An introduction to real world optimisation

Fabio Caraffini

De Montfort University
CCI: Centre for Computational Intelligence

Academic year 2018-2019

Towards real-world optimisation

- We know how to solve problems with an explicit analytical expression, when the function is derivable and well as its derivative,
- and we easily see how the operations become very onerous to analytically solve the problem with the increase of dimensionality.
- In the vast majority of cases, optimisation problems cannot be tackled analytically.

Real-world problems

- Normally, the objective function of real-world problems can be a piece of software, a simulator, an experiment, etc., also known as black box function.
- The objective function is also unknown and no hypotheses can be made on it behaviour.
- On the other hand, some testing of the problem can be made before deciding what approach we want to apply.

Metaheuristics

- From the Greek words metá and heuriskein, which can be interpreted with the expression "beyond the search", metaheuristics are general purpose optimisation methods.
- the search for the optimal configuration is taken to a higher (abstract) level, requiring only that the quality of a candidate solution is calculated via a cost function specific to the problem.
- Meta-heuristics are search algorithms that do not require specific hypotheses (a black box function is enough and we do not need to have any piece of information about its derivatives) and do not offer guarantees of convergence to the global optimum.

Metaheuristics are not magic!

- Albeit general purpose, there is no such thing as a metaheurstic solving all the problems.
- Every problem should be addressed with a proper algorithm that is tailored around the problem features.

What is the best optimiser?

There is no best optimiser! [Wolpert and Macready, 1997]

- The 1^{st} of the No Free Lunch Theorems (NFLT) presented in [Wolpert and Macready, 1997] states that for a given pair of algorithms A and B:

$$\sum_f P(\mathbf{x_m} \big| m, f, A) = \sum_f P(\mathbf{x_m} \big| m, f, B)$$

  where $P(\mathbf{x_m} \big| m, f, A)$ is the probability that algorithm A detects, after m iterations, the optimal solution $\mathbf{x_m}$ for a generic objective function f (analogously for $P(\mathbf{x_m} \big| m, f, B)$).
- The performance of every pair of algorithms over all the possible problems is the same.

see also appendix B of [Caraffini, 2014] and [Eiben and Smith, 2003] regarding NFLT

Limitations of NFLT

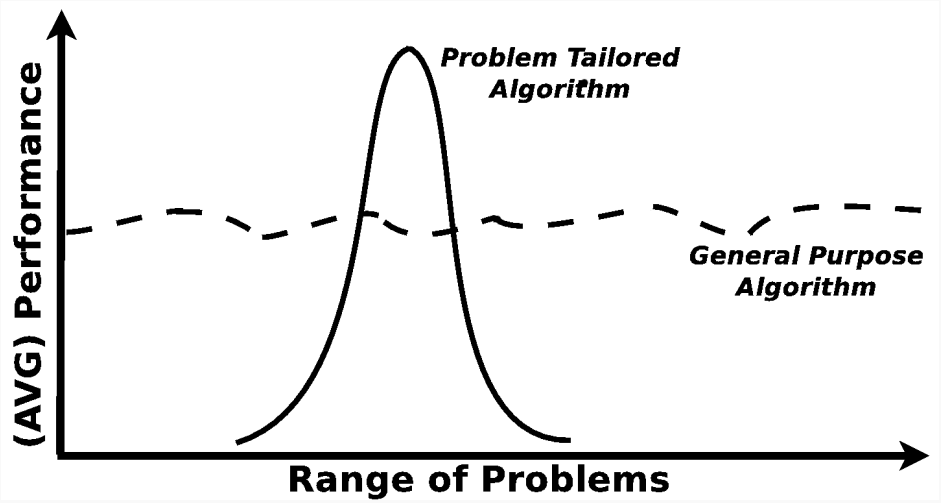
do (continuous) free lunches exist?

- NFLT is valid under some hypotheses: discrete problems, non re-visiting algorithms...
- NFLT is not valid for continuous optimisation problems [Auger and Teytaud, 2010].
- NFLT is not valid in meta-spaces, as for example in hyper-heuristics[1][Poli and Graff, 2009].

[1] for more information about hyper-heuristics give a look at this (link) and these papers [Burke et al., 2010, Özcan et al., 2008]

General message of NFLT

- Still, during the last two decades the idea that there is no general optimiser became clear.
- Every problem is a separate story and the algorithm should be connected to the problem!

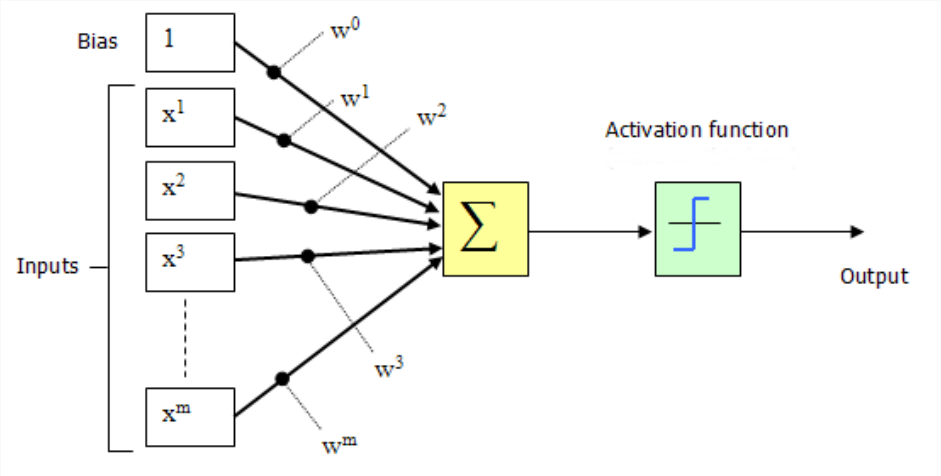


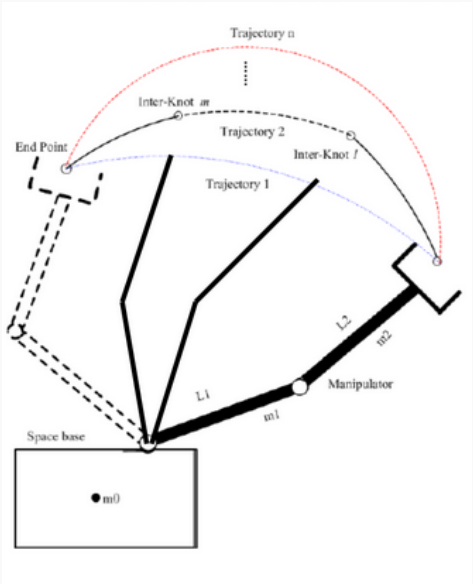
Hard real-life

- Optimisation Problems are often rather easily formulated but very hard to be solved when the problem comes from an application. In fact, some features characterising the problem can make it extremely challenging

Some of these features are summarised in the following slides:

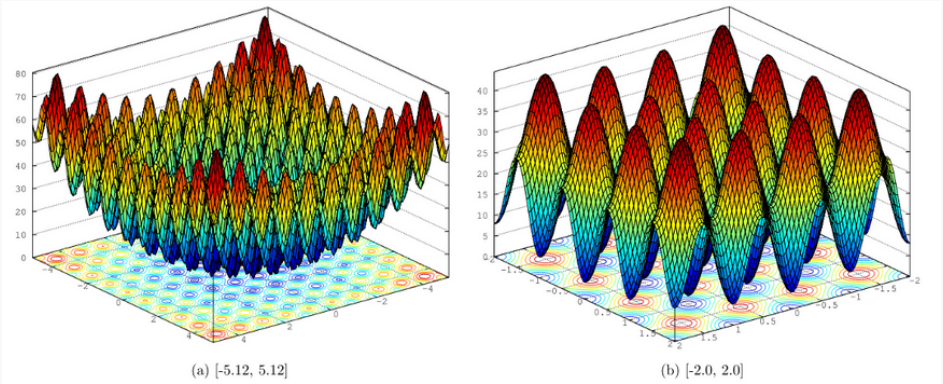
Hard real-life:
HIGH NON-LINEARITY.





- 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.

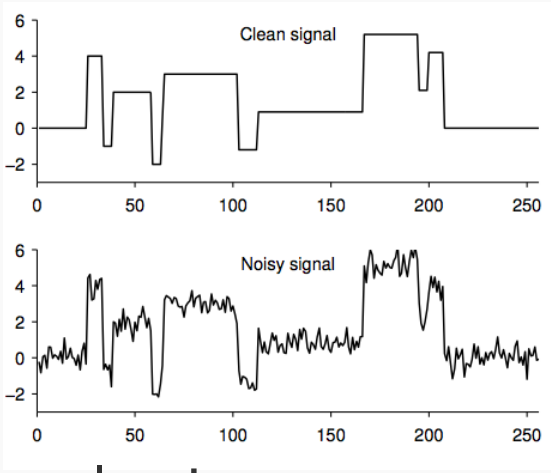
Hard real-life:
HIGH MULTI-MODALITY.



- 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**.[2]

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

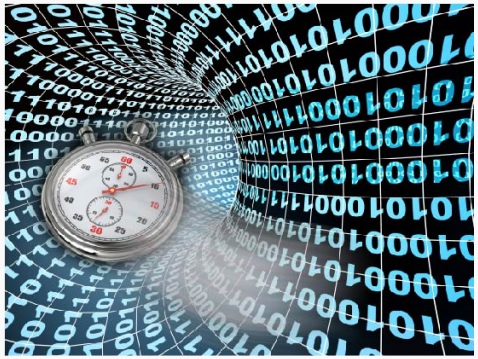
Hard real-life:
OPTIMISATION IN UNCERTAIN ENVIRONMENTS.



- **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 technology, 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!

Single solution deterministic metaheuristics:
generalities.

- One solution as an input, one solution as an output.
- Referred to as deterministic local search.
- Deterministic[3] procedure to improve upon an initial guess.
- Deterministic strategy to generate a trial individual.
- Deterministic criterion to select a new base point.

Their performance can heavily depend on the initial guess!

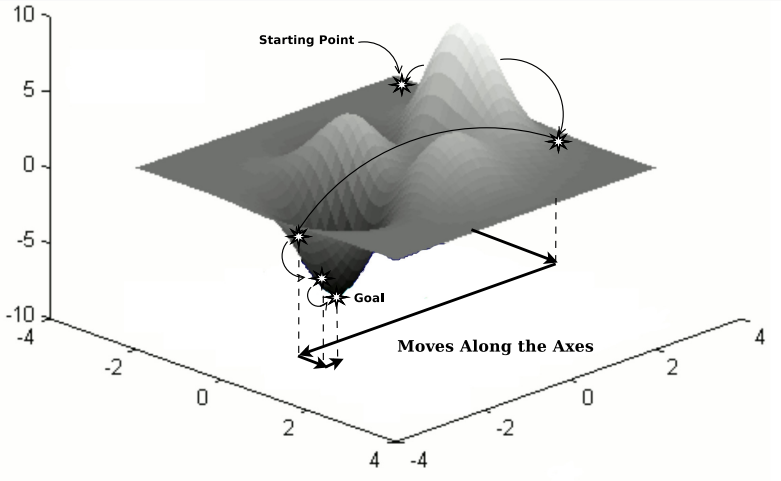
[3] Algorithm free of any form of randomisation logic, performing a predictable sequence of steps. They produce the same output, if the same input is used.

Single solution deterministic metaheuristics:
the role of the gradient.

- They are metaheuristics and thus gradient-free: they work on the function as a black box.
- Nonetheless, some of them may use the implicit information on the gradient, i.e. test of the surrounding area and follow the improvements until an area where no achievable improvements can be reached.

The "S" algorithm

- Very simple algorithm that has a famous version in Hooke-Jeeves implementation [Hooke and Jeeves, 1961], subsequently revisited in [Tseng and Chen, 2008] into S[4].
- It performs the perturbation of each variables and selects the most convenient solution at the end of the exploration.
- This algorithm orthogonally moves along the axes (and not "diagonally"):



[4]The name S is used in [Caraffini et al., 2012b], see the paper for further information.

S: working principle

- An initial point $\mathbf{x_{best}}$ is either sampled within the domain (decision space) $D \subset \mathbb{R}^n$ or obtained from another optimiser;
- an exploratory radius $\boldsymbol{\rho}$ is initialised as $\alpha\%$ of D's length (usually $\alpha = 40\%$);
- the n design variables of the only one solutions are perturbed one at one ($\forall i = 1, 2, 3 \ldots, n$):
  - $\mathbf{x_s}[i] \leftarrow \mathbf{x_s}[i] - \boldsymbol{\delta}[i]$, does $f(\mathbf{x_s})$ improve upon $f(\mathbf{x_{best}})$?
  - If no, we restore the initial value and half[5] step in taken in the opposite direction (i.e. $\mathbf{x_s}[i] = \mathbf{x_{best}}[i] + \frac{\delta}{2}$); does it now?
- If no improvement has been registered at the end of an iteration $\boldsymbol{\rho}$ is halved!
- We let S perform further iterations until a condition on the computational budget (or on the norm of $\boldsymbol{\rho}$) is met.

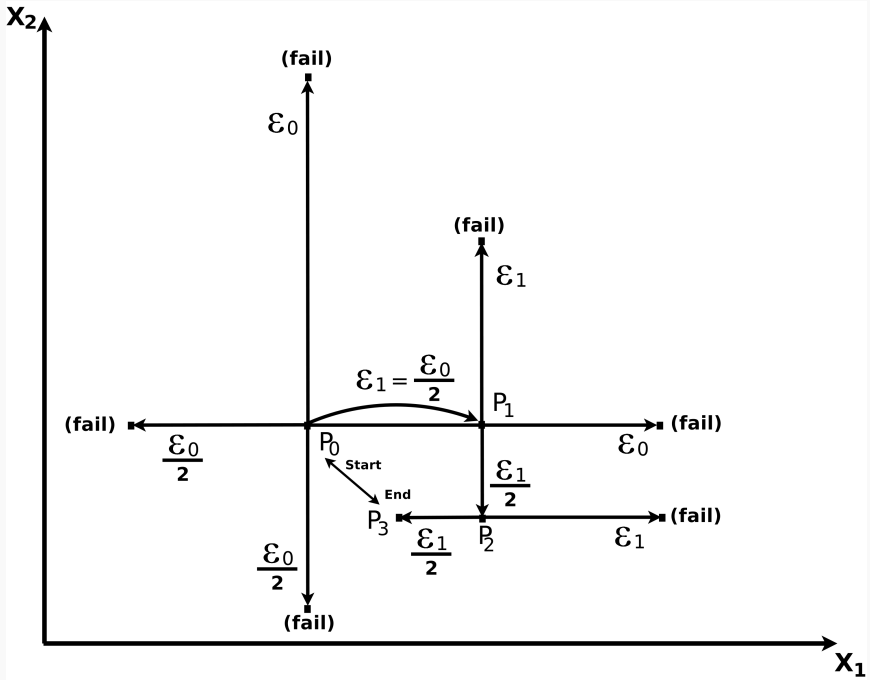
[5] Half step is preferred to avoid useless revisiting situations.

S: implementation details[6]

(hyper-cubical case, condition on budget)

S algorithm pseudo-code

$x_s = x_{best} \leftarrow$ initial guess $\in \mathbb{R}^n$
$\delta = \alpha \left(x^U - x^L \right)$ ▷ $x^{U/L}$: upper and lower bounds
while *condition on budget* **do**
  for $i = 1 : n$ **do**
    $x_s[i] = x_{best}[i] - \delta[i]$
    if $f(x_s) \leq f(x_{best})$ **then**
      $x_{best}[i] = x_s[i]$
    else
      $x_s[i] = x_{best}[i]$
      $x_s[i] = x_{best}[i] + \frac{\delta}{2}$
      if $f(x_s) \leq f(x_{best})$ **then**
        $x_{best}[i] = x_s[i]$
      else
        $x_s[i] = x_{best}[i]$
      end if
    end if
  end for
  if $f(x_{best})$ has never improved **then**
    $\delta \leftarrow \frac{\delta}{2}$
  end if
end while
Output x_{best}

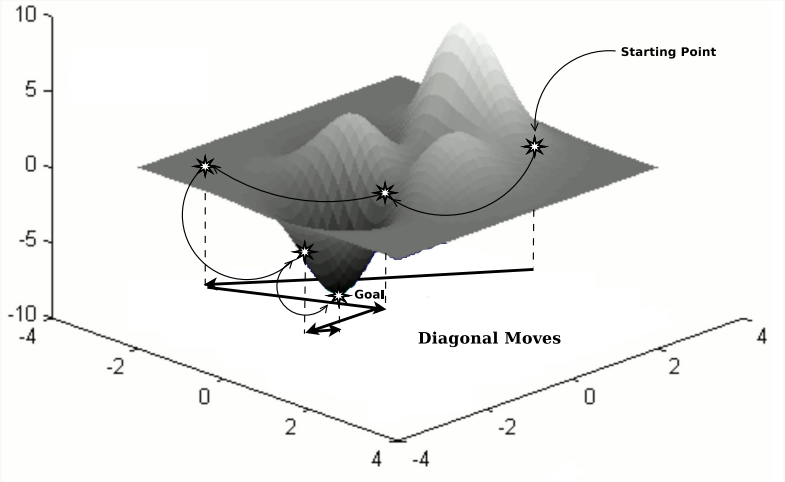


Example with initial radius set to ε_0

[6] N.B. pseudo-code $\neq$ code! Remember that the budget is the number of function calls ($f()$), do not call $f()$ when it not needed! Saturate solutions produced out of bounds!

Rosenbrock Method
[Rosenbrock, 1960]

- Modification of a steepest descent method requiring a rotation matrix $\boldsymbol{\xi} = [\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \boldsymbol{\xi}_3 \ldots, \boldsymbol{\xi}_n]$, $\boldsymbol{\xi}_i \in \mathbb{R}^n$;
- the main idea is to use the n directions ξ_i to perturb the candidate solution $\mathbf{x_k}$;
- the process is repeated as long as at least one of the n generated solutions improves upon $\mathbf{x_k}$;
- subsequently, successful results are used to update the rotation matrix so that at least one direction is rotated towards the estimated gradient $\Rightarrow$ diagonal move!



Rosenbrock
The perturbation

The following perturbation is applied to $\mathbf{x_k}$ in order to generate a new solution $\mathbf{x_r}$ n times:

$$\mathbf{x_r} = \mathbf{x_k} + \mathbf{d}[i]\xi_i\,, \;\; i = 1, 2, 3, \ldots, n$$

- if successful, the scaling factor is amplified $\mathbf{d}[i] = \alpha \cdot \mathbf{d}[i]$ and added on $\boldsymbol{\lambda}[i] = \boldsymbol{\lambda}[i] + \mathbf{d}[i]$;
- conversely, it is reduced and the exploratory direction inverted $\mathbf{d}[i] = -\beta \cdot \mathbf{d}[i]$.

Rosenbrock
Update of the rotation matrix.

If the perturbation fails at improving after a number of successful cases, if $\min(|\mathbf{d}|) > \varepsilon$ *OR* $\min(|\mathbf{x_k} - \mathbf{x_0}|) > \varepsilon$, a reset of $\boldsymbol{\lambda}$ and $\mathbf{d}$ occurs, and the coordinate system is rotated towards the "steepest slope" by means of the Gram-Schmidt orthogonalisation procedure (HERE).

- It can lead to numerical instability!
- we consider an improved version proposed in [Palmer, 1969]:

procedure update$\xi(\mathbf{d}, \boldsymbol{\lambda}, \mathbf{x_0})$
  $\mathbf{A_n} \leftarrow \boldsymbol{\lambda} \circ \boldsymbol{\xi_n}$
  for k=n-1:1 **do** ▷ $\circ$: entrywise product
    $\mathbf{A_k} \leftarrow \mathbf{A_{k+1}} + \boldsymbol{\lambda} \circ \boldsymbol{\xi_k}$
  end for
  $\mathbf{t}[n] \leftarrow \boldsymbol{\lambda}[n]^2$
  for i=n-1:1 **do**
    $\mathbf{t}[i] \leftarrow \mathbf{t}[i+1] + \boldsymbol{\lambda}[i]^2$
  end for
  for i=n:2 **do**
    $i \leftarrow i - 1$
  end for
  $\xi_1 \leftarrow \mathbf{A_1} / \sqrt{\mathbf{t}[1]}$, $\mathbf{x_0} \leftarrow \mathbf{x_k}$, $\boldsymbol{\lambda} \leftarrow \emptyset$, $\mathbf{d} \leftarrow \frac{\mathbf{1}}{\mathbf{10}} ones(n)$
  $\xi_1 \leftarrow \mathbf{A_1} / \sqrt{\mathbf{t}[1]}$
  Output $\boldsymbol{\xi}$
end procedure

Rosenbrock: implementation details

Rosenbrock Algorithm psudo-code

$\mathbf{x_r}, \mathbf{x_k}, \mathbf{x_0} \leftarrow$initial guess
$\mathbf{d} \leftarrow \frac{\mathbf{1}}{\mathbf{10}} ones(n)$ ▷ n-dimensional vector of 0.1
$\boldsymbol{\lambda} \leftarrow \emptyset$
$\boldsymbol{\xi} \leftarrow eye\,(n, n)$ ▷ $n \times n$ identity matrix
$y'' \leftarrow f\,(x_r)$
while *condition on budget* **do**
  $y \leftarrow y''$
  do
    $y' \leftarrow y$
    for $i = 0 : n$ **do**
      $\mathbf{x_r} \leftarrow \mathbf{x_k} + \mathbf{d}[i]\boldsymbol{\xi}_i$
      if $f\,(\mathbf{x_r}) \leq y$ **then**
        $\boldsymbol{\lambda}[i] \leftarrow \boldsymbol{\lambda}[i] + \mathbf{d}[i]$
        $\mathbf{d}[i] \leftarrow \alpha \cdot \mathbf{d}[i]$ ▷ $\alpha = 3$ in [Rosenbrock, 1960], 2 in [Caraffini et al., 2012a]
        $\mathbf{x_k} \leftarrow \mathbf{x_r}$
        $y \leftarrow f\,(\mathbf{x_r})$
      else
        $\mathbf{d}[i] \leftarrow -\beta \cdot \mathbf{d}[i]$ ▷ $\beta = 0.5$ [Rosenbrock, 1960]
      end if
      $i \leftarrow i + 1$
    end for
  while $y < y'$
  if $y < y''$ *AND* $(\min(|\mathbf{d}|) > \varepsilon$ *OR* $\min(|\mathbf{x_k} - \mathbf{x_0}|) > \varepsilon)$ **then** ▷ $\varepsilon = 10^{-5}$ [Rosenbrock, 1960]
    $\boldsymbol{\xi} \leftarrow$update$\xi(\mathbf{d}, \boldsymbol{\lambda}, \mathbf{x_0})$
  end if
end while
Output $\mathbf{x_r}$

More single solution deterministic algorithms.

Here some more popular techniques you may find in the literature:

- powell's direction set method [Powell, 1964]
- Hooke-Jeeves' Method [Hooke and Jeeves, 1961]

They are very similar to the previously analysed optimiser. Insted, we conclude with a popular "borderline" case: The Simplex algorithm [Nelder and Mead, 1965].

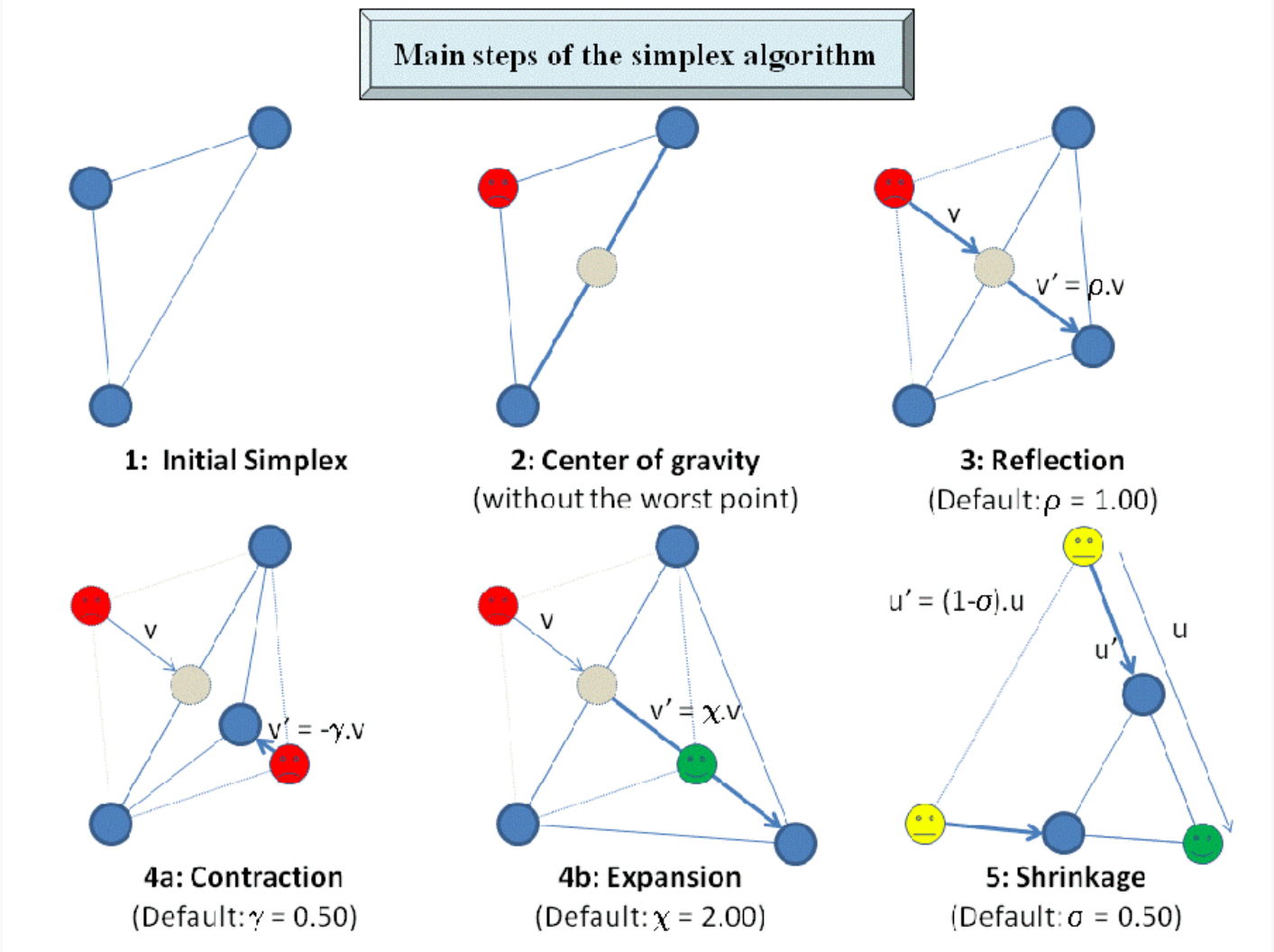
- Still deterministic, but...
- it requires a set of auxiliary solutions (not only one!) for generating a polyhedron.

Neldel-Mead (Simplex) Method

We refer to the popular version in [Lagarias et al., 1998].

- It works by linearly-combining $n+1$ points, forming a simplex;
- the simplex changes in size and shape being subject to transformations: *reflection*, *expansion*, *contraction* and *shrinkage* (see HERE).
- Vertices are sorted in descending order according to their fitness values to to identify the best point $\mathbf{x_l}$, the second worst $\mathbf{x_s}$ and the worst point $\mathbf{x_h}$ (highest fitness value).
- The centroid is then calculated $\overline{\mathbf{x}} \leftarrow \frac{1}{n} \sum_{i \neq h} \mathbf{x_i}$
- Transformations are applied to generate new points for replacing $\mathbf{x_h}$ (according to the logic reported in the pseudo-code).

Nelder-Mead:
working principle.



Nelder-Mead
Implementation details:

Nelder-Mead Algorithm pseudo-code

$x_0 \leftarrow$initial guess
for $i = 1 : n$ **do**
  $x_i \leftarrow x_0 + h_i e_i$ ▷ e_i: i^{th} axis versor
end for
while *stop condition not met*[a] **do**
  $x_l \leftarrow arg\{min_i f(x_i)\}$
  $x_h \leftarrow arg\{max_i f(x_i)\}$
  $x_s \leftarrow arg\{max_i f\left(x_{i \neq h}\right)\}$
  $\bar{x} \leftarrow \frac{1}{n} \sum_{i \neq h} x_i$ ▷ Centroid
  $x_r \leftarrow \bar{x} + \alpha(\bar{x} - x_h)$
  if $f(x_l) \leq f(x_r) < f(x_s)$ **then**
    $x_h \leftarrow x_r$
  else if $f(x_r) < f(x_l)$ **then**
    $x_e \leftarrow \bar{x} + \gamma(x_r - \bar{x})$
    if $f(x_e) < f(x_r)$ **then**
      $x_h \leftarrow x_e$
    else
      $x_h \leftarrow x_r$
    end if
  else if $f(x_s) \leq f(x_r) \leq f(x_h)$ **then**
    $x_c \leftarrow \bar{x} + \beta(x_r - \bar{x})$ ▷ Contraction
    if $f(x_c) \leq f(x_r)$ **then**
      $x_h \leftarrow x_c$
    else
      for $i = 0 : n$ **do**
        $x_i \leftarrow x_l + \delta(x_i - x_l)$ ▷ Shrinkage
      end for
    end if
  else if $f(x_r) \geq f(x_h)$ **then**
    $x_c \leftarrow \bar{x} + \beta(\bar{x} - x_h)$
    if $f(x_c) < f(x_h)$ **then**
      $x_h \leftarrow x_c$
    else
      for $i = 0 : n$ **do**
        $x_i \leftarrow x_l + \delta(x_i - x_l)$ ▷ Shrinkage
      end for
    end if
  end if
end while
Output x_l

[a] max budget or $\sqrt{\sum_{i \neq h} \frac{(f(x_i) - f(\bar{x}))^2}{n}} < 10^{-8}$ [Nelder and Mead, 1965]

Comparative considerations

- The three algorithms do not require derivatives and do not require explicit analytical expression.
- Rosenbrock and Nelder-Mead memory footprint is proportionate to n^2, while S' memory footprint linearly grows with n!
- Rosenbrock and Nelder Mead move in the space along all the directions simultaneously (diagonal) while Hooke Jeeves (S) moves along one direction at once.
- Rosenbrock and Hooke Jeeves (S) have a mathematically proved convergence while Nelder Mead doesn't!

Rosenbrock and Hooke Jeeves (S) have "local properties"

they use an implicit information on the gradient including the stop criterion while Nelder Mead has a certain degree of "global properties" (it searches on a line and then re-starts)

Suggested Laboratory work:

- Apply the S algorithm, equipped with toroidal correction, to the four problems De Jong, Rastrigin, Schwefel, Michalewicz:
  - perform at least 30 runs;
  - test 10, 30 and 50 dimensional values and observe the scalability of the algorithm.

N.B. This is not compulsory...

unless you wish to pick the S algorithm to complete TASK 2. Feel free to pick any algorithm from this lecture, or from the next one, to complete TASK 2. Once you have chosen an optimiser, run the experiment described in this slide, and complete the last section of the "TASK 2" PDF file. Make sure you display data and comment on them adequately.

Discussion board

Make a contribution in the discussion board by sharing your answers to the following questions:

- S and ISPO share a similar idea (working principle), but have a different nature. What is the main difference between the two approaches? What do they have in common? what does this implicate?

N.B. Participation in the discussion board is worth 10% of the final mark.

References I

Auger, A. and Teytaud, O. (2010).
Continuous lunches are free plus the design of optimal optimization algorithms.
Algorithmica, 57(1):21–146.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., and Woodward, J. (2010).
Classification of hyper-heuristic approaches.
In *Handbook of Meta-Heuristics*, pages 449–468. Springer.

Caraffini, F. (2014).
Novel Memetic Computing Structures for Continuous Optimisation.
PhD thesis, De Montfort University, Leicetser, United Kingdom.
https://www.dora.dmu.ac.uk/xmlui/handle/2086/10629.

Caraffini, F., Iacca, G., Neri, F., and Mininno, E. (2012a).
The importance of being structured: A comparative study on multi stage memetic approaches.
In *Computational Intelligence (UKCI), 2012 12th UK Workshop on*, pages 1–8.

References II

Caraffini, F., Neri, F., Iacca, G., and Mol, A. (2012b).
Parallel memetic structures.
Information Sciences, 227(0):60–82.

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

Hooke, R. and Jeeves, T. A. (1961).
Direct search solution of numerical and statistical problems.
Journal of the ACM, 8:212–229.

Lagarias, J. C., Reeds, J. A., Wright, M. H., and Wright, P. E. (1998).
Convergence properties of the nelder-mead simplex method in low dimensions.
SIAM Journal on Optimization, 9:112–147.

Nelder, A. and Mead, R. (1965).
A simplex method for function optimization.
Computation Journal, Vol 7:308–313.

References III

Özcan, E., Bilgin, B., and Korkmaz, E. E. (2008).
A comprehensive analysis of hyper-heuristics.
Intelligent Data Analysis, 12(1):3–23.

Palmer, J. R. (1969).
An improved procedure for orthogonalising the search vectors in rosenbrock's and swann's direct search optimisation methods.
The Computer Journal, 12(1):69–71.

Poli, R. and Graff, M. (2009).
There is a free lunch for hyper-heuristics, genetic programming and computer scientists.
In Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., and Ebner, M., editors, *Genetic Programming*, volume 5481 of *Lecture Notes in Computer Science*, pages 195–207. Springer Berlin Heidelberg.

Powell, M. J. D. (1964).
An efficient method for finding the minimum of a function of several variables without calculating derivatives.
The Computer Journal, 7(2):155–162.

References IV

Rosenbrock, H. H. (1960).
An automatic method for finding the greatest or least value of a function.
The Computer Journal, 3(3):175–184.

Tseng, L.-Y. and Chen, C. (2008).
Multiple trajectory search for large scale global optimization.
In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3052–3059.

Wolpert, D. and Macready, W. (1997).
No free lunch theorems for optimization.
IEEE Transactions on Evolutionary Computation, 1(1):67–82.