DEEC
DEPARTAMENTO DE ENGENHARIA
ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

Applied Computational Intelligence MEEC/MECD (2022/2023 – 1º Sem)

Evolutionary Optimization

Prof. Nuno Horta

OUTLINE

Basic Ideas

- **In evolutionary algorithms** used for optimization, or in other words evolutionary optimization, **the problem to be solved must be well defined**. Any possible solution to the problem must be comparable to another possible solution.

- **Most often**, the comparisons between two or more candidate solutions are based on **quantitative measures** of how well a proposed solution meets the needs of the problem

- **The use of qualitative or even fuzzy descriptors** of measures of fitness are most often found in what's called **interactive evolutionary computation**, in which a human provides a judgment about the quality of proposed solutions.

- **This chapter focuses on quantitative evolutionary optimization**, in which there is a numeric description – a function - that operates on a potential solution and returns either a single real number or multiple numbers that describe the value of the solution.

Basic Ideas

- Within evolutionary optimization, as with all engineering, there are essentially **two forms of optimization problem**.

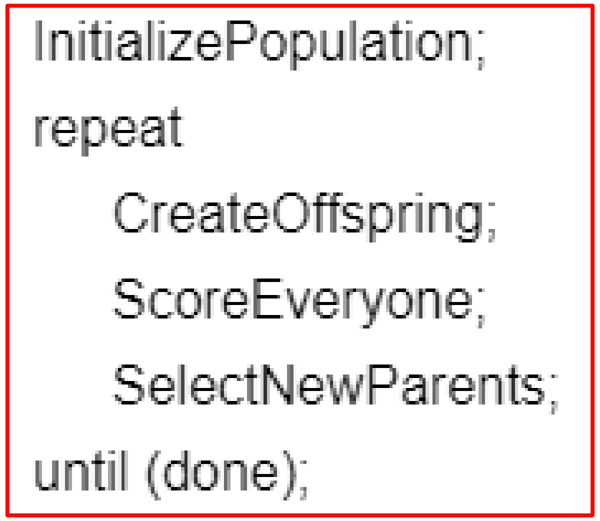
  - **numeric**: In this case you are looking for a point, a solution, in the search space. The solution space can be used as a measure of solution quality (lower is better in a minimization problem).

  - **combinatoric**: In this case, you are looking for a combination of items that can be listed. Depending on the problem the order in the list may matter.

- **This chapter** provides an **introduction to both numeric and combinatorial evolutionary optimization**. It also describes some of the **mathematical properties of representation and selection operators**, and of evolutionary algorithms broadly. Some important **extensions of the basic application of evolutionary algorithms for optimization** are also covered, including **handling constraints** and **allowing the evolutionary algorithm to learn how to optimize its own search parameters** in a process called **self-adaptation**.

Global Numerical Optimization

- Canonical Example in **One Dimension** (1)
  - Consider a one-dimension **black box optimization (BBO) problem**, i.e., you don’t know the analytical description of the function, but you can obtain the value of f(x).
  - The **pseudocode for implementing an EA** is as follows:



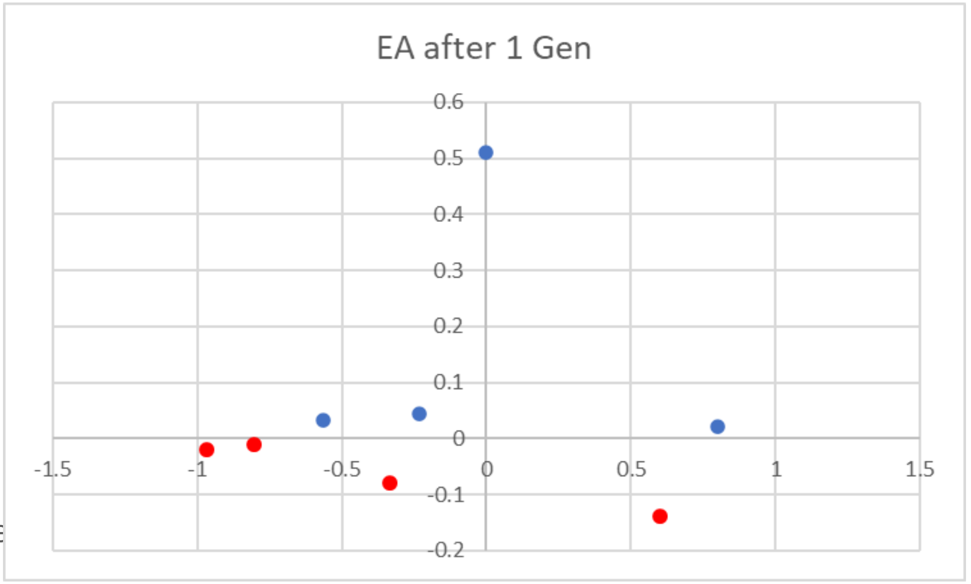
```
InitializePopulation;
repeat
    CreateOffspring;
    ScoreEveryone;
    SelectNewParents;
until (done);
```

Global Numerical Optimization

- Canonical Example in One Dimension (2)
  - Consider a **population of** μ candidate solutions, $\mathbf{x_1, \ldots x_\mu}$ **parents**
  - **Create the initial population**
    - In the case of a BBO **generating the parents at random** using an **uniform distribution** makes sense **as you don´t have any hint about the function**.
    - Assuming, we want the algorithm to search for a minimum in the domain $x \in [-1.5,1.5]$

```
InitializePopulation;
repeat
    CreateOffspring;
    ScoreEveryone;
    SelectNewParents;
until (done);
```

```
i = 0;
repeat
    i = i + 1;
    x[i]= U(-100,100);
until (i == μ);
```



Global Numerical Optimization

- Canonical Example in One Dimension (3)
  - **Create Offsprings**
    - Creating λ offsprings $x_{\mu+1}, \ldots x_{\mu+\lambda}$
    - Assume, the offsprings are generated applying a random variation to parents, for example, using a standard Gaussian distribution. For simplicity consider $\lambda = \mu$

```
InitializePopulation;
repeat
    CreateOffspring;
    ScoreEveryone;
    SelectNewParents;
until (done);
```

```
i = 0;
repeat
    i = i + 1;
    x[μ + i] = x[i] + N(0,1);
until (i == μ);
```

Global Numerical Optimization

- Canonical Example in One Dimension (4)

  – At this stage we have 2μ random candidates

  – **Scoring the Population**

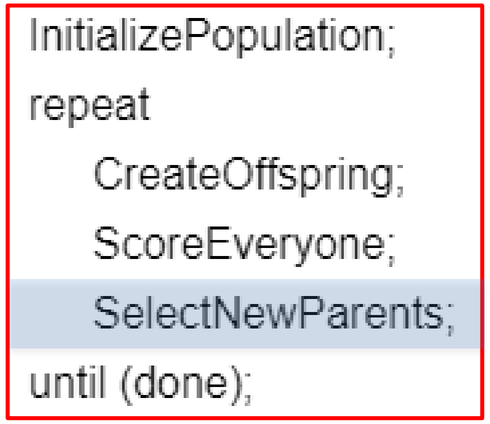
    - Score each element (**only elements not scored before**)

```
InitializePopulation;
repeat
    CreateOffspring;
    ScoreEveryone;
    SelectNewParents;
until (done);
```

```
i = 0;
repeat
    i = i + 1;
    score_x[i] = f(x[i]);
until (i == 2μ);
```

Global Numerical Optimization

- Canonical Example in One Dimension (5)
  - **Select New Parents**
    - **Rank the population** based on the score of each element
    - **Select the μ best-ranking solutions** to become new parents for the next generation.
  - **When should we stop?**
    - If the best solution is below a pre-defined threshold
    - If the best solution is not evolving for a pre-defined time (number of generations)



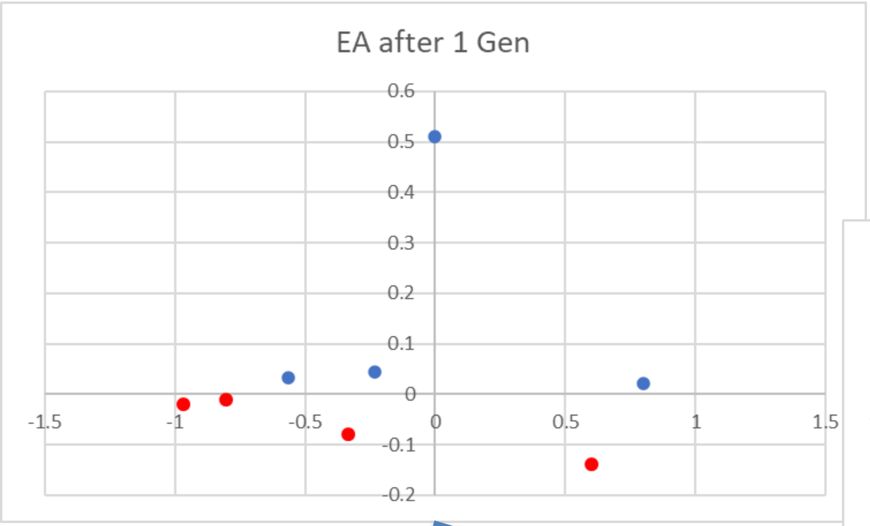
```
InitializePopulation;
repeat
    CreateOffspring;
    ScoreEveryone;
    SelectNewParents;
until (done);
```

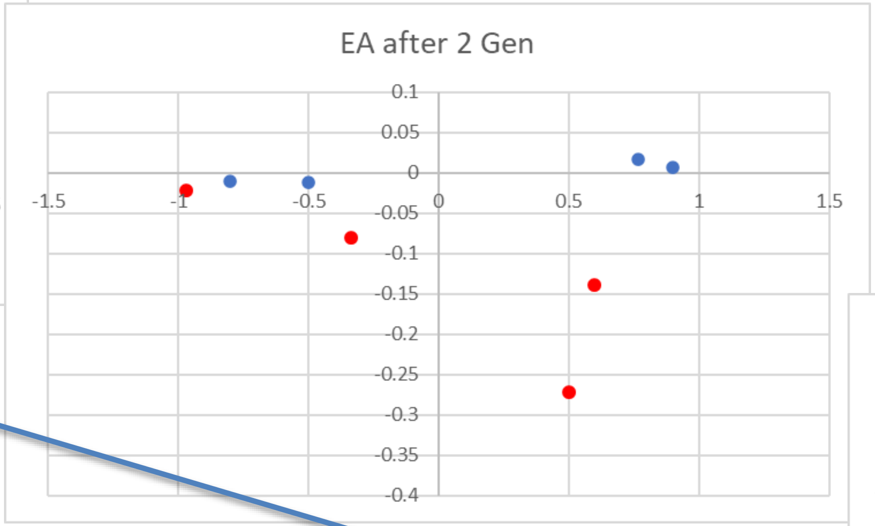
DEEC
DEPARTAMENTO DE ENGENHARIA
ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

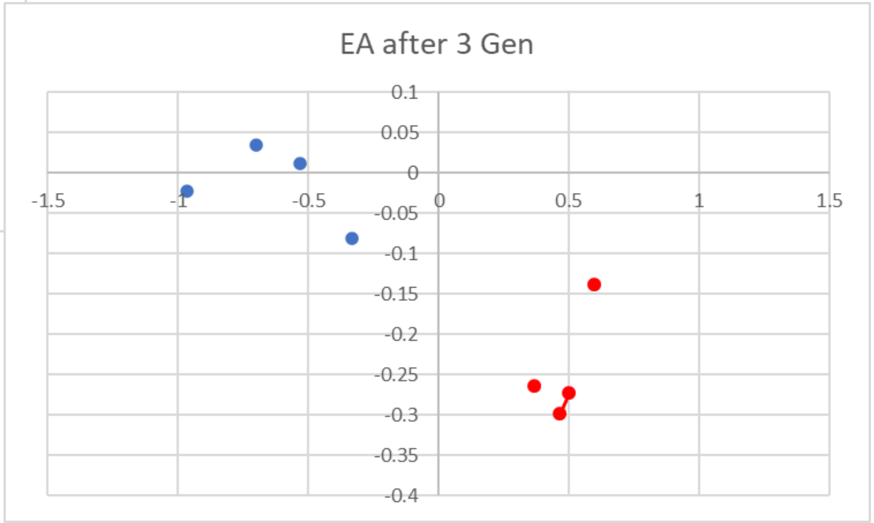
```
InitializePopulation;
repeat
    CreateOffspring;
    ScoreEveryone;
    SelectNewParents;
until (done);
```

Global Numerical Optimization

- Canonical Example in One Dimension (6)
  - **Population of 8 elements**
  - **4 selected for parents**
  - **4 offsprings generated for each new generation**







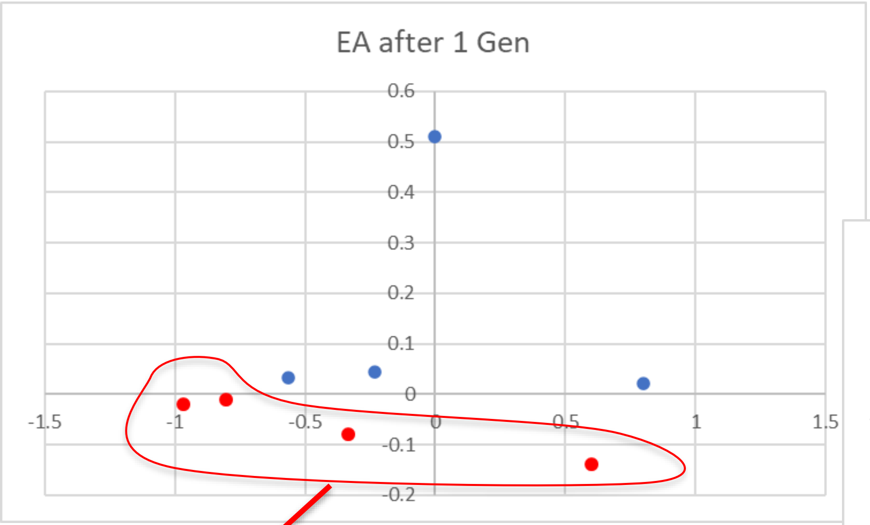
Evolution is made by considering the best fitted to generate the offsprings. The best elements converge to the BB function minimum.

Global Numerical Optimization

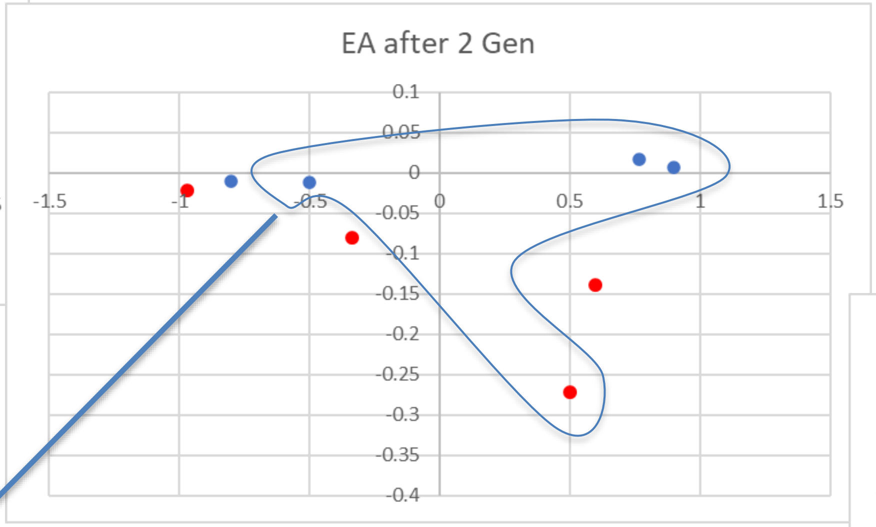
```
InitializePopulation;
repeat
    CreateOffspring;
    ScoreEveryone;
    SelectNewParents;
until (done);
```

- Canonical Example in One Dimension (7)

- **Population of 8 elements**
- **4 selected for parents**
- **4 offsprings generated for each new generation**

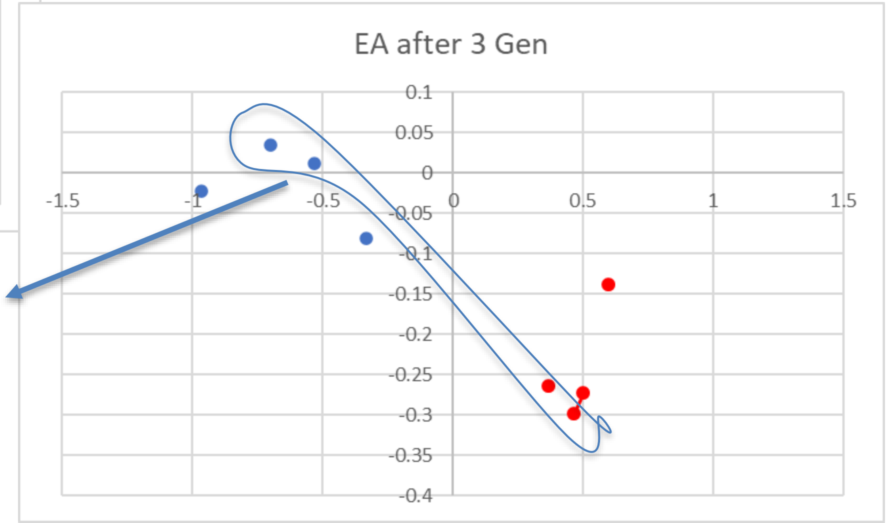


Parents G1



Offsprings from Parents G1

Parents G2



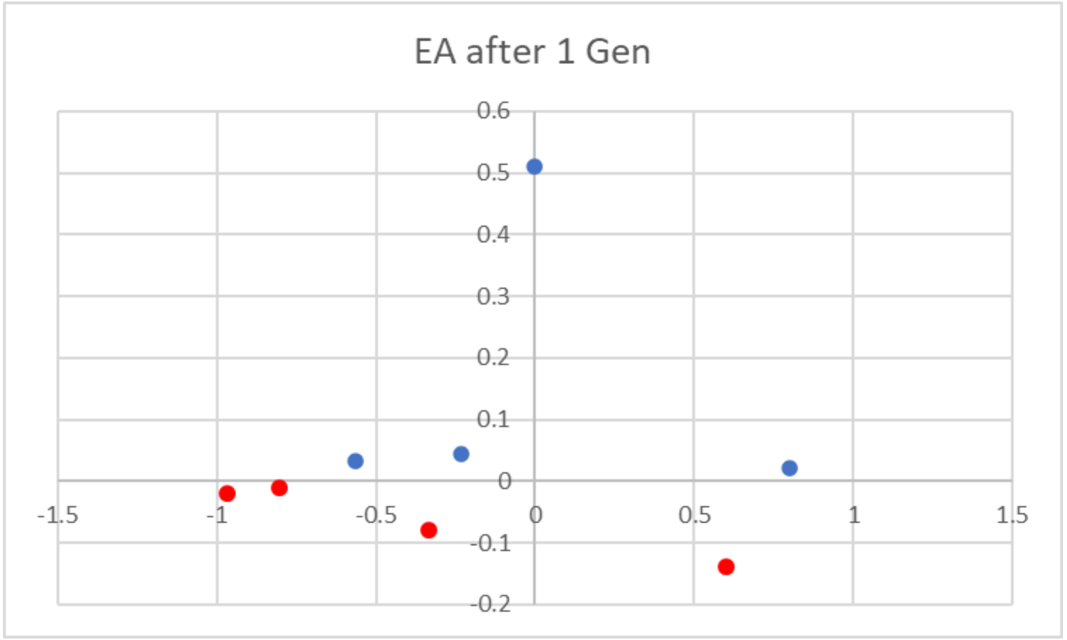
Offsprings from Parents G2

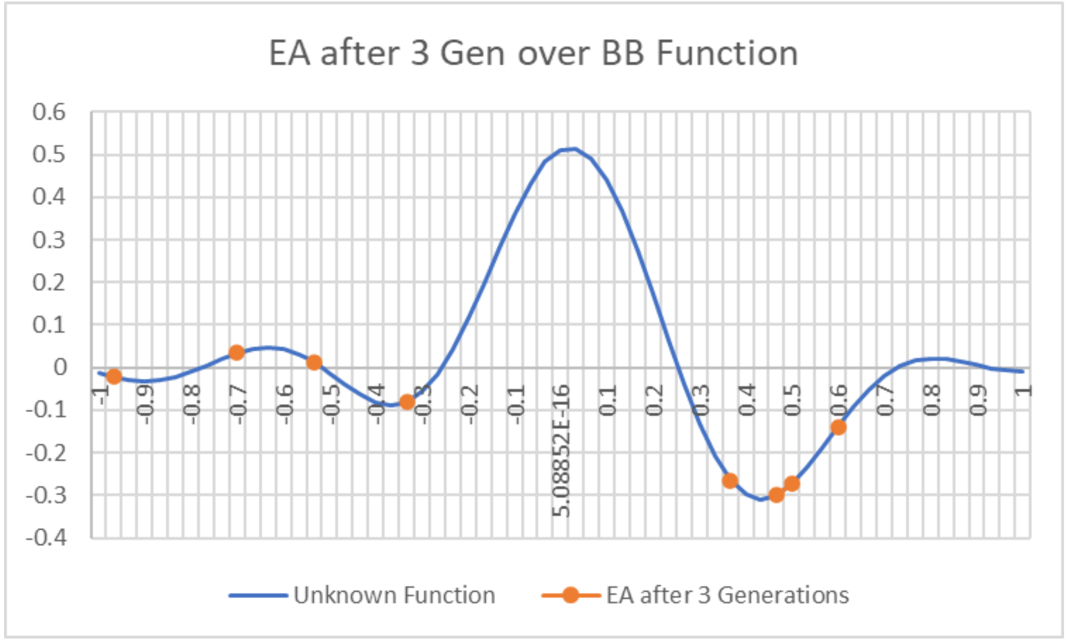
Parents G3

DEEC
DEPARTAMENTO DE ENGENHARIA
ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

Global Numerical Optimization

- Canonical Example in One Dimension (8)
  - **The evolution towards the global minimum is clear after a few generations**





Global Numerical Optimization

- Canonical Example in **Two or More Dimensions** (1)
  - Consider a **n dimension black box optimization problem**, i.e., you don't know the analytical description of the function, but you can obtain the value of f(X), where X is a vector of n variables.
  - The pseudocode for implementing an EA is as follows:

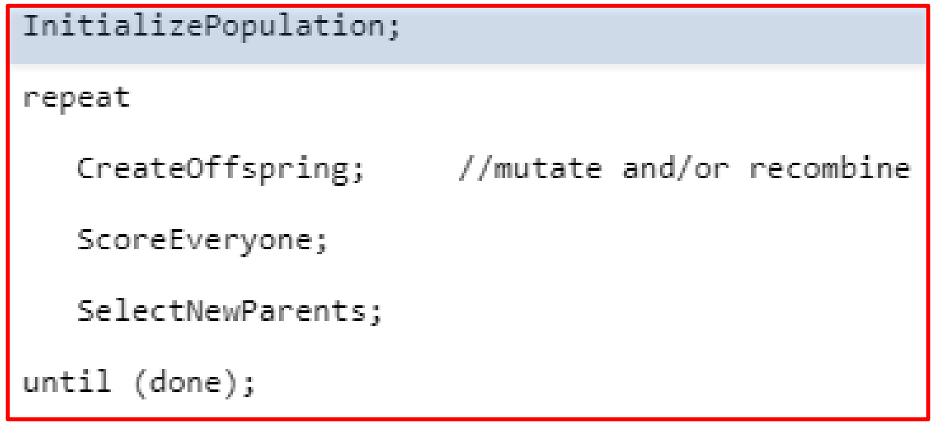
```
InitializePopulation;
repeat
   CreateOffspring;     //mutate and/or recombine
   ScoreEveryone;
   SelectNewParents;
until (done);
```

Suggested homework: Create an illustrative example, similar to the one presented above, but now for the 2-dimensional BBO case.

DEEC
DEPARTAMENTO DE ENGENHARIA
ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

Global Numerical Optimization

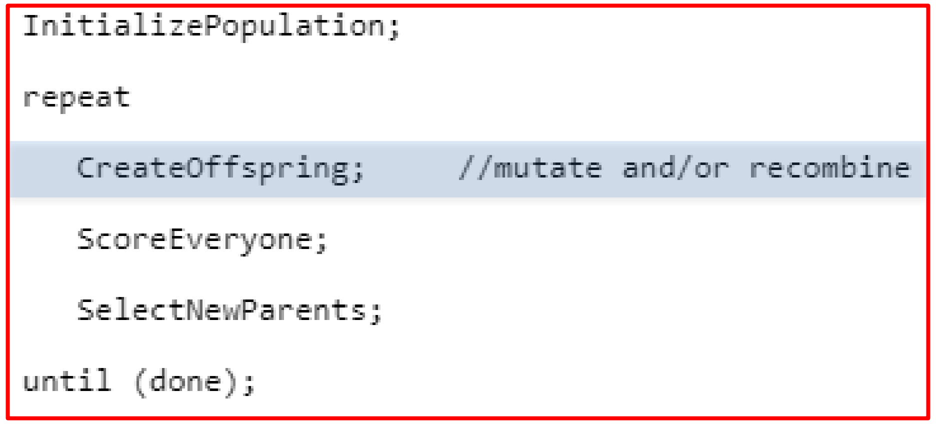
- Canonical Example in Two or More Dimensions (2)
  - Consider a population of μ candidate solutions, x_1, … x_μ parents
  - **Create the initial population**
    - In the case of a BBO generating the parents at random using an uniform distribution makes sense as you don´t have any hint about the function – **similar to one dimension problem.**



```
InitializePopulation;

repeat

   CreateOffspring;     //mutate and/or recombine

   ScoreEveryone;

   SelectNewParents;

until (done);
```

Global Numerical Optimization

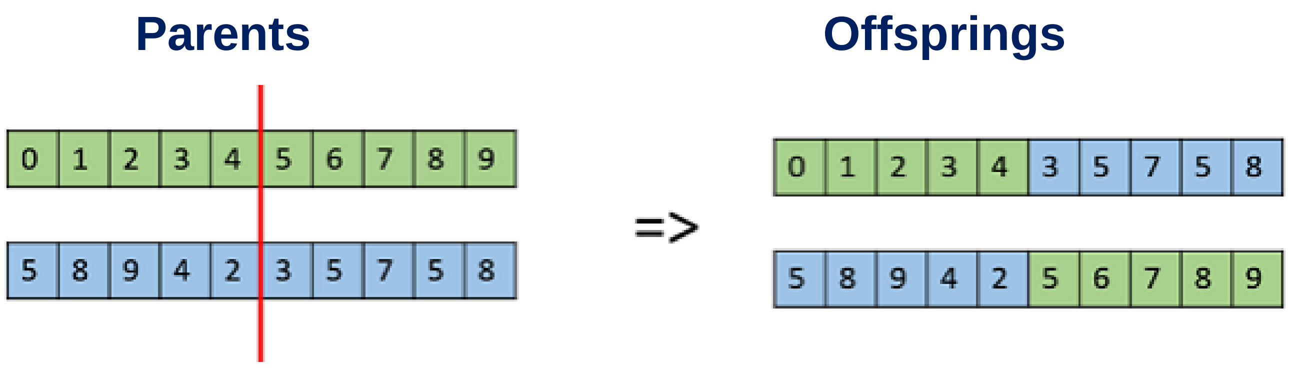
- Canonical Example in Two or More Dimensions (3)
  - Consider a population of μ candidate solutions, $x_1, \ldots x_\mu$ parents
  - **Create Offsprings**
    - Creating λ offsprings $x_{\mu+1}, \ldots x_{\mu+\lambda}$
    - methods that use a single parent to create a single offspring are described under the heading of mutation
    - methods that seek to combine multiple parents to create offspring are described with the term recombination, e.g., **crossover** and **blending**



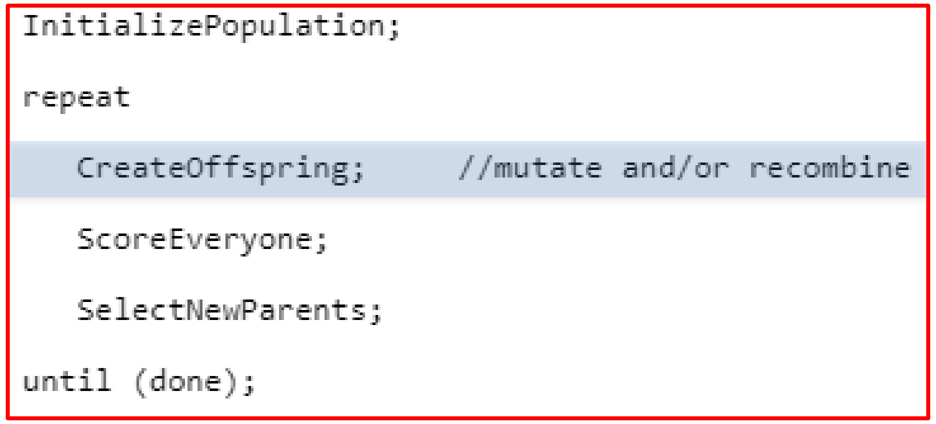
```
InitializePopulation;
repeat
    CreateOffspring;      //mutate and/or recombine
    ScoreEveryone;
    SelectNewParents;
until (done);
```

Global Numerical Optimization

- Canonical Example in Two or More Dimensions (4)
  - **Create Offsprings (Crossover)**
    - **One-point crossover**
    - Multipoint crossover
    - Uniform crossover



check: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm

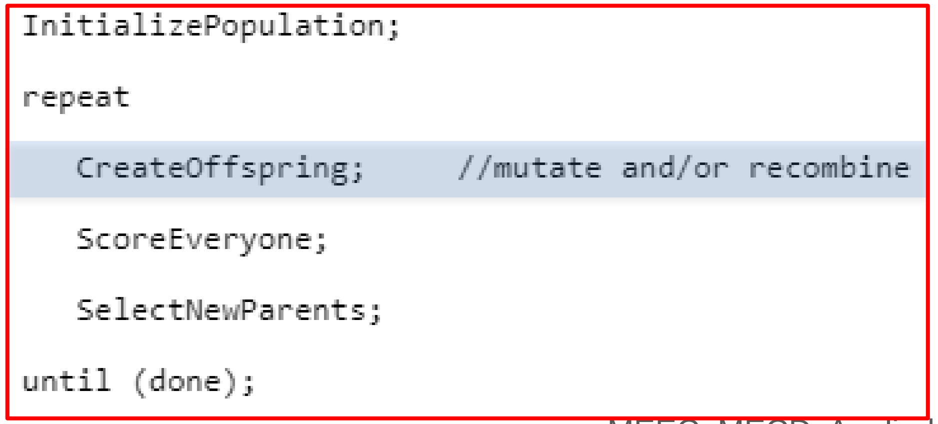


```
InitializePopulation;
repeat
    CreateOffspring;      //mutate and/or recombine
    ScoreEveryone;
    SelectNewParents;
until (done);
```

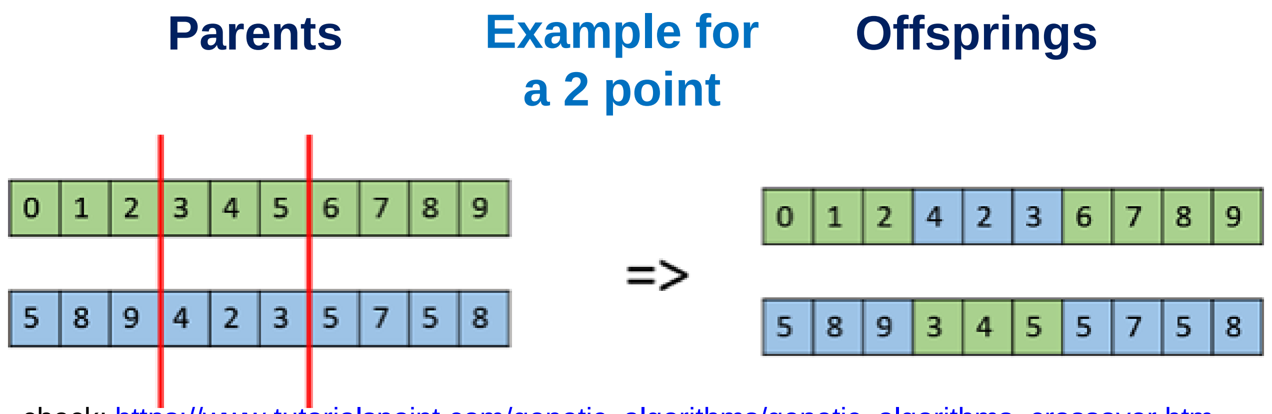
Global Numerical Optimization

- Canonical Example in Two or More Dimensions (5)

  - **Create Offsprings** (**Crossover**)
    - One-point crossover
    - **Multipoint crossover**: treats the solution vectors more like rings in which sections can be exchanged.
    - Uniform crossover: selects one component from either parent at random without regard to maintaining continuous segments and exchanges them.



```
InitializePopulation;
repeat
    CreateOffspring;     //mutate and/or recombine
    ScoreEveryone;
    SelectNewParents;
until (done);
```



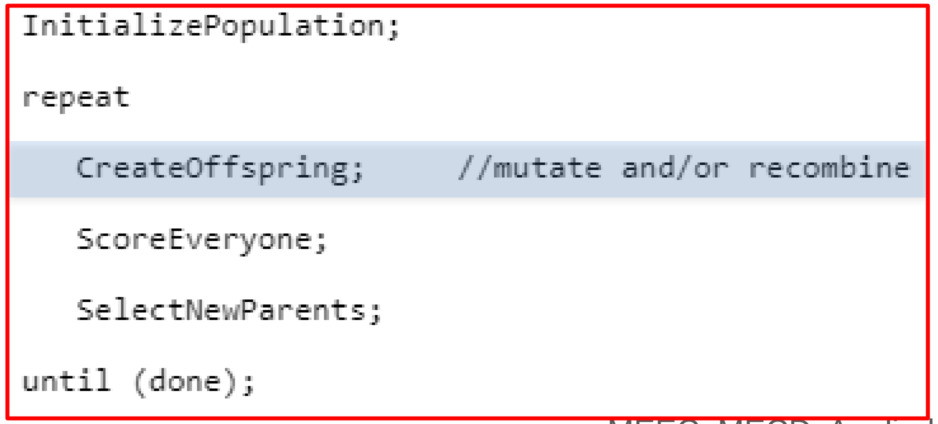
check: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm

DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

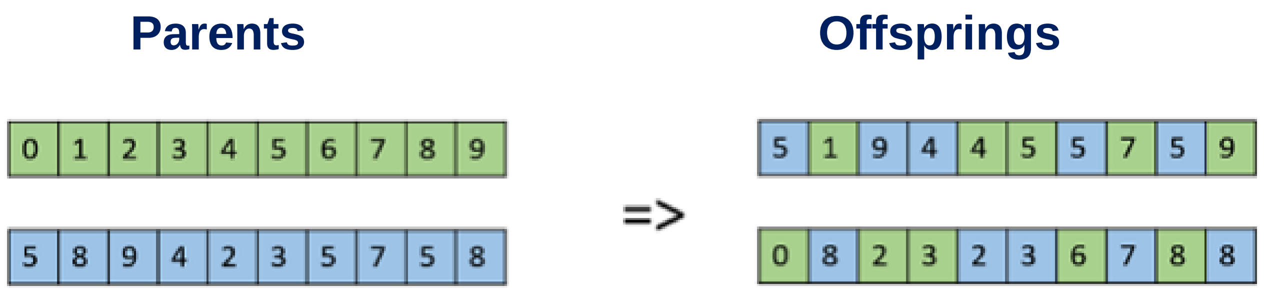
Global Numerical Optimization

- Canonical Example in Two or More Dimensions (5)

  – **Create Offsprings (Crossover)**
    - One-point crossover
    - Multipoint crossover: treats the solution vectors more like rings in which sections can be exchanged.
    - **Uniform crossover**: selects one component from either parent at random without regard to maintaining continuous segments and exchanges them.



```
InitializePopulation;
repeat
    CreateOffspring;     //mutate and/or recombine
    ScoreEveryone;
    SelectNewParents;
until (done);
```



check: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm

Global Numerical Optimization

- Canonical Example in Two or More Dimensions (6)
  - **Create Offsprings (Blending)**
    - **simple arithmetic mean;**
    - **weighted arithmetic mean;**
    - **geometric mean;**
    - **…**

Parents

$x_{11}, x_{12}, \ldots, x_{1n}$

$x_{21}, x_{22}, \ldots, x_{2n}$

Offspring

$(x_{11} + x_{21})/2, (x_{12} + x_{22})/2, \ldots, (x_{1n} + x_{2n})/2$

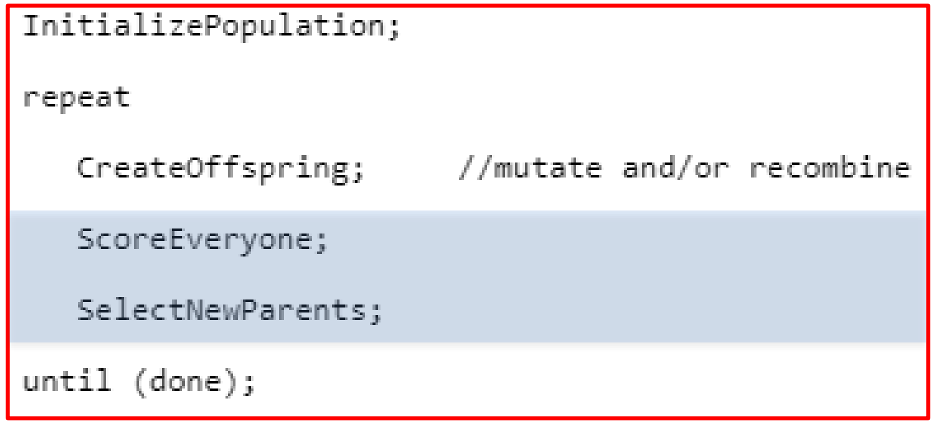
```
InitializePopulation;
repeat
    CreateOffspring;      //mutate and/or recombine
    ScoreEveryone;
    SelectNewParents;
until (done);
```

Global Numerical Optimization

- Canonical Example in Two or More Dimensions (6)

  – **Score and Selection**

    - **Similar to the Canonical Example in One Dimension**

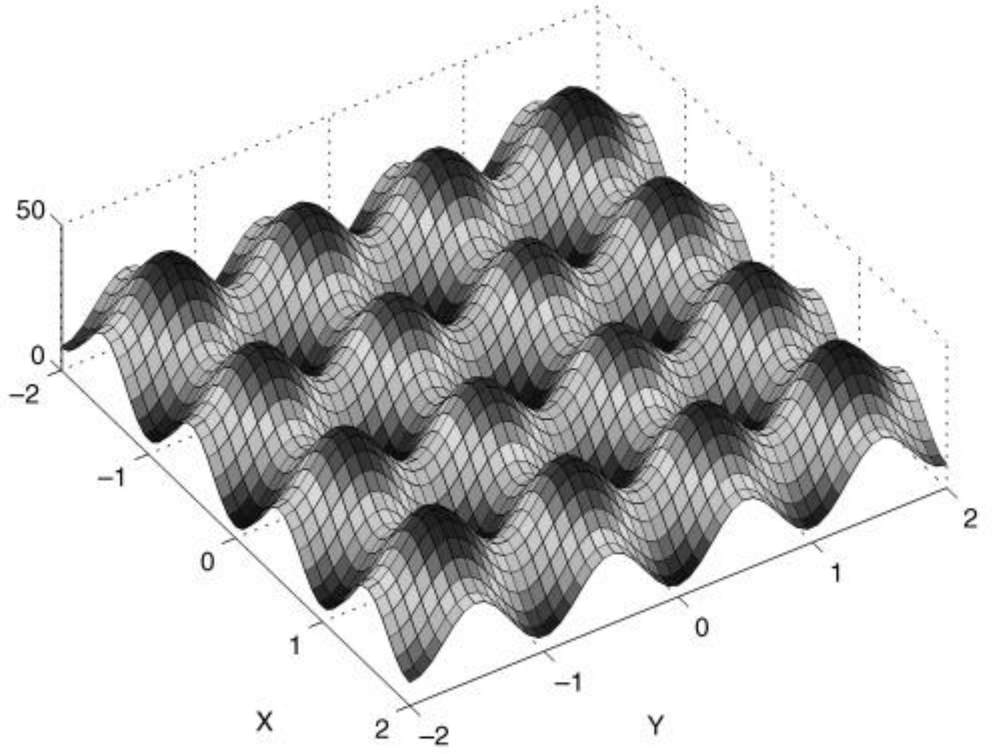


```
InitializePopulation;
repeat
    CreateOffspring;     //mutate and/or recombine
    ScoreEveryone;
    SelectNewParents;
until (done);
```

DEEC
DEPARTAMENTO DE ENGENHARIA
ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

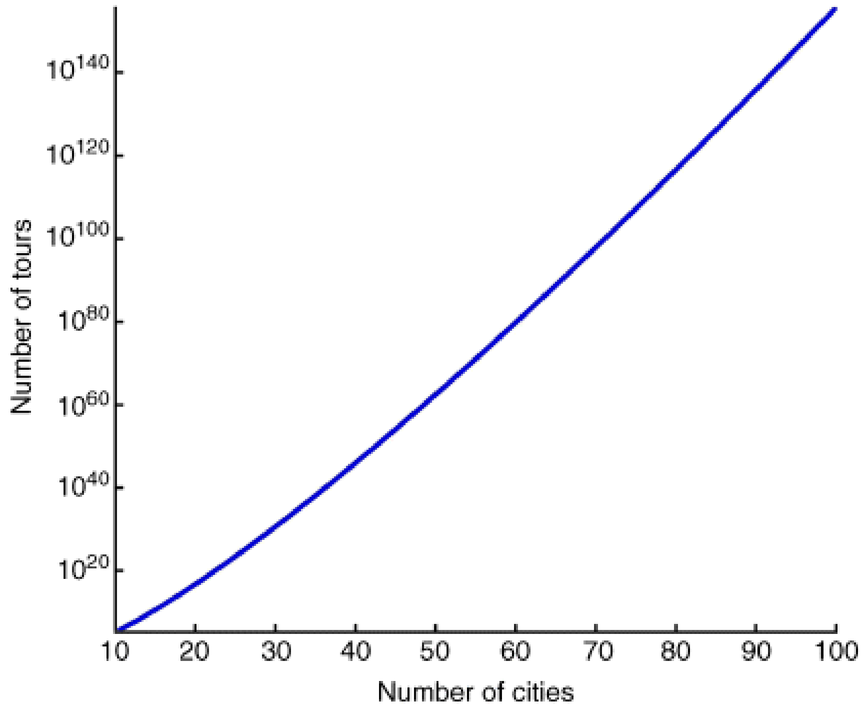
Global Numerical Optimization

- Evolution Algorithms vs Gradient Methods
  - **If the problem** at hand **presents a smooth, convex, continuous landscape** then **gradient or related methods of optimization will be faster** in locating the single optimum point.
  - **if the problem presents** a landscape with **multiple local optima** (see Figure), then **gradient methods will likely fail to find the global optimu** … but **EA apply**.
  - **If the landscape is discontinuous** and/or not smooth, then **gradient-based approaches may be inapplicable** … but **EA apply**!



Combinatorial Optimization

- Canonical Case of Addressing Traveling Salesman Problem (**TSP**) (1)
  - Consider **n cities (Domain)**. The salesman starts at one of these cities and must visit each other city once and only once and then return home. **The salesman wants to do this in the shortest distance** (**Objective**).
  - The problem then is to determine the best ordering of the cities to visit.
  - This is a difficult problem **because the total number of possible solutions increases as a factorial func.** of the number of cities. More precisely, for n cities, the total number of possible solutions is **(n-1)!/2**.
  - The **traveling salesman** problem is **NP-hard**, i.e., there are no known methods for generating solutions in a time that scales as a polynomial function of the number of cities, n.

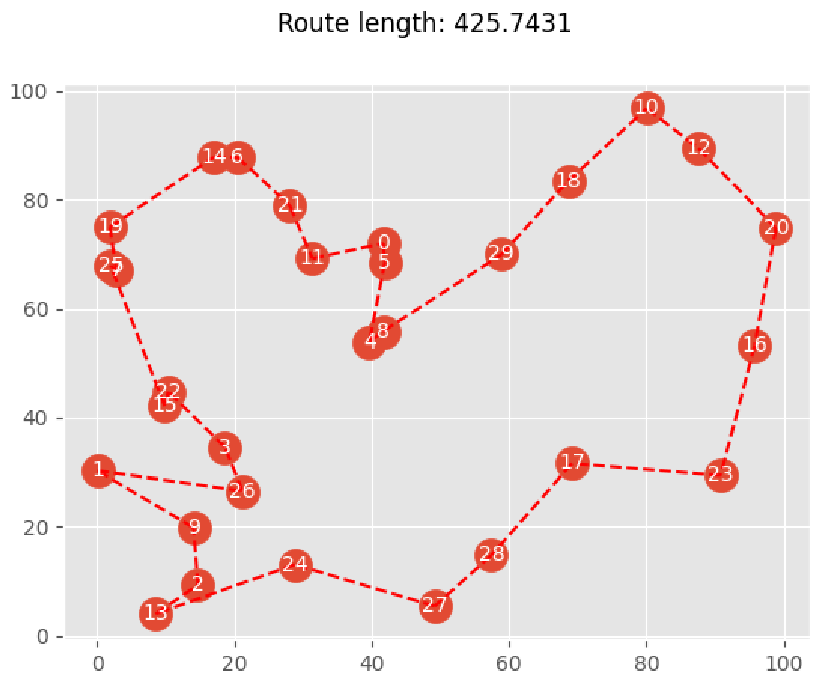


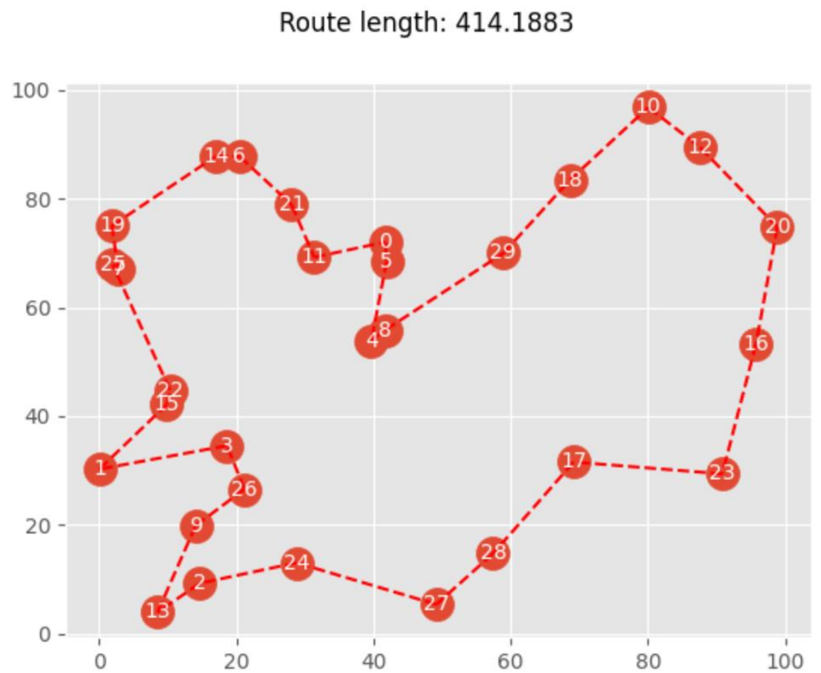
Combinatorial Optimization

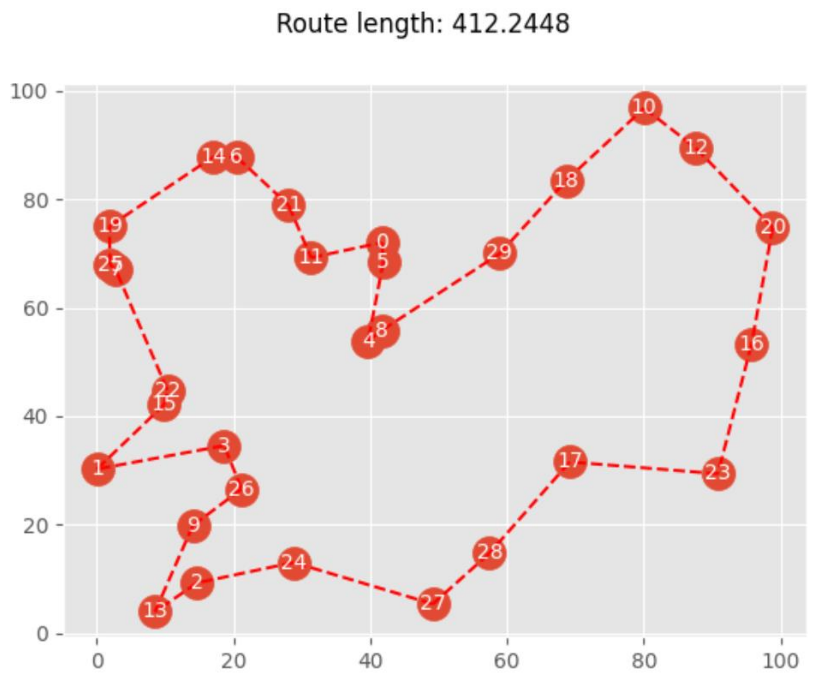
- Traveling Salesman Problem (**Example**)
  - **pymoo – Multi-Objective Optimization in Python**
    - **pymoo** is one python library, as well as **DEAP** and others, which implement several evolutionary algorithms from single-objective to multi-objective.
    - Libraries are extremely useful to start using different algorithms without having to implement from scratch, but we must understand well the use of the library functions.
  - **Suggestion**: check the code for an example of using a Genetic Algorithm to solve the TSP problem in **https://pymoo.org/customization/permutation.html**
    - **Try problems with different complexity, e.g., 30 cities in a square 100x100 (30, 100), (10, 100), (100,100)**
    - **Try different population size (20, 40, 60, 80, 100)**
    - **About the available operators for permutations check https://pymoo.org/operators/index.html**

Combinatorial Optimization

- Traveling Salesman Problem (**Example**)
  - **#cities = 30, #area = 100x100**
  - **#population = 20, 40, 100 Stop Criteria 200 generations without improvement**
  - **Sampling = perm_random (search for Random Permutation)**
  - **Crossover = perm_erx (search for Edge Recombination Operator)**
  - **Mutations = perm_inv (search for Inverse Mutation)**



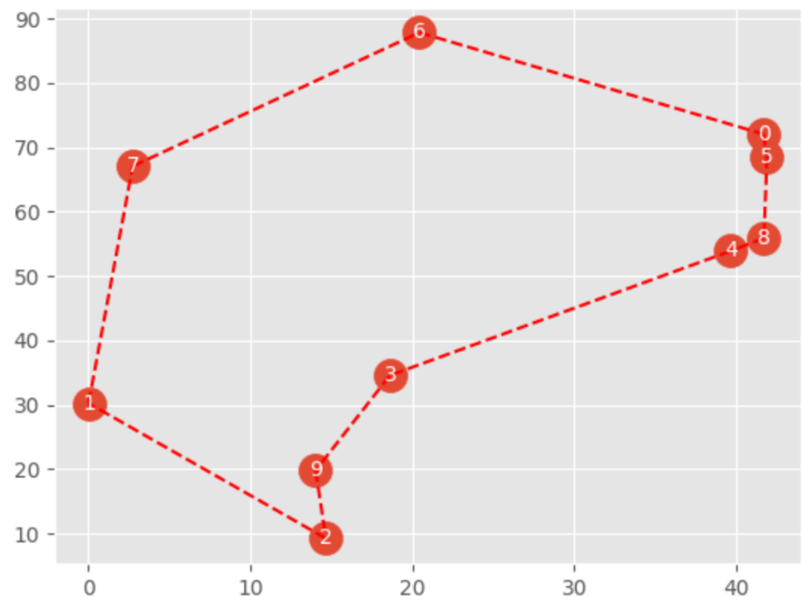




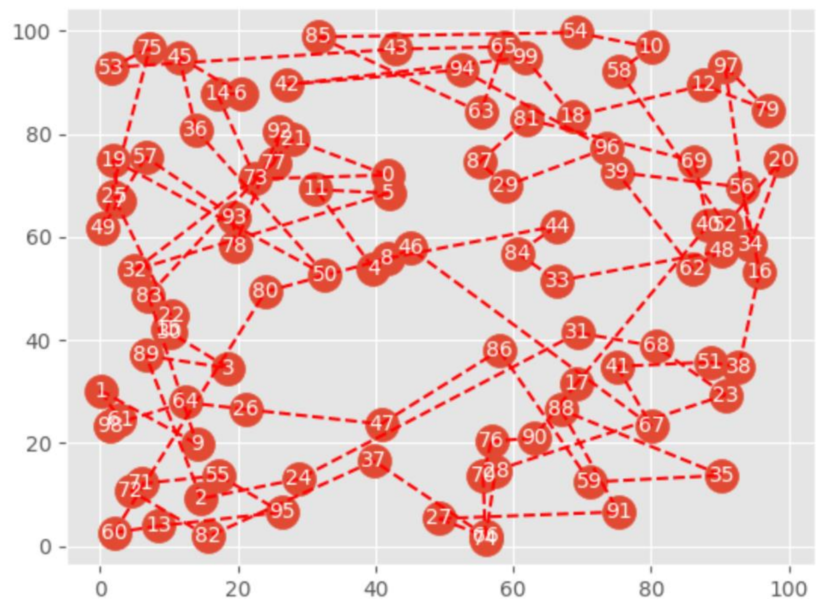
Combinatorial Optimization

- Traveling Salesman Problem (**Example**)
  - **#cities = 10, 100, #area = 100x100**
  - **#population = 40, 20, 40, Stop Criteria 200 generations without improvement**
  - **Sampling = perm_random (search for Random Permutation)**
  - **Crossover = perm_erx (search for Edge Recombination Operator)**
  - **Mutations = perm_inv (search for Inverse Mutation)**

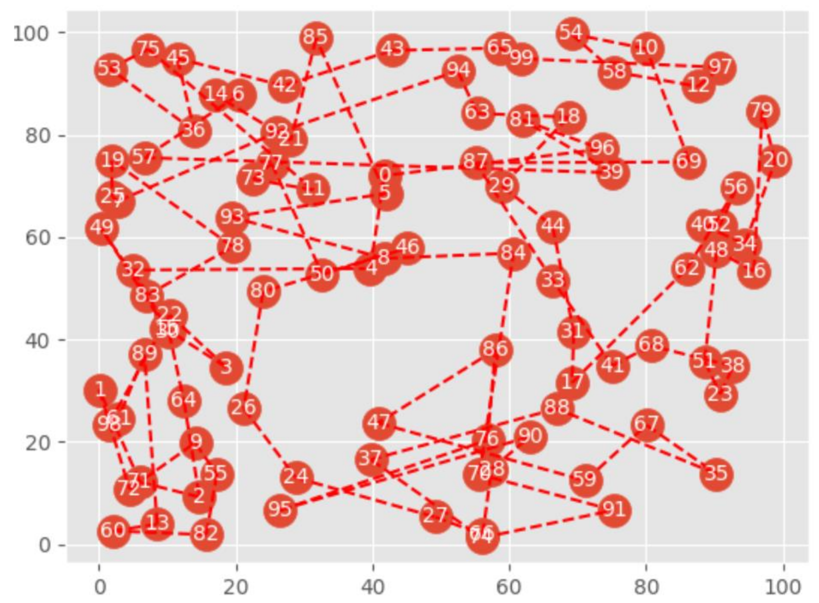
Route length: 189.9341



Route length: 1831.4889



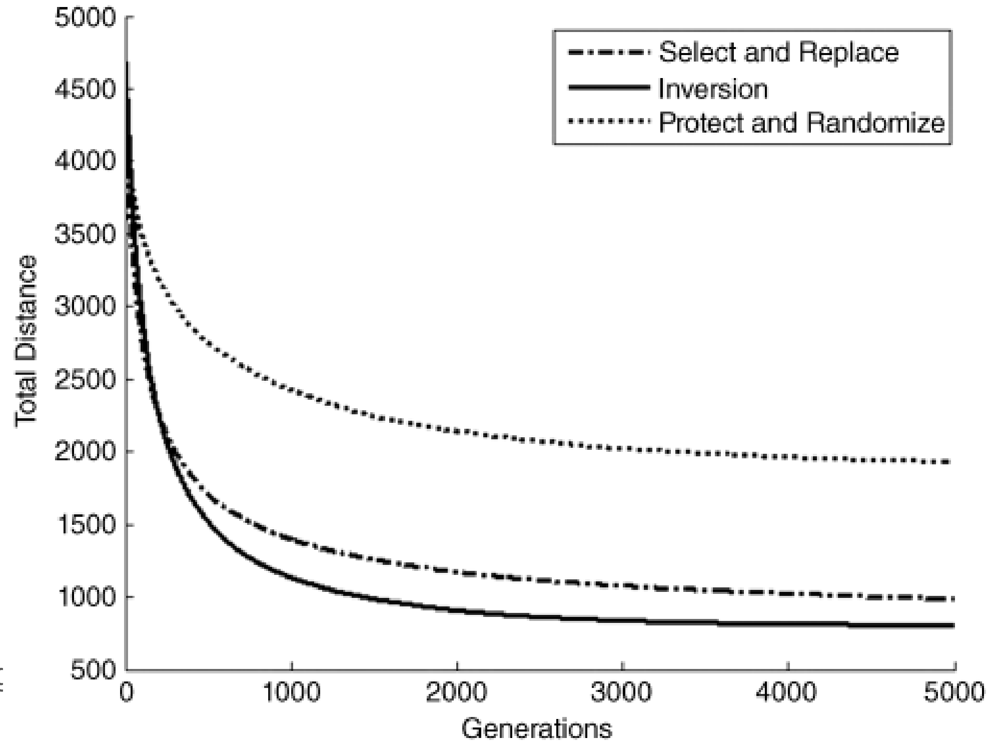
Route length: 1774.0619



DEEC
DEPARTAMENTO DE ENGENHARIA
ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

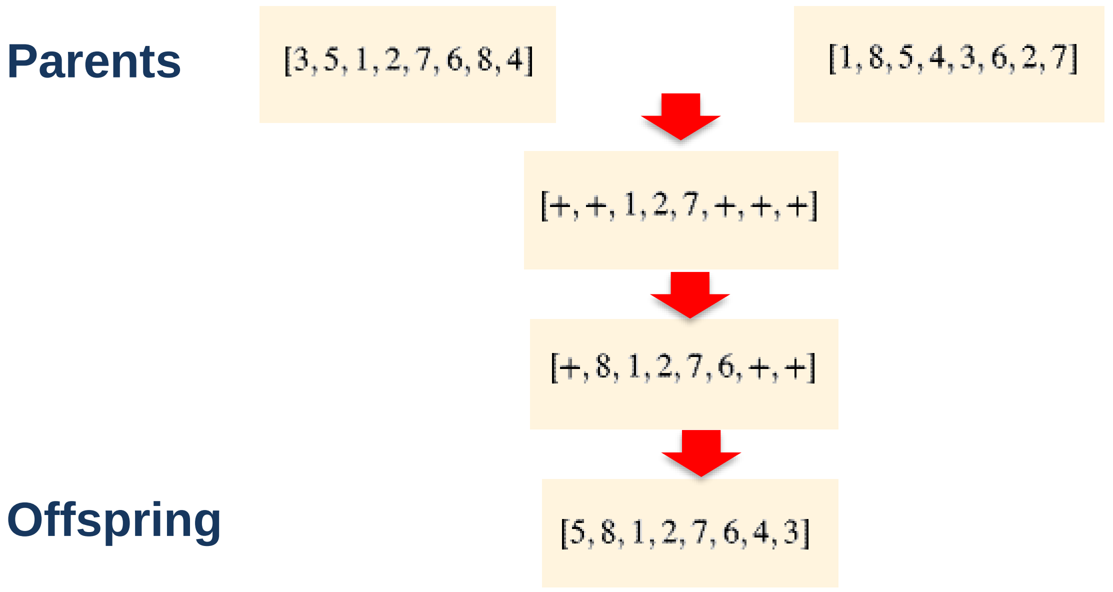
Combinatorial Optimization

- Canonical case of Addressing Traveling Salesman Problem (2)
  - **Creating offsprings** (mutation alternatives):
    - **Select and replace**: Choose a city at random along the list and replace it at a random place along the list.
    - **Invert**: Choose two cities along the list at random and invert the segment between those cities.
    - **Protect and randomize**: Choose a segment of the list to be passed from the parent to the offspring intact, and then randomize the remaining cities in the list.



Combinatorial Optimization

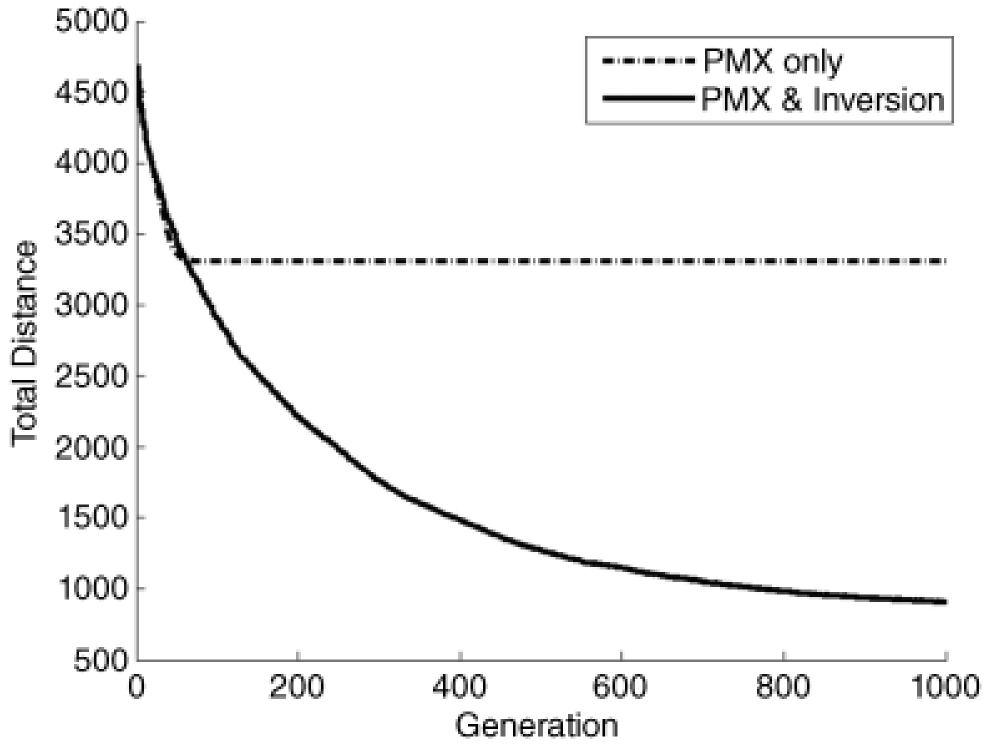
- Canonical case of Addressing Traveling Salesman Problem (3)
  - **Creating offsprings** (alternatives):
    - **PMX**: partially mapped crossover, the operator works on two parents by **choosing a segment of the first parent** to move directly to the offspring. It then **moves the feasible parts from the second parent** to the offspring. Finally, it assigns the **remaining values to the offspring based on the indexing in the first parent**.

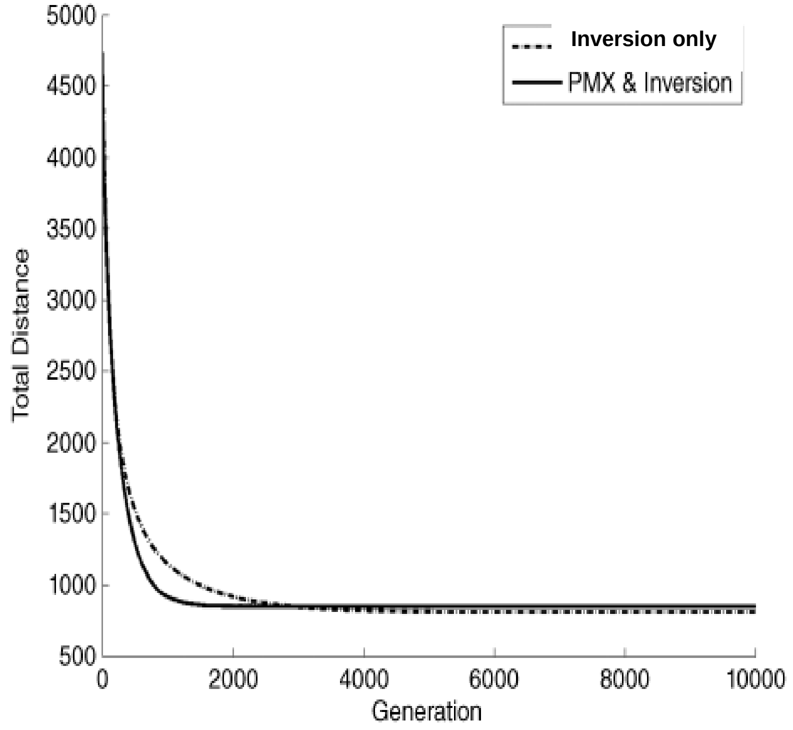


DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

Combinatorial Optimization

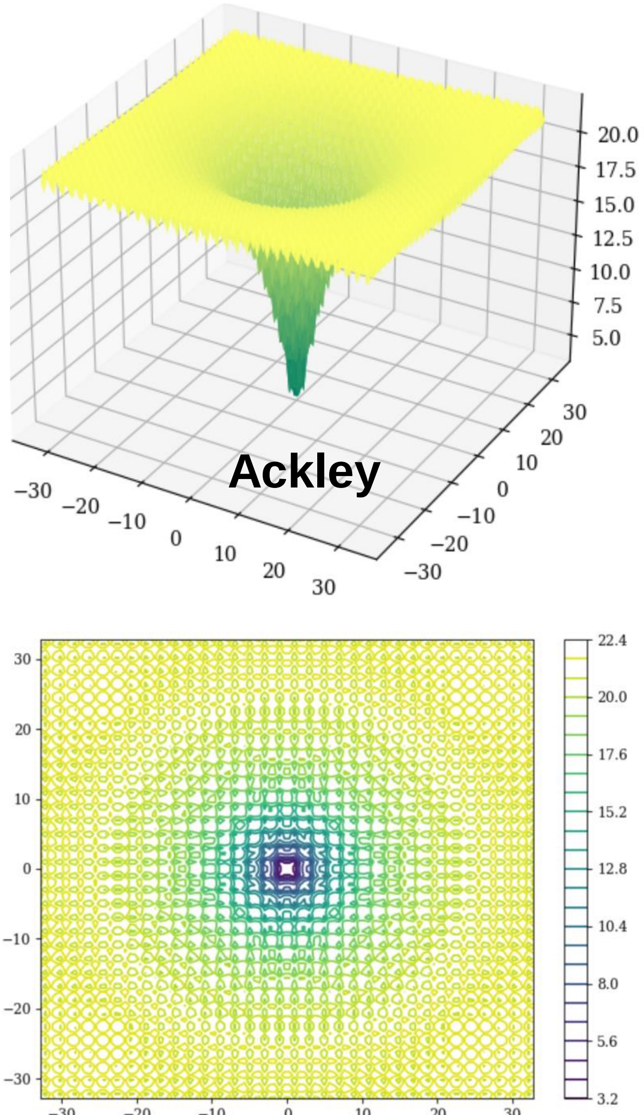
- Canonical case of Addressing Traveling Salesman Problem (4)
  - **Creating offsprings** (alternatives):

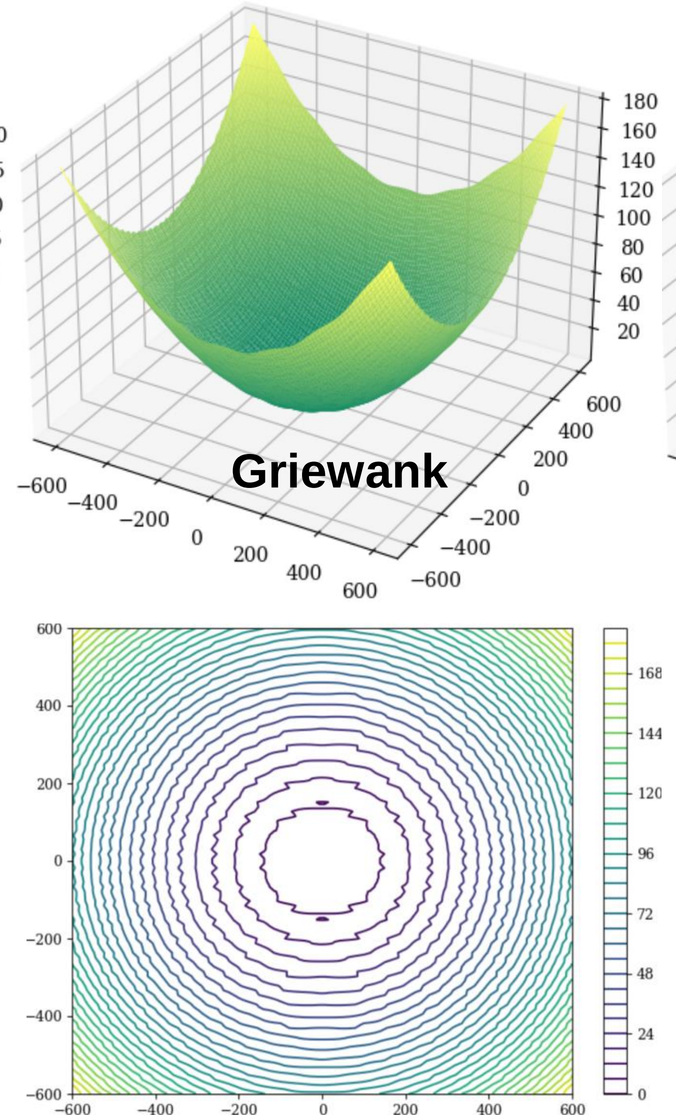


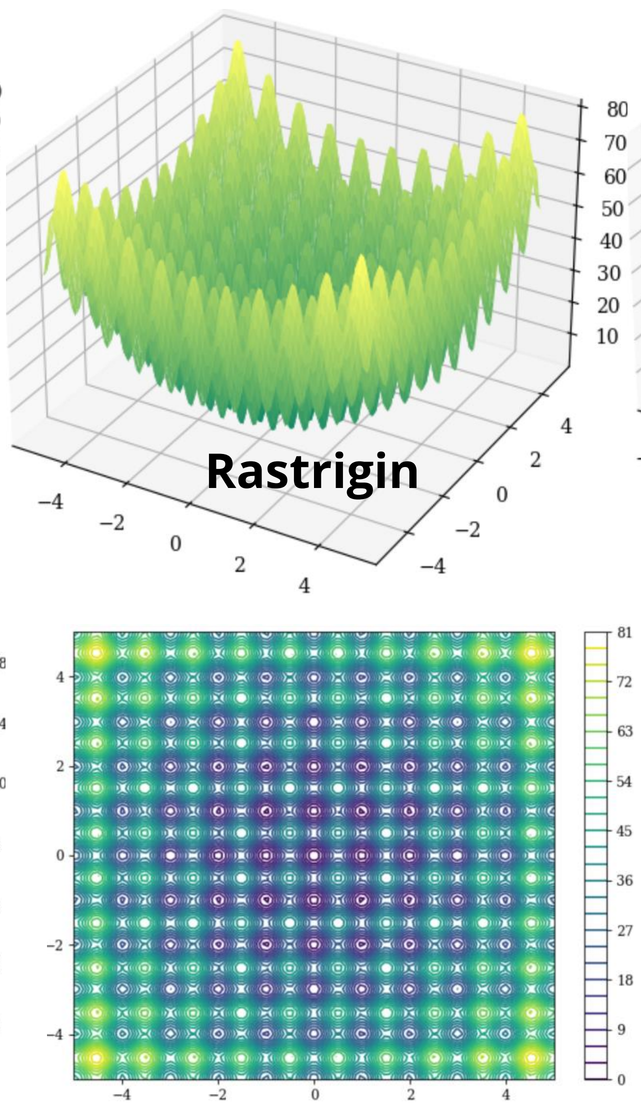


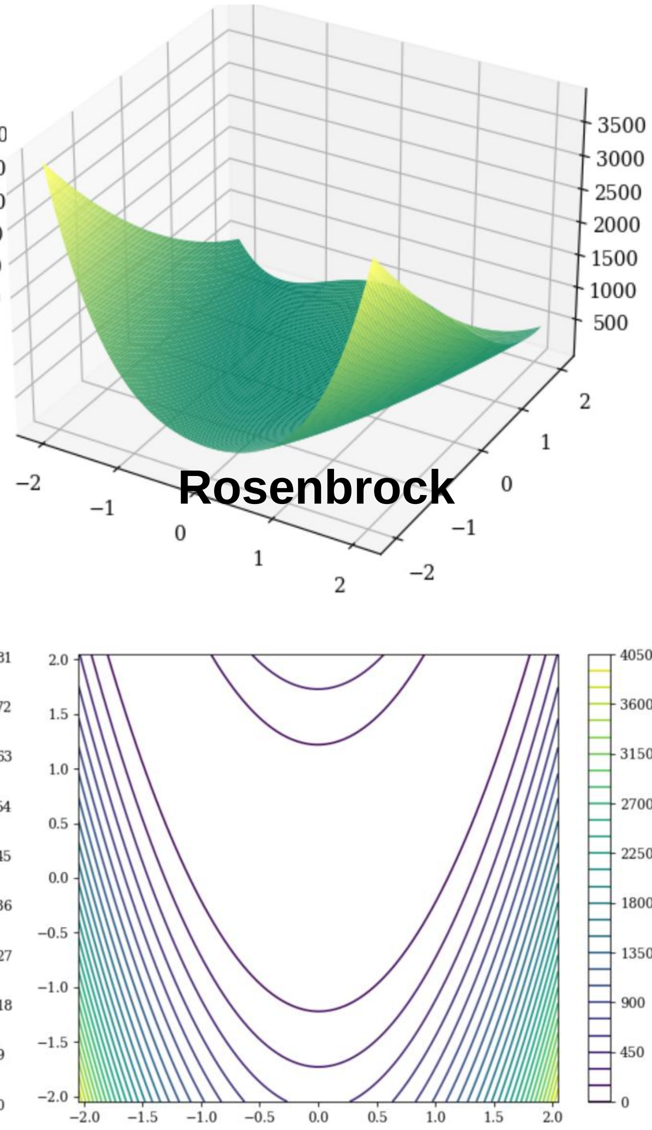
DEEC
DEPARTAMENTO DE ENGENHARIA
ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

Test Functions Examples (1)



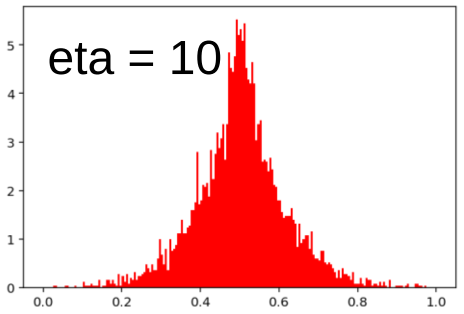


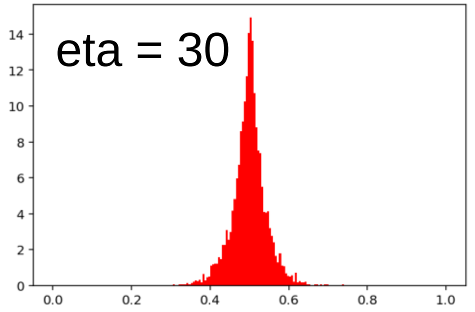


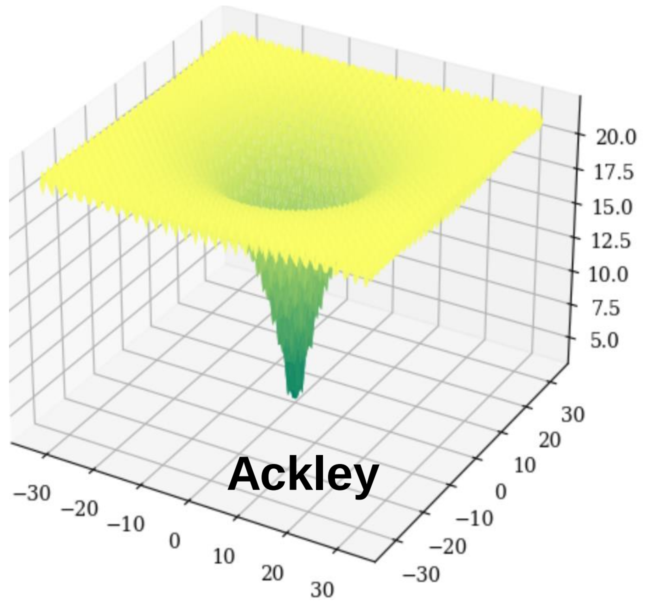


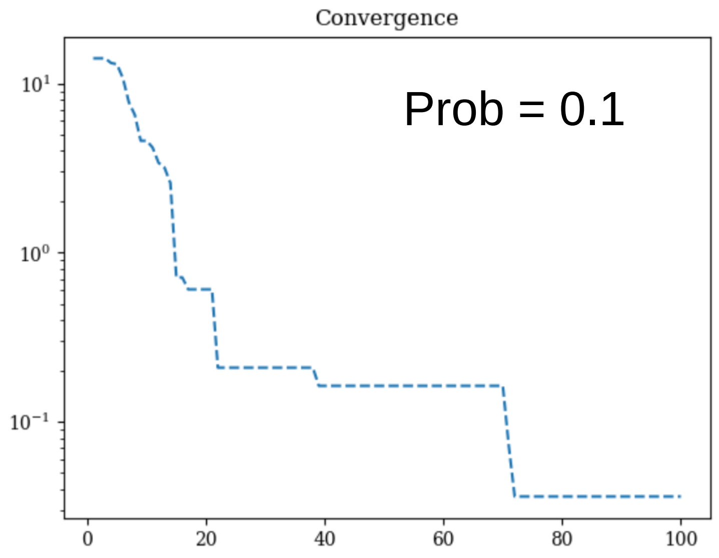
Test Functions Examples (2)

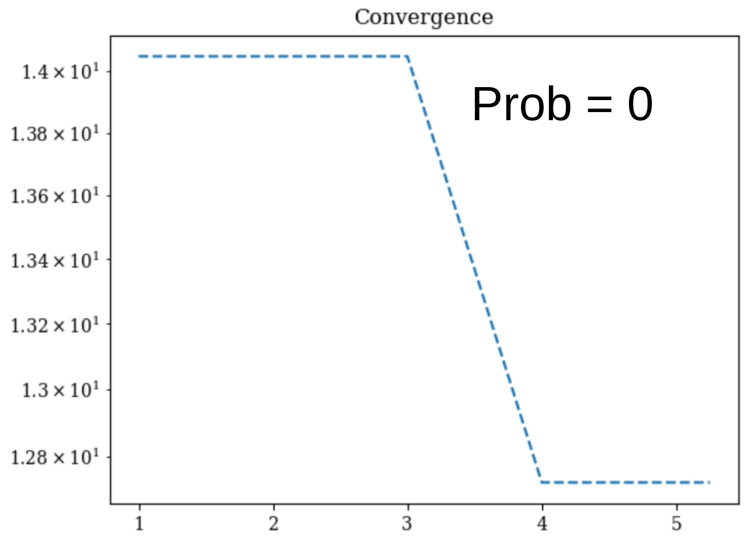
- Pop: 20
- Sampling: Random
- Selection: Random
- Crossover: One Point
- Mutation: Polinomial Mutation (eta=10, prob=0.1)
- Gen: 100

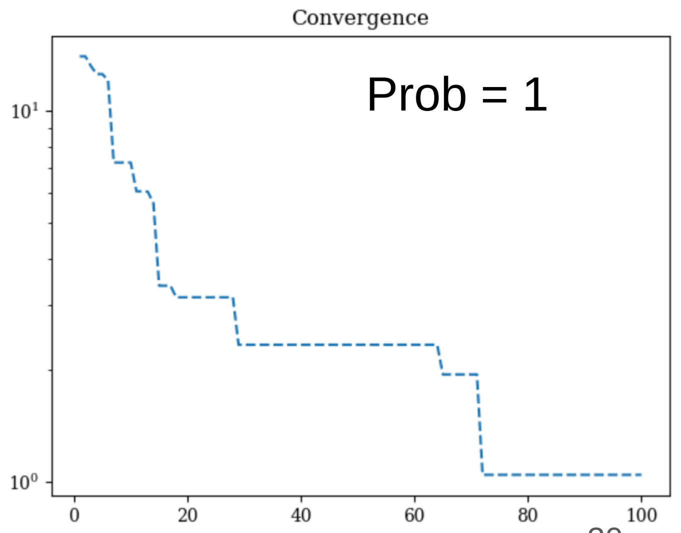








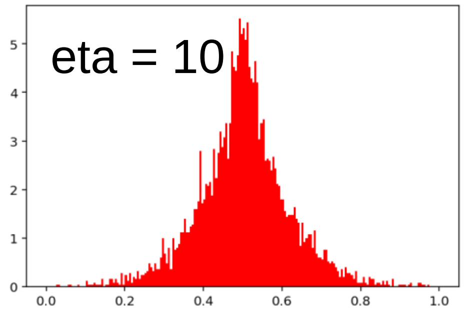


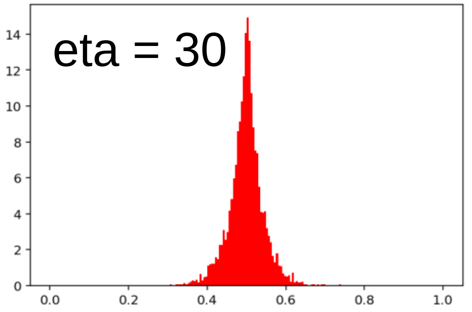


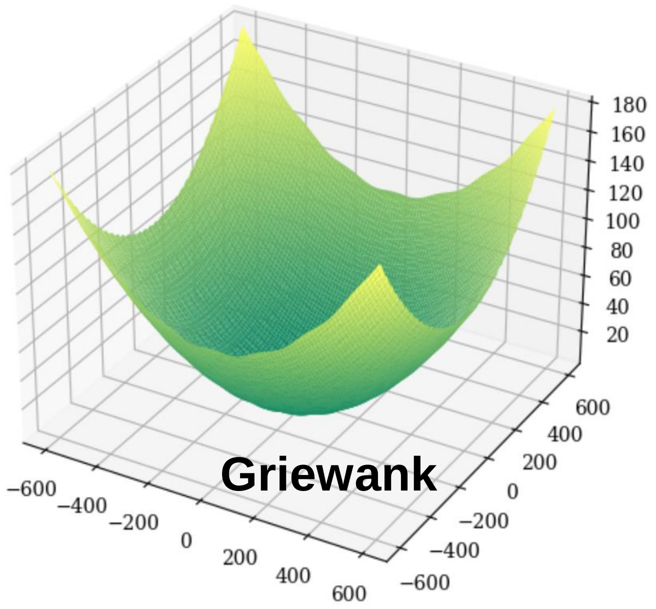
DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

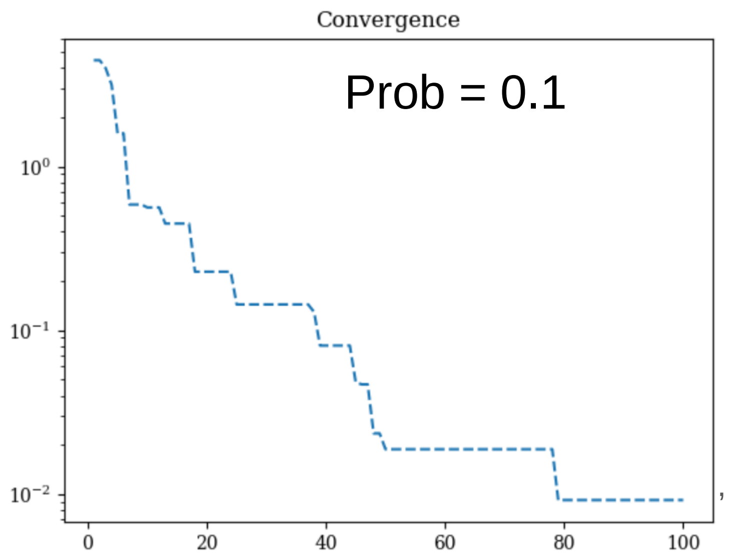
Test Functions Examples (2)

- Pop: 20
- Sampling: Random
- Selection: Random
- Crossover: One Point
- Mutation: Polinomial Mutation (eta=10, prob=0.1)
- Gen: 100



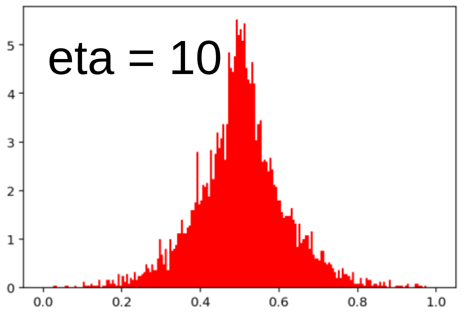


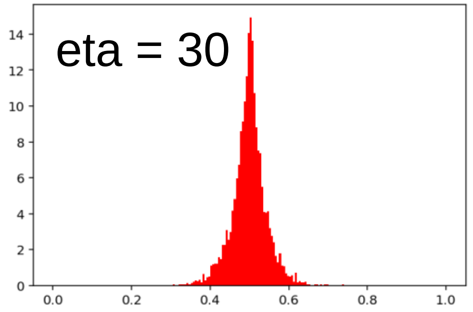


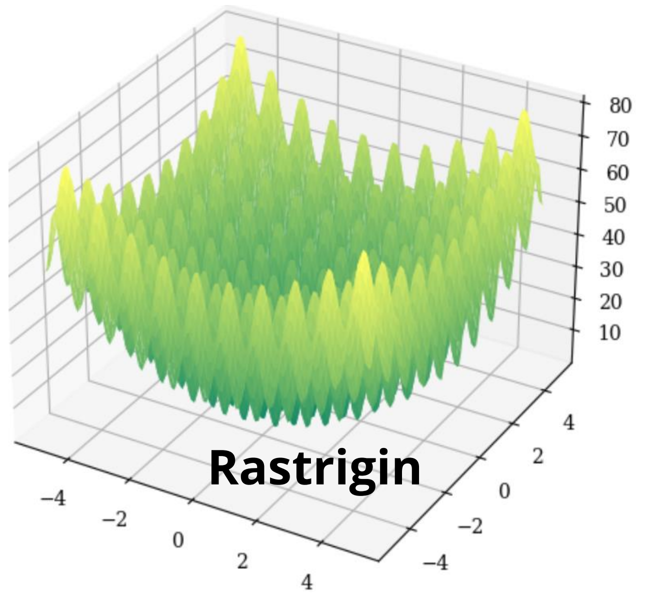


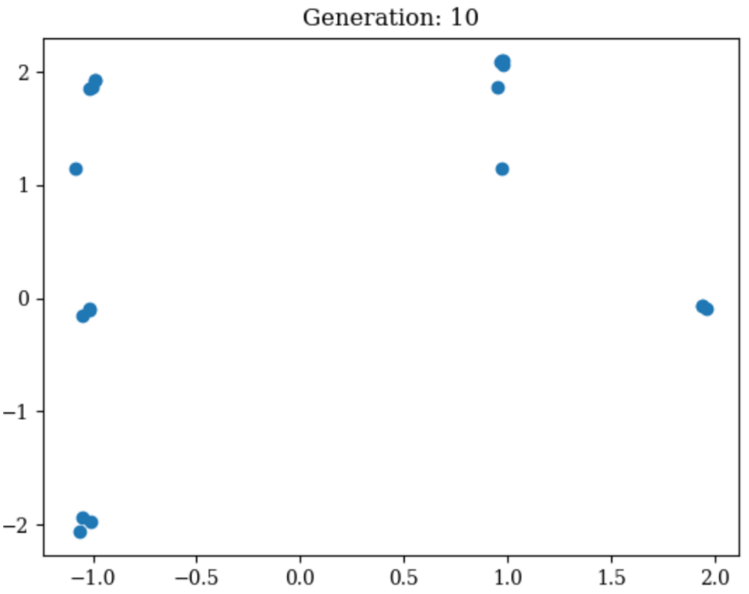
Test Functions Examples (2)

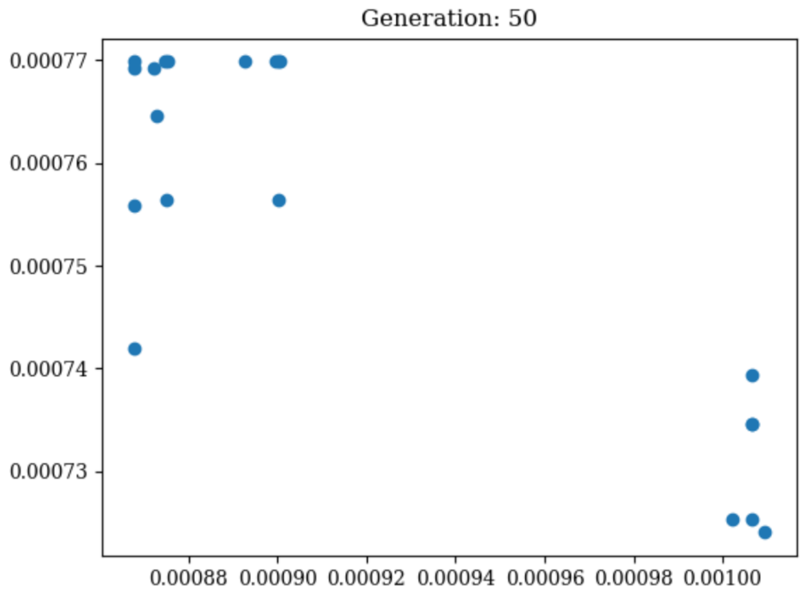
- Pop: 20
- Sampling: Random
- Selection: Random
- Crossover: One Point
- Mutation: Polinomial Mutation (eta=10, prob=0.1)
- Gen: 100

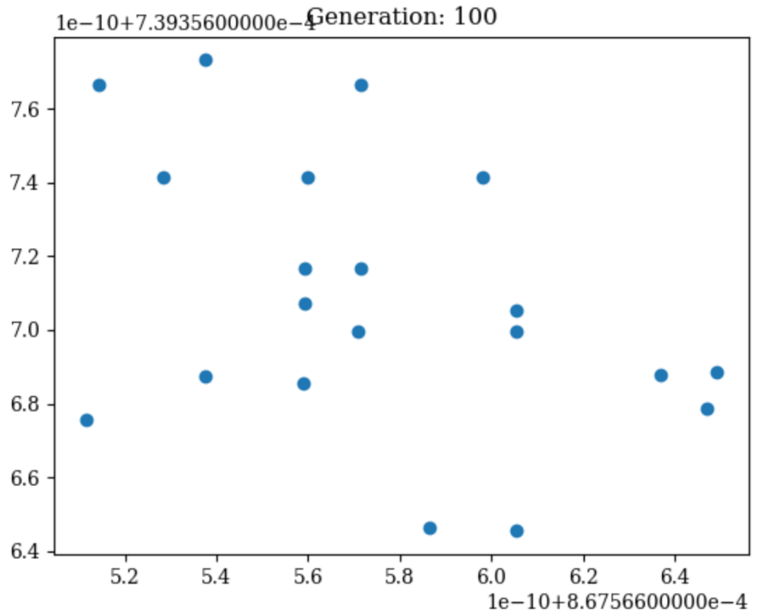






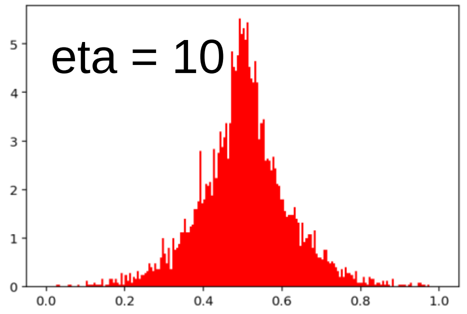


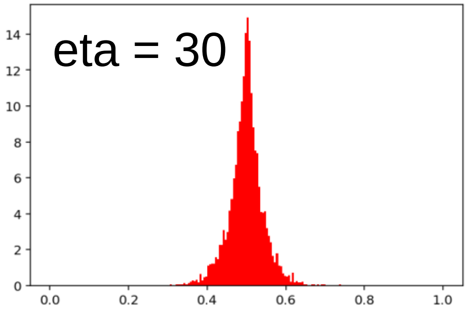


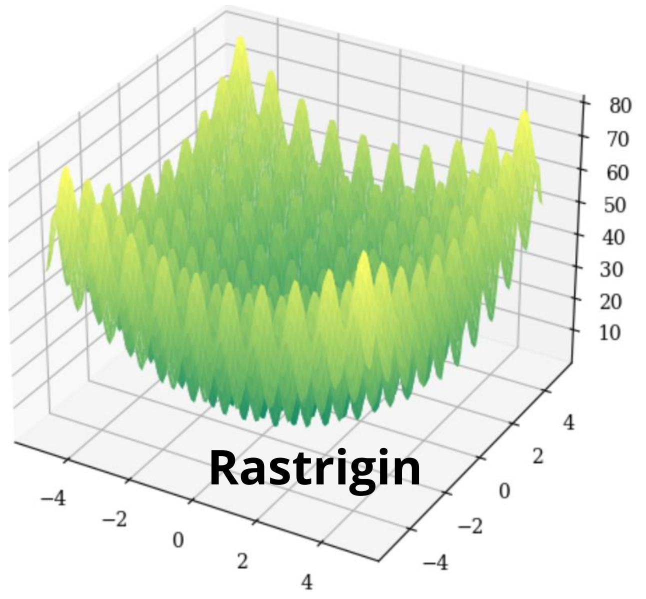


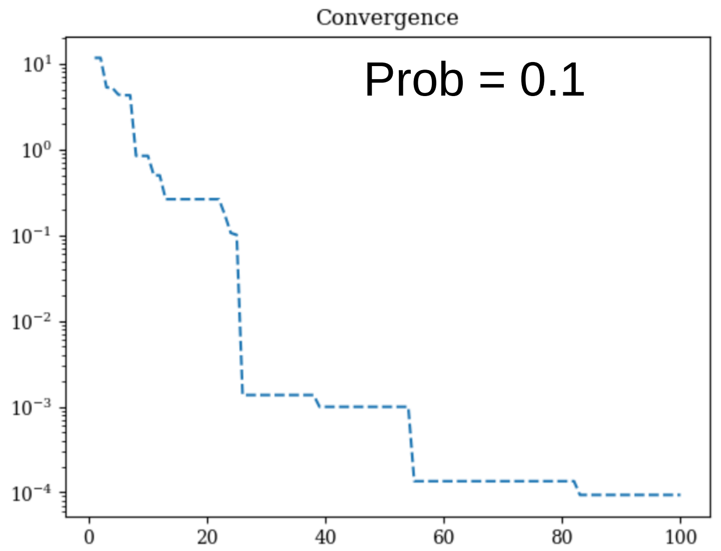
Test Functions Examples (2)

- Pop: 20
- Sampling: Random
- Selection: Random
- Crossover: One Point
- Mutation: Polinomial Mutation (eta=10, prob=0.1)
- Gen: 100





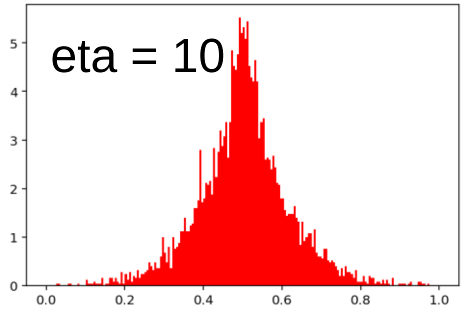


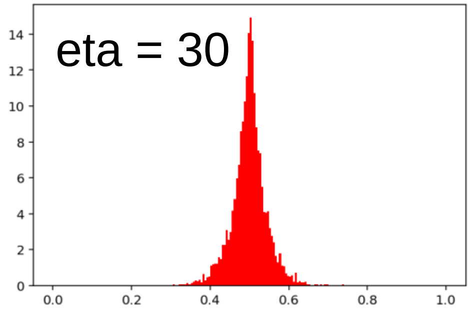


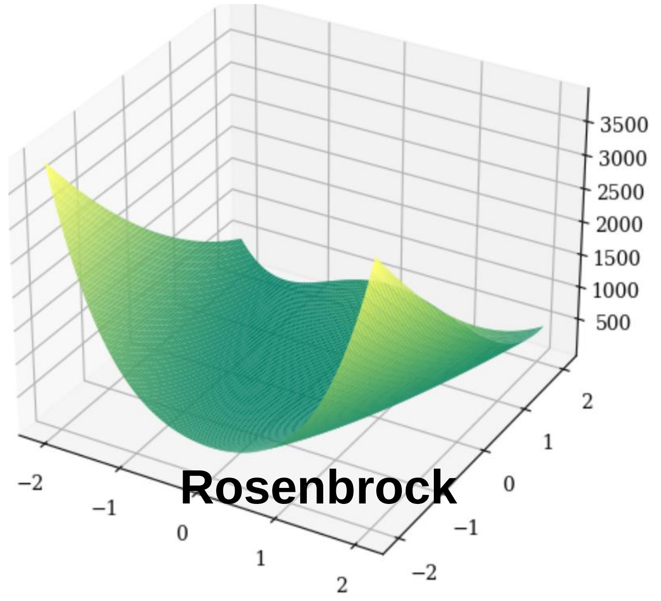
DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

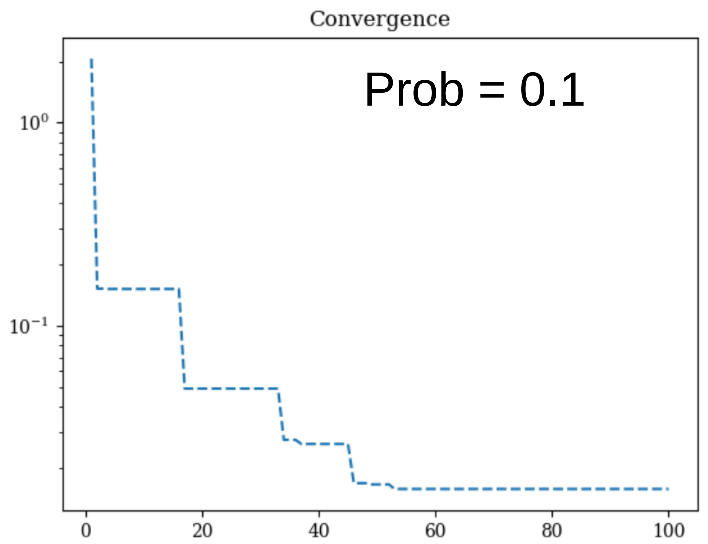
Test Functions Examples (2)

- Pop: 20
- Sampling: Random
- Selection: Random
- Crossover: One Point
- Mutation: Polinomial Mutation (eta=10, prob=0.1)
- Gen: 100









Matematical Considerations

- **Convergence**
  - **Convergence with Probability 1**
    - with **Elitist selection and mutation operator that can reach all possible states**
  - **Premature Convergence**
    - Evolutionary algorithms that rely predominantly on crossover or other recombination methods to generate offspring can sometimes suffer from what is termed **premature convergence**, which **occurs when a population becomes homogeneous at a solution that is not the global optimum**.
- **Representation**
  - **Binary Representation**
    - the notion that binary representations are universally better than other representations is false.
  - **Aspects to consider when Selecting a Representation**
    - **The representation should optimally provide immediate information about the solution** itself. For example, in the traveling salesman problem, the list of cities is suggestive of the solution.
    - **The representation should be amenable to variation operators** that are well understood for their mathematical properties and can exhibit a gradation of change. For example, **when searching for x such that f(x) is minimized, using a Gaussian variation operator on x allows generating offspring that are close to or far from x**, and this can be controlled by changing the standard deviation in each dimension
    - Unless the objective is to explore the utility of a novel representation, **utilizing representations that have been studied and for which results have been published may allow more systematic and meaningful comparisons**.

Matematical Considerations

- **Selection**

  Selection describes either **the process of eliminating solutions from an existing population or making proportionally more offspring from certain parents**. Some common forms of selection include **plus/comma**, **proportional**, **tournament**, and **linear ranking**.

  - **Plus/Comma Selection**
    - notation $(\mu+\lambda)$ and (μ,λ) refer to Plus and Comma selection respectively. In the first, both parents and offsprings to be parents of the next generation. In the former, the parents die and the offsprings are considered for parents in the next generation. **Variants include allowing parents to survive to a maximum of n generations.**
  - **Proportional Selection or Roulette Wheel Selection**
    - proportional selection picks parents for reproduction in proportion to their relative fitness. The probability of selecting an individual in the population for reproduction is determined as: where p_i is the probability of selecting the ith individual, there are μ existing individuals, and f(i) is the fitness of the ith individual

$$p_i = f(i) / \sum_{j=1}^{\mu} f(j)$$

Matematical Considerations

- **Selection**
  - **Tournament Selection**
    - There have been different forms of tournament selection in the history of evolutionary algorithms, but the more common one selects a subset of size q (q = 2) from the existing population and **selects the best of those q individuals to be a member of the next generation**.
    - As with proportional selection, tournament selection allows the possibility that solutions that are less than best can propagate into a future generation.
  - **Linear Ranking Selection**
    - Linear ranking selection **maps individuals to selection probabilities according to a prescribed formula** based on the rank of the solution:

$$p_i = (\eta^+ - (\eta^+ - \eta^-)[i-1]/[\mu - 1])/\mu$$

    - where p_i is the probability of selecting the ith individual,

μ is the number of individuals in the population and η^+ and η^- are user-controlled constants constrained by $1 \leq \eta^+ \leq 2$ and $\eta^- = 2 - \eta^+$. For example, if $\mu = 100$ and $\eta^+ = 1.5$, then $\eta^- = 0.5$, and the probability of selection of the ith-ranked solution in the population is:

$$p_1 = (1.5 - (1)(0/99))/100 = 0.0015$$

$$p_2 = (1.5 - (1)(1/99))/100 = 0.00148989\ldots$$

$$p_{100} = (1.5 - (1)(100/99))/100 = 0.0048989\ldots$$

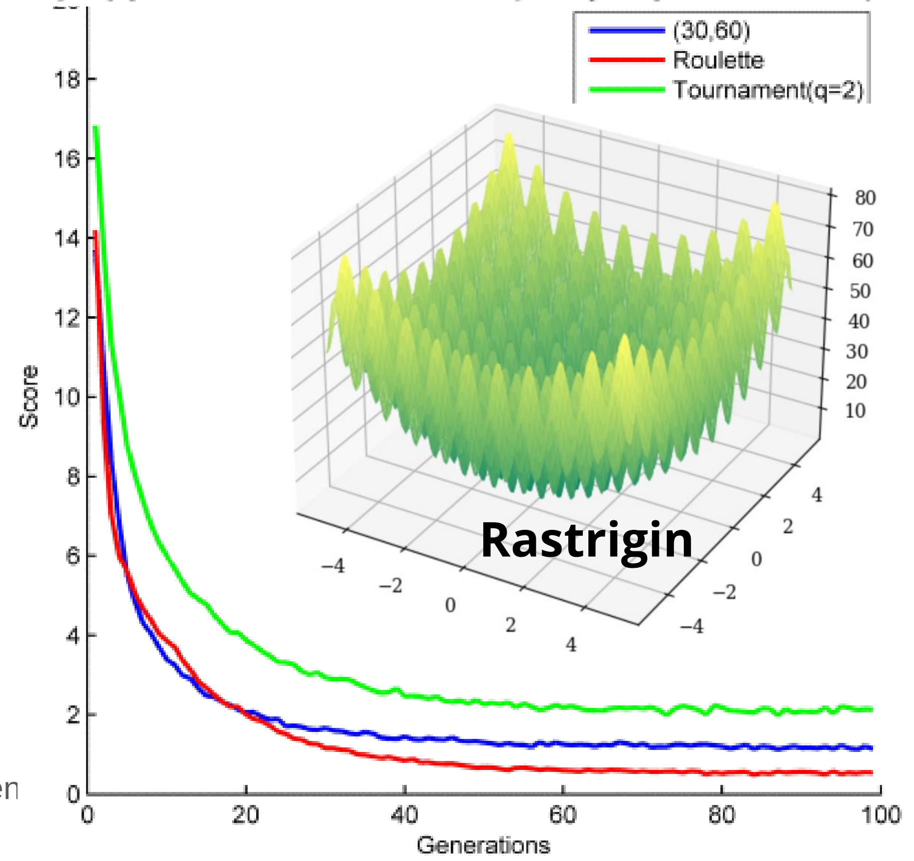
DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

Matematical Considerations

- **Selection** (Example)
  - Let's examine the effects of different selection operators when combined with a simple evolutionary algorithm to search for the minimum of Rastrigin's function in two dimensions.

$$f(x,y) = 20 + x^2 - 10\cos(2\pi x) + y^2 - 10\cos(2\pi y)$$

  - We'll initialize a **population of 30 candidate solutions uniformly at random from [−10, 10]** in each dimension. **Variation** will be accomplished **by a Gaussian mutation**, zero mean, with standard deviation equal to 0.25 applied to each dimension. We'll create **60 offspring from 30 parents** and compare the results of a **(30, 60) selection with those from roulette wheel selection and tournament selection** with q = 2. Figure 11.6 shows the average results of 500 trials with each method through the first 100 generations. Tournament selection with q = 2 evidenced the slowest rate of improvement, while roulette wheel selection found better solutions on average than did the (30, 60) selection method.



Matematical Considerations

- **Selection** (Comments)
  - There are **various extensions** to these basic forms of selection.
    - **Elitist selection** can be applied, which automatically ensures that the best solution in a population is retained in the next generation
    - **Nonlinear ranking methods** can be used
    - There is a continuum of selection methods from those that are **"soft"** to those that are **"hard." The harder the selection, the faster the better solutions can overtake the population**.
      - This is often described in terms of takeover time, which is the expected number of generations required to fill a population with copies of what is initially a single best individual when applying only selection (no variation).
    - As shown by Bäck [1994], for typical settings of the parameters in the procedures above and on given test cases, **the order of selection strength from weakest to strongest** is **proportional selection, linear ranking selection, tournament selection, and then plus/comma selection**.
    - If you are using an evolutionary algorithm that relies heavily on recombination, then diversification is required for the population to search for new solutions

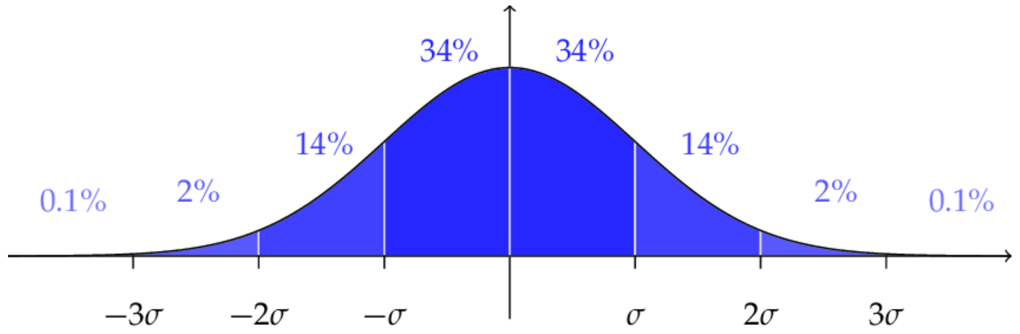
Variation

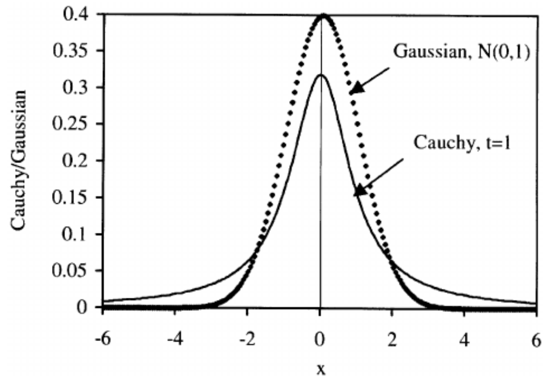
Variation operators provide the **means for searching the solution space for improved solutions**, or potentially for weaker solutions that could lead to improved solutions. There are many traditional variation operators, some of which have already been discussed, such as binary mutation, Gaussian mutation, or one-point, n-point, or uniform crossovers. **The choice of variation operators goes "hand in glove" with the choice of representation**.

- **Real-Valued Variation**
- **Multiparent Recombination Operators**
- **Variations on Variable-Length Structures**

Variation

- **Real-Valued Variation**
  - When searching $\Re^n$, it is typical to use a **Gaussian mutation operator**. Gaussian distribution is defined by **two parameters: the mean and standard deviation** (or variance). Gaussian mutation, the mean is set typically to zero, providing for an unbiased search in the neighborhood of a parent.
  - At times, it may be desirable to have a larger probability of creating a greater distance between an offspring and its parent. A Cauchy random variable is constructed by taking the ratio of two independent identically distributed Gaussian random variables. **The Cauchy distribution has "fatter tails" than a corresponding Gaussian distribution,** thus the probability of mutating a real-valued parameter to a greater extent is markedly greater.





Variation

- **Multiparent Recombination Operators**
  - Earlier discussion highlighted the use of one-point, n-point, and uniform crossovers, as well as blending recombination that averages components of multiple individuals. Nature provides inspiration for evolutionary algorithms but it is up to the designer to find what inspires him or her.
  - **In some cases**, it may be **beneficial to recombine elements or blend parameters of three or more solutions**.

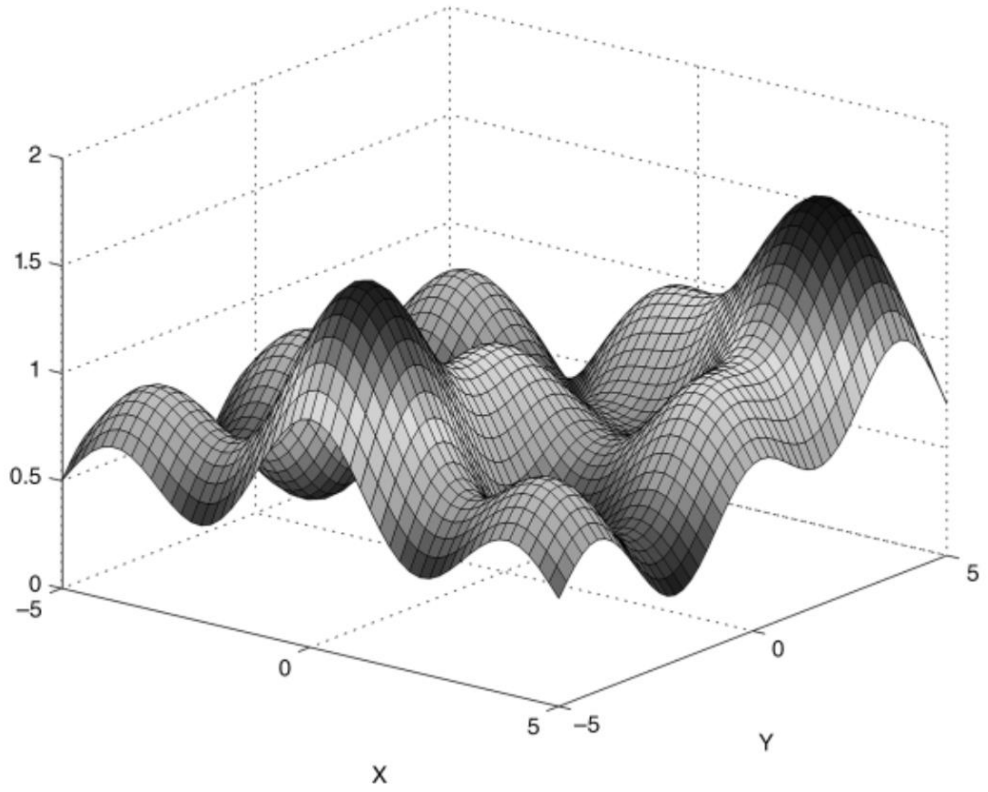
- **Variations on Variable-Length Structures**
  - Certain representations employ **data structures of variable length**. For example, consider the case of evolving a neural network that can adapt not only its weights and bias terms but also the number of nodes it uses and the feedback loops that it employs. Three other common examples involve **finite-state machines**, **symbolic expressions**, and **difference equations**.

DEEC
DEPARTAMENTO DE ENGENHARIA
ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

Variation

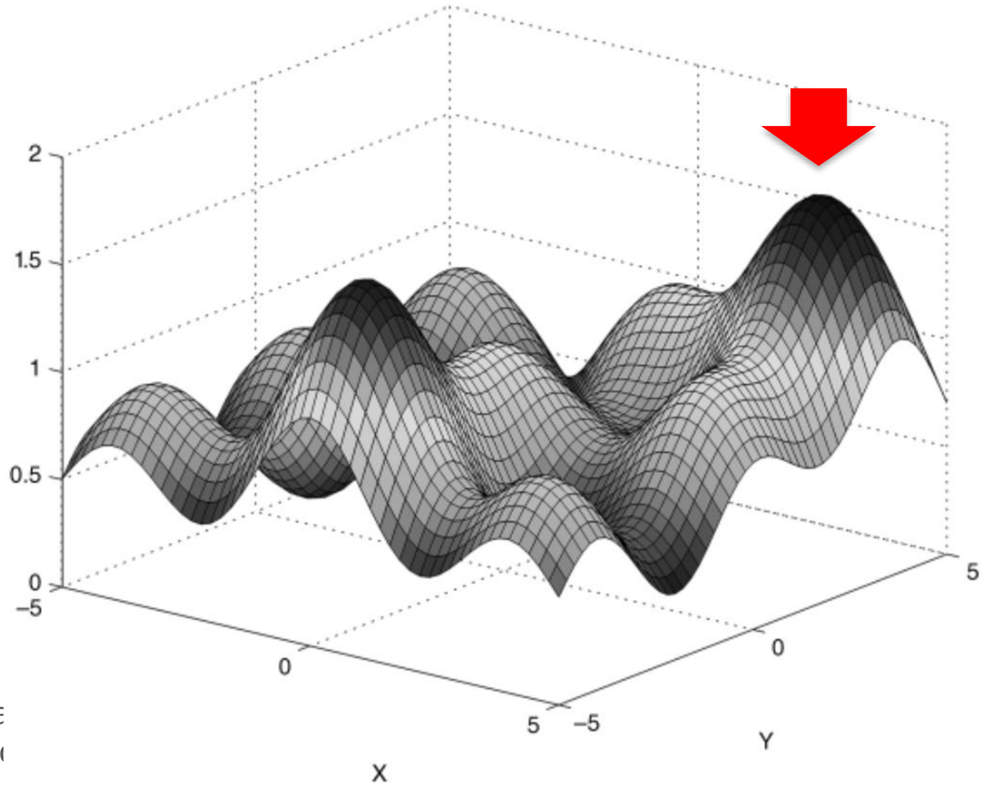
- **Considerations**
  - **One common approach** to evolutionary optimization involves employing a **high rate of crossover** and **low rate of mutation**.
  - As **an example**, let's explore this with a function that poses multiple local optima.

$$f(x, y) = \exp(-[(x-3)^2 + (y-3)^2]/5) + 0.8\exp(-[x^2 + (y+3)^2]/5) + 0.2[\cos(x\pi/2) + \cos(y\pi/2)] + 0.5$$



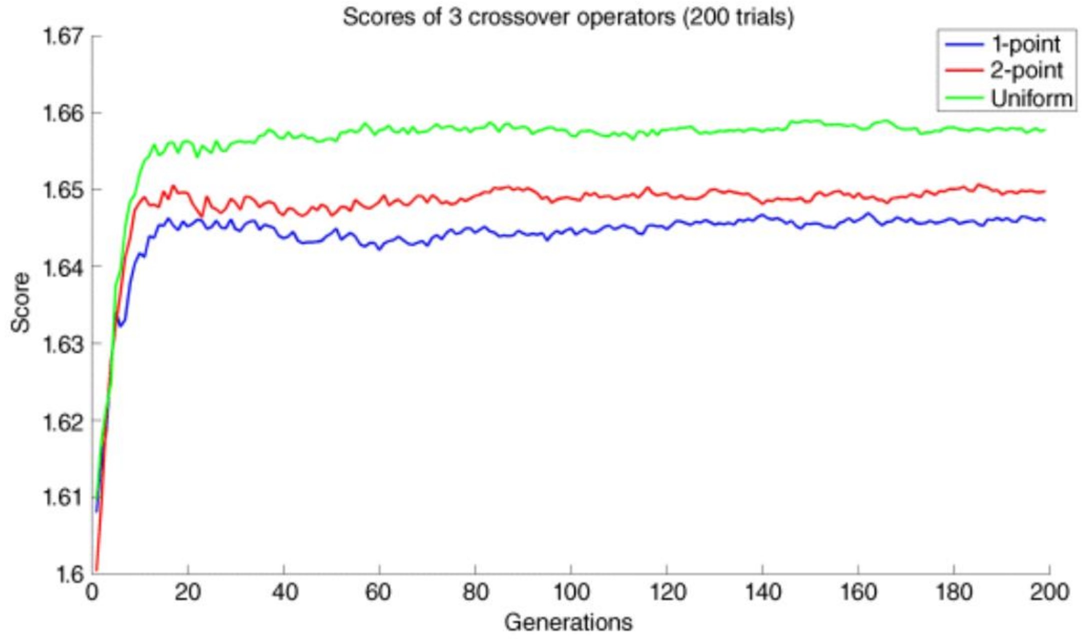
Variation

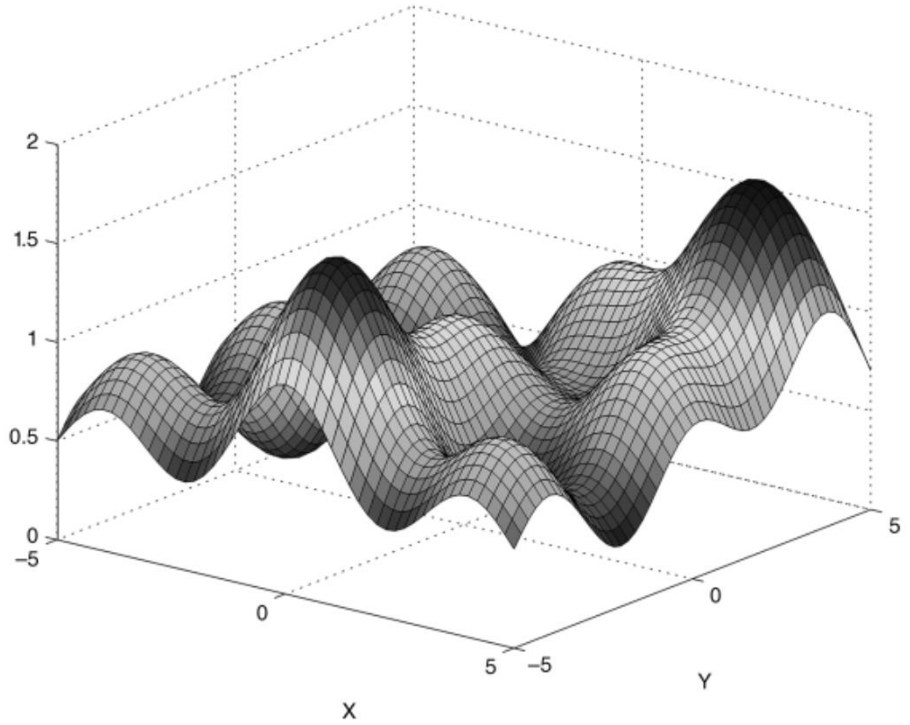
- **Considerations (Example)**
  - The maximum of the function in this range is **1.6903**, which occurs at **(3, 3)**, and let's say we want to find that maximum. In order to assess the possibility for recombining building blocks, we'll choose a **binary representation** that was familiar within one early school of evolutionary algorithms.
  - The representation takes the real values for x and Y and encodes them in eight bits ranging from [00000000]= -5 to [11111111]=5. With eight bits of precision, there are 2^8 possible values for x and y, and a precision of **0.039**. If we wanted **more precision**, we could **increase the number of bits** used to represent the x and y values. For this level of precision, the maximum value of the function is **1.6902**.



Variation

- **Considerations (Example)**
  - We can **compare** the effects of **one-point, two-point, and uniform crossovers** on this function. Fig. shows a comparison over 200 trials, starting in a range of −5 to 5 in x and y, with 100 parents making 100 offspring under **roulette wheel selection** for each of the three recombination operators using **traditional binary encoding**

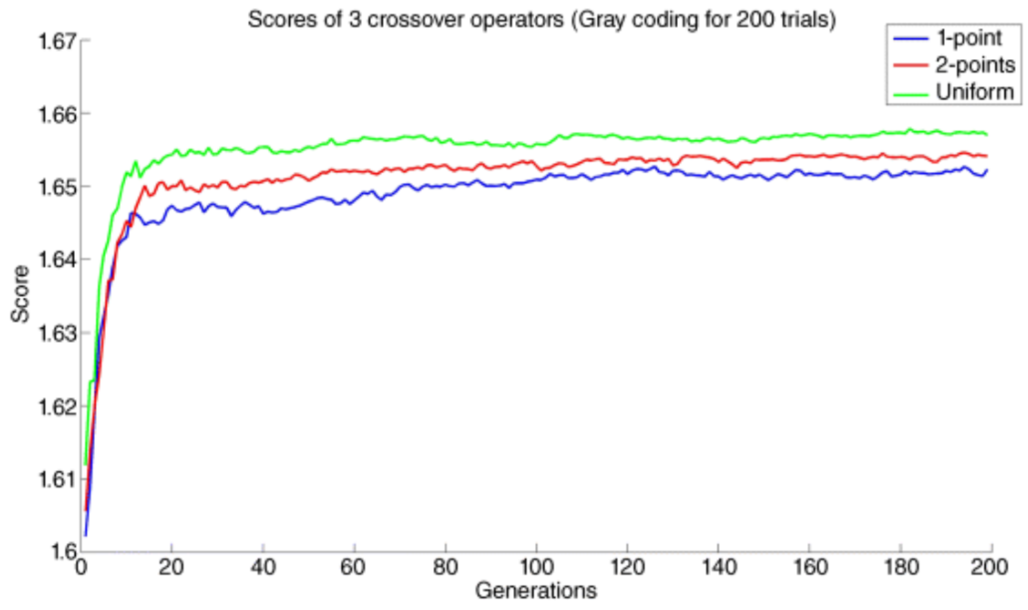


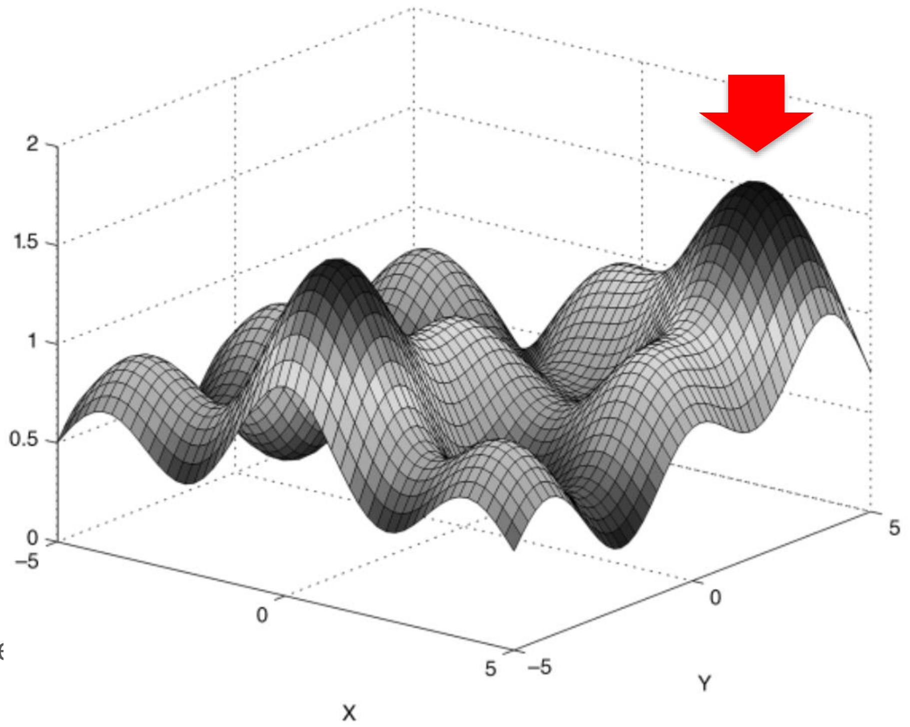


DEEC
DEPARTAMENTO DE ENGENHARIA
ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

Variation

- **Considerations (Example)**
  - We can **compare** the effects of **one-point, two-point, and uniform crossovers** on this function. Fig. shows a comparison over 200 trials, starting in a range of −5 to 5 in x and y, with 100 parents making 100 offspring under roulette wheel selection for each of the three recombination operators using **Gray coding** (which uses binary but encodes numbers so that two successive numbers differ by only one bit).





Variation

- **Considerations**
  - **Suggestion**, try an **evolutionary algorithm that uses a real-valued representation and Gaussian mutation** on this function and see how it compares to the results
  - When designing an evolutionary optimization algorithm keep in mind that **the number of variation operations** applied **in creating an offspring can be greater than 1**.
  - It is easy to become accustomed to a basic plan of applying an evolutionary algorithm for optimization. **Some opt for a plan that involves a high rate of crossover between parents and a low rate of mutation. Others opt for no crossover or other form of recombination at all and employ only mutation** unless the problem at hand suggests that mutation alone is insufficient or inefficient. With experience, you can demonstrate to yourself that neither of these approaches is optimal generally.
  - It is often beneficial to **think imaginatively about how to design variation operators that exploit the characteristics of the objective function** being searched in light of the chosen representation and type of selection.

Constraint Handling

- **Almost all real-world problems are constrained problems**.
- For example, suppose you are **designing an optimal schedule for a bus company**.
  - They have a specific number of existing buses. That is one constraint.
  - They have a limited budget to purchase additional buses. That is another constraint.
  - They have a limited number of qualified drivers for each different type of bus. That is yet another constraint.
  - Each bus has limited capacity, which is yet another constraint.
- We could invent **more constraints for this problem**, including required maintenance, roads that can and cannot be used, the available time for each driver to work each week, and so forth.

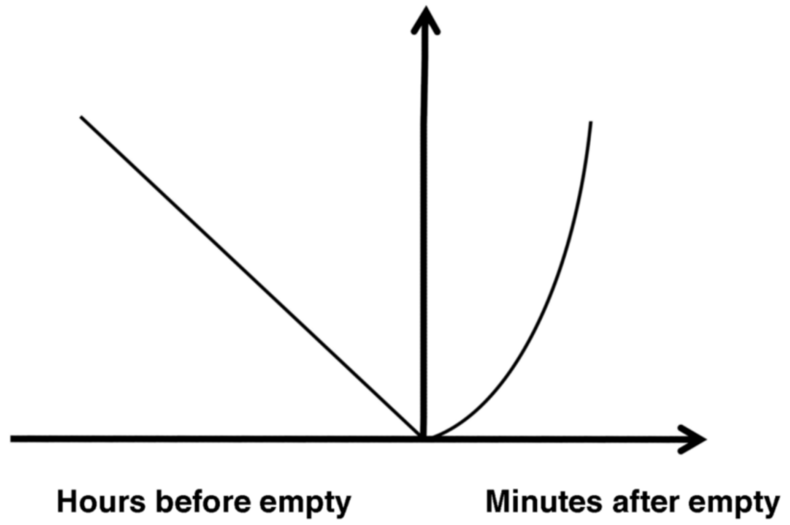
Constraint Handling

- Almost all real-world problems are constrained problems. For example, suppose you are designing an optimal schedule for a bus company. They have a specific number of existing buses. That is one constraint. They have a limited budget to purchase additional buses. That is another constraint. …

- Thus, **when applying evolutionary algorithms, it is important to consider how to treat the constraints of the problem**.
  - **Some of these constraints are part of the objective**, whereas **some are part of** the parameters of a solution and therefore impose boundary conditions on **the search space**.
  - Each of these may pose **hard** or **soft constraints**.

Constraint Handling

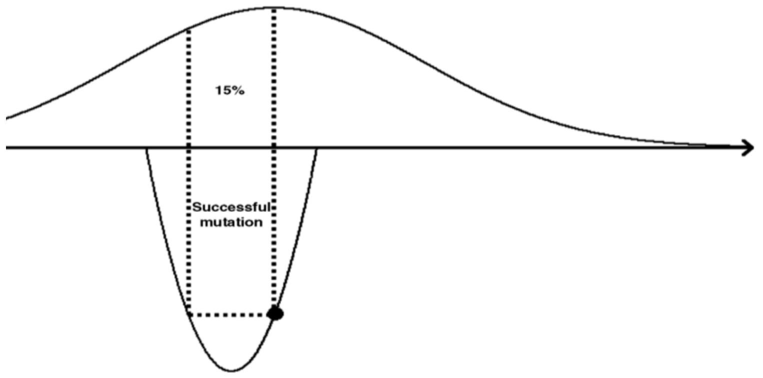
- Thus, when applying evolutionary algorithms, it is important to consider how to treat the constraints of the problem.
  - A **hard constraint** is one that, if violated, makes the entire proposed solution worthless.
  - A **soft constraint** is one that can be violated, but there is some imposed penalty for violating it, and perhaps the penalty increases with the degree of violation.
  - It is often helpful to craft an objective function that comes in two parts. The first involves the primary criteria of interest and the second involves a penalty function that treats constraint violations.

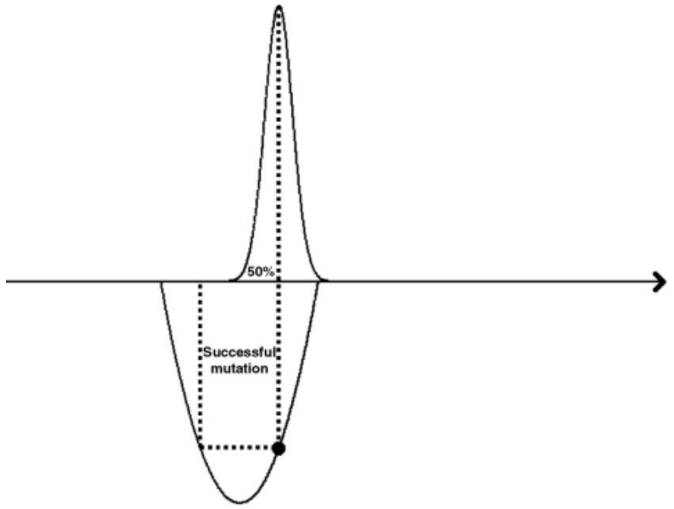


DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

Self-Adaption

- When considering a simple evolutionary algorithm, say, one that employs only Gaussian mutation to real-valued components of potential solutions, x, it is evident that the **setting of the standard deviation of the mutation will affect the degree of progress that can be made toward an optimum**.
- Using a zero-mean Gaussian distribution for mutation, denoted $N(0,\sigma)$, for increasingly **large values of σ**, say $\sigma = 10^6$, the likelihood that an offspring will be better than the parent is extremely low.
- **infinitesimal values of σ,** say 1/(national debt of the United States), the likelihood that an offspring will be better than the parent approaches 0.5. This **increased likelihood of success comes** with the **drawback of a high likelihood of making very little progress** toward the optimum (because the step size is so small)





Self-Adaption

- **The 1/5 Rule**
  - Rechenberg [1973] studied two minimization problems that are linear and quadratic functions of x. … Rechenberg suggested, empirically, **0.2 as the ”ideal” ratio n of successful mutations.**
  - Schwefel [1995] suggested a simple method for computing this empirically. **After every m mutations, determine the success rate over the previous 10m mutations**. If the success rate is less than 0.2, multiply the step size (σ) by 0.85. If the success rate is greater than 0.2, then divide the step size by 0.85.

Self-Adaption

- The 1/5 Rule
  - It is important to recall that the 1/5 rule is a heuristic that was derived only from two simple types of functions, as the number of dimensions tends to infinity
  - **There is no general case to make for the utility of the 1/5 rule**.
  - Still, it illustrates that **static parameters for variation operators are very unlikely to lead to the best rates of progress** toward optimum solutions.
  - Given the limited utility of the 1/5 rule, more general methods for controlling the degree to which variation operators act coarsely or finely are desired.

Self-Adaption

- **Meta-Evolution on Real-Valued Parameters**
  - A **common method** employed in evolutionary **algorithms views the parameters that control the evolutionary search as part of the evolutionary process** to be adapted while searching for an optimum.
  - This poses a form of **meta-evolution** in that adaptation **takes place on two levels** within the evolutionary algorithms: the parameters that are used to evaluate the objective function (**objective parameters**) and the parameters that are used to control variation (**strategy parameters**).

Self-Adaption

- **Meta-Evolution on Real-Valued Parameters**
  - With the problem of minimizing (maximizing) a real-valued function f(x), encode a possible solution as (x,σ), where x is the vector of real-valued objective parameters and σ is the vector of strategy parameters, which designate the positive standard deviation to apply a Gaussian mutation in the given dimension.
  - Each parent creates an offspring via a two-step process applied across all dimensions,
    i = 1, …, n:

$$1.\ \sigma_i' = \sigma_i \exp\left(\tau(N(0,1)) + \tau' N_i(0,1)\right)$$

$$\sigma_i' = \sigma_i \exp\left(\tau' N_i(0,1)\right)$$

$$2.\ x_i' = x_i + N(0, \sigma_i')$$

  - where σ_i is the standard deviation for the offspring in the ith dimension, N(0,1) is a standard Gaussian random variable sampled once and held at the same value for all i dimensions, N(0,1) is a standard Gaussian random variable sampled anew for each of the i dimensions, τ and τ' are constants that are proportional to $1/\sqrt{2\sqrt{n}}$ and $1/\sqrt{2n}$, respectively, and x_i' is the objective parameter value for the offspring in the ith dimension.

Self-Adaption

- Meta-Evolution on Real-Valued Parameters

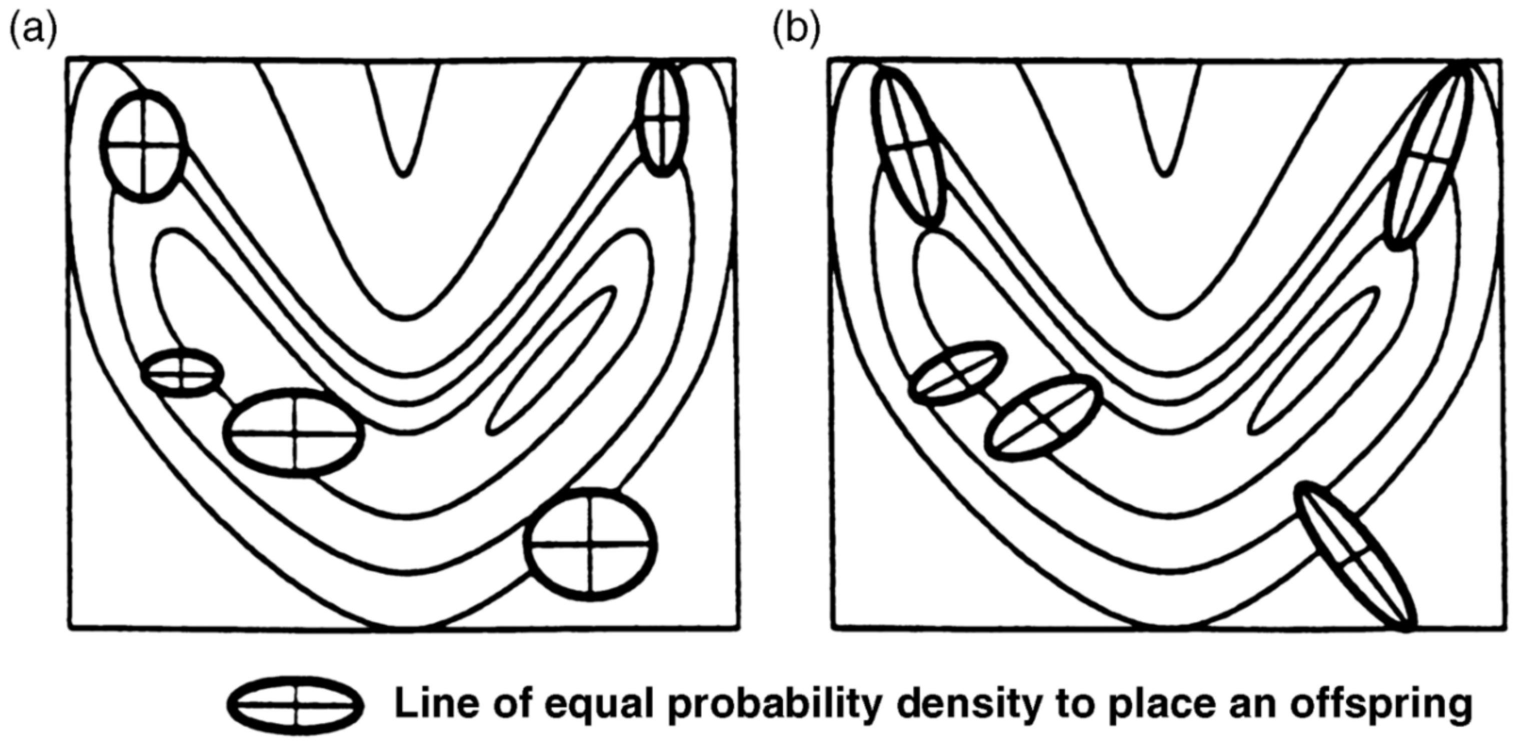


Figure 11.16 Self-adapting the mutation distributions on each solution allows new trials to be generated in light of the contours of the response surface. Independent adjustment of the standard deviation in each dimension provides a mechanism for varying the width of the probability contour in alignment with each axis (a). Correlated standard deviations provide a mechanism for generating trials such that the probability contours are not aligned with the coordinate axes (b). (Taken from Fogel [Fogel, 2000, p. 157] and Back *et al*. [1991].)

Self-Adaption

- **Meta-Evolution on Probabilities of Variation Operators**

  - The concept of **meta-evolution can be extended to support variation operators that are not in the continuous domain**. **Self-adaptation can be applied** both within a variation operator (controlling how that operator functions) and **to determine the likelihood or number of times the variation operator is applied**.

  - Thus, **self-adaptation** is useful potentially not only for applications of evolutionary optimization in $\Re^n$ but also in virtually any real-world application. It **can be used to adjust the likelihood of using mutation, crossover, blending recombination, multiparent recombination**, as well as the types of these operators (e.g., one-point, n-point, uniform crossovers). It is extremely unlikely that any static settings for variation operators will be optimal for solving a problem, and the effectiveness of different variation operators changes during the course of the evolution.

DEEC
DEPARTAMENTO DE ENGENHARIA
ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

Self-Adaption

- **Meta-Evolution on Combinations of Variation Operators**
  – An additional extension of self-adaptive meta-evolutionary methods comes in **blending different operators**. In real-valued evolutionary optimization, there is a trade-off between using Gaussian variation and Cauchy variation. Instead of hand-tuning the probabilities of using these operators, their effects can be adjusted via self-adaptation, implemented potentially as follows:

(1) $\sigma_i' = \sigma_i \exp(\tau' N_i(0,1))$	//step-size control for the Gaussian component
(2) $s_i' = s_i \exp(\tau N_i(0,1))$	//step-size control for the Cauchy component
(3) $\alpha' = \alpha + N(0,\omega)$	//weight to apportion between Gaussian and Cauchy
(4) $x_i' = x_i + \lvert\sin(\alpha')\rvert N(0,\sigma_i') + (1 - \lvert\sin(\alpha')\rvert) C(0,s_i')$	//update with mutation in part by Gaussian and Cauchy components

Self-Adaption

- **Fitness Distributions of Variation Operators**
  - Applying a variation operator to a parent or set of parents generates offspring based on a probability mass or density function. Choosing among variation operators and changing the parameters of those operators alters the likelihood of sampling each point from the solution space. Thus, **a probabilistic distribution of offspring fitness scores can be created given a variation operator and the parent or parents that it operates on**. In certain simple cases, this fitness distribution can be computed mathematically (e.g., leading to Rechenberg's 1/5 rule), but it can be estimated empirically in general cases. Doing so is computationally intensive, as it requires generating a sufficient number of examples of applying an operator to a parent(s) and determining the results statistically; however, the fitness distribution of an operator can yield insight into setting its parameters, how to make them subject to self-adaptation, and whether or not to include them.

Bibliographic References

- **Fundamentals of Computational Intelligence: Neural Networks, Fuzzy Systems, and Evolutionary Computation: : J. Keller, D. Liu, and D. Fogel, 2016 Wiley**

- **Multiobjective Optimization: Interactive and Evolutionary Approaches: J. Branke, K. Deb, K. Miettinen, and R. Słowiński, 2008 Springer**

- **Lecture Slides, “Applied Computational Intelligence”, N. Horta**

next chapter