

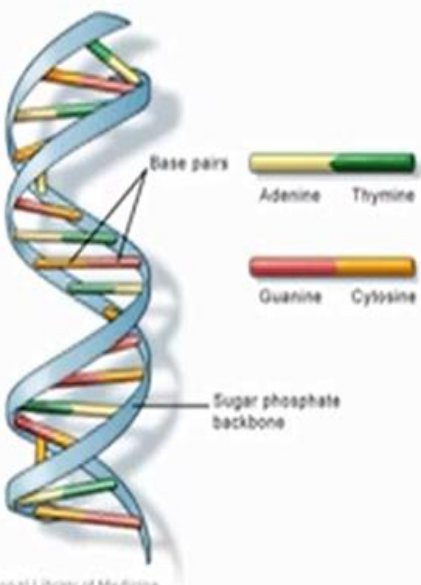
Darwin(1859) states that there is an **implicit struggle for survival**, Because of this implicit struggle, reproducing species create offspring that are genetic variants of the parents. Darwin theorizes that it is this **genetic variation that enables some offspring to survive in a particular environment much better than other offspring with different genetic variations**. These offspring not only survive in the environment, but go on to **reproduce new offspring** that carry some form of this enhanced genetic variation. In addition, those **offspring that are not as suited for the environment do not pass on their genetic variation to offspring, but rather die off**. Darwin then theorizes that over **many generations of reproduction**, new species that are highly adapted to their specific environments will emerge.

It is this theory of natural selection (**Survival for the fittest**) that forms the theoretical foundation for the field of Evolutionary Algorithms (EA).

Outline of this lecture

- Definition of Genetic Algorithms
- The Evolutionary Cycle
- X^2 Example
- Practical Applications Examples
 1. Example1: Solve TSP in GA
 2. Example2: Solve Knapsack Problem in GA

Genetic Algorithms



- ❖ A genetic algorithm is an adaptation procedure based on the mechanics of natural genetics and natural selection.
- ❖ Genetic Algorithms have 2 essential components:
 - ❖ "Survival of the fittest"
 - ❖ Genetic Diversity
- ❖ Originally developed by John Holland (1975).
- ❖ The genetic algorithm (GA) is a search heuristic that mimics the process of natural evolution.
- ❖ Uses concepts of "Natural Selection" and "Genetic Inheritance" (Darwin 1859).

In brief, an EA is a **search algorithm**. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

A population of individuals is maintained within **search space** for a GA, each representing a **possible solution** to a given problem. Each individual is coded as a **finite length** vector of components, or variables, in terms of some alphabet, usually the binary alphabet $\{0,1\}$. To continue the genetic analogy these individuals are likened to **chromosomes** and the variables are analogous to **genes**. Thus a chromosome (solution) is composed of several genes (variables).

A **fitness** score is assigned to each solution. *The GA aims to use selective of the solutions to produce 'offspring' better than the parents by combining information from the chromosomes.*

The common EC terminology uses many synonyms for naming the elements of these two spaces. On the side of the original problem context, **candidate solution**, phenotype, and **individual** are used to denote points of the space of possible solutions. This space itself is commonly called the **phenotype space**. On the side of the EA, genotype, **chromosome**, and again individual can be used for points in the space where the evolutionary search will actually take place. This space is often termed the **genotype space**. Also for the elements of individuals there are many synonymous terms. A place-holder is commonly called a variable, a **locus** (plural: loci), a position, or – in a biology oriented terminology – a **gene**. An object on such a place can be called a value or an allele.

Can we use this Brute-Force idea?

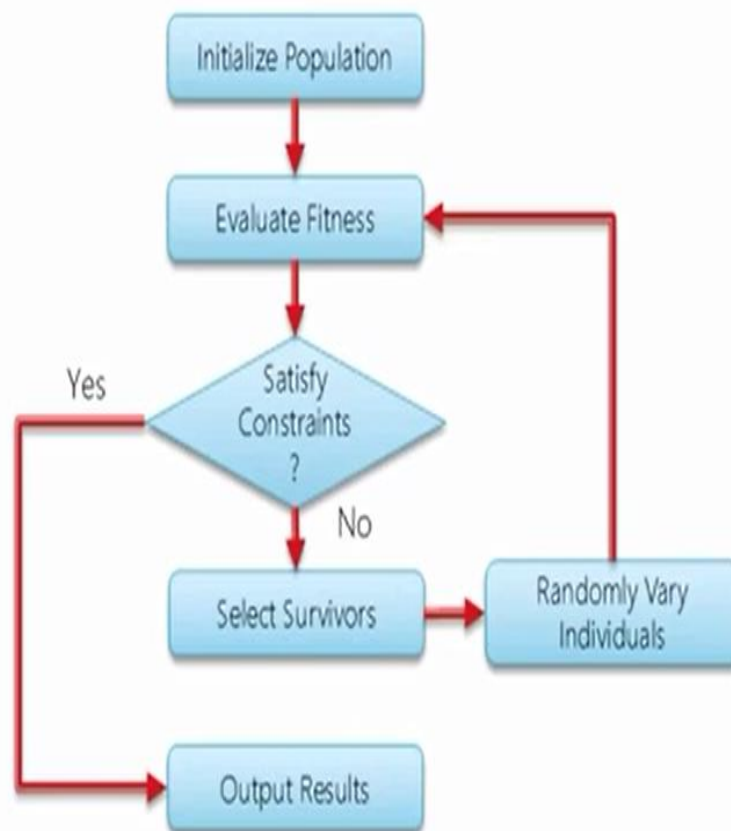
- ❖ Sometimes - **YES**
 - ❖ if there are only a few possible solutions
 - ❖ and you have enough time
 - ❖ then such a method could be used
- ❖ For most problems - **NO**
 - ❖ many possible solutions
 - ❖ with no time to try them all
 - ❖ Therefore, this method cannot be used

The Evolutionary Cycle

Lets go through a simple example to showcase the steps.

Basic Genetic Algorithm Process:

- ❖ Produce an initial population of individuals
- ❖ Evaluate the fitness of all individuals
- ❖ **while** termination condition not met **do**
 - ❖ select fitter individuals for reproduction
 - ❖ recombine between individuals
 - ❖ mutate individuals
 - ❖ evaluate the fitness of the modified individuals
 - ❖ generate a new population
- ❖ **End while**



X² Example

Evolutionary Algorithms is to maximize the function $f(x) = x^2$ with x in the integer interval $[0, 31]$, i.e., $x = 0, 1, \dots, 30, 31$.

1. The first step is encoding of chromosomes; use binary representation for integers; 5-bits are used to represent integers up to 31.
2. Assume that the population size is 4.
3. Generate initial population at random. They are chromosomes or genotypes; e.g., 01101, 11000, 01000, 10011.
4. Calculate fitness value for each individual.

(a) Decode the individual into an integer (called phenotypes),

01101 \rightarrow 13; 11000 \rightarrow 24; 01000 \rightarrow 8; 10011 \rightarrow 19;

(b) Evaluate the fitness according to $f(x) = x^2$,

13 \rightarrow 169; 24 \rightarrow 576; 8 \rightarrow 64; 19 \rightarrow 361.

5. Select parents (two individuals) for crossover based on their fitness in p_i . Out of many methods for selecting the best chromosomes, if roulette-wheel selection is used, then the probability of the i^{th} string in the population is $p_i = F_i / (\sum_{j=1}^n F_j)$, where

F_i is fitness for the string i in the population, expressed as $f(x)$

p_i is probability of the string i being selected,

n is no of individuals in the population, is population size, $n=4$

$n * p_i$ is expected count

Roulette wheel selection (Fitness-Proportionate Selection)

Roulette-wheel selection, also known as Fitness Proportionate Selection, is a genetic operator, used for selecting potentially useful solutions for recombination.

In fitness-proportionate selection :

- the chance of an individual's being selected is proportional to its fitness, greater or less than its competitors' fitness.
- conceptually, this can be thought as a game of Roulette.

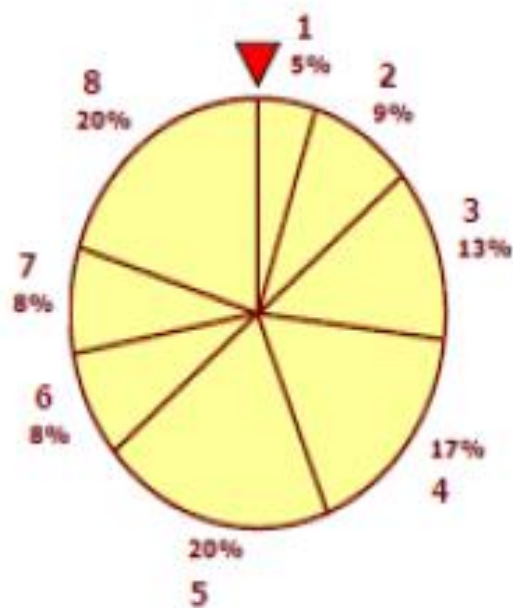


Fig. Roulette-wheel Shows 8 individual with fitness

The Roulette-wheel simulates 8 individuals with fitness values F_i , marked at its circumference; e.g.,

- the 5th individual has a higher fitness than others, so the wheel would choose the 5th individual more than other individuals .
- the fitness of the individuals is calculated as the wheel is spun $n = 8$ times, each time selecting an instance, of the string, chosen by the wheel pointer.

Probability of i^{th} string is $p_i = F_i / (\sum_{j=1}^n F_j)$, where

n = no of individuals, called population size; p_i = probability of i^{th} string being selected; F_i = fitness for i^{th} string in the population.

x² example: selection

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

Crossover

Crossover is a genetic operator that combines (mates) two chromosomes (parents) to produce a new chromosome (offspring). The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents. Crossover occurs during evolution according to a user-definable crossover probability. Crossover selects genes from parent chromosomes and creates a new offspring.

One-Point Crossover

One-Point crossover operator randomly selects one crossover point and then copy everything before this point from the first parent and then everything after the crossover point copy from the second parent. The Crossover would then look as shown below.

Consider the two parents selected for crossover.

Parent 1 **11011 | 00100110110**
Parent 2 **11011 | 11000011110**

Interchanging the parents chromosomes after the crossover points -

The Offspring produced are :

Offspring 1 **11011 | 11000011110**
Offspring 2 **11011 | 00100110110**

Note : The symbol, a vertical line, | is the chosen crossover point.

Two-Point Crossover

Two-Point crossover operator randomly selects two crossover points within a chromosome then interchanges the two parent chromosomes between these points to produce two new offspring.

Consider the two parents selected for crossover :

Parent 1	1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0
Parent 2	1 1 0 1 1 1 1 0 0 0 0 1 1 1 1 0

Interchanging the parents chromosomes between the crossover points -
The Offspring produced are :

Offspring 1	1 1 0 1 1 1 1 0 0 0 0 1 0 1 1 0
Offspring 2	1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0

Uniform Crossover

Uniform crossover operator decides (with some probability – know as the mixing ratio) which parent will contribute how the gene values in the offspring chromosomes. The crossover operator allows the parent chromosomes to be mixed at the gene level rather than the segment level (as with one and two point crossover).

Consider the two parents selected for crossover.

Parent 1	1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0
Parent 2	1 1 0 1 1 1 1 1 0 0 0 1 1 1 1 0

If the mixing ratio is **0.5** approximately, then half of the genes in the offspring will come from parent **1** and other half will come from parent **2**.

The possible set of offspring after uniform crossover would be:

Offspring 1	1₁ 1₂ 0₂ 1₁ 1₁ 1₂ 1₂ 0₂ 0₁ 0₁ 0₂ 1₁ 1₂ 1₁ 1₁ 0₂
Offspring 2	1₂ 1₁ 0₁ 1₂ 1₂ 0₁ 0₁ 1₁ 0₂ 0₂ 1₁ 1₂ 0₁ 1₂ 1₂ 0₁

Note: The subscripts indicate which parent the gene came from.

Arithmetic

Arithmetic crossover operator linearly combines two parent chromosome vectors to produce two new offspring according to the equations:

$$\text{Offspring1} = a * \text{Parent1} + (1 - a) * \text{Parent2}$$

$$\text{Offspring2} = (1 - a) * \text{Parent1} + a * \text{Parent2}$$

where **a** is a random weighting factor chosen before each crossover operation.

Consider two parents (each of 4 float genes) selected for crossover:

Parent 1	(0.3)	(1.4)	(0.2)	(7.4)
Parent 2	(0.5)	(4.5)	(0.1)	(5.6)

Applying the above two equations and assuming the weighting factor **a = 0.7**, applying above equations, we get two resulting offspring.

The possible set of offspring after arithmetic crossover would be:

Offspring 1	(0.36)	(2.33)	(0.17)	(6.87)
Offspring 2	(0.402)	(2.981)	(0.149)	(5.842)

X² example: crossover

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 1	4	0 1 1 0 0	12	144
2	1 1 0 0 0	4	1 1 0 0 1	25	625
2	1 1 0 0 0	2	1 1 0 1 1	27	729
4	1 0 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

Mutation

After a crossover is performed, mutation takes place.

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of chromosomes to the next.

Mutation occurs during evolution according to a user-definable mutation probability, usually set to fairly low value, say 0.01 a good first choice.

Mutation alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With the new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible.

Mutation is an important part of the genetic search, helps to prevent the population from stagnating at any local optima. Mutation is intended to prevent the search falling into a local optimum of the state space.

The Mutation operators are of many type.

- one simple way is, Flip Bit.

Flip Bit

The mutation operator simply inverts the value of the chosen gene.
i.e. 0 goes to 1 and 1 goes to 0.

This mutation operator can only be used for binary genes.

Consider the two original off-springs selected for mutation.

Original offspring 1	1 1 0 1 1 1 1 0 0 0 0 1 1 1 1 0
Original offspring 2	1 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0

Invert the value of the chosen gene as 0 to 1 and 1 to 0


The Mutated Off-spring produced are :

Mutated offspring 1	1 1 0 0 1 1 1 0 0 0 0 1 1 1 1 0
Mutated offspring 2	1 1 0 1 1 0 1 1 0 0 1 1 0 1 0 0

X² example: mutation

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

Solve TSP in Genetic Algorithm



- ❖ One of the major topics in operations research is what is known as the "travelling salesman" problem.
- ❖ The problem can be stated as follows:
 - ❖ There is a set of cities a salesman needs to visit and each city must be visited once.
 - ❖ What's the shortest way through all the cities?
- ❖ This problem has real world implications (but not limited too) related to cost reductions, increase in efficiencies for deliveries, & customer satisfaction.

The Traveling Salesman Problem is defined as: **"We are given a set of cities and a symmetric distance matrix that indicates the cost of travel from each city to every other city. The goal is to find the shortest circular tour, visiting every city exactly once, so as to minimize the total travel cost, which includes the cost of traveling from the last city back to the first city".**

Encoding represent every city with an integer . Consider 6 cities. Thus a path would be represented as a sequence of integers from 1 to 6. This is an example of **Permutation Encoding** as the position of the elements determines the fitness of the solution.

Evaluation The fitness function will be the total cost of the tour represented by each chromosome. This can be calculated as the sum of the distances traversed in each travel segment.

The lesser the sum, the fitter the solution represented by that chromosome.

Distance/Cost Matrix For TSP

	1	2	3	4	5	6
1	0	863	1987	1407	998	1369
2	863	0	1124	1012	1049	1083
3	1987	1124	0	1461	1881	1676
4	1407	1012	1461	0	2061	2095
5	998	1049	1881	2061	0	331
6	1369	1083	1676	2095	331	0

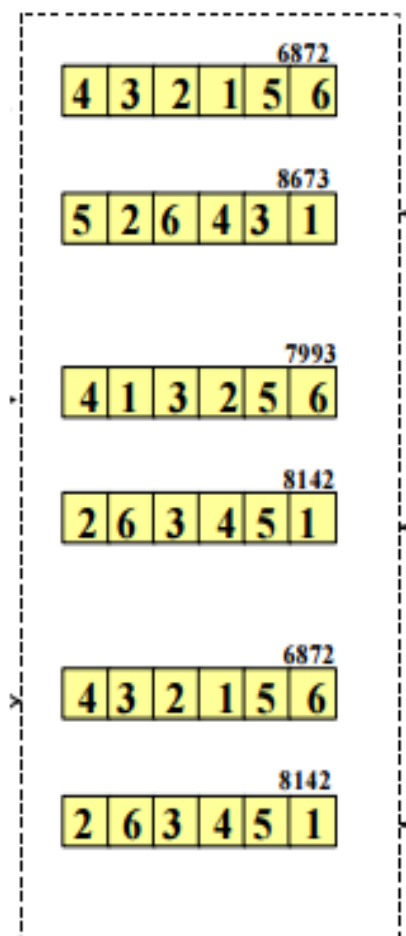
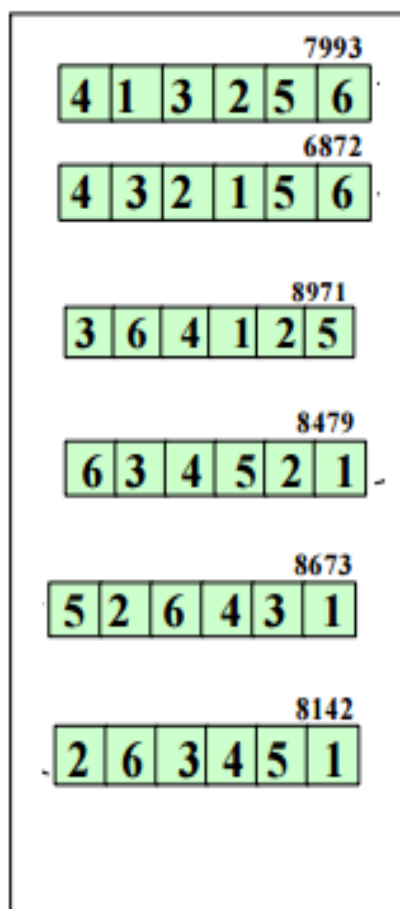
So, for a chromosome **[4 1 3 2 5 6]**, the total cost of travel or fitness will be calculated as shown below

$$\text{Fitness} = 1407 + 1987 + 1124 + 1049 + 331 + 2095 \\ = 7993 \text{ kms}$$

Since our **objective is to Minimize the distance**, the lesser the total distance, the fitter the solution.

Selection

Let us use **Tournament Selection**. As the name suggests *tournaments* are played between two solutions and the better solution is chosen. Two other solutions are picked again and so on...



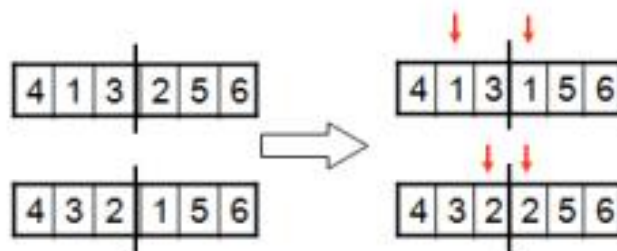
Let the sequence of random individual is:

(1,2),(3,5),(3,1),(4,6),(2,4),(5,6)

Crossover

Why we cannot use single-point crossover?

- Single point crossover method randomly selects a crossover point in the string and swaps the substrings. This may produce some invalid offsprings as shown below:



Crossover in this problem use single point crossover, But, the permutation is copied from the first parent, then the second parent is scanned and if the number is not yet in the offspring it is added, the result of above example is:

4 1 3 2 5 6

Another example:

(1 2 3 | 6 4 5)



(1 2 3 4 5 6)

(4 5 3 6 2 1)

What is the second child?

Mutation

Order changing - two numbers are selected and exchanged

(1 2 3 4 5 6) => (1 5 3 4 2 6)

Let , Replace new offspring with worst individual.

Solve Knapsack Problem in GA

0-1 Knapsack Problem

Formal description: Given two n -tuples of positive numbers

$$\langle v_1, v_2, \dots, v_n \rangle \quad \text{and} \quad \langle w_1, w_2, \dots, w_n \rangle,$$

and $W > 0$, we wish to determine the subset $T \subseteq \{1, 2, \dots, n\}$ (of files to store) that

$$\text{maximizes} \quad \sum_{i \in T} v_i,$$

$$\text{subject to} \quad \sum_{i \in T} w_i \leq W.$$

Remark: This is an optimization problem.

Brute force: Try all 2^n possible subsets T .

Question: Any solution better than the brute-force?



Knapsack Example

- ┌ Typically, a string of 1s and 0s can represent a solution.
- ┌ Possible solutions generated by the system using *Reproduction, Crossover, or Mutations*
 - 1. 0101010
 - 2. 1100100
 - 3. 0100111

Knapsack Example Solution 1

Item	1	2	3	4	5	6	7
Solution	0	1	0	1	0	1	0
Benefit	5	8	3	2	7	9	4
Weight	7	8	4	10	4	6	4

- Benefit $8 + 2 + 9 = 19$
- Weight $8 + 10 + 6 = 24$

Knapsack Example Solution 2

Item	1	2	3	4	5	6	7
Solution	1	1	0	0	1	0	0
Benefit	5	8	3	2	7	9	4
Weight	7	8	4	10	4	6	4

- Benefit $5 + 8 + 7 = 20$
- Weight $7 + 8 + 4 = 19$

Knapsack Example Solution 3

Item	1	2	3	4	5	6	7
Solution	0	1	0	0	1	1	1
Benefit	5	8	3	2	7	9	4
Weight	7	8	4	10	4	6	4

- Benefit $8 + 7 + 9 + 4 = 28$
- Weight $8 + 4 + 6 + 4 = 22$

Knapsack Example

- Solution 3 is clearly the best solution and has met our conditions, therefore, item number 2, 5, 6, and 7 will be taken on the hiking trip. We will be able to get the most benefit out of these items while still having weight equal to 22 pounds.
- This is a simple example illustrating a genetic algorithm approach.

Encoding

In knapsack problem can represent any solution as chromosome by using bit string of length (number of items).

Evaluation

- In our example we will make fitness function as the sum of benefits of all items.

The larger Summation, The Fitter The Solution Represented By That Chromosome.

Selection

Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)

We can use any method of selection for crossover process.

Crossover

- Crossover is performed with probability P_{cross} (crossover probability or crossover rate) between two selected individuals, called *parents*, by exchanging parts of their genes to form two new individuals called *offsprings*. If no crossover was performed, offspring is an exact copy of parents.
- We can use method know as *single point crossover*.

Mutation

- With a mutation probability mutate new offspring at each locus (position in chromosome).
- In our example, It alters some of genes in chromosome with small probability .

Termination Criteria

- Number of generations: in hundreds, in thousands may be in millions according to the problem you have it.
- Convergence: when 95% of populations have the same fitness value we can say the convergence started to appear and the user can stop its genetic program to take the result.

Next Lecture
Linear Regression