COMP7630 – Web Intelligence and its Applications

# Evolutionary Algorithms (for continuous optimization)

Valentino Santucci

(valentino.santucci@unistrapg.it)

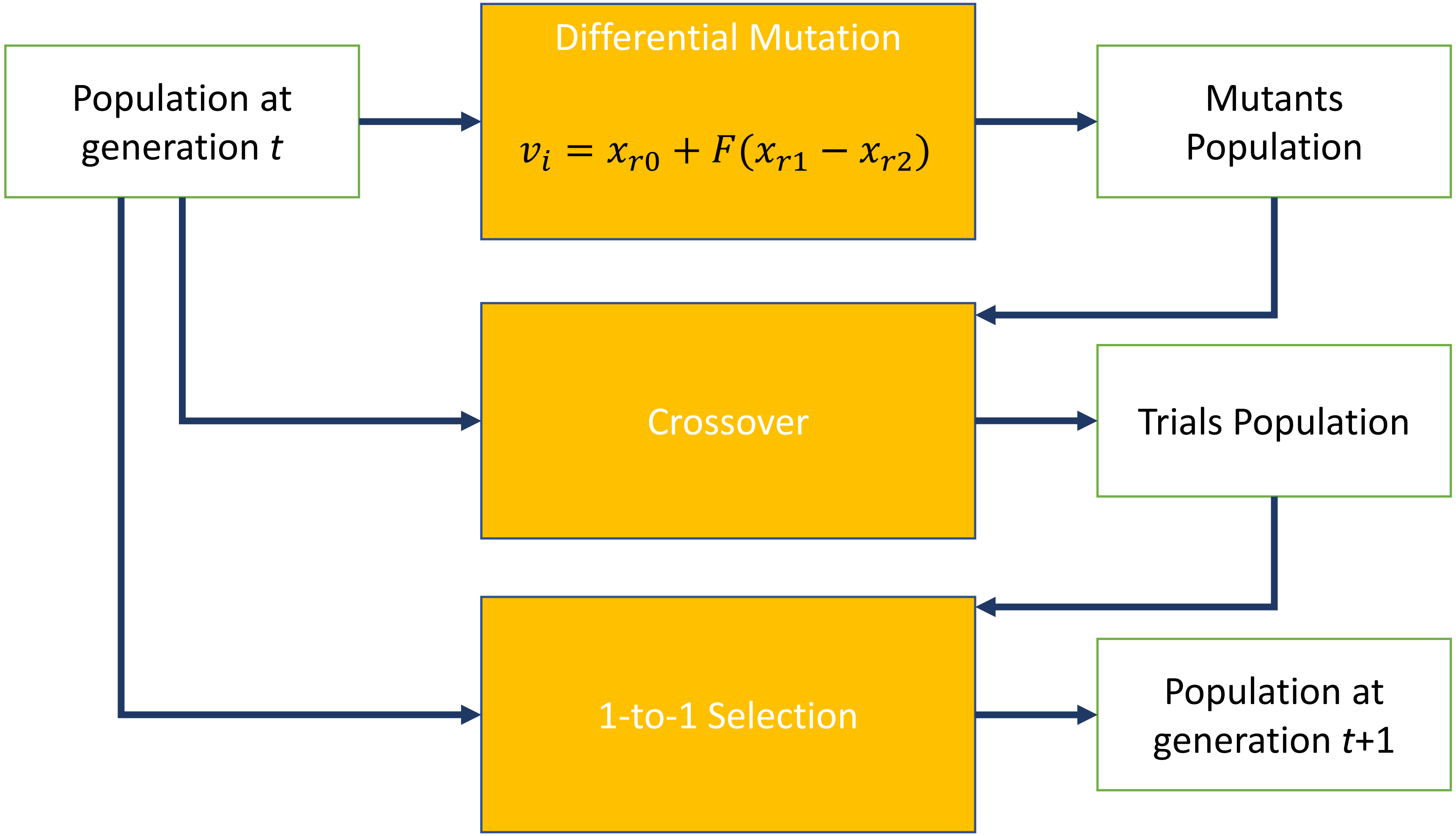
# Outline

- Differential Evolution
- Particle Swarm Optimization
- Nevergrad Python library

# Differential Evolution (DE)

- DE is one of the best optimizers for continuous problems
- Solutions are represented as vectors of real numbers
- 3 genetic operators: differential mutation, crossover, selection
- DE key component is the differential mutation which allows to automatically adjust the balance between exploitation and exploration in the course of the evolution
- (Hyper-)Parameters:
  - $N$ = size of the population
  - $F$ = scale factor >0, but usually in (0,1]
  - $CR$ = crossover probability in [0,1]

# Workflow of DE



# Pseudo-code of DE

1. Randomly initialize a population of $N$ solutions $\{x_1, \dots, x_N\}$
2. While ( termination criterion is not verified )

   a. For $i = 1, \dots, N$
      - Generate a mutant $\boldsymbol{v_i = x_{r_0} + F \cdot (x_{r_1} - x_{r_2})}$

   b. For $i = 1, \dots, N$
      - Generate a trial $u_i = crossover(x_i, v_i)$
      - The previously seen uniform crossover may also work with real vectors

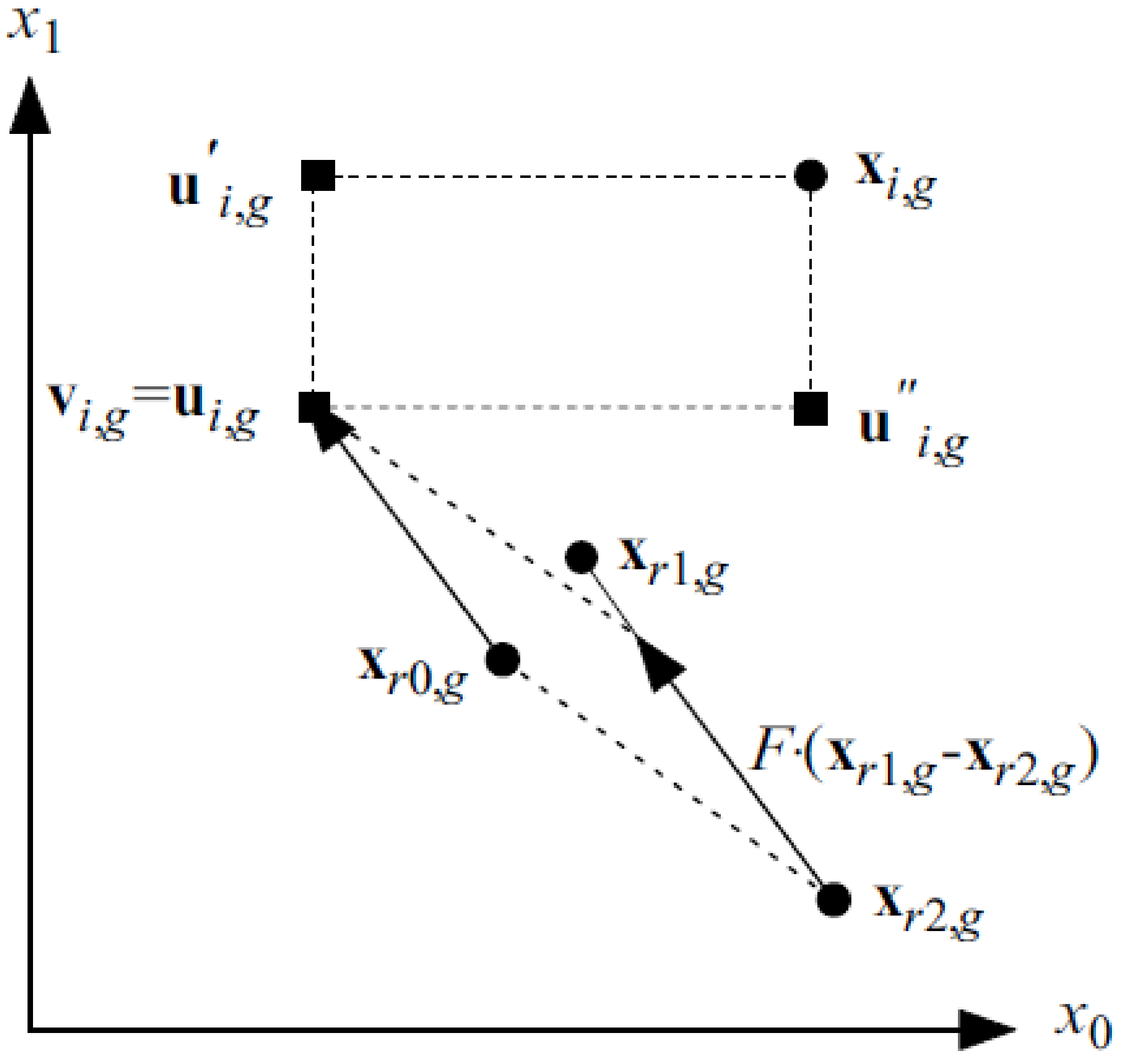
   c. For $i = 1, \dots, N$
      - Evaluate $f(u_i)$
      - Replace $x_i$ with $u_i$ if it is better

Differential Mutation

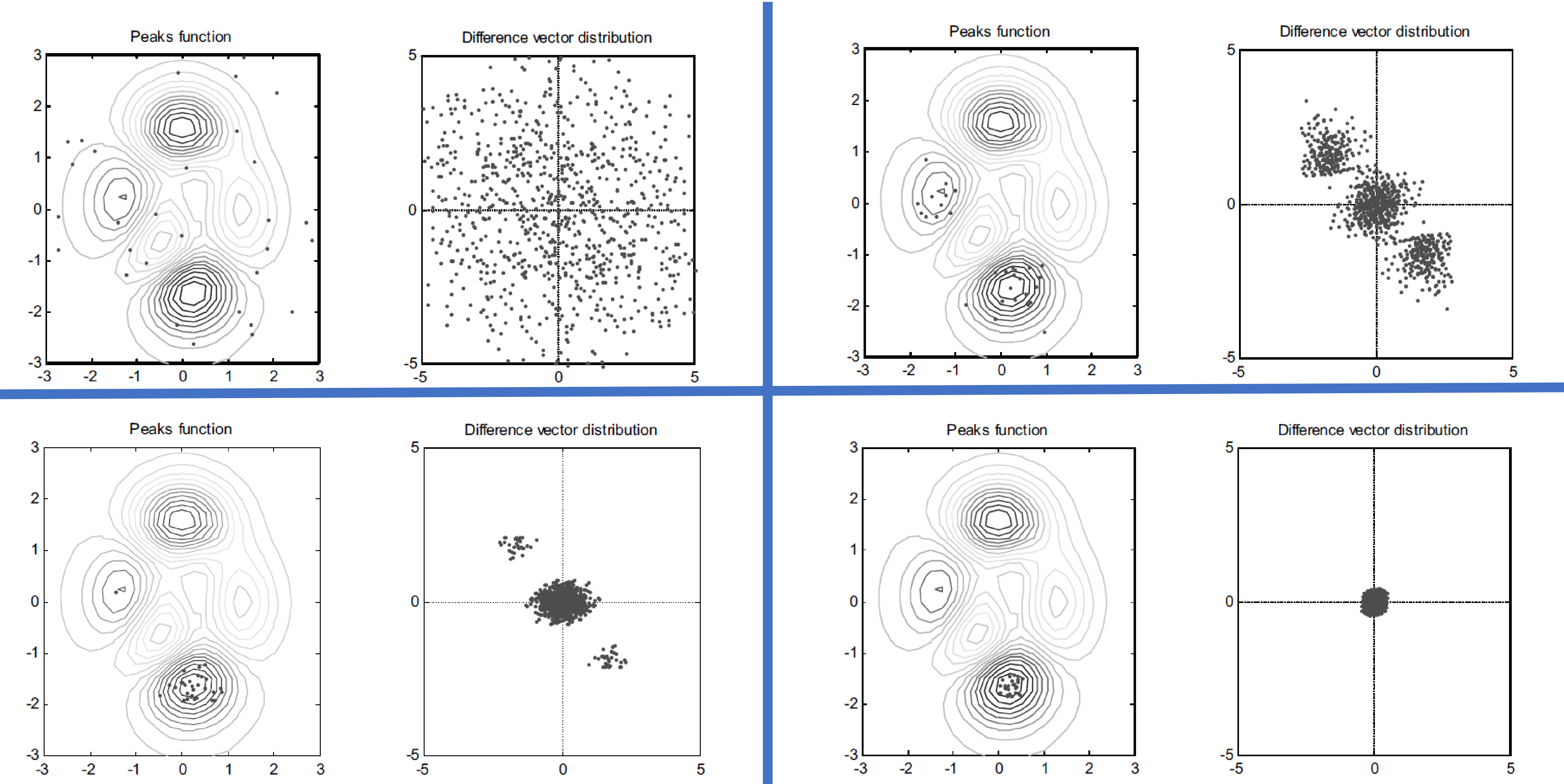
Crossover

Selection

# Differential Evolution



# DE Dynamics (population vs differences)



# Outline

- Differential Evolution
- Particle Swarm Optimization
- Nevergrad Python library

# From Competition to Cooperation: Swarm intelligence

- Swarm intelligence deals with systems composed of many individuals that coordinate using decentralized control and self-organization
- In particular, it focuses on the collective behaviors that emerges from the local interactions of the individuals with each other and with their environment and without the presence of a coordinator.
- Examples:
  - Schools of fish
  - Flocks of birds
  - Colonies of ants and termites
  - …

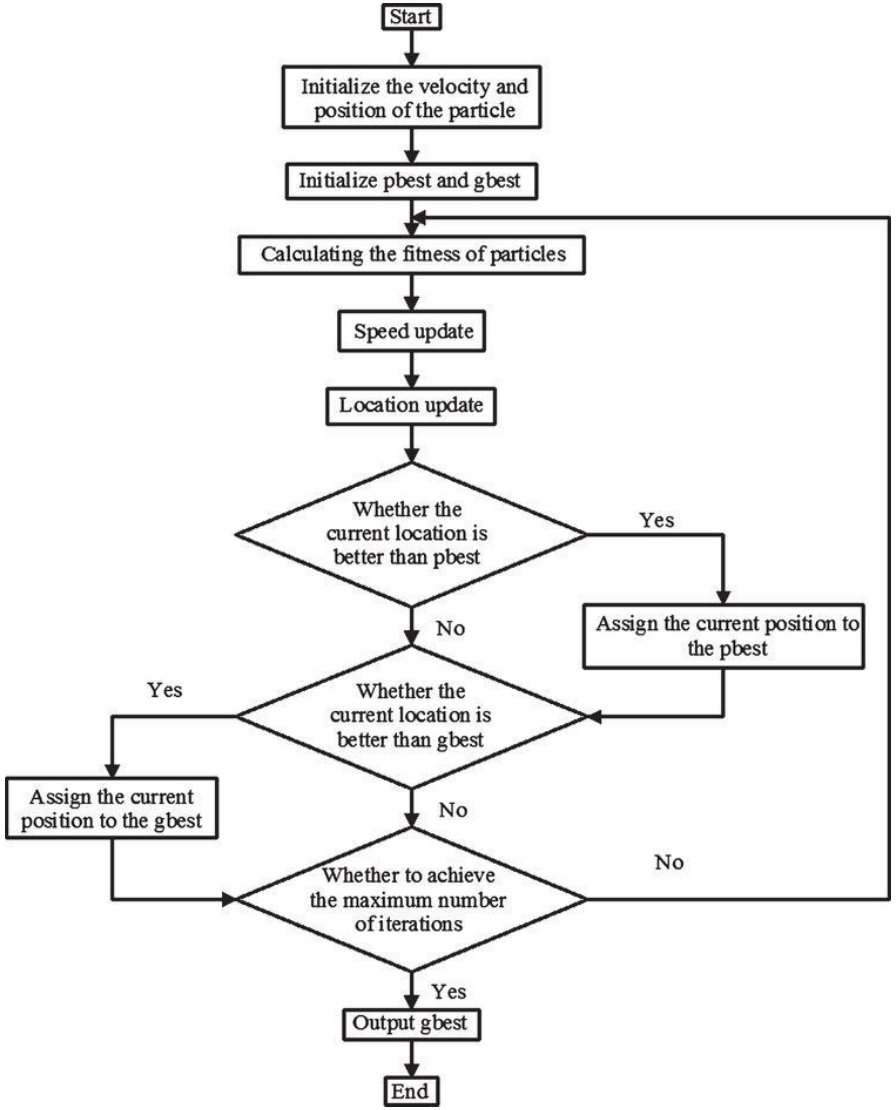
# Particle Swarm optimization

- PSO maintains a population of particles connected in a neighborhood topology.
- Each particle is composed by:
  - $x_i \in \mathbb{R}^n$ => current position
  - $v_i \in \mathbb{R}^n$ => velocity
  - $p_i \in \mathbb{R}^n$ => personal best position
  - $g_i \in \mathbb{R}^n$ => neighborhood best position (eg: global best)
- Particles are arranged in a neighborhood (es: global or ring)

# Particle Swarm optimization

- Randomly initialize particles
- While (termination criterion not met)
  - For each particle:
    - Update particle velocity
    - Update particle position
    - Evaluate particle position
    - Update particle personal best
  - For each particle:
    - Update particle neighborhood best

# Workflow of PSO

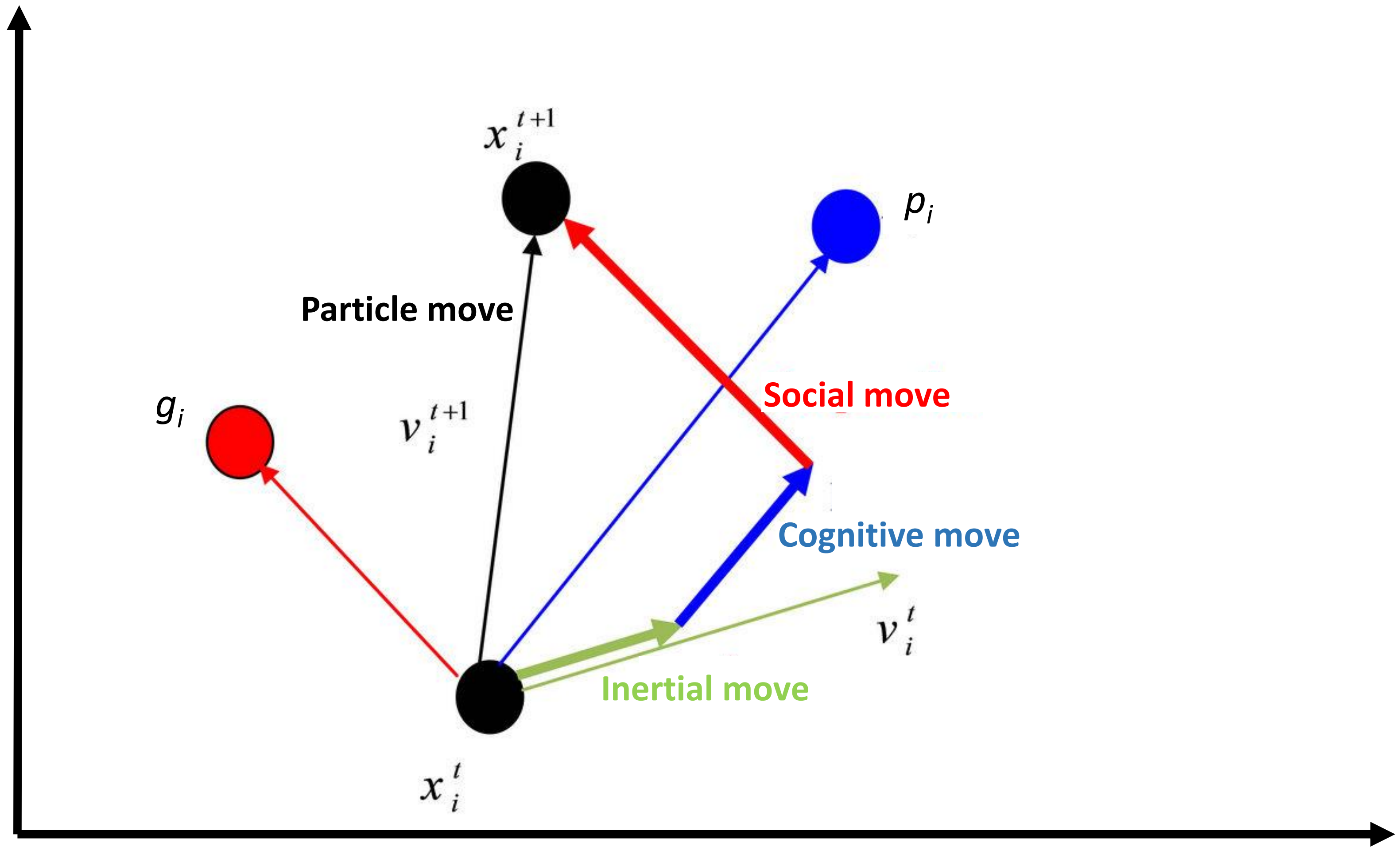


# Particle swarm optimization

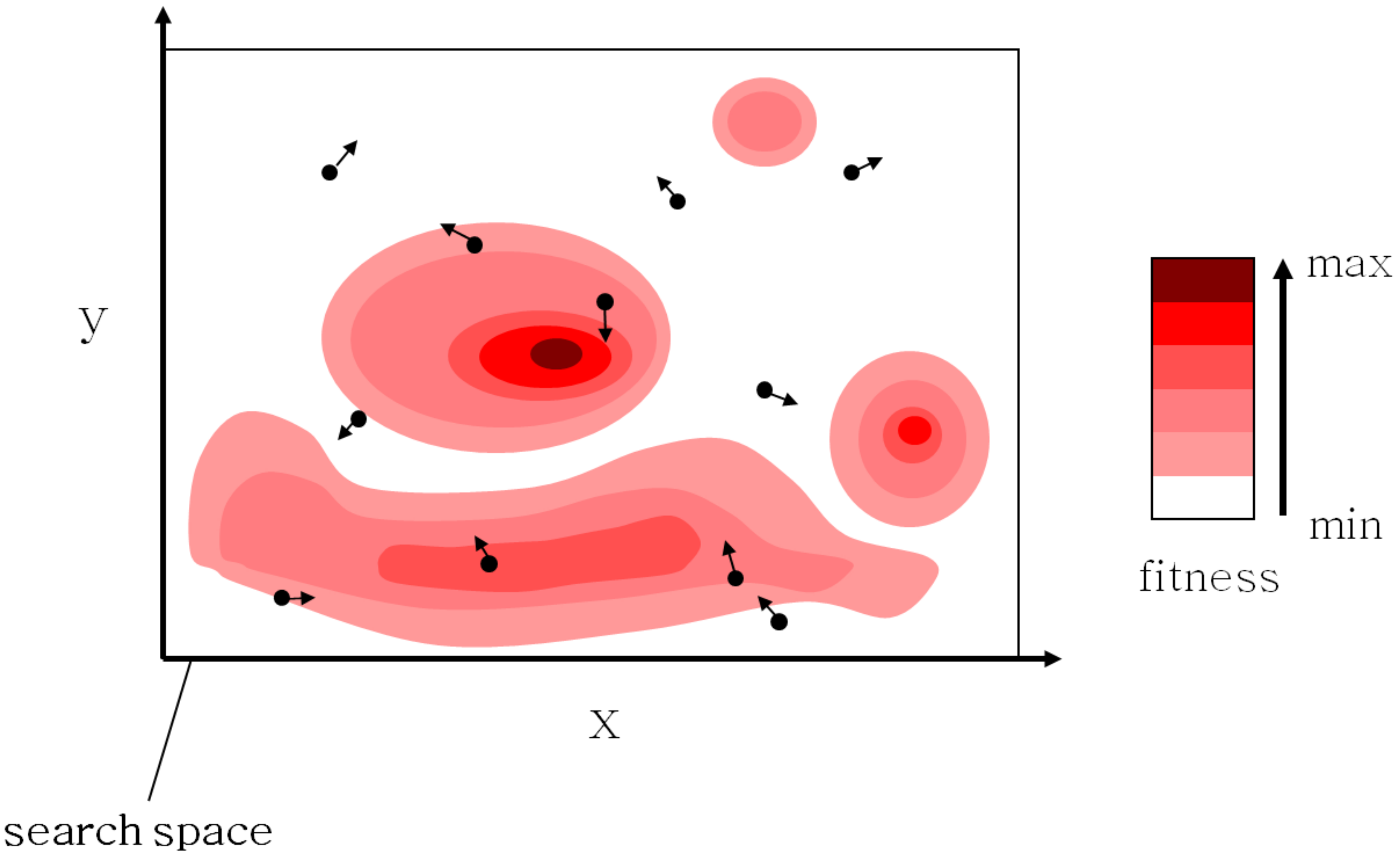
- Position and velocity update:
  - $v_i \leftarrow wv_i + c_1 r_1 (p_i - x_i) + c_2 r_2 (g_i - x_i)$ => velocity update rule
  - $x_i \leftarrow x_i + v_i$ => position update rule

- PSO parameters:
  - $w \in \mathbb{R}^+$ => inertial coefficient
  - $c_1 \in \mathbb{R}^+$ => cognitive coefficient
  - $c_2 \in \mathbb{R}^+$ => social coefficient

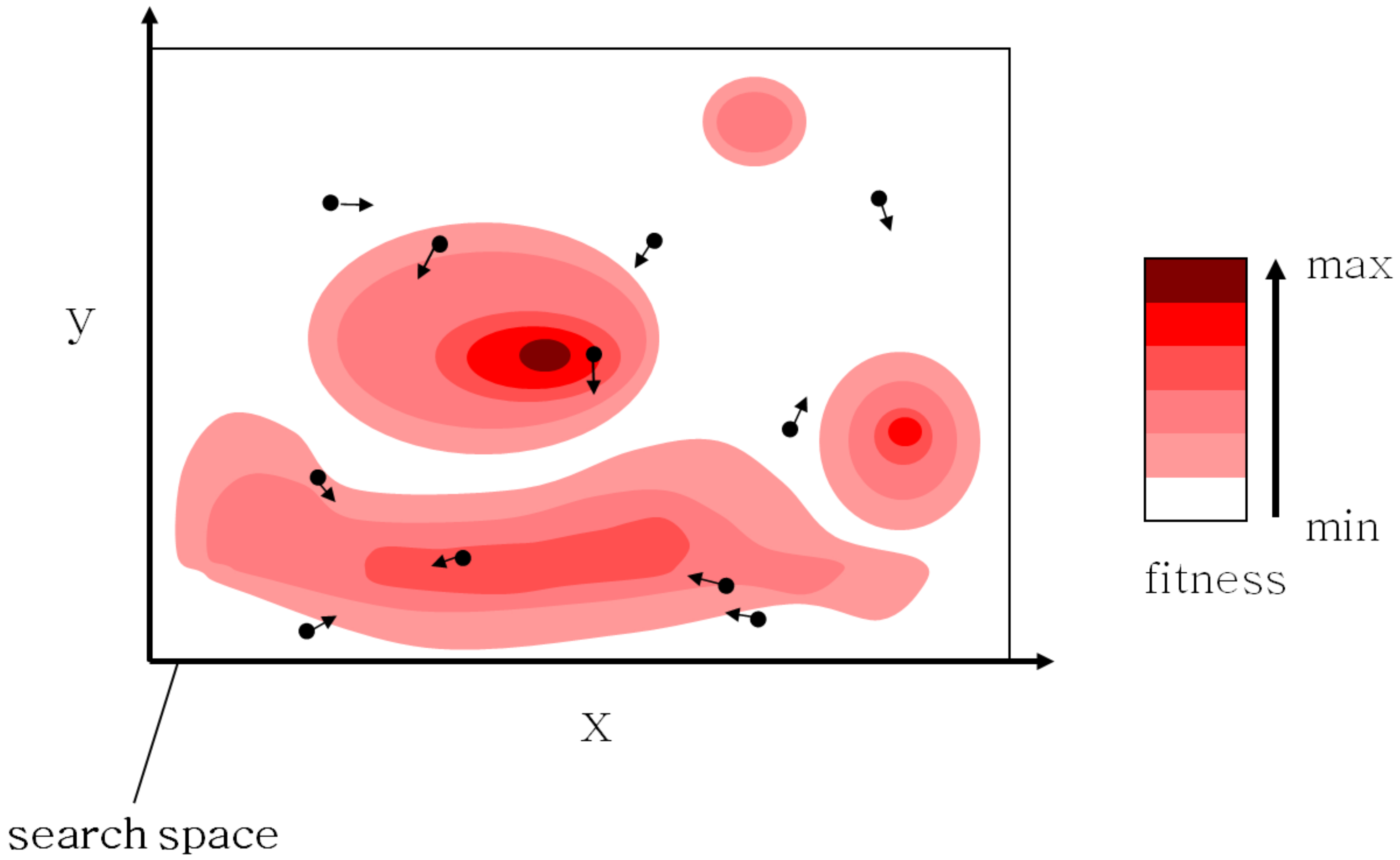
# Movement of a PSO particle



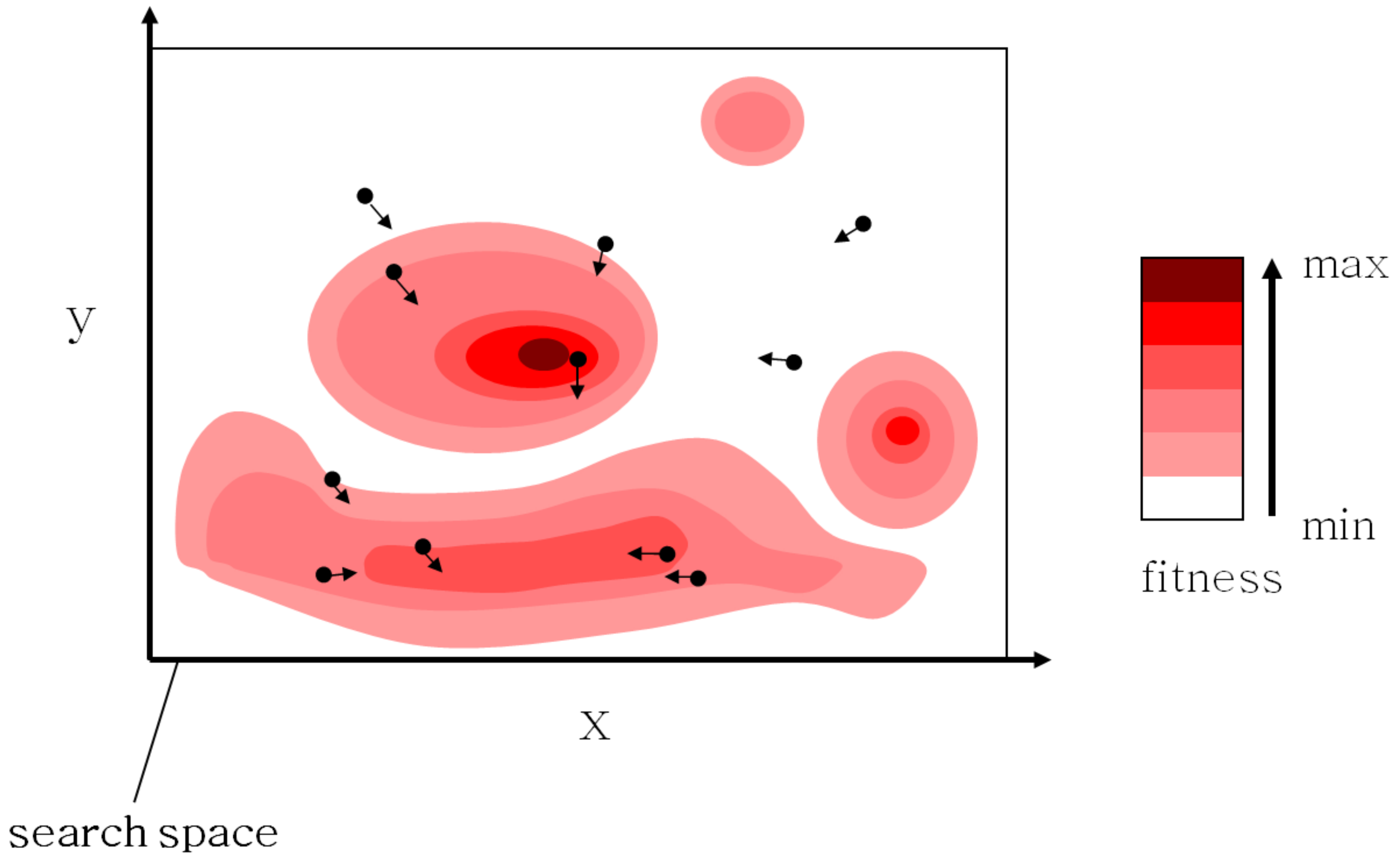
# PSO Dynamics



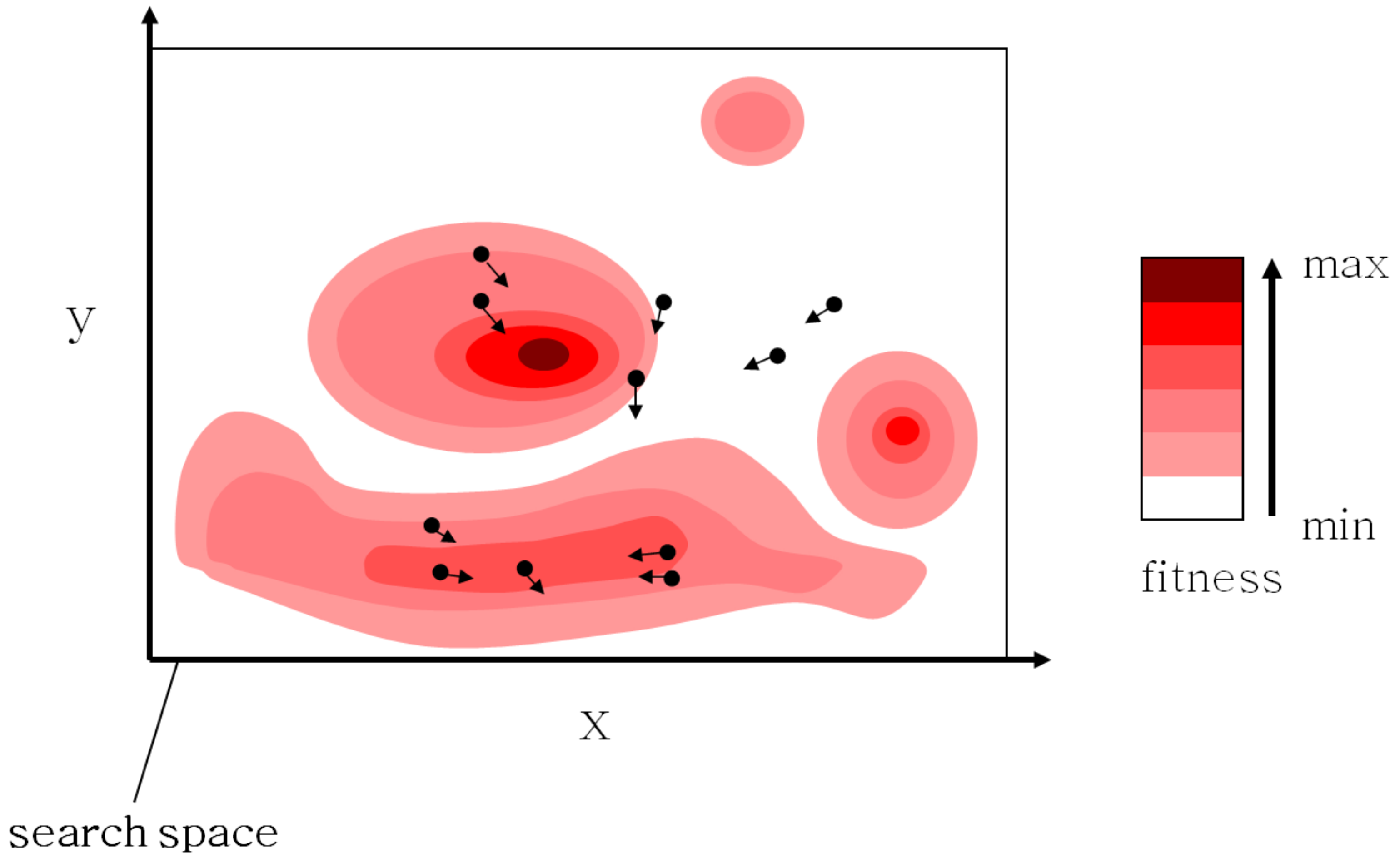
# PSO Dynamics



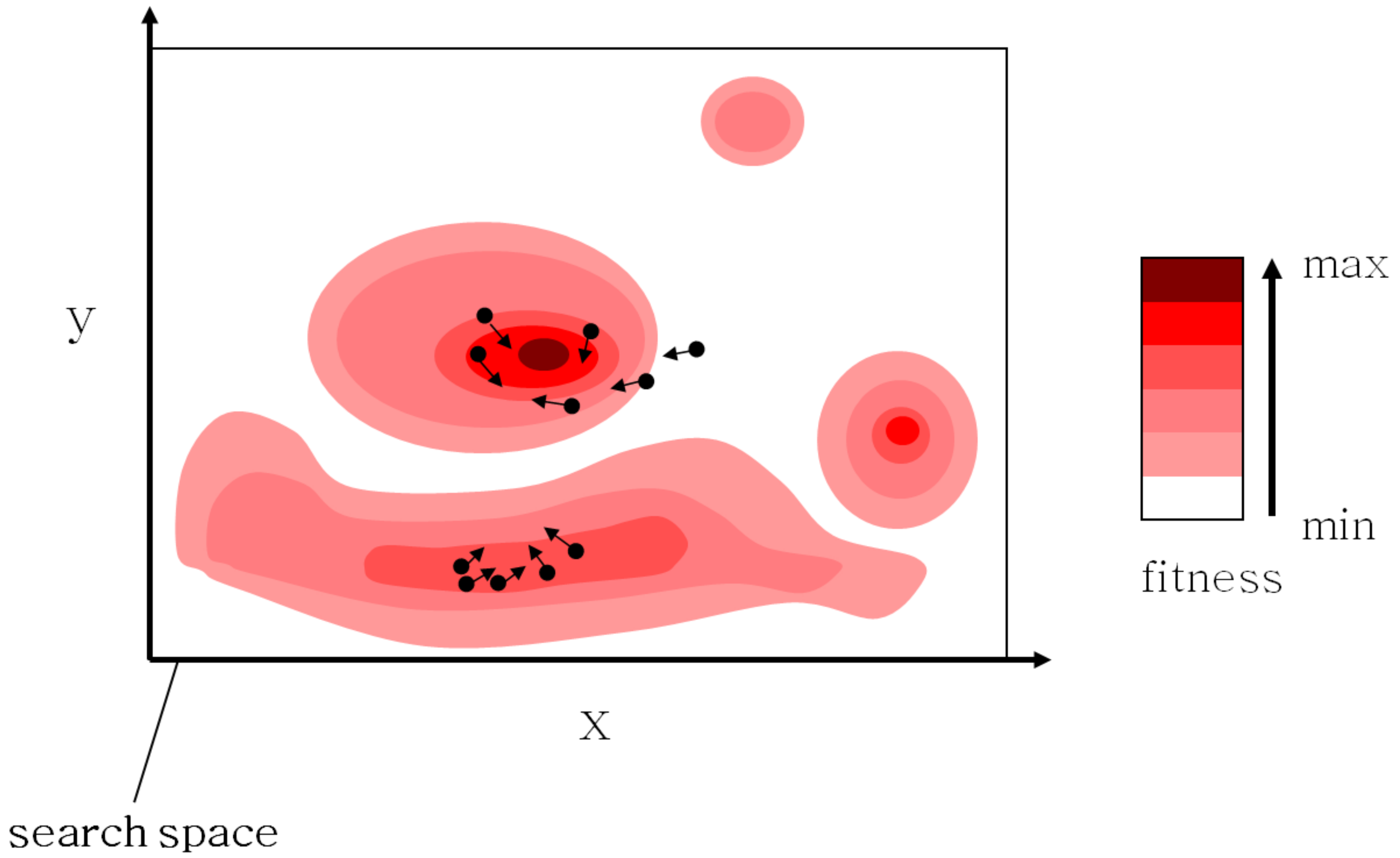
# PSO Dynamics



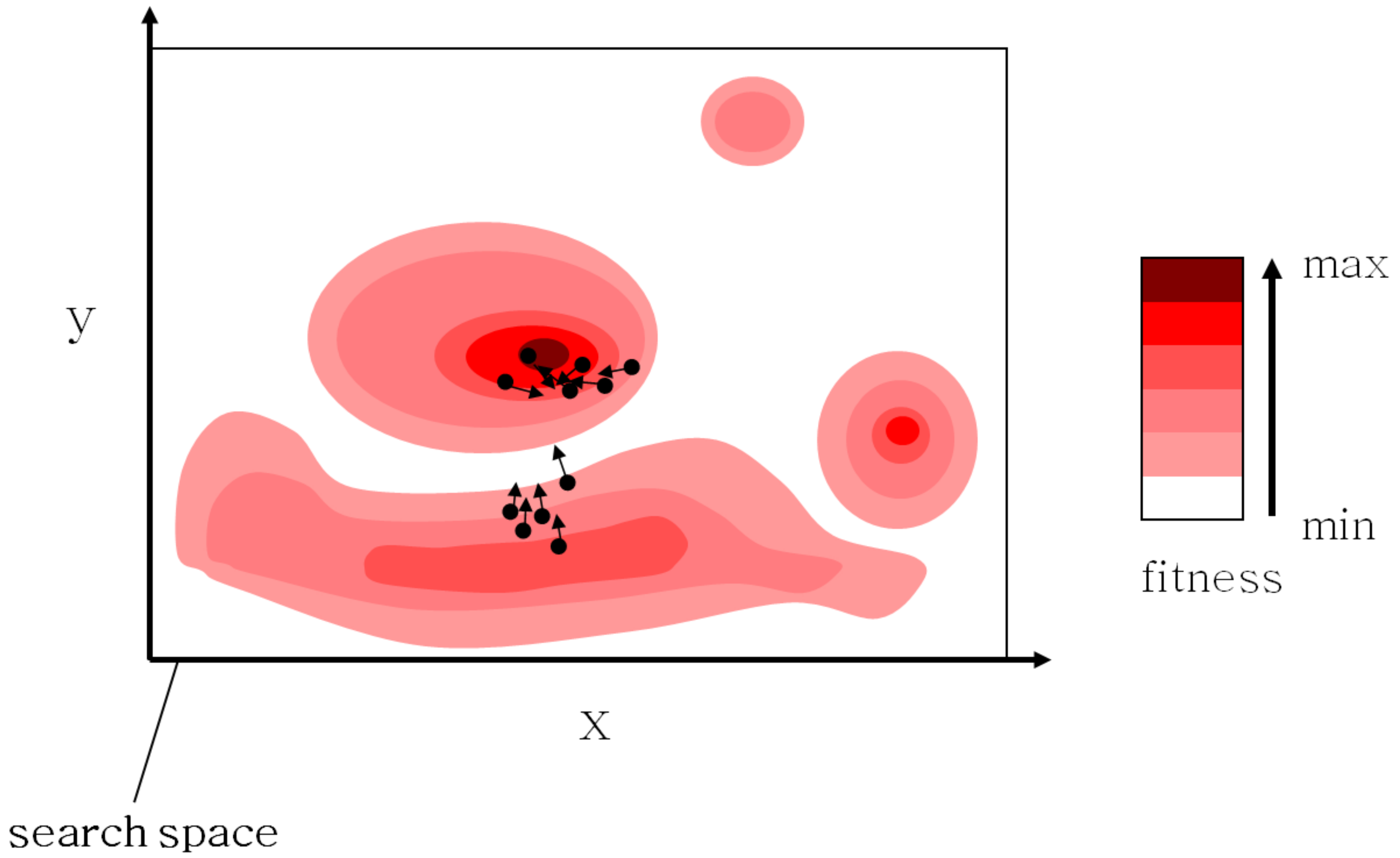
# PSO Dynamics



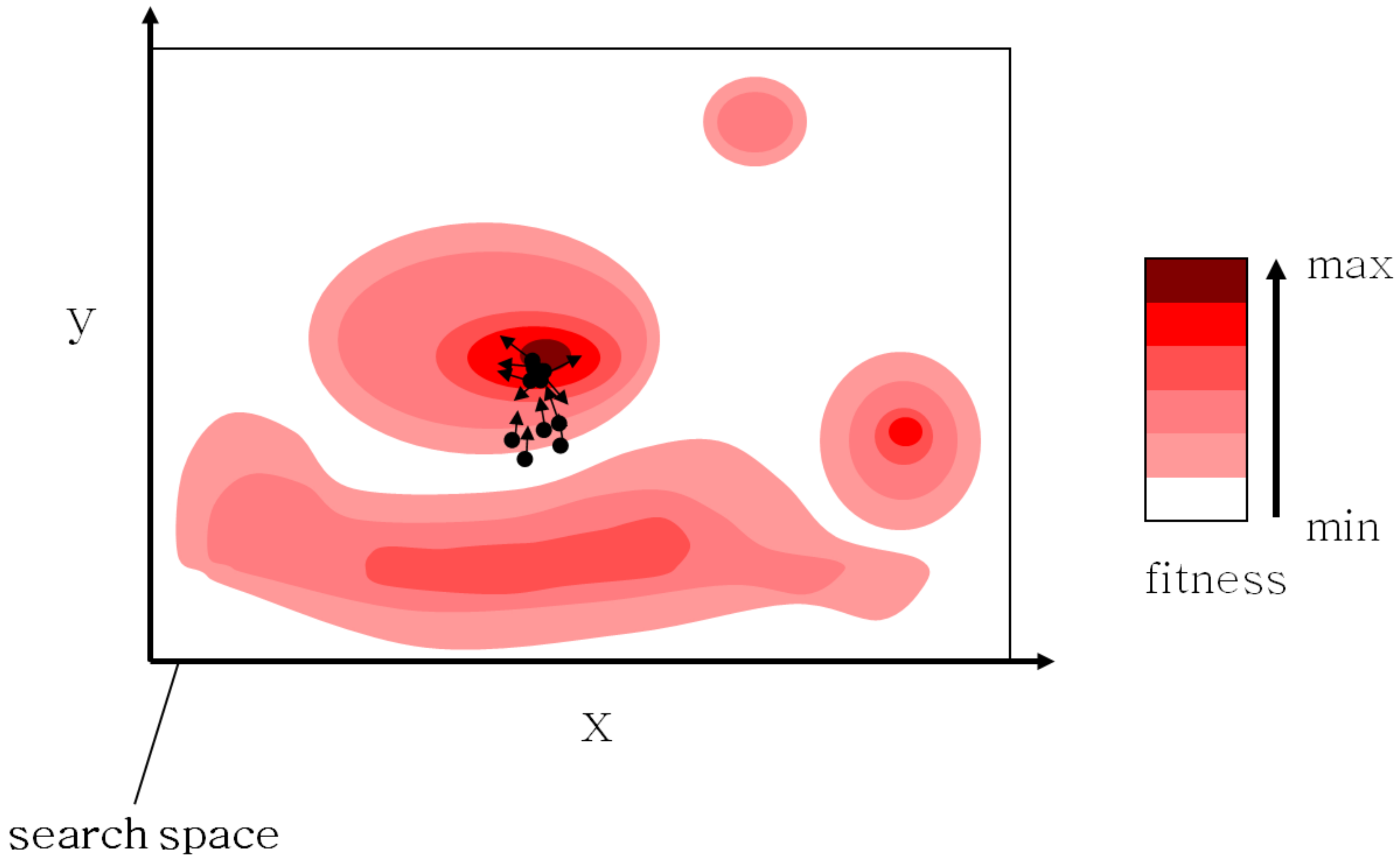
# PSO Dynamics



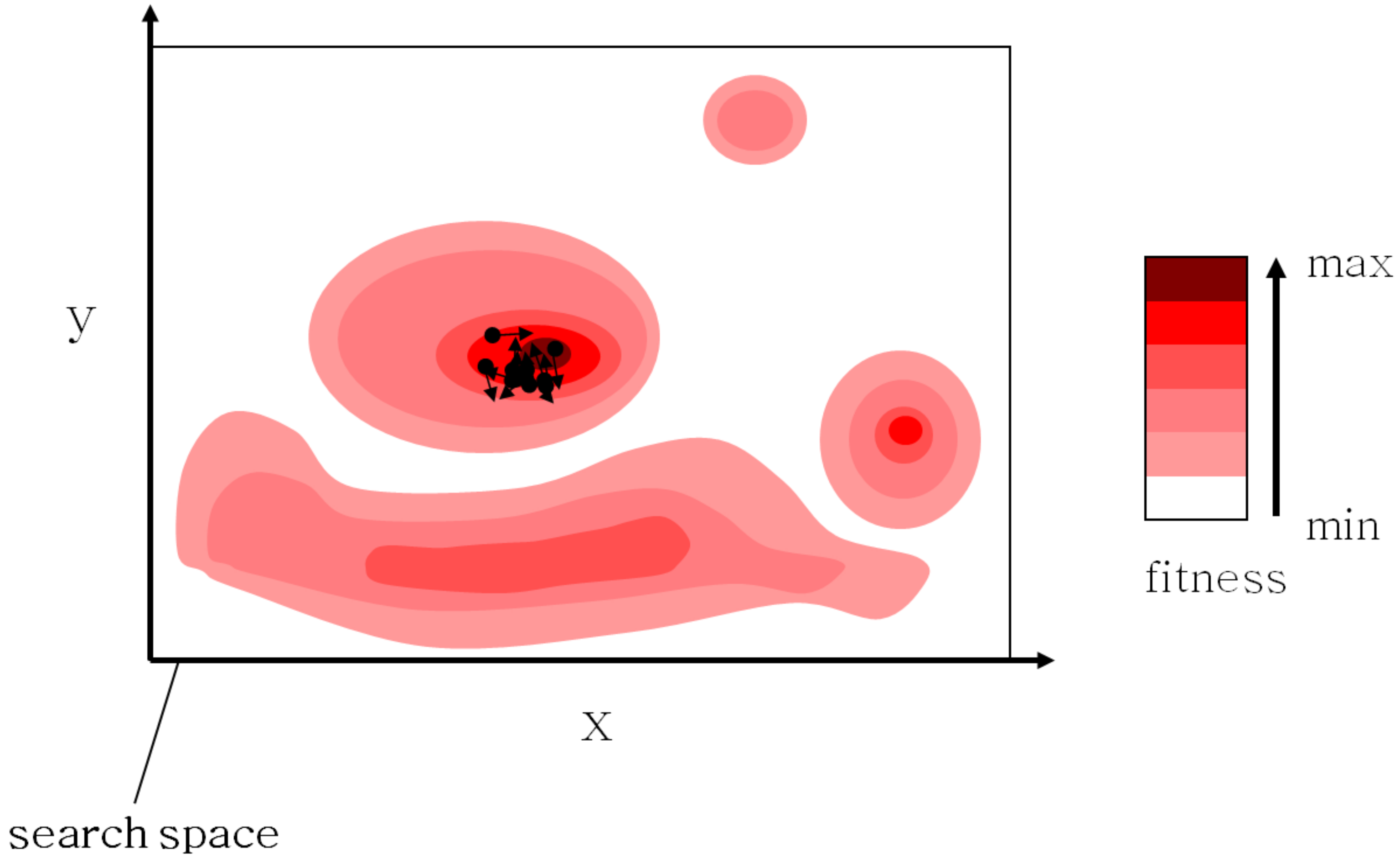
# PSO dynamics



# PSO dynamics



# PSO Dynamics



# Outline

- Differential Evolution
- Particle Swarm Optimization
- Nevergrad Python library

# Nevergrad

- Nevergrad is a Python library containing a collection of evolutionary algorithms

- Install it with: `pip install nevergrad`

# DE with Nevergrad

```
In [43]: import nevergrad as ng

In [44]: import numpy as np

In [45]: #define a simlpe objective function, just as an example

In [46]: def sphere_objective_function(x):
    ...:     return np.sum(x**2)
    ...:

In [47]: #optimize the objective function using Differential Evolution

In [48]: optimizer = ng.optimizers.DE( parametrization=30, budget=10_000 )

In [49]: result = optimizer.minimize( sphere_objective_function )

In [50]: #print the objective value of the optimum and the vector values of the optimum

In [51]: result.loss
Out[51]: 0.2598661999997661

In [52]: result.value
Out[52]:
array([ 8.79658218e-02,  3.85043137e-03, -8.80508374e-04, -2.64700478e-01,
        7.76468147e-02,  8.92546030e-02, -9.55805127e-02, -1.42584614e-01,
        1.07019443e-02, -9.15763670e-02,  1.12404646e-01, -1.48903356e-05,
        4.79471462e-02, -8.49231815e-02,  1.39927968e-01,  2.30893046e-02,
        1.47364718e-01,  5.66846932e-02, -4.80954781e-02, -4.66436229e-03,
        1.14261219e-01,  1.82622537e-03, -1.66863517e-01, -5.37990272e-02,
       -1.21875501e-02, -5.11934821e-02, -6.92388772e-02, -9.57985118e-02,
       -9.84461997e-04, -4.75660532e-03])
```

The sphere function is a common and simple benchmark in the field

`Parametrization` is the dimensionality of solutions' vectors, while `budget` is the number of evaluations allowed

The `loss` is the term used for the best objective value in Nevergrad

This is the best vector found

# PSO with Nevergrad

```
In [54]: import nevergrad as ng

In [55]: import numpy as np

In [56]: #define a simlpe objective function, just as an example

In [57]: def sphere_objective_function(x):
    ...:     return np.sum(x**2)
    ...:

In [58]: #optimize the objective function using Particle Swarm Optimization

In [59]: optimizer = ng.optimizers.RealSpacePSO( parametrization=30, budget=10_000 )

In [60]: result = optimizer.minimize( sphere_objective_function )

In [61]: #print the objective value of the optimum and the vector values of the optimum

In [62]: result.loss
Out[62]: 1.5728934539310212e-07

In [63]: result.value
Out[63]:
array([-5.99897807e-07,  6.97273841e-05, -8.75388255e-05,  4.40332547e-05,
        5.73672210e-05, -1.54354200e-04, -6.99193070e-05, -8.19045940e-06,
        8.08336880e-05, -1.66535426e-04,  3.54851545e-05,  2.00195656e-05,
       -9.71032794e-06,  4.15782172e-05, -3.96327177e-05, -3.89925050e-05,
       -3.45366166e-05,  1.06446165e-04, -1.42194206e-04,  3.72699415e-05,
       -1.48941055e-05, -5.02374742e-05,  4.54592393e-05, -7.58946442e-05,
       -7.79804819e-05, -4.51465661e-05,  5.51560964e-05,  9.87027627e-05,
       -3.89297696e-05,  5.27983658e-05])
```

# References


- Original article about DE: https://www.cp.eng.chula.ac.th/~prabhas//teaching/ec/ec2012/storn_price_de.pdf
- Original article about PSO: http://staff.washington.edu/paymana/swarm/kennedy95-ijcnn.pdf
- Nevergrad Documentation: https://facebookresearch.github.io/nevergrad/