Differential Evolution

Differential Evolution (DE)

generalities:

- Defined in [Storn and Price, 1995] to address a (Chebychev polynomial) fitting problem:
  - $\Rightarrow$ originally thought for real-values optimisation!
  - There are some discrete implementations but they are usually not as efficient as the real-valued version!
- Intermediate features between EAs and SI:
  - it borrows variation operators from EAs,
  - and selection operators from SI.

General DE scheme

Differential Evolution pseudo-code

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

Individuals selection

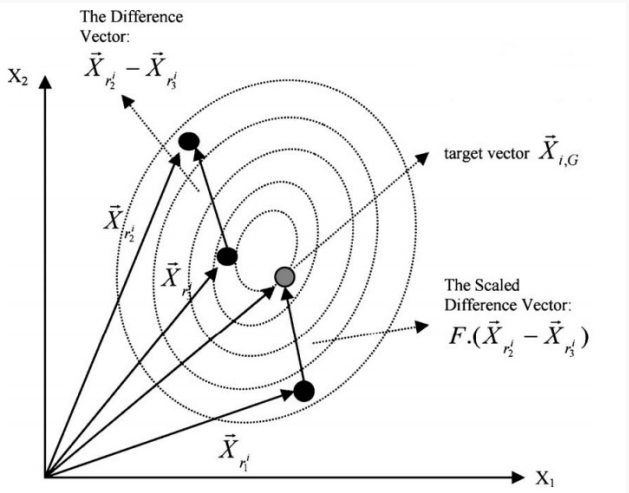
- individual are perturbed one at one;
- the mutation operator selects some individuals at random from the population by means of a uniformly distributed function to operate.
- Like in ES, there is no selection pressure for the choice of the parents undergoing variation operators (recombination and mutation).

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

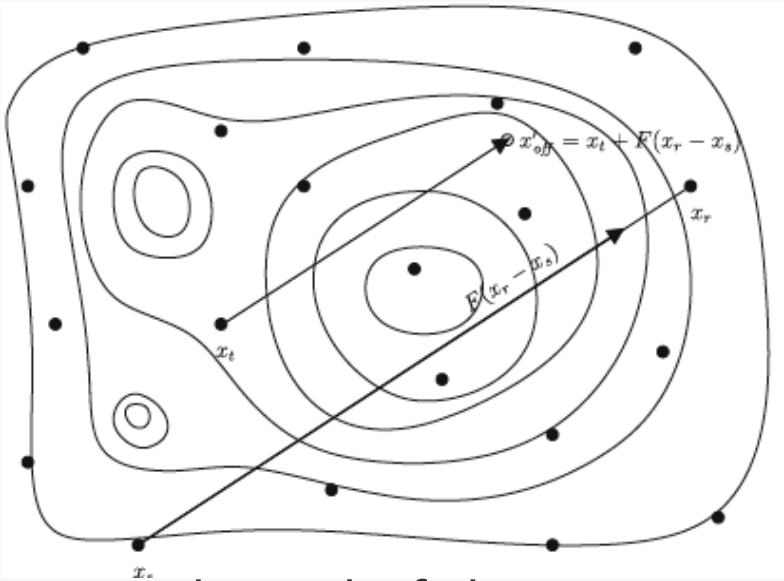
$$\mathbf{x_m} = \mathbf{x_{r_1}} + F\left(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}\right)$$

- 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.



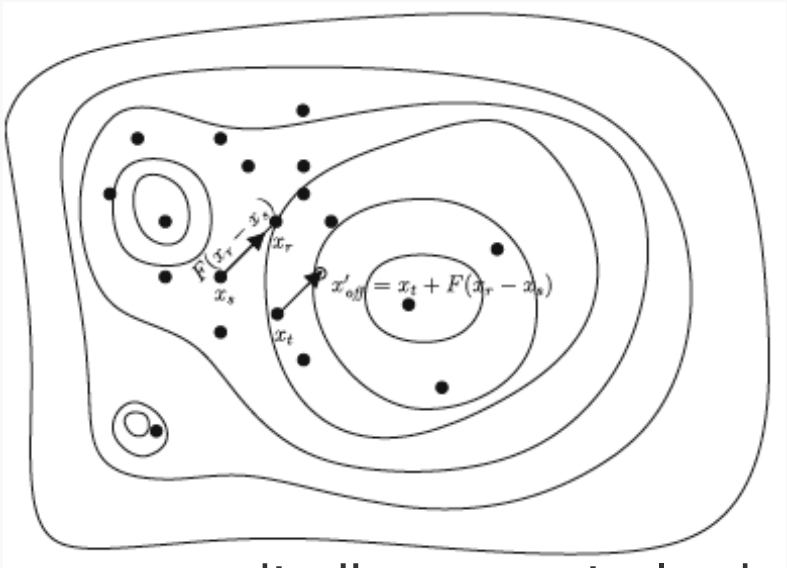
Original mutation

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 ($\forall x_i \in \mathbf{x_j} \exists\, CR$[1] probability to crossover);
  - exponential (a burst is copied with exp-decreasing probability).

```
procedure Binomial-XO(x_1, x_2)
    Index ← rand int [1, n]
    for i = 1 : n do
        if U(0, 1) ≤ CR || i == Index then
            x_off[i] ← x_1[i]
        else
            x_off[i] ← x_2[i]
        end if
    end for
    Output x_off
end procedure
```

```
procedure Exponential-XO(x_1, x_2)
    x_off ← x_2
    Index ← rand int [1, n]
    x_off[Index] ← x_2[Index]
    i ← Index + 1
    while U(0, 1) ≤ CR || i ≠ Index do
        x_off[i] ← x_1[i]
        i ← i + 1
        if i > n then
            i ← 1
        end if
    end while
    Output x_off
end procedure
```

[1] 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 $= 10 * D$
    (arguable, see [Parsopoulos, 2009] and
    [Kononova et al., 2015], and too big for large scale problems!)
- An interesting relation has been found between values for F and corresponding CR, and population population variance in [Zaharie, 2002].

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

Stagnation

- Amongst EAs, DE is particularly prone to stagnation:
  - the algorithm attempts checking similar solutions and "wonder around" in the decision space without detecting interesting areas.
  - the lack of "external" forces to produce new points can lead to configurations of stagnation.

How to improve upon classic DE

- We understood that we need to look after two aspects for improving DE:
  - parameters tuning;
  - stagnation, lack of convergence due to unsuccessful moves.
- This can be done by including explicitly some additional/alternative search operators:
  - combination of multiple classic mutations;
  - design of new additional mutations;
  - hybridisation with extra moves/algorithms (memetic computing variants [Caraffini et al., 2013]);
- By adapting the parameters to the problem at hand. e.g.:
  - parameters sampled from a pool of generally "good" values [Mallipeddi et al., 2010] ("ensemble" strategy);
  - meta-optimisation? (i.e. parameters optimisation)
  - tuning on the go;
  - randomisation of control parameters.

Self-adaptive DE (SaDE):

Adapting F, CR and multiple mutations to the problem.

- The original SaDE [Qin and Suganthan, 2005] makes use of two mutations[2]:
  - DE/rand/1/bin (explorative), activated with probability p;
  - DE/current-to-Best/2/bin (exploitative), activated with probability $1 - p$.
- Initially $p = 50\%$, then is adapted to the problem:

$$p = \frac{ns_1 \cdot (ns_2 + nf_2)}{ns_2 \cdot (ns_1 + nf_1) + ns_1 \cdot (ns_2 + nf_2)}$$

- number of successes (ns_i) and failures (nf_i) of the i^{th} mutation are recorded within learning periods of $LP = 50$ generations.

[2] A subsequent version using 4 mutations also exists [Qin et al., 2009].

Self-adaptive DE (SaDE):

Adapting F, CR and multiple mutations to the problem.

- The original SaDE [Qin and Suganthan, 2005] makes use of two mutations[3]:
  - DE/rand/1/bin (explorative), activated with probability p;
  - DE/current-to-Best/2/bin (exploitative), activated with probability $1 - p$.
- Initially $p = 50\%$, then is adapted to the problem:

$$p = \frac{ns_1 \cdot (ns_2 + nf_2)}{ns_2 \cdot (ns_1 + nf_1) + ns_1 \cdot (ns_2 + nf_2)}$$

- number of successes (ns_i) and failures (nf_i) of the i^{th} mutation are recorded within learning periods of $LP = 50$ generations.
- Mutations use new $F \sim \mathcal{N}(0.5, 0.3)$ and $CR \sim \mathcal{N}(0.5, CR_m)$ every 5 iterations;
  - initially $CR_m = 0.5$, then replaced with the median value of the previous 25 successful CR_m values.

[3] A subsequent version using 4 mutations also exists [Qin et al., 2009].

Novel operators for DE

- Most DE variants make use of combination, or slight modifications, of classic operators:
  - SaDE [Qin et al., 2009] is an example making use of 4 standard mutations;
  - the literature is vast and full of small variants!
- Some novel operator have been proposed as well:
  - for introducing new mutations;
  - for optimising F;
  - for new crossover methods.
- Let us see few significant examples.

Trigonometric mutation (TDE)

to speed up convergence and help prevent stagnation

- In [Fan and Lampinen, 2003] DE/rand/1 and a more exploitative "trigonometric" mutation are randomly alternated according to a prefixed probability ϕ and $1-\phi$ respectively.
- Three randomly selected points (with indexes $r_1 \neq r_2 \neq r_3$) are required to work out the weight coefficients: p_i:

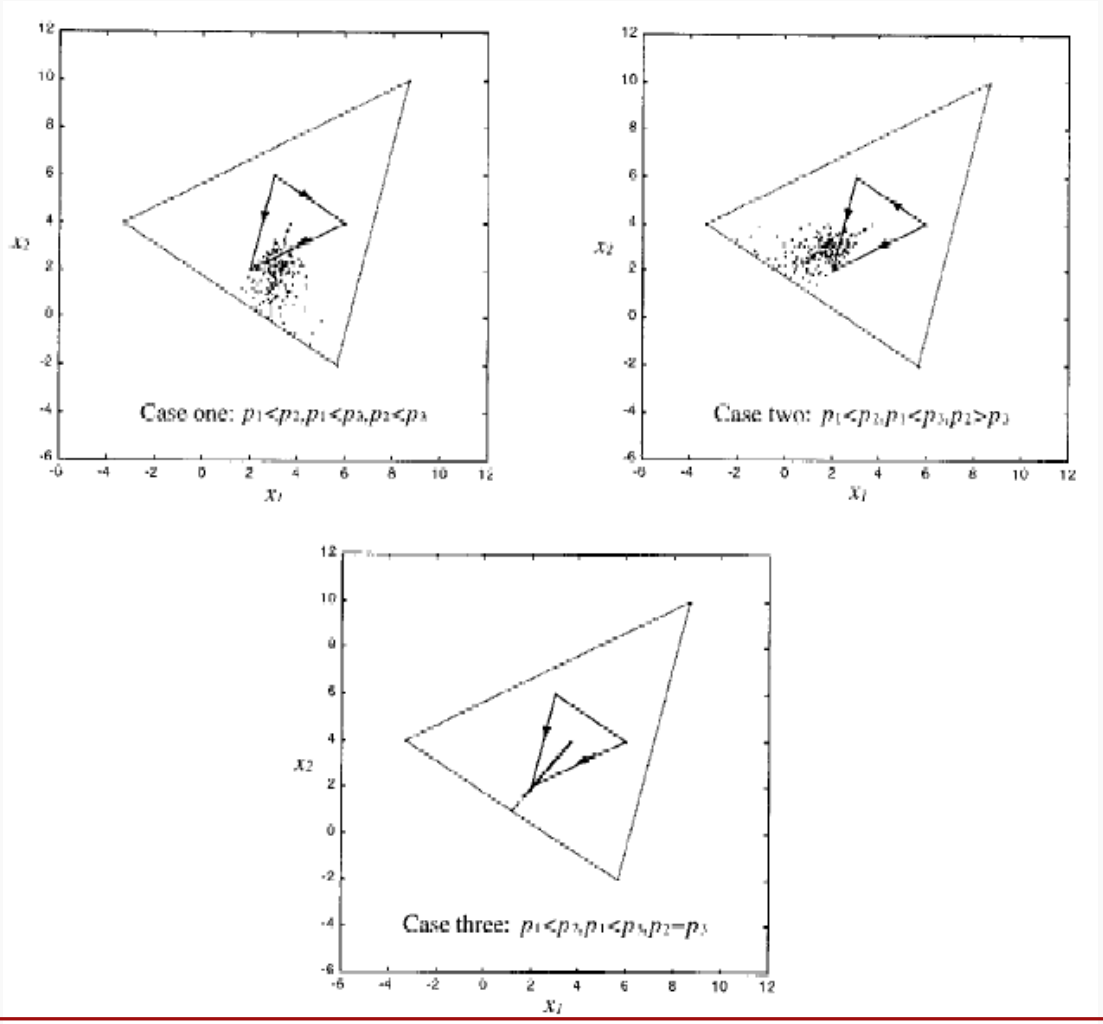
$$p_i = \frac{|f(\mathbf{x_{r_i}})|}{|f(\mathbf{x_{r_1}})| + |f(\mathbf{x_{r_2}})| + |f(\mathbf{x_{r_3}})|}, \quad i = 1, 2, 3$$

- and used them to work out the mutant vector:

$$\mathbf{x_m} = \frac{(\mathbf{x_{r_1}} + \mathbf{x_{r_2}} + \mathbf{x_{r_3}})}{3} +$$
$$(p_2 - p_1)(\mathbf{x_{r_1}} - \mathbf{x_{r_2}}) + (p_3 - p_2)(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}) + (p_1 - p_3)(\mathbf{x_{r_3}} - \mathbf{x_{r_1}})$$

TDE: graphical representation
(of the mutation operator)

- The trigonometric mutation can be seen as a local search within the triangular region ($2D$ and $3D$) formed by the 3 points.



Simplex-DE (SDE)

accelerating DE integrating LS in the crossover

- Multiparents crossover hybridised with the Simplex method to provide the best offspring and accelerate convergence: [Noman and Iba, 2008].
- if Simpelx-XO is successful then is repeated:
  - for a fixed amount of times;
  - or until $f(\mathbf{x_{off}})$ decreases.

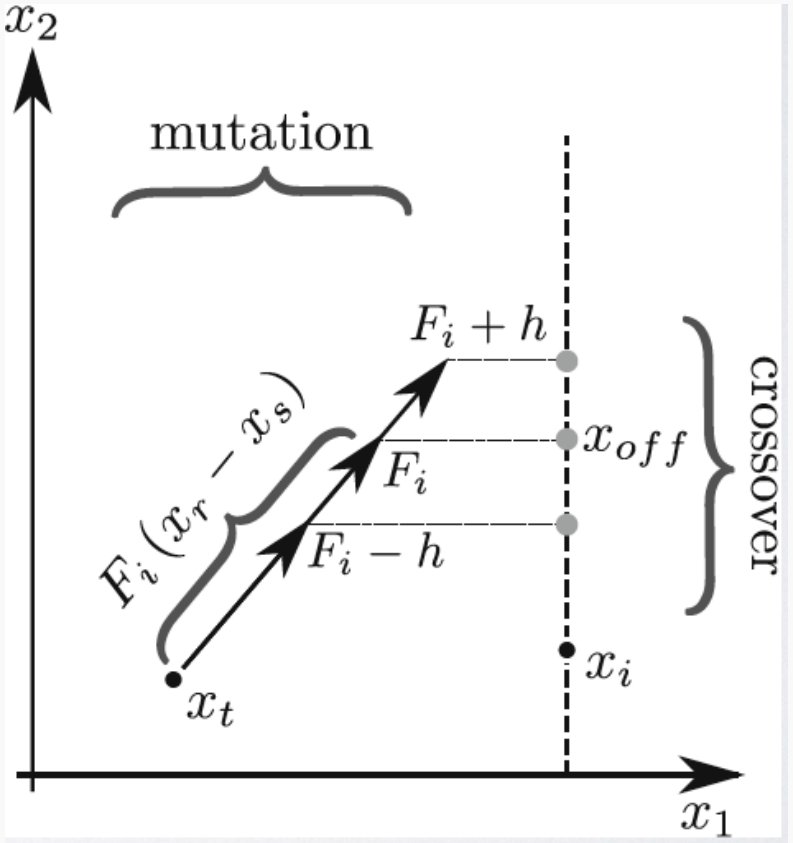
---

procedure Simplex-XO(r_{best}, **Population**)
  Simplex $\leftarrow \mathbf{x_{r_{best}}}$ **from Population**
  Simplex $\leftarrow$ *n-2 randomly selected* $\mathbf{x_{r_i}}$ $(r_i \neq r_{best})$ *from* **Population**
  work out centroid $\bar{\mathbf{x}} \leftarrow \frac{1}{n} \sum_{\forall \mathbf{x} \in \mathbf{Simplex}} \mathbf{x}$
  for $i = 1 : n-1$ **do**
    $\mathbf{r}[i] = \mathcal{U}(0,1)^{\frac{\mathbf{1}}{\mathbf{1}+i}}$
  end for
  for $\forall \mathbf{x_i} \in$ **Simplex do**
    generate a corresponding vector:
    $\mathbf{y_i} \leftarrow \bar{\mathbf{x}} + \alpha(\mathbf{x_i} - \bar{\mathbf{x}})$ $\quad \triangleright\ \alpha \approx 1$: Simplex expansion weight
  end for
  $\mathbf{x_{off}} = \emptyset$
  for $i = 2 : n$ **do**
    $\mathbf{x_{off}} \leftarrow \mathbf{r}[i-1](\mathbf{y_{i-1}} - \mathbf{y_i} + \mathbf{x_{off}})$
  end for
  Output $\mathbf{x_{off}} \leftarrow \mathbf{x_{off}} + \mathbf{y_n}$
end procedure

- It requires at least n individuals! $\Rightarrow$ Large scale ✗

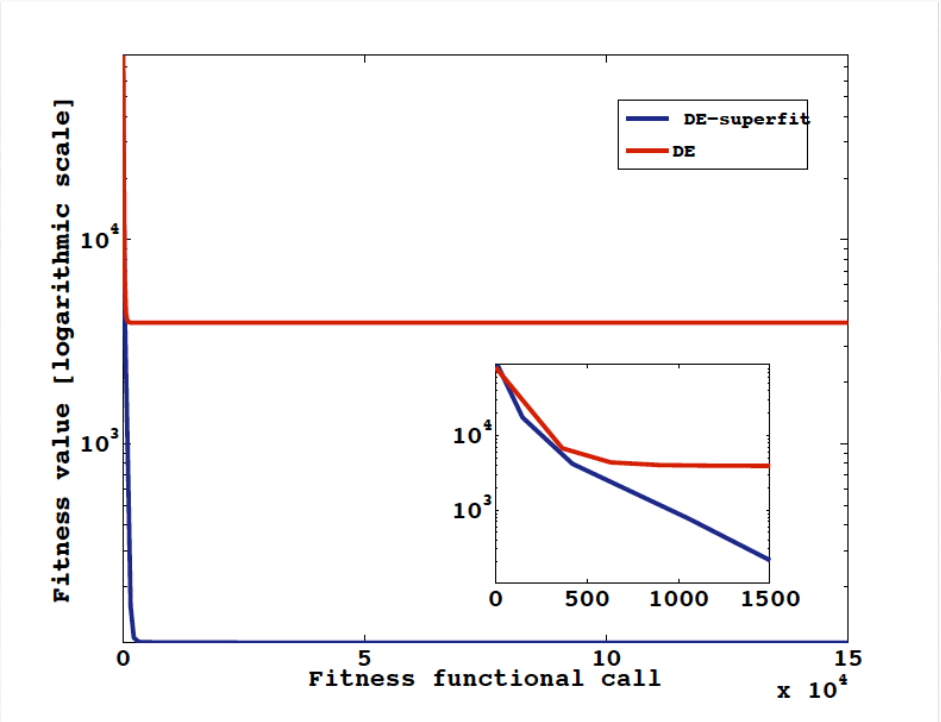
Enhancing DE via by scale factor local search

- A line minimisation algorithm is used to tune F and provide a better offspring [Tirronen et al., 2009].
- Optimising 1 variable requires a modest computational budget (optimising the mutant would be budget consuming).



Super-fit DE

- Unlike GAs, if one individual displays a much better performance than the average of the population, then DE works very efficiently [Caponio et al., 2009].
- One initial candidate solution can be "drugged" by a local search or other algorithm before the DE application, to speed up convergence.

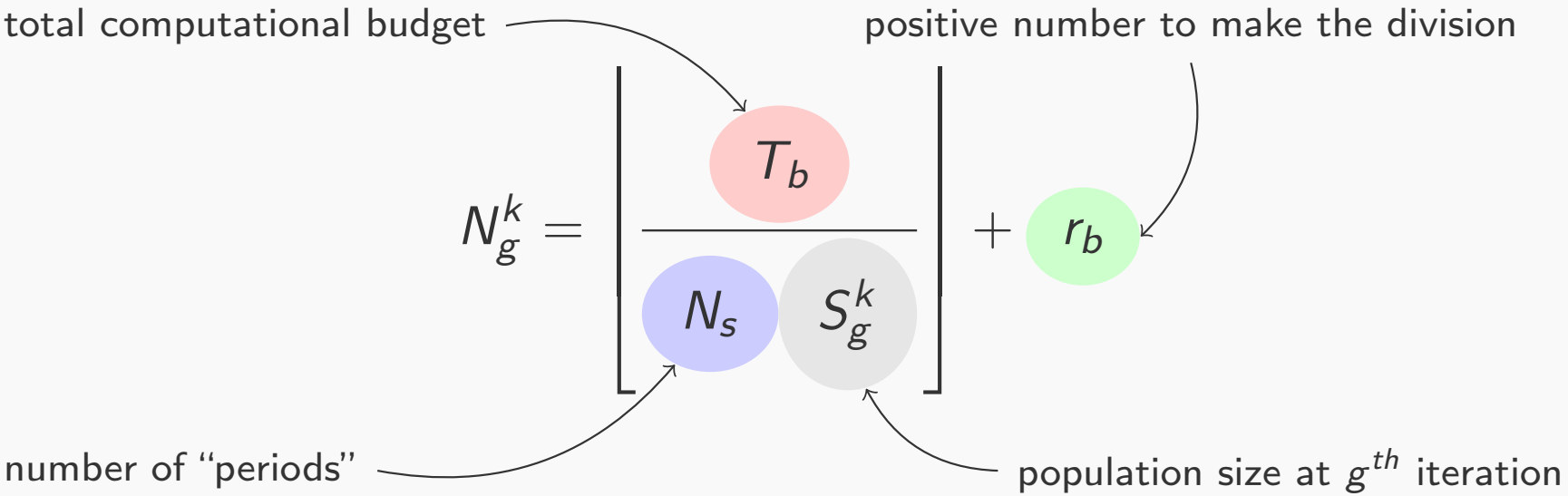


Improving DE by grouping the individuals

- Many papers propose variations in the pool of search moves in order to increase the exploitation of the algorithm;
- this action mitigates the stagnation and promotes quick (exploitative) improvements.
- The third parameter that can be modified to improve upon DE performances is the population size!

Population size reduction
[Brest and Maučec, 2008]

- The population size in this DE variant is variable and reduced every N_g^k generations:



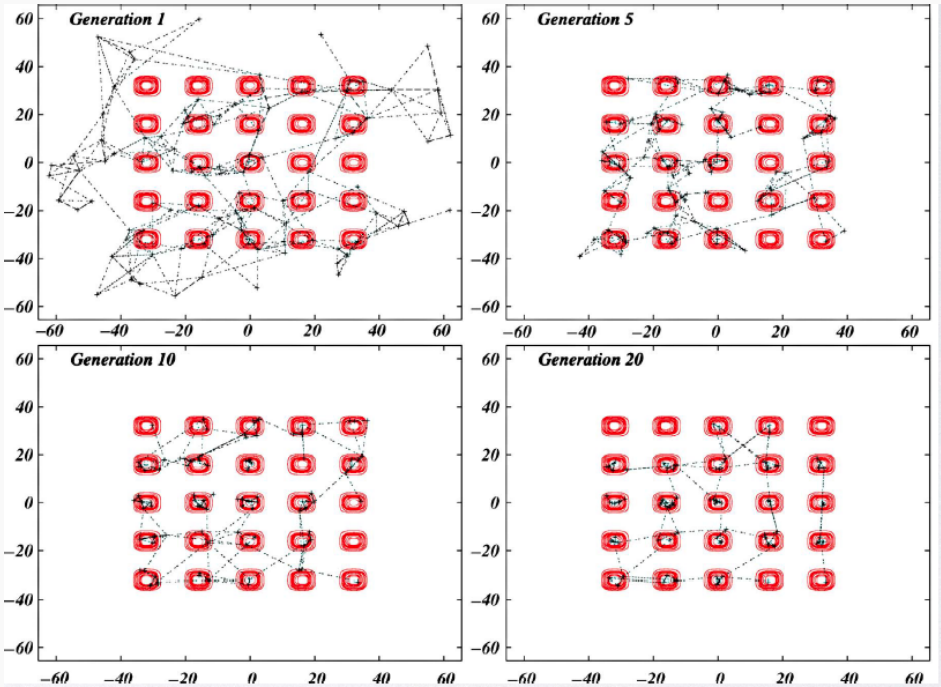
- The population is halved by applying the one-to-one spawning between two random partitions of the population.
- The population reduction aims at focusing the search in progressively smaller search spaces in order to inhibit stagnation.

Structuring the population in islands
(Paralle DE [Tasoulis et al., 2004])

- The DE population gets structured in sub-populations (islands);
- each island exploits a subset of search moves.
- Designed for a parallel implementation where each sub-population is evolved on a different core:
  - sub-populations can exchange information.

Proximity mutation
[Epitropakis et al., 2011]

- The selection of the individuals undergoing mutation is based on the spacial distribution of the solutions:
  - close individuals are more likely to be selected.
- The proximity mutation promotes a local exploitation.
- Can be onerous: euclidean distance between individuals has to be worked out.



Randomisation: dither and jitter

[Das et al., 2005]

- Surprisingly, convergence can be accelerated by simply randomising F.
- Two simple but efficient variants using a fixed CR ($CR = 0.9$ is suggested) and a random F are:
  - *dither DE*: a new $F = 0.5\,(1 + \mathcal{U}\,(0, 1))$ is sampled after every generation;
  - *jitter-DE*: a new $F = 0.5\,(1 + \mathcal{U}\,(0, 1))$ is sampled for each individual.

Controlled randomisation: jDE

[Brest et al., 2006]

- Changing F too many times can make DE behaves like a random search.
- In jDE, both F and CR are randomised, and have a (small) probability to get re-sampled:
  - this allows successfully configurations of $\langle \mathbf{x}, F, CR \rangle$ to be re-used for a number of times before trying different F and CR for the solution $\mathbf{x}$.
- Specifically, jDE is a basic DE/rand/1/bin;
- before generating a mutant
  - if $\mathcal{U}(0,1) < \tau_1$ then a new $F = F_l + \mathcal{U}(0,1) \cdot F_u$ is sampled;
  - if $\mathcal{U}(0,1) < \tau_2$ then a new $CR = \mathcal{U}(0,1)$ is also sampled.
- In [Brest et al., 2006] $\tau_1 = \tau_2 = 0.1$, see the original paper for more details.
- This system, despite its simplicity, led to excellent results in low and high dimensionality values.

Sampling from different distributions

- Other DE schemes sample the control parameters from non-uniform distributions
  - e.g. SaDE has randomised parameters from Gaussian distributions.
- JADE instead, see [Zhang and Sanderson, 2009], makes also use of Cauchy distribution, which is more appropriate for CR:

$$F_j = \mathcal{C}(\mu_F, 0.1) \quad \text{and} \quad CR_j = \mathcal{N}(\mu_{CR}, 0.1)^{a}, \quad \forall j^{th} \text{ individual}$$

- randomisation is controlled by updating

$$\mu_F = (1-c)\cdot\mu_F + c\cdot\frac{\sum_{F\in\mathbf{S_F}} F^2}{\sum_{F\in\mathbf{S_F}} F}, \qquad \mu_{CR} = (1-c)\cdot\mu_{CR} + c\cdot\frac{\sum_{CR\in\mathbf{S_{CR}}} CR}{|\mathbf{S_{CR}}|}$$

  after every generation, where $\mathbf{S_F}$ and $\mathbf{S_{CR}}$ are memory stacks filled up with succe values of F and CR, $c \in [0,1]$.

- JADE features the DE/current-to-pbest/1 mutation, which uses an extra archive
  - $\mathbf{x_m} = \mathbf{x_j} + F_j\left(\mathbf{x}^{\mathbf{p\%}}_{\mathbf{best}} - \mathbf{x_j}\right) + F_j\left(\mathbf{x_{r_1}} - \mathbf{x_{r_2}}\right)$, see [Zhang and Sanderson, 2009] for details.

[a] Unacceptable values like $F_j = 0$ and $CR_j \notin [0,1]$ are discarded.

More DE variants

The literature is really vast, some other papers you may want to read are:

- Opposition based DE [Rahnamayan et al., 2008]:
  - after each generation there is probability of "mirroring" some solutions.
- Compact DE [Mininno et al., 2011], [Iacca et al., 2012]:
  - individuals are sampled from an evolving distribution!

    (click HERE)
- MDE-pBX [Islam et al., 2012]:
  - an adaptive DE with novel mutation and crossover strategies.
- Multi-criteria DE [Cheng et al.,]:
  - Parameters are adapted to the problem via internal multi-objective optimisation approaches.

Summary: DE solutions

- The modern DE variants in the literature tend to:
  - integrate additional (and multiple) search moves within the DE framework;
  - exploit the available search moves in order to quickly (before budget exhaustion) obtain improvements;
  - remove the burden of tuning F and CR.

Laboratory and participation work:

- Implement a DE/rand/1/exp, tune the parameters for solving the four problems under consideration in $10D$, $50D$, and $100D$ (10 runs are fine for parameter tuning);
- Implement jDE and test it on the four problems under consideration in this module for $10D$, $50D$, $100D$ (at least 30 runs), use for the parameters the original setting of the paper [Brest et al., 2006] (use the paper in addition to the slides).
- Test your tuned DE/rand/1/exp against two jDE variants, bin and exp, then selecting the optimiser displaying the best performance.

References I

Brest, J., Greiner, S., Bošković, B., Mernik, M., and Žumer, V. (2006).
Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems.
IEEE Transactions on Evolutionary Computation, 10(6):646–657.

Brest, J. and Maučec, M. S. (2008).
Population size reduction for the differential evolution algorithm.
Applied Intelligence, 29(3):228–247.

Caponio, A., Neri, F., and Tirronen, V. (2009).
Super-fit control adaptation in memetic differential evolution frameworks.
Soft Computing-A Fusion of Foundations, Methodologies and Applications, 13(8):811–831.

Caraffini, F., Neri, F., and Poikolainen, I. (2013).
Micro-differential evolution with extra moves along the axes.
In *Differential Evolution (SDE), 2013 IEEE Symposium on*, pages 46–53. IEEE.

References II

Cheng, J., Zhang, G., Caraffini, F., and Neri, F.
Multicriteria adaptive differential evolution for global numerical optimization.
Integrated Computer-Aided Engineering.

Das, S., Konar, A., and Chakraborty, U. K. (2005).
Two improved differential evolution schemes for faster global search.
In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 991–998. ACM.

Epitropakis, M. G., Tasoulis, D. K., Pavlidis, N. G., Plagianakos, V. P., and Vrahatis, M. N. (2011).
Enhancing differential evolution utilizing proximity-based mutation operators.
Trans. Evol. Comp, 15(1):99–119.

Fan, H.-Y. and Lampinen, J. (2003).
A trigonometric mutation operation to differential evolution.
Journal of Global Optimization, 27(1):105–129.

References III

Iacca, G., Caraffini, F., and Neri, F. (2012).
Compact differential evolution light: high performance despite limited memory requirement and modest computational overhead.
Journal of Computer Science and Technology, 27(5):1056–1076.

Islam, S., Das, S., Ghosh, S., Roy, S., and Suganthan, P. (2012).
An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimizationcriteri.
Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 42(2):482–500.

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.

References IV

Mallipeddi, R., Mallipeddi, S., and Suganthan, P. N. (2010).
Ensemble strategies with adaptive evolutionary programming.
Information Sciences, 180(9):1571–1581.

Mininno, E., Neri, F., Cupertino, F., and Naso, D. (2011).
Compact differential evolution.
IEEE Transactions on Evolutionary Computation, 15(1):32–54.

Noman, N. and Iba, H. (2008).
Accelerating differential evolution using an adaptive local search.
IEEE Transactions on Evolutionary Computation, 12(1):107–125.

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.

Qin, A. K., Huang, V. L., and Suganthan, P. N. (2009).
Differential evolution algorithm with strategy adaptation for global numerical optimization.
IEEE Transactions on Evolutionary Computation, 13:398–417.

References V

Qin, A. K. and Suganthan, P. N. (2005).
Self-adaptive differential evolution algorithm for numerical optimization.
In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 2, pages 1785–1791.

Rahnamayan, S., Tizhoosh, H. R., and Salama, M. M. (2008).
Opposition-based differential evolution.
IEEE Transactions on Evolutionary Computation, 12(1):64–79.

Storn, R. and Price, K. (1995).
Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces.
Technical Report TR-95-012, ICSI.

Tasoulis, D. K., Pavlidis, N. G., Plagianakos, V. P., and Vrahatis, M. N. (2004).
Parallel differential evolution.
In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2023–2029.

References VI

Tirronen, V., Neri, F., and Rossi, T. (2009).
Enhancing differential evolution frameworks by scale factor local search - part I.
In *Proceedings of the IEEE Congress on Evolutionary Computation*. to appear.

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

Zhang, J. and Sanderson, A. C. (2009).
Jade: Adaptive differential evolution with optional external archive.
IEEE Transactions on Evolutionary Computation, 13(5):945–958.