Lecture 10
Genetic Algorithm

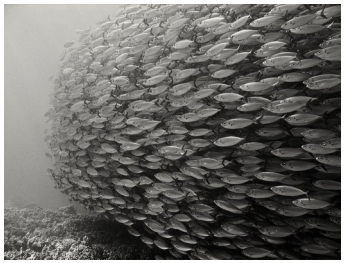
CS 180 – Intelligent Systems

Dr. Victor Chen

Spring 2021

Nature-inspired algorithms

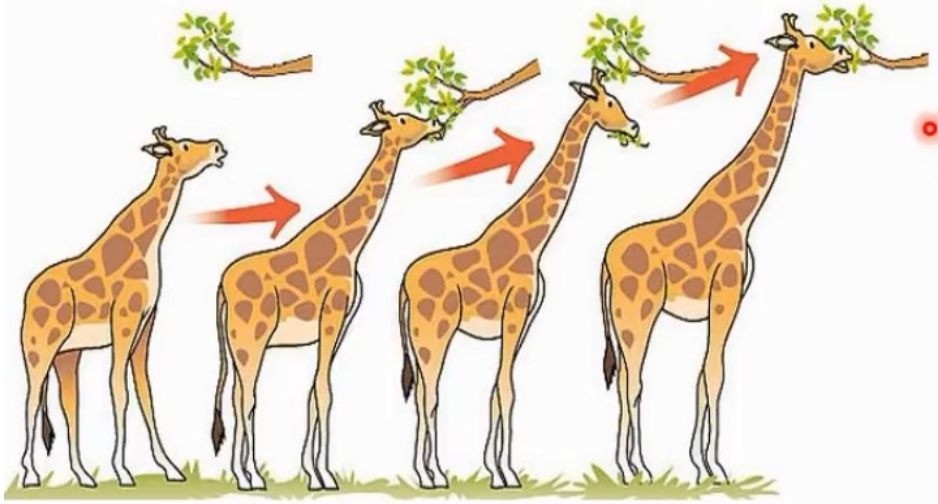
- Genetic Algorithm **(GA)**
- Particle Swarm Optimization (**PSO**)
- Ant Colony Optimization **(ACO)**
- Artificial Bee Colony **(ABC).**
- Bird Swarm Algorithm (**BSA**)



Genetic Algorithm

Based on Darwin's Theory

Natural Selection



Search-based optimization technique

Uses concept from **Evolutionary Biology** (Natural Genetics & Natural Selection)

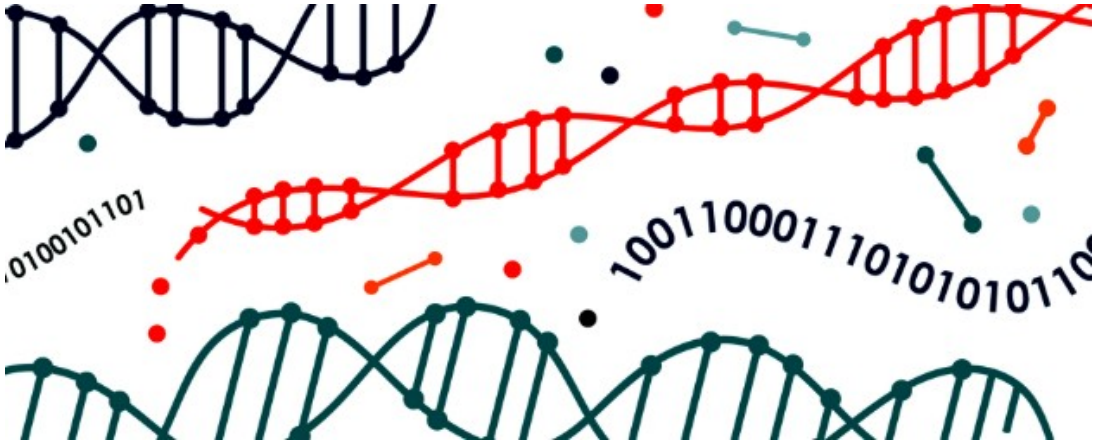
Pool or a population of possible solutions to the given problem

John Holland introduced in 1975

Intuition of GA

You are head of a country, and in order to keep your city safe from bad things, you implement a policy like this:

- You select all the good people, and ask them to extend their generation by having their children.
- This repeats for a few generations.
- You will notice that now you have an entire population of good people.



An Overview Of GA

- GA creates a “population”, or to say, a “group” of possible solutions to the given problem and lets them “evolve” over multiple generations to find better and better solutions.

Applications

When GA Meets Super Mario

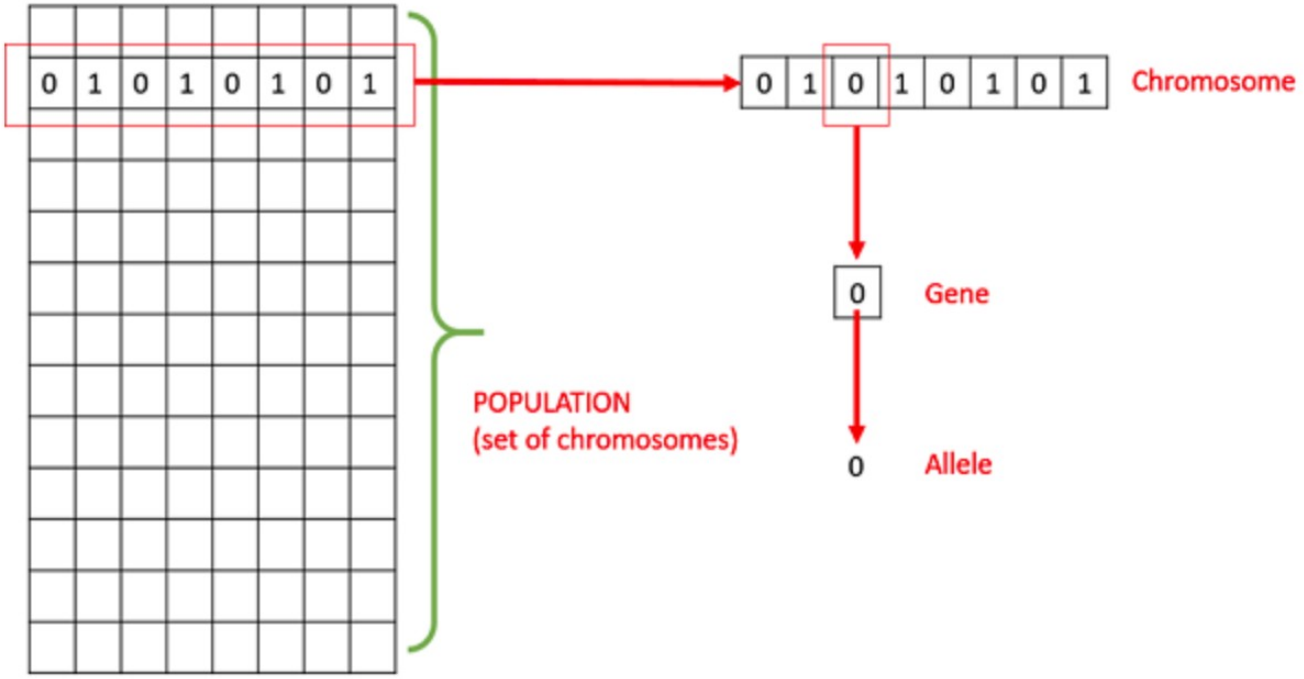
Basic Terminology

Population − It is a subset of possible (encoded) solutions.

Chromosome (individual) − A chromosome (individual) is one possible solution

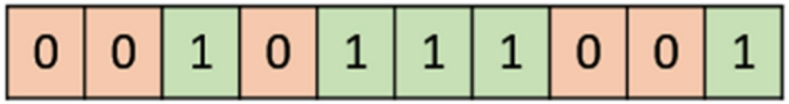
Gene − A gene is one element position of a chromosome.

Allele − It is the value a gene takes.

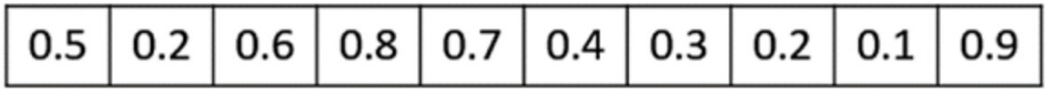


Think about possible solution representations (encoding)

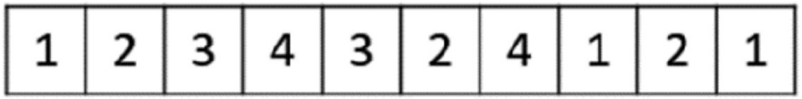
Binary Representation (bit strings)



Real Valued Representation

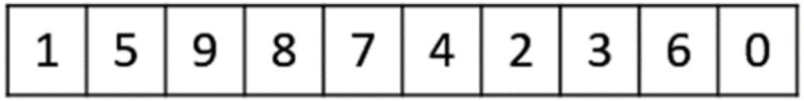


Integer Representation



Permutation Representation (each solution is an ordering of elements)

- E.g., Travelling salesman problem (TSP)

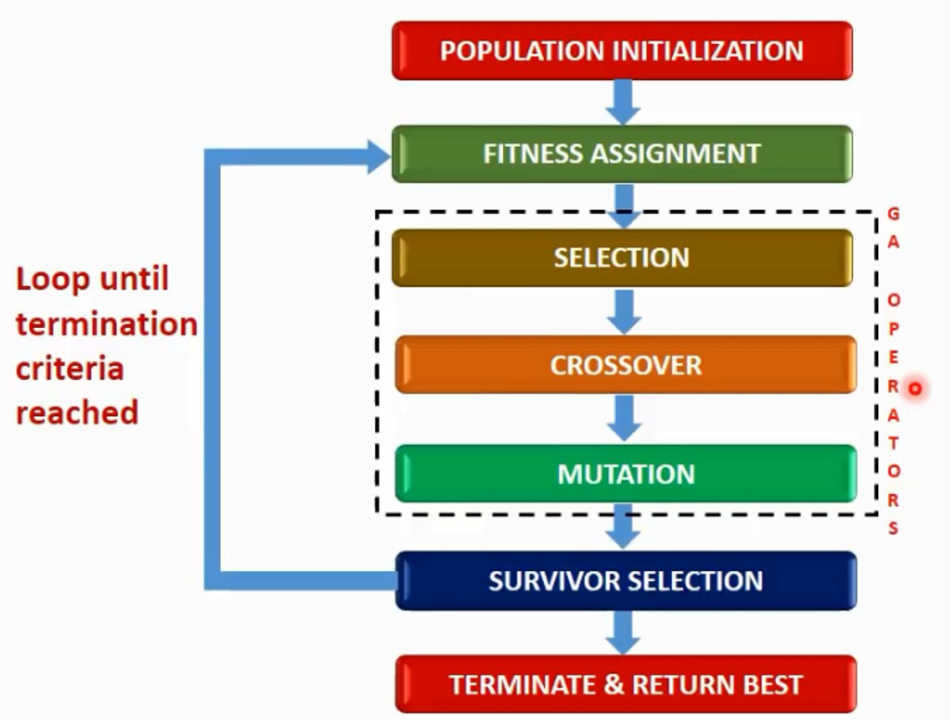


Basic Structure of Genetic Algorithm

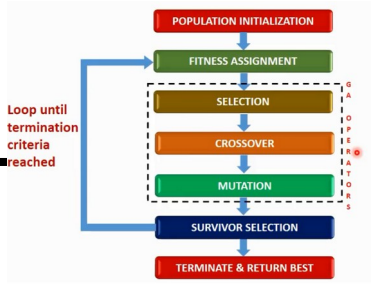
Two possible GA models

- **Steady State GA**: We generate one or two off-springs in each iteration and they will replace one or two individuals from the current population.

- **Generational GA**: We generate 'n' off-springs, where n is the population size, and the entire population will be replaced by those new ones at the end of each iteration.

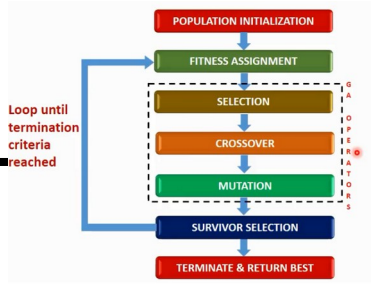


Population Initialization



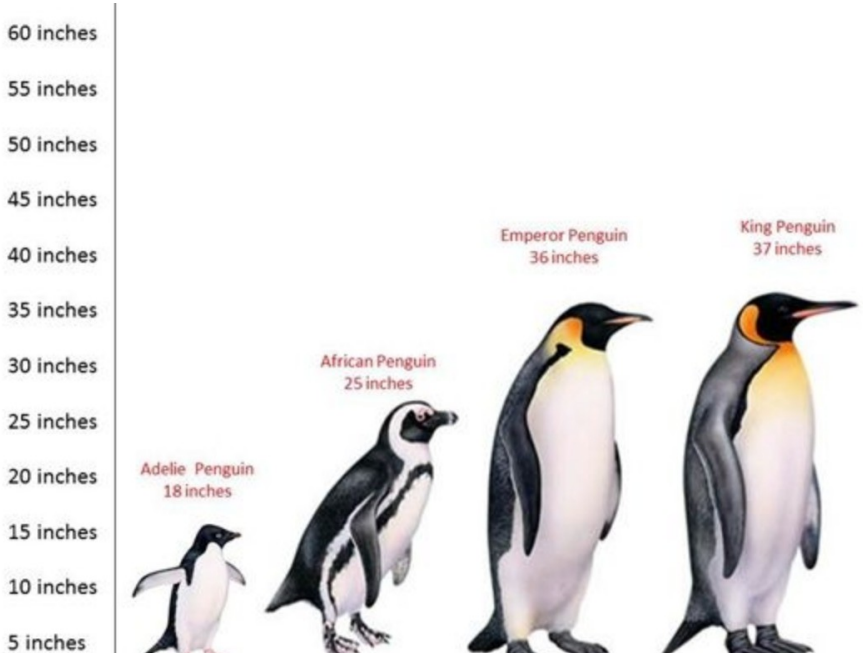
Populate the initial population with random individuals.

Fitness Function

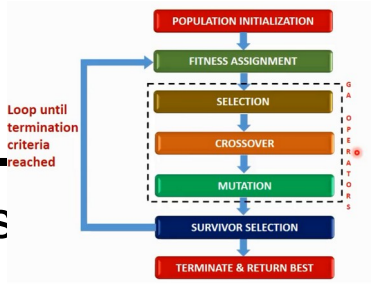


The fitness function measures the goodness value of the individual

The fitness function should be sufficiently fast to compute



Parent Selection

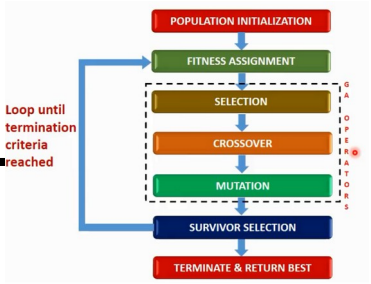


Select parents who are allowed to create their off-springs for the next generation.

Three methods:

1. Fitness Proportionate Selection
2. K-Way Tournament Selection
3. Rank Selection

Fitness Proportionate Selection

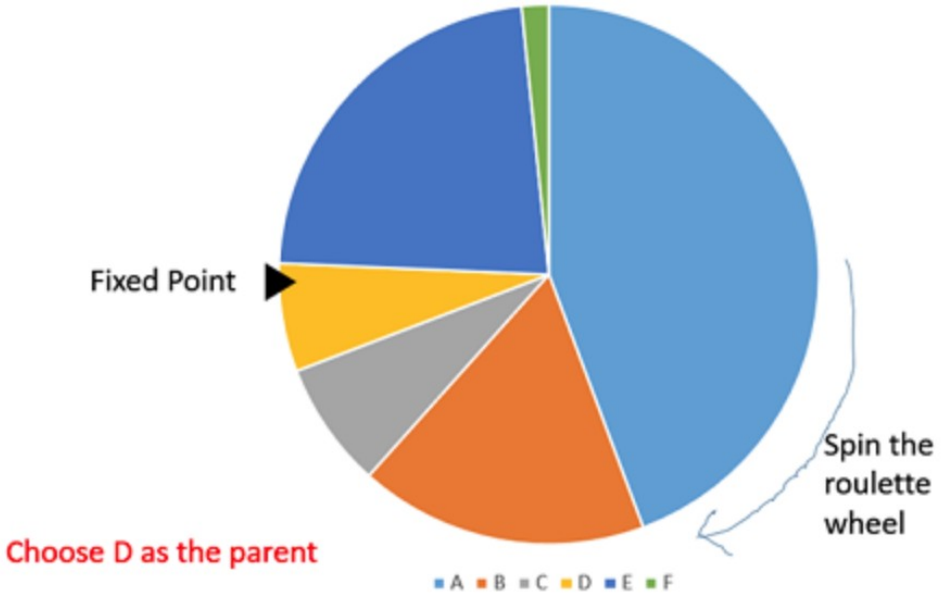


Fitness Proportionate Selection: Every individual can become a parent with a probability which is proportional to its fitness.

- Two implementations:
  - **Roulette Wheel Selection**
  - **Stochastic Universal Sampling (SUS)**

Roulette Wheel Selection

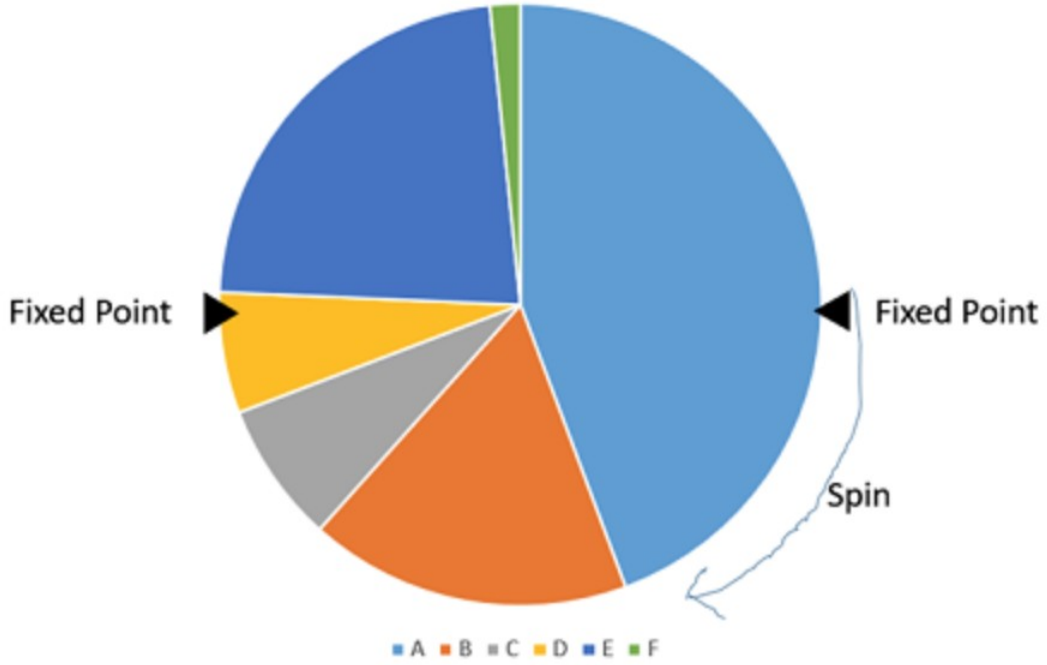
- The wheel is divided into **n pies**, where n is the number of individuals in the population (population size).
- Each individual gets a portion of the circle which is proportional to its fitness value.
- **Just one fixed point.** The region which comes in front of the fixed point is chosen as the parent.
- For the second parent, the same process is repeated.



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Stochastic Universal Sampling (SUS)

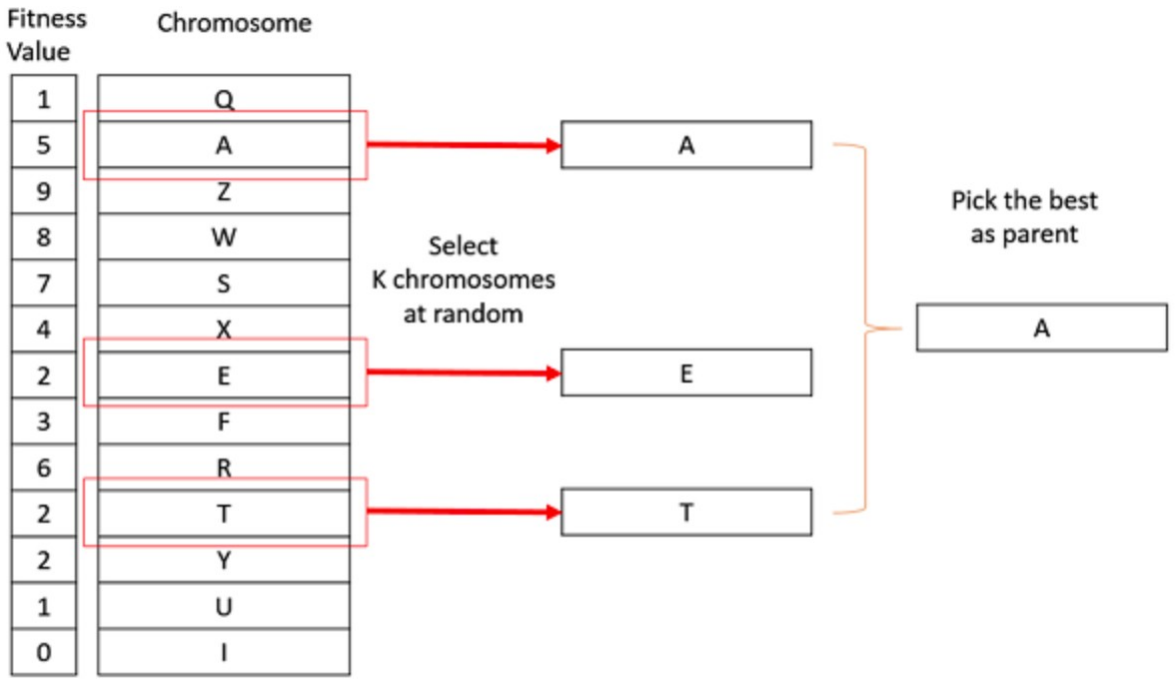
- The wheel has **N equally spaced fixed points**
- Therefore, all **N** parents are chosen in just one spin of the wheel



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

K-Way Tournament Selection

- We select K individuals from the population at random and select the best out of these to become a parent.
- The same process is repeated for selecting the next parent.

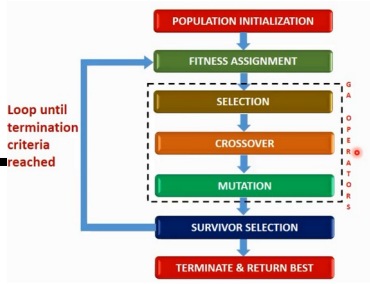


Rank Selection

- Every individual in the population is ranked according to their fitness.
- The selection of the parents depends on the rank of each individual rather than the fitness.
- The higher ranked individuals are preferred more than the lower ranked ones.

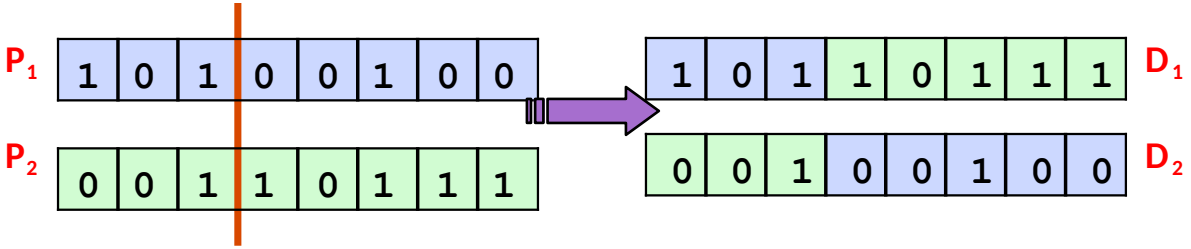
Chromosome	Fitness Value	Rank
A	8.1	1
B	8.0	4
C	8.05	2
D	7.95	6
E	8.02	3
F	7.99	5

Crossover (exploitation)

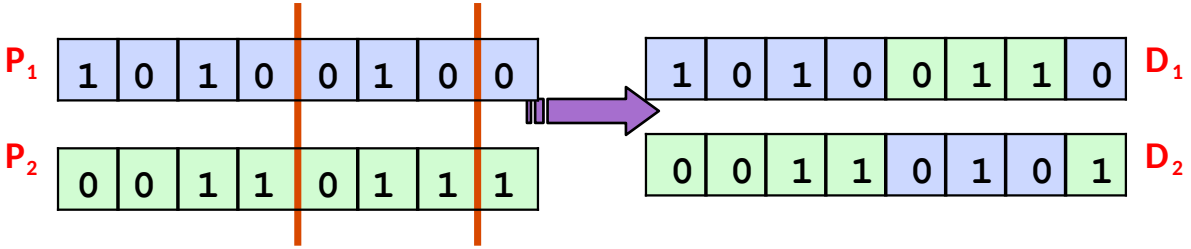


Random crossover point(s) are selected and segments are swapped to get new off-springs.

- 1 point cross-over

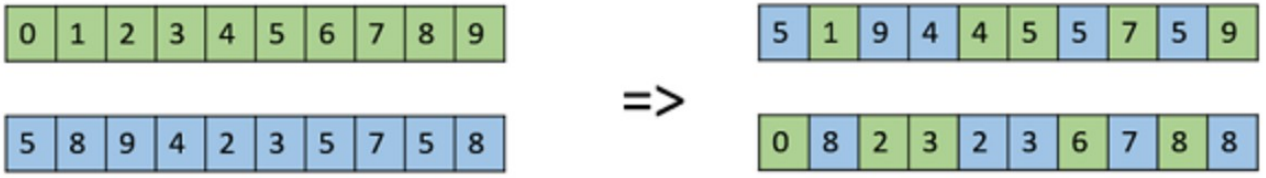


- 2-points cross-over



Crossover (exploitation)

Uniform Crossover: We treat each gene separately. We flip a coin for each gene to decide whether it'll be swapped or not in the off-springs based on heads-up or heads-down.

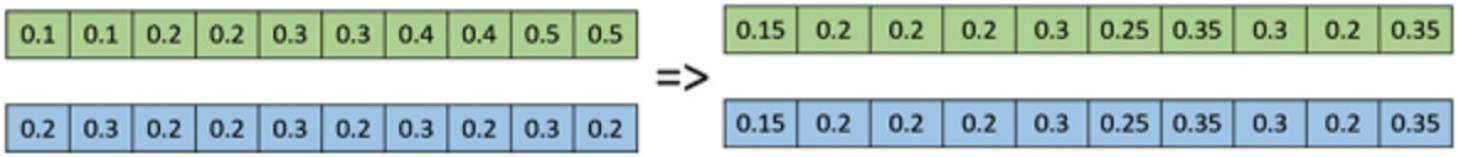


Arithmetic Recombination Crossover

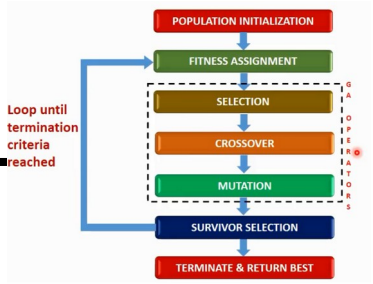
This is commonly for integer/real representations by taking the weighted average of the two parents :

- Child1 = α.x + (1-α).y
- Child2 = α.y + (1-α).x

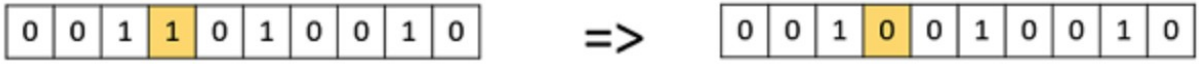
Obviously, if α = 0.5, then both the children will be identical.



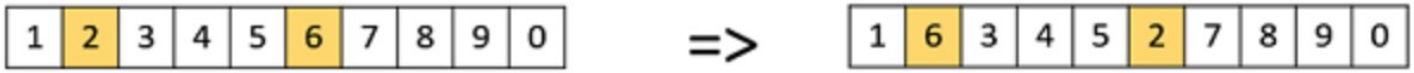
Mutation (exploration)



- Bit Flip Mutation
  - select one or more random bits and flip them. This is used for binary encoded GAs.

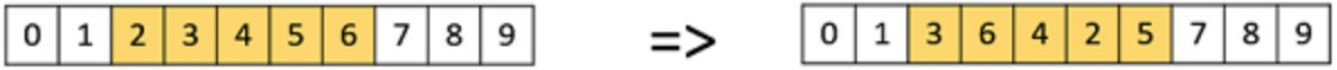


- Swap Mutation
  - select two random positions on the chromosome, and interchange the values

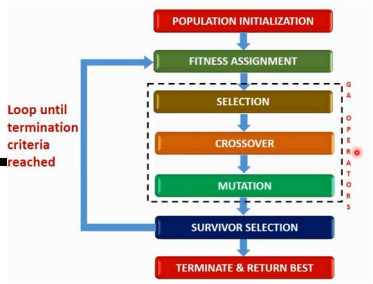


Mutation

- Scramble Mutation
  - a subset of genes is chosen and their values are shuffled randomly.



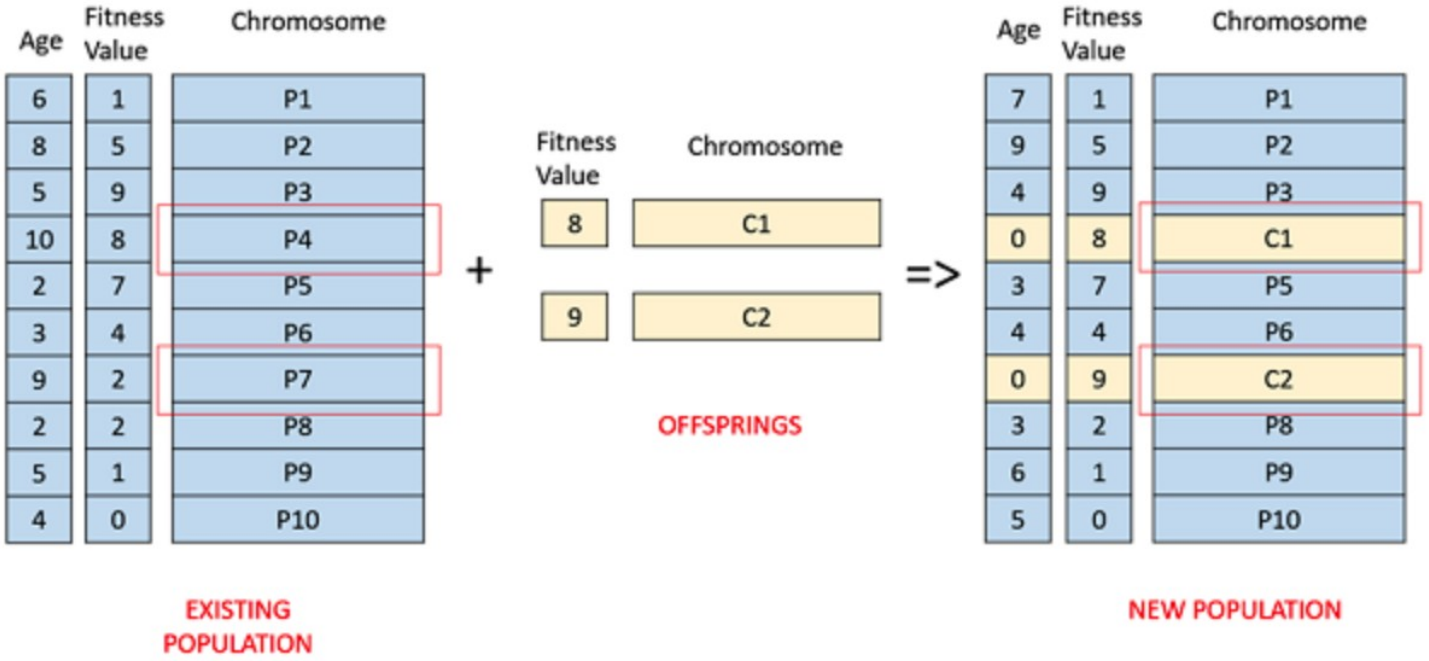
Survivor Selection



- **Steady State GA vs Generational GA**
- Determines which individuals are to be kicked out from the next generation.
- The idea here is to ensure that the fitter individuals are not kicked out, while at the same time diversity should be maintained in the population
  - Age Based Selection
  - Fitness Based Selection

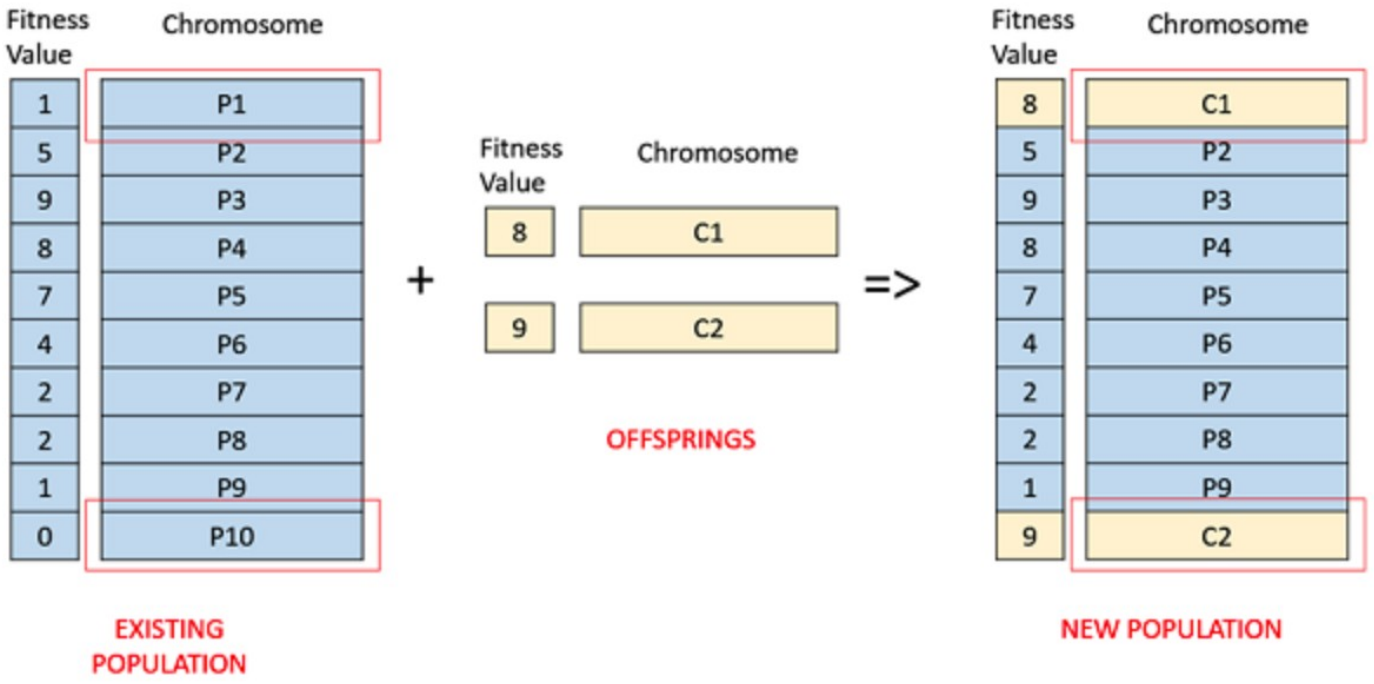
Age Based Selection

- Each individual is allowed in the population for a finite generation. After that, it is kicked out of the population no matter how good its fitness is.
- No fitness used

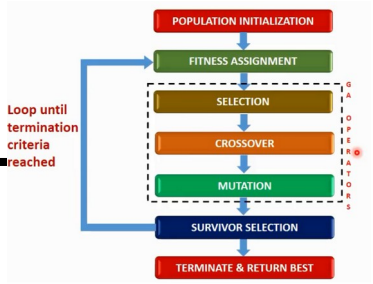


Fitness Based Selection

The children replace the least fit individuals in the population.



Termination Condition



The above process repeats until some termination conditions are reached

- When we reach an certain number of generations.
- When the objective function value has reached a certain pre-defined value.

GA Example 1

FUNCTION MAXIMIZATION

Determine the variables that produce the maximum value for this function:

$$f(w, x, y, z) = w^3 + x^2 - y^2 - z^2 + 2yz - 3wx + wz - xy + 2$$

Questions that need to be answered to use GA

1. **How is an individual represented?**
2. **How is an individual's fitness calculated?**
3. **How are individuals selected for mating?**
4. **How are individuals crossed-over?**
5. **How are individuals mutated?**

1. How is an individual represented?

$$f(w, x, y, z) = w^3 + x^2 - y^2 - z^2 + 2yz - 3wx + wz - xy + 2$$

Choose a binary sequence for each variable, and then concatenate the four values together into a single bit string.

1101 0110 0111 1100

represents a solution where w = 13, x = 6, y = 7, and z = 12

2. How is an individual's fitness calculated?

Assume we have a population of 4 individuals:

Individual	*w*	*x*	*y*	*z*	*f*
1010111010000011	10	14	8	3	671
0110100111110110	6	9	15	6	-43
0111011011101011	7	6	14	11	239
0001011010000000	1	6	8	0	-91

Individual	*w*	*x*	*y*	*z*	*f*
1010111010000011	10	14	8	3	671
0110100111110110	6	9	15	6	-43
0111011011101011	7	6	14	11	239
0001011010000000	1	6	8	0	-91

3. How are individuals selected for mating?

Suppose we use rank selection where individuals are ranked according to their fitness.

- → **the first individual a 40% chance of being selected**
- → **the second a 20% chance**
- → **the third a 30% chance**
- → **the fourth a 10% chance**

Clearly this is giving the better individuals more chances to be selected.

4. How are individuals crossed-over?

Suppose we use 2-points cross-over

1010 1110 1000 0011

0110 1001 1111 0110

0111 0110 1110 1011

0001 0110 1000

Assume that the 1st and 3rd individuals are selected this time.

The 4th and 14th bits are randomly selected to define the substring to be swapped, so the cross-over looks like this:

```
1010111010000011     101*0111010000*11     1011011011101011
                  →                     →
0111011011101011     011*1011011101*011     0110111010000011
```

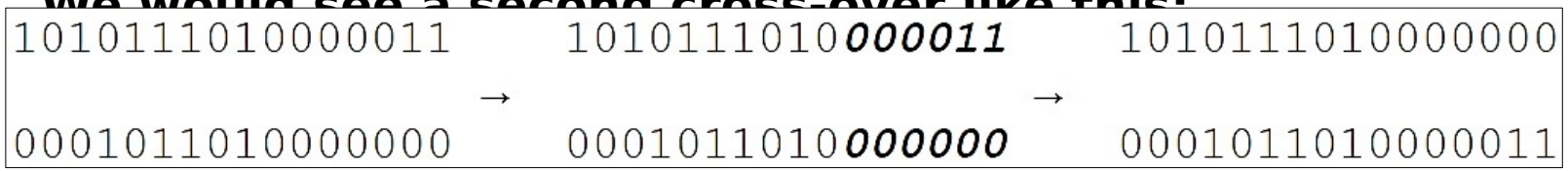
Thus, two new individuals are created.

Suppose we use generational-GA:

- We generate ‘n’ off-springs, where n is the population size, and the entire population is replaced by the new one at the end of the iteration.
- We should keep creating new individuals until we have created enough to replace the entire population.

- **Assume that the 1st and 4th individuals are selected this time.**
- **Note that an individual may be selected multiple times for breeding, while other individuals might never be selected.**
- **Further assume that the 11th and 16th bits are randomly selected for the cross-over point.**

We would see a second cross-over like this:



So, we have four new offsprings:

1011 0110 1110 1011

0110 1110 1000 0011

1010 1110 1000 0000

0001 0110 1000 0011

5. How are individuals mutated?

The selected policy: Allow every single bit a small chance to mutate.

Best practice: setting the mutate probability so that roughly 1 bit per individual will change on average.

The mutation will have the bit "flip" – 1 to 0 and 0 to 1.

1011011011101011	→	1011011011101011
0110***1***1010000011	→	0110***0***1010000011
1010111010***0***0000	→	1010111010***1***0000
0***0***01011010000***1***1	→	0***1***01011010000***0***1

Wrapping Up: We get 2-nd generation

Individual	*w*	*x*	*y*	*z*	*f*
1011011011101011	11	6	14	11	1,045
0110101010000011	6	10	8	3	51
1010111010010000	10	14	9	0	571
0101011010000001	5	6	8	1	-19

GA Example 2

Knapsack Problem

ITEM	WEIGHT	VALUE
A	5 kg	$12
B	3 kg	$ 5
C	7 kg	$ 10
D	2 kg	$ 7

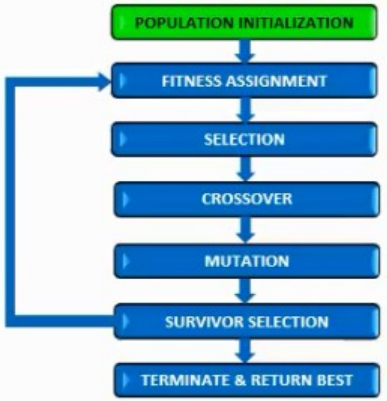
There are four items. Each item is associated with some weight (W) and value at item (v).

There is a knapsack (k) with limited capacity that can hold atmost 12 kg.

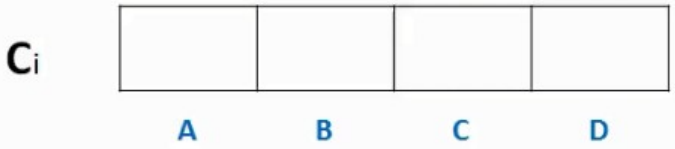
Problem:

The problem is that which item should be kept in the knapsack so as it will maximizes knapsack value without breaking knapsack.

Knapsack Problem by using Genetic Algorithm



STEP -1: Chromosomes Encoding



Gene: **0** – represents absence of item in the knapsack
1 – represents presence of item in the knapsack

ITEM	WEIGHT	VALUE
A	5 kg	$12
B	3 kg	$ 5
C	7 kg	$ 10
D	2 kg	$ 7

4 bits are requested to represent chromosomes encoding

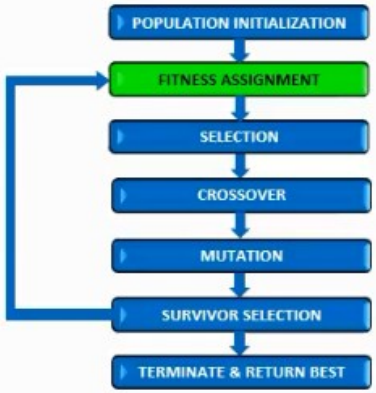
- Set space = 2^4

Initial population is created and chromosomes randomly created

Generation 1

C_1	0	1	1	0
C_2	0	1	0	1
C_3	1	1	0	1
C_4	1	1	1	1

Knapsack Problem by using Genetic Algorithm



STEP -2: Fitness Function

Next Step is to determine fitness function which is used to evaluate how good particular solution is. Lets take C_1

C_1	0	1	1	0
	A	B	C	D

represent that knapsack has presence of item B & C and absence of item A & D

- value of knapsack = value of B + value of C
  = 5 + 10 = 15
- Weight of knapsack = weight of item B + weight of item C
  = 3 + 7 = 10 kg

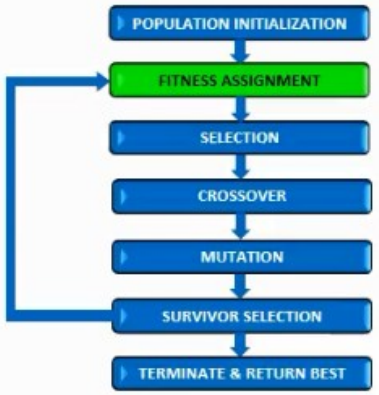
Knapsack capacity = 12 kg as 12kg> 10kg **so C_1 is accepted**

ITEM	WEIGHT	VALUE
A	5 kg	$12
B	3 kg	$ 5
C	7 kg	$ 10
D	2 kg	$ 7

Chromosome Encoding

C1	0	1	1	0
C2	0	1	0	1
C3	1	1	0	1
C4	1	1	1	1

Knapsack Problem by using Genetic Algorithm



STEP -2: Fitness Function

Next Step is to determine fitness function which is used to evaluate how good particular solution is. Lets take C_2

C_2	0	1	0	1
	A	B	C	D

represent that knapsack has presence of item B & D and absence of item A & C

- value of knapsack = value of B + value of D
  = 5 + 7 = 12
- Weight of knapsack = weight of item B + weight of item D
  = 3 + 2 = 5 kg

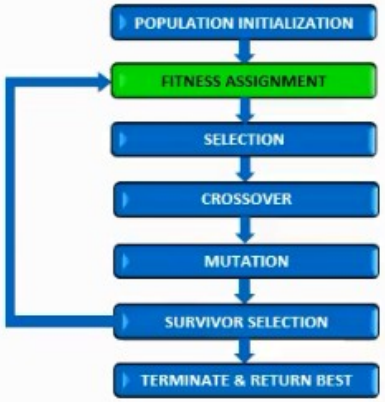
Knapsack capacity = 12 kg as 12kg> 5kg **so C_2 is accepted**

ITEM	WEIGHT	VALUE
A	5 kg	$12
B	3 kg	$ 5
C	7 kg	$ 10
D	2 kg	$ 7

Chromosome Encoding

C1	0	1	1	0
C2	0	1	0	1
C3	1	1	0	1
C4	1	1	1	1

Knapsack Problem by using Genetic Algorithm



STEP -2: Fitness Function

Next Step is to determine fitness function which is used to evaluate how good particular solution is. Lets take C_3

C_3	1	1	0	1
	A	B	C	D

represent that knapsack has presence of item A, B & D and absence of item C.

- value of knapsack = value of A + value of B + value of D
  = 12 + 5 + 7 = 24
- Weight of knapsack = weight of B + weight of C + weight of D
  = 5 + 3 + 7 = 10 kg

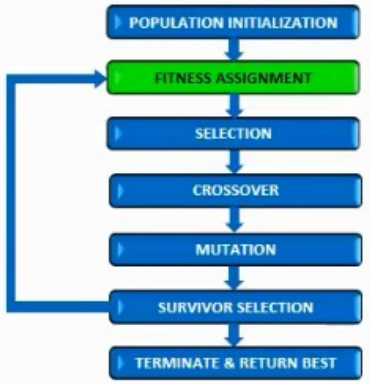
Knapsack capacity = 12 kg as 12kg> 10kg **so C_3 is accepted**

ITEM	WEIGHT	VALUE
A	5 kg	$12
B	3 kg	$ 5
C	7 kg	$ 10
D	2 kg	$ 7

Chromosome Encoding

C1	0	1	1	0
C2	0	1	0	1
C3	1	1	0	1
C4	1	1	1	1

Knapsack Problem by using Genetic Algorithm



STEP -2: Fitness Function

Next Step is to determine fitness function which is used to evaluate how good particular solution is. Lets take C_4

C_4	1	1	1	1
	A	B	C	D

represent that knapsack has presence of all item A, B, C & D

- value of knapsack = value of A + value of B + value of C + value of D
  = 12 + 5 + 10 + 7 = 34
- Weight of knapsack = weight of B + weight of C + weight of B + weight of C
  = 5 + 3 + 7 + 2 = 17 kg

As 17 kg > knapsack capacity; 17 kg > 12 kg **so C_4 is discarded.**

ITEM	WEIGHT	VALUE
A	5 kg	$12
B	3 kg	$ 5
C	7 kg	$ 10
D	2 kg	$ 7

Chromosome Encoding

C1	0	1	1	0
C2	0	1	0	1
C3	1	1	0	1
C4	1	1	1	1

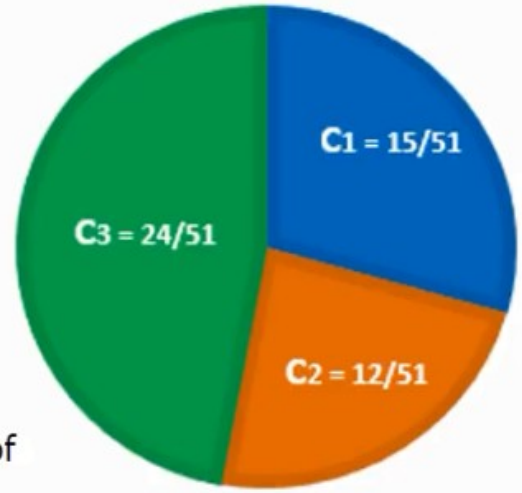
Knapsack Problem by using Genetic Algorithm

STEP -3: Selection

Next step is to collect the filter individual and wake up the next generation, chromosome

By using Roulette Wheel Selection

- Spin the Roulette Wheel and whenever the wheel stops, the individual gets selected at that point.
- The individual that has the highest fitness value gets larger share of the wheel



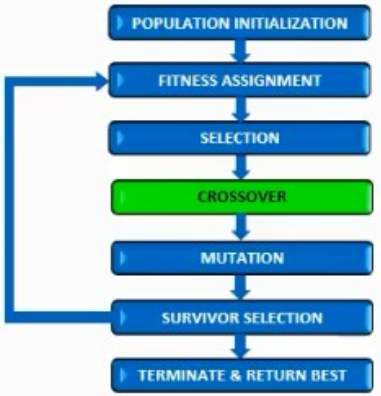
e.g. total fitness value = 15 + 12 +24 + 0 = 51

Fitness value of C_3 = 24 ; largest fitness

So, C_3 occupies half of the wheel as 24/51

C_4 has zero chance of wining, C_3 has the highest probability of getting selected in the next generation

Knapsack Problem by using Genetic Algorithm



STEP -4: Crossover

The crossover operation takes the selected chromosomes for mating and mixes the genetic material to produce offspring.

C_3	1	1	0	1
C_2	0	1	1	0
	A	B	C	D

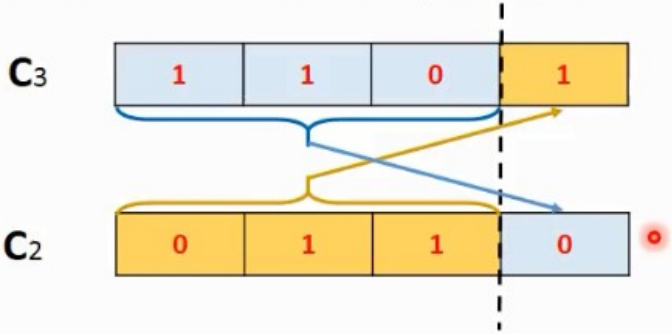
ITEM	WEIGHT	VALUE
A	5 kg	$12
B	3 kg	$ 5
C	7 kg	$ 10
D	2 kg	$ 7

One point crossover – Randomly select the position on the chromosomes about which gene would be exchange.

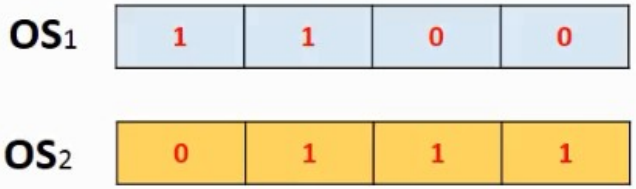
C_3	1	1	0	1
C_2	0	1	1	0

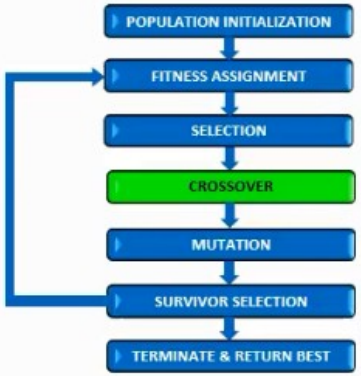
Knapsack Problem by using Genetic Algorithm

One point crossover – Randomly select the position on the chromosomes about which gene would be exchange.



Result of one point crossover i.e. produced offspring





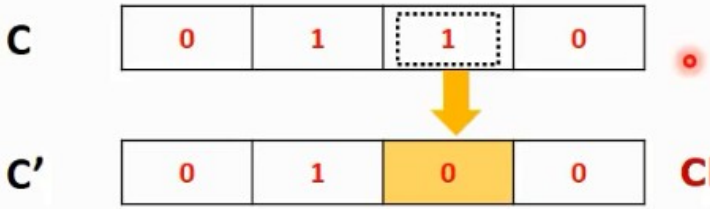
ITEM	WEIGHT	VALUE
A	5 kg	$12
B	3 kg	$ 5
C	7 kg	$ 10
D	2 kg	$ 7

Knapsack Problem by using Genetic Algorithm

STEP -4: Mutation

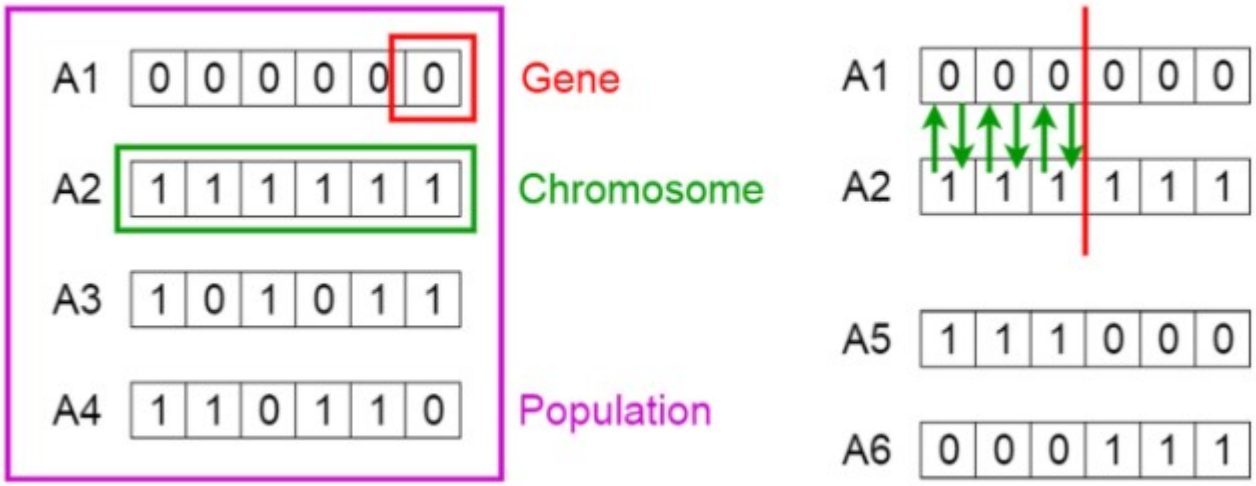
Introduces the diversity within the population so that search algorithm does not necessarily get stuck at local maxima.

Lets us consider chromosomes C. Now randomly select a gene from C. A Flip happens at the selected genes and zero become one and one becomes zero



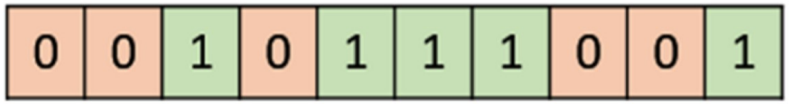
Chromosome after mutation

Practical issues to consider when using GA

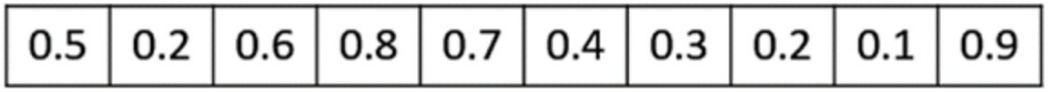


Representation (encoding) affect solution quality

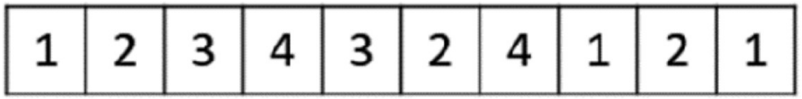
Binary Representation (bit strings)



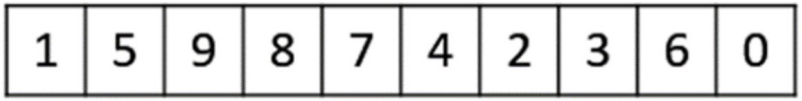
Real Valued Representation



Integer Representation



Permutation Representation (each solution is an ordering of elements)



Revisiting Binary Representations

Hamming Cliffs

- Locality may not be preserved

0111 = 7 but 0011 = 3 !

A small change in one bit may have a massive effect on the solution.

Gray Coding partly overcomes this issue

- Converts the mapping from binary to integer
- Can reach any adjacent integer by single bit-flip

Gray coding (any two successive values differ in only one bit)

Integer	Binary	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

To convert a binary number $d_1\ d_2 \cdots d_{n-1}\ d_n$ to its corresponding binary reflected Gray code, start at the right with the digit d_n (the nth, or last, digit). If the d_{n-1} is 1, replace d_n by $1 - d_n$; otherwise, leave it unchanged. Then proceed to d_{n-1}. Continue up to the first digit d_1, which is kept the same since d_0 is assumed to be a 0. The resulting number $g_1\ g_2 \cdots g_{n-1}\ g_n$ is the reflected binary Gray code.

In-class practice

What is the grey coding for decimal numbers 10 and 15, respectively?

Real-valued representation

Real-valued representation has no hamming cliffs issue

- Any binary arithmetic operator can be used as crossover
- For example, "average"

$\boldsymbol{x}_1$: 0.21 1.87 3.66 1.11 2.25 $\boldsymbol{x}_2$: 2.32 0.77 2.99 2.56 0.11

$\boldsymbol{x}'$: 1.27 1.32 3.33 1.84 1.18

Real-valued representation

Heuristic crossover (where $\boldsymbol{x}_1$ is no worse than $\boldsymbol{x}_2$):

$$\boldsymbol{x}' = u(\boldsymbol{x}_1 - \boldsymbol{x}_2) + \boldsymbol{x}_1 \quad \text{where} \quad u = rand([0, 1])$$

Heuristic crossover example: Assume $f(\boldsymbol{x}_1) \geq f(\boldsymbol{x}_2)$ and $x_i \in [0, 4]$

$\boldsymbol{x}_1$: 0.21 1.87 3.66 1.11 2.25 $\qquad$ $\boldsymbol{x}_2$: 2.32 0.77 2.99 2.56 0.11

$$0.13 \cdot (0.21 - 2.32) + 0.21 \;,\; 0.47 \cdot (1.87 - 0.77) + 1.87 \quad \ldots$$

$$\boldsymbol{x}'\text{: } -0.06 \quad 2.39 \quad \ldots$$

GA troubleshooting

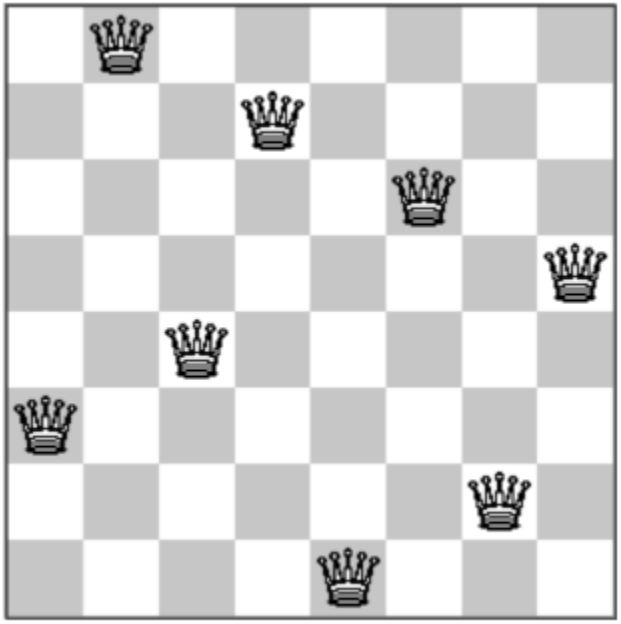
- Individual representation can largely affect the quality of the solution returned by GA. So if you decide to use GA, you should think about the best way to encode solutions for a given problem.

- Try a few different crossover and mutate operators for a given problem.

- If your selected crossover and mutate operators may generate **"invalid offsprings"**, you may assign those "invalid offsprings" **a high penalty in the fitness function** so that they can be excluded from the next generation automatically.

Famous 8-queen problem

The *8* queens puzzle is the problem of placing eight chess queens on an 8 × 8 chessboard so that no two queens attack each other



Famous 8-queen problem

How should we encode each chessboard as a chromosome, i.e., what would be the **best way to create the numeric representation for each chessboard**?

You will work on that in Project 4

GA versus NN

Advantages of GAs

- Does not require any derivative information (failure of gradient based methods).
- Optimizes both continuous and discrete functions and also multi-objective problems.
- Provide near-optimal solutions in a short amount of time
- Provides a list of “good” solutions and not just a single solution.
- Useful when the search space is very large and there are a large number of parameters involved.

Limitations of GAs

- GAs are not suited for all problems where derivative information is available.
- Fitness value is calculated repeatedly which might be computationally expensive for some problems.
- Being stochastic, there are no guarantees on the optimality or the quality of the solution.

GA implementation

https://github.com/DEAP/deap

Project Homepage » DEAP 1.3.1 documentation »

Next topic
Overview

This Page
Show Source

Docs for other versions
DEAP 1.0 (Stable)
DEAP 0.9

Other resources
Downloads
Issues
Mailing List
Twitter

DEAP documentation

DISTRIBUTED EVOLUTIONARY ALGORITHMS IN PYTHON

DEAP is a novel evolutionary computation framework for rapid prototyping and testing of ideas. It seeks to make algorithms explicit and data structures transparent. It works in perfect harmony with parallelisation mechanism such as multiprocessing and SCOOP. The following documentation presents the key concepts and many features to build your own evolutions.

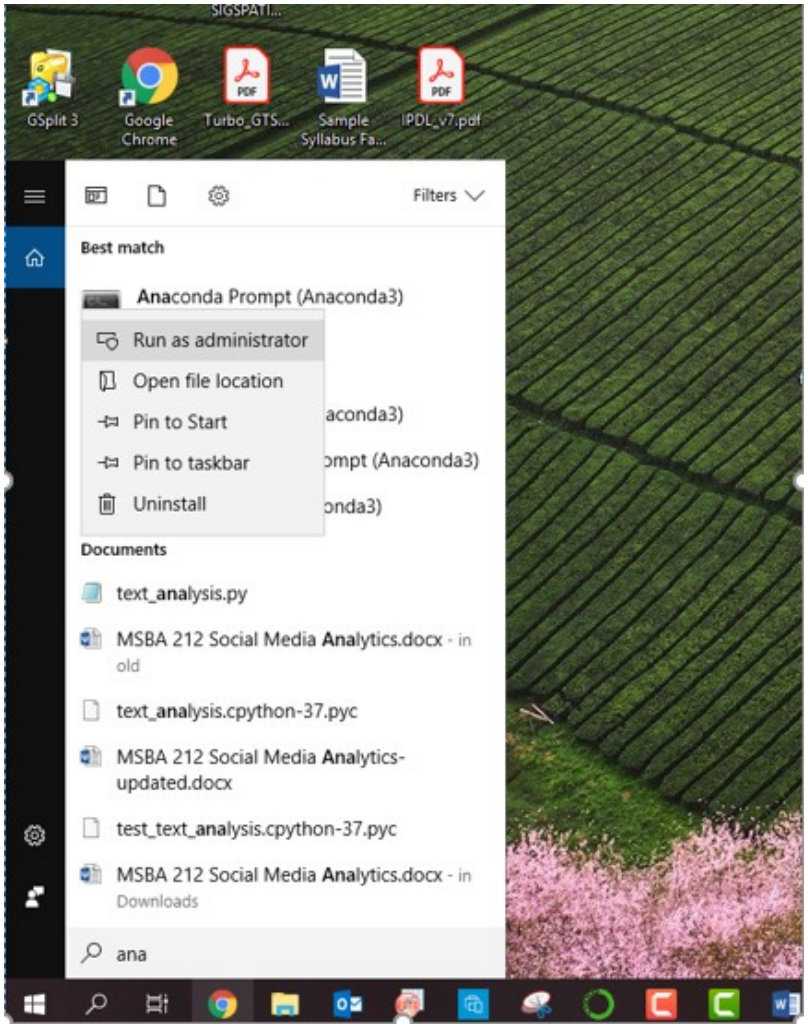
- **First steps:**
  - Overview (Start Here!)
  - Installation
  - Porting Guide
- **Basic tutorials:**
  - Part 1: creating types
  - Part 2: operators and algorithms

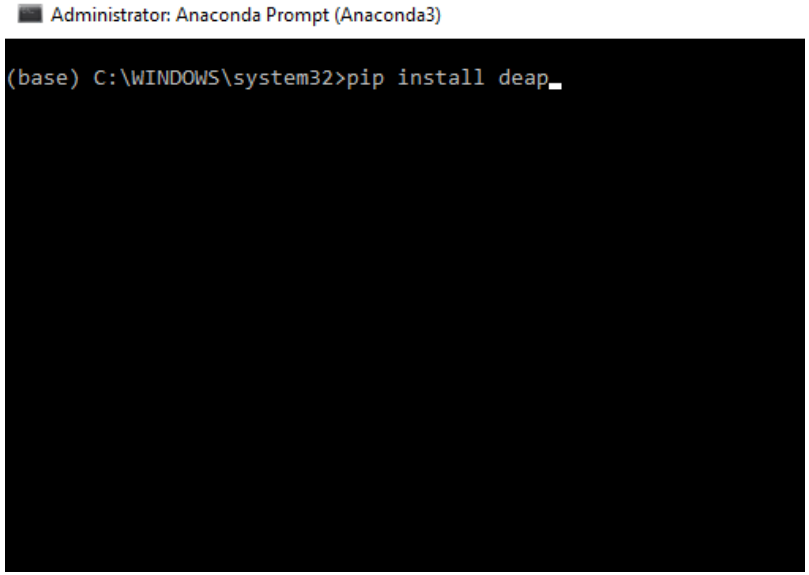
Getting Help

Having trouble? We'd like to help!

- Search for information in the archives of the deap-users mailing list, or post a question.

DEAP installation



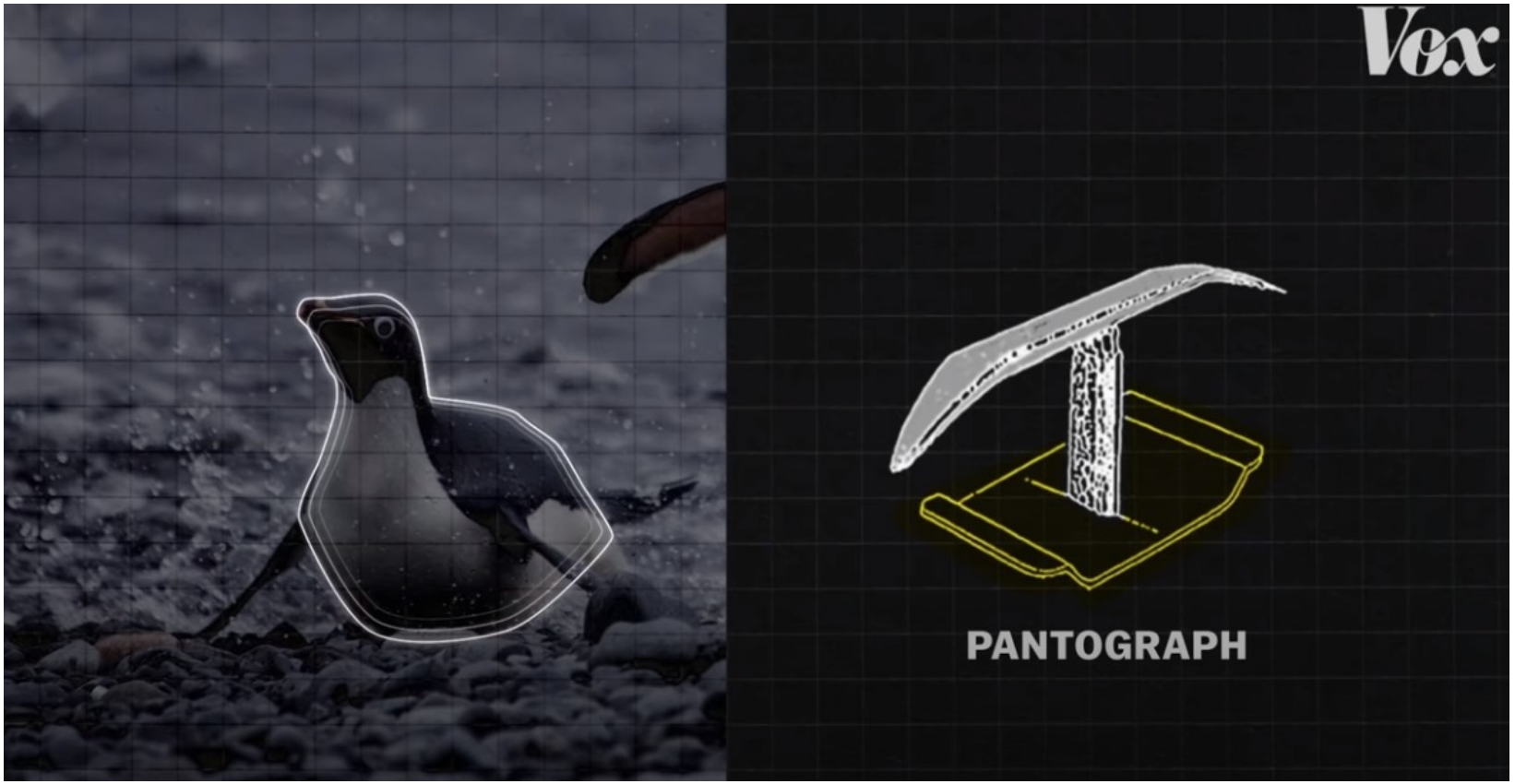


Bioinspired Robotics: Smarter, Softer, Safer

https://www.youtube.com/watch?v=RuLAn3XpYAU&t=184s

Biomimicry design

The world is poorly designed. But copying nature helps.



https://www.youtube.com/watch?v=iMtXqTmfta0