DE/x/y/z
  - x: the vector undergoing mutation (i.e. the one to which difference vectors are added);
  - y: the number of the employed difference vectors;
  - z: crossover strategy.

Exempli gratia:

DE/rand/2/bin, DE/ran/1/exp, DE/current-to-best/1/exp.
N.B. In DE/current-to-rand/1 crossover is redundant!
(It embeds a built-in XO)!

Success of DE

- DE is efficient and very easy to implement.
- DE is sensitive to only 3 parameters (F, CR, Population size)
  - however, performances depends on their tuning!
- DE can thus return decent results in many cases.
- DE application is clear and requires a very little knowledge in mathematics!

Parameters tuning

- Studies on how to tune DE's parameters have been performed in the past, e.g. [Liu and Lampinen, 2005] suggests:
  - $F \in [0.5, 1]$;
  - $CR \in [0.8, 1]$
  - population size $= 10 \times D$
    (arguable, see [Parsopoulos, 2009] and
    [Kononova et al., 2015], and too big for large scale problems!)
- An interesting relation has been found between the combinations of F and the corresponding CR and the population population variance is provided in [Zaharie, 2002].
- Other relevant and recommended articles on DE tuning and algorithmic, properties, and algorithmic behaviours are [Zaharie, 2009, Opara and Arabas, 2019b, Opara and Arabas, 2019a, Caraffini et al., 2019].

Generally, the most successfully DEs, are those tuning/adapting the parameters on the fly!

References I

Caraffini, F., Kononova, A. V., and Corne, D. (2019).
Infeasibility and structural bias in differential evolution.
Information Sciences, 496:161–179.

Kononova, A. V., Corne, D. W., Wilde, P. D., Shneer, V., and Caraffini, F. (2015).
Structural bias in population-based algorithms.
Information Sciences, 298(0):468 – 490.

Liu, J. and Lampinen, J. (2005).
A fuzzy adaptive differential evolution algorithm.
Soft Computing - A Fusion of Foundations, Methodologies and Applications, 9(6):448–462.

Opara, K. R. and Arabas, J. (2019a).
The contour fitting property of differential mutation.
Swarm and Evolutionary Computation, 50:100441.

Opara, K. R. and Arabas, J. (2019b).
Differential evolution: A survey of theoretical analyses.
Swarm and evolutionary computation, 44:546–558.

References II



Parsopoulos, K. E. (2009).
Cooperative micro-differential evolution for high-dimensional problems.
In *Proceedings of the conference on Genetic and evolutionary computation*, pages 531–538.



Zaharie, D. (2002).
Critical values for control parameters of differential evolution algorithm.
In Matuŝek, R. and Oŝmera, P., editors, *Proceedings of 8th International Mendel Conference on Soft Computing*, pages 62–67.



Zaharie, D. (2009).
Influence of crossover on the behavior of differential evolution algorithms.
Applied Soft Computing, 9(3):1126 – 1138.

Evolution Strategy (ES)

Generalities:

- *Developed*: Germany in the (late 1960's) early 1970 's by I. Rechenberg and H.-P. Schwefel [Schwefel, 1965][3] and [Rechenberg, 1971].
- *Attributed features*: meant for real-valued optimisation, relatively much theory, parameter-less (no need for tuning! ✓).
- *Special*: self-adaptation of (mutation) parameters standard; population is probabilistically described by a distribution, which is evolved to adapt to the fitness landscape.

Main idea:

The search for extrema is mainly mutation-driven through a multivariate Gaussian distribution $\mathcal{N}(\overline{\mathbf{x}}, \mathbf{C})$, $\overline{\mathbf{x}} \in \mathbb{R}$, $\mathbf{C} \in \mathbb{R}^2$. At each iteration variation operators first perturb/update strategy parameters (i.e. variances and angles for rotating the distribution).

[3] A version in English is given in [Schwefel, 1981].

ES: representation

- In the most general case an individual is encoded as: $\langle \mathbf{x}, \boldsymbol{\sigma}, \boldsymbol{\alpha} \rangle$
  - $\mathbf{x} = x_1, x_2, \ldots, x_n$: design variables;
  - $\boldsymbol{\sigma} = \sigma_1, \sigma_2, \ldots, \sigma_n$: step sizes vector (standard deviations[4]);
  - $\boldsymbol{\alpha} = \alpha_1, \alpha_2, \ldots, \alpha_{n\frac{n-1}{2}}$: angles vector (between two variables[5]).
- Not every component is always present!
  - simplest case: $\langle x_1, x_2, \ldots, x_n, \sigma \rangle$
  - in the general case the memory cost of $\mathbf{C}$ grows quadratically with n ✗

[4] Taken to the power of two, they form the main diagonal of $\mathbf{C}$.

[5] Needed to work out the cross-correlation between two variables in $\mathbf{C}$, see [Eiben and Smith, 2003].

ES: parent selection

Uniform random.

- Parents are selected by *uniform random distribution* whenever an operator needs one (or some)!
  - Thus, ES parent selection is unbiased: every individual has the same probability to be selected.

NOTE that in ES "parent" means a population member

no need for a mating pool, as in GA, where parents are population members selected to undergo variation.

ES: recombination
(discrete and intermediary recombination)

- Given two parents $\mathbf{x}$ and $\mathbf{y}$ a child $\mathbf{z}$ is generated according to
  - *discrete* recombination: $\mathbf{z}[i] = \mathbf{x}[i]$ or $\mathbf{y}[i]$ (fifty-fifty), $i = 1, 2, \ldots, n$
  - *intermediary* recombination: $\mathbf{z}[i] = \frac{\mathbf{x}[i] + \mathbf{y}[i]}{2}$, $i = 1, 2, \ldots, n$
- Both discrete and intermediary can follow a two-parent or multi-parent logic:
  - *local*: $\mathbf{x}$ and $\mathbf{y}$ are the same $\forall i = 1, 2, \ldots, n$;
  - *global*: $\mathbf{x}$ and $\mathbf{y}$ are randomly re-selected $\forall i = 1, 2, \ldots, n$.
- Thus, recombination plays a minor role but 4 combinations can be used:

	discrete	**intermediary**
local	local discrete recombination	local intermediary recombination
global	global discrete recombination	global intermediary recombination

ES: mutation

main idea

- Mutation is done via a Gaussian distribution centred in the offspring $\mathbf{x}$:

$$\mathbf{x}^{'} \sim \mathcal{N}(\mathbf{x}, \mathbf{C})$$

- it can be seen as adding "noise" to $\mathbf{x}$:

$$\mathbf{x}^{'} \leftarrow \mathbf{x} + \mathcal{N}(\emptyset, \mathbf{C})$$

- $\mathbf{C}$ has to be updated to adapt it to the search space before the sampling!
  - parameters are evolved⇒good ones go ahead together with the good generated solution, and used for the next updated! ✓
- $\mathbf{C}$ contains variances (worked out from $\boldsymbol{\sigma}$) and correlations (worked out from $\boldsymbol{\alpha}$) (click-here):

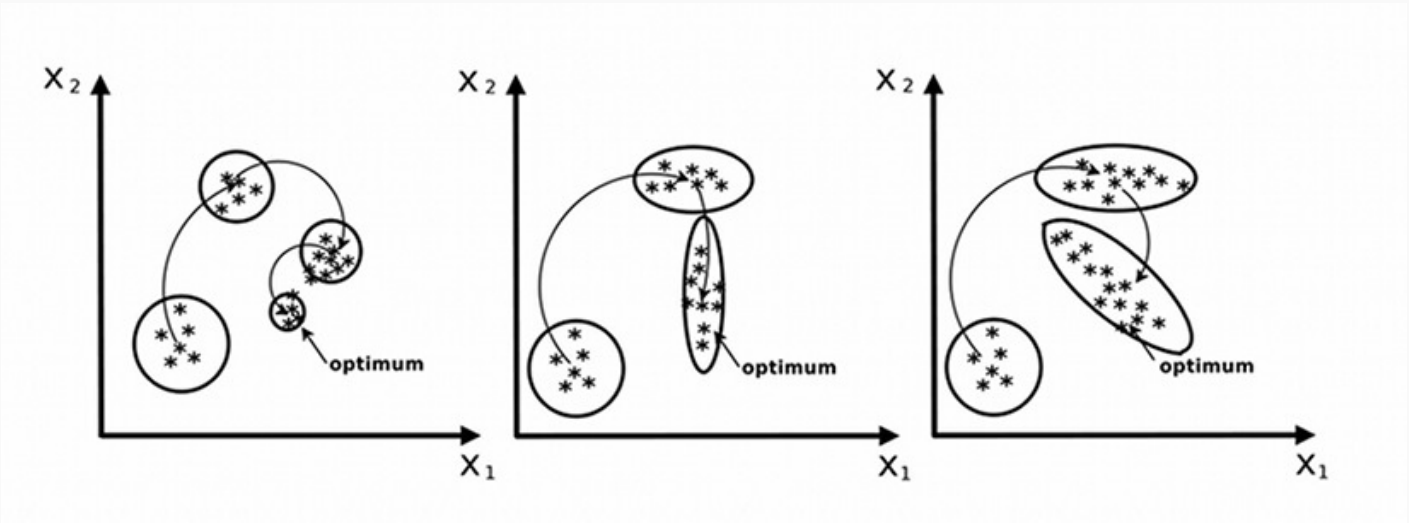
$$\mathbf{C} = [c_{i,j}] = \begin{bmatrix} \sigma^2_{x_1} & \sigma_{x_1 x_2} & \sigma_{x_1 x_3} & \cdots & \sigma_{x_1 x_n} \\ \sigma_{x_1 x_2} & \sigma^2_{x_2} & \sigma_{x_2 x_3} & \cdots & \sigma_{x_2 x_n} \\ \sigma_{x_1 x_3} & \sigma_{x_2 x_3} & \sigma^2_{x_3} & \cdots & \sigma_{x_3 x_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma_{x_1 x_n} & \sigma_{x_2 x_n} & \sigma_{x_3 x_n} & \cdots & \sigma^2_{x_n} \end{bmatrix}$$

ES: mutation

the role of the covariance matrix.

The covariance matrix plays a major role in the mutation:

- Uncorrelated mutation: $\langle \mathbf{x}, \sigma \rangle \Rightarrow \mathbf{C} = \sigma^2 \mathbf{I}$
  - the Gaussian can only change size and move! (light ✓,less efficient ✗)
- Uncorrelated with *n* step sizes: $\langle \mathbf{x}, \boldsymbol{\sigma} \rangle \Rightarrow \mathbf{C} = diag\{\sigma_1, \sigma_1 \ldots, \sigma_n\}$
  - the Gaussian can change size, shape and move! (compromise)
- Correlated mutation: $\langle \mathbf{x}, \boldsymbol{\sigma}, \boldsymbol{\alpha} \rangle$
  - the Gaussian can change size, shape, orientation and move! (efficient ✓, space/time complexity ✗)



ES: uncorrelated mutations
implementation details

$$\forall i = 0, 1, 2 \ldots, n$$

- Uncorrelated mutation (with 1 step size):

$$\sigma^{'} = \sigma e^{\tau \mathcal{N}(0,1)}, \quad (\tau \propto \frac{1}{\sqrt{n}})$$

$$\mathbf{x}^{'}[i] = \mathcal{N}_i\left(\mathbf{x}[i], \sigma^{'2}\right) = \mathbf{x}[i] + \sigma^{'}\mathcal{N}_i(0,1)$$

- Uncorrelated mutation (with n step sizes):

$$\boldsymbol{\sigma}^{'}[i] = \boldsymbol{\sigma}[i] e^{\tau^{'}\mathcal{N}(0,1) + \tau \mathcal{N}_i(0,1)}, \quad (\tau^{'} \propto \frac{1}{\sqrt{2n}}, \tau \propto \frac{1}{\sqrt{2\sqrt{n}}})$$

$$\mathbf{x}^{'}[i] = \mathcal{N}_i\left(\mathbf{x}[i], \sigma^{'2}\right) = \mathbf{x}[i] + \sigma^{'}\mathcal{N}_i(0,1)$$

Always check $\sigma \neq 0$! (see [Eiben and Smith, 2003] for details)

ES: correlated mutation
implementation details

- The covariance matrix is defined as

$$\mathbf{C} = [c_{i,j}] = \begin{cases} c_{ii} = \sigma_i^2 \\ c_{i,j} = \frac{1}{2}\left(\sigma_i^2 - \sigma_j^2\right) tan\left(2\alpha_{i,j}\right) \end{cases}$$

- first we update the diagonal

$$\boldsymbol{\sigma}'[i] = \boldsymbol{\sigma}[i] e^{\tau' \mathcal{N}(0,1) + \tau \mathcal{N}_i(0,1)}, \quad (\tau' \propto \frac{1}{\sqrt{2n}}, \tau \propto \frac{1}{\sqrt{2\sqrt{n}}})$$

- then the correlations

$$\alpha_j' = \alpha_j + \beta \mathcal{N}(0,1), \quad (\beta \approx 5^\circ)$$

Always check σs and αs are acceptable (see [Eiben and Smith, 2003] for details).

ES: survivor selection

comma and plus strategies.

Survivor selection is deterministic:

- Comma selection $\Rightarrow (\mu, \lambda)$-ES:
  - generate $\lambda > \mu$ offspring, the fittest μ of them replace the old population.
- Plus strategy $\Rightarrow (\mu + \lambda)$-ES:
  - generate λ offspring, the fittest μ out of a total of $\mu + \lambda$ points replace the old population.

Advantages of the comma strategy:

- better in leaving local optima;
- better in following moving optima;
- unlike the $(\mu + \lambda)$, prevents from retaining mediocre solutions for many generations;
- unlike the $(\mu + \lambda)$, the scheme is not too exploitative.

Advantages of the plus strategy:

- unlike (μ, λ), useful solutions are not cancelled.
- the $(\mu + \lambda)$-selection can be preferred when are looking for "marginal" enhancements and when it is important not to lose the previous enhancements;
- the $(\mu + \lambda)$ strategy can be preferable in large scale problems when exploitation is needed;

(1+1)-ES with One-Fifth Success Rule

- A very simple (single solution) ES has been originally proposed.
- This variant employed a basic uncorrelated mutation with single step size σ.
- Theoretical studies motivated an update on-the-fly for σ according to the popular $\frac{1}{5}$ success rule [Rechenberg, 1971].
- Two variants are described in this document (click-here).

References I

Eiben, A. E. and Smith, J. E. (2003).
Introduction to Evolutionary Computation.
Springer-verlag, Berlin.

Rechenberg, I. (1971).
Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.
PhD thesis, Technical University of Berlin.

Schwefel, H. (1981).
Numerical Optimization of Computer Models.
Wiley, Chichester, England, UK.

Schwefel, H.-P. (1965).
Kybernetische Evolutioals Strategie der experimentellen Forschung in der Strömungstechnik.
PhD thesis, Technical University of Berlin, Hermann Föttinger–Institute for Fluid Dynamics.

Covariance Matrix Adaptation ES
CMA-ES

- one of the most elegant and efficient meta-heuristic for real-valued optimisation!
  - designed on solid mathematical considerations [Hansen and Ostermeier, 1996] ✓
  - lacks of problem dependent parameters and is invariant to many transformations ✓
  - handles ill-conditioned problems ✓
  - nowadays treated as a benchmark for real-valued optimisation;
- many variants exist (e.g. sep-CAME-ES, g-CMA-ES, (1+1)-CMA-ES etc.);
- complex and memory expensive! $\Rightarrow$ not very suitable for real-time and large sale optimisation ✗
- preferable in mono-modal landscape rather than multi-modal.

Standard CMA-ES

(namely rank-μ-update CMA-ES with weighted global recombination [Nikolaus and Stefan, 2004])

- Standard CMA-ES is self-tuning: optimal parameters setting is given and reported in Table 1.
  - N:B. Also the optimal population size is self-tuned! ✓

Table: Description of parameters in CMA-ES

Parameter	Description	Value
λ	offspring number, population size	$\lambda = 4 + \lfloor 3ln(n) \rfloor$
μ	parents number, used to update mean value, covariance and step-size	$\mu = \lfloor \frac{\lambda}{2} \rfloor$
$w_{i=1\ldots\mu}$	recombination weights	$w_{i=1\ldots\mu} = \frac{ln(\mu+1) - ln(i)}{\sum_{j=1}^{\mu} ln(\mu+1) - ln(j)}$
μ_{eff}	variance effective selection mass	$\mu_{eff} = \frac{1}{\sum_1^{\mu} w_i^2}$
		($w_{i=1\ldots\mu}$ are chosen so that $\mu_{eff} \approx \frac{\lambda}{4}$)
c_c	decay rate for the evolution path	$c_c = \frac{4}{n+4}$
μ_{cov}	learning coefficient for the covariance matrix	$\mu_{cov} = \mu_{eff}$
c_{cov}	learning rate for the covariance matrix	$c_{cov} = \frac{1}{\mu_{cov}} \frac{2}{(n+\sqrt{2})^2} + \left(1 - \frac{1}{\mu_{cov}}\right) min\left(1, \frac{2\mu_{eff}-1}{(n+2)^2 + \mu_{eff}}\right)$
c_σ	decay rate for the conjugate evolution path	$c_\sigma = \frac{\mu_{eff}+2}{n+\mu_{eff}+3}$
d_σ	damping parameter for σ-change	$d_\sigma = 1 + 2max\left(0, \sqrt{\frac{\mu_{eff}-1}{1+n}} - 1\right) + c_\sigma$

Standard CMA-ES
working logic:

CMA-ES pseudo-code

–problem dependent user defined input parameters–

initialise n, $\langle \mathbf{x} \rangle_w^{(0)}$ and $\sigma^{(0)}$ ▷ $\sigma^{(0)} = 0.5$, $\langle \mathbf{x} \rangle_w^{(0)} \sim \mathcal{U}(0, 1)$ in [Nikolaus and Stefan, 2004]

–strategy parameter setting: selection–

initialise λ, μ, μ_{eff} and $w_{i=1\ldots,\mu}^{(0)}$ as in Table 1 ▷ λ can be modified if needed

–strategy parameter setting: adaptation–

initialise c_c, μ_{cov}, c_σ, d_σ and d_σ as in Table 1

–initialise dynamic (internal) matrices parameters and constants–

$\mathbf{P_c} = \mathbf{P_\sigma} = [\emptyset]$, $\mathbf{B} = \mathbf{D} = \mathbf{I}$, $\mathbf{C} = \mathbf{B} * \mathbf{D} (\mathbf{B} * \mathbf{D})^T$

–main loop–

while condition on budget **do**

- sample λ new individuals from distribution (*mutation*) ▷ Formula 1
- the new λ points replace the old population ((λ, λ) *strategy*)
- evaluate individuals and sort them based on their fitness (*recombination*)
- update $\langle \mathbf{x} \rangle$ based on a weighted sum of the best μ individuals ▷ Formula 2
- update the evolution paths $\mathbf{P}_\sigma$ and $\mathbf{P_c}$ ▷ Fomula 5 and 3
- update covariance matrix $\mathbf{C}$ and step-size σ consequently ▷ Formula 4 and 6

end while

Output best individual ever found $\mathbf{x_e}$

Standard CMA-ES
the sampling

- Mathematically, at a generic generation g, λ individuals are sampled from:

$$\mathbf{x}_{\mathbf{k}}^{(\mathbf{g}+\mathbf{1})} \sim \mathcal{N}\left(\langle \mathbf{x} \rangle_{w}^{(g)}, \left(\sigma^{g}\right)^{2} \mathbf{C}^{(g)}\right) \tag{1}$$

which is centred in the on the point obtained as weighted sum of the best $\mu \leq \lambda$ candidate solutions:

$$\langle \mathbf{x} \rangle_{w}^{(g)} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i \in \{\textit{indexes of the best } \mu \textit{ points}\}}^{g} \tag{2}$$

- Implementation-wise, we decompose $\mathbf{C}$ into its diagonal $(\mathbf{D})$ and orthonormal $(\mathbf{B})$ components via PCA:

$$\mathbf{C}^{(\mathbf{g})} = \mathbf{B}^{(\mathbf{g})} \mathbf{D}^{(\mathbf{g})\,2} \mathbf{B}^{(\mathbf{g})\,T}$$

such that Formula 1 can be implemented as:

$$\mathbf{x}_{\mathbf{k}}^{(\mathbf{g}+\mathbf{1})} = \langle \mathbf{x} \rangle_{w}^{(g)} + \sigma^{(g)} \mathbf{B}^{(\mathbf{g})} \mathbf{D}^{(\mathbf{g})} \mathcal{N}\left(\emptyset, \mathbf{I}\right)$$

Standard CMA-ES
the covariance matrix

- $\mathbf{C}$ is updated to approximate the local behaviour of the fitness landscape (Hessian matrix) in a nighbouroigh of the optimal solution!
- first, we update the evolution path $\mathbf{P_c}$:

$$\mathbf{p_c}^{(g+1)} = (1 - c_c)\mathbf{p_c}^{(g)} + H_\sigma^{(g+1)} \sqrt{c_c \cdot (2 - c_c)} \cdot \frac{\sqrt{\mu_{eff}}}{\sigma^{(g)}} \left(\langle \mathbf{x} \rangle_w^{(g+1)} - \langle \mathbf{x} \rangle_w^{(g)} \right) \tag{3}$$

and then:

$$\mathbf{C}^{(g+1)} = (1 - c_{cov})\mathbf{C}^{(g)} + c_{cov} \cdot \frac{1}{\mu_{cov}} \mathbf{p_c}^{(g+1)} \left(\mathbf{p_c}^{(g+1)} \right)^T$$

$$+ c_{cov} \cdot \left(1 - \frac{1}{\mu_{cov}} \right) \sum_{i=1}^{\mu} \left(\mathbf{x}_{i:\lambda}^{(g+1)} - \langle \mathbf{x} \rangle_w^{(g)} \right) \left(\mathbf{x}_{i:\lambda}^{(g+1)} - \langle \mathbf{x} \rangle_w^{(g)} \right)^T \tag{4}$$

with

$$H_\sigma^{(g+1)} \begin{cases} 1 & \text{if} \quad \dfrac{||\mathbf{P_\sigma}^{(g+1)}||}{\sqrt{1 - \left(1 - c_\sigma^{(g+1)2}\right)}} < \left(1.5 + \dfrac{1}{n - 0.5} \right) E\left(||\mathcal{N}(\emptyset, \mathbf{I})|| \right). \\ 0 & \text{otherwise} \end{cases}$$

and "conjugate evolution path" $\mathbf{P_\sigma}$ equal to:

$$\mathbf{P_\sigma}^{(g+1)} = (1 - c_\sigma) + \sqrt{c_\sigma \cdot (2 - c_\sigma)} \underbrace{\mathbf{D}^{(g)} \mathbf{B}^{(g)-1} \mathbf{D}^{(g)T}}_{\mathbf{C}^{(g)-\frac{1}{2}}} \frac{\sqrt{\mu_{eff}}}{\sigma^{(g)}} \left(\langle \mathbf{x} \rangle_w^{(g+1)} - \langle \mathbf{x} \rangle_w^{(g)} \right) \tag{5}$$

- finally, the step size is updated:

$$\sigma^{(g+1)} = \sigma^{(g)} e^{\frac{c_c}{d_\sigma} \left(\frac{||\mathbf{P_c}||}{E||N(\emptyset, I)||} - 1 \right)} \tag{6}$$

References I



Hansen, N. and Ostermeier, A. (1996).
Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation.
In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 312–317.



Nikolaus, H. and Stefan, K. (2004).
Evaluating the cma evolution strategy on multimodal test functions.
In *Parallel Problem Solving from Nature-PPSN VIII*, pages 282–291. Springer.

Algorithmic design

- Historically, it is done empirically
  - (an art form other than science)
- This led to replicability and other issues:
  - nomenclature is not uniform
  - some operators are taken for granted in articles [Kononova et al., 2024]
  - redundant components, etc.

An informed approach chooses operators based on a fine landscape analysis.

Modular algorithms

You were provided with two readings for this topic →[Vermetten et al., 2023, Vermetten et al., 2022]

- **Enables Fair Comparisons**
  A common framework enables consistent algorithm benchmarking, enhancing fairness and reliability.
- **Simplifies Implementation and Integration**
  Modular components reduce complexity, minimising inconsistencies, bugs, and overlooked edge cases.
- **Enhances Reproducibility**
  Standardised modules ensure consistent behaviour across implementations, addressing reproducibility issues.
- **Promotes Reusability and Specialisation**
  Core algorithms can be reused with different modules, enabling specialisation without full redesign.
- **Supports Systematic Study of Interactions**
  Modular frameworks allow isolation and analysis of individual components and their interactions.
- **Encourages Community Collaboration**
  Shared modular infrastructure fosters collaboration and knowledge sharing within the research community.
- **Accelerates Development of New Variants**
  New ideas can be tested quickly by swapping or modifying modules, speeding up innovation.

Modular DE and CAMES

Something you may want to explore in your teamwork sessions

- The DE framework encourages modularisation:
  - ModularDE is described in the article [Vermetten et al., 2023]
  - Explore its code and extra resources on GitHub, Figshare, and Zenodo (links provided in the articles)

- ModularDE in [Kononova et al., 2024] studied box-constraint handling methods and algorithmic behaviours (code fully available).

- Discover ModularCAMES in [Vermetten et al., 2022] and dive into its code and uncover structural bias in multiple configurations.

References I



Kononova, A. V., Vermetten, D., Caraffini, F., Mitran, M.-A., and Zaharie, D. (2024).
The importance of being constrained: Dealing with infeasible solutions in differential evolution and beyond.
Evolutionary Computation, 32(1):3–48.



Vermetten, D., Caraffini, F., Kononova, A. V., and Bäck, T. (2023).
Modular differential evolution.
In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '23, page 864–872, New York, NY, USA. Association for Computing Machinery.



Vermetten, D., Caraffini, F., van Stein, B., and Kononova, A. V. (2022).
Using Structural Bias to Analyse the Behaviour of Modular CMA-ES, page 9.
Association for Computing Machinery, New York, NY, USA.

Structural Bias

What is Structural Bias for an optimisation heuristc? An informal definition...

An optimisation algorithm is said to possess Structural Bias (SB) if it gives preference to some parts of the domain regardless of $f(x)$ [Kononova et al., 2015]

Structural Bias

What is Structural Bias for an optimisation heuristc? An informal definition...

An optimisation algorithm is said to possess Structural Bias (SB) if it gives preference to some parts of the domain regardless of $f(x)$ [Kononova et al., 2015]

- imagine an algorithm (**general purpose and thus unbiased**) as a pinball machine where a player aims at keeping the ball on the game surface as long as possible:



- if the ball is the candidate solution;
  - and multiple games are happening to conceptualise population-based frameworks;
- then trajectories drawn by the balls should access the entire game surface...

Structural Bias

What is Structural Bias for an optimisation heuristc? An informal definition...

An optimisation algorithm is said to possess Structural Bias (SB) if it gives preference to some parts of the domain regardless of $f(x)$ [Kononova et al., 2015]

- Imagine an algorithm (**general purpose and thus unbiased**) as a pinball machine where a player aims at keeping the ball on the game surface as long as possible:
- if the ball is the candidate solution;
  - and multiple games are happening to conceptualise population-based frameworks;
- then trajectories drawn by the balls should access the entire game surface...



However, this is not the case:

no matter how trained the player is and the precise mechanical structure of the pinball, due to gravity, the balls will eventually roll to its lower side!

Structural Bias

What is Structural Bias for an optimisation heuristc? An informal definition...

An optimisation algorithm is said to possess Structural Bias (SB) if it gives preference to some parts of the domain regardless of $f(x)$ [Kononova et al., 2015]

However, this is not the case:

no matter how trained the player is and the precise mechanical structure of the pinball, due to gravity, the balls will eventually roll to its lower side!



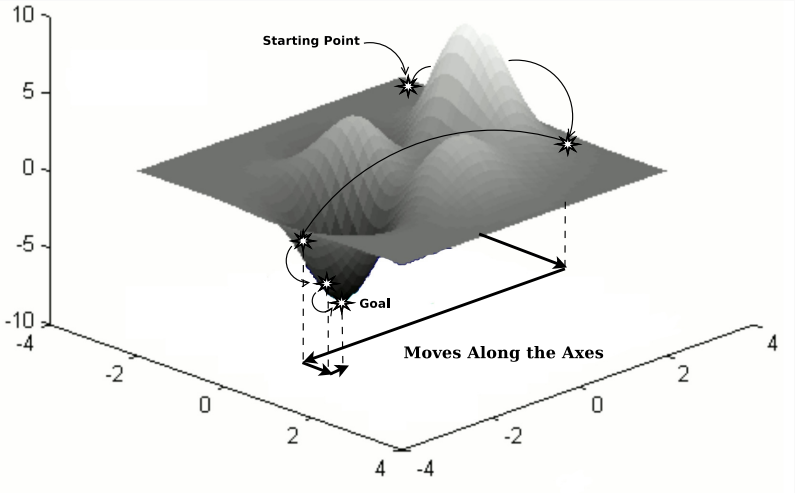
Similarly, metaheuristics algorithms intrinsically tend to privilege regions of the search space, ending up with:

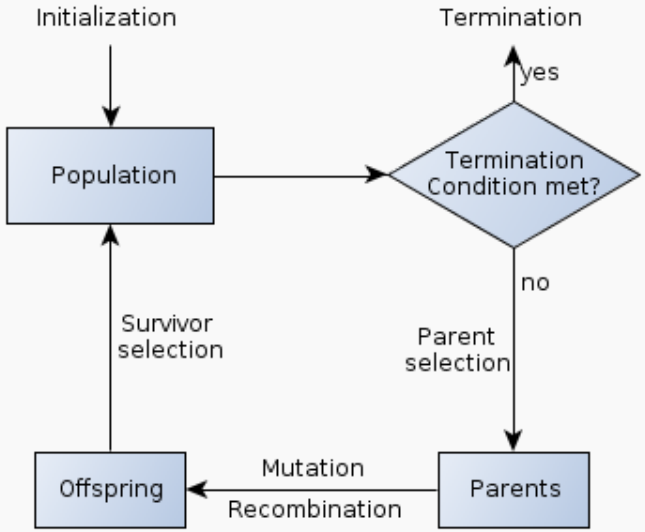
- premature convergence;
- unsatisfactory solutions;
- low exploration of the search space, etc.

Looking for Structural Bias

For a general objective function, the movement of a solution or population of solutions is dictated by a superposition of two forces of:

1. the "gradient" formed by the values of objective function in the current population
2. SB that originates from the structure of the iteratively operating algorithm.





These are not necessarily in agreement in terms of direction and strength!

To detect 1, one must nullify 2!

Looking for Structural Bias
The testbed

- For the purpose of identifying SB, we make use of an optimisation problem defined as:

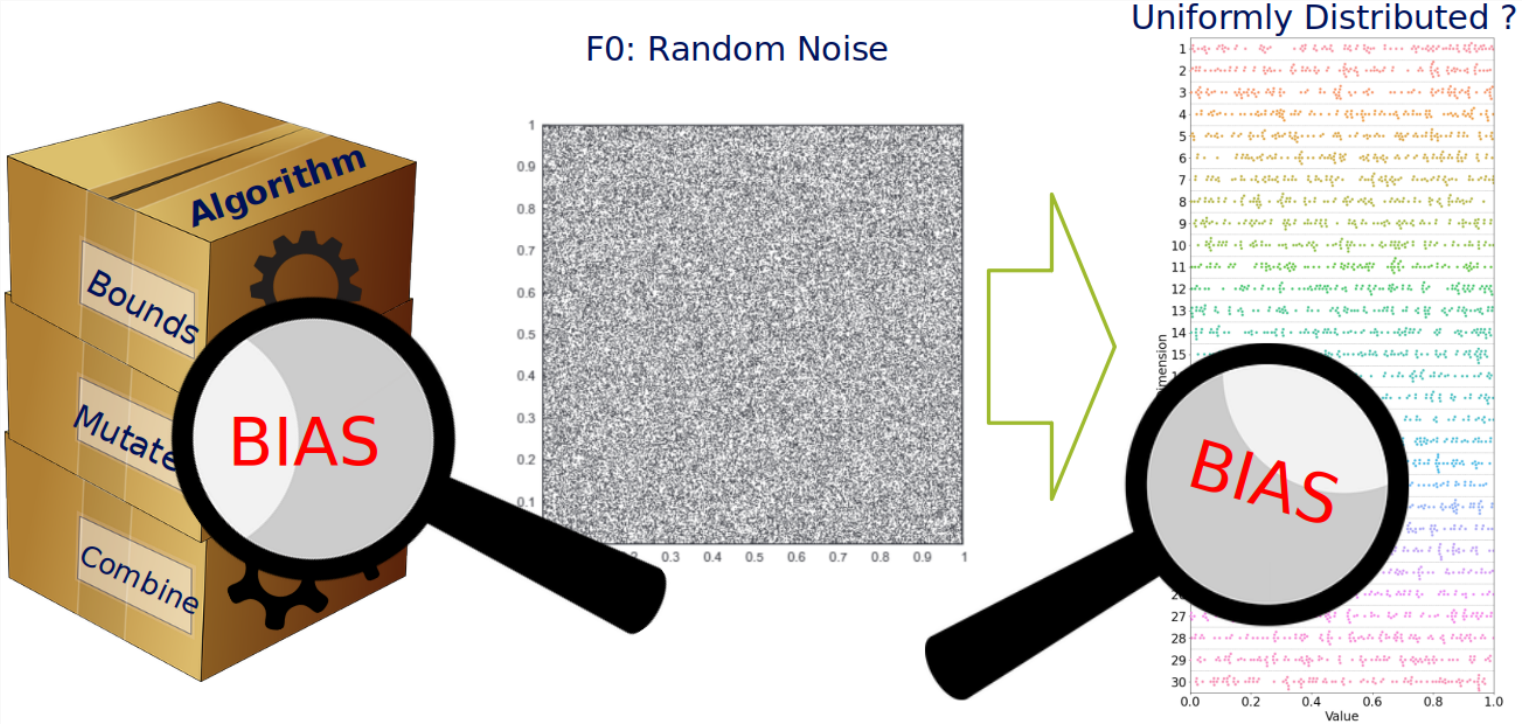
$$f_0 \rightarrow min, \quad f_0 : [0,1]^n \subset \mathbb{R}^n \rightarrow [0,1], \quad f_0(x) = Uniform[0,1]$$

- An ideal unbiased algorithm should return a uniform distribution of the best final solutions, over a series of independent runs [Kononova et al., 2015].

Looking for Structural Bias

A formal (operative) definition

An optimisation algorithm is structurally biased if the distribution of the best solutions obtained with multiple runs on $f_0(x)$ is not uniform.



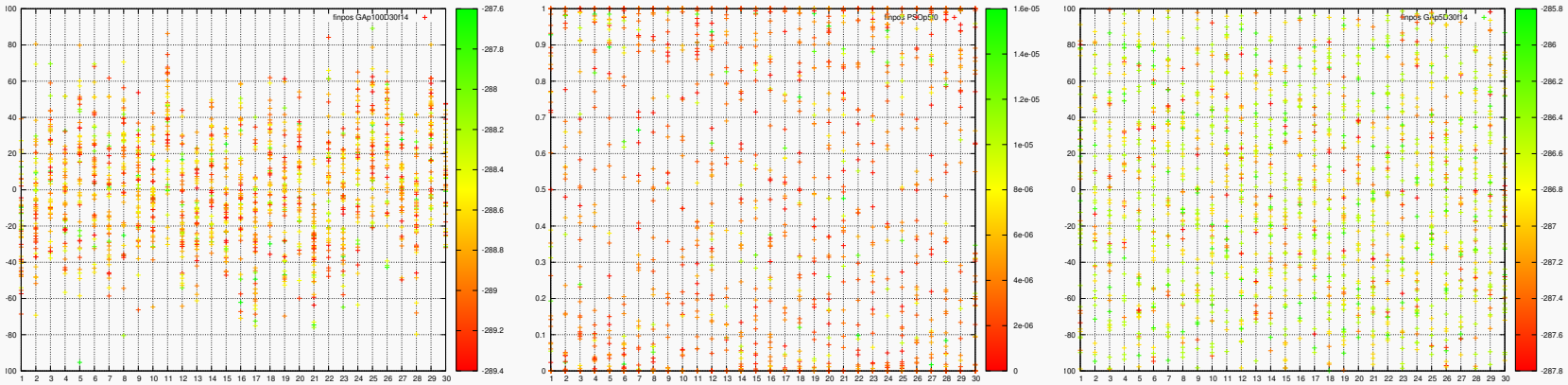
Looking for Structural Bias

A visual approach

- Final populations (obtained in different runs) were displayed in parallel coordinates,
  - i.e. coordinates of a vector in an n-dimensional space are marked on n equally spaced parallel lines.

An unbiased structure should return points homogeneously filling the entire interval in each dimension.

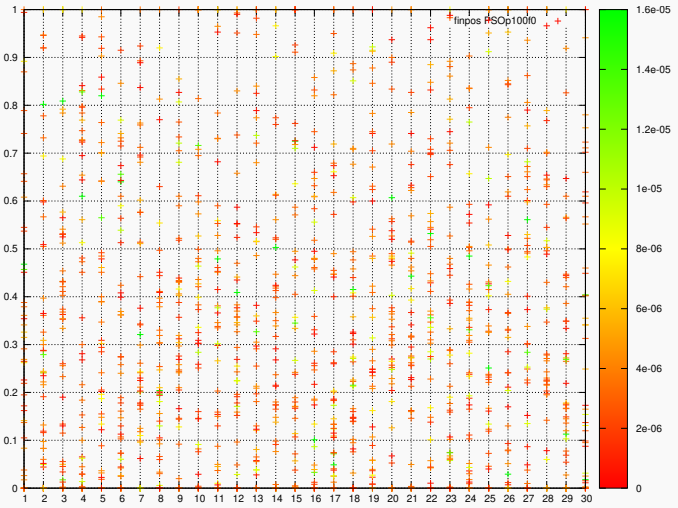
Conversely, a biased one, should show clusters of best solutions accumulated in one or more segments of parallel lines.



Looking for Structural Bias

A visual approach

- Easy to produce and intuitive 👍 (in the presence of clear SB)
- Subjective and time-onerous to inspect in large-scale experiments 👎



- We visualised SB for hundreds and hundreds of algorithm configurations [Kononova et al., 2015, Caraffini and Kononova, 2019, Caraffini et al., 2019, Kononova et al., 2020a, Kononova et al., 2020b, Vermetten et al., 2022c]
- and found multiple bias types and interesting patterns and correlations to performance, parameters, bound constraint handling methods,...[Vermetten et al., 2021, van Stein et al., 2021, Vermetten et al., 2022a]

Want to see more?

Look at the online repositories 🔗🔗🔗 (also referenced in the articles).

Looking for Structural Bias
A statistical approach

Testing for uniformity

- Staistic is an established field ✓
- Tests are easily automatised and displayed even in large-scale settings ✓
  - Degree (strength) of SB computed as # of rejected dimensions ✓

Kind of SB test:	visual					statistical				
Configuration:	CUTN	dismiss	mirror	saturation	toroidal	CUTN	dismiss	mirror	saturation	toroidal
cDE/rand/1/bin	=	=	•	•	=	0.00	0.00	0.02	∞	0.13
cDE/rand/1/exp	=	•	=	•	=	0.00	0.00	0.00	∞	0.02
cDE/rand/2/bin	•	=	•	•	•	0.02	0.00	0.02	∞	0.21
cDE/rand/2/exp	=	=	=	•	=	0.00	0.00	0.00	∞	0.06
cDE/current-to-rand/1	=	=	=	•	•	0.00	0.02	0.00	∞	0.00
cDE/best/1/bin	=	•	•	•	=	0.01	0.00	0.00	∞	0.00
cDE/best/1/exp	•	=	=	•	•	0.00	0.00	0.00	∞	0.00
cDE/best/2/bin	=	=	=	•	•	0.01	0.00	0.00	∞	0.01
cDE/best/2/exp	=	=	=	•	•	0.00	0.00	0.00	∞	0.01
cDE/current-to-best/1/bin	•	=	=	•	•	0.00	0.00	0.01	∞	0.02
cDE/current-to-best/1/exp	•	=	=	•	•	0.00	0.00	0.00	1.00	0.01
cDE/rand-to-best/2/bin	•	=	=	•	=	0.00	0.00	0.00	∞	0.01
cDE/rand-to-best/2/exp	•	•	•	•	•	0.02	0.00	0.00	∞	0.02
cDE-Light	•	•	•	•	•	0.00	0.00	0.00	0.01	0.00
cPSO	=	•	=	•	=	0.03	0.00	0.44	∞	0.01
cBFO	=	=	=	=	=	1.00	0.92	1.00	0.96	0.93
cGA (no SDIS, shown as dismiss)	×	•	×	×	×	×	0.01	×	×	×
Found strong SB/total cases:	1/16	1/17	2/16	1/16	1/16	1/16	1/17	2/16	15/16	2/16
Found mild SB/total cases:	8/16	6/17	4/16	13/16	9/16	5/16	2/17	3/16	1/16	10/16
Strong/mild/no SB cases:	6/40/35					21/26/59				
Agreement between visual and statistical tests (in %, calculated 'post factum'):	56	65	69	6	44	all cases				
	100	100	100	100	100	cases with strong SB only*				
	38	17	25	0	55	cases with mild SB only				
	71	90	80	0	17	cases with no SB only				

Looking for Structural Bias
A statistical approach

Testing for uniformity

- Staistic is an established field ✓
- Tests are easily automatised and displayed even in large-scale settings ✓
  - Degree (strength) of SB computed as # of rejected dimensions ✓

Challenges

[Kononova et al., 2015, Kononova et al., 2020b, Kononova et al., 2020a, van Stein et al., 2021]

- Sensitivity, etc. depends on the sample size N! ✗
- different test might disagree with low/high sample sizes ✗
- Kolmogorov–Smirnov→Cramér-Von Mises→Anderson–Darling + Benjamini–Hochberg correction ($N = 100$, detect Strong SB)
- To detect mild SB with good results $N = 600$ is needed (unpractical 👎)

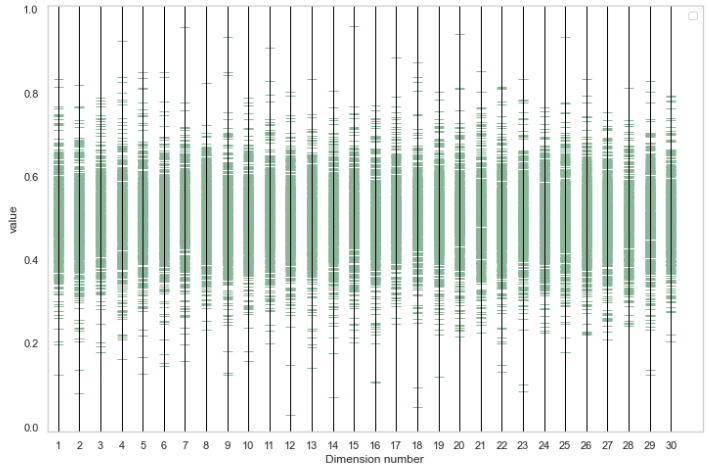
Looking for Structural Bias

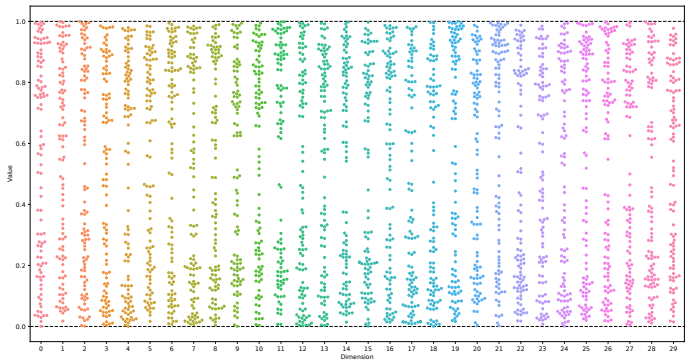
A statistical approach

BIAS TOOLBOX [Vermetten et al., 2022b]

(also available on)

- 36 per-dimension statistical tests
- 32 across-dimension statistical tests
  - + Random Forest
- Comes with data and plot generator! 👍
- Detects/predicts bias 👍
- Classifies its kind 👍
- Measure SB strength 👍
- $N = 30 - 50$ 👍



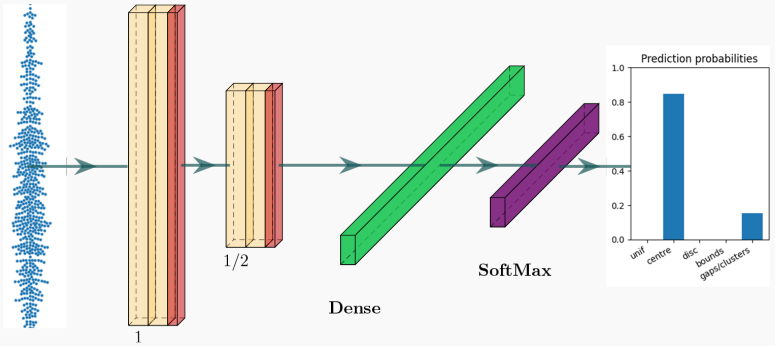


Looking for Structural Bias

A Deep Learning approach

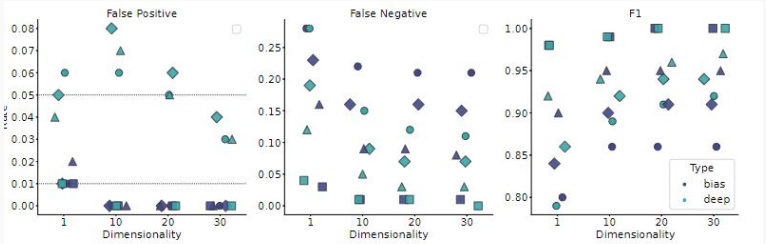
[code also available on GitHub]

Deep-BIAS extends the BIAS toolbox with a one-dimensional CNN to detect and classify SB [Van Stein et al., 2023].



Compared to BIAS, Deep-BIAS displays interesting and advantageous features:

- It outperforms the statistical test-based approach in classifying the type of SB.
- It gives a better measure of the strength of the SB by using the class probabilities.
- It provides additional insight by using XAI, where regions of interest in distributions can be further analysed to understand the mechanism behind the classifications



igure 1. Comparison (with $\alpha = 0.01$) of the original BIAS toolbox (blue) and the Deep-BIAS eal) in terms of false positives (left), false negatives (middle) and F1-score (right). On all gures, markers identify the used sample size: ○, ◇, △ and □ are 30, 50, 100 and 600, espectively.

Strucutral Bias
Why bother?

- to improve upon performances of "general purpose" algorithms;
- to achieve a more informed algorithm design, which is currently done in a partially blind manner by researchers;
- to help practitioners pick the right "off-the-shelf" optimiser and produce solutions of satisfactory quality for their real-world scenarios.

References I

Caraffini, F. and Kononova, A. V. (2019).
Structural bias in differential evolution: A preliminary study.
AIP Conference Proceedings, 2070(1):020005.

Caraffini, F., Kononova, A. V., and Corne, D. W. (2019).
Infeasibility and structural bias in differential evolution.
Information Sciences, 496:161–179.

Kononova, A. V., Caraffini, F., Wang, H., and Bäck, T. (2020a).
Can compact optimisation algorithms be structurally biased?
In Bäck, T., Preuss, M., Deutz, A., Wang, H., Doerr, C., Emmerich, M., and Trautmann, H., editors, *Parallel Problem Solving from Nature – PPSN XVI*, pages 229–242, Cham. Springer International Publishing.

Kononova, A. V., Caraffini, F., Wang, H., and Bäck, T. (2020b).
Can single solution optimisation methods be structurally biased?
In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–9, Glasgow. IEEE.

References II



Kononova, A. V., Corne, D. W., Wilde, P. D., Shneer, V., and Caraffini, F. (2015).
Structural bias in population-based algorithms.
Information Sciences, 298:468–490.



van Stein, B., Caraffini, F., and Kononova, A. V. (2021).
Emergence of structural bias in differential evolution.
In *Proceedings of the 2021 Genetic and Evolutionary Computation Conference Companion*, GECCO '21 Companion, New York, NY, USA. Association for Computing Machinery.



Van Stein, B., Vermetten, D., Caraffini, F., and Kononova, A. V. (2023).
Deep bias: Detecting structural bias using explainable ai.
In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, GECCO '23 Companion, page 455–458, New York, NY, USA. Association for Computing Machinery.

References III



Vermetten, D., Caraffini, F., van Stein, B., and Kononova, A. V. (2022a).
Using Structural Bias to Analyse the Behaviour of Modular CMA-ES, page 9.
Association for Computing Machinery, New York, NY, USA.



Vermetten, D., Kononova, A. V., Caraffini, F., Wang, H., and Bäck, T. (2021).
Is there anisotropy in structural bias?
In *Proceedings of the 2021 Genetic and Evolutionary Computation Conference Companion*, GECCO '21 Companion, New York, NY, USA. Association for Computing Machinery.



Vermetten, D., van Stein, B., Caraffini, F., Minku, L. L., and Kononova, A. V. (2022b).
Bias: A toolbox for benchmarking structural bias in the continuous domain.
IEEE Transactions on Evolutionary Computation, 26(6):1380–1393.

References IV



Vermetten, D., van Stein, B., Kononova, A. V., and Caraffini, F. (2022c).
Analysis of Structural Bias in Differential Evolution Configurations, pages 1–22.
Springer Singapore, Singapore.