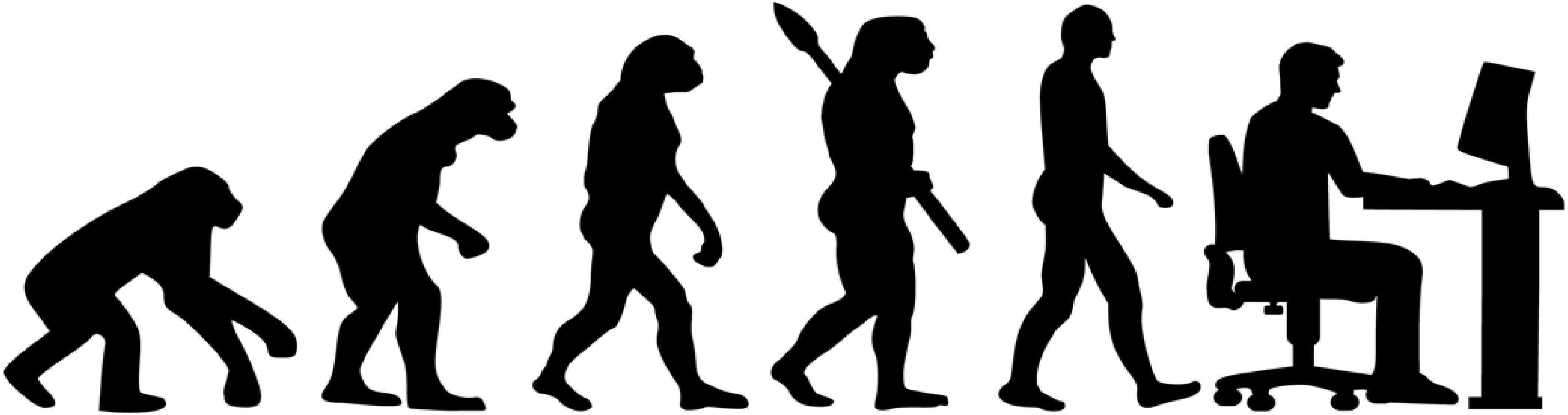
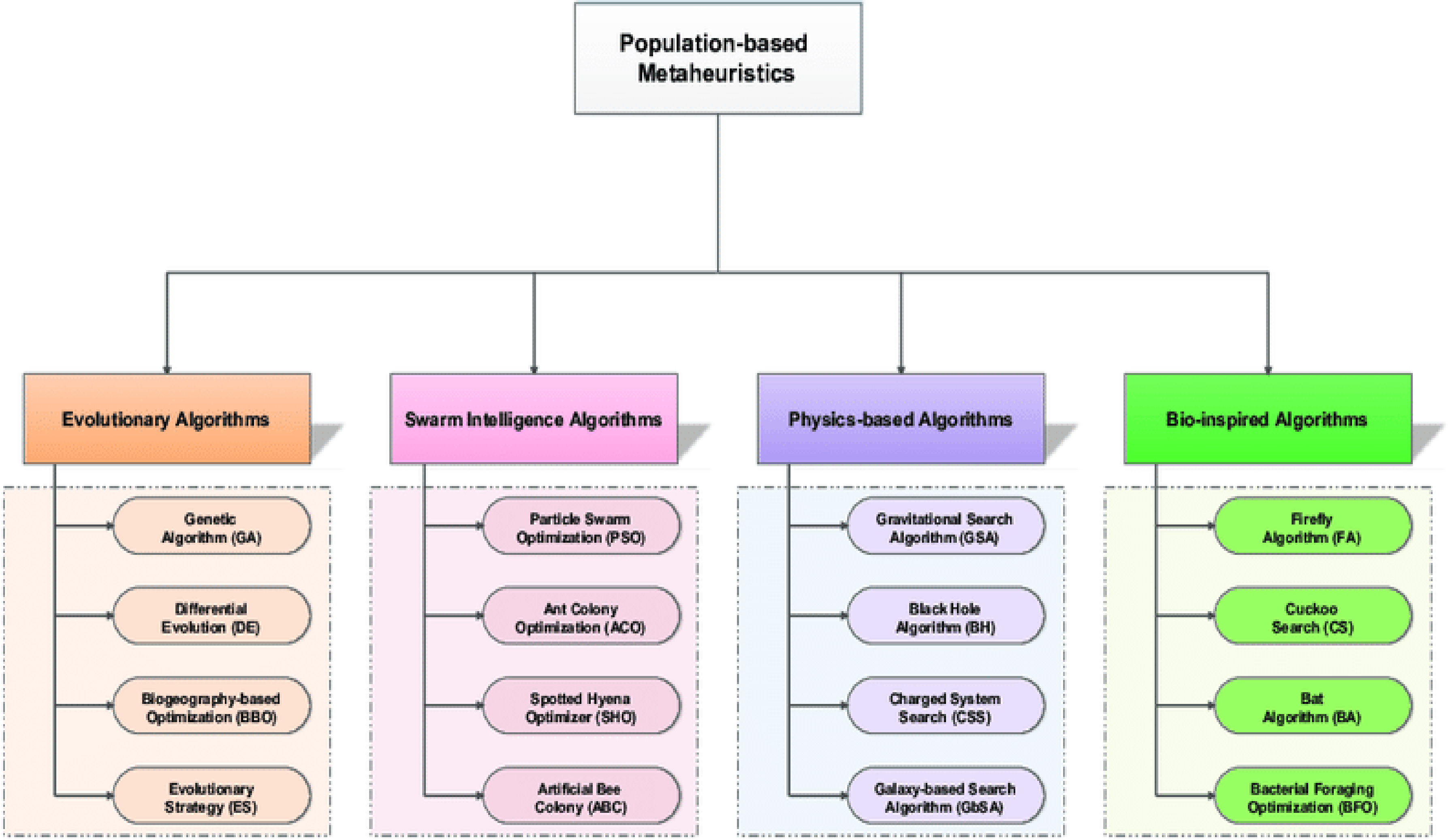
Evolutionary Computation

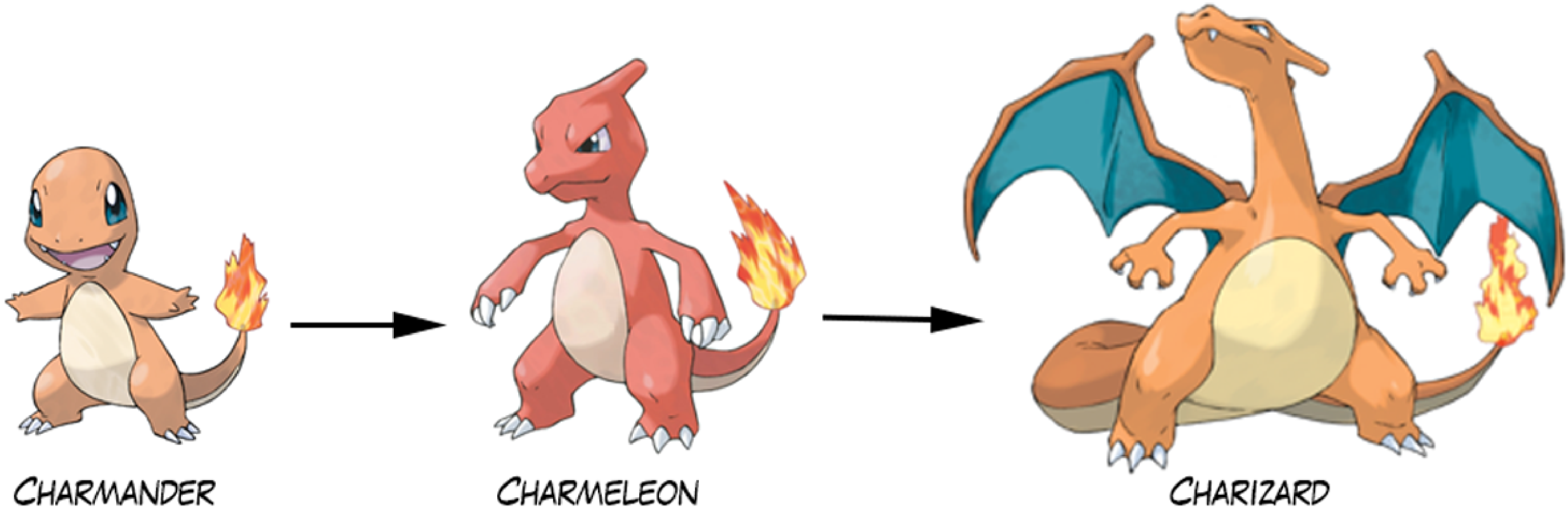


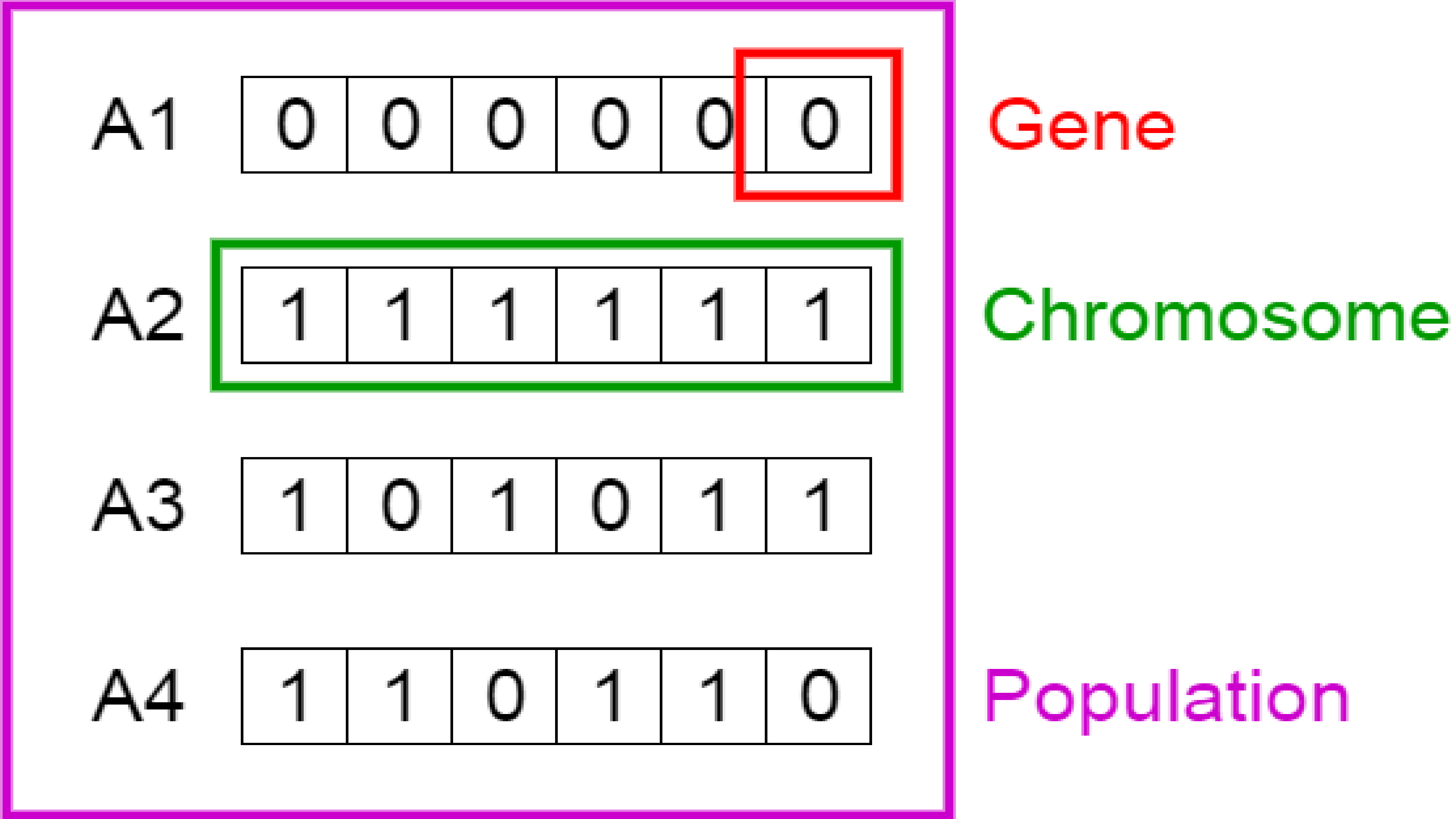
made by Bartosz Paulewicz

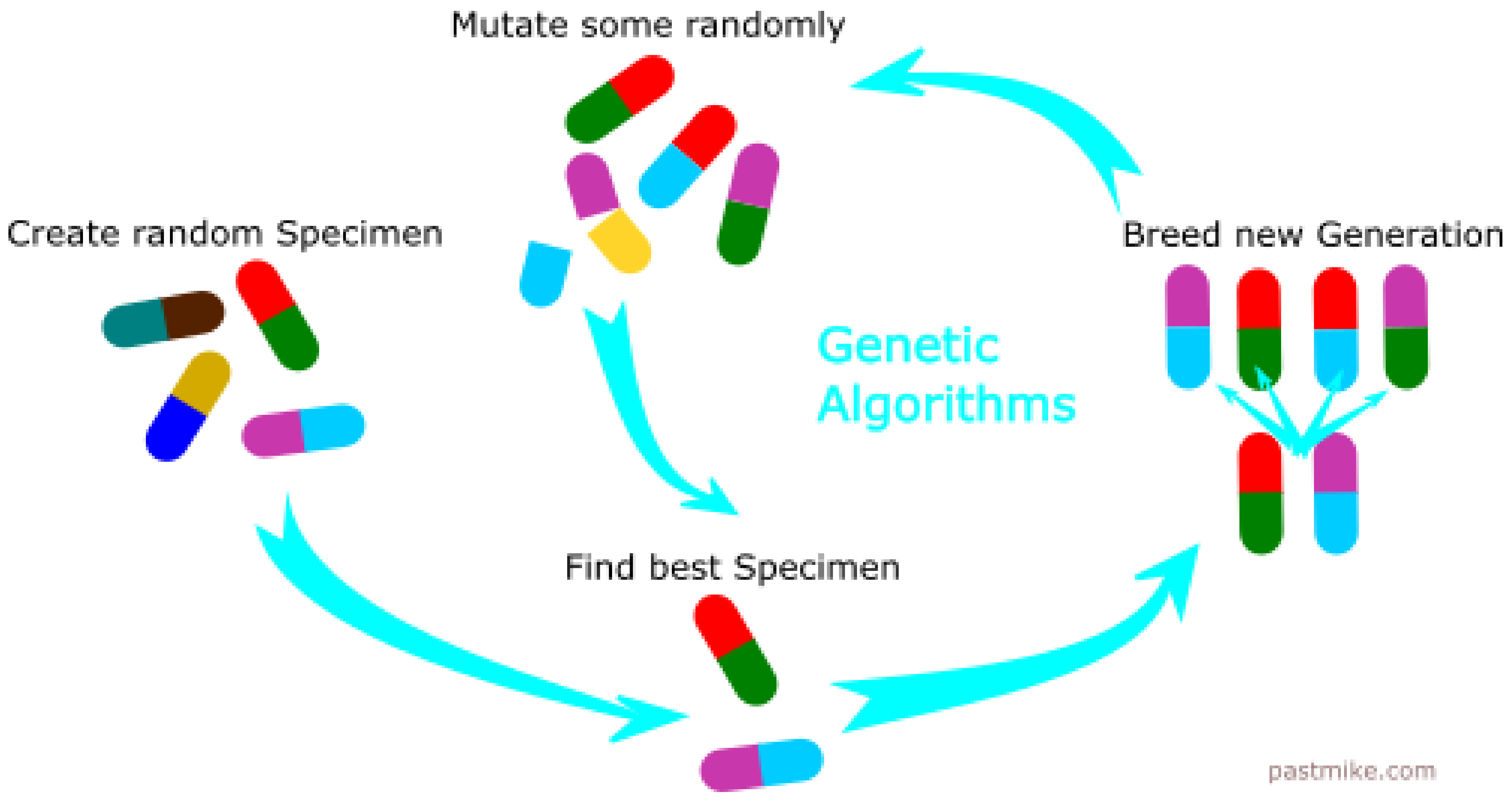

Try
Fail
Try Again
SUCCESS

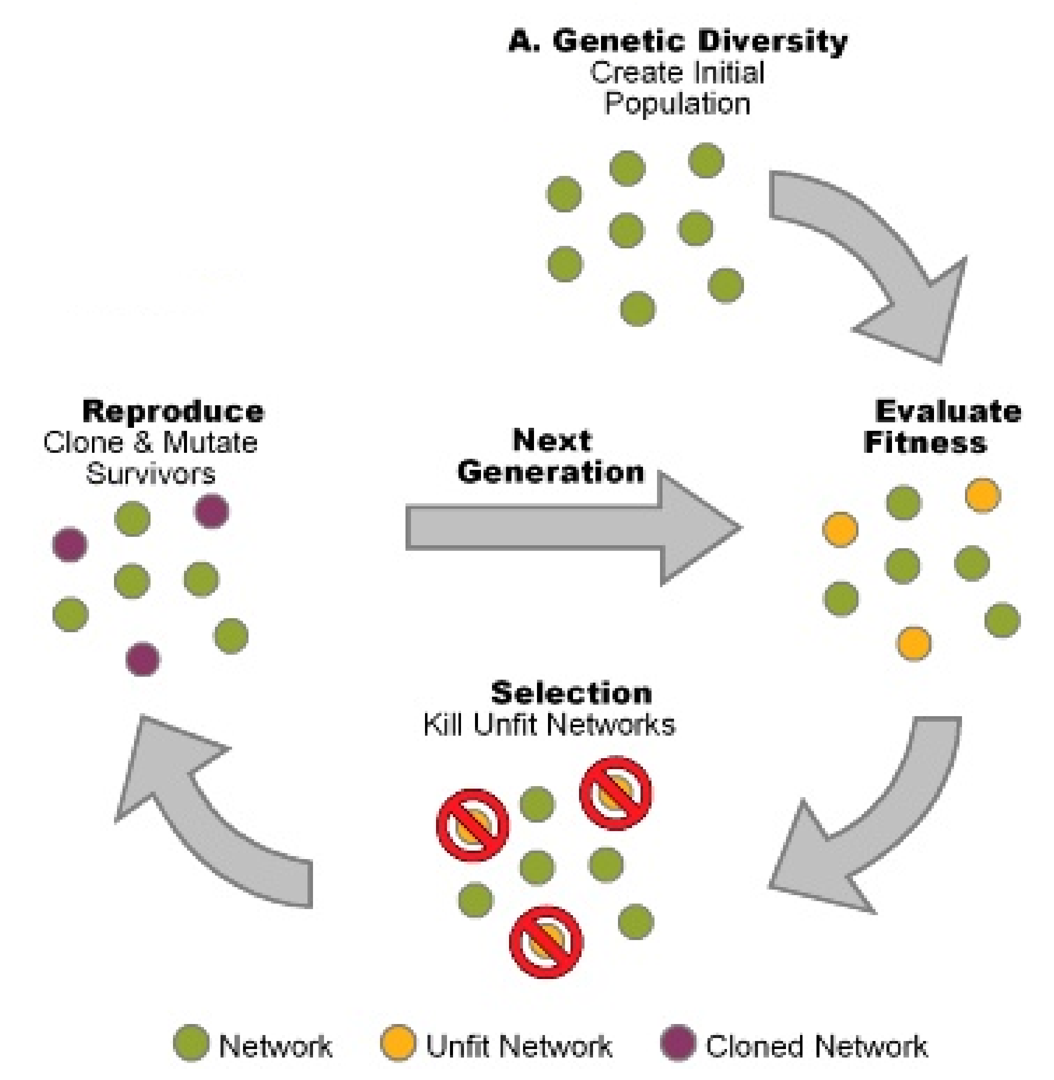


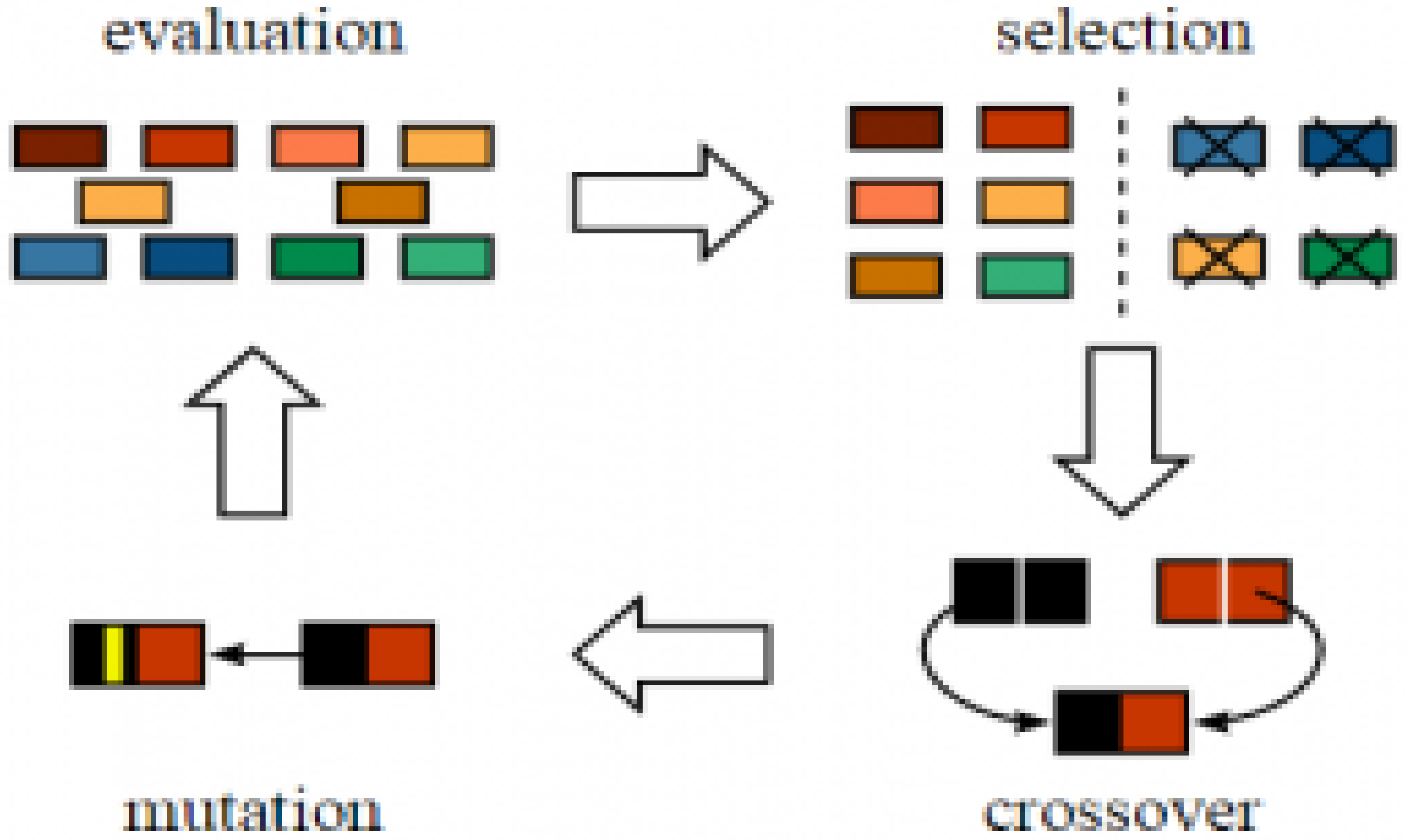
Population-based Metaheuristics
Evolutionary Algorithms
Swarm Intelligence Algorithms
Physics-based Algorithms
Bio-inspired Algorithms
Genetic Algorithm (GA)
Differential Evolution (DE)
Biogeography-based Optimization (BBO)
Evolutionary Strategy (ES)
Particle Swarm Optimization (PSO)
Ant Colony Optimization (ACO)
Spotted Hyena Optimizer (SHO)
Artificial Bee Colony (ABC)
Gravitational Search Algorithm (GSA)
Black Hole Algorithm (BH)
Charged System Search (CSS)
Galaxy-based Search Algorithm (GbSA)
Firefly Algorithm (FA)
Cuckoo Search (CS)
Bat Algorithm (BA)
Bacterial Foraging Optimization (BFO)

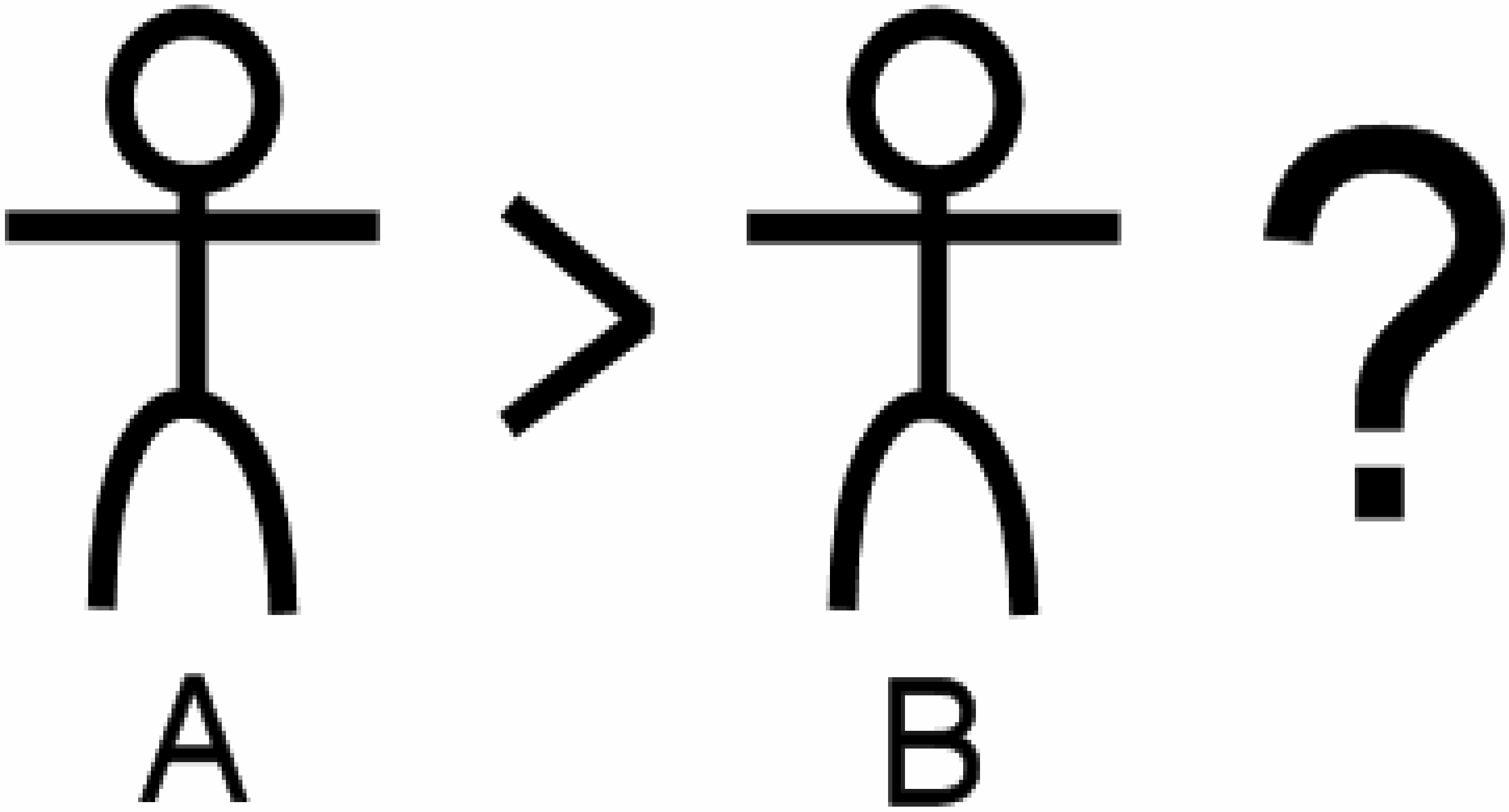

CHARMANDER
CHARMELEON
CHARIZARD


A1 0 0 0 0 0 0
A2 1 1 1 1 1 1
A3 1 0 1 0 1 1
A4 1 1 0 1 1 0
Gene
Chromosome
Population


Mutate some randomly
Create random Specimen
Breed new Generation
Genetic Algorithms
Find best Specimen
pastmike.com


A. Genetic Diversity
Create Initial
Population
Evaluate
Fitness
Selection
Kill Unfit Networks
Reproduce
Clone & Mutate
Survivors
Next
Generation
Network
Unfit Network
Cloned Network


evaluation
selection
crossover
mutation


A
B

Short description of the task

“You’ve just bought a house and discover that the attic is full of wasps' nests. You’ve decided to kill the wasps, before you move into your new home so you visit your local store featuring insecticides but found only 3 "insect-bomb" which have a specific effect range and must be placed very close to the nest to kill the wasps inside. Unfortunately the 3 containers are not enough to kill all the wasps in the attic. Fortunately, luck helps you find a map left by the previous owner, showing the location of the nests as well as the number of wasps that each nest has (using an array of 100x100)”

Nest number	Wasp population	Nest position	
		X axis	Y axis
1	100	25	65
2	200	23	8
3	327	7	13
4	440	95	53
5	450	3	3
6	639	54	56
7	650	67	78
8	678	32	4
9	750	24	76
10	801	66	89
11	945	84	4
12	967	34	23

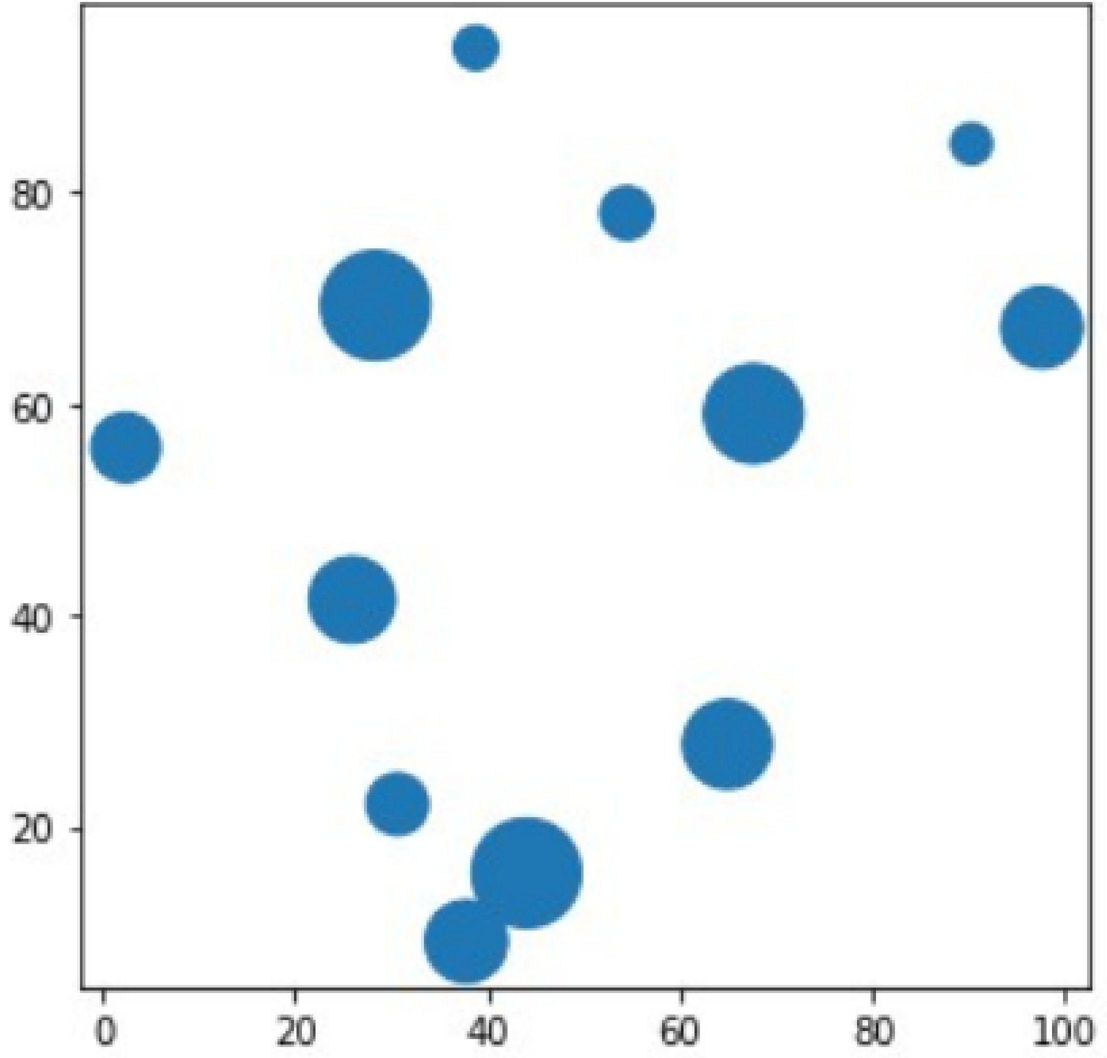
Preparation

```
from numpy import argmax, argmin, argsort, array,
from numpy import concatenate, empty, floor, sqrt
from numpy.random import seed, uniform, normal
from numpy.linalg import norm
seed(0)
```

Generating the initial population (first generation)

```
nBombs = 3
populationSize = 100

population = uniform(low=0, high=100, size=(populationSize, nBombs, 2))
```



80
60
40
20
0
20
40
60
80
100

Evaluating the fitness of each individual in population.

$$F(X_1, Y_1, \ldots, X_N, Y_N) = \sum_{j=0}^{M} n_{Nj} \quad \text{where:}$$

$F(X_1, Y_1, \ldots, X_N, Y_N)$ – fitness function equal the to the number of all remaining wasps after bomb N

X_i and Y_i – co-ordinates of bomb i

N – number of bombs

M – number of nests

n_{0j} – number of wasps in a nest j before any bomb

n_{ij} – number of wasps remaining in a nest j after bomb i calculated by equation:

$$n_{ij} = \left\lfloor \frac{n_{((i-1)j)} \cdot d_{ij}}{d_{max}} \right\rfloor \quad \text{where:}$$

d_{max} – greatest possible distance between two nests

d_{ij} – Euclidean distance between bomb i and nest j

Evaluating the fitness of each individual in population.

```
# greatest possible distance between two nests
dMax = sqrt(20000)
#all bomb coordinates of currently processed solution
for i, bombs in enumerate(population):
     # temporary numbers of wasps in each nest
     tmpWasps = wasps[:]
     # co-ordinates of currently processed bomb
     for coords in bombs:
          # distances between each nest and currently processed bomb
          distances = norm(positions-coords, axis=1)
          # information about what part of wasps in each nest will remain after the bomb
          parts = distances/dMax
          # updating numbers of wasps in each nest
          tmpWasps = floor(parts*tmpWasps)
     # fitness of currently processed solution
          fitness[i] = sum(tmpWasps)
```

Selecting the parents

```
# number of parents used for reproduction
nParents = 20
# indices that would sort an array
indices = argsort(fitness)
# indices of the nParents best solutions
parentIds = indices[:nParents]
# nParents best solutions indicated by parentIndices
parents = population[parentIds]
```

Breeding new individuals through crossover and mutation operations

Simple Evolution strategy

```
# nOffspring clones
clones = repeat(parents[newaxis], nOffspring, 0)
# offspring created by adding noise to clones
offspring = clones + normal(size=(nOffspring, nBombs, 2))
```

Simple Genetic strategy


1
2
2
1

```
# container for nOffspring solutions derived from-parents
offspring = empty((nOffspring,nBombs,2))
for offspringId in range(nOffspring):
     # parents used to create new offspring solution
     mom = parents[offspringId%nParents]
     dad = parents[(offspringId+1)%nParents]
     # temporary container for offspring solution
     tmpOffspring = empty((nBombs, 2))
     # co-ordinates of currently processed mom bomb
     for i, coords in enumerate(mom):
          # distances between each dad bomb and currently processed mom bomb
          distances = norm(coords-dad, axis=1)
          # index of the dad bomb nearest to the currently processed mom bomb
          nearestId = argmin(distances)
          # dad bomb nearest to the currently processed mom bomb
          nearest = dad[nearestId]
          # child bomb derived from crossover between parent bombs
          childBomb = ( nearest + coords)/2
          # child bomb mutated by adding sample from a normal (Gaussian) distribution
          mutated = childBomb + normal(size=2)
          # filling temporary offspring container with bombs
          tmpOffspring[i] = mutated
     # filling offspring container with ready solutions
     offspring[offspringId] = tmpOffspring
```

Covariance-Matrix Adaptation Evolution strategy (CMA-ES)

```
# cotainer for bomb groups
clusters = empty((nBombs, nOffspring, 2))
# nParents best solutions indicated by parentIndices
parents = population[parentIds]
# flattened population, co-ordinates of all bombs
flatPopulation = reshape(parents, (-1, 2))
# tool used to group bombs according to their co-ordinates
kMeans = KMeans(n_clusters=nBombs).fit(flatPopulation)
# centers of each group
centers = kMeans.cluster_centers_
# labels of bombs
labels = pairwise_distances_argmin(flatPopulation, centers)
for i in range(nBombs):
      # all bomb belonging to the group i
      cluster = flatPopulation[labels == i]
      # covariance matrix within cluster
      covMatrix = cov(cluster, rowvar=False)
      # new bomb co-ordinates sampled from a multivariate normal distribution
      clusters[i] = multivariate_normal(centers[i], covMatrix, nOffspring)
      # container for nOffspring solutions from combining bomb groups
      population = empty((nOffspring, nBombs, 2))
# each new individual takes one bomb from each group
for i, bombs in enumerate(zip(*clusters)):
      population[i] = bombs
```

Updating population

Experiments


That's all Folks!