CONTENTS

GENETIC ALGORITHM

- Many computational problems in the advanced technology world requires a computer program to be *adaptive- to continue to perform well in the changing environment.*
- *Biological evolution is an appealing source of inspiration for addressing these problems.*
- *One common application of genetic algorithm Is function optimization, mathematically minimizing or maximizing the fitness function that can take several values like* ***f(y)= y+|sin(32 y)| , 0<= y<π.***
- *In short genetic algorithm can be said consist a population of chromosomes and follows Darwin's theory of evolution.*

GENETIC ALGORITHM OPERATORS

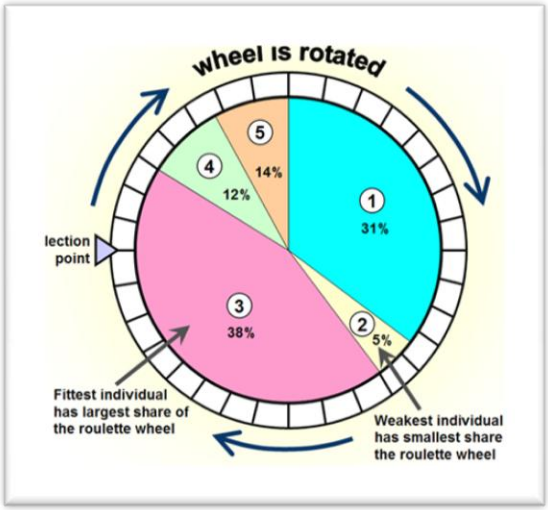
- The simplest form of genetic algorithm involves three types of operators: selection, crossover (single point), and mutation.
- *Selection This operator selects chromosomes in the population for reproduction. The fitter the chromosome, the more times it is likely to be selected to reproduce.*
- *Crossover This operator randomly chooses a locus and exchanges the subsequences before and after that locus between two chromosomes to create two offspring. For example, the strings 10000100 and 11111111 could be crossed over after the third locus in each to produce the two offspring 10011111 and 11100100. The crossover operator roughly mimics biological recombination between two single−chromosome (haploid) organisms.*
- *Mutation This operator randomly flips some of the bits in a chromosome. For example, the string 00000100 might be mutated in its second position to yield 01000100. Mutation can occur at each bit position in a string with some probability, usually very small (e.g., 0.001).*

ENCODING A PROBLEM FOR A GENETIC ALGORITHM

- Binary Encoding : Binary encodings (i.e., bit strings) are the most common encodings for a number of reasons. Much of the existing GA theory is based on the assumption of fixed−length, fixed−order binary encodings.
- Many−Character and Real−Valued Encodings: For many applications, it is most natural to use an alphabet of many characters or real numbers to form chromosomes, examples include , representation of neural network weights.
- Tree Encodings : Tree encoding schemes, such as John Koza's scheme for representing computer programs, have several advantages, including the fact that they allow the search space to be open−ended (in principle, any size tree could be formed via crossover and mutation)

SELECTION METHOD

- **Fitness−Proportionate Selection with "Roulette Wheel" and "Stochastic Universal" Sampling :** Holland's original GA used fitness−proportionate selection, in which the "expected value" of an individual (i.e., the expected number of times an individual will be selected to reproduce) is that individual's fitness divided by the average fitness of the population. The most common method for implementing this is "roulette wheel".



- **Sigma Scaling** : GA researchers have experimented with several "scaling" methods—methods for mapping "raw" fitness values to expected values so as to make the GA less susceptible to premature convergence. One example is "sigma scaling". Under sigma scaling, an individual's expected value is a function of its fitness, the population mean, and the population standard deviation. A example of sigma scaling would be.

ExpVal(I,t)= $1+(f(i)-f(t))/ 2\alpha(t)$ if $\alpha(t)\neq 0$

1.0 if $\alpha(t)=0$

where ExpVal(i,t) is the expected value of individual i at time t, f(i) is the fitness of i,f(t) is the mean fitness of

the population at time t, and Ã(t) is the standard deviation of the population fitnesses at time t

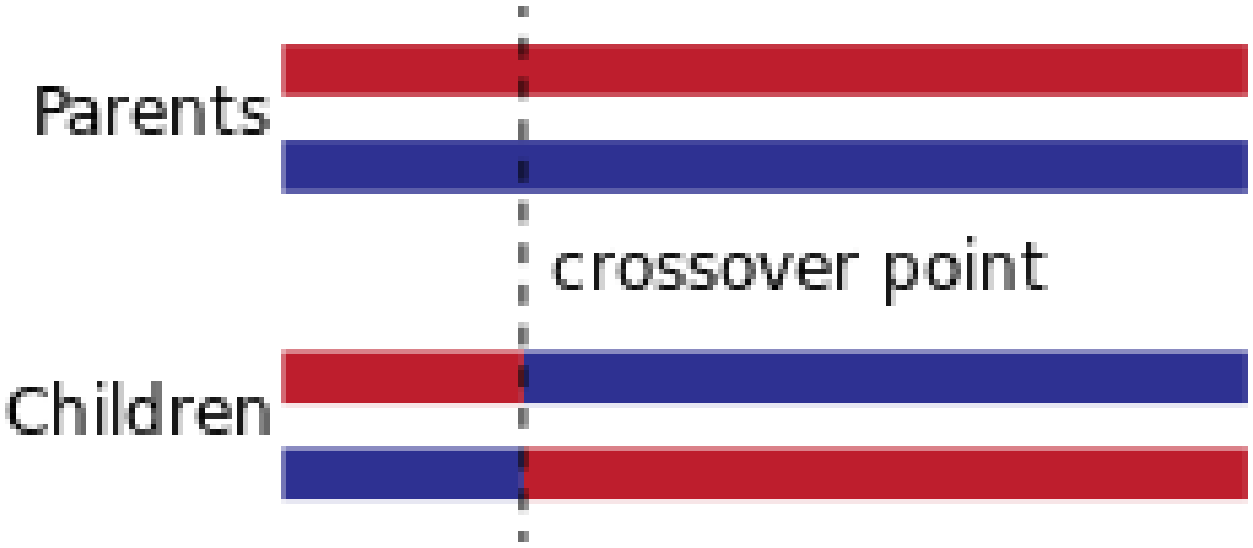
- **Elitism :** "Elitism," first introduced by Kenneth De Jong (1975), is an addition to many selection methods that forces the GA to retain some number of the best individuals at each generation**.**
- **Boltzmann Selection :** Sigma scaling keeps the selection pressure more constant over a run. But often different amounts of selection pressure are needed at different times in a run—for example, early on it might be good to be liberal, allowing less fit individuals to reproduce at close to the rate of fitter individuals, and having selection occur slowly while maintaining a lot of variation in the population.
- **Rank Selection :** Rank selection is an alternative method whose purpose is also to prevent too−quick convergence. In the version proposed by Baker (1985), the individuals in the population are ranked according to fitness, and the expected value of each individual depends on its rank rather than on its absolute fitness.
- **Tournament Selection** : Tournament selection is similar to rank selection in terms of selection pressure, but it is computationally more efficient and more amenable to parallel implementation

CROSSOVER

- Crossover allows sexual reproduction in evolutionary algorithms. In nature, crossover occurs when a single male and female mate and produce offspring. Hermaphrodite organisms, such as snails, can play the role of either mother or father.
- In short, programmers can implement crossover in many ways :-

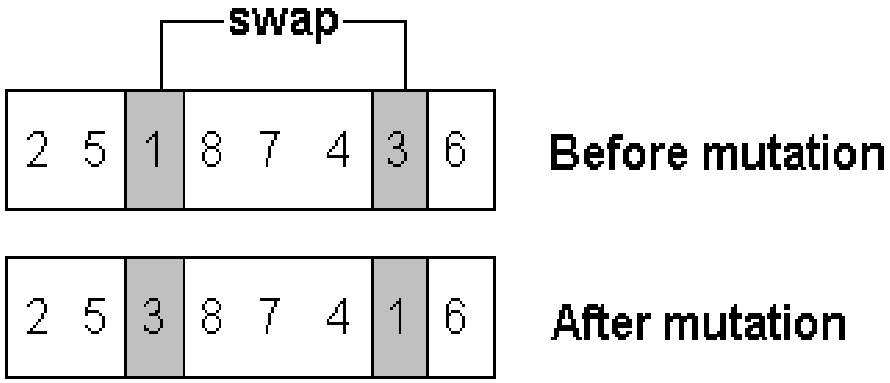
Splice Crossover : Both numeric and categorical data use splice crossover. It works by taking two parents and producing two children (Mitchell, 1998). The parents are split according to two cut points, and this split produces three sub-arrays for each parent that are subsequently

spliced together to produce two children. Each child gets one sub-array from one parent and two sub-arrays from the other parent, like in the example on the next slide

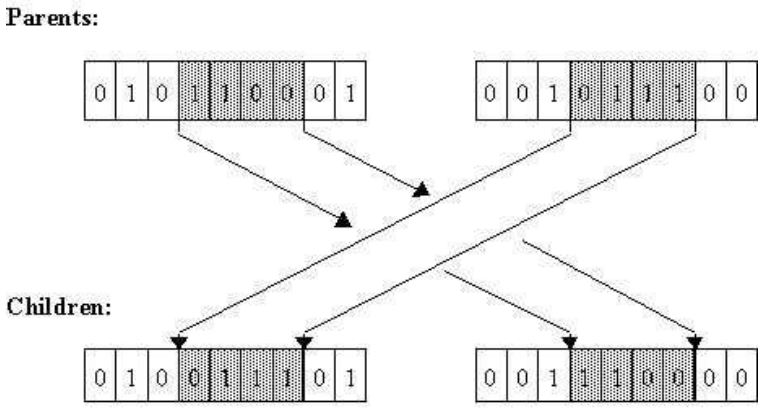


Mutation

- A common view in the GA community, dating back to Holland's book Adaptation in Natural and ARtificial Systems, is that crossover is the major instrument of variation and innovation in GAs, with mutation insuring the population against permanent fixation at any particular locus and thus playing more of a background role. This differs from the traditional positions of other evolutionary computation methods, such as evolutionary programming and early versions of evolution strategies, in which random mutation is the only source of variation.

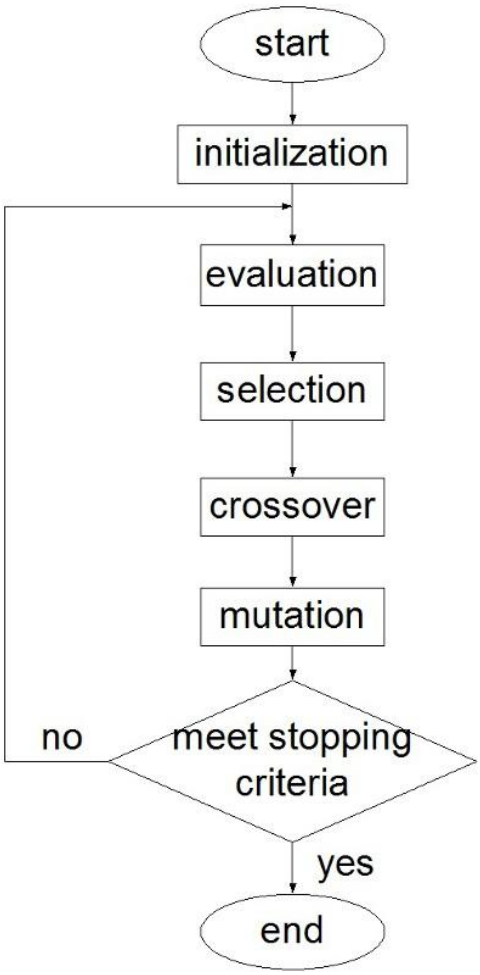


- Mutation in genetic algorithm and genetic programming is generally not random like the Darwin's evolution theory says, here we use, either swapping of bits or characters or random toggling of bits in case of binary encoding, shown in the example on side..
- Mutation rate Is specified before that tells the degree of mutation in the population.



BASIC STEPS FOLLOWED IN THE GENETIC ALGORITHM

•



CLUSTERING USING GENETIC ALGORITHM:

ALGORITHM AND STEPS FOLLOWED IN THE GENETIC ALGORITHM USED

- INPUT:

The input file used is the pixel data of the satellite image, data.txt;
Other input parameters include:

1. **No. of clusters** - 3
2. **Population size** – 10
3. **Centroid initialization** – (1) random initialization from search space (2) randomly selecting existing datapoint.
4. **Selection type** - (1) roulette wheel (2) tournament selection
5. **Tournament size** - Size of the tournament if tournament selection is selected.
6. **Crossover type** - (1) One point crossover (2) arithmetic crossover
7. **Mutation type** - (1) swap mutation (2) Gaussian mutation
8. **Number of iterations** – 100/50
9. **Crossover probability** - 0.7

INITIALIZATION AND FITNESS EVALUATION

Initialization

No. of Pixels = dataset length

Dataset [ ] = pixels

Individual = chromosome [100] ← Population size

Centroid [No. of clusters (3)]

// If centroidInitialization = 1 (random initialization)

range Min [3] = Minimum value of dimensions
range Max [3] = Maximum value of dimensions

Centroid [3]
for (y=0 ; y< numCluster)
  for (x=0 ; x < NumDimension)
① [value = range Min[x] + (range Max [x] - range Min [x]) * r.nextDouble()
  centroid[y]. dimension [x] = value ,

// If centroid Initialization = 2 (set centroid = randomly selected point)

random value = select random point in dataset

centroid [i]. dimensions = dataset [randomvalue]. dimensions

Fitness Evaluation

for (i=0 ; i< datasetlength)

$$\text{getfitness} = \frac{\text{Intraclusterdistance [i]}}{\text{Intercluster distance [i]}}$$

$$\text{InterCluster distance} = \frac{\sum_{0}^{y=\text{datasetlength}} \text{Euclidian distance (dataset[y], Nearest centroid)}}{Y}$$

$$\text{IntraCluster distance} = \frac{\text{for } (x=0 ; x<3) \; \{ \text{for } (y=0 ; y<3) \; \{\text{Euclidian distance (centroid [x], centroid [y])}}}{Y}$$

$$\text{Euclidian distance} = \sqrt{(R_1-R_2)^2 + (G_1-G_2)^2 + (B_1-B_2)^2}$$

Pixel1 → $R_1 G_1 B_1$
Pixel2 → $R_2 G_2 B_2$

Selection

// Type=1 // roulette wheel

Sum of fitness = 0

Sum of fitness += chromosome [i]. getfitness (i=0 ; i< datasetlength)

Select random value -- (using 1)

If randomvalue < fitness array [i]

Return chromosome [i]

// If Type = 2 // Tournament Selection

Array -> Contestant [tournament size]

for (i=0; i < tournament size)

Contestant [i] = ~~randomvalue~~ Chromosome [random value]

Return (fittest among Contestants)

Crossover

// If type = 1 (One point crossover)

rangeMin = 0, rangeMax = numClusters.

Select random value (using ①)

// Swap centroids bet'n both parents.

-> Parent_1

-> Parent_2

double temp = Parent 1. centroid [random value]. clone

Parent 1. centroid [random value] = Parent 2. centroid [random value]

Parent 2. centroid [randomvalue] = temp.

Return, (fittest among both siblings after crossover)

// If type = 2 (Arithmetic Crossover)

Centroid [x]. dimension [y] = Average of centroid dimensions of both Parent.

$$\left(\frac{\text{Parent 1. Centroid [x]. dimension [y]} + \text{Parent 2. centroid [x]. dimension [y].}}{2}\right)$$

Return.

Chromosome [dataset length, numDimension, numClusters, dataset, centroid]

// new Individual with modified centroid.

Mutation

// if type = 1 (Swap Mutation)

double Swap 1 = random Centroid 1
double Swap 2 = random Centroid 2.

int randomValue = random Dimensions
double temp = Swap 1. dimension[randomvalue]
Swap 1 . dimension [randomValue] = Swap2. dimension [randomvalue]
Swap 2 . dimension [random value] = temp

// if type = 2 (Gaussian Mutation)

Mutator Centroid = randomCentroid
double Min = get SearchSpace Min [i]
double Max = get Search Space Max [i]
double Gaussian = get next gaussian()

Get Next gaussian {
V_1 = random value
V_2 = random value } {between −0.1 & 0.1}

return
Gaussian = $V_1 * \sqrt{-2 * \frac{\log (V_1^2 + V_2^2)}{V_1^2 + V_2^2}}$

}

While (Mutator Centroid . dimension [i] + gaussian < Min or > Max)
call (get Next Gaussian)

Mutator Centroid . dimension [i] += Gaussian [for(i=0; i<3)]

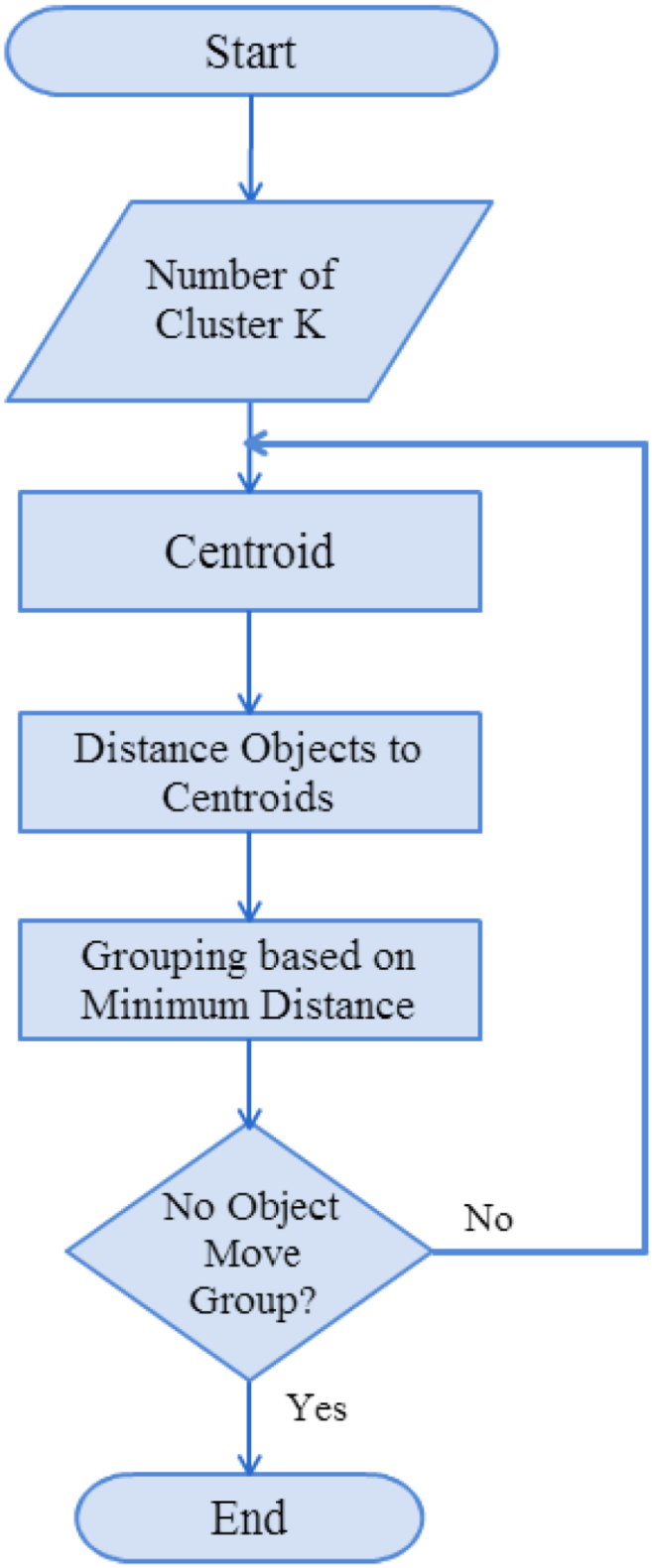
→ for (i=0; i < numIterations ; i++)
→ Repeat Mutation & crossover

→ Retrieve the fittest Parent (Individual) from final population.

→ fittest parent. centroid [x] . dimension [y] = optimized centroid dimensions.

→ Output = optimized Centroid dimensions.

BASIC STEPS FOLLOWED IN KMEANS ALGORITHM(FLOWCHART)



STEPS OF THE ALGORITHM USED IN THE FINAL CLUSTERING OF IMAGE

K-Means

→ Minimizing the square of distance (Euclidian) from data point to the cluster,

$$\arg\min_{c} \sum_{i=1}^{k} \sum_{x \in C_i} d(x, \mu_i)^2 = \arg\min_{c} \sum_{i=1}^{k} \sum_{x \in C_i} \| x - \mu_i \|_2^2$$

Algorithm

- Input Image
- Retrieve pixel values
- Initialize ~~rand~~ the number of clusters

  μ_i = something , $i = 1, 2, 3 \cdots K$
- Initialize random clusters in the dataset.
- Assign each object (pixel) to the group that has the closest centroid

  $|C|$ = No of elements in C

$$C_i = \{ j : d(x_j, \mu_i) \leq d(x_j, \mu_t), \ t \neq i, \ j = 1, \cdots, K \}$$

- Set the position of each cluster to the mean of all data points belonging to that cluster.
- Repeat assigning the object (pixels) until convergence // till the centroids don't move.

Disadvantages

→ Different initial partitions can result in different final clusters

→ Generate empty clusters at different instances due to bad initialization.