EVOLUTIONARY COMPUTATION ALGORITHM

WEEK #1

MENGSAY'S NOTES

CONTENT

1. Optimization Problem and Evalutionary Computing
2. Genetic Algorithm (GA)
3. Ant Colony Optimization (ACO)
4. Artificial Bee Colony Algorithm (ABC)
5. Particle Swarm Optimization (PSO)
6. Firefly Algorithm
7. Bat Algorithm
8. Cuckoo Search
9. Harmony Search

OPTIMIZATION PROBLEM

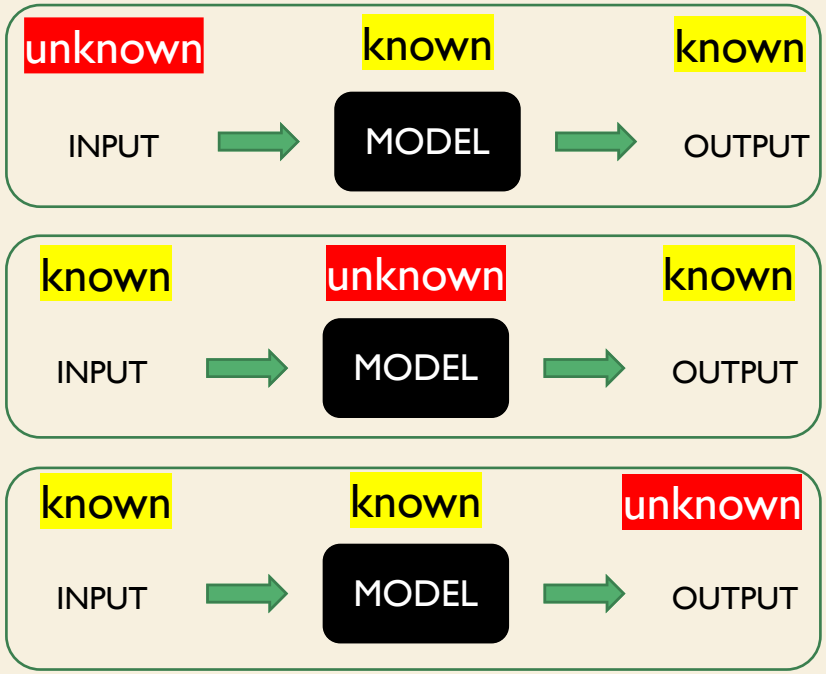
EVALUATIONARY COMPUTATION ALGORITHM

A BLACK BOX SYSTEM

- Consider a black box view of systems with three components:
  - Input, Model, and Output
  - Example: A voice control system for smart home
    - Input: Electrical Signal produced from microphone
    - Output: Heating system, TV set, or lights
    - Model: Mapping from patterns in electrical waveforms from audio input onto the outputs

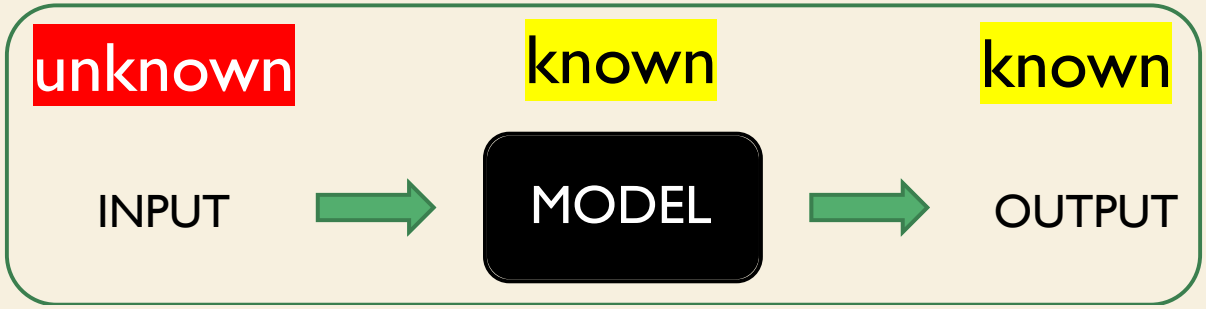
TYPE OF PROBLEMS

- Based on the black box system concept, we can consider three type of problems
- Optimization
  - Travelling salesman problem
- Modeling
  - To find a formulate relation between input(E.g. gold price…) and the Dow Jones index
- Simulation
  - Weather forecast system



OPTIMIZATION

- Here we mainly focus on Optimization Problems
- Optimization
  - Model is known
  - (Description of) Desired output
  - The task is to find the input leading to desired output



OPTIMIZATION

- Example: Travelling Salesman Problem
  - Input: Sequence of cities
  - Model: A formula that for each given input will compute the output
  - Output: Length of the tour
- The problem is to find a sequence of cities with optimal length.

CONSTRAINT SATISFACTION

- In optimization problem, a set of constraint(s) is considered.
- Many possibilities that satisfy the constraint are called **Candidate solutions**.
- When a solution to the problem can be represented by combination or permutation, the problem is called **Combinatorial Optimization Problem**.
- Among the candidate solutions, the best one is called **Optimal solution**.
- Function to calculate numerical value represented the quality of a candidate solution is called **Objective Fuction**.

KNAPSACK PROBLEM

- Consider a problem of choosing a set of multiple items and pack into a knapsack with capacity B.
- Assume i-th item has its own size w_i and value v_i.
- The task is to find a set of items (N) which maximizes the value of all chosen items.
- Constraint: $\sum_{i \in N} w_i \leq B$
- Objective Function: $\sum_{i \in N} v_i$

FLOYD PROBLEM

- By Robert W. Floyd
- Consider a division of a given set $\{\sqrt{1}, \sqrt{2}, \cdots, \sqrt{50}\} = A \cup B\ ,\ A \cap B = \emptyset$
- The task is to define A , B which minimizes the absolute value of difference between sum of elements in each set.

$$\text{objective function} = \left| \sum_{a \in A} a - \sum_{b \in B} b \right|$$

TRAVELLING SALESMAN

- Travelling salesman problem (TSP)
- Consider a situation of visiting N cities
- Constraint : To visit all cities
- Objective function: Cost of travelling

DIFFICULTY

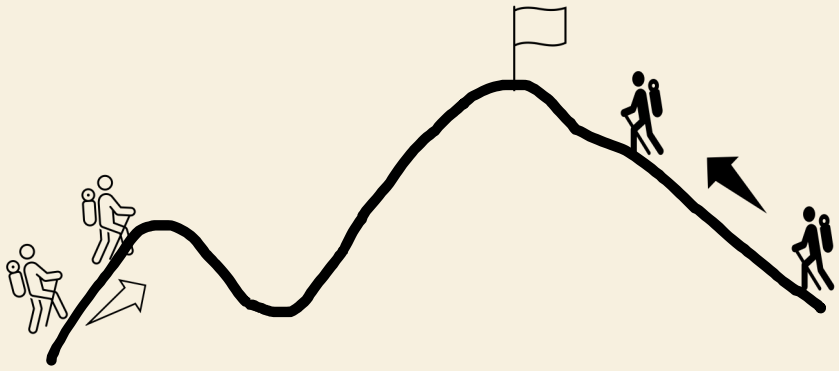
- A simple method to solve a combinatorial optimization problem is to enumerate all possible candidate solutions and check the constraint satisfaction.
- In Knapsack problem with 50 types of items, the total number of possible candidate solutions is

$$2^{50} \approx 1.125 \times 10^{15}$$

- It is not an easy task when looking at every single candidate solution.

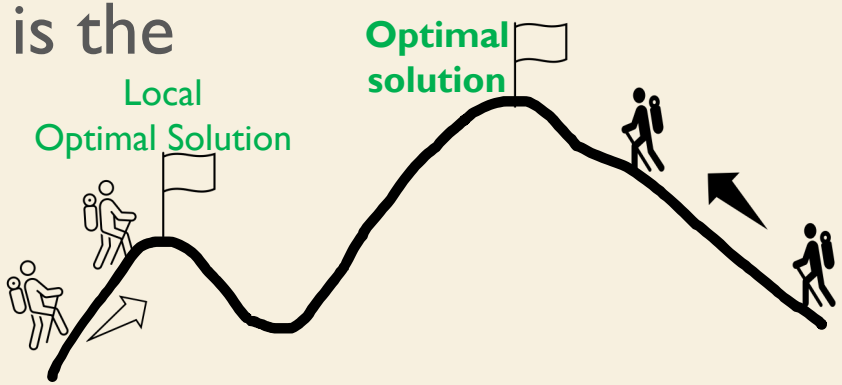
SOLUTION SEARCH METHOD

- To find the optimal solution by searching among all candidate solutions has a problem of time complexity when the number of candidates is large.
- In stead of search from all candidate, an approximation methods (e.g. Hill Climbing) are used.



HILL CLIMBING METHOD

- When you climb a hill, it is a good idea to move forward direction which has higher altitude compare to present location.
- In optimization problem, we consider a strategy which uses information of candidate solution we are looking at to choose a next better candidate.
- Problem in this method is the possibility of falling into a local optimal solution



APPROXIMATE SOLUTION

- In many real problems, it is difficult to obtain the optimal solution in a suitable time consumption.
- Another counterplan to this kind of problem is using **Heuristic Algorithm**, which can obtain approximate (optimal) solution in short time.
- However, there is no guaranty of optimality in this method.
- Heuristic algorithm which can be applied to general type of problems is called **Metaheuristic Algorithm**.

EVOLUTIONARY COMPUTING

- In order to adapt to living environment and changes, lifes on earth continue to evolve.
- Current ecosystem can be considered as a result of evolutionary to adapt today's environment.
- It can be said that evolutionary is repetition process to solve a complex problem of environment adaption.
- A method to solve optimization problem using hints of biological evolutionary is called **Evolutionary Computation Algorithm**.

EVOLUTIONARY COMPUTING

- Each organism makes use of its own characteristics and situation to find food and other targets.
- In Evolutionary Computation Algorithms, we use ideas from living things as hints to design algorithms to solve optimization problems.

GENETIC ALGORITHM

PROBLEM AND SOLUTION REPRESENTATION

BIOLOGICAL EVOLUTION

- Lifes continue to evolve to adapt the environment
- The various properties of **individual** organisms are determined by their genes.
- In organisms that reproduce sexually, **genes** are passed on to offspring through **crossover** in the **chromosomes** of the two parent individuals.
- It is thought that by crossing two individuals that are adapted to the environment, the probability of producing a child that is more adapted to the environment increases.

BIOLOGICAL EVOLUTION

- If the more adaptable an individual is to the environment, the more likely it is to pass on its genes to the next generation, the more adaptable individuals will flourish in the **population** of the species.
- By crossover, the offspring inherit the genes of their parents, but **mutations** can also occur that result in the offspring having genes that neither parent had.

BIOLOGICAL EVOLUTION

- As a result of mutations, the offspring may have a much higher capacity to adapt to the environment than the parents.
- On the other hand, a mutation can change the desirable genes inherited from the parents, resulting in a child with less adaptability to the environment than the parents.

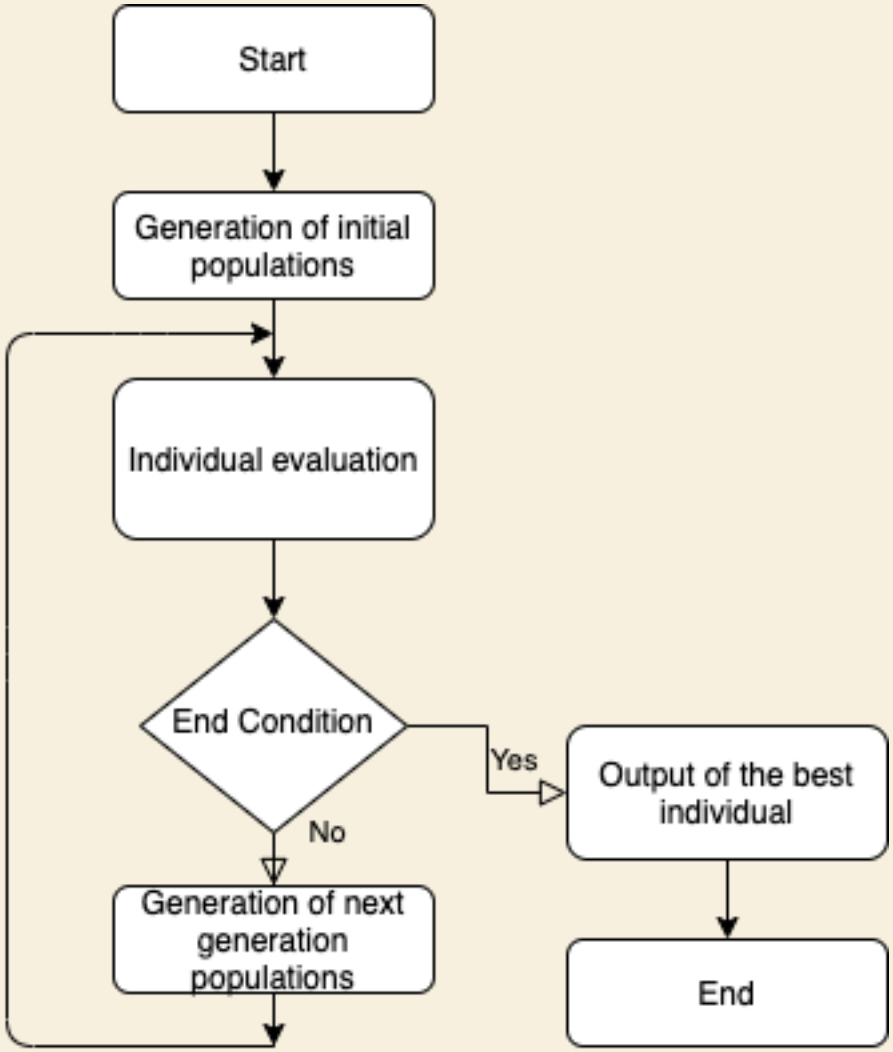
GENTIC ALGORITHM (GA)

- By John Henry Holland
- Mimicking the mechanism of biological evolution to search for optimal solution of optimization problem
- Not a simulation of evolutionary process, but using it as hint in designing algorithm

GENTIC ALGORITHM (GA)

- Focus points:
  - the **more adaptable** an individual is to the environment, the more likely it is to pass on its genes to the next generation
  - **crossover** of two individuals creates a child
  - Occasionally **mutations** occur.
- In GA we simplify the conditions
  - All individuals will change generations at the same time.
  - The gender of each individual is not considered.
  - The number of individuals belonging to a population is variable and

GENETIC ALGORITHM (GA)



SOLUTION REPRESENTATION

- In GA, we consider the following correspondence
  - Solutions of problem ⟷ **Chromosomes**
  - Component of solution ⟷ **Genes**

- The optimal solution is found by evolving multiple individuals to find one that is more adaptable to the environment.

SOLUTION REPRESENTATION

- Consider a case where solution of the problem defined by N values. The solution $\overrightarrow{x_i}$ can be represented by a chromosome vector of N dimensions. If the genes of individual I_i is $\{x_1^i, x_2^i, \cdots, x_N^i\}$ then the chromosome $\overrightarrow{x_i}$ is

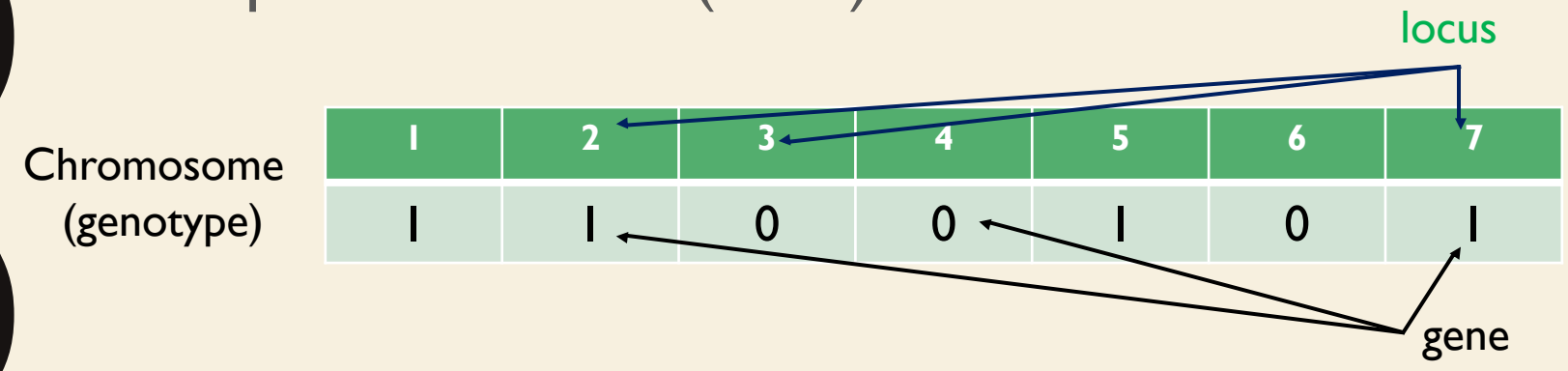
$$\overrightarrow{x_i} = \begin{pmatrix} x_1^i \\ x_2^i \\ \vdots \\ x_N^i \end{pmatrix}$$

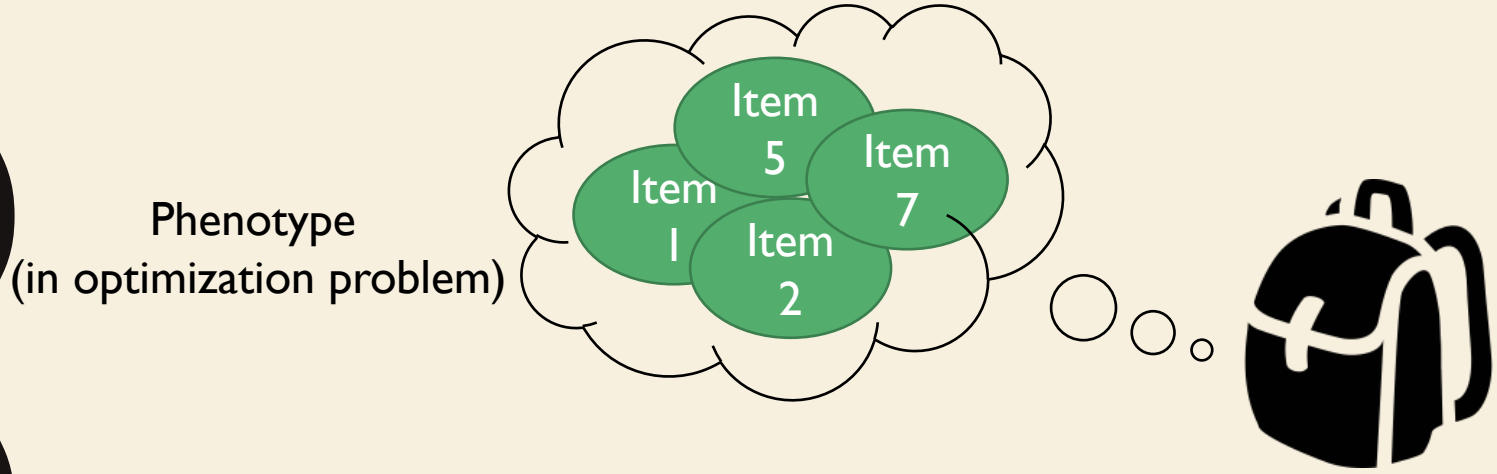
- If the number of individuals belonging to a population is M, the set of candidate solutions X is as follows.

$$X = \{\overrightarrow{x_1}, \overrightarrow{x_2}, \cdots, \overrightarrow{x_M}\}$$

SOLUTION REPRESENTATION

- Knapsack Problem (N=7)





FITNESS VALUE AND FUNCTION

- Each individual is evaluated on the basis of its phenotype in terms of its degree of adaptation to the environment, i.e., how good it is as a solution to the problem.
- The numerical value representing the evaluation result is called the **fitness value**, and the function to calculate the fitness value is called the **fitness function**.

FITNESS VALUE AND FUNCTION

- Sometimes the objective function of an optimization problem is directly used as the fitness function.
- Sometimes it is defined in such a way that the fitness value varies depending on whether the constraints are satisfied or not.

FITNESS VALUE AND FUNCTION

- Knapsack problem (capacity B)
  - j-th item's size and value : w_i , v_i
  - Information of j-th item in i-th individual(I_i) : x_j^i
    - Value: $\vec{v}$, Size: $\vec{w}$, Chromosome: $\vec{x}$

$$\vec{v} = \begin{pmatrix} v_1 \\ \vdots \\ v_N \end{pmatrix}, \vec{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_N \end{pmatrix}, \vec{x_i} = \begin{pmatrix} x_1^i \\ \vdots \\ x_N^i \end{pmatrix}$$

$$fitness(I_i) = \begin{cases} 0 & \text{if } \sum_j w_j x_j^i > B \\ \sum_j v_j x_j^i & \text{if } \sum_j w_j x_j^i \leq B \end{cases}$$

PARENT INDIVIDUALS SELECTION

- Review of Focus points:
  1. the **more adaptable** an individual is to the environment, the more likely it is to pass on its genes to the next generation
  2. **crossover** of two individuals creates a child
  3. Occasionally **mutations** occur.
- **From 2**: we need to make decisions in **parent selections**.
- **From 1**: we need consider **selection strategy** base on **fitness value**

PARENT INDIVIDUALS SELECTION

- **Roulette Selection**:
  - Use ratio of fitness value as selection probability
  - Selection probability

$$\text{rouPro}(I_k) = \frac{fitness(I_k)}{\sum_i fitness(I_i)}$$

  - In case of negative value, we can transform fitness value to positive value before calculate the probability.

$$\text{trFitMin}(I_i) = \frac{fitness(I_i) - \min(fitness)}{\max(fitness) - \min(fitness)}$$

Or

$$\text{trFitMin}(I_i) = \frac{max(fitness) - fitness(I_i)}{\max(fitness) - \min(fitness)}$$

PARENT INDIVIDUALS SELECTION

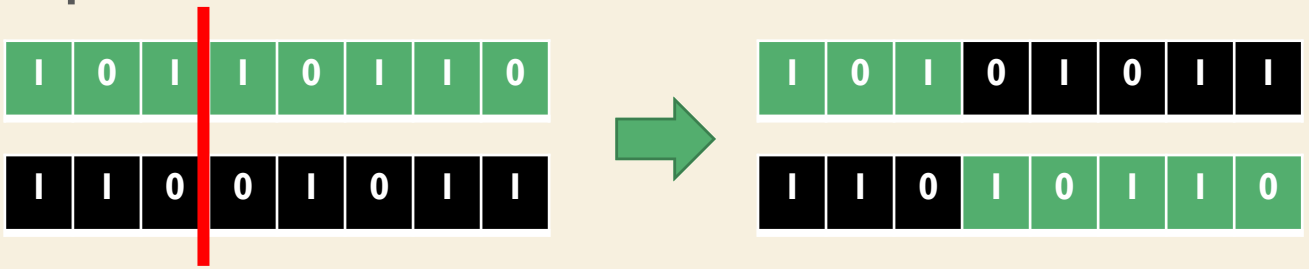
- **Ranking Selection**:
  - Define selection probability by each individual ranking
  - Number of Individuals in Population = M

$$\text{rankPro}(I_k) = \frac{M - rank(I_k) + 1}{\sum_i rank(I_i)}$$

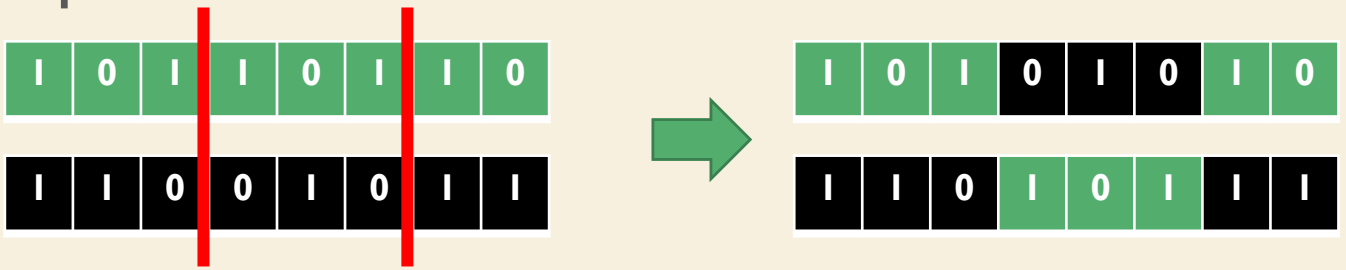
- **Tournament Selection**:
  - Random select S individuals from the population and select one which with the highest fitness value

CROSSOVER

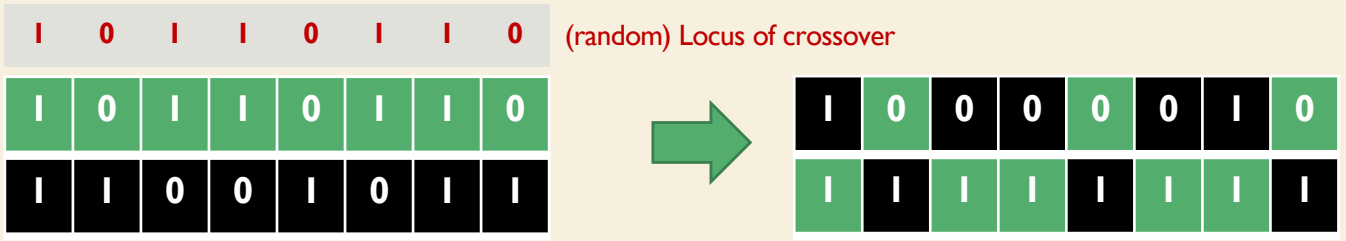
- One-point crossover



- Two-point crossover



- Uniform crossover



MUTATION

- Result in the offspring having genes that neither parent had

EVOLUTIONARY STRATEGY

- In order to obtain the optimal solution efficiently, the evolutionary strategy of how to generate the population of each generation is important.
- E.g.
  - making the selection target of the parent individuals a part of the population instead of the entire population
  - attaching conditions to the individuals to be adopted as child individuals
  - changing the method of parental selection for each generation

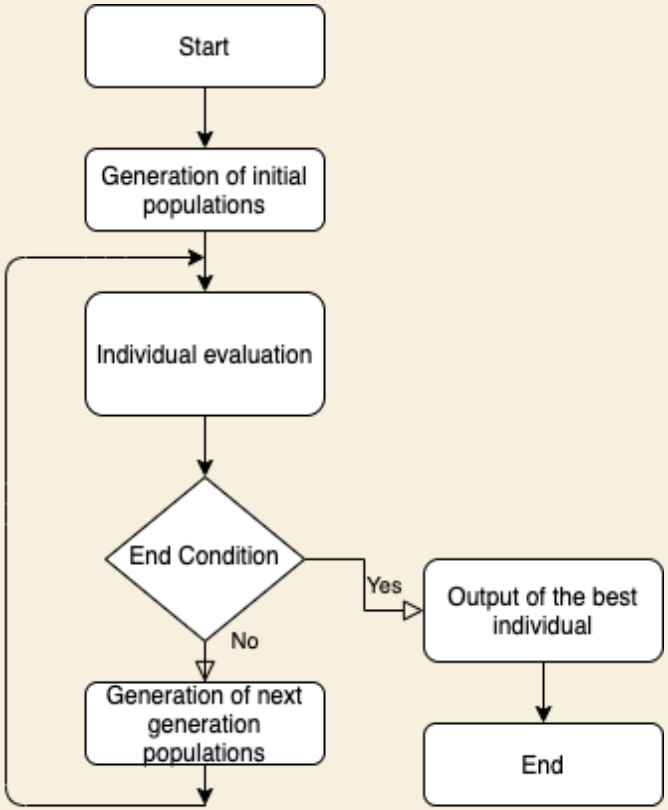
ELITE PRESERVATION STRATEGY

- Even if a very good individual is produced in one generation, and that individual is selected as the parent individual, there is a possibility that the good gene sequence will not be passed on to the next generation because of the bad combination of the parent individuals.
- As a countermeasure to this situation, the strategy is to leave good individuals to the next generation unconditionally.
- However, leave too many elite to next generation increases the probability of falling into local optimal solution.

GA ON KNAPSACK PROBLEM

- Fitness function (number of items = N , capacity = B)

$$fitness(I_i) = \begin{cases} 0 & \text{if } \sum_j w_j x_j^i > B \\ \sum_j v_j x_j^i & \text{if } \sum_j w_j x_j^i \leq B \end{cases}$$



- Apply Mutation with probability 0.01
- Apply one-point crossover
- Use ranking selection method in parent individuals selection process
- Population size = 500
- End condition: number of generation=500

Implement with Python

PROBLEM SETTING

config.py

```
import random

GEN_MAX = 500
POP_SIZE = 500
ELITE = 1
MUTATE_PROB = 0.01
N = 200
B = 800

weights = [random.randint(1,10) for _ in range(N)]
values = [random.randint(10, 100) for _ in range(N)]

ITEMS = [(weight, value) for (weight,value) in zip(weights,values)]
```

INDIVIDUAL CLASS

Individual.py

```
import random
import math
import config

class Individuale(object):
     def __init__(self):
          self.chrom = [random.randint(0,1) for _ in range(config.N)]
          self.fitness = 0.0
          self.capacity = 0.0
          def evaluate(self):
          self.fitness = 0.0
          self.capacity = 0.0
          for i in range(config.N):
               self.fitness += (self.chrom[i]) * config.ITEMS[i][1]
               self.capacity += (self.chrom[i]) * config.ITEMS[i][0]
          if self.capacity > config.B:
               self.fitness = 0.0
     def crossover(self, p1, p2):
          point = random.randint(0,config.N-2)
          for i in range(point):
               self.chrom[i] = p1.chrom[i]
          for i in range(point, config.N):
               self.chrom[i] = p2.chrom[i]

     def mutate(self):
          for i in range(config.N):
               if random.random() < config.MUTATE_PROB:
                    self.chrom[i] = 1 - self.chrom[i]
```

POPULATION CLASS(1)

Population.py

```
import config
from Individual import Individuale
import random

class Population(object):
      def __init__(self):
            self.ind = []
            self.nextInd = []
            for i in range(config.POP_SIZE):
                  self.ind.append(Individuale())
                  self.nextInd.append(Individuale())
            self.evaluate()
      def evaluate(self):
            for i in range(config.POP_SIZE):
                  self.ind[i].evaluate();
                  self.ind = sorted(self.ind, key=lambda x: x.fitness, reverse=True)
            def alternate(self):
                  for i in range(config.ELITE):
                        for j in range(config.N):
                              self.nextInd[i].chrom[j] = self.ind[i].chrom[j]

                  for i in range(config.ELITE, config.POP_SIZE):
                        p1 = self.select()
                        p2 = self.select()
                        self.nextInd[i].crossover(self.ind[p1], self.ind[p2])
                  for i in range(1, config.POP_SIZE):
                        self.nextInd[i].mutate()
                  tmp = self.ind
                  self.ind = self.nextInd
                  self.nextInd = tmp

                  self.evaluate()
```

POPULATION CLASS(2)

Population.py

```
    def select(self):
        denom = config.POP_SIZE * (config.POP_SIZE + 1) / 2.
        r = random.random()
        for rank in range(1, config.POP_SIZE):
            prob = (config.POP_SIZE - rank + 1)/denom
            if r <= prob:
                break
            r -= prob
        return rank-1
    def printResult(self):
        print("In knapsack: ", end='')
        for i in range(config.N):
            if self.ind[0].chrom[i] == 1:
            print("%d "%(i+1), end='')
            print("\nValue = %f\n"%(self.ind[0].fitness))
    def getMeanFitness(self):
        return sum([i.fitness for i in self.ind])/config.POP_SIZE
```

MAIN

main.py

```
import config
from Population import Population
import matplotlib.pyplot as plt

def main():
      meanFitness = []
      maxFitness = []
      pop = Population()
      for i in range(config.GEN_MAX):
            pop.alternate()
            if i%100 == 0:
                  print("Gen. #%3d : Fitness= %f (Capacity:%4d/%4d)"%(i,
      pop.ind[0].fitness, pop.ind[0].capacity, config.B))
      meanFitness.append(pop.getMeanFitness())
      maxFitness.append(pop.ind[0].fitness)
      plt.plot(list(range(config.GEN_MAX)), meanFitness)
      plt.plot(list(range(config.GEN_MAX)), maxFitness)
      plt.legend(['mean fitness', 'max fitness'])
      plt.xlabel('Generation')
      plt.ylabel('Fitness')
      plt.title('Knapsack N='+str(config.N))
      plt.show()
      pop.printResult()
      del pop

if __name__ == "__main__":
      main()
```

RESULT

```
Generation #  0 : Fitness= 6809.000000 (Capacity: 679/ 800)
Generation #100 : Fitness= 9578.000000 (Capacity: 789/ 800)
Generation #200 : Fitness= 9742.000000 (Capacity: 798/ 800)
Generation #300 : Fitness= 9818.000000 (Capacity: 797/ 800)
Generation #400 : Fitness= 9835.000000 (Capacity: 798/ 800)
In knapsack: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 23 28 30 31 33 34 36 37
 40 41 42 43 44 45 46 47 48 49 50 53 54 55 56 57 58 59 60 62 64 65 68 69 71 74 76 77 78 80
 81 82 83 84 86 87 88 89 90 91 92 93 96 97 99 100 101 102 103 106 107 108 111 112 113 114
115 117 118 119 120 121 123 124 125 127 128 129 130 132 134 135 136 137 138 139 140 141 14
2 143 144 145 146 148 149 150 152 153 154 155 156 157 158 159 160 162 163 164 165 166 167
168 169 170 171 172 173 175 176 177 178 180 181 183 184 187 190 191 192 194 196 197 198 19
9 200
Value = 9843.000000


Size  = 800.000000
```