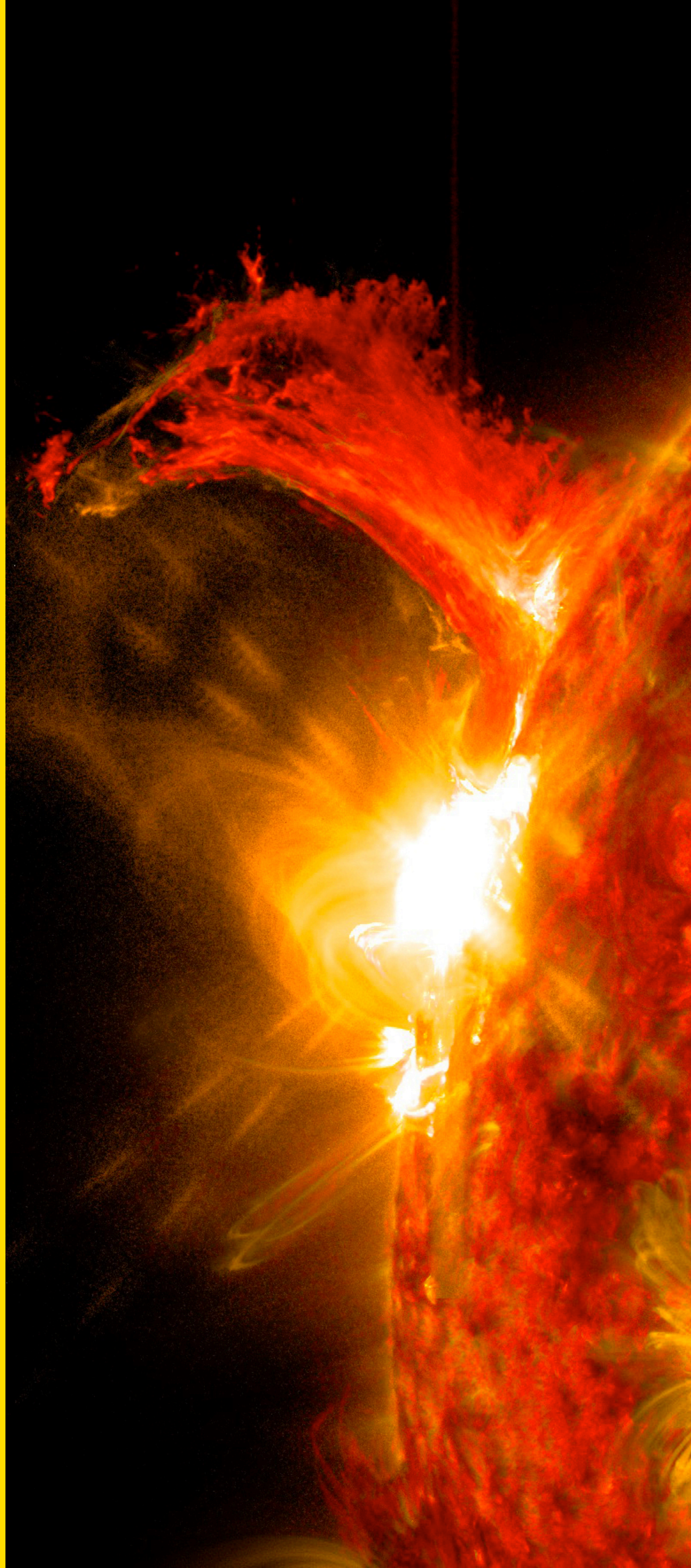
UNIVERSITÀ DEGLI STUDI DI BRESCIA

INFN
Istituto Nazionale di Fisica Nucleare

Metaheuristic optimization for artificial neural networks and deep learning architectures

D. Pagano

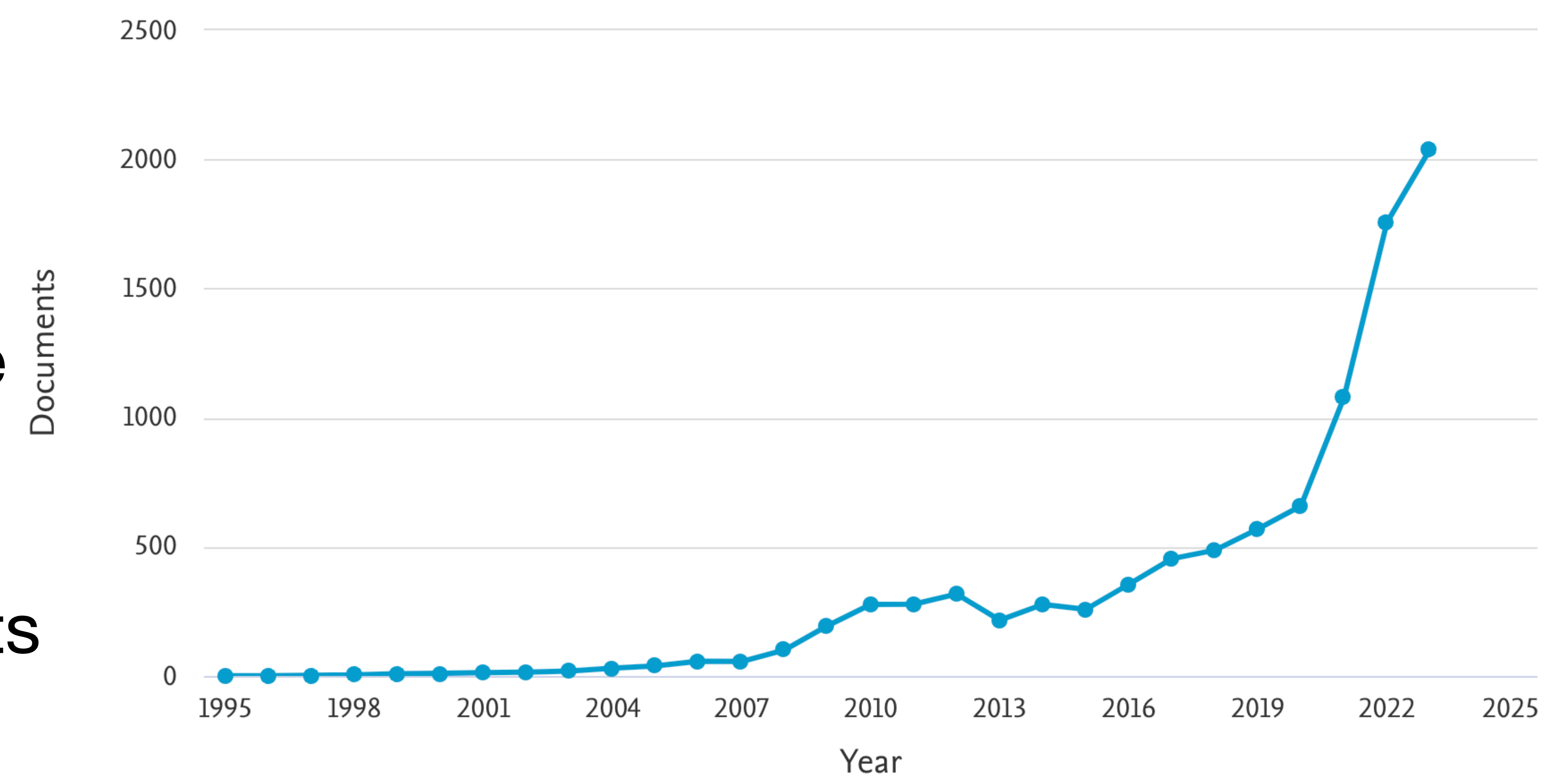
UNIVERSITÀ DEGLI STUDI DI BRESCIA & INFN PAVIA



Outline

- Metaheuristic algorithms for optimization
- Artificial neural networks and deep learning architectures
- EvoMiP library for python
- Few examples

Metaheuristic algorithms



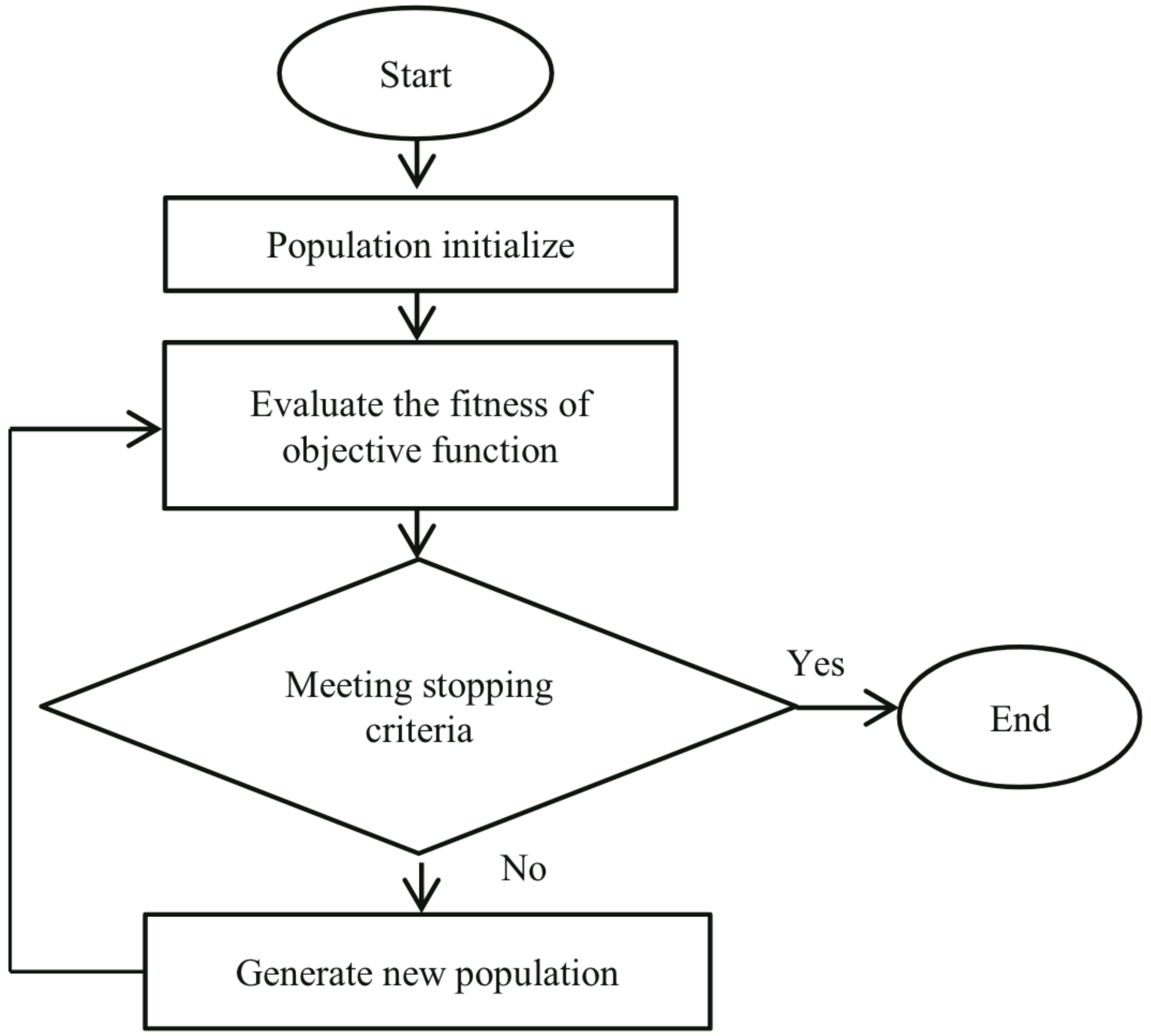
- The interest on **metaheuristic** (also know as **evolutionary**) algorithms (MHAs) has been growing steadily in the last decade

- (Glover, 1986) *heuristic*: algorithms with stochastic components
  meta: beyond

- MHAs are generally based on a metaphor of a natural or a man-made process: the search for food, the haunting of nearly any species of animals, musicians playing together, ...

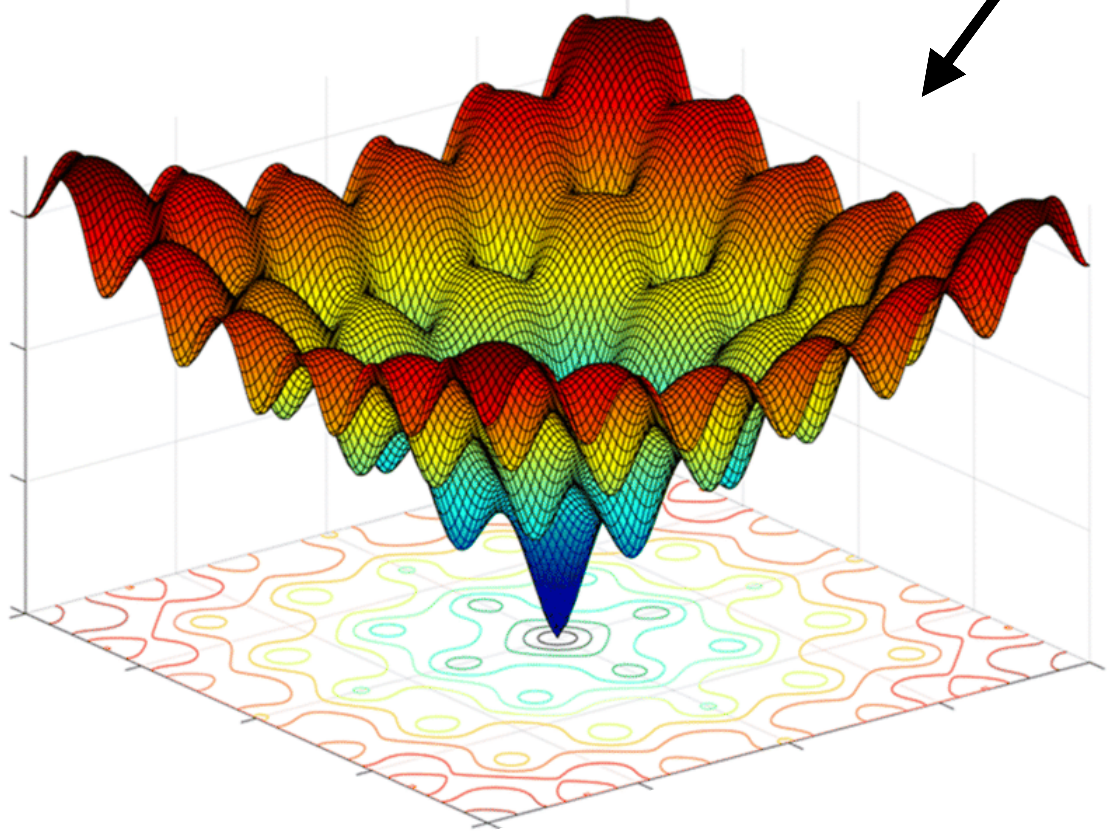
- This got out of hand though... https://doi.org/10.1111/itor.12001

- Nowadays, MHAs are all population-based and differences between them concern how to handle the *exploration* and *exploitation* of the search space

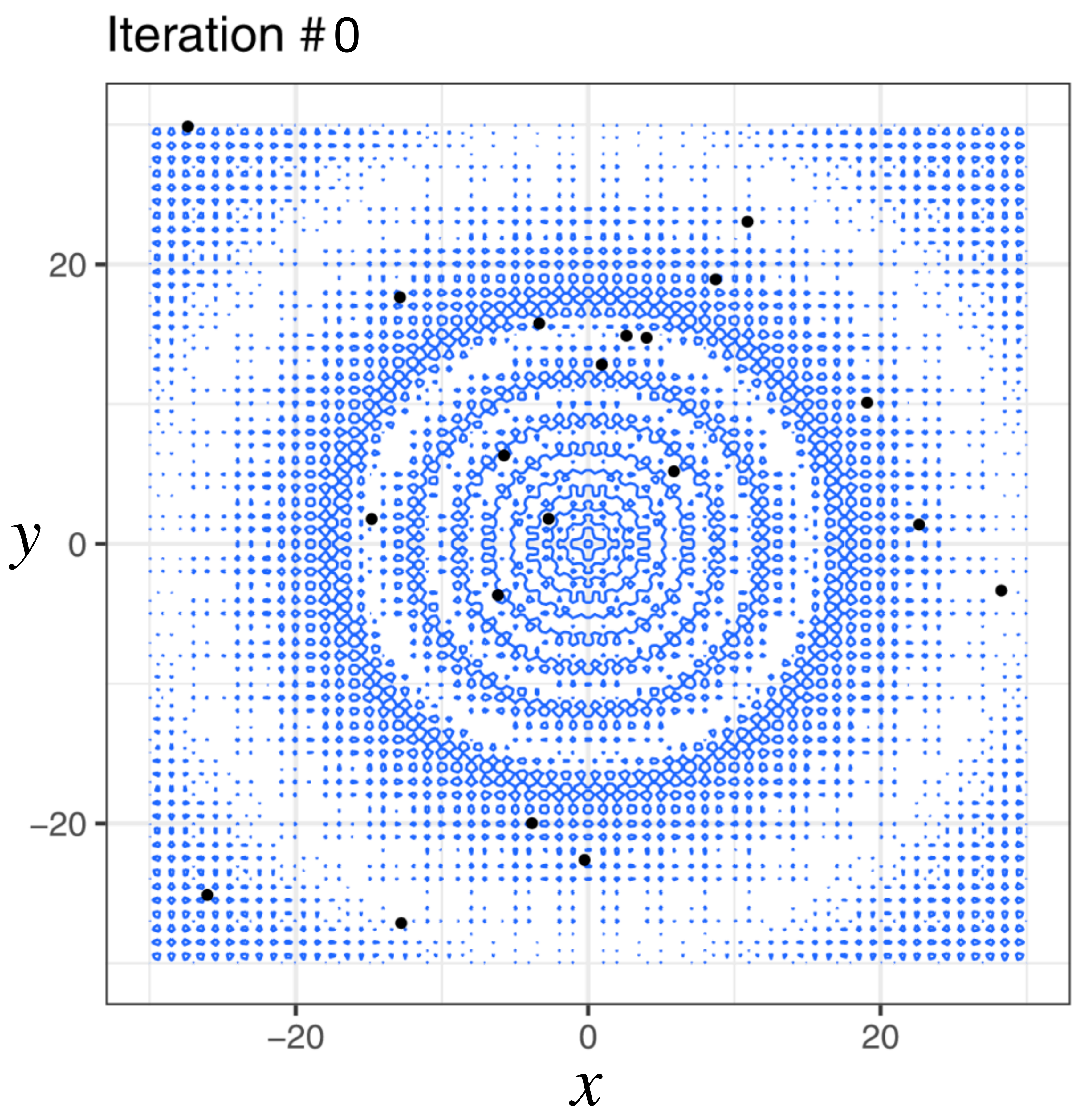


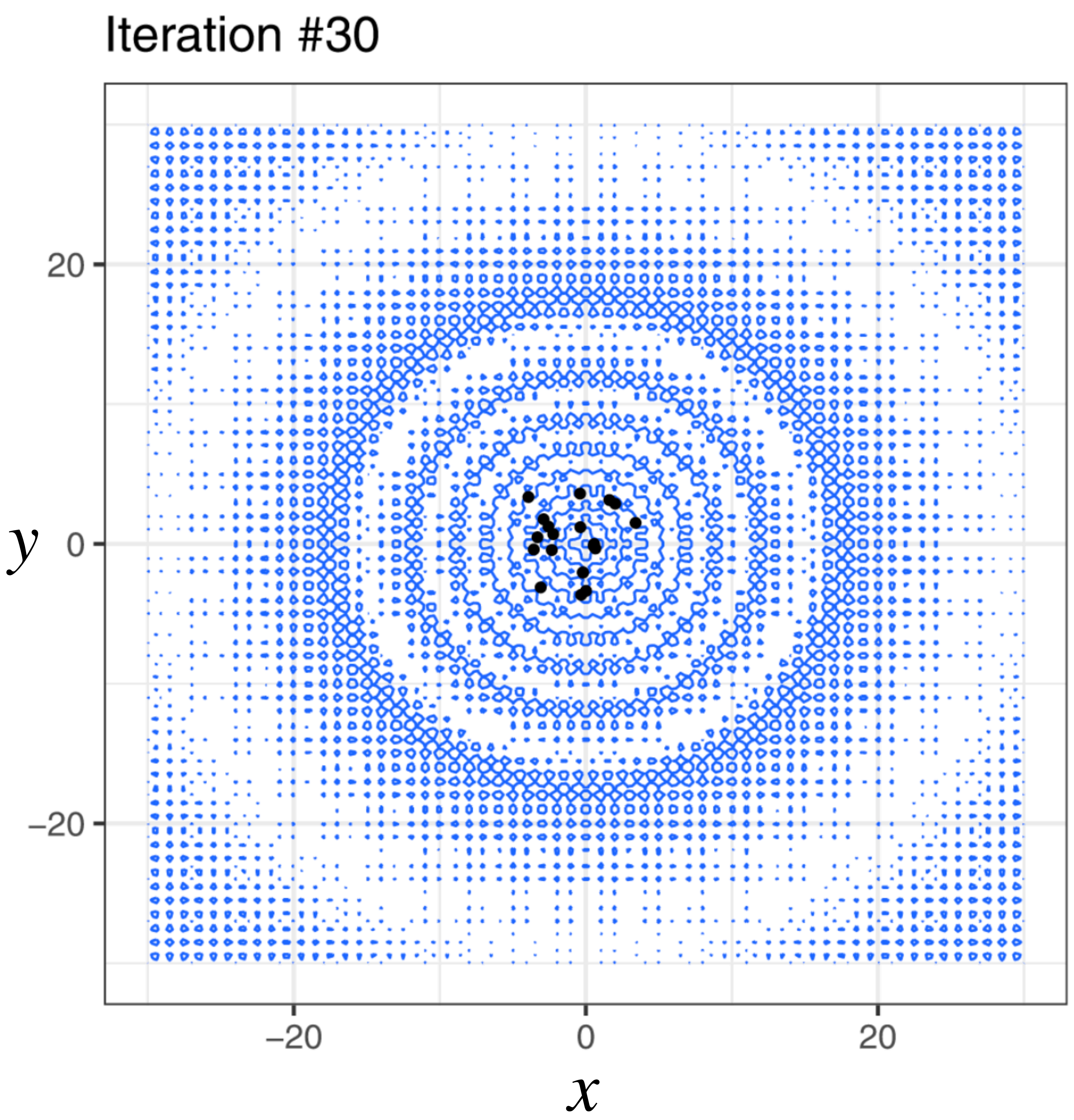
Example

- Minimization of the **2D-Ackley function** with the Particle Swarm algorithm



$$f(x,y) = -20\exp\left[-0.2\sqrt{0.5\left(x^2+y^2\right)}\right] + -\exp\left\{0.5\left[\cos(2\pi x)+\cos(2\pi y)\right]\right\}$$

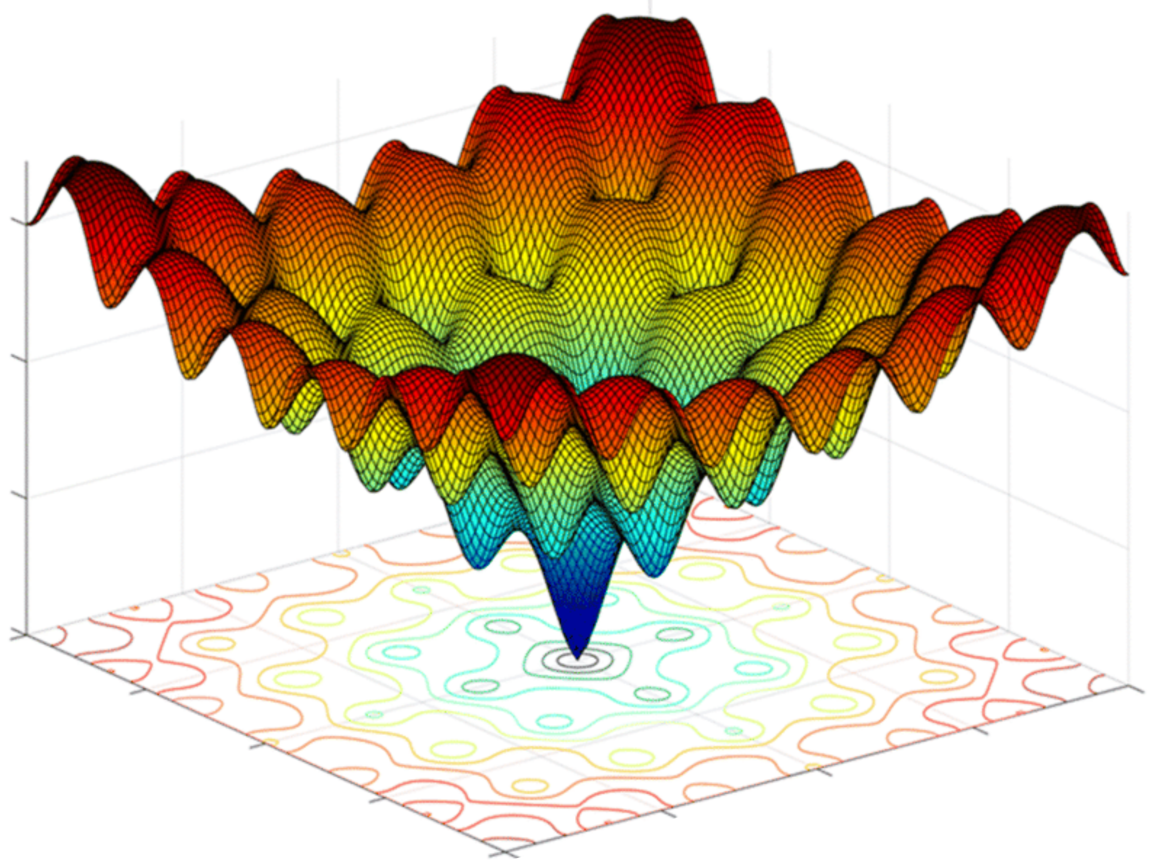




Gradient-based methods

- Include a large set of methods based on the calculation of the gradient of the cost (objective) function:
  - Gradient descent
  - Stochastic gradient descent
  - Back propagation
  - Levenberg Marquardt
  - Conjugate gradient
  - Adaptive Moment Estimation (ADAM)
  - ...
- They are extremely popular and very efficient for convex functions
- On the other hand, they are sensitive to the choice of the initial point, to step size, and to noise in the function
- Moreover, they can get stuck in local optima, or saddle points, failing to explore the global optimum

2D-Ackley function vs gradient descent



$$f(x,y) = -20\exp\left[-0.2\sqrt{0.5\left(x^2+y^2\right)}\right] - \exp\left\{0.5\left[\cos(2\pi x)+\cos(2\pi y)\right]\right\} + e + 20$$

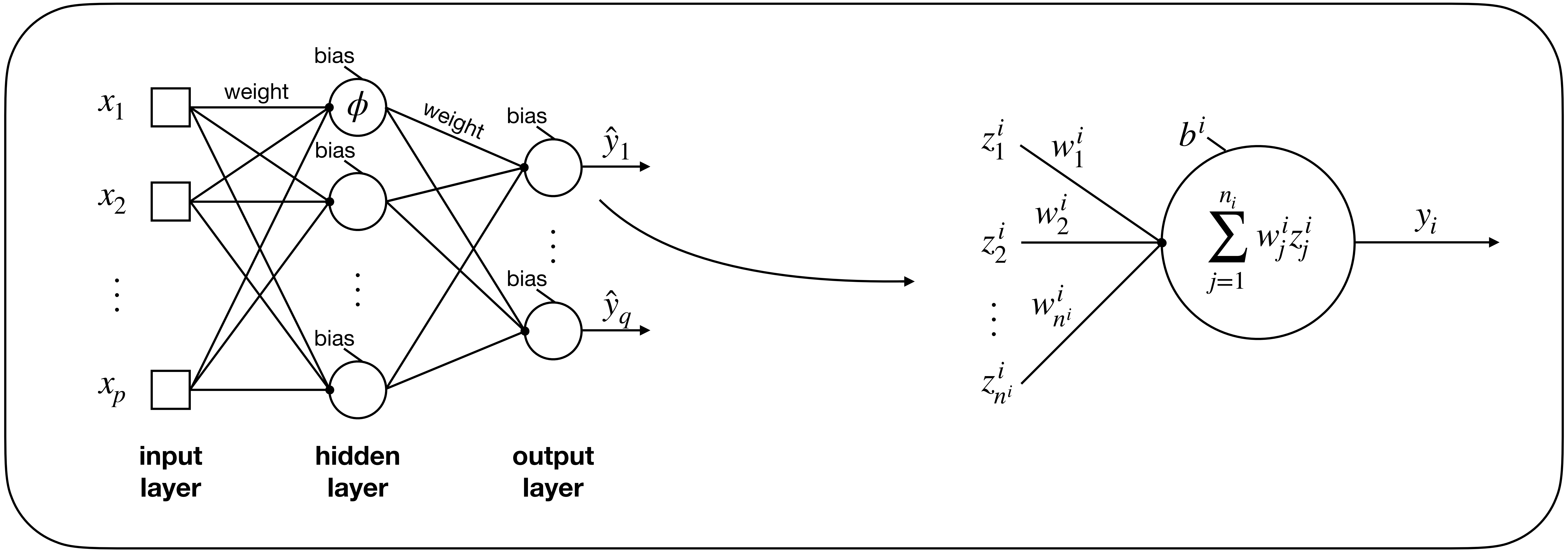
Derivatives

$$\frac{\partial}{\partial x} f(x,t) = \frac{4\sqrt{0.5}x \cdot \exp\left[-0.2\sqrt{0.5\left(x^2+y^2\right)}\right]}{\sqrt{x^2+y^2}} + \pi\sin(2\pi x)\cdot\exp\left\{0.5\left[\cos(2\pi x)+\cos(2\pi y)\right]\right\}$$

$$\frac{\partial}{\partial y} f(x,t) = \frac{4\sqrt{0.5}y \cdot \exp\left[-0.2\sqrt{0.5\left(x^2+y^2\right)}\right]}{\sqrt{x^2+y^2}} + \pi\sin\left(2\pi y\right)\cdot\exp\left\{0.5\left[\cos(2\pi x)+\cos(2\pi y)\right]\right\}$$

Notebook

Optimization of neural networks: a FNN example



- $y_i = \phi_i \left(\sum_{j=1}^{n^i} w^i_j z^i_j + b^i \right)$, where ϕ_i is the activation function, z^i is the input, w^i is the weight and b^i is the bias

- Previous FNN can be seen as a function $\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{w})$, where $\mathbf{x} = \langle x_1, x_2, \ldots, x_p \rangle$, $\mathbf{w} = \langle w_1, w_2, \ldots, w_n \rangle$

Components of an FNN optimization

- **Architecture**
  - number of layers in the network
  - number of nodes in the hidden layers
  - arrangement of the connections between nodes
  - ...
- **Activation function**
- **Learning environment**
  - supervised learning, reinforcement learning, ...
- **Learning algorithm**
- **Weights:** $\mathbf{w} = \langle w_1, w_2, \ldots, w_n \rangle$

In many cases (especially in our field) it is the only component which is optimised

Optimization of weights

- In a supervised learning, we want to minimise the difference/distance between the desired output $\mathbf{y}$ and the model's output $\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{w})$ measured by a cost function: $c_f : Y \times \hat{Y} \longrightarrow \mathbb{R}_{\geq 0}$

- Popular choices: *mean squared error* (regression); *accuracy* and *misclassification rate* (classification) $\longrightarrow$ $c_f(\mathbf{y}_i, \hat{\mathbf{y}}_i) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{q} \left(y_{ij} - \hat{y}_{ij}\right)^2$

- Learning algorithm: *Backpropagation*

Delta rule: $\mathbf{w}^{t+1} = \mathbf{w}^t + \Delta \mathbf{w}^t$

$$\Delta \mathbf{w}_l^t = \alpha^t \mathbf{w}_l^{t-1} + \eta^t \cdot \frac{\partial c_f}{\partial \mathbf{w}^t} \mathbf{y}_{l-1}$$

- Learning algorithm: *Metaheuristic*

$\mathbf{w}^t$: **known** values (at t) with minimum $c_f(\mathbf{y}_i, \hat{\mathbf{y}}_i)$

EvoMiP

- **EvoMiP** is a Python library, based on the package for **R** called **EmiR** (from the same authors)

- It includes some of the most popular population-based metaheuristic algorithms:
  - Artificial Bee Colony algorithm (ABC)
  - Bat algorithm (BAT)
  - Cuckoo Search (CS)
  - Genetic Algorithms (GA)
  - Gravitational Search Algorithm (GSA)
  - Grey Wolf Optimization (GWO)
  - Harmony Search (HS)
  - Improved Harmony Search (IHS)
  - Moth-flame Optimization (MFO)
  - Particle Swarm optimization (PS)
  - Simulated Annealing (SA)
  - Whale Optimization Algorithm (WOA)

SoftwareX 18 (2022) 101083

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

ELSEVIER

Original software publication

EmiR: Evolutionary minimization for R

Davide Pagano *, Lorenzo Sostero

Department of Mechanical and Industrial Engineering, University of Brescia, Brescia, Italy

A R T I C L E I N F O

Article history:
Received 21 January 2022
Received in revised form 22 March 2022
Accepted 5 April 2022

Keywords:
Evolutionary algorithms
Optimization
R

A B S T R A C T

Classical minimization methods, like the steepest descent or quasi-Newton techniques, have been proved to struggle in dealing with optimization problems with a high-dimensional search space or subject to complex nonlinear constraints. In the last decade, the interest on metaheuristic nature-inspired algorithms has been growing steadily, due to their flexibility and effectiveness. In this paper we present EmiR, a package for R which implements several metaheuristic algorithms for optimization problems. Unlike other available tools, EmiR can be used not only for unconstrained problems, but also for problems subjected to inequality constraints and for integer or mixed-integer problems. Main features of EmiR, its usage and the comparison with other available tools are presented.

Yet another library?

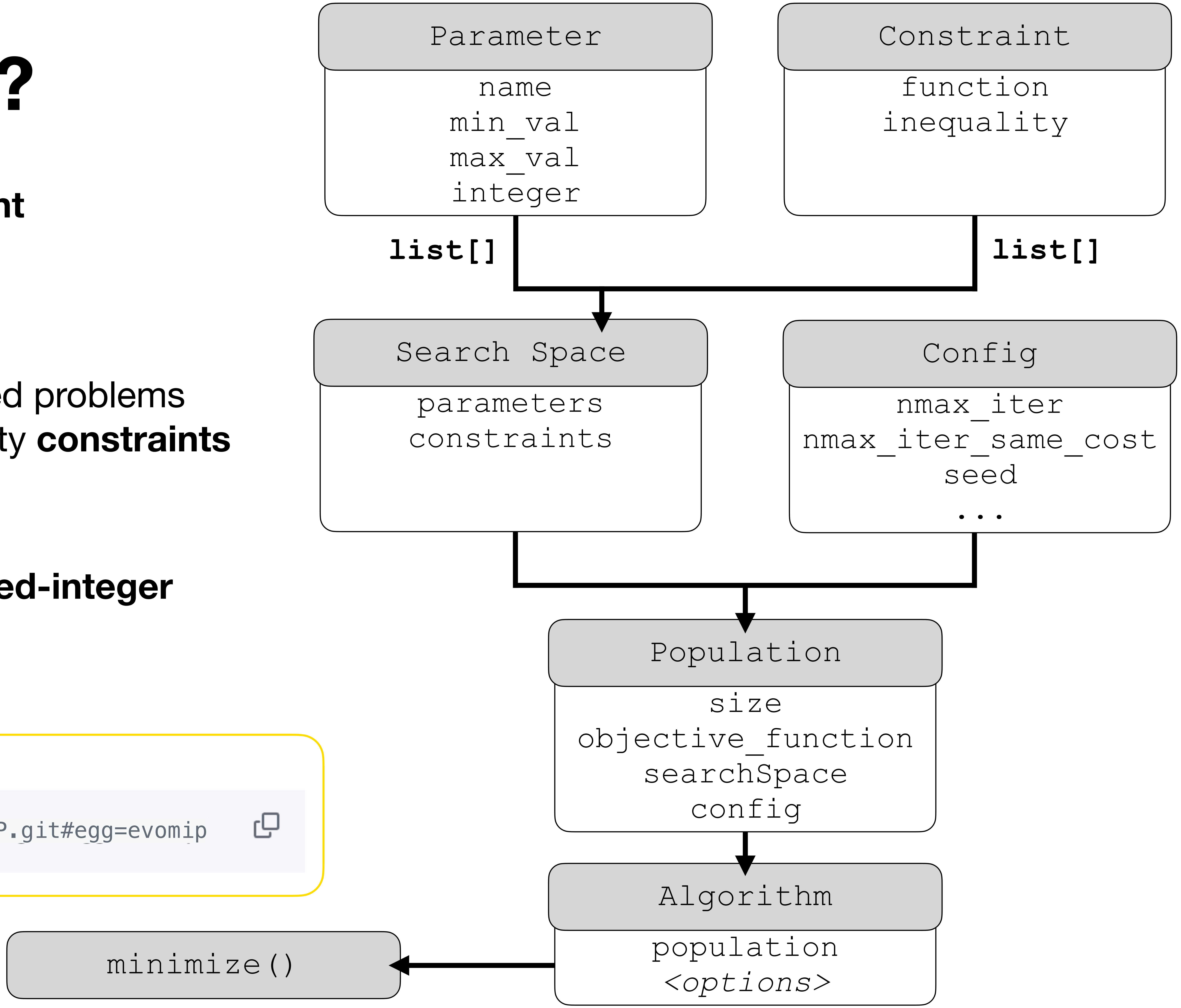
- EvoMiP, just like EmiR, offers an **efficient** implementation of provided algorithms

- It can be used not only for unconstrained problems but also problems subjected to inequality **constraints**

- It can also be used for **integer** and **mixed-integer** problems

Installation

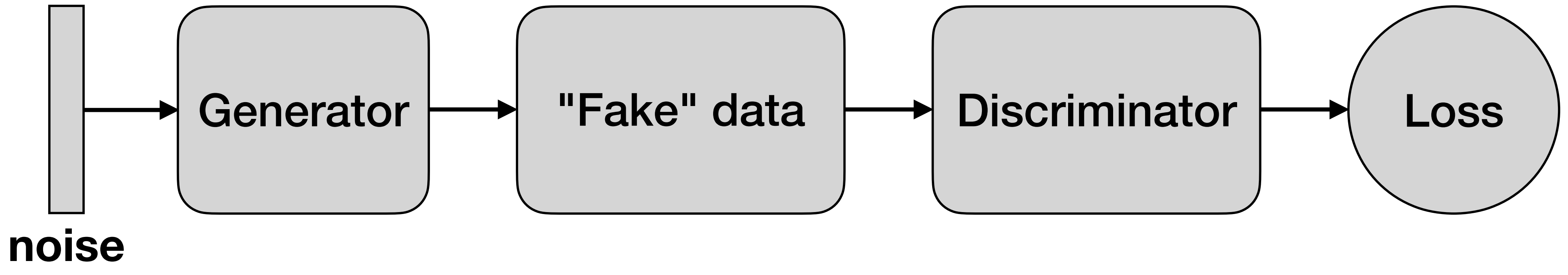
```
pip3 install git+https://github.com/dr4kan/EvoMiP.git#egg=evomip
```



Notebook

An "easy" example...

- Assume you want to train an ANN to generate three integer numbers a, b, c such as:
    - $a \geq 0$
    - $b \leq 0$
    - $c = 0$
- One possible approach could be to use a GAN



    - We need to generate a sample of "good" (a, b, c) vectors
    - This approach is usually slow
- Actually, we don't need the discriminator as long as we can use a proper loss function...
- ...but it won't work with gradient-based approaches...

```
def objective_function(self, opt_par):
    self.update_model_with_parameters(opt_par)
    sum = 0.
    y_pred = self.model.predict(self.latent_points)
    for i in range(0, len(y_pred)):
        a = int(y_pred[i,0])
        b = int(y_pred[i,1])
        c = int(y_pred[i,2])
        if (a < 0):
            sum += abs(a)
        if (b > 0):
            sum += b
        sum += abs(c)
    return sum
```

Notebook