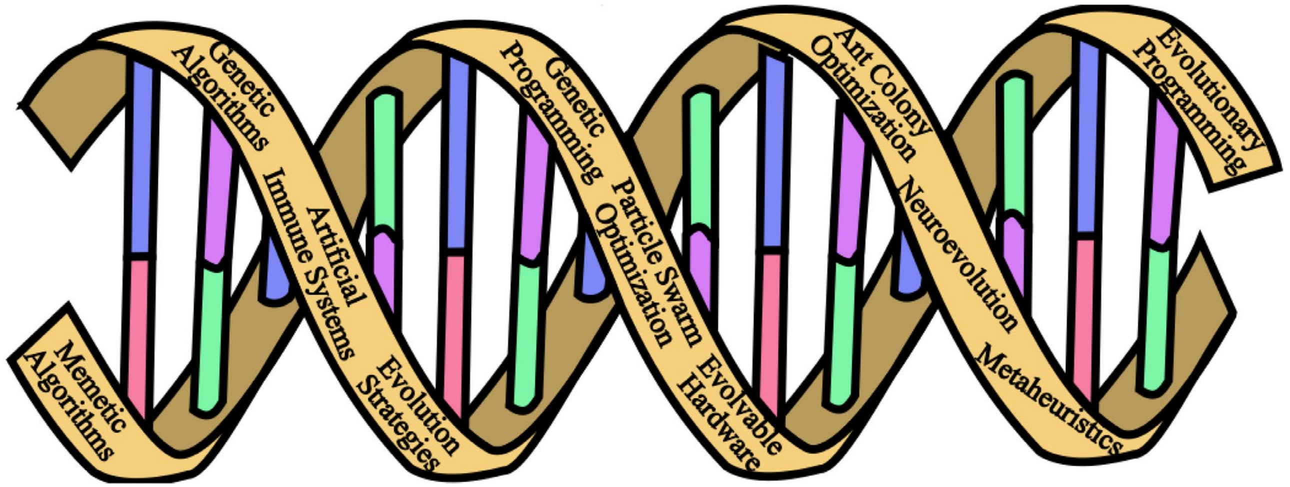
Introduction to Evolutionary Algorithms (EA)



CJ Chung
Lawrence Technological University
Computer Science

Lawrence Technological University

Intro to Genetic & Evolutionary Algorithms and Evolutionary Computing, videos

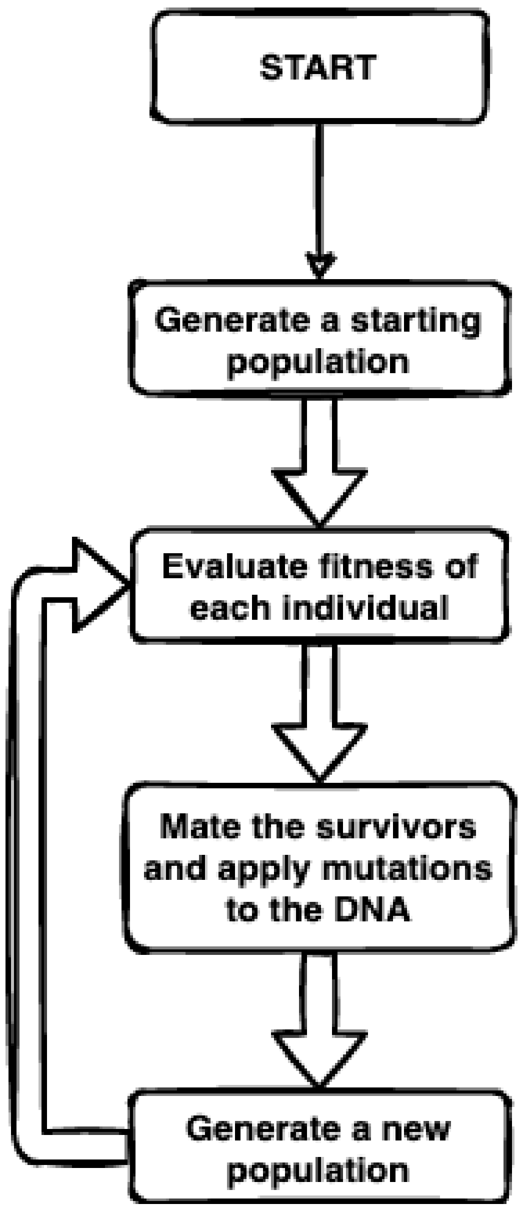
- https://youtu.be/L--IxUH4fac

  By Shahin Rostami at Bournemouth University, UK
- https://youtu.be/6l6b78Y4V7Y

  By Daniel Shiffman at NYU, USA
- https://youtube.com/playlist?list=PLXBbGVSkQJqEtpEGPUCAsyW1eYZwPAoNj

  IEE/CSE 598 (Bio-Inspired AI and Optimization) at Arizona State University, as taught by Theodore Pavlic.
- https://youtu.be/uQj5UNhCPuo?si=FTYRpeGi8kvFk1RD (11 min)

Please let us know if you find good videos.


START
Generate a starting population
Evaluate fitness of each individual
Mate the survivors and apply mutations to the DNA
Generate a new population

https://www.r-bloggers.com/2021/08/intro-to-evolutionary-algorithms-with-r-for-beginners-from-scratch-part-1

$$x^{t+1} = \underset{\text{Select}}{s}\Big(\underset{\text{Vary}}{v}\big(\underset{\uparrow}{x^{t}}\big)\Big)$$

A vector of candidate solutions at time t

Evolutionary Algorithm (Abstract)

$$\mathbf{x}^{t+1} = s(v(\mathbf{x}^{t}))$$

```
Initialize ($\mathbf{x}$) // a vector of
                          // candidate sol.
while (! Termination_Condition)
{   vary($\mathbf{x}$); // reproduce, Xover, mutate
    select($\mathbf{x}$);
}
```

Genetic Algorithm

```
$t$ = 0;
Initialize $pop_t$;
Evaluate each individual in the $pop_t$ by the objective function $f$
Repeat
    Repeat
        Select a pair from $pop_t$ for reproduction;
        Crossover and mutation;  // Usually mutation is embedded
                                 // within crossover
        Evaluate each offspring by $f$
    Until $\lambda$ offspring is generated;
    $t = t + 1$
    New $\lambda$ offspring becomes $pop_t$
Until the termination condition is met;
```

Representation - Chromosomes could be:

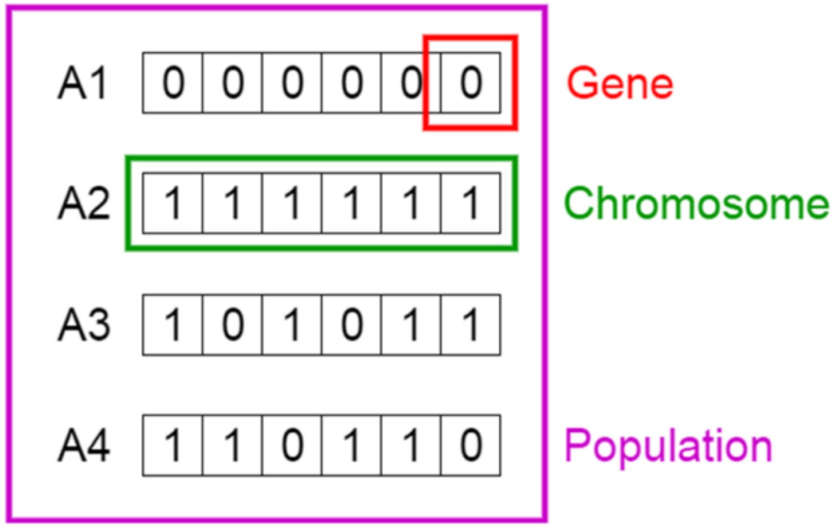
• Binary Bit strings	(0101 ... 1100)
• Integer numbers	(1 2 4 6 … 7 -2 5 9)
• Real numbers	(43.0 -33.1 ... 0.0 89.2)
• Permutations of element	(E11 E3 E7 ... E1 E15)
• Lists of rules	(R1 R2 R3 ... R22 R23)
• Program elements	(genetic programming)

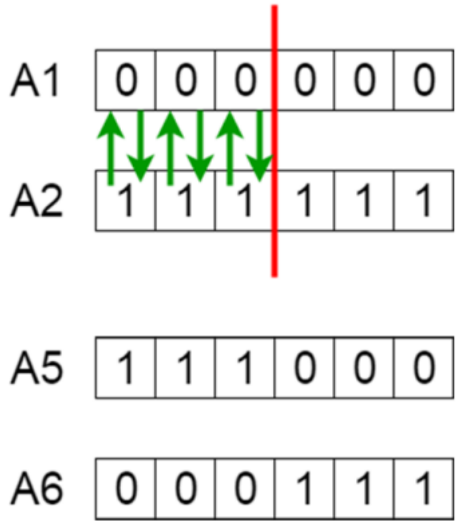
- Any data structure
    - Decision Trees
    - Neural Networks, CNNs, RNNs, …
    - Graph NNs
    - LLMs / LFMs
    - …

EDL

Binary GA (BGA)

The Binary Genetic Algorithm (BGA) uses a binary representation of data. This means that each individual (solution) is represented as a string of bits (0 and 1)





Offspring by single point crossover

Integer GA Example: Cookie World

(Winston, 1992)

Cookie quality is dependent on the amount of flour and sugar. An ***expert*** knows the following table:

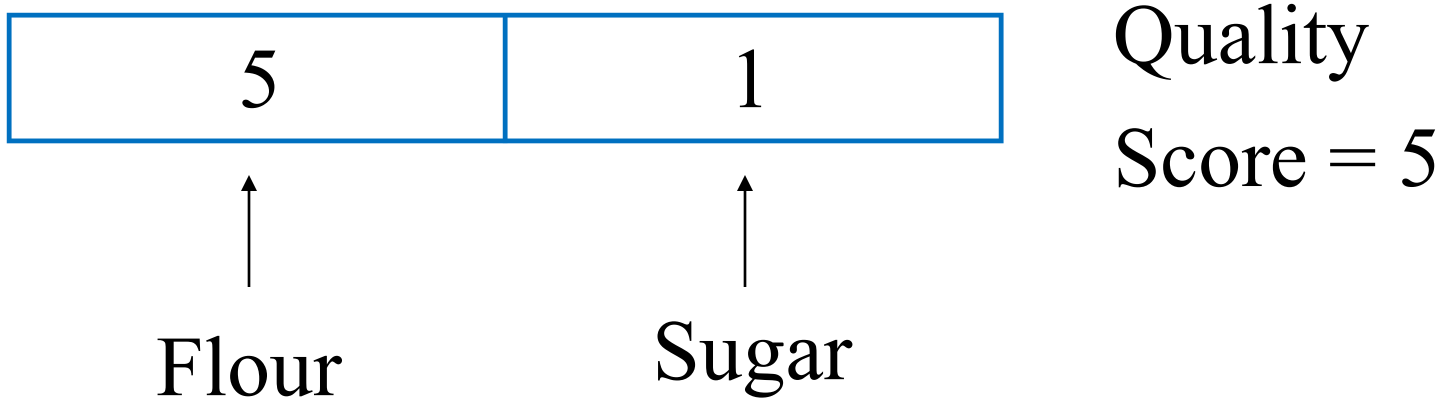
sugar

9	1	2	3	4	5	4	3	2	1
8	2	3	4	5	6	5	4	3	2
7	3	4	5	6	7	6	5	4	3
6	4	5	6	7	8	7	6	5	4
5	5	6	7	8	**9**	8	7	6	5
4	4	5	6	7	8	7	6	5	4
3	3	4	5	6	7	6	5	4	3
2	2	3	4	5	6	5	4	3	2
1	1	2	3	4	5	4	3	2	1
	1	2	3	4	**5**	6	7	8	9

flour

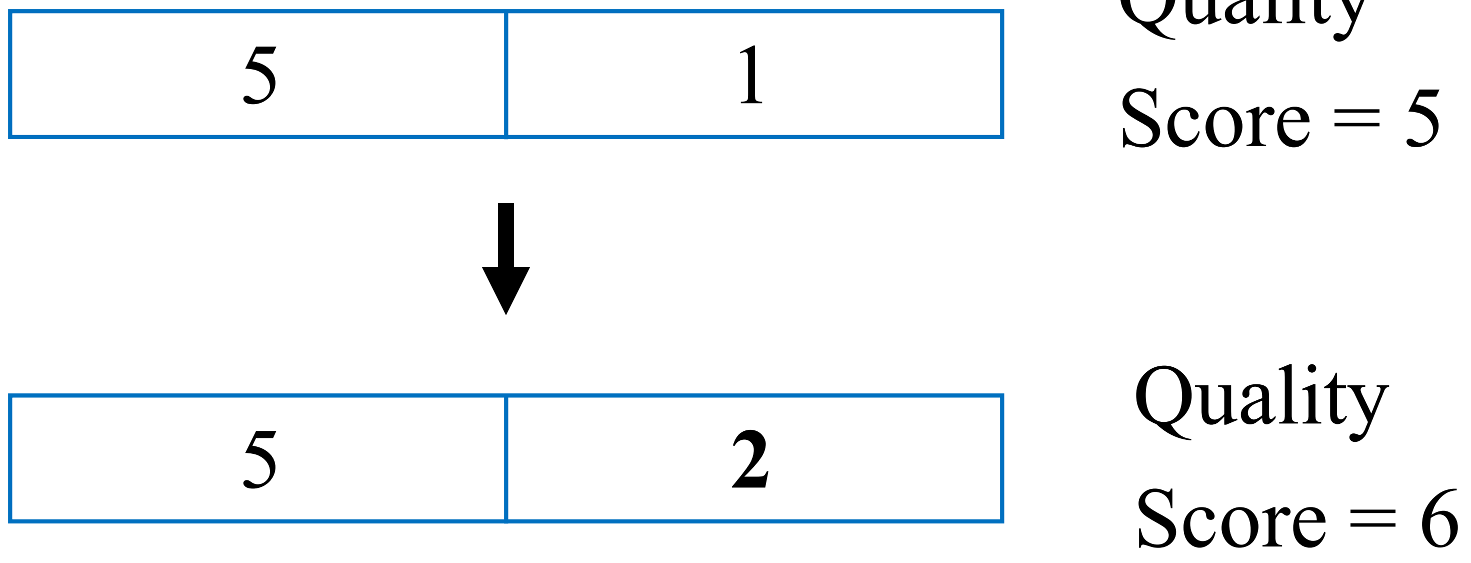
Cookie World

- A chromosome:



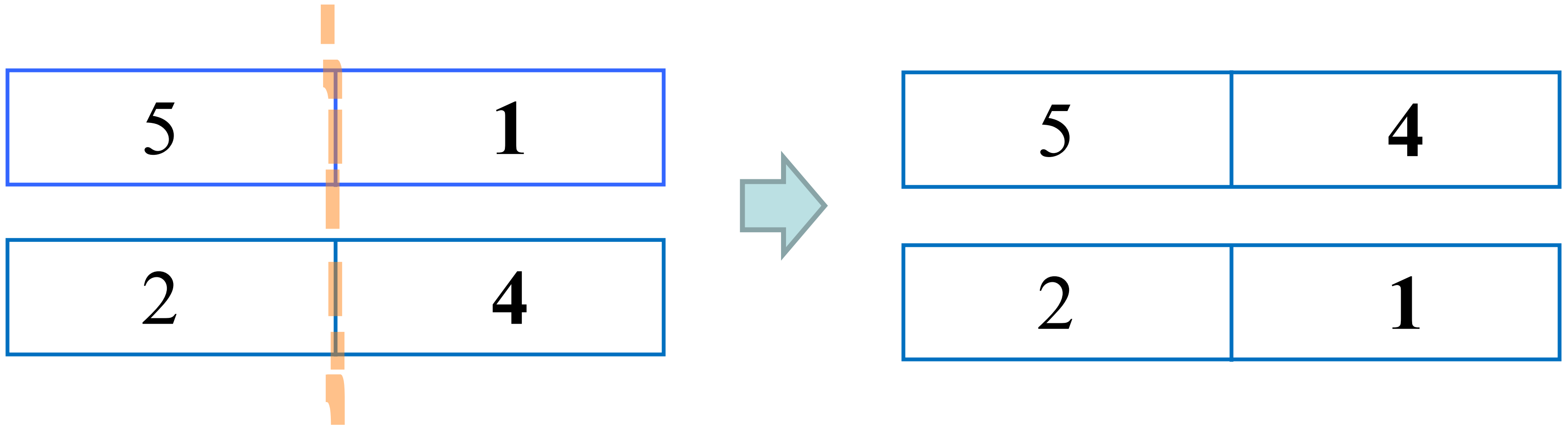
Example: Cookie World, II

- Mutation



Example: Cookie World, III

- Crossover



Quality Score = 8

Quality Score = 2

Cookie GA

- Create an initial “population” of chromosomes
- Mutate one or more genes in one or more of the current chromosomes, producing a new offspring
- Mate one or more pairs of chromosome
- Create a new generation by keeping the best, along with others selected at random or based on assessed fitness

Questions

- How many individuals in the initial population?
- The world is too small… Only a limited number of individuals can survive. How many chromosomes are to be in the population?
- How to select?
- How to mate. Crossover point
- How to mutate. Mutation rate.

Natural Selection Methods

- Lottery: everyone has an equal chance to be selected regardless of his/her performance value. – Uniform Random Selection
- Extinctive Selection
  - Dynamic: scores greater than 60% can pass the course
  - Static: only a half can survive
- **Roulette Wheel (proportional)**
- **Rank (non-linear)**
- **Tournament Selection**: Popular method in EA

Proportional Selection

$$f_i = \frac{q_i}{\sum_j q_j}$$

- Standard fitness, Roulette Wheel
- Note: if quality is zero, fitness is zero
- There is no way to influence the selection

Selection: Rank Method

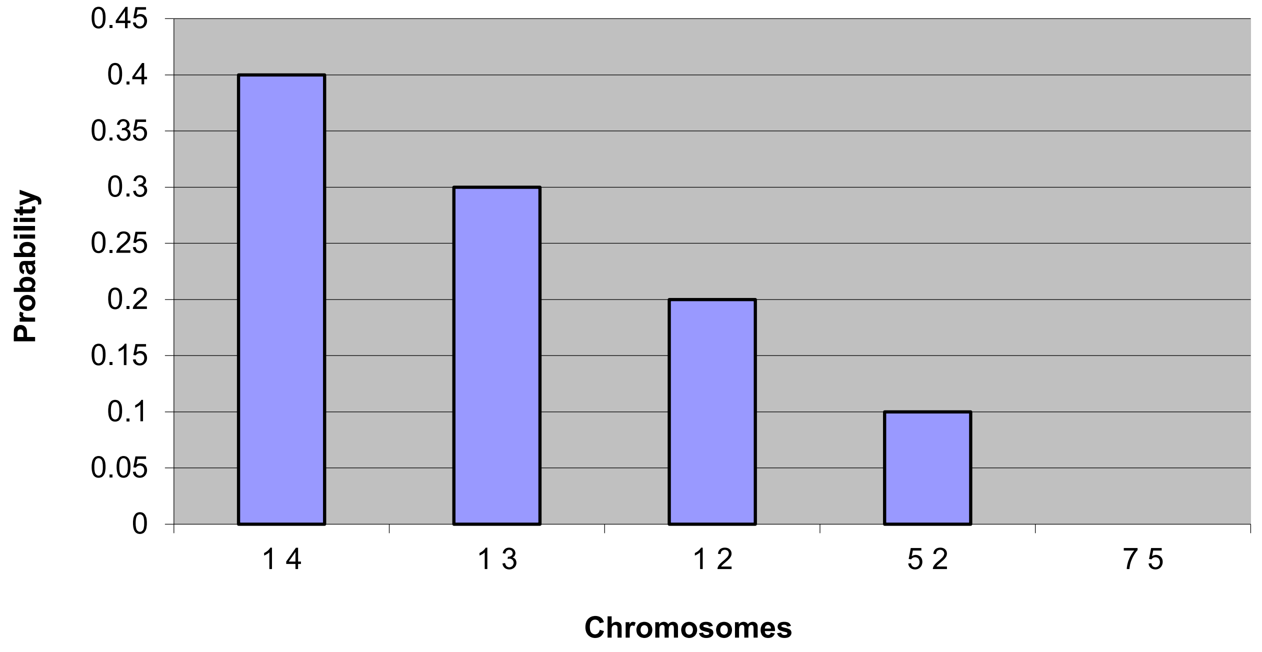
- Not only offers a way of controlling the bias toward the best chromosome
- But also eliminates implicit biases, introduced by unfortunate choices of the measurement scale
- E.g. with a fixed constant *p* = 0.667 (prob. of survival)
  - 1*0.667 = 0.667 for the 1st
  - (1 - 0.667)*0.667 = 0.222 for the 2nd
  - (1 - (0.667+0.222))*0.667 = 0.074 for the 3rd
  - …
  - Quality zero has a chance of surviving, 0.012

Selection Methods

$\frac{4}{4+3+2+1+0}$

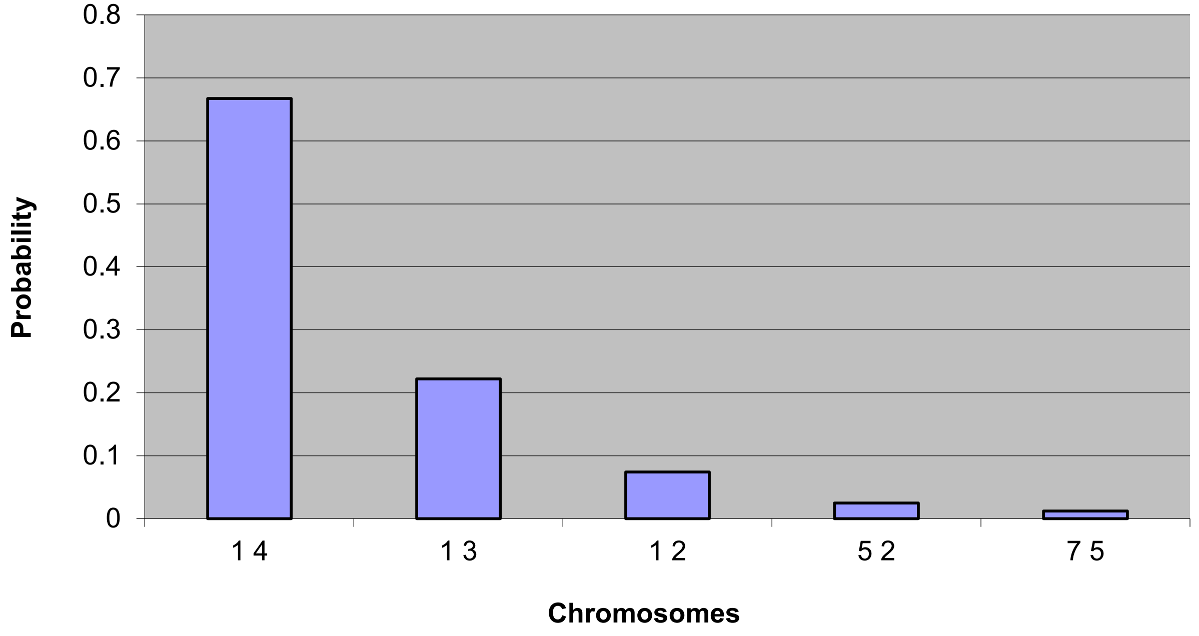
Chromosome	Quality	Rank	Std. fitness	Rank fitness
1 4	4	1	0.4	0.667
1 3	3	2	0.3	0.222
1 2	2	3	0.2	0.074
5 2	1	4	0.1	0.025
7 5	0	5	0.0	0.012

Comparisons: Standard vs. Ranking Fitness

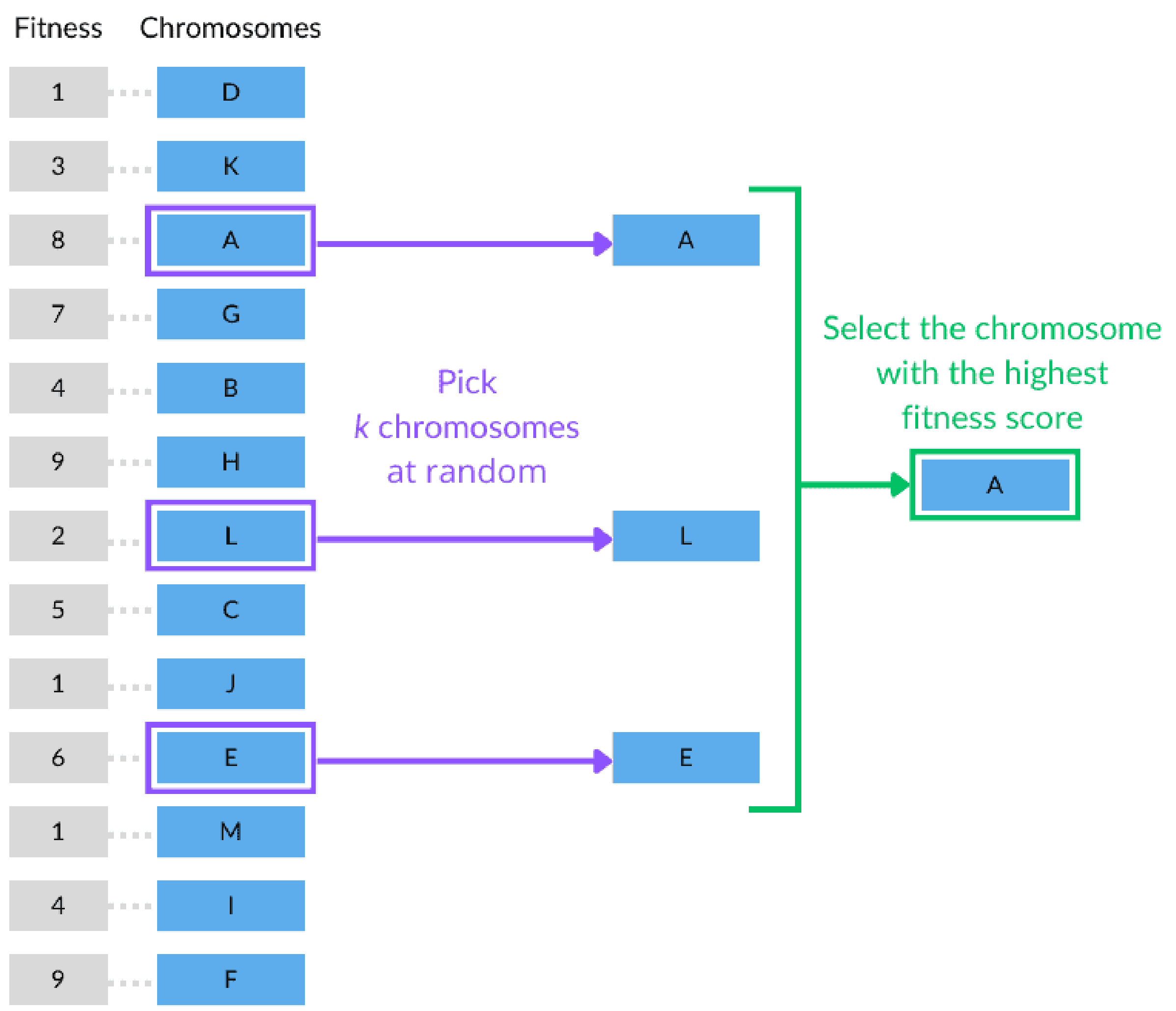


Proportional (Roulet Wheel)

Ranking



Tournament Selection Example, *k*=3



Tournament Selection (1 /3)

- The process involves selecting a subset of individuals from the population and then choosing the best (most fit) individual from that subset
- Repeat this process until we've selected the desired number of individuals (population size) for the next generation
- Each time you start a new tournament, individuals are "usually" chosen with replacement - meaning the same individual can be selected in multiple tournaments, and even multiple times in the same tournament.

Tournament Selection (2 /3)

Widely used because it allows the selection pressure to be easily adjusted

- If k is larger, weak individuals have a smaller chance to be selected
- If k is smaller, weak individuals have a higher chance to be selected
- If the subset size (*k*, tournament size) is 1, random selection

Tournament Selection (3 /3)

The suggested tournament size *k* in evolutionary algorithms typically depends on the balance between exploration and exploitation:

- **Small k (2 or 3):** Leads to more exploration, maintaining genetic diversity and preventing premature convergence. This helps avoid local optima but may slow down convergence.
- **Larger k (5 or more):** Increases selective pressure, favoring the fittest individuals more strongly. This accelerates convergence but risks reducing diversity and getting stuck in local optima.

- Future work: dynamic *k*

Simple Function Optimization using Binary GA

- https://youtu.be/EJeTWRP3Bd0?si=gzRc4AQuhSnctIKd
- https://github.com/bnsreenu/python_for_microscopists/blob/master/314_How_to_code_the_genetic_algorithm_in_python.ipynb

Please test the code and watch the video

Numpy randint()

```
from numpy.random import randint
# len(pop) is 5
# k-1 is 3
random_indices = randint(0, len(pop), k-1)
```

This generates an array of 3 random integers, where each integer is between 0 (inclusive) and 5 (exclusive).

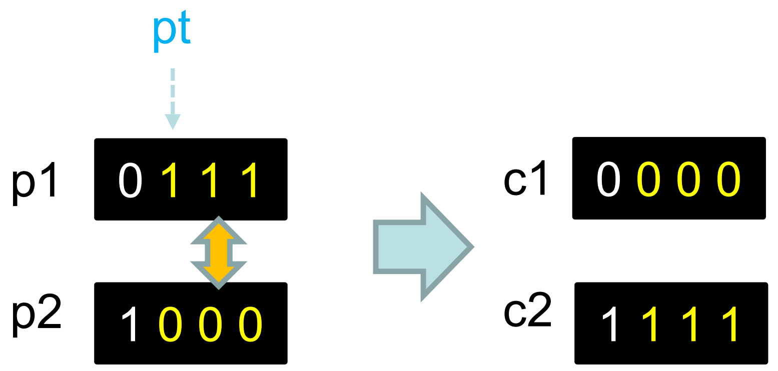
random_indices could be [0, 4, 1], [3, 3, 0], [2, 4, 3], etc.

Randomly selects *k* individuals from the population and performs a **tournament** among them to choose the one with the best score

```
def selection(pop, scores, k=3):
  # Randomly select one index from the population as the initial selection
  selection_ix = randint(len(pop))
  # Perform a tournament among k randomly selected individuals
  for ix in randint(0, len(pop), k-1):
      # Check if the current ind. has a better score than the selected one
      if scores[ix] < scores[selection_ix]:
          # Update the selected individual if a better one is found
          selection_ix = ix
  # Return the best individual from the tournament
  return pop[selection_ix]
```

Create two children from two parents using the crossover operation. The children are created by copying the parents, and recombination occurs if a random value is less than the crossover rate

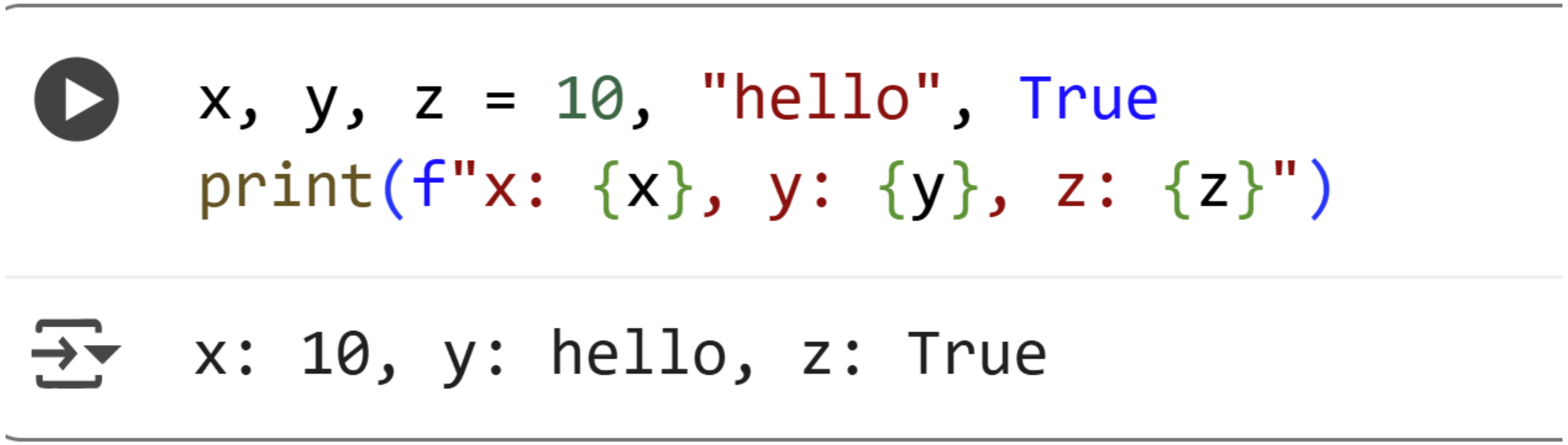
```python
def crossover(p1, p2, r_cross): # r_cross = 0.9
  # Children are copies of the parents by default
  c1, c2 = p1.copy(), p2.copy()
  # Check if recombination should occur
  if rand() < r_cross:
      # Select a crossover point (not at the end of the string)
      pt = randint(1, len(p1)-2) # suppose len(p1) is 4 and pt is 1
      # Perform crossover in the children
      c1 = p1[:pt] + p2[pt:]
      c2 = p2[:pt] + p1[pt:]
  # Return the two children
  return [c1, c2]
```



The mutation process changes the value of some features in the offspring at random to maintain the diversity in the population. A standard value for the mutation rate is 1/*m* where *m* is the number of features.

```
def mutation(bitstring, r_mut):
  rand = random.random
  for i in range(len(bitstring)):
      # Check for a mutation
      if rand() < r_mut:
          # Flip the bit
          bitstring[i] = 1 - bitstring[i]
  return bitstring
```

Assigning Multiple Values to Multiple Variables in Python



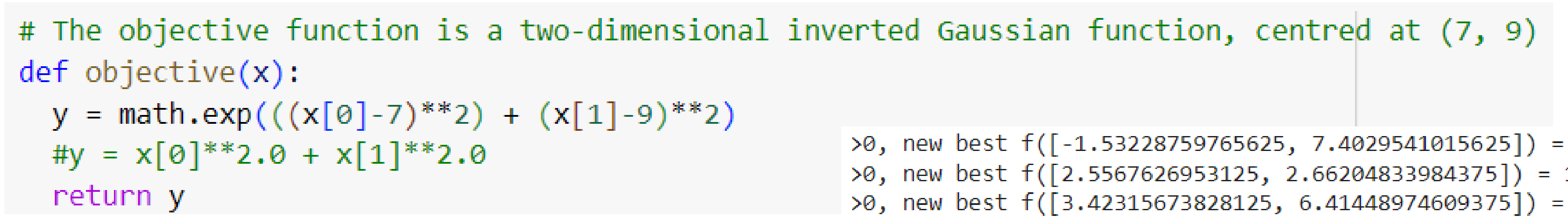
```python
x, y, z = 10, "hello", True
print(f"x: {x}, y: {y}, z: {z}")
```

```
x: 10, y: hello, z: True
```

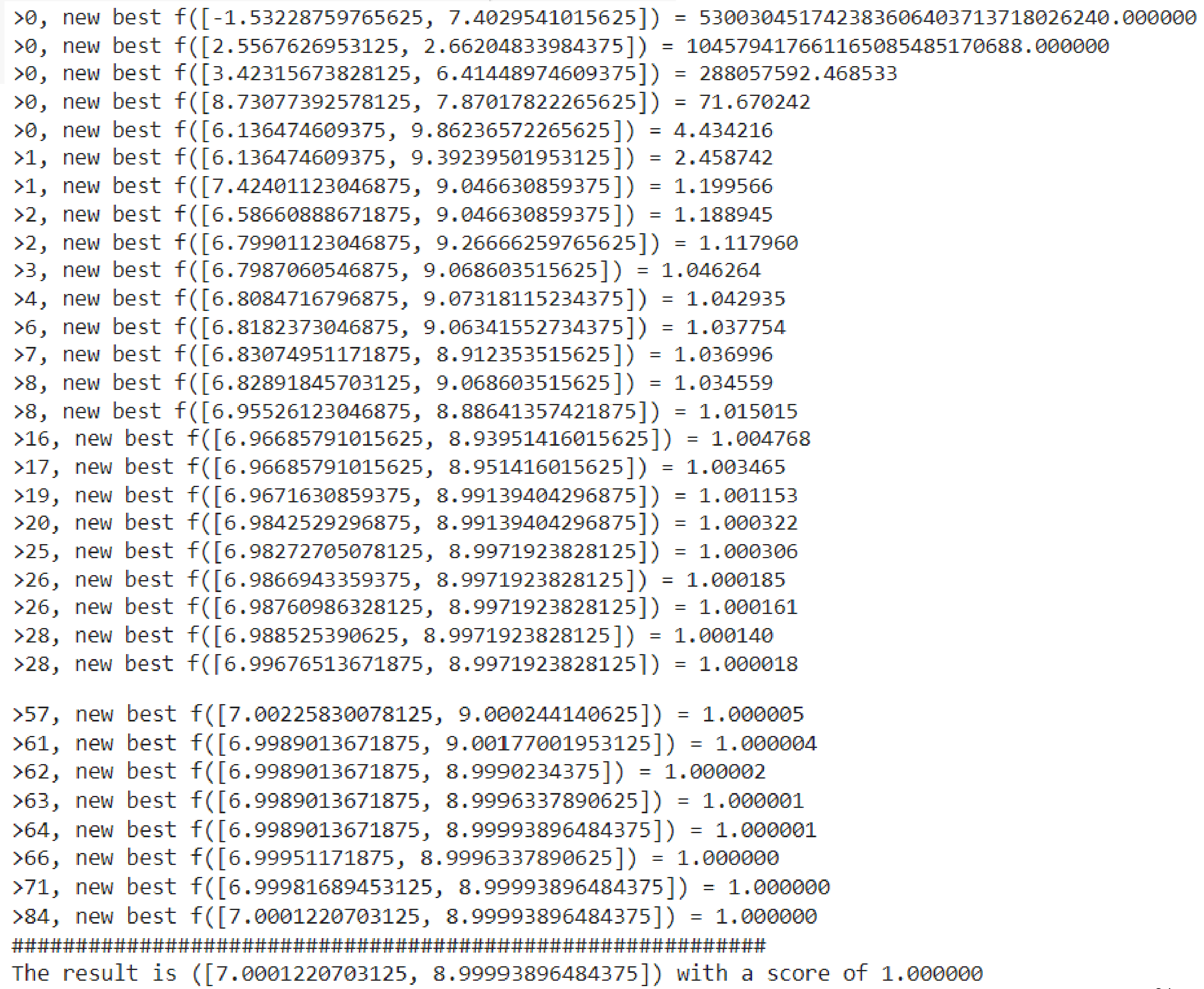
```python
def genetic_algorithm(objective, bounds, n_bits, n_iter, n_pop, r_cross, r_mut):
    pop = [randint(0, 2, n_bits * len(bounds)).tolist() for _ in range(n_pop)] # init pop with random bit strs
    best, best_eval = 0, objective(decode(bounds, n_bits, pop[0])) # track the best solution found so far

    for gen in range(n_iter):  # iterate over generations
        # decode the population
        decoded = [decode(bounds, n_bits, p) for p in pop]
        # evaluate all candidates in the population
        scores = [objective(d) for d in decoded]
        # check for a new best solution
        for i in range(n_pop):
            if scores[i] < best_eval:
                best, best_eval = pop[i], scores[i]
                print(">%d, new best f(%s) = %f" % (gen,  decoded[i], scores[i]))
        # select parents
        selected = [selection(pop, scores) for _ in range(n_pop)]
        # create the next generation - children
        children = list()
        for i in range(0, n_pop, 2):
            # get selected parents in pairs
            p1, p2 = selected[i], selected[i + 1]
            # crossover and mutation
            for c in crossover(p1, p2, r_cross): # c has 2 children
                # mutation
                mutation(c, r_mut)
                # store for next generation
                children.append(c)
        # replace the population
        pop = children
    return [best, best_eval]
```

binaryGA_fnOptim.ipynb



```python
# The objective function is a two-dimensional inverted Gaussian function, centred at (7, 9)
def objective(x):
  y = math.exp(((x[0]-7)**2) + (x[1]-9)**2)
  #y = x[0]**2.0 + x[1]**2.0
  return y
```



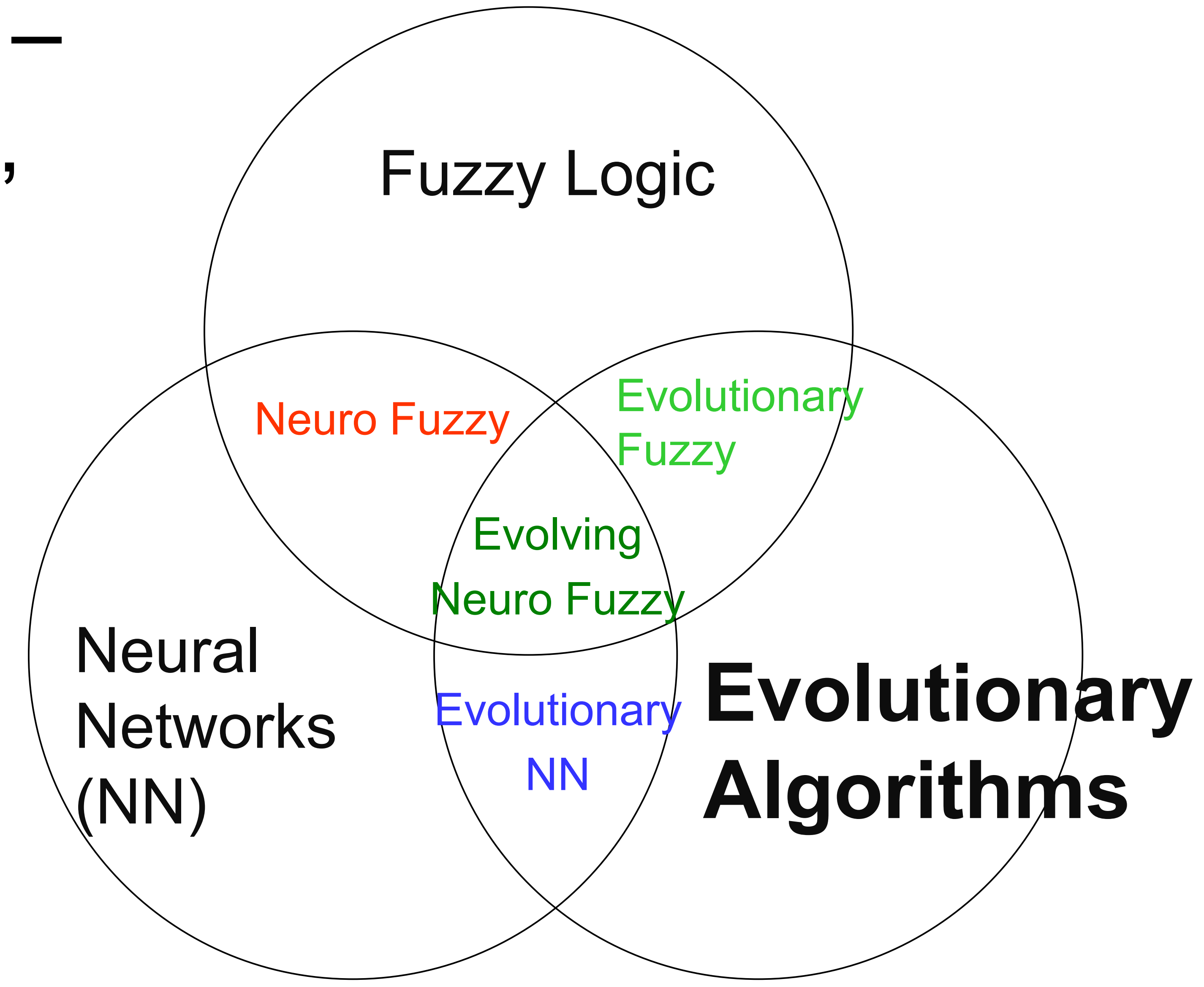
```
>0, new best f([-1.53228759765625, 7.4029541015625]) = 530030451742383606403713718026240.000000
>0, new best f([2.5567626953125, 2.66204833984375]) = 1045794176611650854851706888.000000
>0, new best f([3.42315673828125, 6.41448974609375]) = 288057592.468533
>0, new best f([8.73077392578125, 7.87017822265625]) = 71.670242
>0, new best f([6.136474609375, 9.86236572265625]) = 4.434216
>1, new best f([6.136474609375, 9.39239501953125]) = 2.458742
>1, new best f([7.42401123046875, 9.046630859375]) = 1.199566
>2, new best f([6.58660888671875, 9.046630859375]) = 1.188945
>2, new best f([6.79901123046875, 9.26666259765625]) = 1.117960
>3, new best f([6.7987060546875, 9.068603515625]) = 1.046264
>4, new best f([6.8084716796875, 9.07318115234375]) = 1.042935
>6, new best f([6.8182373046875, 9.06341552734375]) = 1.037754
>7, new best f([6.83074951171875, 8.912353515625]) = 1.036996
>8, new best f([6.82891845703125, 9.068603515625]) = 1.034559
>8, new best f([6.95526123046875, 8.88641357421875]) = 1.015015
>16, new best f([6.96685791015625, 8.93951416015625]) = 1.004768
>17, new best f([6.96685791015625, 8.951416015625]) = 1.003465
>19, new best f([6.9671630859375, 8.99139404296875]) = 1.001153
>20, new best f([6.9842529296875, 8.99139404296875]) = 1.000322
>25, new best f([6.98272705078125, 8.9971923828125]) = 1.000306
>26, new best f([6.9866943359375, 8.9971923828125]) = 1.000185
>26, new best f([6.98760986328125, 8.9971923828125]) = 1.000161
>28, new best f([6.988525390625, 8.9971923828125]) = 1.000140
>28, new best f([6.99676513671875, 8.9971923828125]) = 1.000018

>57, new best f([7.00225830078125, 9.000244140625]) = 1.000005
>61, new best f([6.9989013671875, 9.00177001953125]) = 1.000004
>62, new best f([6.9989013671875, 8.9990234375]) = 1.000002
>63, new best f([6.9989013671875, 8.9996337890625]) = 1.000001
>64, new best f([6.9989013671875, 8.99993896484375]) = 1.000001
>66, new best f([6.99951171875, 8.9996337890625]) = 1.000000
>71, new best f([6.99981689453125, 8.99993896484375]) = 1.000000
>84, new best f([7.0001220703125, 8.99993896484375]) = 1.000000
############################################################
The result is ([7.0001220703125, 8.99993896484375]) with a score of 1.000000
```

Application Areas of EC

- Problems with little or no domain knowledge
- NP-hard problems
- Problems with near optimal solution is acceptable
- Problems with non-smooth (discontinuous; not differentiable) and noisy search space
- Problems whose environments are uncertain and/or dynamic

Soft Computing – Hybriding Fuzzy, NN, and EA



To show how EA can be applied in soft computing

EA and EC resources

- https://youtu.be/kHyNqSnzP8Y?si=Ga3irEg6okPnJ8cj (by P Winston, MIT Open Course)
- …
- Please suggest us good YouTube videos to introduce EA, GA, and EC

Simple Example Codes on the web

- https://medium.com/thecyphy/travelling-salesman-problem-using-genetic-algorithm-130ab957f165 (TSP)
- https://youtu.be/nhT56blfRpE?si=atXy_IAA6bi65mgj (Knapsack Problem)
- https://www.youtube.com/watch?v=dhWfbY2K2mk (Optimizing random forest)
- …
- Please also suggest us other simple examples

Review Q: Complete the field

```
def selection(pop, scores, k=3):
  # Randomly select one index from the population as the initial selection
  selection_ix = randint(len(pop))
  # Perform a tournament among k randomly selected individuals
  for ix in randint(0, len(pop),     ):
      # Check if the current ind. has a better score than the selected one
      if scores[ix] < scores[selection_ix]:
          # Update the selected individual if a better one is found
          selection_ix = ix
  # Return the best individual from the tournament
  return pop[selection_ix]
```

Self-Ex: Simple Function Optimization using Binary GA

- https://youtu.be/EJeTWRP3Bd0?si=gzRc4AQuhSnctIKd
- https://github.com/bnsreenu/python_for_microscopists/blob/master/314_How_to_code_the_genetic_algorithm_in_python.ipynb

Test the following function and other functions with this code.

```
def objfunc(x):  # Easy Rosenbrock function
    return (1-x[0])**2 + (x[1]-x[0]**2)**2
```