Mahidol University
Faculty of Medicine Ramathibodi Hospital
Department of Clinical Epidemiology and Biostatistics

Genetic Algorithm

RADI608: Data Mining and Machine Learning

RADI602: Data Mining and Knowledge Discovery

Lect. Anuchate Pattanateepapon, D.Eng.

Section of Data Science for Healthcare

Department of Clinical Epidemiology and Biostatistics

Faculty of Medicine Ramathibodi Hospital, Mahidol University

© 2022

Wisdom of the Land

Guideline

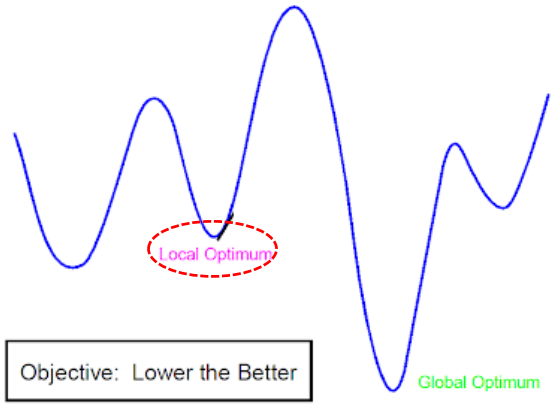
- Overview of Local Search and Local Search-based Metaheuristics
- Introduction to Genetic Algorithm (GA)
- Examples
- Practice with Python

Local Search

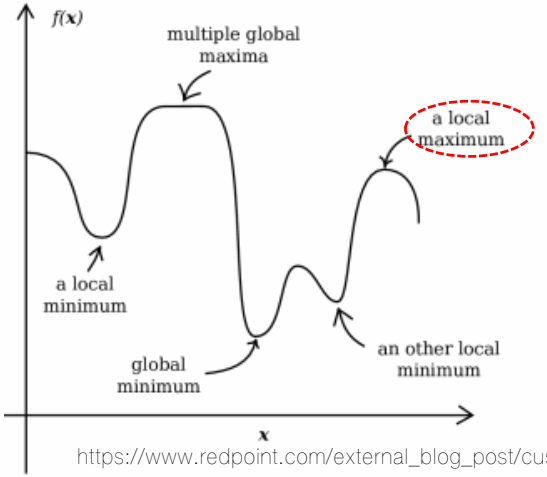
- Basic Idea: Improve the current solution
- Start with some solution
- Find a set of solutions (called neighbors) that are ”close” to the current solution
- If one of these neighbors are better than the current solution, move to that solution
- Repeat until no improvements can be made

Local Search

- Variations
  - Best Improvement (always select the best neighbor)
  - First Improvement (select the first improving neighbor)
  - Random Descent (select neighbors at random)
  - Random Walk (move to neighbors at random)
- Problem: Gets stuck in a local optimum/maximum



http://hotrodanglican.blogspot.com/2010_11_03_archive.html



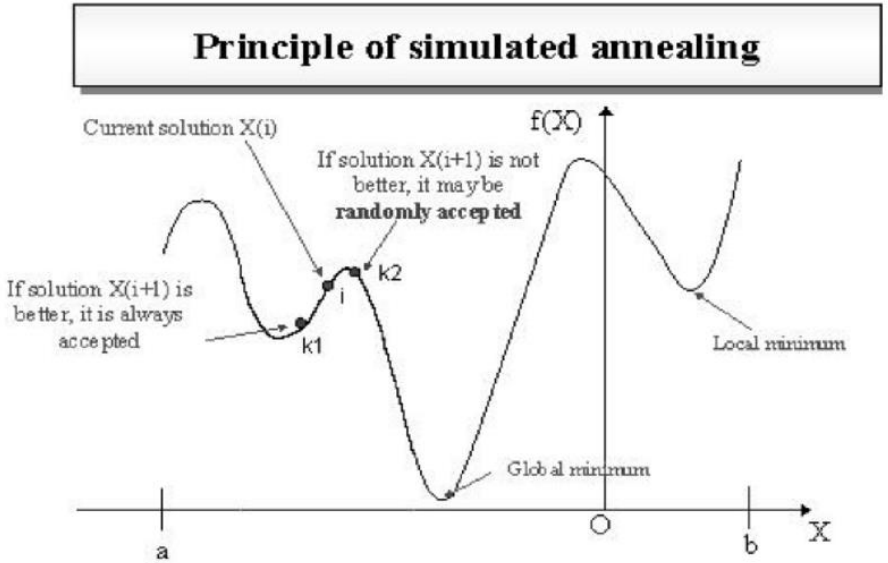
https://www.redpoint.com/external_blog_post/customer-operations-idea-maximizing-efficient-growth-saas-companies/global-local-optima-png/

Local Search Based Metaheuristics

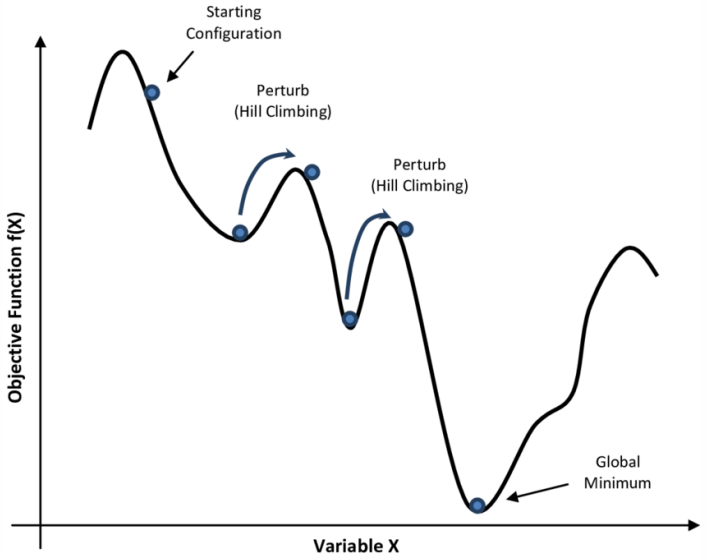
- Main goal
  - To avoid getting stuck in local optima
- Additional goals
  - Explore a larger part of the search space
  - Attempt to find the global (not just a local) optimum
  - Give a reasonable alternative to exact methods (especially for large/hard problem instances, and where the solution time is important)

Local Search Based Metaheuristics

- The different methods employ very different techniques in order to escape local optima or explore a larger part of the search space
  - Simulated Annealing (SA) relies on controlled random movement



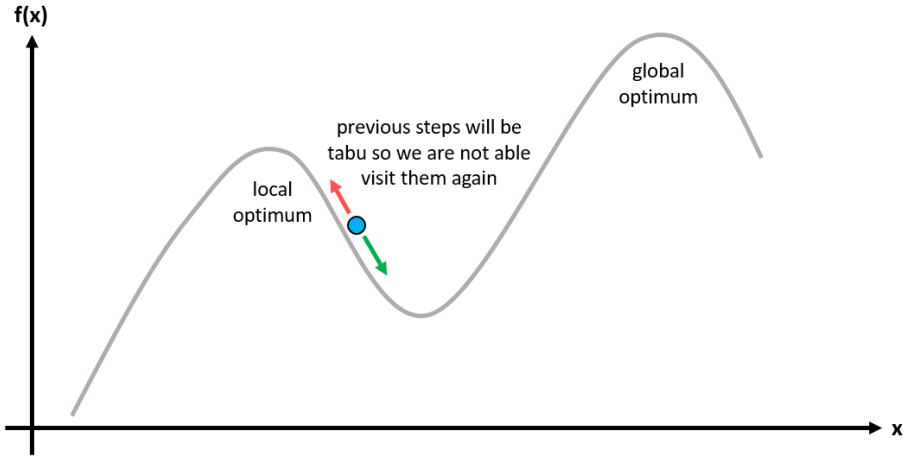
https://www.researchgate.net/figure/Simulated-Annealing-optimization-of-a-one-dimensional-objective-function_fig1_308786233



https://www.semanticscholar.org/paper/Implementation-of-a-Simulated-Annealing-algorithm-Moins/0f175f0f5ba0eae1783f03d6d39e52e6cd1f6726

Local Search Based Metaheuristics

- Tabu Search (TS) relies on memory structures recording enough information to prevent looping between solutions, recording enough information to guide the search to different areas of the search space (e.g., frequency based diversification)



https://medium.com/rideos/tabu-search-for-the-vehicle-routing-problem-b1fd993f4301

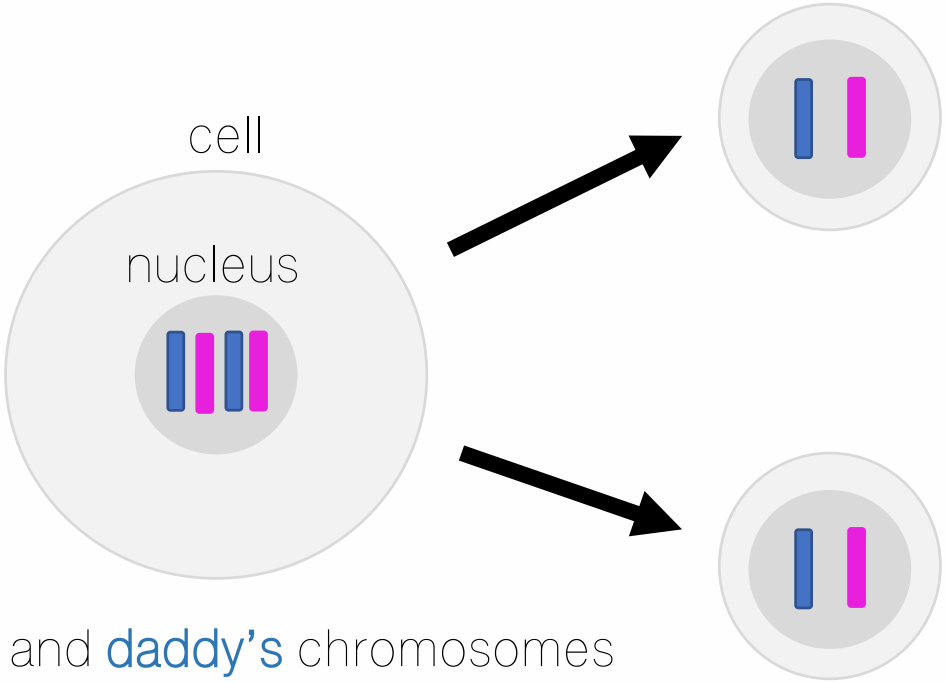
Introduction to Genetic Algorithm (GA)

a) John Holland introduced genetic algorithms in 1960 based on the concept of Darwin's theory of evolution.

b) GA is a *metaheuristic* inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA).

c) GA is commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as *mutation*, *crossover* and *selection*

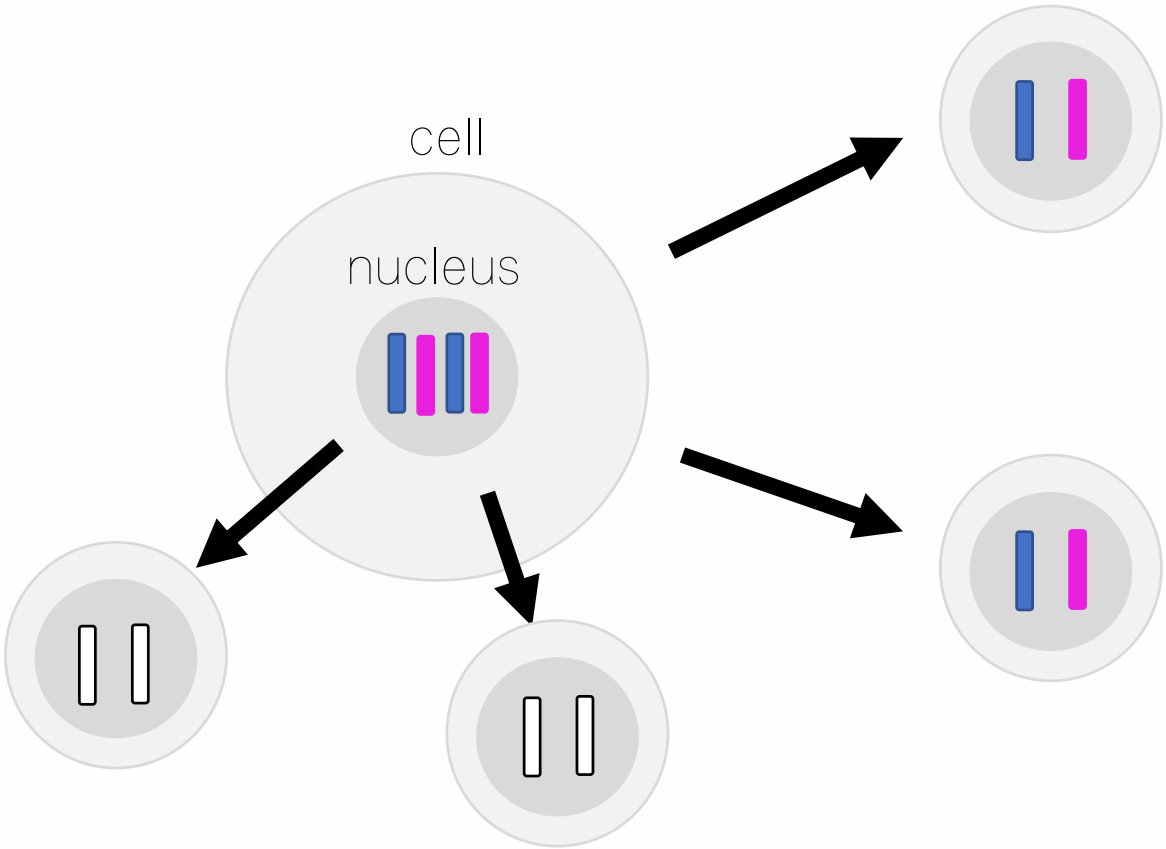
Introduction to Genetic Algorithm (GA)



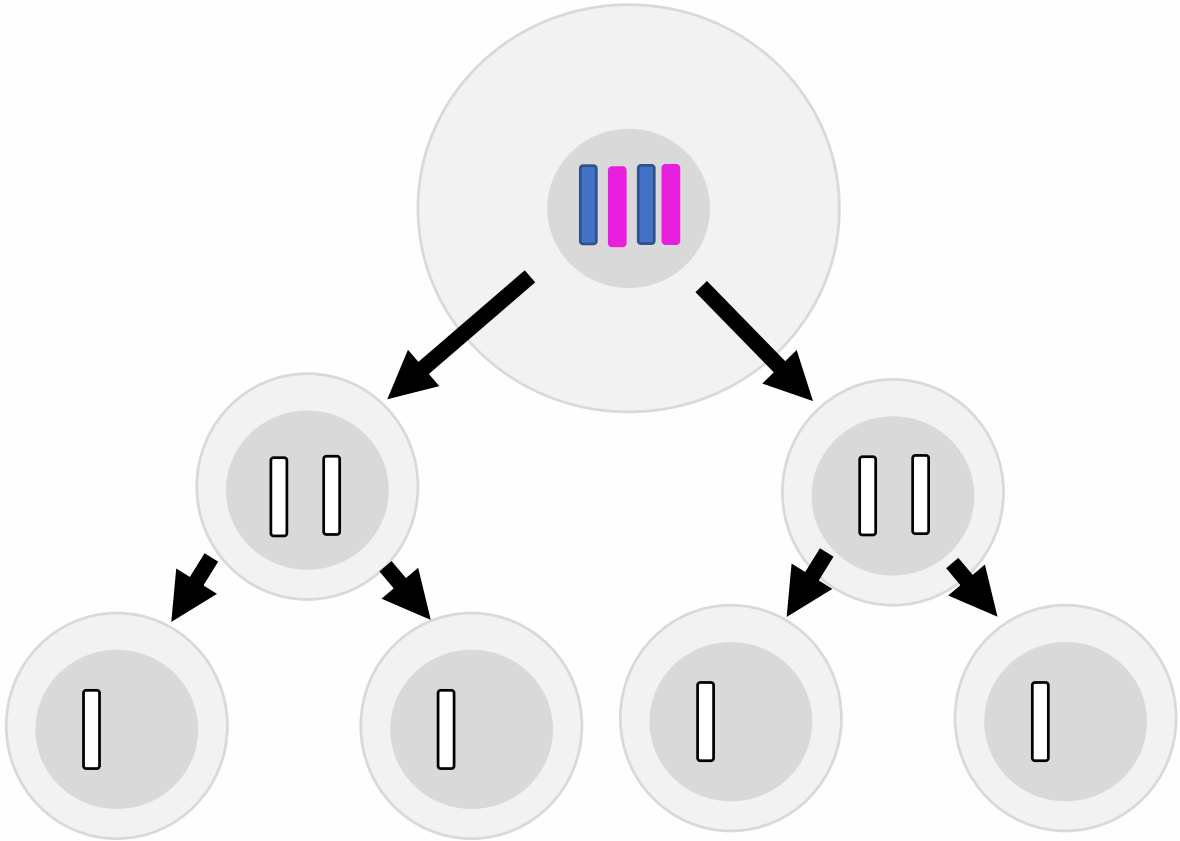
Mommy and daddy's chromosomes

(pretend there's one pair)

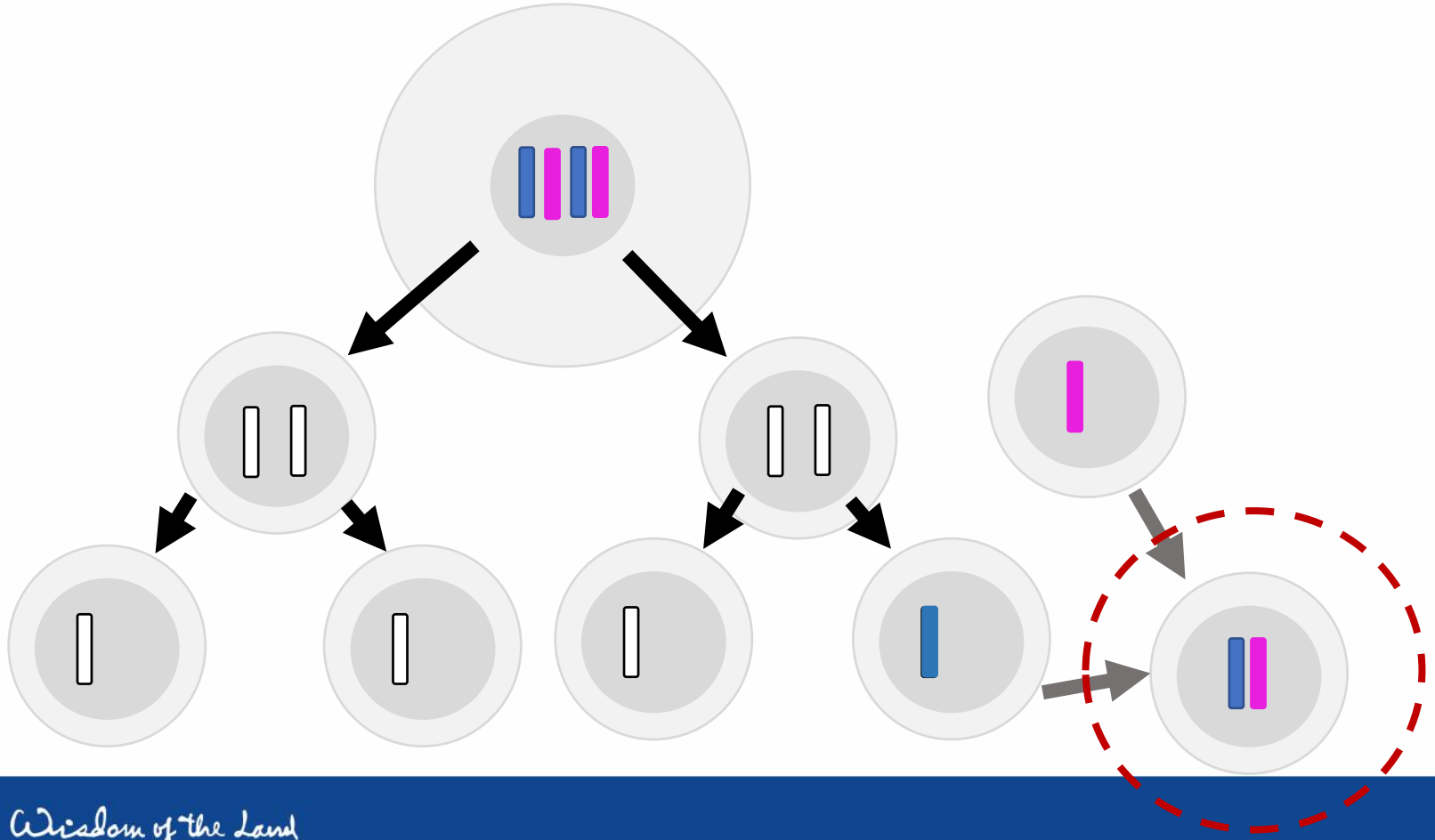
Introduction to Genetic Algorithm (GA)



Introduction to Genetic Algorithm (GA)



Introduction to Genetic Algorithm (GA)



The Simple Genetic Algorithm (SGA)

- Holland's original GA is now known as the simple genetic algorithm (SGA)
- Other GAs use different:
  - Representations
  - Mutations
  - Crossovers
  - Selection mechanisms

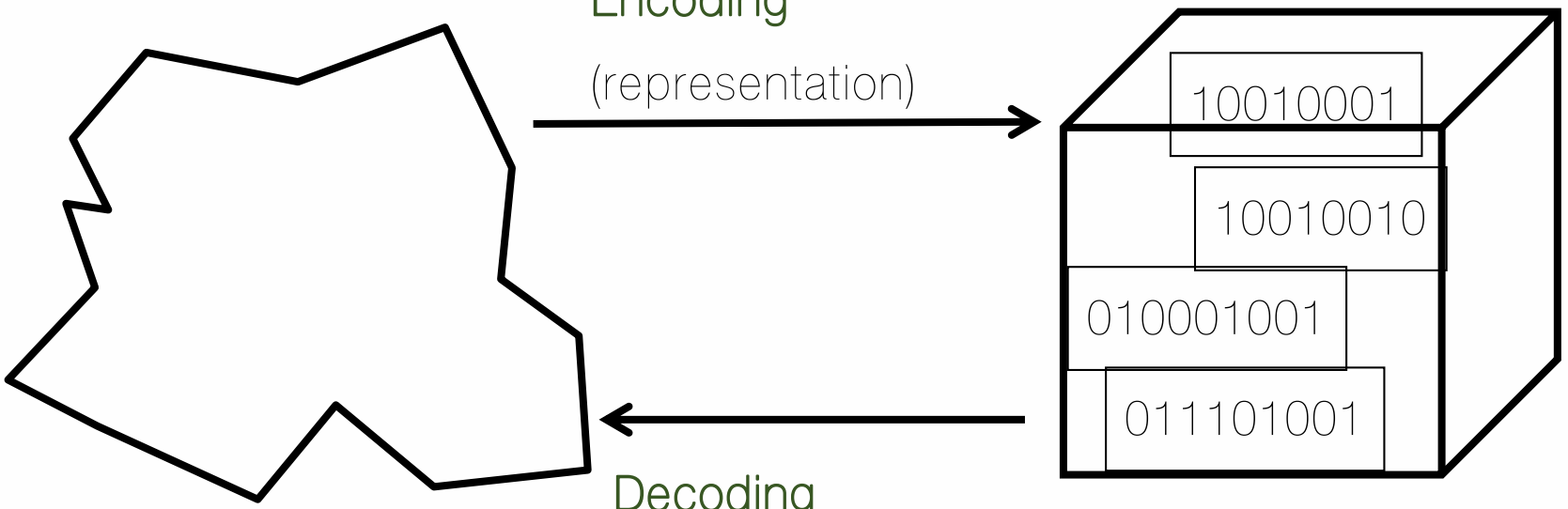
Representation

Phenotype space

features = {f1, f2, f3, f4, f5, f6, f7, f8}

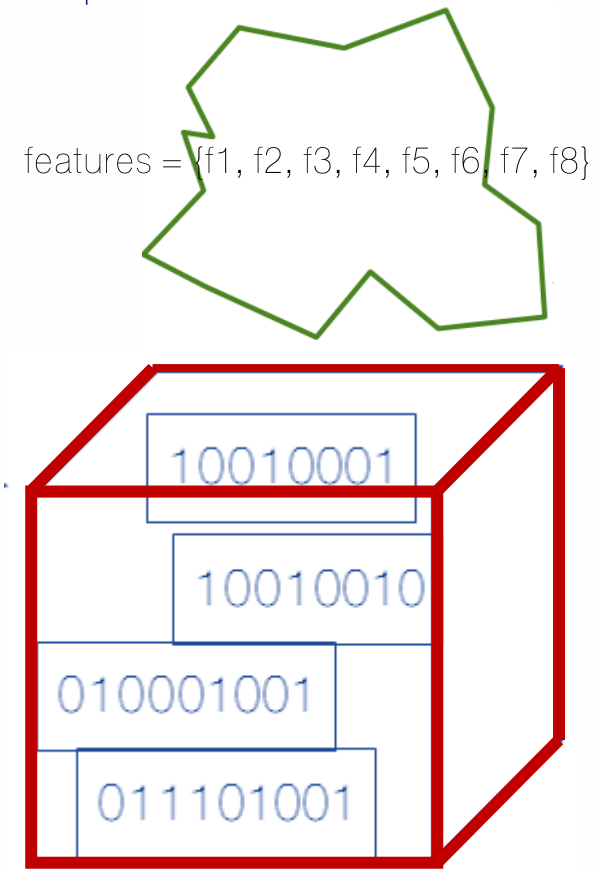
Genotype space = $\{0,1\}^L$

(1-select, 0-not select)

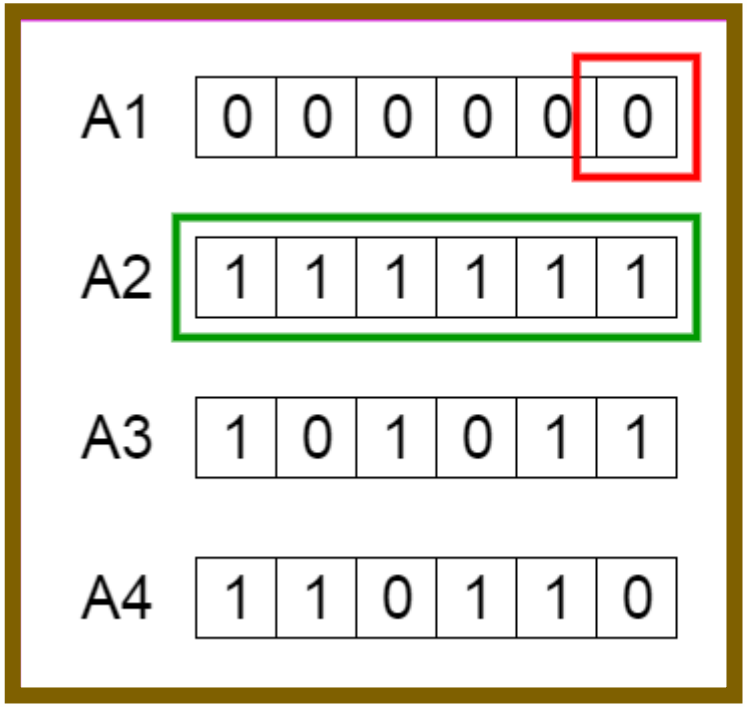


Phenotype, Genotype, Population

- Phenotype
  - The physical expression
  - properties of a set of solutions
- Genotype
  - chromosome
  - coding of chromosomes
  - coded string, set of coded strings
- Population – a set of solutions



The Chromosomes in GA



Gene

Chromosome

(Binary Chromosomes, $\{0, 1\}^L$)

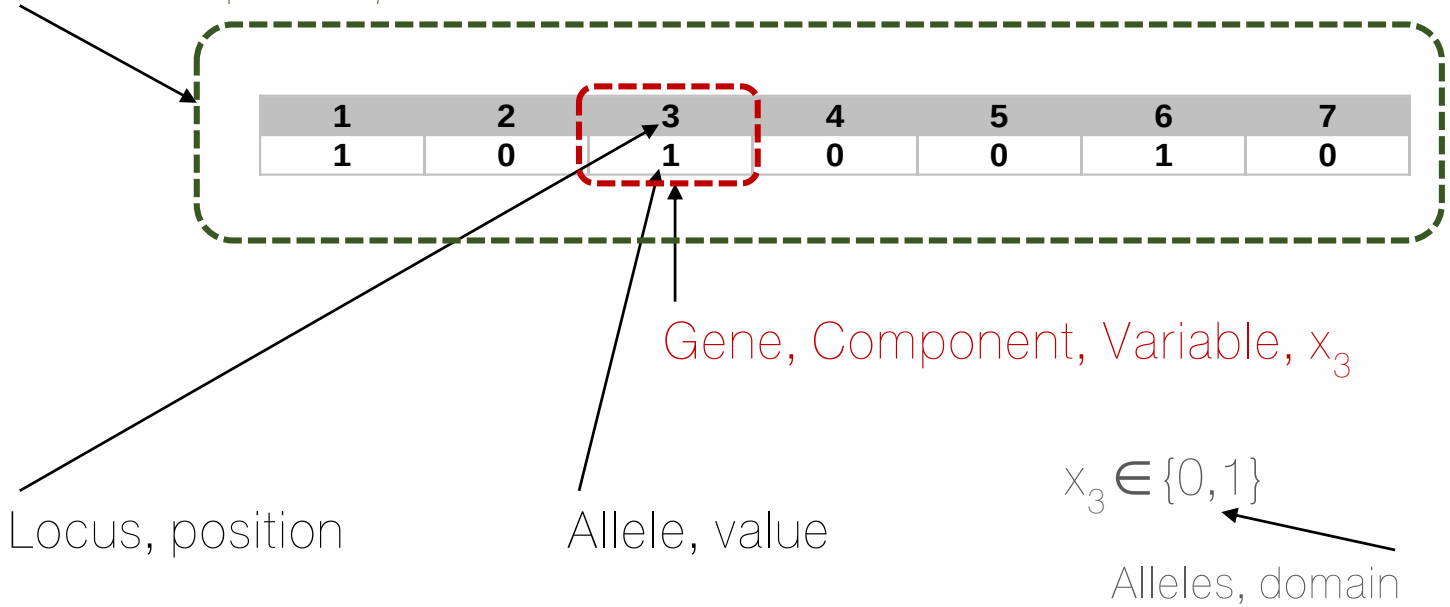
Genotype Population

(parent candidates = A1, A2, A3, A4)

https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

Classical GA: Binary Chromosomes

Chromosome, component vector, vector, string, solution, individual $x=(x_1, \ldots, x_7)$

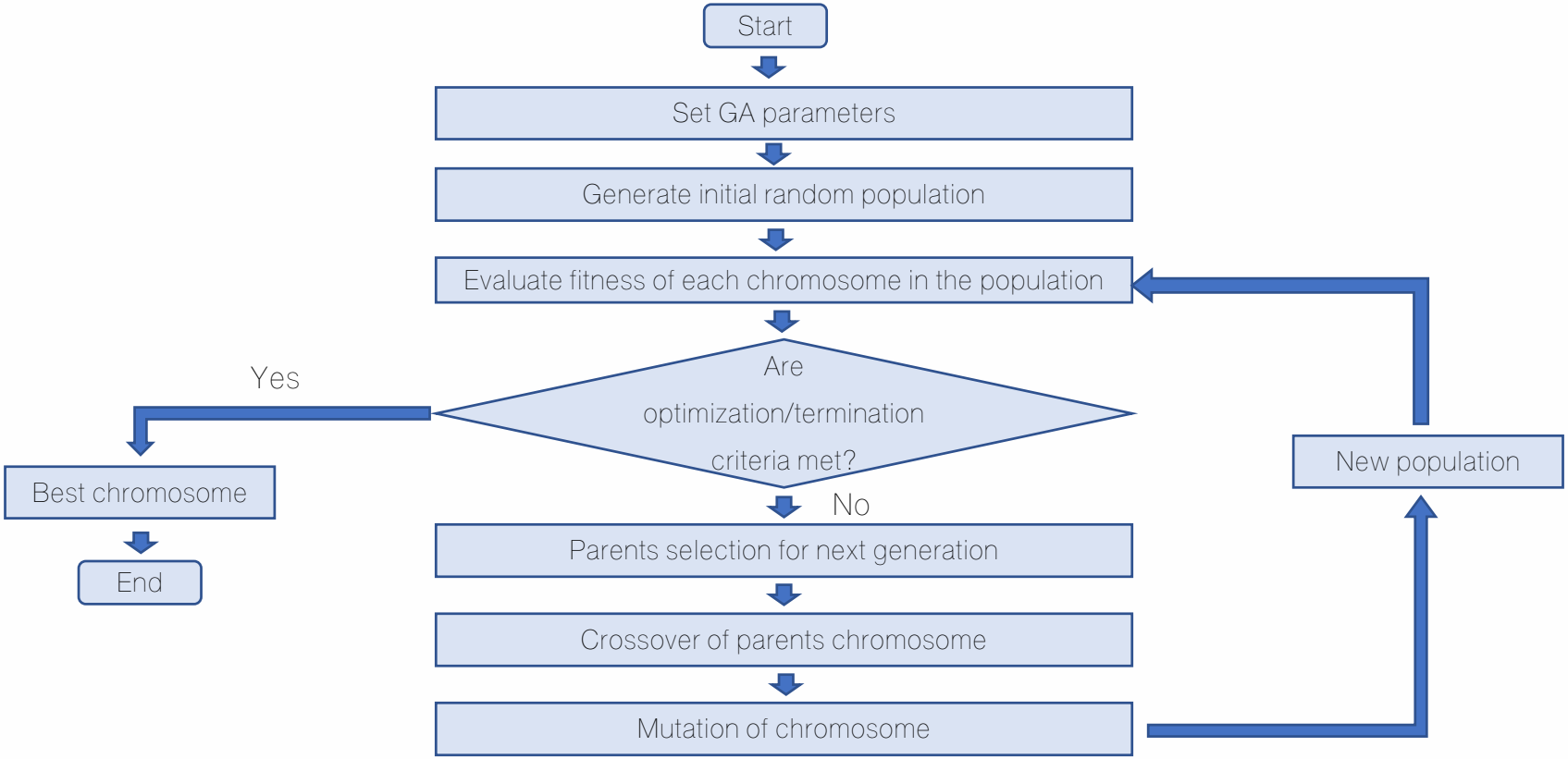


GA Pseudocode

Genetic Algorithm

```
1: Choose an initial population of chromosomes
2: while stopping criterion not met do
3:    while sufficient offspring has not been created do
4:       if condition for crossover is satisfied then
5:          Select parent chromosomes
6:          Choose crossover parameters
7:          Perform crossover
8:       end if
9:       if condition for mutation is satisfied then
10:         Choose mutation points
11:         Perform mutation
12:      end if
13:      Evaluate fitness of offspring
14:   end while
15: end while
```

The GA Cycle of Reproduction



Components of a GA

A problem to solve, and ...

- Encoding & Decoding technique (*gene, chromosome*)
- Initialization procedure *(creation)*
- Evaluation function *(environment)*
- Selection of parents *(reproduction)*
- Genetic operators *(mutation, recombination)*
- Parameter settings *(practice and art)*

Example 1(Optimization)

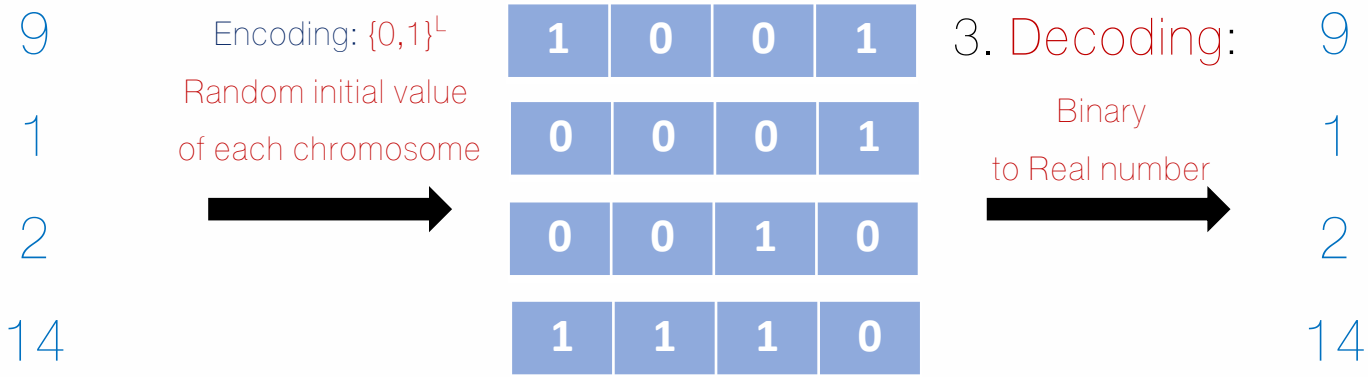
Find the optimal point of this function: $f(x) = 15x - x^2$

- $0 < x < 15$
- If $x = 15$ then the results is 0
- If $x = 0$ then the results is also 0
- And we exlcude the negative results.

Example 1(Optimization)

Find the optimal point of this function: $f(x) = 15x - x^2$

1. Encoding: A decision variable is x, where x is a real number
2. Initialization: Initial 4 chromosomes(4 genes each) population and encode the decision variable to binary chromosomes.



Example 1(Optimization)

1	0	0	1
x	x	x	x
2^3	2^2	2^1	2^0
8	0	0	1

→ 8+1 = 9

Example 1(Optimization)

0	0	0	1
x	x	x	x
2^3	2^2	2^1	2^0
0	0	0	1

→ 1

Example 1(Optimization)

0	0	1	0
x	x	x	x
2^3	2^2	2^1	2^0
0	0	2	0

→ 2

Example 1(Optimization)

1	1	1	0
x	x	x	x
2^3	2^2	2^1	2^0
8	4	2	0

→ 8+4+2 = 14

Example 1 (Optimization)

If we decode parent chromosome now, we will have 4 results.

1	0	0	1	Decoding and apply fitness function $f(x) = 15x - x^2$	54
0	0	0	1		14
0	0	1	0		26
1	1	1	0	➡	14

Example 1 (Optimization)

4. Evaluation function: Evaluate the fitness of each chromosome in the population

$f(9) = 15(9) - (9)^2 = 54$

$f(2) = 15(2) - (2)^2 = 26$

$f(1) = 15(1) - (1)^2 = 14$

$f(14) = 15(14) - (14)^2 = 14$

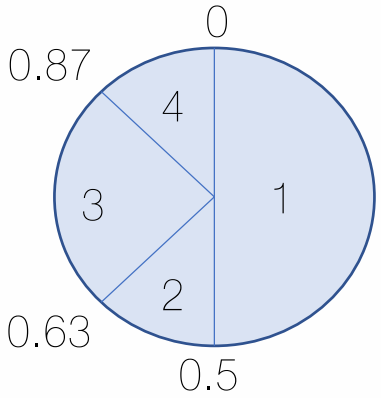
5. Selection of parents: Parents (chromosomes) selection for next-generation by using a roulette wheel. The roulette wheel is partitioned by using the ratio of fitness, and Genetic operators: crossover of parents chromosome

Chromosome 1 → 54/54+14+26+14 = 0.50

Chromosome 2 → 14/54+14+26+14 = 0.13

Chromosome 3 → 26/54+14+26+14 = 0.24

Chromosome 4 → 14/54+14+26+14 = 0.13



Example 1 (Optimization)

5. Selection of parents: Parents (chromosomes) selection for next-generation by using a roulette wheel. The roulette wheel is partitioned by using the ratio of fitness, and Genetic operators: crossover of parents chromosome.

crossover = # chromosome/2 = 4/2 = 2 pairs

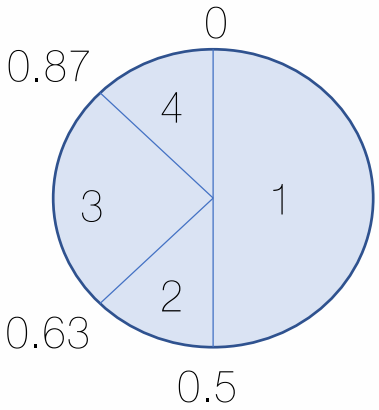
Random the number from 0:1

- random 1 = 0.1, then choose chromosome 1 is Daddy
- random 2 = 0.9, then choose chromosome 4 is Mommy

First pair

- random 3 = 0.4, then choose chromosome 1 is Daddy
- random 4 = 0.75, then choose chromosome 3 is Mommy

Second pair



Example 1 (Optimization)

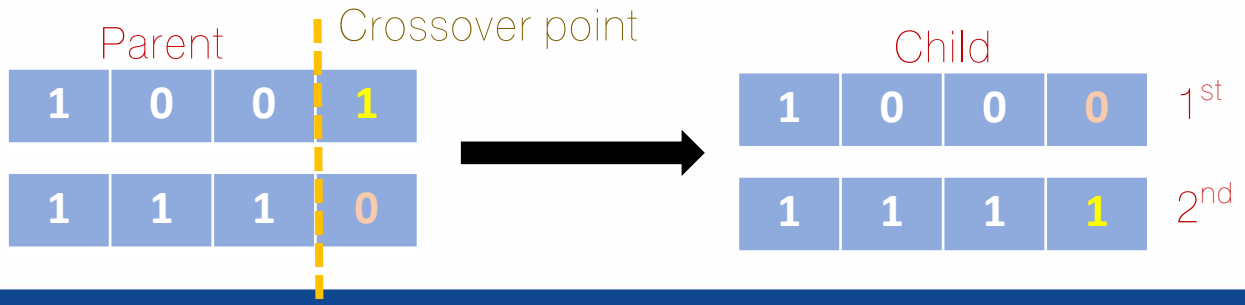
5. Selection of parents: Parents (chromosomes) selection for next-generation by using a roulette wheel. The roulette wheel is partitioned by using the ratio of fitness, and Genetic operators: crossover of parents chromosome.

Random select crossover probability (P_C=0.8) : then random choose P_c between 0 - -> 1

First pair:

a). Let this P_c = 0.7 for this pair, then < 0.8, So, we create an off-spring.

b). Choose crossover point [1:4] : at third genes (basically, they have many selecting methods)



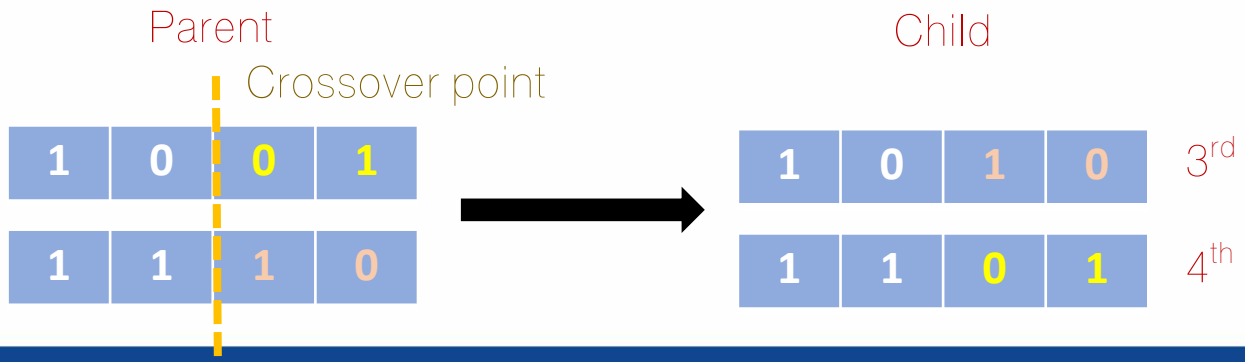
Example 1 (Optimization)

5. Selection of parents: Parents (chromosomes) selection for next-generation by using a roulette wheel. The roulette wheel is partitioned by using the ratio of fitness, and Genetic operators: crossover of parents chromosome.

Second pair:

a). Let this P_c = 0.75 for this pair, then < 0.8, So, we create off-spring.

b). Choose crossover point [1:4]: at second genes (basically, they have many selecting methods)

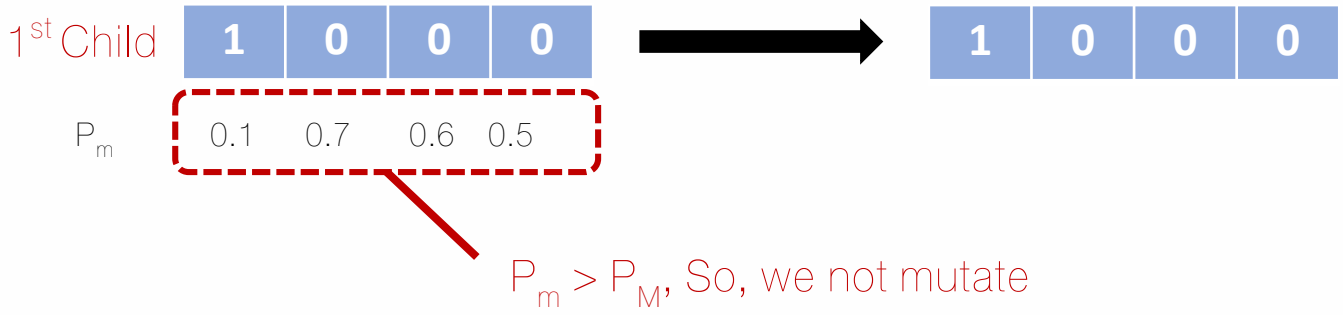


Example 1 (Optimization)

6. Genetic operators: Mutation of chromosome: Random select mutation probability that is a degree of mutation (P_M=0.05) : P_M is choose between 0 - -> 1. (less mutation < P_M < high mutation)

First child chromosome

Let random initial P_m for all genes.

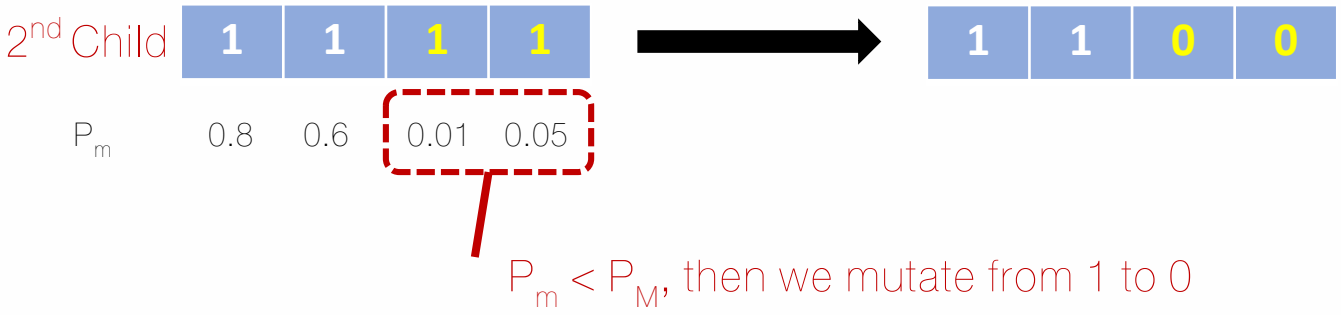


Example 1 (Optimization)

6. Genetic operators: Mutation of chromosome: Random select mutation probability that is a degree of mutation (P_M=0.05) : P_M is choose between 0 - -> 1. (less mutation < P_M < high mutation)

Second child chromosome

Let random initial P_m for all genes.

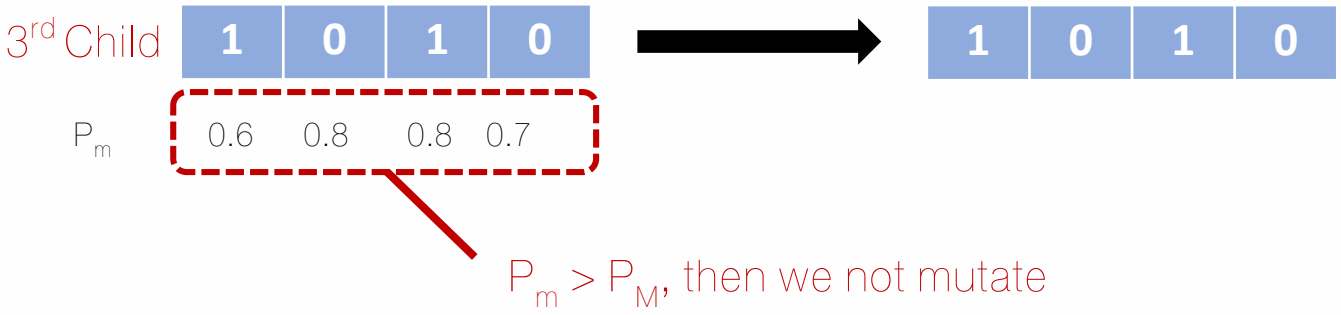


Example 1 (Optimization)

6. Genetic operators: Mutation of chromosome: Random select mutation probability that is a degree of mutation (P_M=0.05) : P_M is choose between 0 - -> 1. (less mutation < P_M < high mutation)

Third child chromosome

Let random initial P_m for all genes.

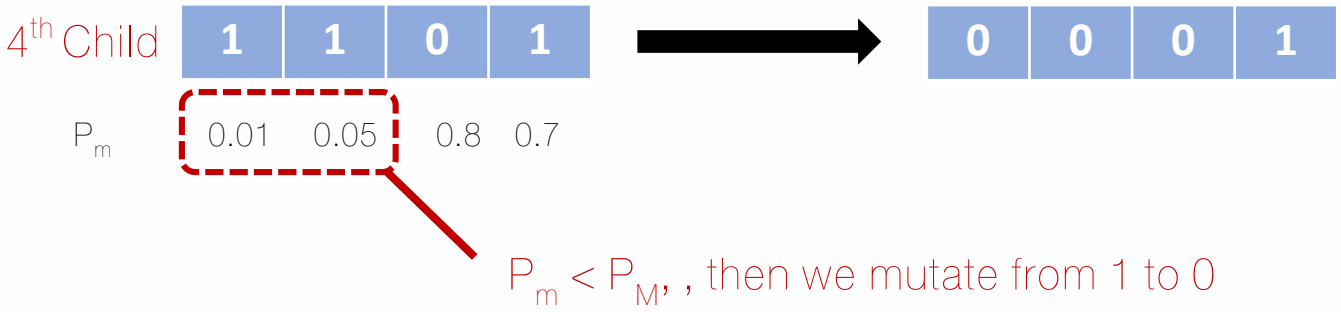


Example 1 (Optimization)

6. Genetic operators: Mutation of chromosome: Random select mutation probability that is a degree of mutation (P_M=0.05) : P_M is choose between 0 - -> 1. (less mutation < P_M < high mutation)

Fourth child chromosome

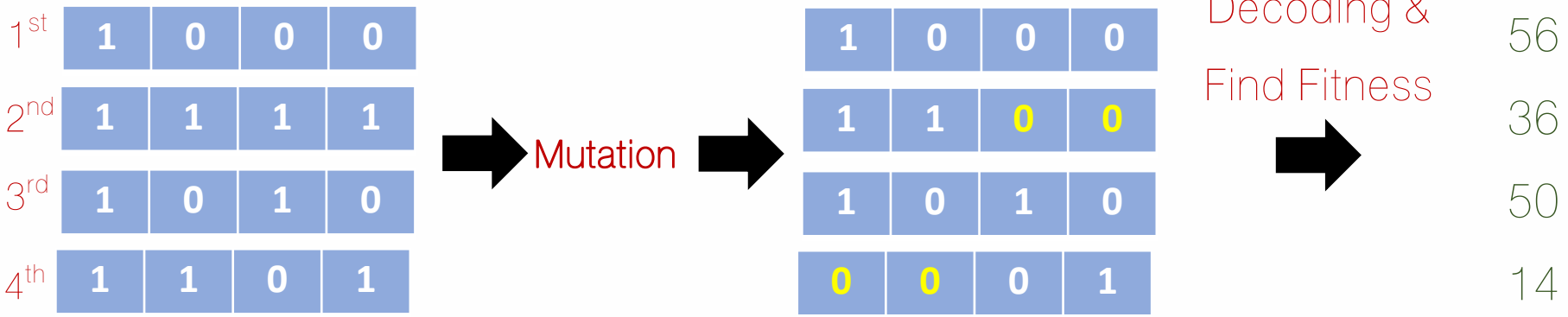
Let random initial P_m for all genes.



Example 1 (Optimization)

6. Genetic operators: Mutation of chromosome: Random select mutation probability that is a degree of mutation (P_M=0.05) : P_M is choose between 0 - -> 1. (less mutation < P_M < high mutation)

the bellows chromosomes have already mutated



Example 1 (Optimization)

7. Sort parent chromosomes and child chromosomes (pool to population)

1	0	0	0	56	✓
1	0	0	1	54	✓
1	0	1	0	50	✓
1	1	0	0	36	✓
0	0	1	0	26	✗
1	1	1	0	14	✗
0	0	0	1	14	✗
0	0	0	1	14	✗

8. Are optimization / termination criteria met?

a). Meet the initial value of iteration

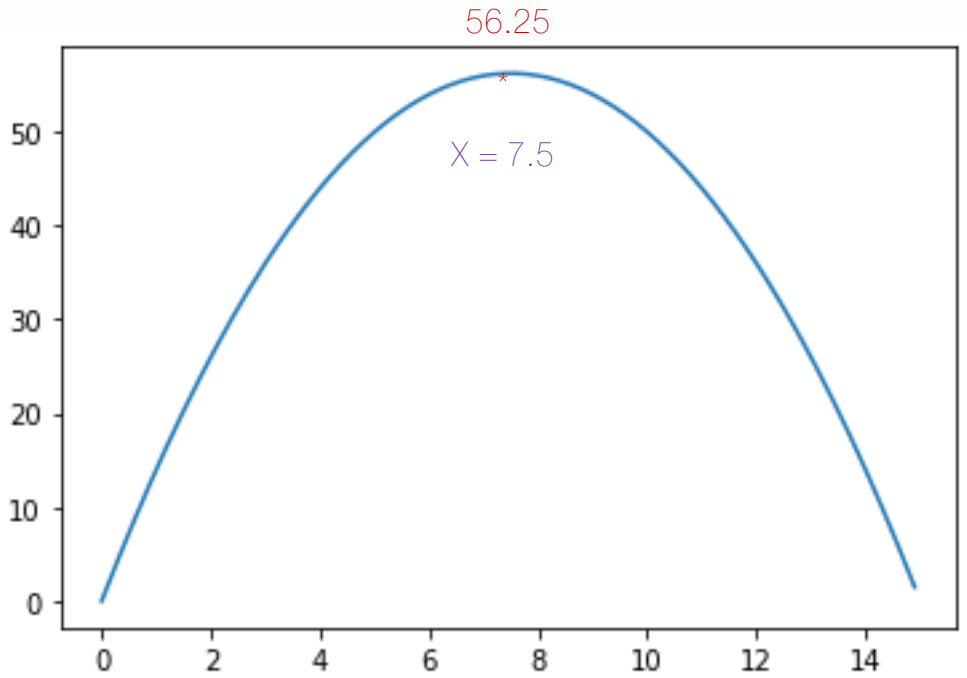
b). The fitness values of a new population (chromosomes) are not increase t times (t is a initial value)

c). A chromosome with highest fitness value is 1.80 time of a poorest chromosome

Example 1(Optimization)

Find the optimal point of this function: $f(x) = 15x - x^2$

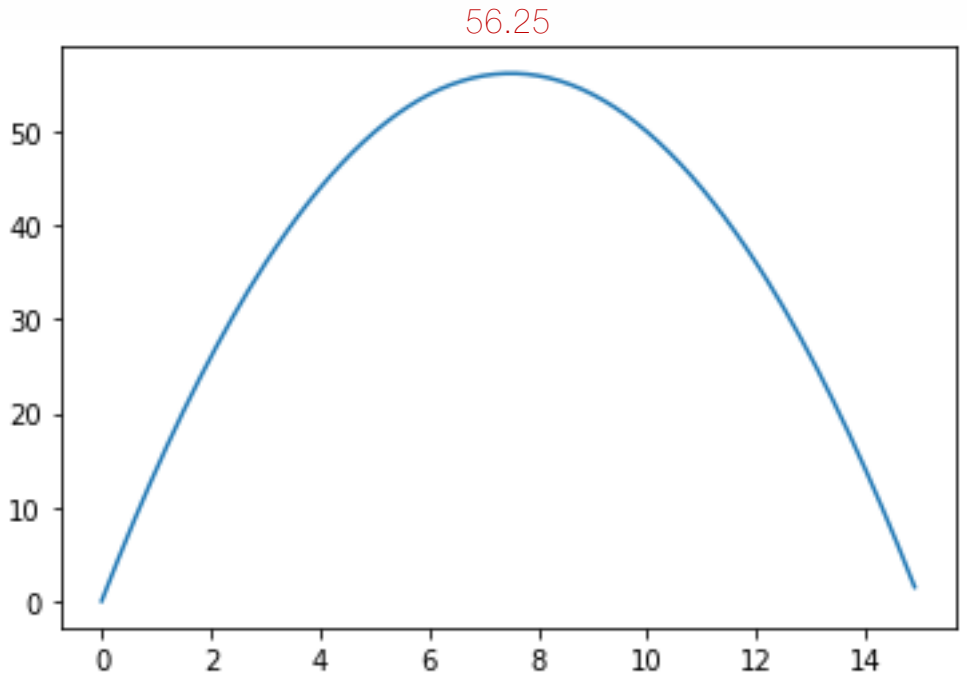
- $0 < x < 15$
- If $x = 15$ then the results is 0
- If $x = 0$ then the results is also 0
- And we exlcude the negative results.



Example 1 (Optimization)

GA will to find the value of x the close to optimal with less computational times

1	0	0	0	56



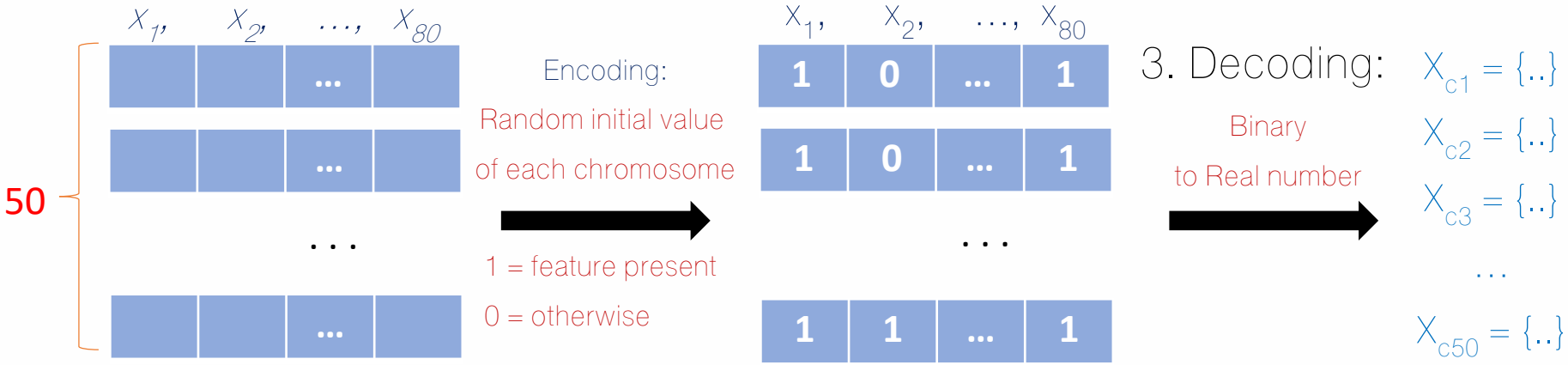
SGA technical summary tableau

Representation	Binary strings
Recombination	N-point or uniform
Mutation	Bitwise bit-flipping with fixed probability
Parent selection	Fitness-Proportionate
Survivor selection	All children replace parents
Speciality	Emphasis on crossover

Example 2 (Features Selection)

For colon risk prediction model that have 80 features and 900 samples. we will set chromosome length = 80 genes. (genes = independent variables)

1. Let assume feature name are $x_1, x_2, \ldots, x_{80}$
2. In our case, we initial the number of chromosomes = 50 sets



Example 2 (Features Selection)

4. Evaluate the fitness of each chromosome in the population

X_{c1} -> trains a classifier with training set(900 samples)+evaluate with testing set -> (1-Error rate)

X_{c2} -> trains a classifier with training set(900 samples)+evaluate with testing set -> (1-Error rate) ....

X_{c50} -> trains a classifier with training set(900 samples)+evaluate with testing set -> (1-Error rate)

Do the same from Example 1 for step 5. Parents (chromosomes) selection for next-generation + Crossover, step 6. Mutation, step 7. Sort parent chromosomes and child chromosomes, and step 8. Termination criteria.

Example 2 (Features Selection)

We select the best chromosomes, means the best combination of features at each cycle

X_1,	X_2,	…,	X_{80}	(1-Error rate)	
1	0	…	1	0.95	✓
1	0	…	0	0.94	✗
1	0	…	0	0.89	✗
0	0	…	0	0.86	✗
1	1	…	0	0.75	✗
0	0	…	1	0.73	✗
….					✗
1	1	…	1	0.40	✗

Example 2 (Features Selection)

You can apply the final chromosomes to rank the features (relevant feature)

	X_1,	X_2,	…,	X_{80}	(1-Error rate)
Top 10	1	0	…	1	0.95
	1	0	…	0	0.94
	1	0	…	0	0.89
	0	0	…	0	0.86
	1	1	…	0	0.75
	0	0	…	1	0.73
	….				
	1	1	…	1	0.40

Count the frequency of each feature that appeared in top 10 chromosomes. And find it proportion ratio.

$$f_i = \sum_i^{80} \frac{f_i}{f_i + \cdots + f_{80}}$$

$i = \{1, 2, \ldots, 80\}$

Ex. f_1 = 8/(8+…..) = 0.10

Pros and Cons of GA

Pros:

1. Faster than other algorithms.
2. Easier. If vector representation of individual is right, we can find out a solution without a deep analysis work.

Pros and Cons of GA

Cons:

1. The random, sometimes doesn't find the optimum.
2. It is not a complete algorithm (not always the algorithm finds a suitable solution). Sometimes it can get stuck with a local maximum problem. Nevertheless, crossover operation (we will point out further down) helps to mitigate it, although this implies more iterations.

Genetic Algorithm:
Binary Example in Python
(Code reading practice)

https://pythonhealthcare.org/2018/10/01/94-genetic-algorithms-a-simple-genetic-algorithm/

Genetic Algorithm: Binary Example in Python

In this example we will look at a basic genetic algorithm (GA). We will set up the GA to try to match a pre-defined ‘optimal. solution. Often with GAs we are using them to find solutions to problems which

1) cannot be solved with ‘exact’ methods (methods are guaranteed to find the best solution)

2) where we cannot recognize when we have found the optimal solution. GAs therefore fall into a collection of algorithms called heuristic (from Greek for ‘search’) algorithms that hunt down good solutions, without us knowing how far off the theoretical optimal solution they are.

Genetic Algorithm: Binary Example in Python

Open file bga_example.py

```
1 import random
2 import numpy as np
3
4 def create_starting_population(individuals, chromosome_length):
5     # Set up an initial array of all zeros
6     population = np.zeros((individuals, chromosome_length))
7     # Loop through each row (individual)
8     for i in range(individuals):
9         # Choose a random number of ones to create
.0        ones = random.randint(0, chromosome_length)
.1        # Change the required number of zeros to ones
.2        population[i, 0:ones] = 1
.3        # Sfuffle row
.4        np.random.shuffle(population[i])
.5
.6    return population
```

Genetic Algorithm: Binary Example in Python

```
import random
import numpy as np

def create_starting_population(individuals, chromosome_length):
    # Set up an initial array of all zeros
    population = np.zeros((individuals, chromosome_length))
    # Loop through each row (individual)
    for i in range(individuals):
        # Choose a random number of ones to create
        ones = random.randint(0, chromosome_length)
        # Change the required number of zeros to ones
        population[i, 0:ones] = 1
        # Sfuffle row
        np.random.shuffle(population[i])

    return population
```

Genetic Algorithm: Binary Example in Python

```
def calculate_fitness(population):

    fitness_scores = np.zeros(len(population))
    for i in range(len(population)):
        b = population[i]
        a = b.dot(2**np.arange(b.size)[::-1])
        fitness_scores[i] = (15*a) - (a*a)

    return fitness_scores
```

Genetic Algorithm: Binary Example in Python

```
def select_individual_by_tournament(population, scores):
    # Get population size
    population_size = len(scores)
    # Pick individuals for tournament
    fighter_1 = random.randint(0, population_size-1)
    fighter_2 = random.randint(0, population_size-1)
    # Get fitness score for each
    fighter_1_fitness = scores[fighter_1]
    fighter_2_fitness = scores[fighter_2]
    # Identify undividual with highest fitness,  Fighter 1 will win if score are equal
    if fighter_1_fitness >= fighter_2_fitness:
        winner = fighter_1
    else:
        winner = fighter_2

    # Return the chromsome of the winner
    return population[winner, :]
```

Genetic Algorithm: Binary Example in Python

```
def breed_by_crossover(parent_1, parent_2):
    # Get length of chromosome
    chromosome_length = len(parent_1)

    # Pick crossover point, avoding ends of chromsome
    crossover_point = random.randint(1,chromosome_length-1)

    # Create children. np.hstack joins two arrays
    child_1 = np.hstack((parent_1[0:crossover_point],
                         parent_2[crossover_point:]))

    child_2 = np.hstack((parent_2[0:crossover_point],
                         parent_1[crossover_point:]))

    # Return children
    return child_1, child_2
```

Genetic Algorithm: Binary Example in Python

```
def randomly_mutate_population(population, mutation_probability):

    # Apply random mutation
    random_mutation_array = np.random.random(
        size=(population.shape))

    random_mutation_boolean = \
        random_mutation_array <= mutation_probability

    population[random_mutation_boolean] = \
    np.logical_not(population[random_mutation_boolean])

    # Return mutation population
    return population
```

Genetic Algorithm: Binary Example in Python

```
# Set general parameters
chromosome_length = 4
population_size = 8
maximum_generation = 50
best_score_progress = [] # Tracks progress

# Create starting population
population = create_starting_population(population_size, chromosome_length)

# Display best score in starting population
scores = calculate_fitness(population)
best_score = np.max(scores)
print ('Starting best score, % target: ',best_score)

# Add starting best score to progress tracker
best_score_progress.append(best_score)
```

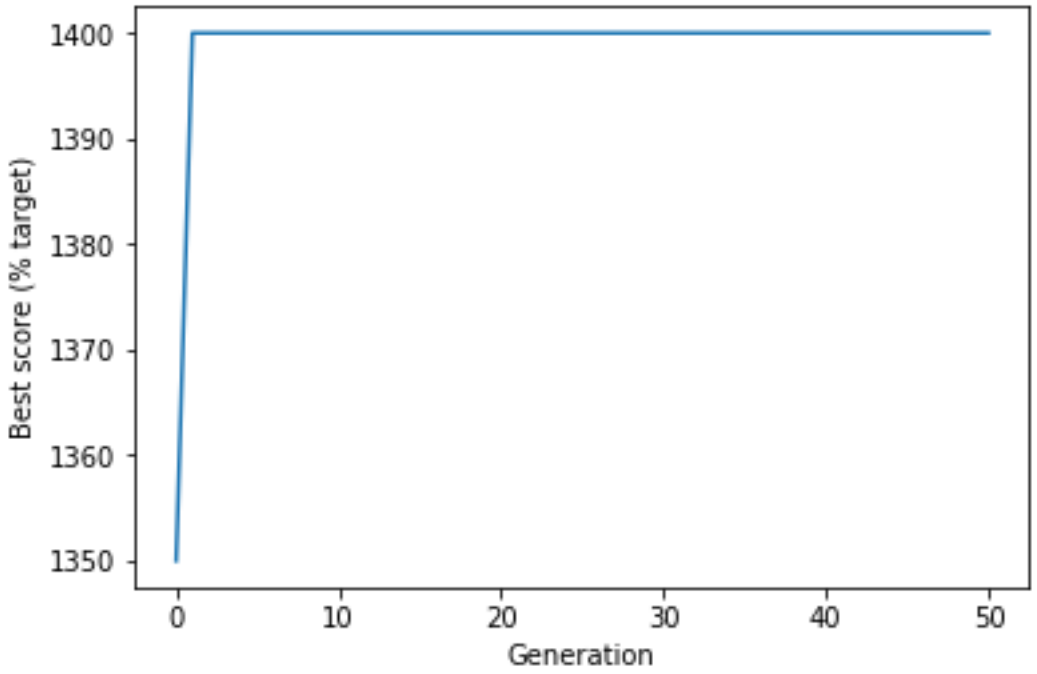
Genetic Algorithm: Binary Example in Python

```
# Now we'll go through the generations of genetic algorithm
for generation in range(maximum_generation):
    # Create an empty list for new population
    new_population = []
    # Create new popualtion generating two children at a time
    for i in range(int(population_size/2)):
        parent_1 = select_individual_by_tournament(population, scores)
        parent_2 = select_individual_by_tournament(population, scores)
        child_1, child_2 = breed_by_crossover(parent_1, parent_2)
        new_population.append(child_1)
        new_population.append(child_2)
    # Replace the old population with the new one
    population = np.array(new_population)
    # Apply mutation
    mutation_rate = 0.01
    population = randomly_mutate_population(population, mutation_rate)
    # Score best solution, and add to tracker
    scores = calculate_fitness(population)
    best_score = np.max(scores)
    best_score_progress.append(best_score)
```

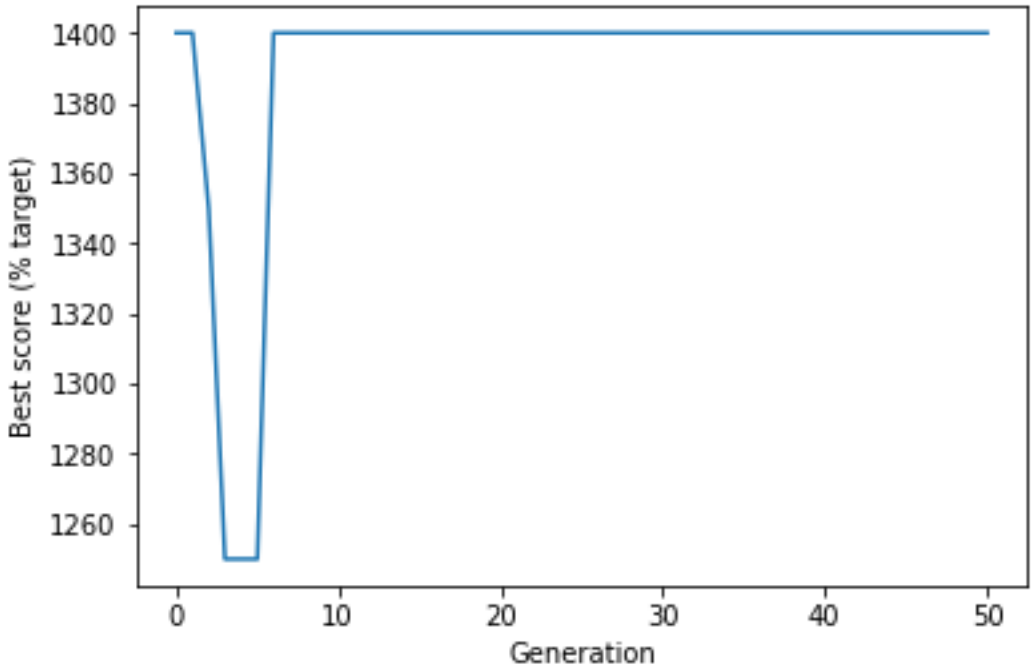
Genetic Algorithm: Binary Example in Python

```
# GA has completed required generation
print ('End best score, % target: ', best_score)

# Plot progress
import matplotlib.pyplot as plt
plt.plot(best_score_progress)
plt.xlabel('Generation')
plt.ylabel('Best score (% target)')
plt.show()
```

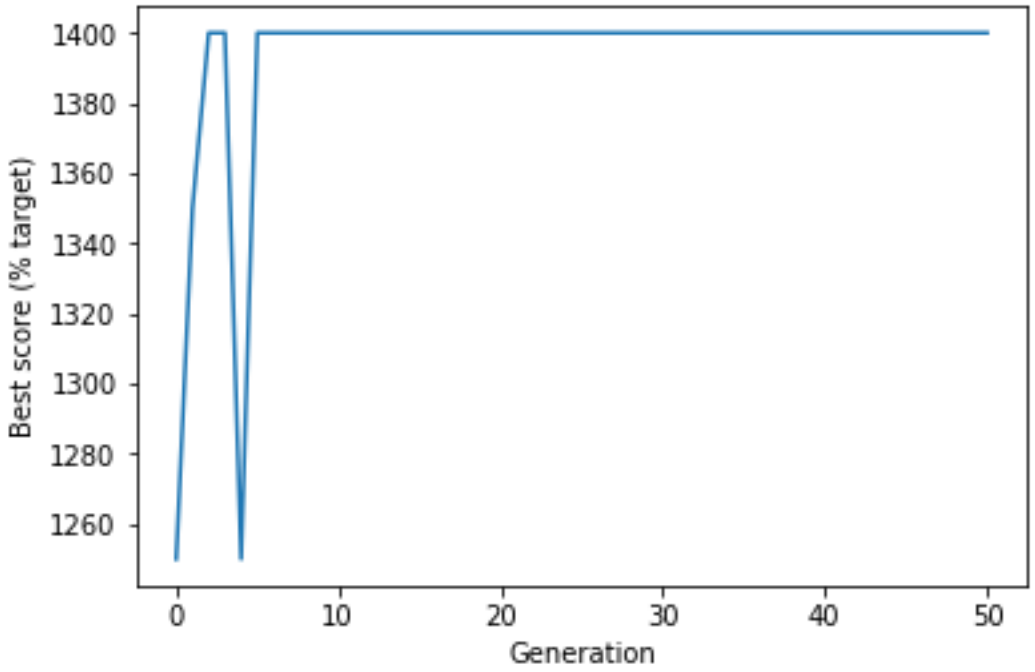
Genetic Algorithm: Binary Example in Python



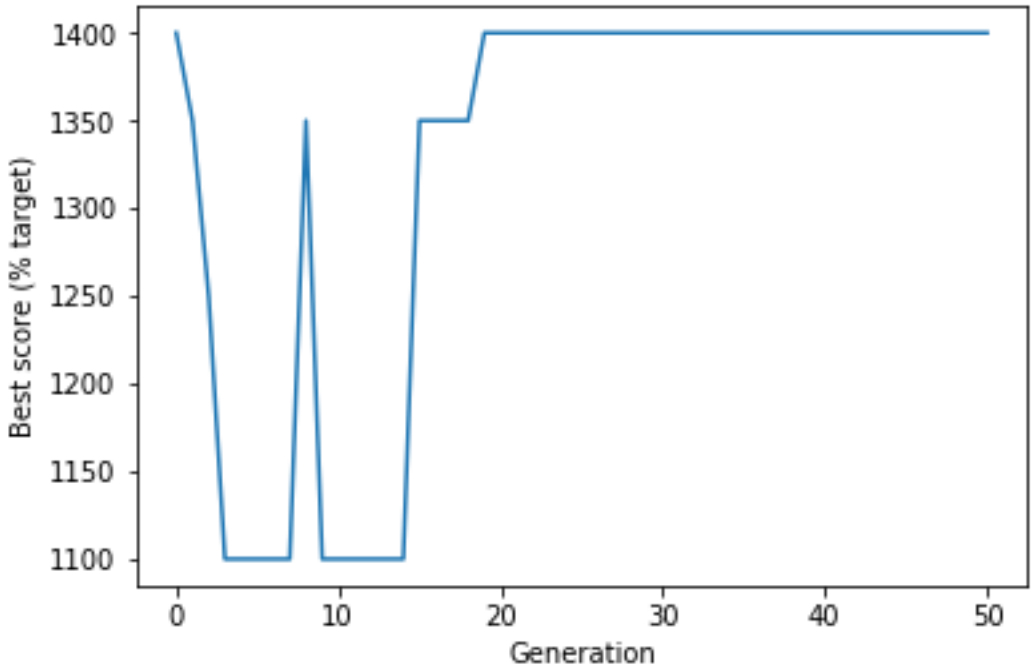
Genetic Algorithm: Binary Example in Python



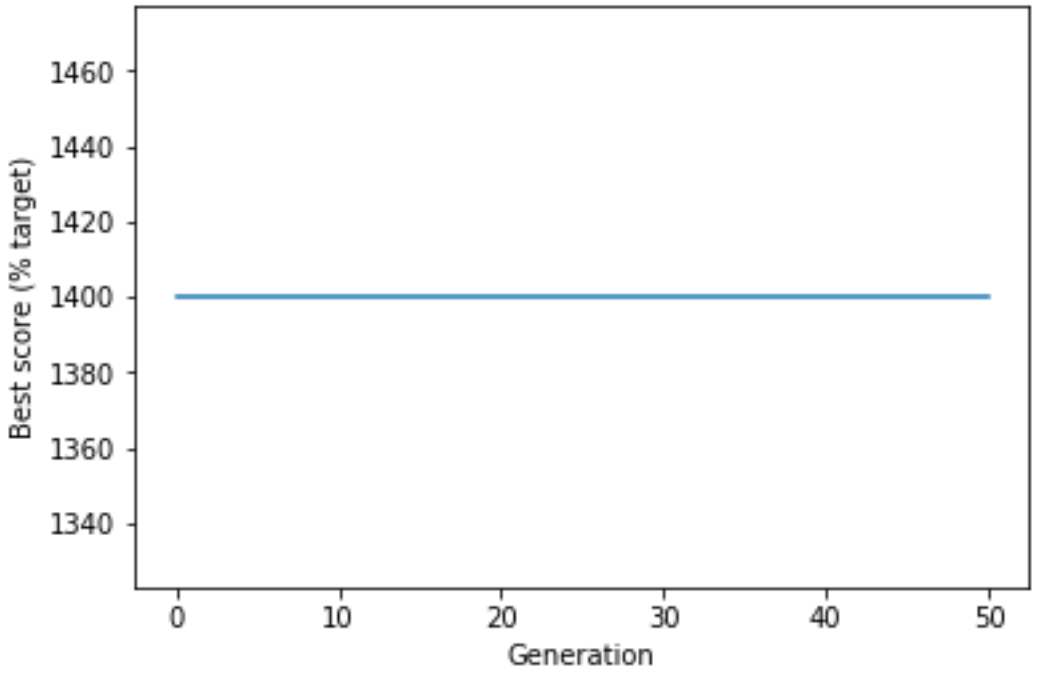
Genetic Algorithm: Binary Example in Python



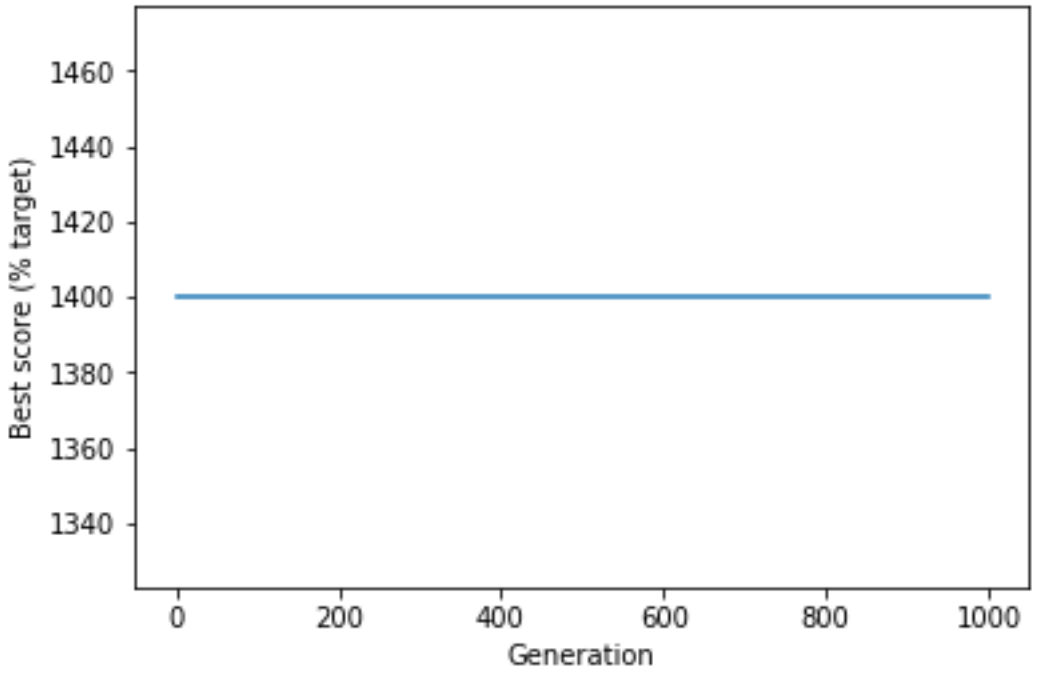
Genetic Algorithm: Binary Example in Python



Genetic Algorithm: Binary Example in Python



Genetic Algorithm: Binary Example in Python



Practice with Python

https://towardsdatascience.com/genetic-algorithm-implementation-in-python-5ab67bb124a6

Practice with Python

```
import sys

sys.path.append(r'H:/xxxxxxxxxxxxxxx')

import numpy

import GA as ga


"""

The y=target is to maximize this equation:

    y = w1x1+w2x2+w3x3+w4x4+w5x5+w6x6

    where (x1,x2,x3,x4,x5,x6)=(4,-2,3.5,5,-11,-4.7)

    What are the best values for the 6 weights w1 to w6?

    We are going to use the genetic algorithm for the best possible values after a number of generations.

"""
```

Practice with Python

```
# Inputs of the equation.
equation_inputs = [4,-2,3.5,5,-11,-4.7]


# Number of the weights we are looking to optimize.
num_weights = len(equation_inputs)

"""
Genetic algorithm parameters:
    Mating pool size
    Population size
"""
sol_per_pop = 8
num_parents_mating = 4
```

the number of genes per chromosomes,
inputs = 6 then number of genes = 6

the number of chromosomes or solutions per population

Practice with Python

```
# Defining the population size.

pop_size = (sol_per_pop,num_weights)

# The population will have sol_per_pop chromosome where each chromosome has
num_weights genes.


#Creating the initial population.

new_population = numpy.random.uniform(low=-4.0, high=4.0, size=pop_size)

print(new_population)
```

Random the number between -4 : 4, it will be of shape (8, 6)

8 chromosomes and each one has 6 genes

Practice with Python

print(new_population)

	0	1	2	3	4	5
0	0.289414	-2.75222	-2.19136	3.37397	-3.93658	2.03454
1	-1.68041	-2.17578	0.530518	-3.21558	-2.73432	1.34845
2	2.34469	-2.26015	2.5806	0.609567	-0.470554	-2.13787
3	1.16707	-3.6603	-3.86874	-2.31917	1.77777	-0.419897
4	1.23874	0.508638	-0.353144	1.13417	-0.78586	-0.41118
5	-3.68861	-2.66044	1.10824	-1.05021	-3.51416	1.30239
6	1.52446	0.668088	-1.21606	-2.52879	-0.65993	1.65469
7	-3.7708	2.11813	0.0882154	-2.75555	-3.32023	1.30451

Practice with Python

```
best_outputs = []
num_generations = 50
for generation in range(num_generations):
    #start for loop
    print("Generation : ", generation)
    # Measuring the fitness of each chromosome in the population.
    fitness = ga.cal_pop_fitness(equation_inputs, new_population)
    print("Fitness")
    print(fitness)
```

the number of generations = 50

Find the fitness value

Practice with Python

```
best_outputs.append(numpy.max(numpy.sum(new_population*equation_inputs, axis=1)))
```

Find max of the summation of $x_i * w_i$

Append to array([x, x, x, ..., x, x, x])

```
# The best result in the current iteration.
print("Best result : ", numpy.max(numpy.sum(new_population*equation_inputs, axis=1)))


# Selecting the best parents in the population for mating.
parents = ga.select_mating_pool(new_population, fitness,
                                num_parents_mating)
print("Parents")
print(parents)
```

Practice with Python

```
# Generating next generation using crossover.
offspring_crossover = ga.crossover(parents,
                                   offspring_size=(pop_size[0]-parents.shape[0], num_weights))
print("Crossover")
print(offspring_crossover)
```

Start from first index of chromosomes of parents

	0	1	2	3	4	5
0	0.969319	-2.32143	274.976	2.57639	-3.69265	-333.061
1	0.969319	-2.32143	274.879	2.57639	-3.69265	-332.182
2	0.969319	-2.32143	275.96	2.57639	-3.69265	-333.668
3	0.969319	-2.32143	272.793	2.57639	-3.69265	-334.171

Practice with Python

```
# Adding some variations to the offspring using mutation.
offspring_mutation = ga.mutation(offspring_crossover, num_mutations=2)
print("Mutation")
print(offspring_mutation)
```

Start mutation of offspring by using index of mutation = 2

Mutation changes a number of genes as defined by the num_mutations argument

	0	1	2	3	4	5
0	0.969319	-2.32143	274.976	2.57639	-3.69265	-333.061
1	0.969319	-2.32143	274.879	2.57639	-3.69265	-332.182
2	0.969319	-2.32143	275.96	2.57639	-3.69265	-333.668
3	0.969319	-2.32143	272.793	2.57639	-3.69265	-334.171

Practice with Python

```
    # Creating the new population based on the parents and offspring.
    new_population[0:parents.shape[0], :] = parents
    new_population[parents.shape[0]:, :] = offspring_mutation
#end for loop
```

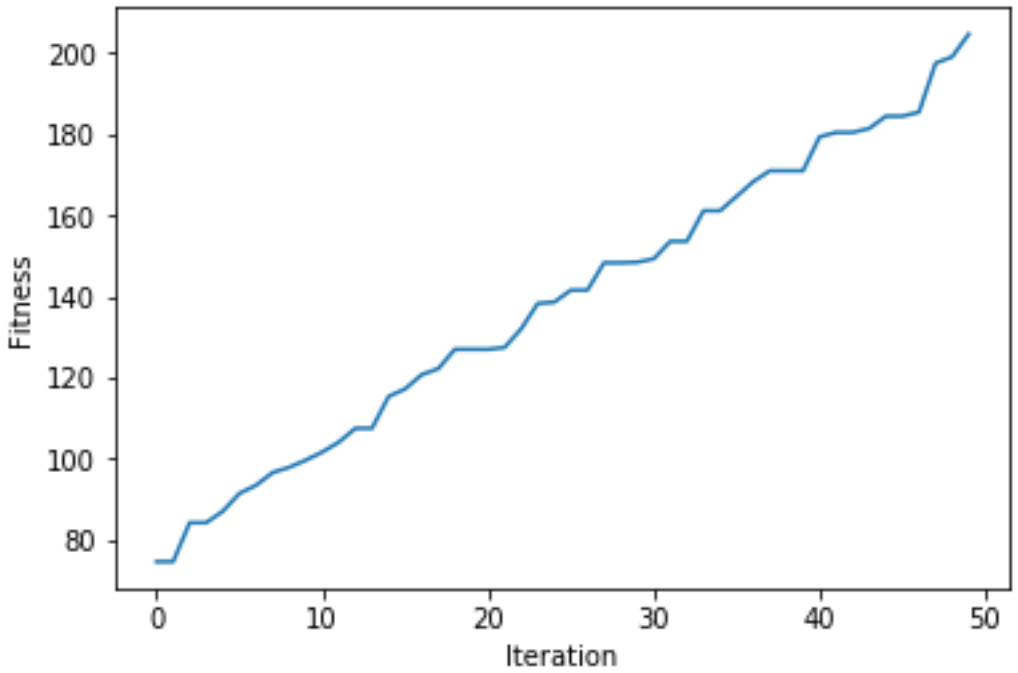
Practice with Python

```
# Getting the best solution after iterating finishing all generations.

#At first, the fitness is calculated for each solution in the final generation.

fitness = ga.cal_pop_fitness(equation_inputs, new_population)

# Then return the index of that solution corresponding to the best fitness.

best_match_idx = numpy.where(fitness == numpy.max(fitness))


print("Best solution : ", new_population[best_match_idx, :])

print("Best solution fitness : ", fitness[best_match_idx])
```

Practice with Python

```
import matplotlib.pyplot

matplotlib.pyplot.plot(best_outputs)

matplotlib.pyplot.xlabel("Iteration")

matplotlib.pyplot.ylabel("Fitness")

matplotlib.pyplot.show()
```

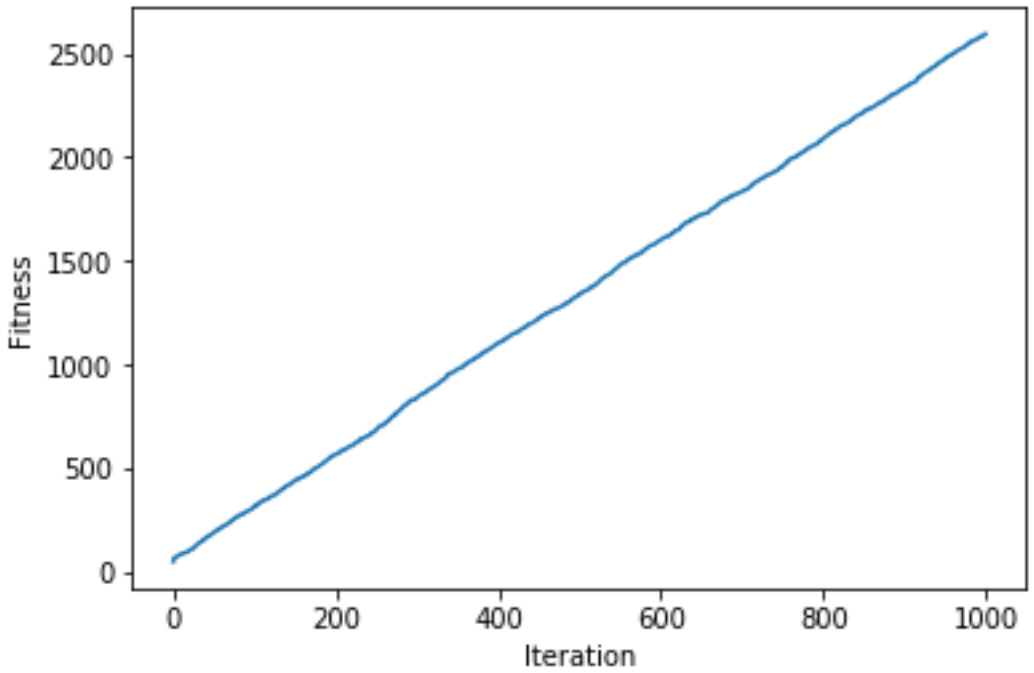
Practice with Python



Best solution : [[[-0.45324199 2.06439104 18.91093727 1.57189856 -3.89165109 -19.96822908]]]

Best solution fitness : [204.76486188]

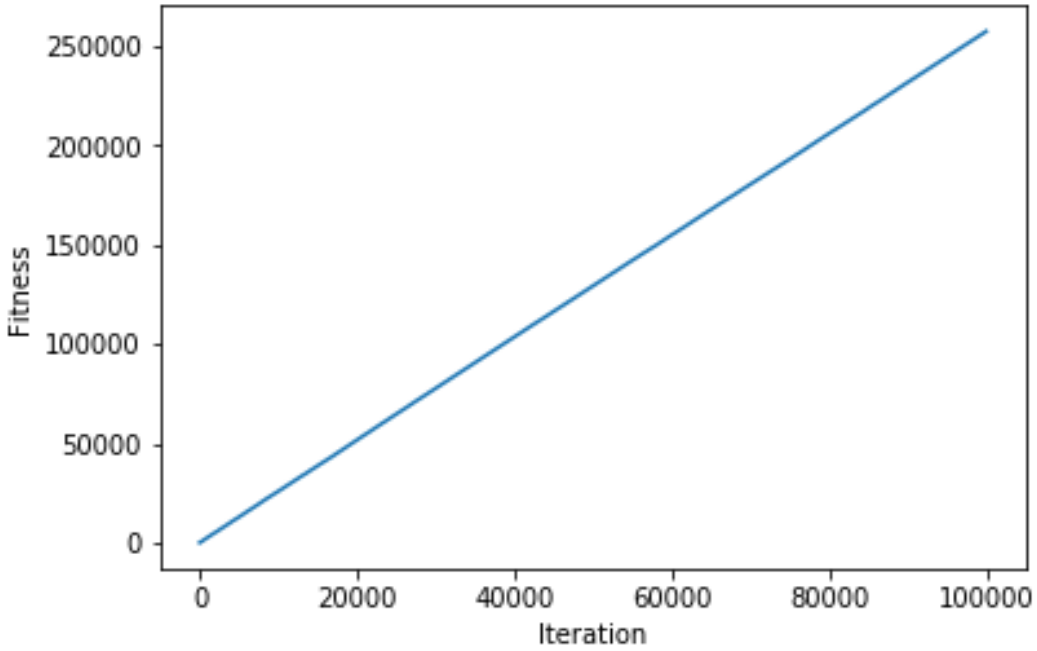
Practice with Python



Best solution : [[[0.79591366 0.6671946 284.60828432 2.91541882 -2.69427761 -330.17839578]]]

Best solution fitness : [2594.03086848]

Practice with Python



Best solution : [[[3.37599087e+00 7.22642927e-01 2.93093095e+04 2.61024369e+00 -3.40993887e+00 -3.28859556e+04]]]

Best solution fitness : [257209.19387975]

In-class Assignment:

Please modify code bga_example.py to get the results (X) more closer to the optimal value (56.25).

Hint:

- Modify the number of the parent chromosomes or the number of genes in each chromosome
- Modify fitness calculation

Assignment:

due on November 15, 2022 (20 points)

$$Y = w_1 x_1^2 + w_2 x_2^3 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

The equation has 6 inputs (x_1 to x_6) and 6 weights (w_1 to w_6) as shown and inputs values are (x_1, x_2, x_3, x_4, x_5, x_6) = (4, -2, 7, -5, 11, 1).

We are going to find the parameters (weights) that maximize such equation. If P_C=0.8 and P_M=0.1, then

1) Find the best weights at generation-2 by manual and start with 8 chromosomes (10 points)

2) Find the best weights at generation-1000 by using Python and start with 50 chromosomes (10 points)

References

1. *Anke Meyer-Baese, Volker Schmid, in Pattern Recognition and Signal Analysis in Medical Imaging (Second Edition), 2014.*
2. *Xin-She Yang, in Metaheuristics in Water, Geotechnical and Transport Engineering, 2013.*
3. *Jasbir S. Arora, in Introduction to Optimum Design (Third Edition), 2012.*