

Notas de Asignatura

OPTIMIZACIÓN Y METAHEURÍSTICAS

Claudia Nallely Sánchez Gómez

Elaborado por:

Profesora: Claudia Nallely Sánchez Gómez

Universidad Panamericana campus Aguascalientes
Facultad de Ingeniería
Academia de Cómputo



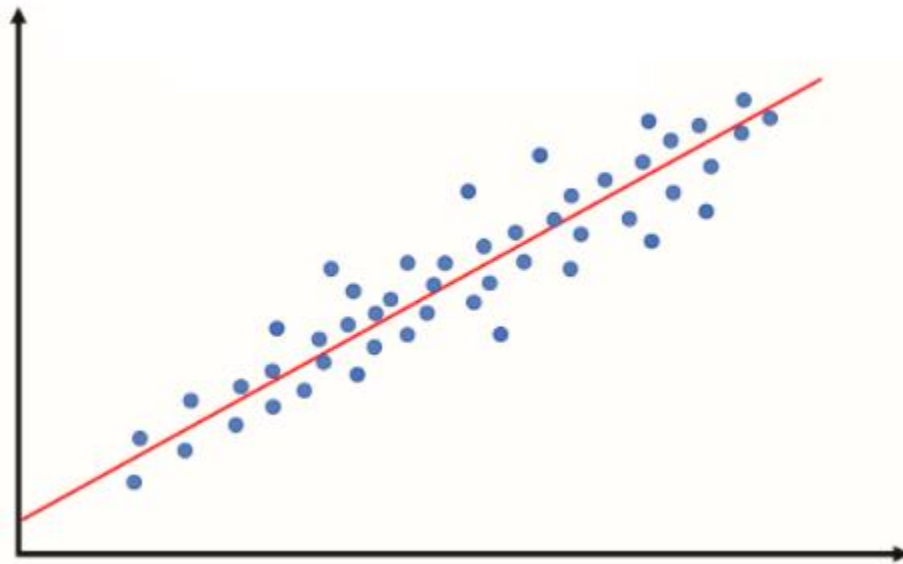
Contenido

Introduction to Optimization	4
What is optimization?.....	5
Applications.....	6
Optimization types	9
Techniques for optimization solving problems.....	9
Deterministic Optimization	10
Gradient vector and Hessian matrix	10
Finite difference.....	12
Gradient with finite difference	12
Hessian with finite difference	13
Gradient and Hessian multiplied by a vector with finite difference	14
Gradient descent	16
Métodos de Región de Confianza	20
Gradiente Conjugado.....	24
Mínimos Cuadrados (Least squares).....	28
Programación Lineal.....	31
Evolutionary Computing	33
Genetic Algorithms (GA).....	36
Evolution Strategies (ES).....	44
Evolutionary Programming (EP)	47
Diferencial Evolution (DE)	49
Constraint Handling	52
Genetic Programming (GP).....	54
Metaheuristics.....	60
Particle Swarm Optimization	60
Ant Colony Optimization (ACO).....	61
Bibliografía.....	64

Introduction to Optimization

Before introducing the optimization concepts, we will think about how to solve a couple of problems.

Problem 1: Imagine you have many points in 2D (x, y), and you want to find the line that fits those points (Linear Regression). How can you calculate the line parameters (slope and intercept)?



Problem 2: Imagine you have several digital images of a landscape. You want to combine them to generate a panoramic photo. For each image, you need to calculate the new position (x, y), the rotation angle (θ), or the new size of the image (α).

How can you calculate those values to generate your panoramic image?



Problem 3: Imagine you have an unbalanced mechanism, and you want to add some counterweights to balance it and reduce the shaking force and shaking moment. Shaking force and shaking moment can be seen as functions that, given the positions (x, y) and thickness (t) of the counterweights, return a value of “stress”.

Ver video: <https://youtu.be/Bk1QmccUaZY>

How can you select the positions and thickness of counterweights for balancing the mechanism?



What is optimization?

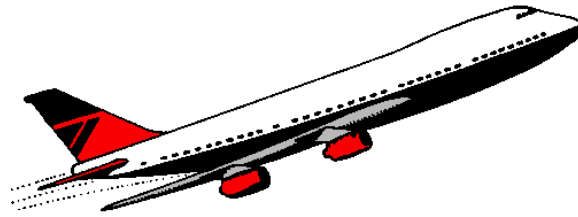
An **optimization** problem consists of **maximizing or minimizing a function by choosing feature values**. Optimization is a discipline of mathematics and mainly uses algorithms for solving problems. It lies in applied mathematics.

The most important in optimization is defining the **objective function**. This performance measurement can distinguish good solutions from bad ones. It represents the quantity we want to maximize or minimize and needs to be a number. For example, it could be money, time, energy, error, force, space, etcetera.

Sometimes the feature values need to accomplish some requirements. For example, the number of people working in a process needs to be more or equal to 1. These requirements are called **constraints**.

Applications

Transportation systems



Definition of distribution centers location



Panoramic images



Images fusion



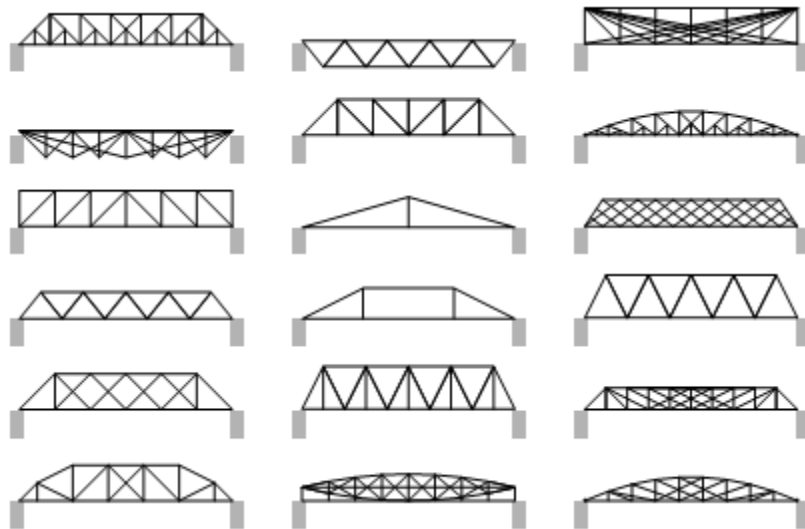
House design plans



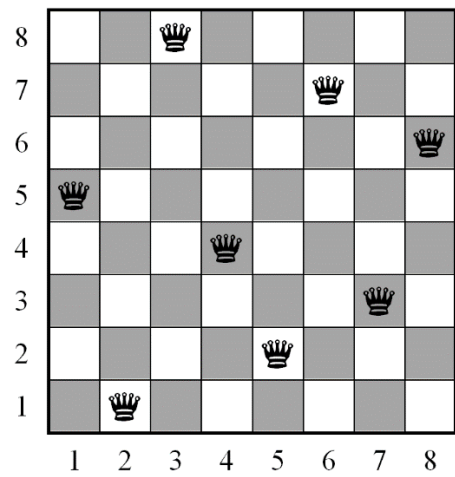
Robotic arms design



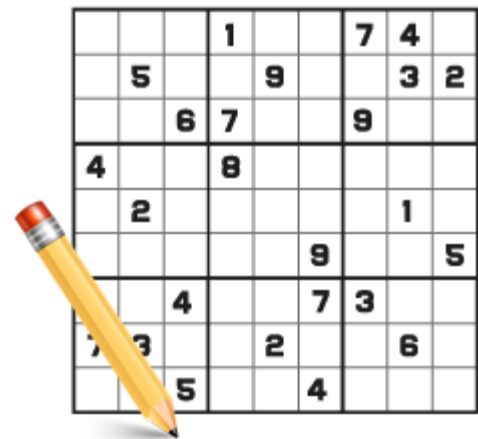
Bridge structures design



8 queens problem



Sudoku



Academic schedule design

Time	Mon	Tue	Wed
7:30 AM	Breakfast	Breakfast	Breakfast
8:00 AM	Business: Lecture Bldg B, Rm 256	Physics: Lab Bldg J, Rm 309	Business: Lecture Bldg B, Rm 256
8:30 AM			
9:00 AM	Applied Math Bldg H, Rm 100		Applied Math Bldg H, Rm 100
9:30 AM			
10:00 AM			
10:30 AM			
11:00 AM			
11:30 AM			

Optimization types

By the variables' types:

- Discrete
 - Knapsack problem
 - Shortest path
 - Traveling salesman problem
- Continuous
 - Panoramic images
 - Robotic arms design

By the characteristics of the problems:

Constrains satisfaction problems: Consists of finding at least a solution that satisfies the constraints.

		Objective function	
		Yes	No
Constraints	Yes	Constrained optimization problem Knapsack problem	Constraint satisfaction problems 8 queen problem
	No	Free optimization problem Linear Regression	There is no problem

Techniques for optimization solving problems

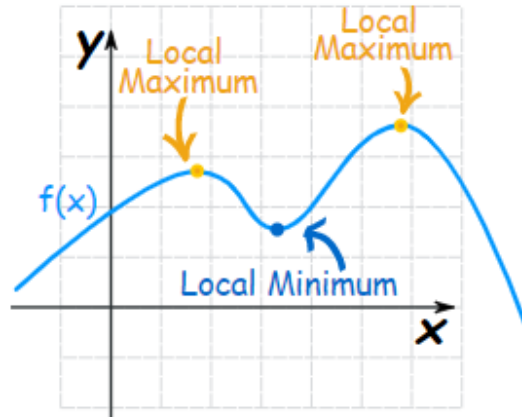
Deterministic: These methods or techniques get the same result, in all the executions with the same inputs. Generally, they use mathematical tools and theorems: calculus, discrete mathematics, etc.

Heuristics: A heuristic is an approach for solving a specific problem. We say that a method is a heuristic if it comes from an idea that it is not strong fundamental.

Metaheuristics: algorithms for solving problems that deterministic techniques cannot solve. Generally, these algorithms are inspired by nature, like the evolutive process, insect behaviors, etc. The results are not warranted; however, they generate very good solutions. They are based on a stochastic process; this is, they can return different results even if they were executed with the same inputs.

Deterministic Optimization

The objective is finding the minimum or the maximum of a function. This point is known as X^* , a vector whose entries correspond to the values of the variables in the optimum point $X^* = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$.



Be $X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ and $f(X)$ a vectorial function $f: R^n \rightarrow R$

Solution: X^* is a global minimum if $f(X^*) \leq f(X) \forall x$

Gradient vector and Hessian matrix

Gradient: $\nabla f(X)$

It represents the first derivative. It contains the partial derivatives of the function respect each variable. Its size is $n \times 1$, where n is the number of variables.

$$\nabla f(X) = \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

Hessian $\nabla^2 f(X)$

It represents the second derivative. It contains the second partial derivatives of the function respect each pair of variables. Its size is $n \times n$, where n is the number of variables. It is a symmetric matrix because $\frac{\partial^2 f}{\partial x y} = \frac{\partial^2 f}{\partial y x}$.

$$\nabla^2 f(X) = \nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

X^* is a local minimum if:

- $\|\nabla f(X^*)\| = 0$
- $\nabla^2 f(X^*)$ is positive semidefinite, this is, $v^T \nabla^2 f(X^*) v \geq 0 \quad \forall v$

Remembering, in Calculus x is a minimum if:

- $f'(x) = 0$
- $f''(x) > 0$

Example: Calculate the Gradient and Hessian

$$f(x, y, z) = 2x^3 + 5x^2y - xz^4 - e^{2y}$$

Gradient

$$\nabla f(X) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{bmatrix} = \begin{bmatrix} 6x^2 + 10xy - z^4 \\ 5x^2 - 2e^{2y} \\ -4xz^3 \end{bmatrix}$$

Hessian

$$\nabla^2 f(X) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial xy} & \frac{\partial^2 f}{\partial xz} \\ \frac{\partial^2 f}{\partial yx} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial yz} \\ \frac{\partial^2 f}{\partial zx} & \frac{\partial^2 f}{\partial zy} & \frac{\partial^2 f}{\partial z^2} \end{bmatrix} = \begin{bmatrix} 12x + 10y & 10x & -4z^3 \\ 10x & -4e^{2y} & 0 \\ -4z^3 & 0 & -12xz^2 \end{bmatrix}$$

Activity: Calculate the Gradient and Hessian of the following functions

1. $f(x, y, z) = x^2y + 3xz^3 + 2z^2$

2. $f(x_1, x_2) = 3x_1^2 + 2x_1x_2 + 4x_2^3$

3. $h(a, b) = a^4b^3 + \frac{\exp(a^3)}{4} + \frac{a^2}{2}$

4. $\Phi(a, b, c, d) = ab^3c + \frac{db^2}{2} - 3a + 14$

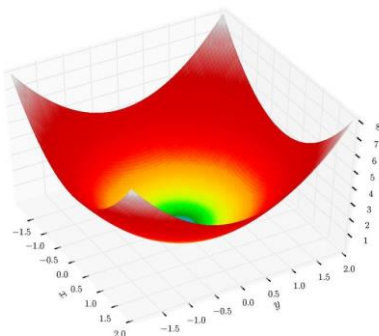
Examples of functions:

1. Sphere $f(x) = \sum_{i=1}^n x_i^2$

2. Rosenbrock $f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$

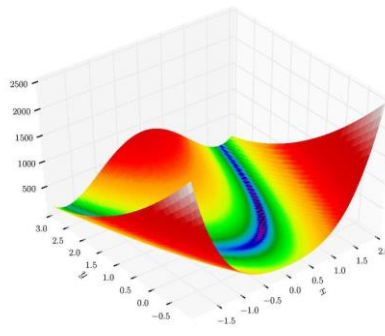
3. Ackley $f(x) = 20 + e - 20 \exp \left[-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \right] - \exp \left(\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i) \right)$

1. Sphere



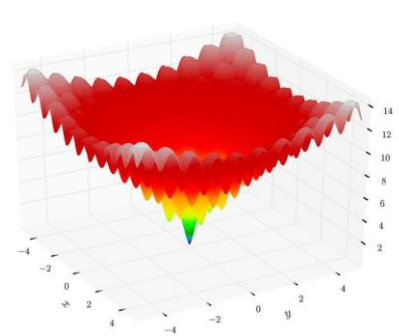
Minimum: (0,...,0)

2. Rosenbrock



Minimum: (1,...,1)

3. Ackley



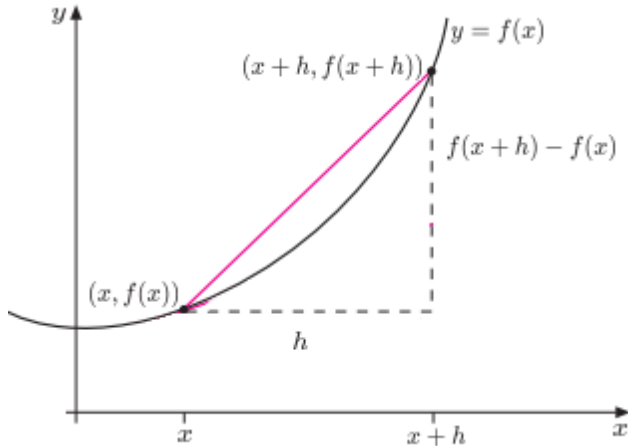
Minimum: (0,...,0)

Finite difference

Finite difference is a technique for approximating the Gradient vector and Hessian matrix in a specific point with a numerical formula.

Gradient with finite difference

Using the derivative definition:



$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

And knowing that the gradient vector contains the partial derivatives $\nabla f(X) = \left[\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right]^T$, it could be calculated with the formula:

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x + \varepsilon e_i) - f(x)}{\varepsilon} \approx \frac{f(x + \varepsilon e_i) - f(x - \varepsilon e_i)}{2\varepsilon} \approx \frac{f(x) - f(x - \varepsilon e_i)}{\varepsilon}$$

where:

ε is a small scalar ($\varepsilon = 1e^{-5}$)

e_i is the i-th unit vector, for example $e_2 = [0, 1, \dots, 0]^T$

Example: Calculate the Gradient vector using finite difference

$f(x, y) = x^2 + y^2$, Calculate $\nabla f([2, 3]^T)$

Finite difference ($\varepsilon = 0.1$)

$$\nabla f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} + 0.1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right)}{0.1} \\ \frac{f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} + 0.1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right)}{0.1} \end{bmatrix} = \begin{bmatrix} \frac{f \left(\begin{bmatrix} 2.1 \\ 3 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right)}{0.1} \\ \frac{f \left(\begin{bmatrix} 2 \\ 3.1 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right)}{0.1} \end{bmatrix} = \begin{bmatrix} \frac{(2.1^2 + 3^2) - (2^2 + 3^2)}{0.1} \\ \frac{(2^2 + 3.1^2) - (2^2 + 3^2)}{0.1} \end{bmatrix} = \begin{bmatrix} 4.1 \\ 6.1 \end{bmatrix}$$

Analytical derivatives

$$\nabla f \left(\begin{bmatrix} x \\ y \end{bmatrix} \right) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix} \quad \nabla f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right) = \begin{bmatrix} 2(2) \\ 2(3) \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \end{bmatrix}$$

Activity:

1. Calculate the Gradient vector using analytical derivatives and finite difference.

$$f(x, y) = 2x^2y + 8y^3, \quad \nabla f(-2, 4)$$

$$\varphi(x_1, x_2, x_3) = 2x_1^4x_2^2 - x_3, \quad \nabla \varphi(10, 25, -50)$$

$$\phi(i, j, k) = 3ij^3 - k^3i^2 + 5, \quad \nabla \phi(2, -1, 4)$$

2. Code a function that receives a function and a point as parameters and returns the Gradient vector.

Hessian with finite difference

The Hessian matrix is formed by the second partial derivatives:

$$\nabla^2 f(X) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

We can say that $\nabla^2 f(X) \approx \left[\frac{\partial \nabla f(x)}{\partial x_1}, \frac{\partial \nabla f(x)}{\partial x_2}, \dots, \frac{\partial \nabla f(x)}{\partial x_n} \right]$, this is, $\nabla^2 f$ is formed by the partial derivative of the Gradient derivative respect to each variable. For calculating the partial derivative of the Gradient we can use the following equation:

$$\frac{\partial \nabla f(x)}{\partial x_i} \approx \frac{\nabla f(x + \epsilon e_i) - \nabla f(x)}{\epsilon} \approx \frac{\nabla f(x + \epsilon e_i) - \nabla f(x - \epsilon e_i)}{2\epsilon} \approx \frac{\nabla f(x) - \nabla f(x - \epsilon e_i)}{\epsilon}$$

Example: Calculate the Hessian using finite difference

$f(x, y) = x^2 + y^2$, Calculate $\nabla^2 f([2, 3]^T)$

Finite difference ($\epsilon = 0.1$)

$$\nabla^2 f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right) = \left[\frac{\partial \nabla f}{\partial x}, \frac{\partial \nabla f}{\partial y} \right] = \left[\frac{\nabla f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} + 0.1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) - \nabla f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right)}{0.1}, \frac{\nabla f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} + 0.1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) - \nabla f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right)}{0.1} \right] = \left[\frac{\nabla f \left(\begin{bmatrix} 2.1 \\ 3 \end{bmatrix} \right) - \nabla f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right)}{0.1}, \frac{\nabla f \left(\begin{bmatrix} 2 \\ 3.1 \end{bmatrix} \right) - \nabla f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right)}{0.1} \right]$$

$$\nabla f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right) = \begin{bmatrix} \frac{f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} + 0.1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right)}{0.1} \\ \frac{f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} + 0.1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right)}{0.1} \end{bmatrix} = \begin{bmatrix} \frac{f \left(\begin{bmatrix} 2.1 \\ 3 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right)}{0.1} \\ \frac{f \left(\begin{bmatrix} 2 \\ 3.1 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right)}{0.1} \end{bmatrix} = \begin{bmatrix} \frac{(2.1^2 + 3^2) - (2^2 + 3^2)}{0.1} \\ \frac{(2^2 + 3.1^2) - (2^2 + 3^2)}{0.1} \end{bmatrix} = \begin{bmatrix} 4.1 \\ 6.1 \end{bmatrix}$$

$$\nabla f \left(\begin{bmatrix} 2.1 \\ 3 \end{bmatrix} \right) = \begin{bmatrix} \frac{f \left(\begin{bmatrix} 2.1 \\ 3 \end{bmatrix} + 0.1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2.1 \\ 3 \end{bmatrix} \right)}{0.1} \\ \frac{f \left(\begin{bmatrix} 2.1 \\ 3 \end{bmatrix} + 0.1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2.1 \\ 3 \end{bmatrix} \right)}{0.1} \end{bmatrix} = \begin{bmatrix} \frac{f \left(\begin{bmatrix} 2.2 \\ 3 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2.1 \\ 3 \end{bmatrix} \right)}{0.1} \\ \frac{f \left(\begin{bmatrix} 2.1 \\ 3.1 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2.1 \\ 3 \end{bmatrix} \right)}{0.1} \end{bmatrix} = \begin{bmatrix} \frac{(2.2^2 + 3^2) - (2.1^2 + 3^2)}{0.1} \\ \frac{(2.1^2 + 3.1^2) - (2.1^2 + 3^2)}{0.1} \end{bmatrix} = \begin{bmatrix} 4.3 \\ 6.1 \end{bmatrix}$$

$$\nabla f \left(\begin{bmatrix} 2 \\ 3.1 \end{bmatrix} \right) = \begin{bmatrix} \frac{f \left(\begin{bmatrix} 2 \\ 3.1 \end{bmatrix} + 0.1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2 \\ 3.1 \end{bmatrix} \right)}{0.1} \\ \frac{f \left(\begin{bmatrix} 2 \\ 3.1 \end{bmatrix} + 0.1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2 \\ 3.1 \end{bmatrix} \right)}{0.1} \end{bmatrix} = \begin{bmatrix} \frac{f \left(\begin{bmatrix} 2.1 \\ 3.1 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2 \\ 3.1 \end{bmatrix} \right)}{0.1} \\ \frac{f \left(\begin{bmatrix} 2 \\ 3.2 \end{bmatrix} \right) - f \left(\begin{bmatrix} 2 \\ 3.1 \end{bmatrix} \right)}{0.1} \end{bmatrix} = \begin{bmatrix} \frac{(2.1^2 + 3.1^2) - (2^2 + 3.1^2)}{0.1} \\ \frac{(2^2 + 3.2^2) - (2^2 + 3.1^2)}{0.1} \end{bmatrix} = \begin{bmatrix} 4.1 \\ 6.3 \end{bmatrix}$$

$$\nabla^2 f \left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} \right) = \begin{bmatrix} \frac{4.3 - 4.1}{0.1}, \frac{6.3 - 6.1}{0.1} \end{bmatrix} = \begin{bmatrix} \frac{0.2}{0.1}, \frac{0.2}{0.1} \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

Analytical derivatives

$$\nabla^2 f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial xy} \\ \frac{\partial^2 f}{\partial yx} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \quad \nabla f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix}\right) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

We can add the following to guarantee that the Hessian is symmetric:

$$H = \frac{H + H^T}{2}$$

Activity:

1. Calculate the Hessian matrix using analytical derivatives and finite difference.

$$f(x, y) = 2x^2y + 8y^3, \quad \nabla^2 f([-2, 4]) = ?$$

$$\phi(i, k) = 3i^3 - 5k^2, \quad \nabla \phi([2, -1]) = ?$$

2. Code a function that receives a function and a point as parameters and returns the Hessian matrix.

Gradient and Hessian multiplied by a vector with finite difference

Gradient multiplied by a vector:

$$\nabla f(x)^T d \approx \frac{f(x + \epsilon d) - f(x)}{\epsilon}$$

The result is a scalar.

Hessian multiplied by a vector:

$$\nabla^2 f(x) d \approx \frac{\nabla f(x) - \nabla f(x - \epsilon d)}{\epsilon}$$

The result is a vector whose size is nx1.

Example: Calculate the Gradient multiplied by a vector

$$f(x, y) = x^2 + y^2, \quad \nabla f(2, 3)^T \begin{bmatrix} 5 \\ 1 \end{bmatrix} = ? , \quad \nabla^2 f(2, 3) \begin{bmatrix} 5 \\ 1 \end{bmatrix} = ?$$

Finite difference ($\epsilon = 0.1$)

$$\nabla f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix}\right)^T \begin{bmatrix} 5 \\ 1 \end{bmatrix} = \frac{f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} + 0.1 \begin{bmatrix} 5 \\ 1 \end{bmatrix}\right) - f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix}\right)}{0.1} = \frac{f\left(\begin{bmatrix} 2.5 \\ 3.1 \end{bmatrix}\right) - f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix}\right)}{0.1} = \frac{(2.5^2 + 3.1^2) - (2^2 + 3^2)}{0.1} = 28.6$$

$$\nabla^2 f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix}\right)^T \begin{bmatrix} 5 \\ 1 \end{bmatrix} = \frac{\nabla f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} + 0.1 \begin{bmatrix} 5 \\ 1 \end{bmatrix}\right) - \nabla f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix}\right)}{0.1} = \frac{\nabla f\left(\begin{bmatrix} 2.5 \\ 3.1 \end{bmatrix}\right) - \nabla f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix}\right)}{0.1} = \frac{\begin{bmatrix} 5.1 \\ 6.3 \end{bmatrix} - \begin{bmatrix} 4.1 \\ 6.1 \end{bmatrix}}{0.1} = \frac{\begin{bmatrix} 1 \\ 0.2 \end{bmatrix}}{0.1} = \begin{bmatrix} 10 \\ 2 \end{bmatrix}$$

$$\nabla f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix}\right) = \frac{f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix} + 0.1 \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) - f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix}\right)}{0.1} = \frac{f\left(\begin{bmatrix} 2.1 \\ 3 \end{bmatrix}\right) - f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix}\right)}{0.1} = \frac{(2.1^2 + 3^2) - (2^2 + 3^2)}{0.1} = \begin{bmatrix} 4.1 \\ 6.1 \end{bmatrix}$$

$$\nabla f\left(\begin{bmatrix} 2.5 \\ 3.1 \end{bmatrix}\right) = \frac{f\left(\begin{bmatrix} 2.5 \\ 3.1 \end{bmatrix} + 0.1 \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) - f\left(\begin{bmatrix} 2.5 \\ 3.1 \end{bmatrix}\right)}{0.1} = \frac{f\left(\begin{bmatrix} 2.6 \\ 3.1 \end{bmatrix}\right) - f\left(\begin{bmatrix} 2.5 \\ 3.1 \end{bmatrix}\right)}{0.1} = \frac{(2.6^2 + 3.1^2) - (2.5^2 + 3.1^2)}{0.1} = \begin{bmatrix} 5.1 \\ 6.3 \end{bmatrix}$$

Analytical derivatives

$$\nabla f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix}\right)^T \begin{bmatrix} 5 \\ 1 \end{bmatrix} = [2x \ 2y] \begin{bmatrix} 5 \\ 1 \end{bmatrix} = [4 \ 6] \begin{bmatrix} 5 \\ 1 \end{bmatrix} = 26$$

$$\nabla^2 f\left(\begin{bmatrix} 2 \\ 3 \end{bmatrix}\right) \begin{bmatrix} 5 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 5 \\ 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 2 \end{bmatrix}$$

Activity:

1. Calculate (using analytical derivatives and finite difference):

$$f(x, y) = 2x^2y + 8y^3, \quad \nabla f(-2, 4)^T \begin{bmatrix} 3 \\ 1 \end{bmatrix} = ? , \quad \nabla^2 f(-2, 4)^T \begin{bmatrix} 3 \\ 1 \end{bmatrix} = ?$$

$$\phi(i, k) = 3i^3 - 5k^2, \quad \nabla\phi(2, -1, 4)^T \begin{bmatrix} 5 \\ 2 \end{bmatrix} = ? , \quad \nabla\phi(2, -1, 4)^T \begin{bmatrix} 5 \\ 2 \end{bmatrix} = ?$$

2. Code a function that receives a function, a vector, and a point as parameters and returns the Gradient multiplied by the vector.

3. Code a function that receives a function, a vector, and a point as parameters and returns the Hessian multiplied by the vector.

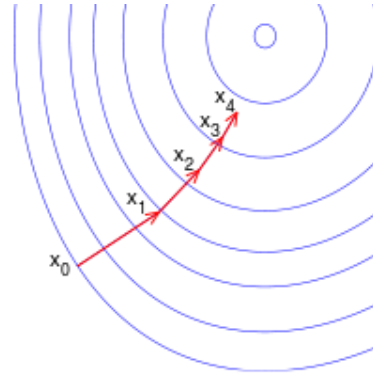
Gradient descent

Gradient descent is an algorithm for finding the local minimum in a multivariate function. Given an initial point X_0 , it improves the solution in several iterations. It stops when there is not better solution or when a good solution has been founded.

$$X_{k+1} = X_k + \alpha_k P_k$$

Each iteration we need to calculate:

- The **direction** p_k (P_k must be a unit vector)
- The **step size** α_k (α_k must be a scalar)



Algorithm

Parameter: Initial point X_0

While ($k < \text{MaxIterations}$ && $||\nabla f(X_k)|| > \varepsilon$)

 Calculate the step size α_k

 Calculate the direction P_k

$X_{k+1} = X_k + \alpha_k P_k$

$k = k + 1$

Return X_k

Calculate the direction P_k

Descent Gradient essence is to move each iteration in the inverse of the gradient direction. The direction must be a unit vector to don't affect the step size. It is calculated as follows:

$$P_k = \frac{-\nabla f(X_k)}{||\nabla f(X_k)||}$$

where $||V|| = \sqrt{V_1^2 + V_2^2 + \dots + V_n^2}$.

Calculate the step size α_k

For calculating the step size:

- **Constant**, for example:

$$\alpha_k = 0.01 \quad o \quad \alpha_k = 1 \times 10^{-5}$$

- **Newton step**, from the Taylor's Theorem:

$$f(x_k + \alpha_k p_k) = f(x_k) + \nabla f(x_k)^T (\alpha_k p_k) + \frac{(\alpha_k p_k)^T \nabla^2 f(x) (\alpha_k p_k)}{2} + \dots$$

Optimizing with respect to α_k

$$\frac{d f(x_k + \alpha_k p_k)}{d \alpha_k} = \nabla f(x_k)^T p_k + \alpha_k p_k^T \nabla^2 f(x) p_k = 0$$

$$\alpha_k p_k^T \nabla^2 f(x) p_k = -\nabla f(x_k)^T p_k$$

$$\alpha_k = \frac{-\nabla f(x_k)^T p_k}{p_k^T \nabla^2 f(x_k) p_k}$$

- **Backtracking**, it uses the following Wolfe condition:

$$f(X_k + \alpha_k P_k) \leq f(X_k) + c \nabla f(X_k)^T P_k \quad \text{with } 0 < c < 1$$

Initial algorithm

Parameters: $\alpha > 0$, $0 < \rho, c < 1$

$\alpha_k = \alpha$

While $f(X_k + \alpha_k P_k) > f(X_k) + c \nabla f(X_k)^T P_k$

$\alpha_k = \rho \alpha_k$

Return α_k

Backtracking with inertia - Algorithm

// a and m are global variables and they are initialized as m=0, a=1

Parameters: a, m

$\alpha_k = a$

While $f(X_k + \alpha_k P_k) > f(X_k) + c_0 \nabla f(X_k)^T P_k$ // If α_k is not good enough, it becomes smaller

$m = 0$

$\alpha_k = \frac{\alpha_k}{c_1}$

Si $\alpha_k < \varepsilon$: break

End while

$m = m + 1$ // Counting the times that we use the same step size α_k

If $m > c_2$ // If c_2 times we have been used the same step size, it becomes bigger

$m = 0$

$a = c_3 \alpha_k$

Else // Save the step size

$a = \alpha_k$

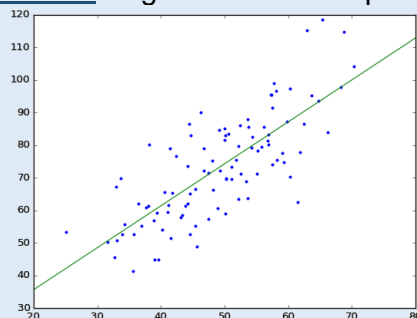
End if

Return and update α_k, a, m

Recommended values

$$a = 1, c_0 = 1 \times 10^{-4}, c_1 = 2, c_2 = 5, c_3 = 3$$

Actividad: Regresión lineal simple



Línea que se quiere encontrar:

$$y = mx + b$$

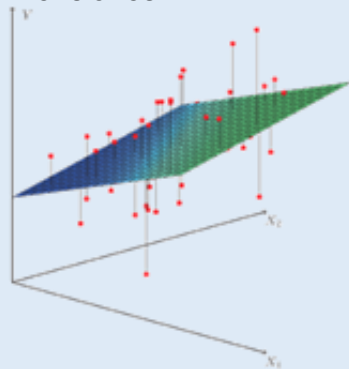
La función a optimizar es:

$$\min_{m,b} \theta = \frac{1}{2} \sum_{i=1}^M (Y^i - (mX^i + b))^2$$

$$\nabla \theta = \begin{bmatrix} \sum_{i=1}^M (Y^i - (mX^i + b)) (-X^i) \\ \sum_{i=1}^M (Y^i - (mX^i + b)) (-1) \end{bmatrix}$$

$$\nabla^2 \theta = \begin{bmatrix} \sum_{i=1}^M (X^i)^2 & \sum_{i=1}^M (X^i) \\ \sum_{i=1}^M (X^i) & M \end{bmatrix}$$

En General, se puede hacer regresión lineal en N dimensiones. En la siguiente figura se muestra una regresión en 2 dimensiones:



Función del hiperplano:

$$f(X) = B_0 + B_1 X_1 + \dots + B_N X_N$$

N es la dimensión de los datos

B son los valores a optimizar

La función a optimizar es:

$$\min_B \theta(B) = \frac{1}{2} \sum_{i=1}^M (f(X^i) - Y^i)^2,$$

M es el número de muestras que se tienen

Gradiente

$$\nabla \theta = \begin{bmatrix} \frac{\partial \theta}{\partial B_0} \\ \frac{\partial \theta}{\partial B_1} \\ \vdots \\ \frac{\partial \theta}{\partial B_N} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^M (f(X^i) - Y^i) \\ \sum_{i=1}^M (f(X^i) - Y^i) (X_1^i) \\ \vdots \\ \sum_{i=1}^M (f(X^i) - Y^i) (X_N^i) \end{bmatrix}$$

Hessiano

$$\nabla^2 \theta = \begin{bmatrix} \frac{\partial^2 \theta}{\partial B_0^2} & \frac{\partial^2 \theta}{\partial B_0 \partial B_1} & \dots & \frac{\partial^2 \theta}{\partial B_0 \partial B_N} \\ \frac{\partial^2 \theta}{\partial B_1 \partial B_0} & \frac{\partial^2 \theta}{\partial B_1^2} & \dots & \frac{\partial^2 \theta}{\partial B_1 \partial B_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \theta}{\partial B_N \partial B_0} & \frac{\partial^2 \theta}{\partial B_N \partial B_1} & \dots & \frac{\partial^2 \theta}{\partial B_N^2} \end{bmatrix} = \begin{bmatrix} M & \sum_{i=1}^M X_1^i & \dots & \sum_{i=1}^M X_N^i \\ \sum_{i=1}^M X_1^i & \sum_{i=1}^M X_1^i X_1^i & \dots & \sum_{i=1}^M X_1^i X_N^i \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^M X_N^i & \sum_{i=1}^M X_1^i X_N^i & \dots & \sum_{i=1}^M X_N^i X_N^i \end{bmatrix}$$

Métodos de Búsqueda en Línea

En general, el método del Descenso del Gradiente pertenece a la familia de los métodos de Búsqueda en Línea. El nombre de estos métodos es porque son iterativos de la siguiente forma:

$$x_{k+1} = x_k + \alpha_k p_k$$

En cada iteración se debe determinar:

- La dirección hacia donde se debe mover p_k (p_k debe ser un vector unitario)
- El tamaño de paso, el cual indica que tanto se debe mover α_k (α_k debe ser un escalar)

Otras formas para calcular la dirección de descenso es:

- **Newton**

Para calcular la dirección P_k se resuelve el siguiente sistema de ecuaciones:

$$\nabla^2 f(X_k) P_k = -\nabla f(X_k)$$

Y después se calcula el vector unitario.

- **Quasi Newton**

Parecido a Newton, difiere en que utiliza una aproximación al Hessiano.

$$B_k \approx \nabla^2 f(X_k)$$

Y al igual que en Newton se resuelve el sistema de ecuaciones y después se calcula el vector unitario.

$$[B_k] P_k = -\nabla f(X_k)$$

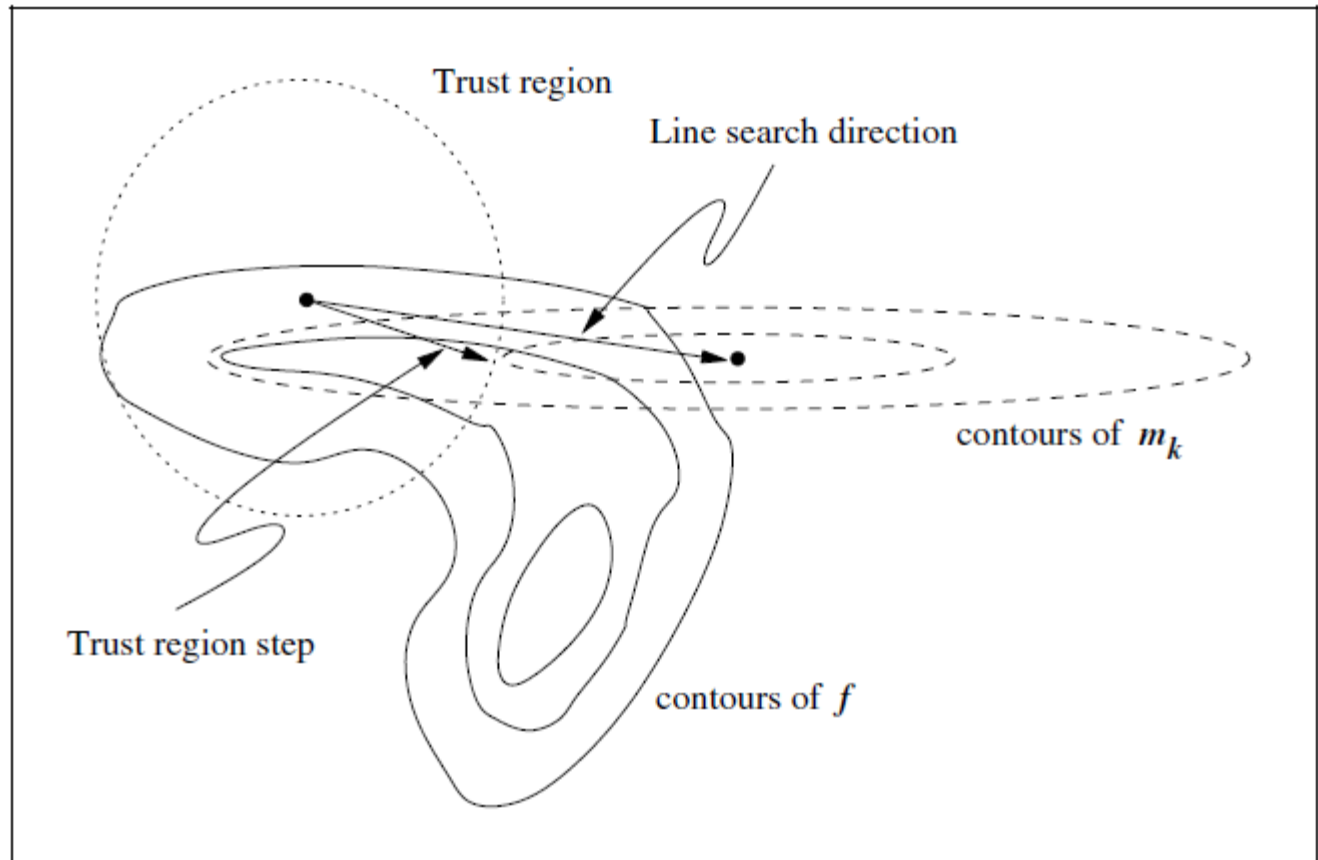
Métodos de Región de Confianza

Los Métodos de Región de Confianza se utilizan para optimizar funciones; se basan en encontrar el punto óptimo en una serie de iteraciones donde cada punto es mejor al anterior, la estrategia es resolver en cada iteración un modelo que sea más simple de resolver y limitar el paso a la región donde el modelo ajuste bien a la función.

Sea k la iteración actual y x_k el punto actual, entonces el modelo es:

$$m_k(p) = f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k p$$

Donde: $B_k = \nabla^2 f_k + \lambda I$, λ pequeña (probar 0.1)



Lo que se hace en este tipo de métodos es:

Dado un X_k

1. Construir un modelo $m_k(p)$.
 2. Resolver el problema $\min_{p \in \mathbb{R}^n} m_k(p)$.
 3. Truncar el modelo dependiendo de su eficiencia, y actualizar X_k si es pertinente.
- Y esto se itera hasta llegar a la convergencia

Para que la modificación del punto sea “segura”, se establece un radio en cada iteración Δ_k donde el modelo es válido. Según el tamaño del radio, el incremento P_k se calcula de la siguiente manera:

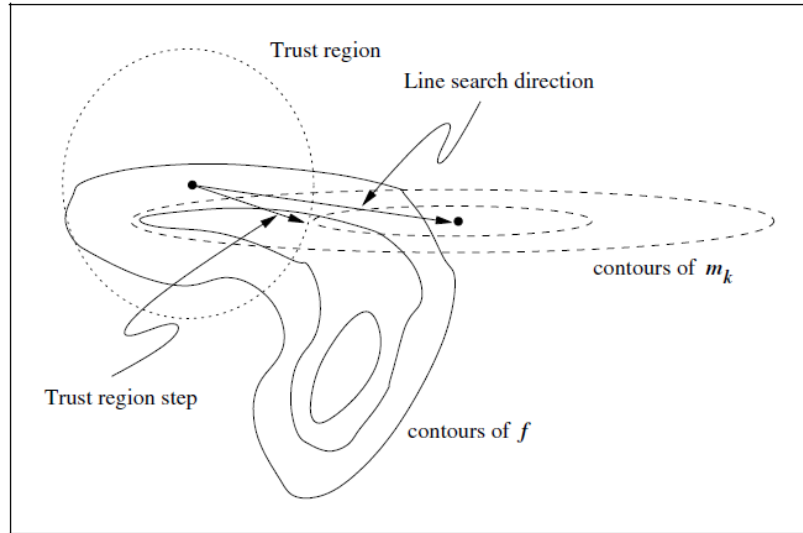
$$P_k^B = -B_k^{-1} \nabla f_k$$

$$P_k^U = -\frac{\nabla f_k}{\|\nabla f_k\|}$$

Si $\|P_k^B\| \leq \Delta_k$, entonces $P_k = P_k^B$

Si no, $P_k = \Delta_k P_k^U$

El radio Δ_k se ajusta en cada iteración. Además, el punto se actualiza sólo si el modelo “ajusta” bien al problema con el radio dado.



La medida o razón de eficiencia del modelo está dada por

$$\delta_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)} = \frac{\text{Reducción de la función}}{\text{Reducción del modelo}}$$

El análisis de esta eficiencia se puede ver de la siguiente manera:

	Casos que pueden suceder	Casos que NO suceden
Casos buenos	$\delta_k = \frac{\text{Red. función}}{\text{Red. modelo}} = \frac{\text{Positivo pequeño}}{\text{Positivo grande}}$ <p>El caso más probable, indica que el modelo se optimiza mejor que la función.</p> $0 < \delta_k < 1$ <p>Son mejores los valores grandes.</p>	$\delta_k = \frac{\text{Red. función}}{\text{Red. modelo}} = \frac{\text{Positivo grande}}{\text{Positivo pequeño}}$ <p>En este caso la función se optimizaría mejor que el modelo.</p> $\delta_k > 1$
Casos malos	$\delta_k = \frac{\text{Red. función}}{\text{Red. modelo}} = \frac{\text{Negativo}}{\text{Positivo}}$ <p>En este caso, el modelo si se optimiza pero no la función, es decir, el modelo no se ajusta a la función real.</p> $\delta_k < 0$	$\delta_k = \frac{\text{Red. función}}{\text{Red. modelo}} = \frac{\text{Negativo}}{\text{Negativo}}$ <p>No se optimiza ni el modelo ni la función.</p> $\delta_k > 0, \text{ validar}$

Algoritmo

Dado un $\bar{\Delta} > 0$ (radio máximo), $\Delta_0 \in (0, \bar{\Delta})$, $\eta \in [0, 1/4]$

Entonces para cada $k=0,1,2,3,\dots$

1. Obtener P_k resolviendo

$$P_k^B = -B_k^{-1} \nabla f_k$$

$$P_k^U = -\frac{\nabla f_k}{\|\nabla f_k\|}$$

Si $\|P_k^B\| \leq \Delta_k$:

$$P_k = P_k^B$$

$P_{gradiente} = False$

Si no,

$$P_k = \Delta_k P_k^U$$

$P_{gradiente} = True$

2. Calcular δ_k

$$\delta_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}$$

3. Evaluar el modelo y modificar Δ_k

Si $\delta_k < 1/4$

$$\Delta_{k+1} = \eta \Delta_k$$

Sino, si $\delta_k > 3/4$ y $P_{gradiente} == True$:

$$\Delta_{k+1} = \min(2\Delta_k, \bar{\Delta})$$

Si no

$$\Delta_{k+1} = \Delta_k$$

4. Actualizar X_k sólo en caso de que el modelo funcione

Si $\delta_k > 1/4$

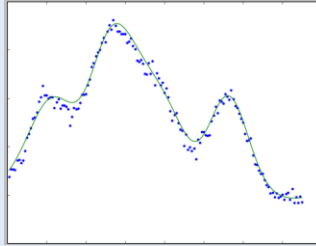
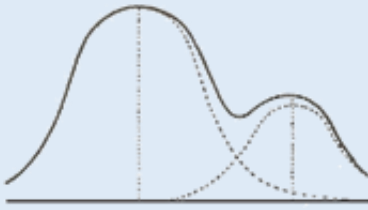
$$X_{k+1} = X_k + P_k$$

Si no

$$X_{k+1} = X_k$$

Actividad: Aproximación de función con Suma de Gaussianas

La suma de Gaussianas sirve para calcular funciones mediante (como su nombre lo dice) la suma de varias funciones Gaussianas.



La función de la suma de Gaussianas es:

$$f(x) = \sum_{k=1}^M B_k \exp\left(-\frac{1}{2\sigma^2}(x - C_k)^2\right)$$

Donde M es el número de Gaussianas
 B_k la altura de la Gaussiana k
 C_k representa el centro de la Gaussiana k

Dada una serie de puntos X,Y (obtenidos de simulaciones de señales) se deben encontrar las M Gaussianas que al sumarlas aproximen mejor la función. Para encontrar las Gaussianas óptimas se debe resolver el siguiente problema:

$$\min_{B_k, C_k} \sum_{x=1}^N \left[y - \sum_{k=1}^M B_k \exp\left(-\frac{1}{2\sigma^2}(x - C_k)^2\right) \right]^2$$

Donde N.- número de puntos M.- número de Gaussianas
 B_k .- la altura de la Gaussiana k C_k .- representa el centro de la Gaussiana k
 En este caso $\sigma = 10$ constante

Gradiente

$$\begin{bmatrix} \frac{d}{dB_k} = \sum_{x=1}^N [-2\exp(C_k)] [Fun] \\ \frac{d}{dC_k} = \sum_{x=1}^N \left[-\frac{2}{\sigma^2}(x - C_k) \right] [B_k \exp(C_k)] [Fun] \end{bmatrix}$$

Hessiano

$$\begin{bmatrix} \frac{d}{dB_k B_j} = \sum_{x=1}^N [2\exp(C_k)] [\exp(C_j)] \\ \frac{d}{dB_k C_k} = \frac{d}{dC_k B_k} = \sum_{x=1}^N \left[2B_k \frac{(x - C_k)}{\sigma^2} [\exp(C_k)]^2 - 2 \frac{(x - C_k)}{\sigma^2} [\exp(C_k)] [Fun] \right] \\ \frac{d}{dB_k C_j} = \frac{d}{dC_j B_k} = \sum_{x=1}^N [2\exp(C_k)] [\exp(C_j)] \\ \frac{d}{dC_k C_j} = \sum_{x=1}^N \left[\frac{2}{\sigma^4} (x - C_k)(x - C_j) \right] [B_j \exp(C_j)] [B_k \exp(C_k)] \\ \frac{d}{dC_k C_k} = \sum_{x=1}^N \left[\left[\frac{2}{\sigma^2} \right] [B_k \exp(C_k)] [Fun] - \left[\frac{2}{\sigma^4} (x - C_k)^2 \right] [B_k \exp(C_k)] [Fun] + \left[\frac{2}{\sigma^4} (x - C_k)^2 \right] [B_k \exp(C_k)]^2 \right] \end{bmatrix}$$

Donde:

$$\exp(C_i) = \exp\left(-\frac{1}{2\sigma^2}(x - C_i)^2\right) \quad Fu = [y - \sum_{k=1}^M B_k \exp(C_k)]$$

Parámetros:

$$\bar{\Delta} = \Delta_0 = 0.1$$

$$\eta = 0.05$$

Gradiente Conjugado

El Gradiente Conjugado es un método iterativo para minimizar funciones cuadráticas convexas que tienen la siguiente forma:

$$\min_x f(x) = \frac{1}{2}x^T A x - x^T b$$

donde $A \in R^{n \times n}$, $x, b \in R^n$. A es una matriz simétrica positiva definida.

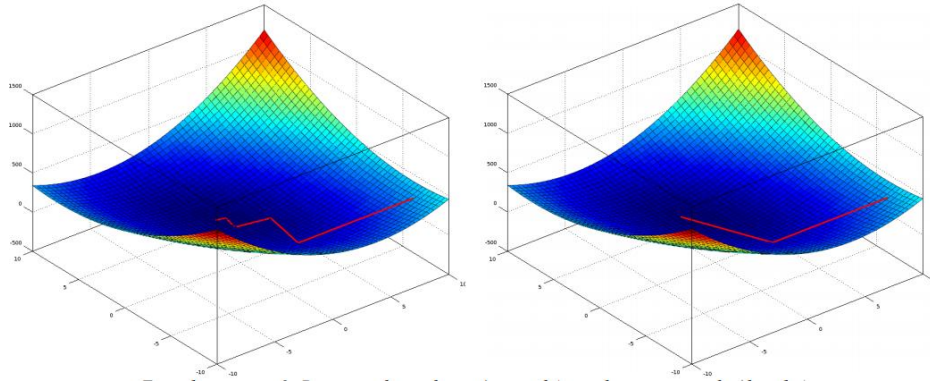


Figura 1 Descenso del Gradiente a la izquierda, Gradiente Conjugado a la derecha. (www.cimat.mx/~miguelvargas)

Como x es nuestra variable, podemos derivar e igualar a 0 para encontrar la solución:

$$\begin{aligned}\nabla f(x) &= Ax - b = 0 \\ Ax &= b\end{aligned}$$

Como se puede ver en la ecuación anterior, el método del Gradiente Conjugado se puede utilizar también como un método numérico para resolver sistemas de ecuaciones lineales cuando A es no invertible.

El Gradiente y el Hessiano se pueden calcular fácilmente:

$$\begin{aligned}\nabla f(x) &= Ax - b \\ \nabla^2 f(x) &= A\end{aligned}$$

Al igual que el método Descenso del Gradiente es un método iterativo de la siguiente forma:

$$X_{k+1} = X_k + \alpha_k P_k$$

en cada iteración se debe determinar:

- La dirección hacia donde se debe mover P_k (P_k debe ser un vector unitario)
- El tamaño de paso, el cual indica que tanto se debe mover α_k

Cálculo del tamaño de paso

$$\alpha_k = \min_{\alpha_k} f(x_k + \alpha_k P_k)$$

Si $f(x) = \frac{1}{2}x^T A x - x^T b$ entonces

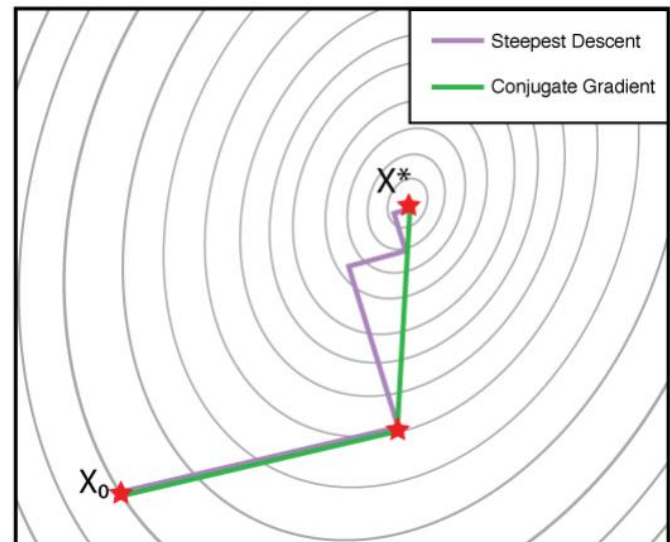
$$\begin{aligned}f(x_k + \alpha_k P_k) &= \frac{1}{2}(x_k + \alpha_k P_k)^T A (x_k + \alpha_k P_k) - (x_k + \alpha_k P_k)^T b \\ &= \frac{1}{2}x_k^T A x_k + \frac{1}{2}\alpha_k x_k^T A P_k + \frac{1}{2}\alpha_k P_k^T A x_k + \frac{1}{2}\alpha_k^2 P_k^T A P_k - x_k^T b - \alpha_k P_k^T b \\ &= \frac{1}{2}x_k^T A x_k + \alpha_k P_k^T A x_k + \frac{1}{2}\alpha_k^2 P_k^T A P_k - x_k^T b - \alpha_k P_k^T b\end{aligned}$$

Al derivar e igualar a 0

$$\begin{aligned}\frac{\partial f(x_k + \alpha_k P_k)}{\partial \alpha_k} &= P_k^T A x_k + \alpha_k P_k^T A P_k - P_k^T b = 0 \\ \alpha_k P_k^T A P_k &= P_k^T b - P_k^T A x_k \\ \alpha_k &= \frac{P_k^T b - P_k^T A x_k}{P_k^T A P_k} \\ \alpha_k &= \frac{-(P_k^T A x_k - P_k^T b)}{P_k^T A P_k} \\ \alpha_k &= \frac{-P_k^T (A x_k - b)}{P_k^T A P_k} \\ \alpha_k &= \frac{-P_k^T g_k}{P_k^T A P_k}\end{aligned}$$

Cálculo de la dirección

La idea es utilizar direcciones conjugadas como direcciones de descenso para **llegar a lo más en n pasos** al punto óptimo (n es la dimensión del vector de soluciones).



Se utilizan direcciones A conjugadas:

$$v_1^T A v_2 = 0$$

Entonces, si las direcciones son conjugadas:

$$P_k^T A P_{k+1} = 0$$

Luego, cada dirección puede ser una combinación lineal del gradiente y la dirección anterior:

$$P_{k+1} = -g_{k+1} + \beta P_k$$

Entonces:

$$\begin{aligned}P_k^T A P_{k+1} &= 0 \\ P_k^T A (-g_{k+1} + \beta P_k) &= 0 \\ -P_k^T A g_{k+1} + \beta P_k^T A P_k &= 0 \\ \beta P_k^T A P_k &= P_k^T A g_{k+1} \\ \beta &= \frac{P_k^T A g_{k+1}}{P_k^T A P_k}\end{aligned}$$

Algoritmo de Gradiente Conjugado (Versión inicial)

Recibe como parámetro: X_0 y G_0

$k = 0, P_0 = -G_0$

Mientras ($k < \text{MaxIteraciones} \ \&\& \|g_k\| > \varepsilon$)

$$\alpha_k = \frac{-P_k^T g_k}{P_k^T A P_k}$$

$$X_{k+1} = X_k + \alpha_k P_k$$

$$g_{k+1} = A X_{k+1} - b$$

$$\beta_{k+1} = \frac{P_k^T A g_{k+1}}{P_k^T A P_k}$$

$$P_{k+1} = -g_{k+1} + \beta_{k+1} P_k$$

$$k = k + 1$$

Fin mientras

Regresar el valor de X_k

Optimizando el cálculo del gradiente

$$g_{k+1} = A x_{k+1} - b$$

$$g_{k+1} = A(x_k + \alpha_k P_k) - b$$

$$g_{k+1} = A x_k + \alpha_k A P_k - b$$

$$g_{k+1} = g_k + \alpha_k A P_k$$

Optimizando el cálculo de β

$$g_{k+1} = g_k + \alpha_k A P_k \text{ despejando } A P_k = \frac{g_{k+1} - g_k}{\alpha_k}$$

Luego

$$\beta_{k+1} = \frac{P_k^T A g_{k+1}}{P_k^T A P_k} = \frac{g_{k+1}^T A P_k}{P_k^T A P_k} = \frac{g_{k+1}^T \left(\frac{g_{k+1} - g_k}{\alpha_k} \right)}{P_k^T \left(\frac{g_{k+1} - g_k}{\alpha_k} \right)}$$

$$\beta_{k+1} = \frac{g_{k+1}^T (g_{k+1} - g_k)}{P_k^T (g_{k+1} - g_k)} = \frac{g_{k+1}^T g_{k+1} - g_{k+1}^T g_k}{P_k^T g_{k+1} - P_k^T g_k}$$

Si las direcciones son conjugadas $P_{k+1}^T P_k = 0, P_k \cong -g_k$

$$\beta_{k+1} = \frac{g_{k+1}^T g_{k+1} - g_{k+1}^T g_k}{P_k^T g_{k+1} - P_k^T g_k}$$

$$\beta_{k+1} = -\frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}$$

Algoritmo de Gradiente Conjugado (Versión práctica)

Recibe como parámetro: $X_0, g_0 = \nabla f(X_0), A = \nabla^2 f(X_0)$

$k = 0, P_0 = -g_0$

Mientras ($k < \text{MaxIteraciones} \ \&\& \|g_k\| > \varepsilon$)

$$\alpha_k = \frac{-P_k^T g_k}{P_k^T A P_k}$$

$$X_{k+1} = X_k + \alpha_k P_k$$

$$g_{k+1} = g_k + \alpha_k A P_k$$

$$\beta_{k+1} = -\frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}$$

$$P_{k+1} = -g_{k+1} + \beta_{k+1} P_k$$

$$k = k + 1$$

Fin mientras

Regresar el valor de X_k

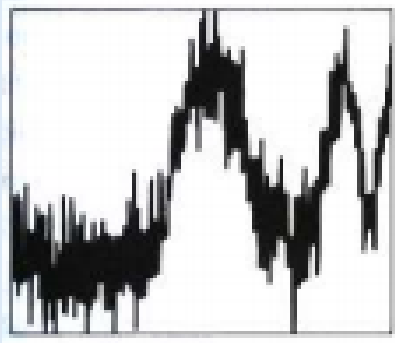
Actividad: Suavizado de señales utilizando Gradiente Conjugado

Si definimos:

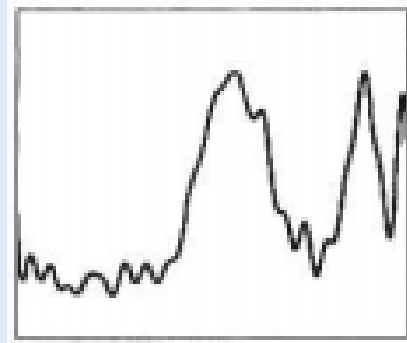
Y : es un vector con la señal de entrada con ruido

Y^c : es un vector con la señal de salida suavizada (debe ser calculada por el algoritmo)

Y : señal con ruido



Y^c : señal suavizada



La función a optimizar es:

$$\min_{Y^c \in \mathbb{R}} F(X) = \frac{1}{2} \left[\sum_{i=1}^n (Y_i^c - Y_i)^2 + \lambda \sum_{i=2}^n (Y_i^c - Y_{i-1}^c)^2 \right]$$

Se recomienda utilizar un valor de $\lambda = 4$. Se pueden utilizar las derivadas analíticas:

$$\nabla F(X) = \left[\frac{\delta F}{\delta Y_i^c} \right] = [Y_i^c - Y_i + \lambda(Y_i^c - Y_{i-1}^c)|_{i>1} + \lambda(Y_i^c - Y_{i+1}^c)|_{i<n}]$$

$$\nabla^2 F(X) = \begin{bmatrix} 1+\lambda & -\lambda & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\lambda & 1+2\lambda & -\lambda & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\lambda & 1+2\lambda & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\lambda & 1+2\lambda & -\lambda & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & -\lambda & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 1+2\lambda & -\lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\lambda & 1+2\lambda & -\lambda & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\lambda & 1+2\lambda & -\lambda \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\lambda & 1+\lambda \end{bmatrix}$$

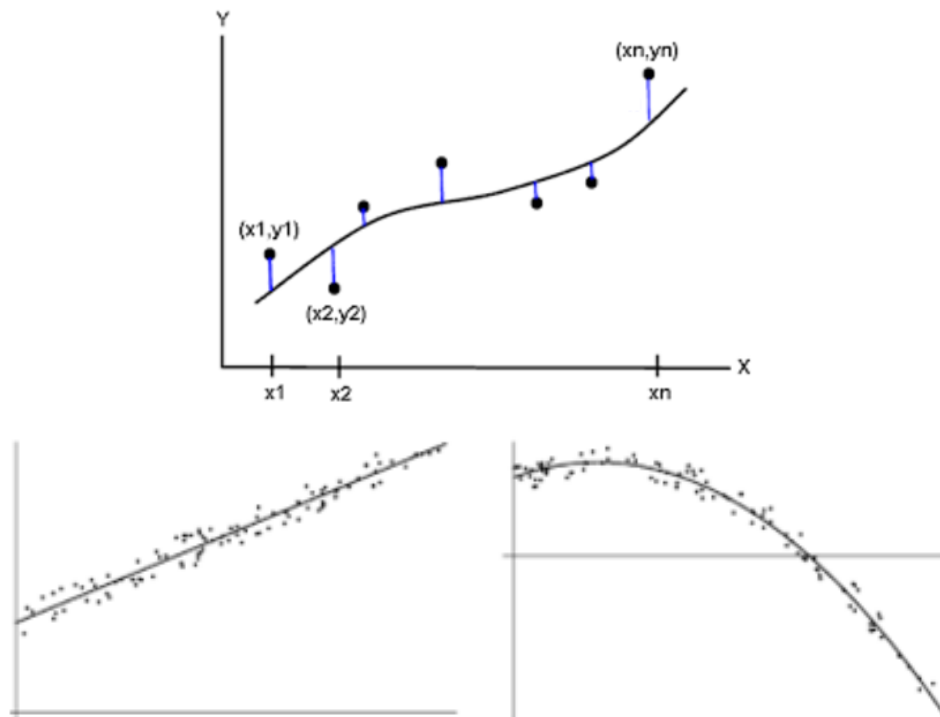
$$\nabla^2 F(X) * Z = \{ Z_i + \lambda(Z_i - Z_{i-1})|_{i>1} + \lambda(Z_i - Z_{i+1})|_{i<n} \}_{i=1,2,\dots}$$

Se deben calcular los valores para suavizar cuatro señales y entregar un reporte de resultados donde contenga:

- La gráfica de las señales originales
- Las gráficas de las señales suavizadas (utilizando Gradiente Conjugado)

Mínimos Cuadrados (Least squares)

En algunas ocasiones es necesario encontrar una función NO LINEAL que modele la respuesta de un fenómeno observado, por ejemplo:



Para resolver este problema primero se debe fijar un modelo, por ejemplo una parábola, un polinomio, etc., y después ajustar los N parámetros de dicho modelo utilizando optimización. Por lo tanto el problema a optimizar es:

$$\min_B \theta(B) = \frac{1}{2} \sum_{i=1}^M (f(X^i) - Y^i)^2, \quad r_i(B) = f(X^i) - Y^i$$

Sabemos que existen M residuos, uno por cada muestra, por lo tanto se define el vector de residuos $r(B)$ como:

$$r(B) = \begin{bmatrix} r_1(B) \\ r_2(B) \\ \vdots \\ r_M(B) \end{bmatrix}$$

Así como el vector Jacobiano $J(B)$, con las derivadas parciales de los residuos respecto a cada parámetro:

$$J(B) = \left[\frac{\partial r_j}{\partial B_i} \right]_{\substack{j=1,2,\dots,M \\ i=1,2,\dots,N}} = \begin{bmatrix} \nabla r_1(B)^T \\ \nabla r_2(B)^T \\ \vdots \\ \nabla r_M(B)^T \end{bmatrix}_{M \times N} = \begin{bmatrix} \frac{\partial r_1}{\partial B_1} & \frac{\partial r_1}{\partial B_2} & \cdots & \frac{\partial r_1}{\partial B_N} \\ \frac{\partial r_2}{\partial B_1} & \frac{\partial r_2}{\partial B_2} & \cdots & \frac{\partial r_2}{\partial B_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial r_M}{\partial B_1} & \frac{\partial r_M}{\partial B_2} & \cdots & \frac{\partial r_M}{\partial B_N} \end{bmatrix}_{M \times N}$$

Entonces, se pueden calcular el Gradiente y el Hessiano de la siguiente manera:

$$\min_B \theta(B) = \frac{1}{2} \sum_{i=1}^M (r_i(B))^2$$

- Gradiente

$$\nabla \theta = \sum_{i=1}^M \nabla r_i(B) r_i(B) = J(B)^T r(B) = \begin{bmatrix} \frac{\partial r_1}{\partial B_1} & \frac{\partial r_1}{\partial B_2} & \cdots & \frac{\partial r_1}{\partial B_N} \\ \frac{\partial r_2}{\partial B_1} & \frac{\partial r_2}{\partial B_2} & \cdots & \frac{\partial r_2}{\partial B_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial r_M}{\partial B_1} & \frac{\partial r_M}{\partial B_2} & \cdots & \frac{\partial r_M}{\partial B_N} \end{bmatrix}_{N \times M} \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_M \end{bmatrix}_{M \times 1}$$

- Hessiano

$$\nabla^2 \theta = \sum_{i=1}^M \nabla r_i(B) \nabla r_i(B) + \sum_{i=1}^M r_i(B) \nabla^2 r_i(B) = J(B)^T J(B) + \sum_{i=1}^M r_i(B) \nabla^2 r_i(B)$$

$$J(B)^T J(B) = \begin{bmatrix} \frac{\partial r_1}{\partial B_1} & \frac{\partial r_1}{\partial B_2} & \cdots & \frac{\partial r_1}{\partial B_N} \\ \frac{\partial r_2}{\partial B_1} & \frac{\partial r_2}{\partial B_2} & \cdots & \frac{\partial r_2}{\partial B_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial r_M}{\partial B_1} & \frac{\partial r_M}{\partial B_2} & \cdots & \frac{\partial r_M}{\partial B_N} \end{bmatrix}_{N \times M} \begin{bmatrix} \frac{\partial r_1}{\partial B_1} & \frac{\partial r_1}{\partial B_2} & \cdots & \frac{\partial r_1}{\partial B_N} \\ \frac{\partial r_2}{\partial B_1} & \frac{\partial r_2}{\partial B_2} & \cdots & \frac{\partial r_2}{\partial B_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial r_M}{\partial B_1} & \frac{\partial r_M}{\partial B_2} & \cdots & \frac{\partial r_M}{\partial B_N} \end{bmatrix}_{M \times N}$$

$$\sum_{i=1}^M r_i(B) \nabla^2 r_i(B) = \sum_{i=1}^M r_i(B) \begin{bmatrix} \frac{\partial^2 r_i}{\partial B_1^2} & \frac{\partial^2 r_i}{\partial B_1 \partial B_2} & \cdots & \frac{\partial^2 r_i}{\partial B_1 \partial B_N} \\ \frac{\partial^2 r_i}{\partial B_1 \partial B_2} & \frac{\partial^2 r_i}{\partial B_2^2} & \cdots & \frac{\partial^2 r_i}{\partial B_2 \partial B_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 r_i}{\partial B_N \partial B_1} & \frac{\partial^2 r_i}{\partial B_N \partial B_2} & \cdots & \frac{\partial^2 r_i}{\partial B_N^2} \end{bmatrix}_{N \times N}$$

Método de Gauss – Newton para resolver Mínimos Cuadrados NO Lineales

En Gauss – Newton se calcula el Hessiano como una aproximación utilizando sólo:

$$\nabla^2 \theta \approx J(B)^T J(B)$$

En base al ahorro computacional de calcular los hessianos de cada uno de los residuos: $\nabla^2 r_i(B)$ en el segundo término: $\nabla^2 \theta = J(B)^T J(B) + \sum_{i=1}^M r_i(B) \nabla^2 r_i(B)$, además de que el término de primer orden generalmente domina al término de segundo orden.

Algoritmo General de Mínimos Cuadrados

Recibe como parámetro: B_0 ,

$k = 0$

Mientras $(k < \text{MaxIteraciones} \ \&\& \ ||g_k|| > \varepsilon)$

Calcular el Hessiano H_k

$$H_k = J(B)^T J(B) \quad \text{ó} \quad H_k = J(B)^T J(B) + \sum_{i=1}^M r_i(B) \nabla^2 r_i(B)$$

Calcular la dirección P_k resolviendo el siguiente sistema de ecuaciones

$$H_k P_k = -G_k$$

$$P_k = \frac{P_k}{||P_k||}$$

Calcular α_k con alguno de los métodos propuestos en Descenso del Gradiente

$$X_{k+1} = X_k + \alpha_k P_k$$

$k = k + 1$

Fin mientras

Regresar el valor de X_k

Actividad: Regresión NO LINEAL

Se tiene la siguiente información

Hydrogen	N-Pentane	Isopentane	Reaction Rate
470	300	10	8.55
285	80	10	3.79
470	300	120	4.82
470	80	120	0.02
470	80	10	2.75
100	190	10	14.39
100	80	65	2.54
470	190	65	4.35
100	300	54	13.00
100	300	120	8.50
100	80	120	0.05
285	300	10	11.32
285	190	120	3.12

Y se cree que se puede crear una función como la siguiente para modelar la Taza de Reacción de acuerdo a los parámetros Hydrogen, N-Pentane, Isopentane:

$$R = \frac{B_1 \text{Hydrogen} - \text{Isopentane} / B_5}{1 + B_2 \text{Hydrogen} + B_3 \text{NPentane} + B_4 \text{Isopentane}}$$

Programación Lineal

La Programación lineal es una técnica para resolver problemas de optimización sobre una función lineal con restricciones lineales.

En la programación lineal hay tres elementos clave:

- **Función Objetivo.** Que es el modelo del problema a resolver, por ejemplo: ¿cómo se calcula la utilidad (maximización)? ¿cómo se calculan los costos (minimización)?
- **Variables.** Representan los valores que pueden modificarse, y a partir de los cuales podemos encontrar una solución óptima.
- **Restricciones.** Es todo aquello que limita la libertad de los valores, por ejemplo: ¿Cuántos empleados como máximo puedo tener? ¿Cuánta materia prima tengo?

Modelado

La fábrica de Hilados y Tejidos "X" requiere fabricar dos tejidos de calidad diferente M y N; se dispone de 500 Kg de hilo a, 300 Kg de hilo b y 108 Kg de hilo c. Para obtener un metro de M diariamente se necesitan 125 gr de a, 150 gr de b y 72 gr de c; para producir un metro de N por día se necesitan 200 gr de a, 100 gr de b y 27 gr de c. El M se vende a \$4000 el metro y el N se vende a \$5000 el metro. Si se debe obtener el máximo beneficio, ¿cuántos metros de M y N se deben fabricar?

Variables:

X_m -> metros de M que se deben fabricar

X_n -> metros de N que se deben fabricar

Restricciones:

Límite de a, $125X_m + 200X_n \leq 500,000$

Límite de b, $150X_m + 100X_n \leq 300,000$

Límite de c, $72X_m + 27X_n \leq 108,000$

No puedo hacer metros negativos, $X_n, X_m > 0$

Función objetivo:

¿Minimizar o maximizar? ¿Cómo obtengo la ganancia?

Maximizar $G = 4000X_m + 5000X_n$

Por lo tanto, el problema es:

Maximizar $G = 4000X_m + 5000X_n$

Sujeto a

$125X_m + 200X_n \leq 500,000$

$150X_m + 100X_n \leq 300,000$

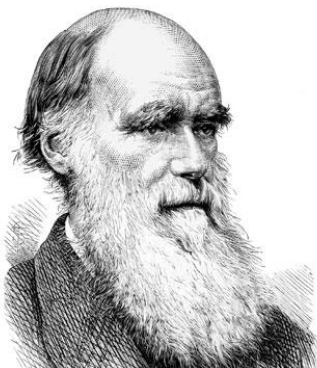
$72X_m + 27X_n \leq 108,000$

Actividad: Modelado de problemas de programación lineal

1. Un herrero con 80 Kg. de acero y 120 Kg. de aluminio quiere hacer bicicletas de paseo y de montaña que quiere vender, respectivamente a 20.000 y 15.000 pesos cada una para sacar el máximo beneficio. Para la de paseo empleará 1 kg de acero y 3 kg de aluminio, y para la de montaña 2 kg de ambos metales. ¿Cuántas bicicletas de paseo y de montaña deberá fabricar para maximizar las utilidades?
2. Una compañía fabrica y venden dos modelos de lámpara L1 y L2. Para su fabricación se necesita un trabajo manual de 20 minutos para el modelo L1 y de 30 minutos para el L2; y un trabajo de máquina de 10 minutos para L1 y de 10 minutos para L2. Se dispone para el trabajo manual de 100 horas al mes y para la máquina 80 horas al mes. Sabiendo que el beneficio por unidad es de 15 y 10 euros para L1 y L2, respectivamente, planificar la producción para obtener el máximo beneficio.
3. En una granja de pollos se da una dieta, para engordar, con una composición mínima de 15 unidades de una sustancia A y otras 15 de una sustancia B. En el mercado sólo se encuentra dos clases de compuestos: el tipo X con una composición de una unidad de A y 5 de B, y el otro tipo, Y, con una composición de cinco unidades de A y una de B. El precio del tipo X es de 10 euros y del tipo Y es de 30 €. ¿Qué cantidades se han de comprar de cada tipo para cubrir las necesidades con un coste mínimo?
4. Con el comienzo del curso se va a lanzar unas ofertas de material escolar. Unos almacenes quieren ofrecer 600 cuadernos, 500 carpetas y 400 bolígrafos para la oferta, empaquetándolo de dos formas distintas; en el primer bloque pondrá 2 cuadernos, 1 carpeta y 2 bolígrafos; en el segundo, pondrán 3 cuadernos, 1 carpeta y 1 bolígrafo. Los precios de cada paquete serán 6.5 y 7 €, respectivamente. ¿Cuántos paquetes le conviene poner de cada tipo para obtener el máximo beneficio?
5. Una escuela prepara una excursión para 400 alumnos. La empresa de transporte tiene 8 autobuses de 40 plazas y 10 de 50 plazas, pero sólo dispone de 9 conductores. El alquiler de un autocar grande cuesta 800 € y el de uno pequeño 600 €. Calcular cuántos autobuses de cada tipo hay que utilizar para que la excursión resulte lo más económica posible para la escuela.

Evolutionary Computing

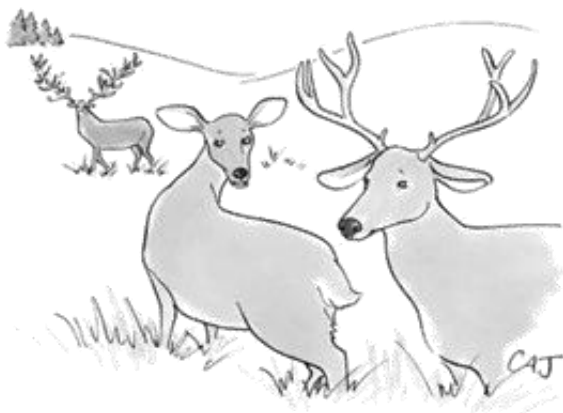
(Book: Eiben. Introduction to Evolutionary Computing)



Evolutionary Computing is a research area within computer science. As the name suggests, it is inspired by Darwin's theory of natural evolution.

In an environment with limited resources, the fittest individuals survive. Also, they have more chances to have offspring.

Natural selection

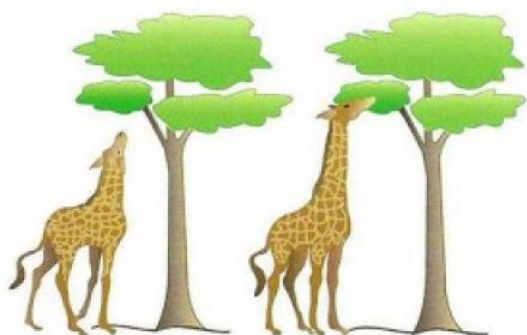


https://www.allposters.com/-sp/It-s-not-you-it-s-natural-selection-New-Yorker-Cartoon-Posters_i9184757_.html

Reproduction

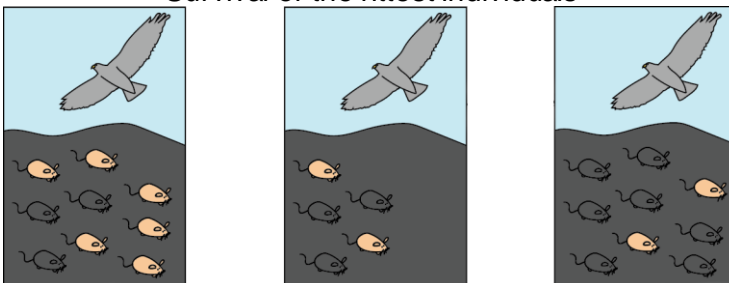


Mutation



<http://www.mercados.lat/index.php/otros/item/1052-seleccion-natural>

Survival of the fittest individuals



<https://es.khanacademy.org/science/biology/her/evolution-and-natural-selection/a/darwin-evolution-natural-selection>

Evolution		Problem solving
Environment	↔	Problem
Individual	↔	Solution
Fitness	↔	Quality

The main idea of all evolutive algorithms is the same: first, a **population** of **individuals** is randomly generated, and the **fitness** of all of them is calculated. Iteratively, where each iteration represents a generation, we do the following steps. The fittest individuals are **selected** as parents of the next generation, aiming to improve the average of the population's fitness. We use **crossover** for creating new individuals (offspring) based on parents. Some of the new individuals are **mutated** to explore new solutions. This process is executed until an individual is good enough or the computational limits have been reached.

```

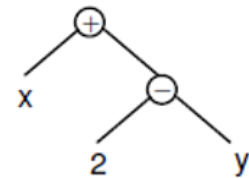
BEGIN
  INITIALISE population with random candidate solutions;
  EVALUATE each candidate;
  REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
    1 SELECT parents;
    2 RECOMBINE pairs of parents;
    3 MUTATE the resulting offspring;
    4 EVALUATE new candidates;
    5 SELECT individuals for the next generation;
  OD
END

```

Individuals' representation: An individual represents a solution. The representation of the individual in the original problem is called **phenotype**. The representation used for solving the problem is called **genotype**. In other words, an individual's genotype encodes its phenotype. For example, the phenotype of the variable age is 4, but the genotype is 000100. Another example is:

Phenotype:
 $F(x,y) = x + (2-y)$

Genotype:



Fitness: It is a numeric value that represents the quality of the solution.

Population: It is a group of individuals (solutions) that recombine and mutate their properties. The initial population is randomly created.

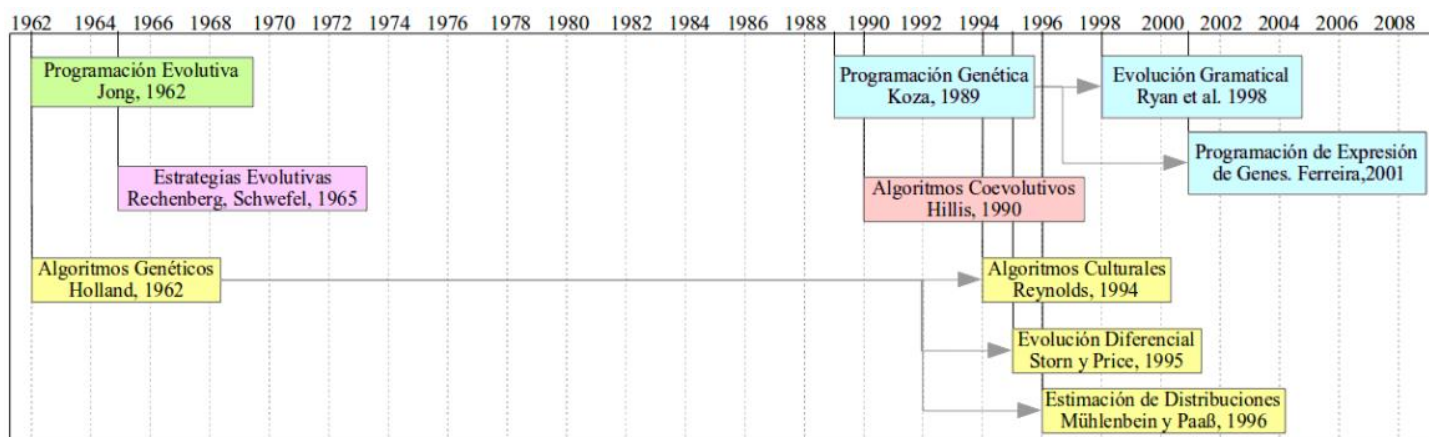
Selection of parents: For creating a new generation of individuals (offspring), the parents need to be selected based on their fitness. The main idea is to inherit the characteristics of the fittest individuals.

Crossover (reproduction): The parents inherit their characteristics to their offspring.

Mutation: Individuals modify their characteristics or behavior to improve themselves.

Survival: The fittest individuals survive and can live more time.

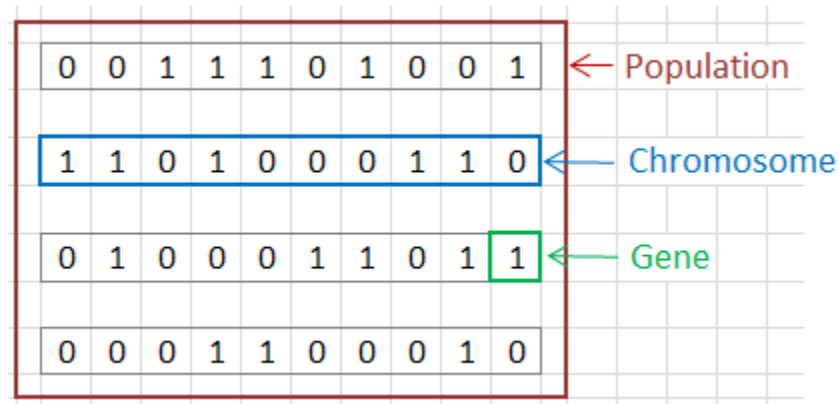
Termination condition: If you know the value of good fitness, the algorithm can stop when you find an individual with good fitness. However, generally, we do not know the values. In those cases, we can stop the algorithm when: the number of maximum fitness evaluations have been reached, after a defined number of iterations, or when the fitness of the best individual has not been changed after a defined number of generations.



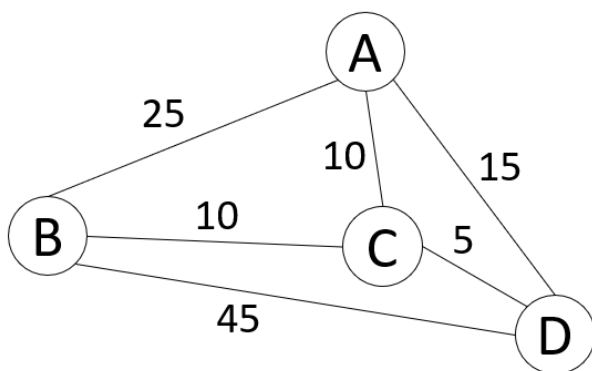
Genetic Algorithms (GA)

John Holland proposed genetic Algorithms in the 1970s. Initially, they were called “Reproductive Plans.” These algorithms are maybe the most famous of the evolutive algorithms family.

The inspiration comes from the DNA structure, which is people's genetic code. All the information is stored in chromosomes that have a lot of genes. Holland's proposal consists of representing the solutions by binary arrays.



Example 1: Traveling salesman problem

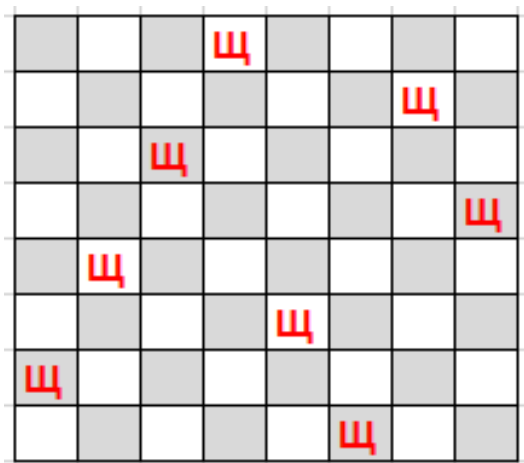


The problem consists of determining the path a salesperson must follow to minimize the distance. He/she starts and ends in the same node, and nodes cannot be repeated.

Example:

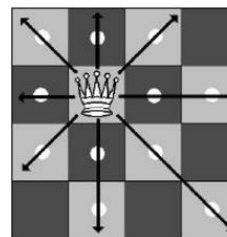
$$A - B - D - C - A = 25 + 45 + 5 + 10 = 85$$

Example 2: Traveling salesman problem



The problem consists of placing 8 queens on an 8x8 chessboard. The queens must not attack among them.

A queen attacks all the pieces that are in the same row, column, or diagonal.



wikipedia.org

Individuals' representation

The individuals' representation can be divided into genotype and phenotype:

- **Genotype:** it is the codified version of the solution
- **Phenotype:** it is the solution that represents an individual

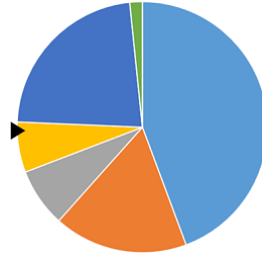
We can use some of the following representations:

- **Binary representation:** It is the Holland's original proposal. It consists of representing the solution using a binary array. It could be beneficial because a lot of problems can be represented using this technique.
- **Integer representation:** The representation of individuals as integer arrays is the best option for specific problems. For example: for evolving the trajectory of an agent using numbers for directions (left, right, up, and down).
- **Real representation:** Many problems can be represented as real arrays, it is, $[x_1, x_2, \dots, x_n]$ where $x_i \in \mathbb{R}$.
- **Permutation representation:** Some problems, such as sudoku or traveling agent, can be represented as a permutation of a set.

Selection of parents

There are some techniques for parents' selection:

Roulette selection, also called proportional selection, was the original proposal of Holland. The idea is simple. You can imagine a roulette where each section is assigned to an individual. If we have 10 individuals, the roulette is divided into 10 sections. The section size is proportioned to the individual's fitness.



The roulette's implementation can be as follows. Given an individual i , f_i represents its fitness, and, p_i is the individual's proportion, where it can be calculated as $p_i = \frac{f_i}{\sum_k f_k}$. The sum of all proportions must be 1. The segment between 0 and 1 is divided using those proportions for calculating ranges for each individual. Finally, to select an individual, a number is randomly calculated between 0 and 1, and the individual whose range corresponds to that number is selected as a parent.

Individual	Fitness f_i	Proportion p_i	Range
A	2	0.04	[0.00 – 0.04]
B	10	0.20	[0.04 – 0.24]
C	7	0.14	[0.24 – 0.38]
D	1	0.02	[0.38 – 0.40]
E	30	0.60	[0.40 – 1.00]

The roulette has some problems:

- If an individual has a fitness significantly bigger than others, the algorithm selects the same individual several times.
- If the individuals' fitness is similar among them, there is no pressure in the selection. In other words, the individuals will be selected as entirely random.

Tournament selection, may be the most known technique and easy to implement for parent selection. It consists of randomly choosing k individuals and selecting the fittest one. k represents the tournament size.

Advantages of the tournament:

- It is too easy to implement.
- To change the selection's pressure, we can change the value of k . The bigger the value, the more the pressure.

Real-valued representation

- **Discrete reproduction:** It is the same idea of a uniform crossover. For each element of the children, we randomly copy the value of parent 1 or parent 2.
- **Asymmetric reproduction:** This technique consists of calculating the offspring o as the average of their parents p_1 and p_2 . Formally, $o = \alpha p_1 + (1 - \alpha p_2)$, where α is a number between 0 and 1, generally, its value is 0.5.

Permutation representation

- **Simple permutation crossover:** the steps are described as follows:
 Step 1: Divide the individuals into two sections and copy the elements.
 Step 2: Calculate the repeated and the missing elements.
 Step 3: Randomly assign the missing values in the positions of the repeated elements.

Step 1										Step 2										Step 3									
Parent 1	1	2	3	4	5	6	7	8	9	Parent 1	1	2	3	4	5	6	7	8	9	Parent 1	1	2	3	4	5	6	7	8	9
Parent 2	9	3	7	8	2	6	5	1	4	Parent 2	9	3	7	8	2	6	5	1	4	Parent 2	9	3	7	8	2	6	5	1	4
Offspring	1	2	3	4	2	6	5	1	4	Offspring	1	2	3	4	2	6	5	1	4	Offspring	1	2	3	4	8	6	5	9	7
Missing										Missing	7	8	9																

- **Partially mapped crossover:** the steps are described as follows:
 Step 1: Divide the individuals into 3 sections and copy the elements of the intermediate section of the first parent.
 Step 2: Copy the elements of the second parent (first and third sections) but ignore the elements that appear in the offspring.
 Step 3: Assign the missing values using the elements of the second part of the second parent that do not appear in the offspring.

Step 1										Step 2										Step 3									
Parent 1	1	2	3	4	5	6	7	8	9	Parent 1	1	2	3	4	5	6	7	8	9	Parent 1	1	2	3	4	5	6	7	8	9
Parent 2	9	3	7	8	2	6	5	1	4	Parent 2	9	3	7	8	2	6	5	1	4	Parent 2	9	3	7	8	2	6	5	1	4
Offspring				4	5	6	7			Offspring	9	3		4	5	6	7		1	Offspring	9	3	8	4	5	6	7	1	2

Mutation

Mutation's goal is to modify individuals to explore the search space. Some of the techniques are the following:

- **Bitwise mutation** is used in binary representation and consists of randomly selecting one or several genes and changing their values.

0	1	1	1	0	1	0	0	1
			↓					
0	1	1	0	0	1	0	0	1

- **Random resetting** is used in integer presentation and consists of randomly selecting one or several genes and resets its values.
- **Uniform mutation** is used in real-valued representation. It randomly selects one or several genes and chooses a random value between the minimum and maximum values.
- **Swap mutation** is used in permutation representation and consists of randomly selecting two elements and swapping their values.

Holland's original proposal of Genetic Algorithms was:

- Representation: binary
- Parents' selection: roulette
- Crossover: 1 point
- Mutation: bitwise
- Population model: generational

Implementation of the Genetic Algorithm phases

Crossover

Parameters:

- Population of N parents
- Pr : probability of reproduction

Return: new population of N individuals

Begin:

For each individual in the new population:

 If a random number between 0 and 1 is less than Pr :

 Select two parents (roulette or tournament) and combine them

 Add the offspring to the new population

 Else

 Select one parent (roulette or tournament) and clone it

 Add the clone to the new population

 End if

Return the new population

End

Mutation

Parameters:

- Population of N parents
- Pm : probability of mutation

Return: new population of N individuals

Begin:

For each individual in the population:

 If a random number between 0 and 1 is less than Pm :

 Mutate the individual

 Add the mutated individual to the new population

 Else

 Clone the individual and put it in the new population

 End if

Return the new population

End

Elite individual

If in the new population, we find an individual better than the elite:

 Replace the elite with the fittest individual

Else:

 Select an individual from the population with negative tournament

 Replace the selected individual with the elite

Genetic Algorithm

Parameters:

N, population size
 G, Maximum number of generations
 Pr, Reproduction's probability
 Pm, Mutation's probability

Return: the elite individual

Begin

Create the initial population
 Calculate the population fitness
 Get the elite
 While the number of generations is less than G or we haven't found a good solution
 Apply crossover selecting the parents
 Apply mutation
 Calculate the population fitness
 Get the elite or include the elite in the population

End while

End

Recommended values:

N = 30, G = 100, Pr = 0.8, Pm = 0.3

Population models

We can distinguish between two evolution models:

- **Steady-state model:** In each iteration, a few number of individuals (mostly only one) are created using crossover and mutation, and those individuals replace other individuals from the population. For selecting the individual that will be replaced, it can be used a negative tournament, it is, randomly choosing several individuals and remove the one with worse fitness.
- **Generational model:** Mostly used in Genetic Algorithms. It randomly creates an initial population of N individuals. Each generation, the whole population is modified applying the following process. First, the parents are selected based on their fitness. Those parents are using for creating the offspring using the crossover and mutation algorithms. To warranty the algorithm convergence, we need to store the elite individual, the fittest individual. If in the new generation we found an individual better than the elite, we replace the elite. If not, we include the elite in the generation.

Evolution Strategies (ES)

Evolution Strategies were proposed by Rechenberg y Schwefel (Technical University of Berlin) in 1960's. The goal is to solve continuous multidimensional optimization problems. The main characteristic is the **self-adaptation** of parameters. It means that some evolutive algorithm parameters change during the execution. Those parameters are included in the individual representation and evolve at the same time that the solution.

Individuals' representation

Given the goal is solving continuous multidimensional optimization problems, the individuals' solutions are represented as vectors whose inputs are the values of the variables, the individual i 's solution is represented as the vector $\vec{x}_i \in \mathbb{R}^d$, where d represents the number of features. One of the main characteristics of Evolution Strategies is self-adaptation of parameters, where each individual contains the mutation parameters σ_i , in addition to the values of the variables that are stored in the vector \vec{x}_i . The individual's representation is as follows:

$$\langle \vec{x}_i, \sigma_i \rangle$$

Mutation

The individuals can be seen as points in a d –multidimensional space, where the mutation's goal is to move those points so that the position of the mutated individual is close to the position of the individual before mutation.

The individual's position \vec{x}_i is modified by adding a random number, noise, to each entry. The noise follows a normal distribution zero-centered and with standard deviation σ_i . The value σ_i is known as *mutation step size*, where the bigger the value, the bigger the individual modification.

Rechenberg proposed the famous **1/5 success rule**, it states that the ratio of successful mutations (those in which the child is fitter than the parent) to all mutations should be 1/5. Hence if the ratio is greater than 1/5 the step size should be increased to make a wider search of the space, and if the ratio is less than 1/5 then it should be decreased to concentrate the search more around the current solution. The rule is executed at periodic intervals, for instance, after k iterations, each σ is reset by

$$\sigma = \begin{cases} \sigma/c & \text{si } p > 1/5 & \text{Increase} \\ \sigma * c & \text{si } p < 1/5 & \text{Decrease} \\ \sigma & \text{si } p = 1/5 \end{cases}$$

Where p is the relative frequency of successful mutations measured over a number of trials, and the parameter c is in the range $0.817 \leq c \leq 1$.

We describe two special cases of mutation in evolution strategies:

- **Uncorrelated Mutation with One Step Size.** In this case, the individual is represented by its position \vec{x} and one scalar that represents the mutation step size σ .

$$\underbrace{\langle x_1, x_2, \dots, x_d \rangle}_{\vec{x}} \quad \underbrace{\sigma}_{\sigma}$$

In the mutation, we add to all the vector entries a random number obtained from a normal distribution zero-centered with standard deviation equals to standard σ . The mutation step size must also be modified for self-adaptation. The mutation is performed as follows:

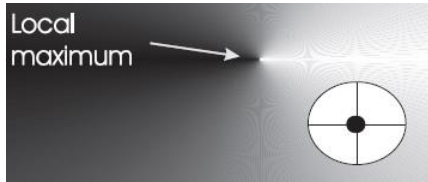
$$\begin{aligned}\sigma' &= \sigma * e^{N(0, \tau)} \\ x'_i &= x_i + N(0, \sigma)\end{aligned}$$

the recommended value for τ is $\tau = 1/\sqrt{n}$. For avoiding step sizes equals to 0, it is recommended

$$\text{If } \sigma' < \epsilon \Rightarrow \sigma' = \epsilon$$

where ϵ is an small scalar, it is recommended $\epsilon = 1e - 3$.

The mutation can be seen as modification in a hypersphere, where the modifications near to the original position have more probability than the ones far from the original position. This mutation is recommended for problems whose features have the same values range.



En este caso, la mutación de un individuo se puede ver como una nueva posición en una hiperesfera cuyo centro es la posición actual del individuo, además de que posiciones cercanas al individuo son más probables (por la distribución Gaussiana).

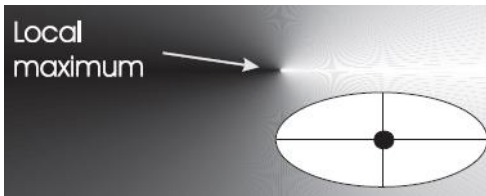
- **Uncorrelated Mutation with d Step Sizes.** The main idea is to treat each feature independently. It is useful where the features have different ranges values. In this case, the mutation parameter is a vector $\vec{\sigma} \in \mathbb{R}^d$. The individuals' representation is:

$$\underbrace{\langle x_1, x_2, \dots, x_d \rangle}_{\vec{x}} \quad \underbrace{\langle \sigma_1, \sigma_2, \dots, \sigma_d \rangle}_{\vec{\sigma}}$$

The mutation is performed as follows:

$$\begin{aligned}\sigma'_i &= \sigma_i * e^{N(0, \tau_1) + N(0, \tau_2)} \\ x'_i &= x_i + N(0, \sigma'_i)\end{aligned}$$

where $\tau_1 = 1/\sqrt{2n}$ y $\tau_2 = 1/\sqrt{2\sqrt{n}}$. It is important to avoid values of σ_i near to 0.



Recombination

The recombination consists of create one child from two parents. In Evolution Strategies there are two recombination variants. In *intermediate recombination* the values of the parents are averaged. Using *discrete recombination* one of the parent's values is randomly chosen with equal chance for either parents. The parents can be represented as the vectors \vec{p}_1 and \vec{p}_2 , and the child is the resultant vector \vec{c} . Using this notation, the recombination techniques are defined as:

$$c_i = \begin{cases} (p_{1i} + p_{2i})/2 & \text{intermediate recombination} \\ \text{random selection: } p_{1i} \text{ or } p_{2i} & \text{discrete recombination} \end{cases}$$

Selection of Parents

Parent selection in ES is completely random, it is because here the whole population is seen as parent.

Survivor Selection

After creating λ offspring and calculating their fitness, the best μ of them are chosen deterministically. There are two schemes of survivor selection:

- (μ, λ) selection, where only the best μ offspring are selected
- $(\mu + \lambda)$ selection, where the best μ individuals (from the union of parents and offspring) are selected

To sum up, the book Introduction to Evolutionary Algorithms (Eiben) presents the following summary of ES:

- Representation: real-valued vectors
- Recombination: discrete or intermediary
- Mutation: Gaussian perturbation
- Parent selection: Uniform random
- Survivor selection: (μ, λ) or $(\mu + \lambda)$
- Specialty: self-adaptation of mutation step sizes

Evolution Strategies Algorithm

Parameters:

- N, population size
- λ , offspring size
- G, Maximum number of generations

Return: the elite individual

Begin

Create the initial population

Calculate the population fitness

Get the elite

While the number of generations is less than G or we haven't found a good solution

Recombination, generate λ offspring

Mutation of parents and offspring

Calculate the population fitness

Survivor selection (the best individuals)

Get the elite or include the elite in the population

End while

End

Evolutionary Programming (EP)

Evolutionary Programming (EP) was proposed by Lawrence Fogel in 60's.

In the classical example of EP, predictors were evolved in the form of finite state machines. A finite state machine (FSM) is a transducer that can be stimulated by a finite alphabet of input symbols and can respond in a finite alphabet of output symbols. It consists of a number of states S and a number of state transitions. The state transitions define the working of the FSM: depending on the current state and the current input symbol, they define an output symbol and the next state to go to.

The idea was evolving finite state machines (FSMs). There are five generally usable mutation operators to generate new FSMs:

- Changing an output symbol
- Changing a state transition
- Adding a state
- Deleting a state
- Changing the initial state

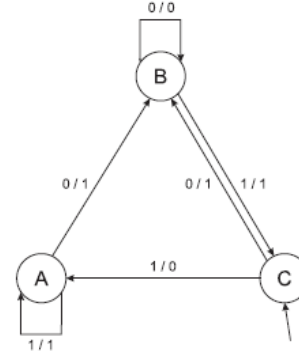


Figura 2 An example of finite state machine consisting of three states: A, B, and C. The input alphabet is $I=\{0,1\}$, and the output alphabet is $O=\{a,b,c\}$.

Since 90's, some variants of Evolutionary Programming were proposed, they used real vectors for solving continuous multidimensional optimization problems. In fact, they are the new standard for Evolutionary Programming.

In Evolutionary Programming, **the solutions represent species** instead of individuals.

Representation

The representation is the same of Evolution Strategies using uncorrelated mutation with d step sizes. The solutions are represented as vectors whose inputs are the values of the variables, the i -th solution is represented as the vector $\vec{x}_i \in \mathbb{R}^d$, where d represents the number of features. In addition, the solution contains the mutation parameters $\vec{\sigma}_i \in \mathbb{R}^d$.

$$\underbrace{\langle x_1, x_2, \dots, x_d \rangle}_{\vec{x}} \quad \underbrace{\langle \sigma_1, \sigma_2, \dots, \sigma_d \rangle}_{\vec{\sigma}_i}$$

Mutation

The species can be seen as points in a d – multidimensional space, where the mutation's goal is to move those points so that the position of the mutated individual is close to the position of the individual before mutation. The specie's position \vec{x}_i is modified adding an random number to each component following the next equations:

$$\begin{aligned} \sigma'_i &= \sigma_i * (1 + N(0, \alpha)) \\ x'_i &= x_i + N(0, \sigma'_i) \end{aligned}$$

where $\alpha \approx 0.2$. It is recommended avoiding values of σ_i close to 0. If $\sigma' < \epsilon \Rightarrow \sigma' = \epsilon$.

Crossover

It does not apply. It is because the solutions are seen as species.

Parent selection

It does not apply. Each specie (or solution) generates a new one using the mutation operator.

Survivor selection

The survivor selection used in Evolution Programming is known as $(\mu + \mu)$. Each specie generates a new one by mutation. Then, from the set of all the species (the original and the new ones), the μ best are deterministically selected as be part of the new generation.

To sum up, the book Introduction to Evolutionary Algorithms (Eiben) presents the following summary of EP:

- Representation: real-valued vectors
- Parent selection: Deterministic (each parent creates one offspring via mutation)
- Recombination: None
- Mutation: Gaussian perturbation
- Survivor selection: $(\mu + \mu)$.
- Specialty: self-adaptation of mutation step sizes

Evolutionary Programming Algorithm

Parameters:

μ , population size
G, Maximum number of generations

Return: the elite individual

Begin

Create the initial population

Calculate the population fitness

Get the elite

While the number of generations is less than G or we haven't found a good solution

 Mutation of all the species

 Calculate the population fitness

 Survivor selection

 Get the elite or include the elite in the population

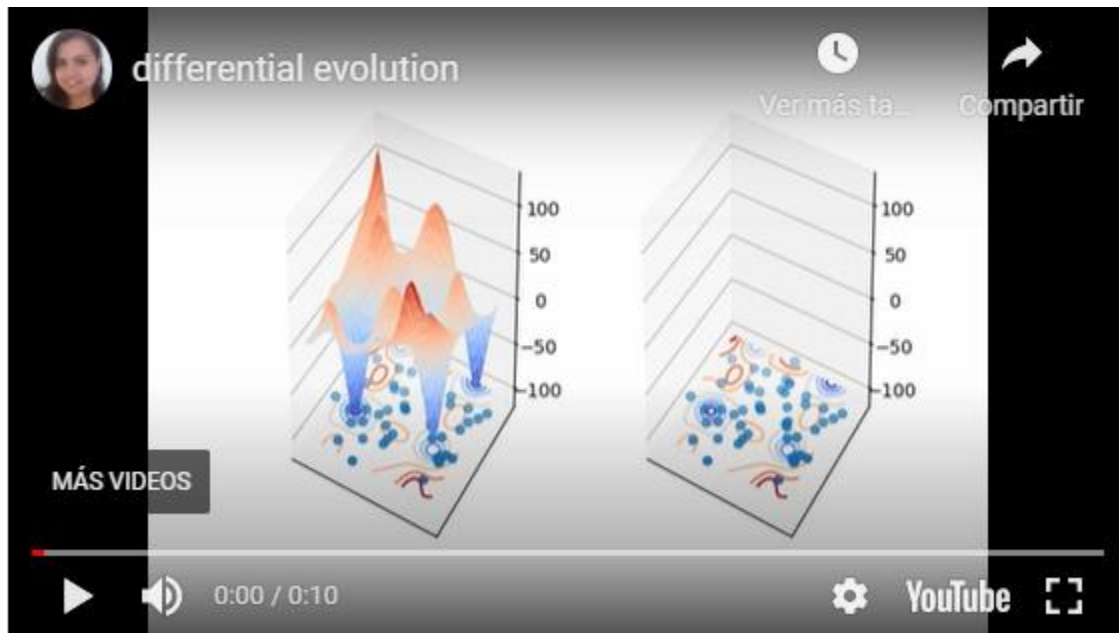
End while

End

Diferencial Evolution (DE)

Differential Evolution was proposed by Storn and Price in 1995. It is a robust algorithm for solving continuous multidimensional optimization problems. In this algorithm, individuals are seen as vectors. The novelty is the mutation operator, that uses three individuals to mutate another one, and the mutation depends on the distance

It is possible to see that Evolution Differential's individuals converge to the local minimums, and eventually, all converge to the global minimum. In the next video you can see the main idea:



<https://youtu.be/BsfJDg0a0Z4>

Representation

The individuals are represented as vectors whose entries are the variables values. In this case, the i -th individual is represented with the vector $x^i \in \mathbb{R}^d$.

$$x^i = \langle x_1^i, x_2^i, \dots, x_d^i \rangle$$

The first generation of individuals must be initialized using a range of values. Each variable has a minimum x_k^{min} and a maximum value x_k^{max} . Formally, each variable k of i -th individual can be calculated as follows:

$$x_k^i = x_k^{min} + r(x_k^{max} - x_k^{min})$$

Where r is a random number from 0 to 1.

Mutation

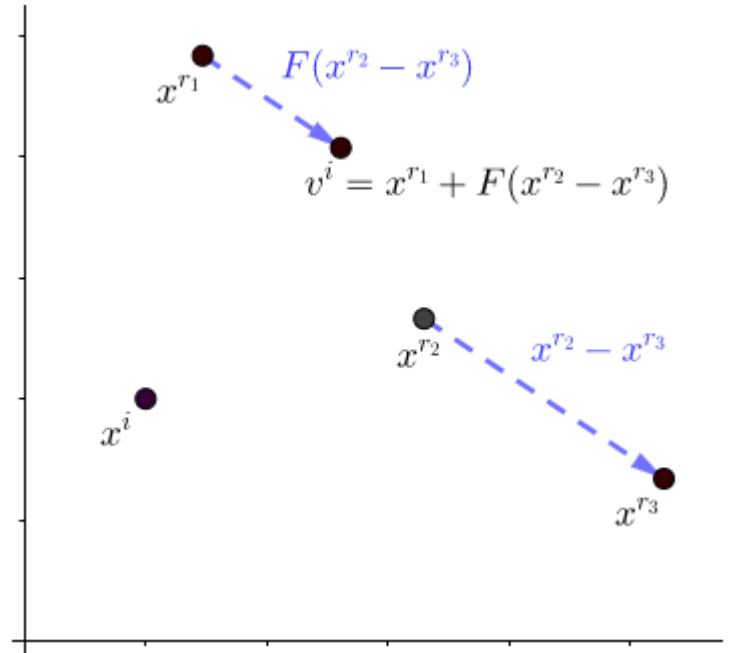
Mutation is the main operation in Differential Evolution. In this operator, for each individual x^i , another one called v^i is generated. The first step consists of randomly selecting three individuals from the population: x^{r1} , x^{r2} , x^{r3} . The new individual v^i is calculated as follows:

$$v^i = x^{r1} + F(x^{r2} - x^{r3})$$

Where F is a random number between 0 and 2.

The difference between x^{r2} and x^{r3} defines the direction and the magnitude of mutation. F slightly changes the magnitude. x^{r1} Represents the initial point.

At the beginning, all the individuals are dispersed and $x^{r2} - x^{r3}$ is big, but when the algorithm is converging, the individuals are concentrated in some local minimums and the value of $x^{r2} - x^{r3}$ is smaller. It is magic!



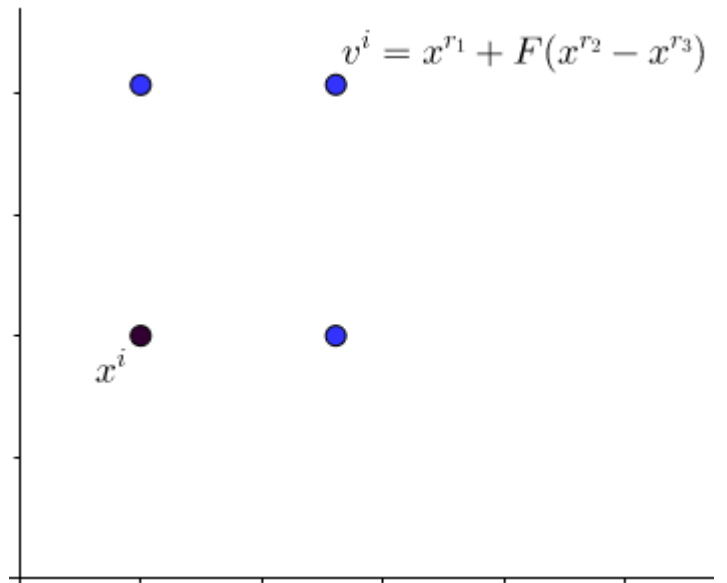
Crossover

Crossover's objective is to combine the original vector x^i with the new one v^i for creating another one u^i .

$$u_k^i = \begin{cases} v_k^i & \text{If } \text{rand}(0,1) \leq Cr \text{ or } k = l \\ x_k^i & \text{otherwise} \end{cases}$$

For each variable k of u^i , the value is selected randomly between v^i or x^i . If a random number between 0 and 1 is less than Cr , the value is taken from v^i . For guarantying that at less one value is taken from v^i , the value of a variable randomly selected l , is assigned with the value of v^i .

The new vector u^i is positioned in one of the corners of the hyperrectangle generated with the positions of x^i and v^i .



Selection

The selection is performed by tournament. The best individual of u^i and x^i is selected to be part of the next generation.

Differential Evolution Algorithm

Parameters:

- N, population size
- G, Maximum number of generations
- Cr, crossover probability

Return: the best individual

Begin

Create the initial population of N individuals

Calculate the population fitness

While the number of generations is less than G or we haven't found a good solution

For each individual x^i in the population

Mutation: Create a new individual (v^i)

Crossover: combine the individuals $x^i, v^i \rightarrow u^i$

Calculate the fitness of u^i

Selection: select the best individual between $x^i = best(x^i, u^i)$

End for

End while

End

Return the best individual

Constraint Handling

In an optimization problem, we can identify some key concepts:

- **Objective function:** it is a numeric value that represents how good is a solution. It could be money, time, energy, error, force, space, etcetera.
- **Variables:** are the values that we can change for improving the objective function.
- **Constraints:** Sometimes, features' values must accomplish some requirements. For example, the number of people working in a process needs to be more or equal to 1.

The presence of constraints implies that not all possible combinations of variable values represent valid solutions to the problem at hand. Unfortunately, constraint handling is not straightforward in an EA, because the variation operators (mutation and recombination) are typically "blind" to constraints. That is, there is no guarantee that even if the parents satisfy some constraints, the offspring will satisfy them as well.

		Objective function	
		Yes	No
Constraints	Yes	Constrained optimization problem Knapsack problem	Constraint satisfaction problems 8 queen problem Sudoku
	No	Free optimization problem Linear Regression Image Transformation	There is no problem

Constraint Handling can be performed:

1. Indirect constraint handling (transformation) coincides with penalizing constraint violations.
2. Direct constraint handling (enforcement) can be carried out in two different ways:
 - a) Allowing the generation of candidate solutions that violate constraints: repairing infeasible candidates.
 - b) Not allowing the generation of candidate solutions that violate constraints: preserving feasibility by suitable operators (and initialization).
3. Mapping constraint handling is the same as decoding, i.e., transforming the search space into another one.

In general, the presence of constraints will divide the space of potential solutions into two or more disjoint regions, the **feasible region** (or regions), containing those candidate solutions that satisfy the given feasibility condition, and the **infeasible region** containing those that do not.

PENALTY FUNCTIONS

Assuming a minimization problem, the use of penalty functions constitutes a mapping from the objective function $f(x)$ to $f'(x)$ such that

$$f'(x) = f(x) + P(d(x, F))$$

where F is the feasible region as before, $d(x, F)$ is a distance metric of the infeasible point to the feasible region (this might be simply a count of the number of constraints violated) and the penalty function P is monotonically increasing nonnegatively such that $P(0) = 0$.

If the penalty function is too severe, then infeasible points near the constraint boundary will be discarded, which may delay, or even prevent, exploration of this region. Equally, if the penalty function is not sufficient in magnitude, then solutions in infeasible regions may dominate those in feasible regions, leading to the algorithm spending too much time in the infeasible regions and possibly stagnating there. In general, for a system with m constraints, the form of the penalty function is a weighted sum

$$P(d(x, F)) = \sum_{i=1}^m w_i d_i(x)$$

The function $d_i(x)$ represents the distance from the point x to the boundary for the constraint i , and w_i represents the weight of the constraint i . Penalization functions are classified as static and dynamic.

Static Penalty Functions: Three methods have commonly been used with static penalty functions, namely *extinctive* penalties (where all of the w_i are set so high as to prevent the use of infeasible solutions), *binary* penalties (where the value d_i is 1 if the constraint is violated, and zero otherwise), and distance-based penalties. It has been reported that of these three the latter gives the best results, and the literature contains many examples of this approach. This approach relies on the ability to specify a distance metric that accurately reflects the difficulty of repairing the solution, which is obviously problem-dependent, and may also vary from constraint to constraint. However, the main problem in using static penalty functions remains the setting of the values of w_i . In some situations, it may be possible to find these by experimentation, using repeated runs and incorporating domain-specific knowledge, but this is a time-consuming process that is not always possible.

Dynamic Penalty Functions: One approach is described as follows. A population is evolved in a number of stages - the same number as there are constraints. In each stage i , the fitness function used to evaluate the population is a combination of the distance function for constraint i with a death penalty for all solutions violating constraints $j \neq i$. In the final stage all constraints are active, and the objective function is used as the fitness function. It should be noted that different results may be obtained, depending on the order in which the constraints are dealt with.

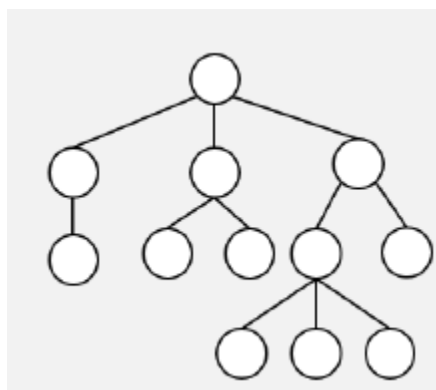
REPAIR FUNCTIONS

The repair algorithm works by taking an infeasible point and generating a feasible solution based on it. In some problems, this technique is relatively simple. However, in general, defining a repair function may be as complex as solving the problem itself.

In continuous optimization problems, the most popular methodology consists of taking an infeasible point and another feasible point and using binary search for finding a feasible point that stands in the segment joining those points.

Genetic Programming (GP)

Genetic programming (GP), proposed by John Koza around the 1990s, is one of the youngest members of the evolutionary algorithm family. It automatically solves problems without requiring the user to know or specify the structure of the solution in advance. The main idea of GP is to evolve a population of computer programs, where individuals are commonly represented as syntax trees, as it is shown in the next figure. While the EAs are typically applied to optimization problems, **GP could instead be positioned in machine learning**.



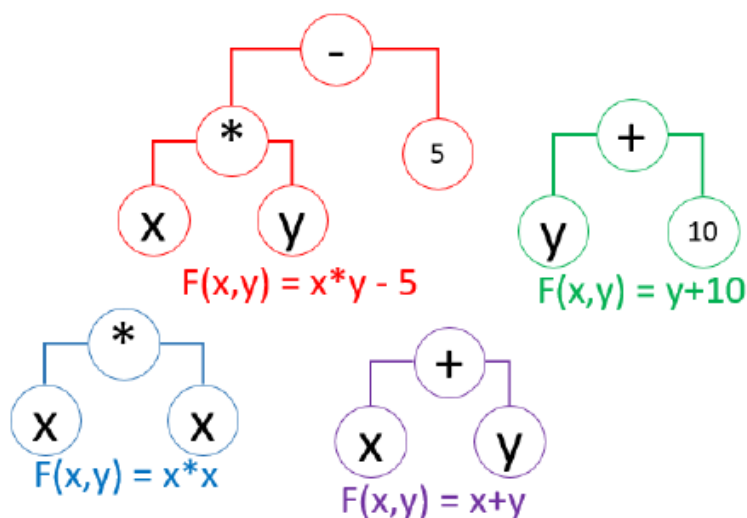
A supervised learning problem consists of mapping several inputs with outputs.

x	y	F(x,y)
5	9	21
0	7	3
1	5	5
4	7	15
1	2	8
9	7	75
5	6	24
3	8	8
3	3	13
2	4	8

$$F(x,y) = ?$$

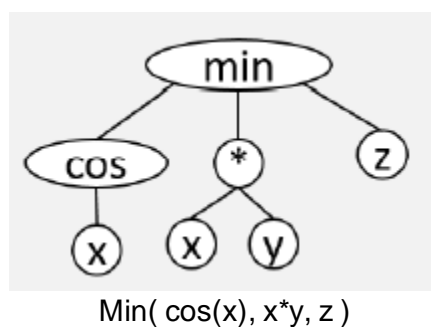
When Genetic Programming is used for solving supervised learning problem, each GP individual represents a solution, as an equation, to solve the problem.

x	y	F(x,y)
5	9	21
0	7	3
1	5	5
4	7	15
1	2	8
9	7	75
5	6	24
3	8	8
3	3	13
2	4	8



Representation

In Genetic Programming, individuals are usually represented as syntax trees.



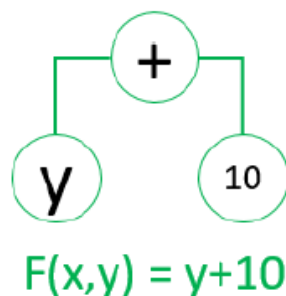
The elements of a GP individual are:

- *Terminals*, which correspond to the leaf nodes in a syntax tree. They can be inputs, typically called variables (e.g., x and y), functions without arguments (e.g., $\text{rand}()$), or constants that can be predefined or randomly generated. The terminal set T defines all the terminal elements.
- *Functions*, which correspond to the internal nodes in a syntax tree and can be seen as the operations. The function set F is defined by the problem's nature. For example, for numeric problems, it could be formed by arithmetic operators and trigonometric functions.

Fitness

A fitness function returns a numeric value that indicates how well the individual solves the problem. Where Genetic Programming is used to solve supervised learning problems, the objective is to find an individual whose function matches inputs with outputs. For each training sample the inputs values can be evaluated in the individual function for calculating the output. In this case, the fitness function must measure how much the individual outputs are similar to the real outputs.

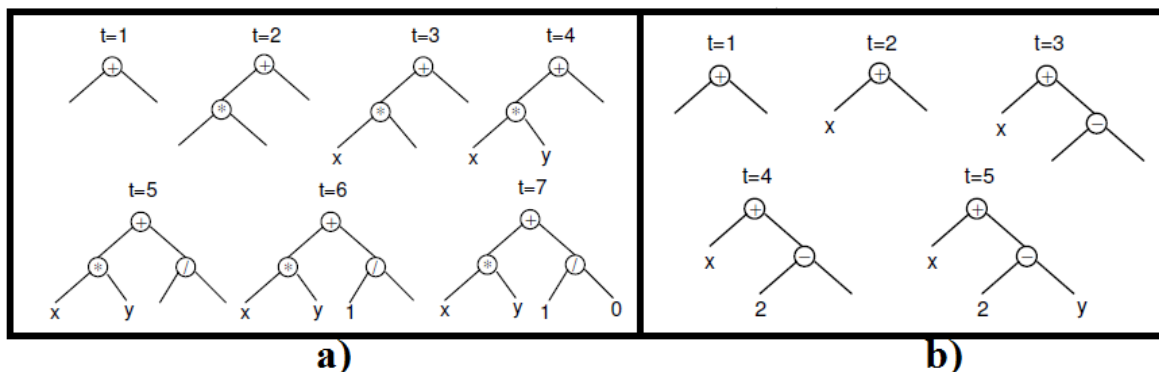
x	y	F(x,y)
5	9	21
0	7	3
1	5	5
4	7	15
1	2	8
9	7	75
5	6	24
3	8	8
3	3	13
2	4	8



19
17
15
17
12
17
16
18
13
14

Initial population

There are three classical techniques to generate the initial population. The first one is called *full*, where all the trees are randomly created, and all the leaves have the same depth. It means, in all levels, except the last one, the nodes are internal nodes selecting elements randomly from the function set F and only in the last level the elements are selected randomly from the terminal set T . The second technique is called *grow*; in this case, each node selects an element randomly from either the function set F or the terminal set T . Finally, the technique *Ramped half-and-half* where half of the population is created with the *full* technique and the other half with *grow*. After creating the initial population, the fitness of all individuals is calculated. Also, the best program, based on fitness, and its fitness are stored.



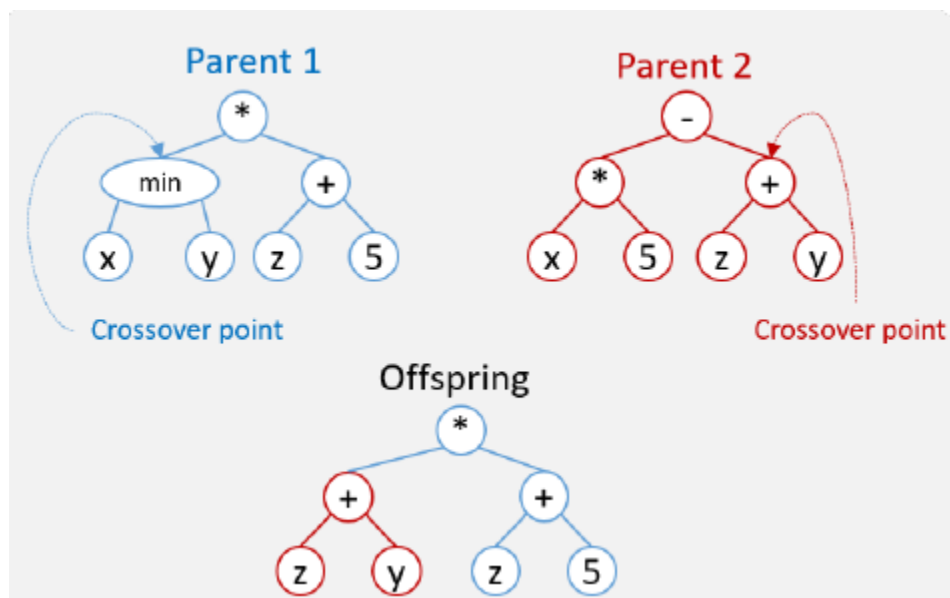
Initial population a) Full b) Grow

Selection

As most evolutionary algorithms, in GP, the genetic operators (crossover and mutation) are applied to individuals that are stochastically selected based on fitness. This is, more adapted individuals have more chances to be part of the evolutionary process than the ones who are less adapted. In GP, tournament selection is the most popular technique. *Tournament selection* is a direct comparison of fitness among individuals. It can be either binary (two individuals) or with more than two individuals that are randomly selected from the population. The fittest one wins.

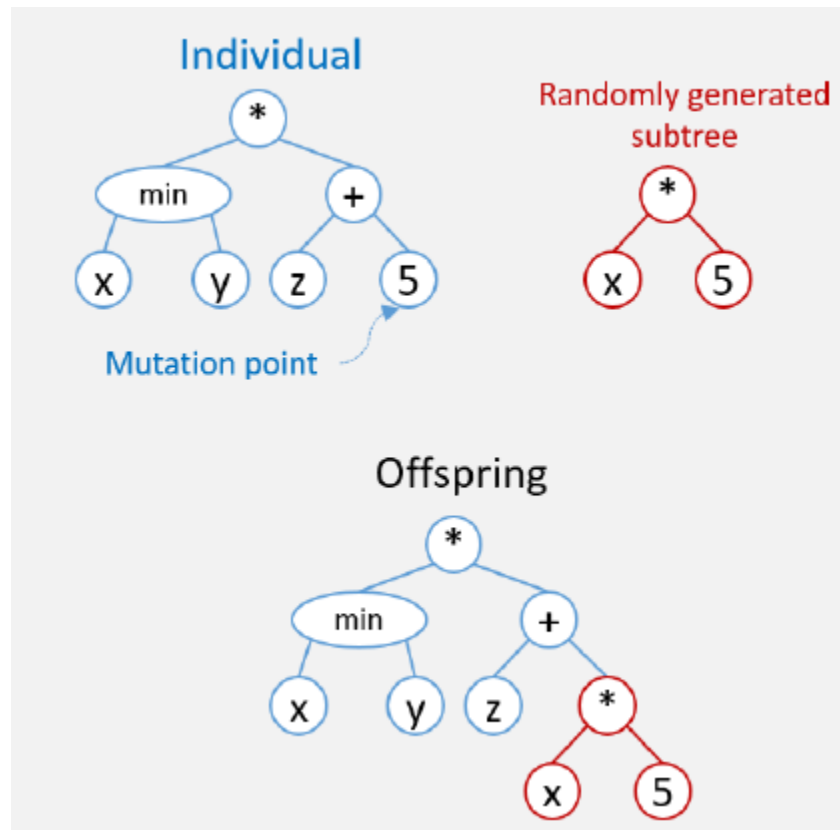
Crossover

The objective of crossover in GP, as well in other evolutive algorithms, is to combine the characteristics of two or more individuals, called parents, to generate the offspring. The classic crossover consists of randomly selecting a crossover point in each parent and create an offspring based on the first parent but replacing the selected node by the subtree of the second parent on the crossover point.



Mutation

The goal of mutation is to change an individual. The most used form of mutation in GP, called subtree mutation, randomly selects a mutation point in a tree and substitutes the subtree rooted there with a randomly generated subtree. Another common mutation is point mutation, where a function node is replaced for another one selecting an element from the function set F with the same arity.



Algorithm

Two different population models exist in Evolutionary Computation: the **generational model** and the **steady-state model**. In the first one, in each generation, we begin with a population of individuals, and several parents are selected from that population to generate the offspring by the application of variation operators. After each generation, the whole population is replaced by its offspring, which is called the next generation. On the other side, in the **steady-state model**, the entire population is not changed at once, but rather a part of it. Usually, in each iteration, an individual is generated, and it replaces another individual in the population. Most of the recent GP implementation uses the steady-state model. Once the new program is created, it is added to the population, and to maintain the population size, it replaces another one that is chosen using negative selection. In GP, the most used technique for negative selection is negative tournament, it consists of randomly selecting several programs from the population, and the worst of them, based on fitness, is the one that is going to be replaced. In addition, the new program is compared with the best one, and the fittest is kept as the best program.

The population is iteratively changed until a **termination criterion** is reached. The termination criterion may be a maximum number of iterations to be run as well as a problem-specific success predicate. Also, a practical criterion is called **early stopping**, where the training set is divided into a training and a validation sets. The training set is used for guiding the learning process. The evolution stops when the fitness individual on the validation set has not been updated in a defined number of iterations.

The final model corresponds to the fittest individual on the validation set.

Algorithm 1: Evolution process of Genetic Programming. The underlined steps correspond to the ones that are analyzed in this dissertation.

```

Create an initial population  $\mathcal{P}$  of programs;
Calculate the fitness of all programs in  $\mathcal{P}$ ;
 $best, best_{fitness} \leftarrow$  The best program of  $\mathcal{P}$  and its fitness;
while a termination criterion is not reached do
     $parent_1, parent_2 \leftarrow$  Two programs that are selected from  $\mathcal{P}$  based on fitness;
     $new \leftarrow$  The program that is the result of applying the variation
        operators (crossover and mutation) to the programs  $parent_1$  and
         $parent_2$ ;
     $old \leftarrow$  A program that is chosen from  $\mathcal{P}$  using negative selection;
    Remove  $old$  from  $\mathcal{P}$ ;
    Add  $new$  to  $\mathcal{P}$ ;
     $new_{fitness} \leftarrow$  The fitness of  $new$ ;
    if  $new_{fitness} > best_{fitness}$  then
         $best \leftarrow new$ ;
         $best_{fitness} \leftarrow new_{fitness}$ ;
    end
end
Return  $best$ ;
```

Metaheuristics

Particle Swarm Optimization

Kennedy and Eberhart proposed Particle Swarm Optimization in 1995. It is inspired by the movement of a flock of birds when searching for food.

Each particle i represents a solution for the problem. In the time t , it has a position $x^i(t) \in \mathbb{R}^d$ and a velocity $v^i \in \mathbb{R}^d$.

$$x^i(t) = \langle x_1^i, x_2^i, \dots, x_d^i \rangle$$

The positions and velocities are updated following the next equations, where P_{best}^i is the best position where the particle i has been, G_{best} is the best location founded until the moment, r_1 and r_2 are random numbers between 0 and 1, and w, c_1 , and c_2 are hyper parameters. Those last values can be initialized at 0.9 and gradually reducing it until 0.1.

$$\begin{aligned} v^i(t+1) &= w v^i(t) + c_1 r_1 (P_{best}^i - x^i(t)) + c_2 r_2 (G_{best} - x^i(t)) \\ x^i(t+1) &= x^i(t) + v^i(t+1) \end{aligned}$$

PSO Algorithm

Parameters:

- N, Number of particles
- MaxIter, Maximum number of iterations
- func, objective function
- bounds, the search-space

Return: the best position G_{best}

Begin

Initialize c_1, c_2, w

Create the particles positions and velocities randomly

Calculate the objective function values

Calculate P_{best}^i as the current positions

Calculate G_{best}

While $t < \text{MaxIter}$ or we haven't found a good solution

For each particle i

Update the velocity:

$$v^i(t+1) = w v^i(t) + c_1 r_1 (P_{best}^i - x^i(t)) + c_2 r_2 (G_{best} - x^i(t))$$

Update the position:

$$x^i(t+1) = x^i(t) + v^i(t+1)$$

Calculate $\text{func}(x^i)$

If $f(x^i) < \text{func}(P_{best}^i)$: update P_{best}^i

If $f(x^i) < \text{func}(G_{best})$: update G_{best}

End for

Decrease c_1, c_2, w

End while

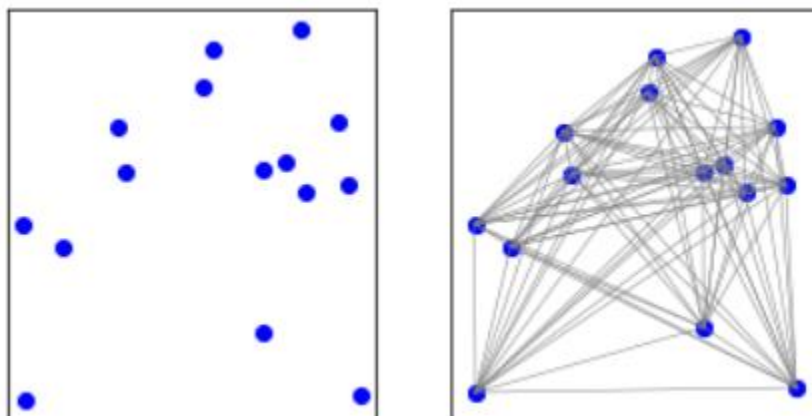
End

Return G_{best}

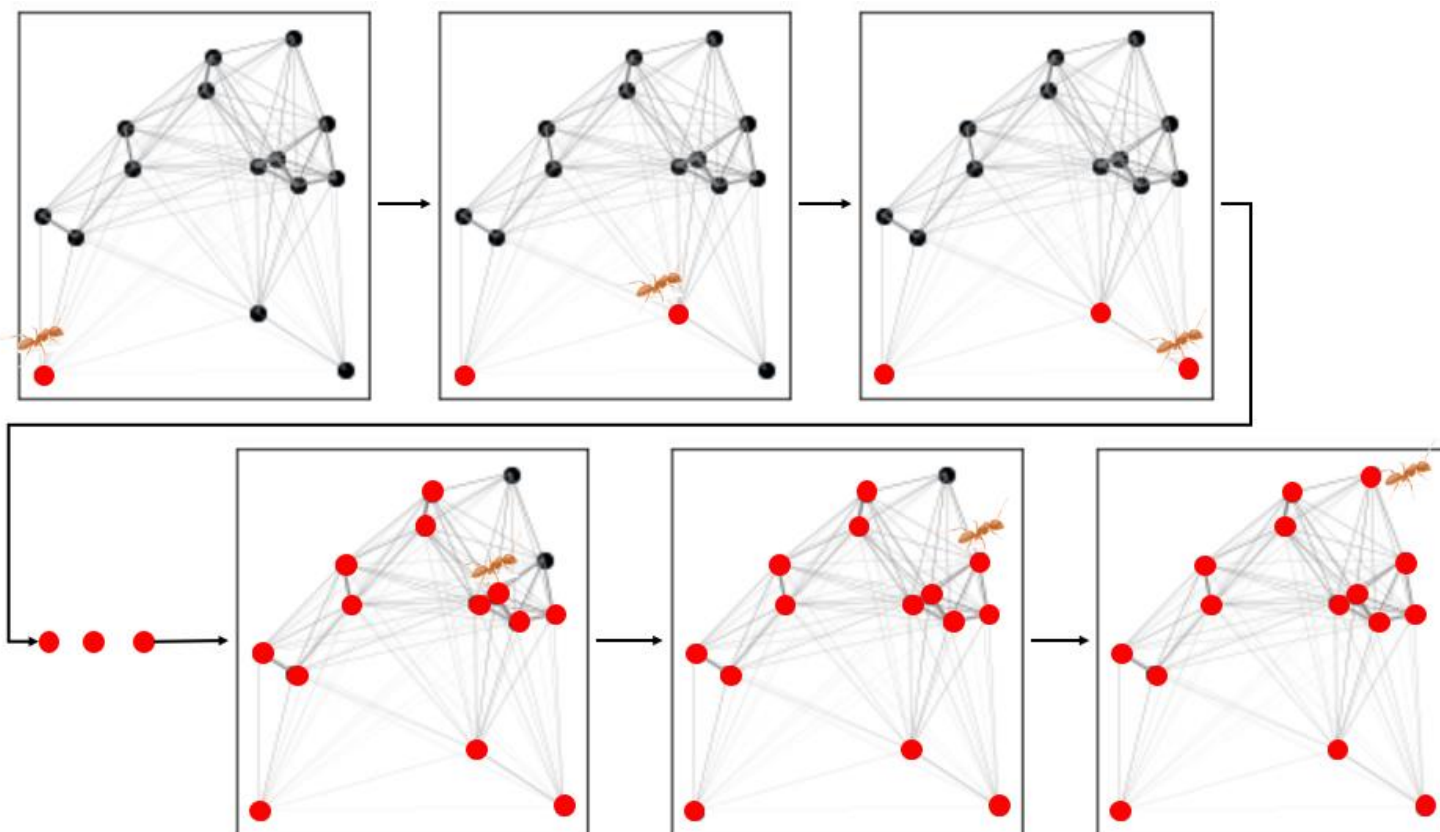
Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) solves problems of finding paths in graphs. It is inspired by the ants' behavior when searching for food. The ants leave pheromones that allow other ants to follow the path to food. The more ants go for a specific path, the more pheromones.

The following example aims to find the shortest path to visit all nodes without repetition.



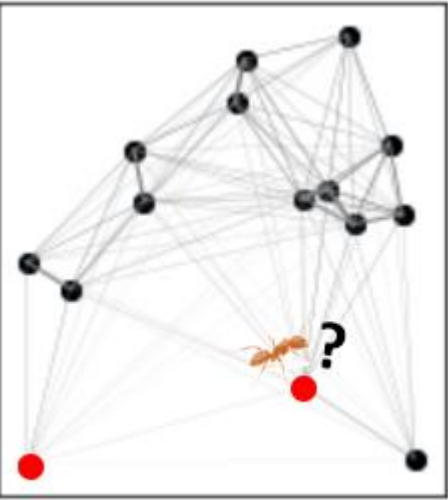
In this algorithm, an artificial ant must simulate a path starting at a specific point. It moves node by node, choosing based on the pheromones of each path.



First, we start with the abbreviations:

- c_{ij} : path from the node i to the node j
- τ_{ij} : pheromones in the path from the node i to the node j
- η_{ij} : heuristic in the path from the node i to the node j
- ρ : evaporation rate, between 0 and 1

Selecting the following node:



$$P(c_{ik}) = \frac{\tau_{ik}^{\alpha} * \eta_{ik}^{\beta}}{\sum_{j \in N_i} \tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}$$

Where N_i represents all the nodes where the ant can move in the next step. In the image, it represents all the black nodes. α and β are parameters that control the relative importance of the pheromone versus the heuristic information. $\eta_{ik} = 1/d_{ik}$, where d_{ik} is the length of component c_{ik} .

At the beginning of the algorithm:

- All the pheromones can be initialized with a small value.

Move the ants one by one:

- Ants start to move around the graph node by node using the previous equation.
- All the ants must move to the graph.

When all the ants record the graph:

- The pheromones must be updated.
- Ants deposit pheromones to their path proportional to its distance.
- The pheromones evaporate

The ant a update its path based on the following equation:

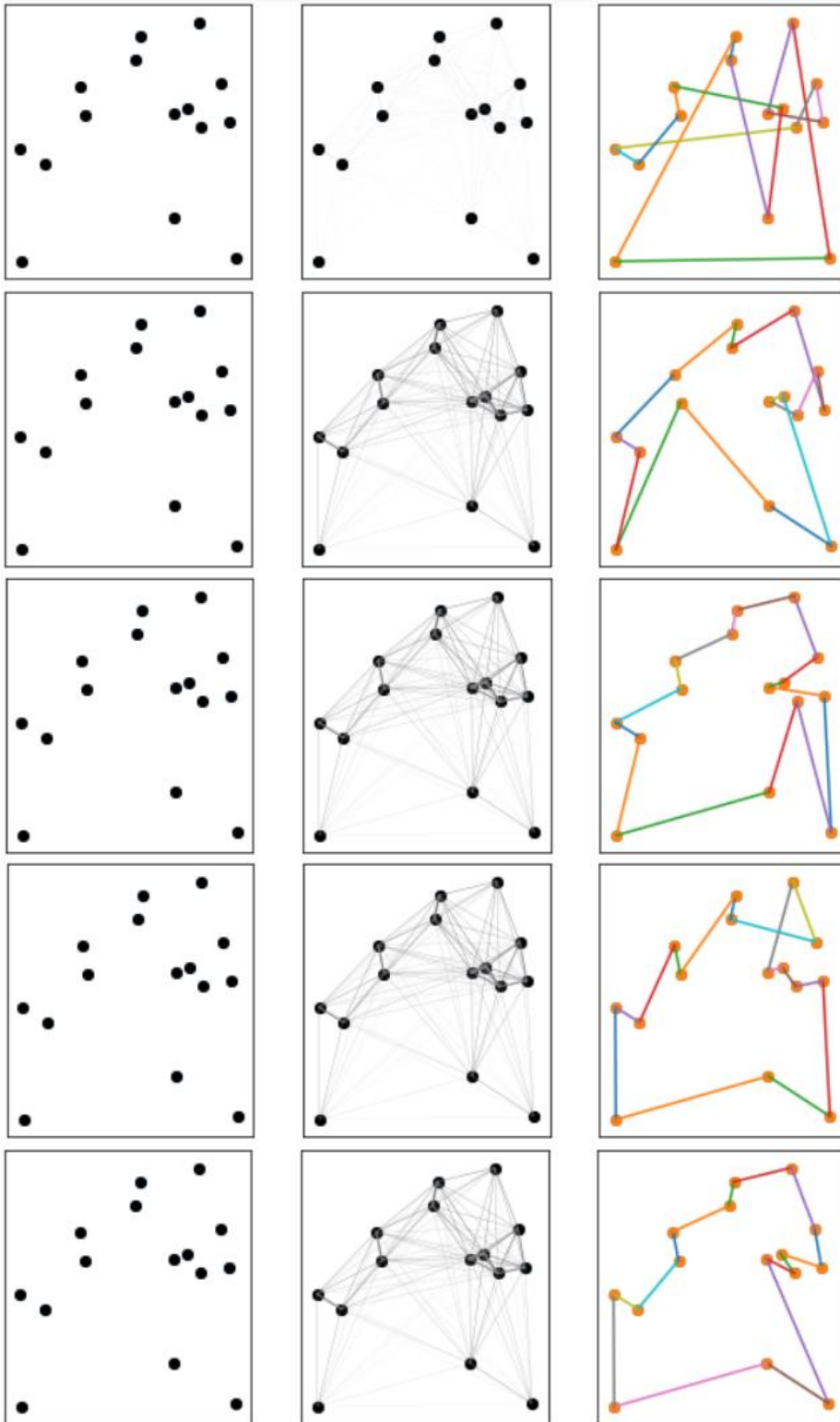
$$\Delta\tau_{ij}^a = \begin{cases} \frac{1}{L_a} & \text{if ant } a \text{ used } c_{ij} \\ 0 & \text{otherwise} \end{cases}$$

where, L_a is the total length of the path used by the ant a .

All the pheromones evaporate based on the following equation:

$$\tau_{ij} = (1 - \rho) * \tau_{ij}$$

Example:



Bibliografía

- [1] A. Eiben and J. Smith, Introduction to Evolutionary Computing, Amsterdam: Springer, 2007.
- [2] R. Poli, W. Langdon and N. McPhee, A Field Guide to Genetic Programming, 2008.