



# Genetic Algorithm

## Introduction to Operators



# Representation of Individuals (Encoding)

- The most critical in any application is to decide how best to represent a candidate solution – **encoding**
- After this, decide about the crossover and mutation operators suitable for that representation
- In choosing a representation for a specific problem, one has to make sure that the encoding allows all possible solutions to be represented



# **Representation of Individuals (Encoding)**

- Types of representations usually used:
  - Binary representation
  - Integer representation
  - Real-valued or floating-point representation
  - Permutation representation



# Binary Representation

- Candidate solution consists simply of a string of binary digits – a **bit-string**
- For a particular application we have to decide how long the string should be, and how we will interpret it



## Integer Representation

- Binary representations are not always the most suitable if our problem more naturally maps onto a representation where different genes can take one of a set of values
- For example: we are trying to evolve a path on a square grid, we might restrict the values of the set  $\{0, 1, 2, 3\}$  representing {North, East, South, West}.



# Real-Valued or Floating-Point Representation

- Often the most sensible way to represent a candidate solution to a problem is to have a string of real values
- This occurs when the values that we want to represent as genes come from a continuous rather than discrete distribution



# Permutation Representation

- Many problems naturally take the form of deciding on the order of in which a sequence of events should occur
- An ordinary GA string allows numbers to occur more than once
- We need new variation operators to preserve the permutation property that each possible value occurs exactly once in the solution



# Permutation Representation

- There are two classes of problems that are represented by permutations
- In first, the **order** in which events occur is important
- For example, in “job scheduling” problem the order of occurrence of tasks is more important; in this case  $[1, 2, 3, 4]$  and  $[1, 3, 2, 4]$  have similar fitness if tasks 1, 2 and 3 should complete before task 4



# Permutation Representation

- In second, order based problems depends on **adjacency**
- For example, in Travelling Salesman Problem (TSP) two tours [1, 2, 3, 4] and [2, 3, 4, 1] are same tours if starting point of the tour is not important



# Permutation Representation

- There are two ways to encode a permutation
  - In first, which most commonly used, the  $i^{th}$  element of the representation denotes the event that happens in that place in the sequence
  - In the second, the value of the  $i^{th}$  element denotes the position in the sequence in which the  $i^{th}$  event happens
- For Example: for the four cities [A, B, C, D], and the permutation [3, 1, 2, 4], the first encoding denotes the tour [C, A, B, D] and the second [B, C, A, D]



## Mutation

- This is the variation operator that use only one parent and create only one child by applying some kind of randomized change to the representation



# Mutation for Binary Representation

- The most common mutation operator used for binary encodings is **bit-flip mutation**
- Considers each gene separately and allows each bit to flip with a small probability  $p_m$
- The actual number of values changed is thus not fixed, but depends on the sequence of random numbers drawn
- For an encoding of length  $L$ , on average  $L.p_m$  values will be changed

1	0	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---	---



1	0	0	1	0	0	0	0	0
---	---	---	---	---	---	---	---	---



# Mutation operators for Integer Representation

- Two different ways, both of which mutate each gene independently with user-defined probability  $p_m$ 
  - **Random Resetting**
    - An extension of binary bit-flip mutation
    - With probability  $p_m$  a new value is chosen at random from the set of permissible values in each position
  - **Creep Mutation**
    - Adding a small (positive or negative) value to each gene with probability  $p_m$



# Mutation Operators for Real-Valued of Floating-Point Representation

- The value of each gene is randomly changed within its domain given by a lower  $L_i$  and upper  $U_i$  bound, resulting in the following transformation:

$$\langle x_1, \dots, x_n \rangle \rightarrow \langle x'_1, \dots, x'_n \rangle \text{ where } x_i, x'_i \in [L_i, U_i]$$

- According to probability distribution, there may be two types as follows:
  - Uniform mutation
  - Non-uniform mutation



# Mutation Operators for Real-Valued of Floating-Point Representation

- **Uniform mutation**
  - The value of  $x'_i$  is drawn uniformly randomly from  $[L_i, U_i]$
  - It is most straight forward option and is analogous to binary bit-flip mutation
- **Non-uniform mutation**
  - It is analogous to creep mutation for integers
  - A gene is added by a value which is drawn randomly from a Gaussian distribution

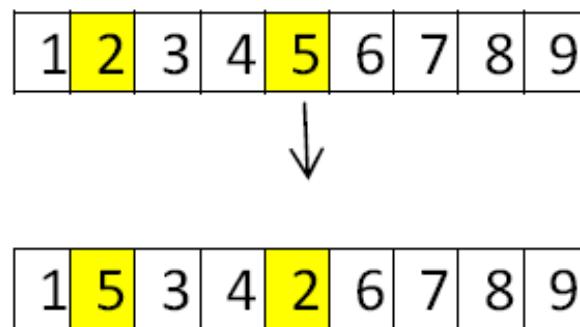


## Mutation for Permutation Representations

- It is no longer possible to consider each gene independently; rather finding legal mutations is a matter of moving alleles around in the chromosome
- Following are the three order-based mutations for first type of representations where the **order** in which events occur is important:
  - Swap mutation
  - Insert mutation
  - Scramble mutation

# Mutation for Permutation Representations

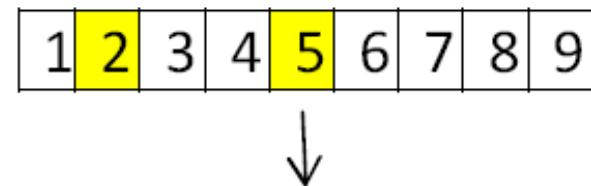
- For second type of representation, where **adjacency** is important, “inversion mutation” is more common as the above three mutations can cause huge number of links to be broken
- **Swap Mutation**
  - It randomly picks two positions (genes) in the string and swaps their allele values.



# Mutation for Permutation Representations

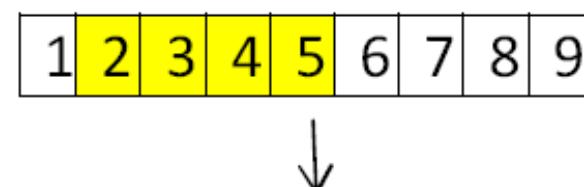
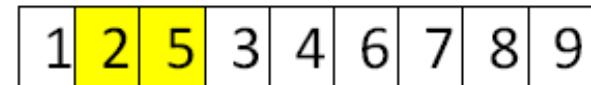
- **Insert Mutation**

- It picks two alleles at random and moves one so that it is next to the other, shuffling along the others to make room



- **Scramble Mutation**

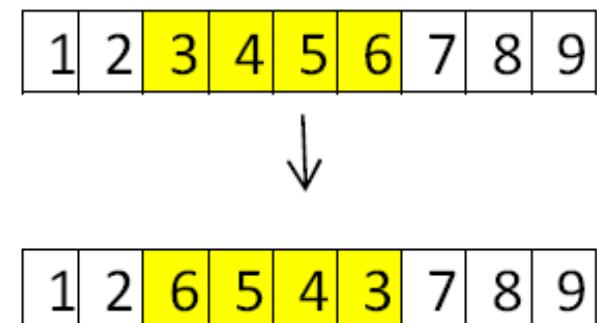
- Here, the entire string, or some randomly chosen subset of values within it, have their position scrambled



# Mutation for Permutation Representations

- **Inversion Mutation**

- It works by randomly selecting two positions in the string and reversing the order in which the values appear between these two positions
- It effectively breaks the string into three parts with all links inside a part being preserved, and only the two links between the parts being broken
- The inversion of a randomly chosen substring is the thus smallest change that can be made to an adjacency-based problem





## Recombination (Crossover)

- One of the most important features in genetic algorithm as it creates a new solution
- Recombination operator is applied probabilistically according to a crossover rate  $p_c$
- If the random variable drawn from  $[0, 1)$  is lower than  $p_c$ , two offsprings are created via recombination of the two parents
- Otherwise they are created by copying the parents.

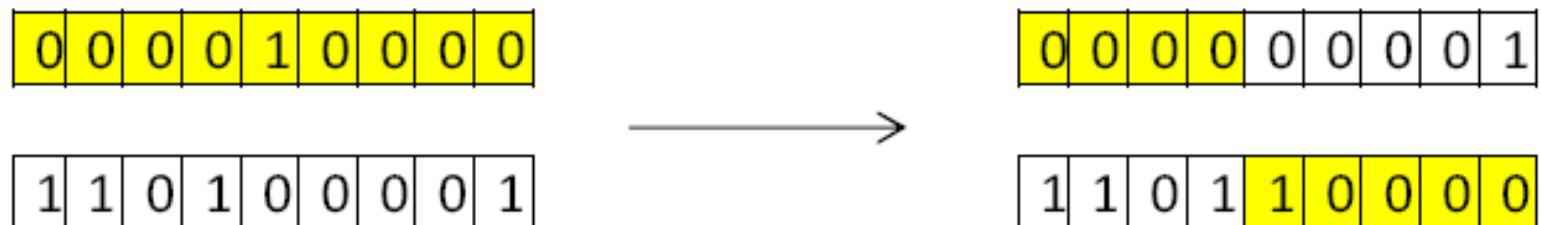


# Recombination Operators for Binary Representations

- Three standard forms of recombination:
  - One-point crossover
  - N-point crossover
  - Uniform crossover

# One-point Crossover

- It works by choosing a random number  $r$  in the range  $[1, l-1]$  (here,  $l$  is the length of encoding), and then splitting both parents at this point and creating the two children by exchanging the tails



# N-point Crossover

- One-point crossover can easily be generalized to n-point crossover, where the representation is broken into more than two segments of contiguous genes, and then the offspring are created by taking alternative segments from the two parents

0	0	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---	---



1	1	0	1	0	0	0	0	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

1	1	0	1	1	0	0	0	1
---	---	---	---	---	---	---	---	---

# Uniform Crossover

- It works by treating each gene independently
- Make a random choice as to which parent it should be inherited from
- In each position, if the value is below a parameter  $p$  (usually 0.5), the gene is inherited from the first parent; otherwise from the second
- The second offspring is created similarly using the inverse mapping

0	0	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---	---



1	1	0	1	0	0	0	0	1
---	---	---	---	---	---	---	---	---

0	1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

1	0	0	1	1	0	0	0	1
---	---	---	---	---	---	---	---	---



# Recombination Operators for Integer Representations

- It is normal to use the same set of operators as for binary representations



# Recombination operators for Floating-point Representations

- There are two options:
  - Discrete recombination
  - Intermediate or arithmetic recombination



## Discrete Recombination

- It is similar to binary or integer recombination
- It has the property that if we are creating an offspring  $z$  from parents  $x$  and  $y$ , then the allele value for gene  $i$  is given by  $z_i = x_i$  or  $y_i$  with equal likelihood



# Arithmetic Recombination

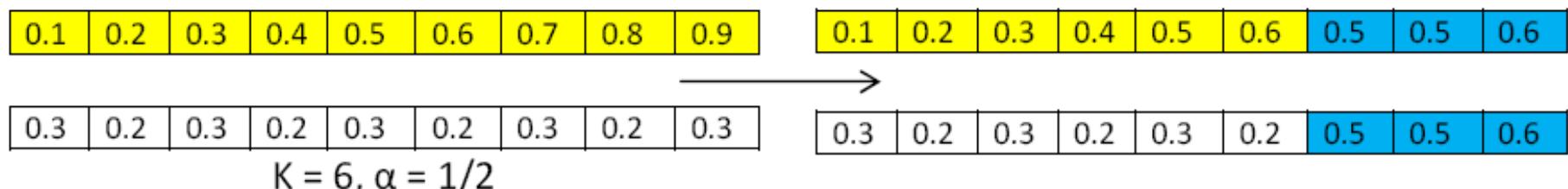
- In each gene position, it creates a new allele value in the offspring that lies between those of the parents
- We have  $z_i = \alpha x_i + (1 - \alpha)y_i$  for some  $\alpha$  in  $[0, 1]$
- In this way the recombination is now able to create new gene material
- Three types of arithmetic recombination operators:
  - **Simple Recombination**
  - **Single Arithmetic Recombination**
  - **Whole Arithmetic Recombination**

# Simple Recombination

- First pick a recombination point  $k$ , then,
- for child 1, take the first  $k$  floats of parent 1 and put them into the child
- The rest is the arithmetic average of parent 1 and 2:

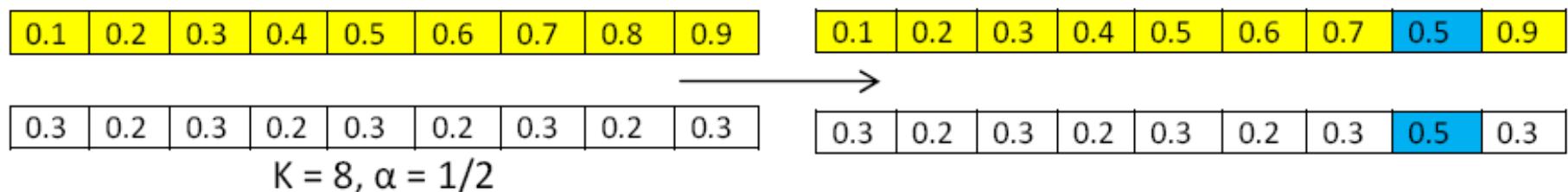
Child 1:  $x_1, \dots, x_k, \alpha y_{k+1} + (1-\alpha)x_{k+1}, \dots, \alpha y_n + (1-\alpha)x_n$

Child 2 is analogous, with  $x$  and  $y$  reversed



# Single Arithmetic Recombination

- Pick a random allele  $k$  at that position;
- Take the arithmetic average of the two parents
- The other points are the points from the parents  
$$\text{Child 1: } x_1, \dots, x_{k-1}, \alpha y_k + (1-\alpha)x_k, \dots, x_{k+1}, \dots, x_n$$
- The second child is created in the same way with  $x$  and  $y$  reversal



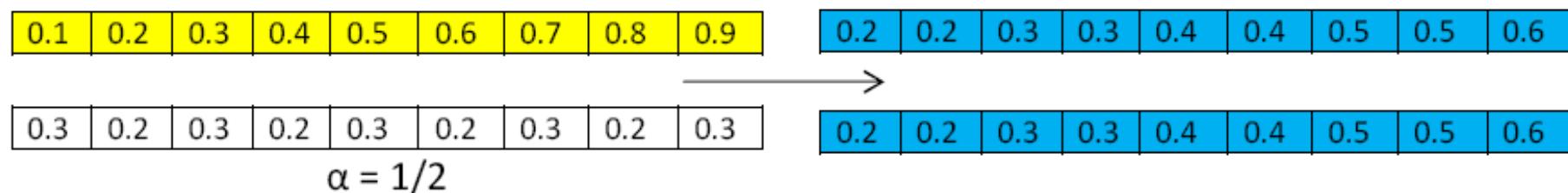
# Whole Arithmetic Recombination

- It works by taking the weighted sum of the two parental alleles for each gene, i.e.,

$$\text{Child 1} = \alpha.x_i + (1-\alpha).y_i \text{ and}$$

$$\text{Child 2} = \alpha.y_i + (1-\alpha).x_i$$

- If  $\alpha=1/2$  the two offsprings will be identical for this operator





# Recombination Operators for permutation Representations

- They are difficult as it is not generally possible to exchange substrings and still maintain permutation property
- Here, we need to take care of either an order in which elements occur or a set of linking pairs of elements
  - **Partially Mapped Crossover (PMX)**
  - **Order Crossover**



## Partially Mapped Crossover (PMX)

- It is one of the most widely used operators for adjacency-type problems, and works as follows:
  - Choose two crossover points at random, and copy the segment between them from the first parent (P1) into the first offspring
  - Starting from the first crossover point look for elements in that segment of the second parent (P2) that have not been copied
  - For each of these (say  $i$ ), look in the offspring to see what element (say  $j$ ) has been copied in its place from P1
  - Place  $i$  into the position occupied by  $j$  in P2, since we know we will not be putting  $j$  there (as we already have it in our string)
  - If the place occupied by  $j$  in P2 has already been filled in the offspring by an element  $k$ , put  $i$  in the position occupied by  $k$  in P2
  - Having dealt with elements from the crossover segment, the rest of the offspring can be filled from P2, and the second child is created analogously with the parental roles reversed

# Partially Mapped Crossover (PMX)

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

Step 1



			4	5	6	7		
--	--	--	---	---	---	---	--	--

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

Step 2, 3, 4 and 5



		2	4	5	6	7		8
--	--	---	---	---	---	---	--	---

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

Step 6



9	3	2	4	5	6	7	1	8
---	---	---	---	---	---	---	---	---



## Order Crossover

- It is used for order based permutation problem, and is very close to PMX crossover
- Its intention is to transmit information about **relative order** from the second parent.
  - Choose two crossover points at random, and copy the segment between them from the first parent (P1) into the first offspring
  - Starting from the second crossover point in the second parent, copy the remaining unused numbers into the first child in the order that they appear in the second parent, wrapping at the end of the list

# Order Crossover

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

Step 1



			4	5	6	7		
--	--	--	---	---	---	---	--	--

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

Step 2



3	8	2	4	5	6	7	1	9
---	---	---	---	---	---	---	---	---

- Create the second offspring in an analogous manner, with the parent roles reversed



# Population Model

- After the variation operators, the other important element in the evolutionary process is survival of individuals based on their relative fitness
- Two different GA models:
  - **Generational model**
    - In each generation, we begin with a population of size  $\mu$ , from which a mating pool of  $\mu$  parents is selected
    - Next,  $\lambda$  ( $=\mu$ ) offsprings are created from the mating pool by the application of variation operators and evaluated
    - After each generation, the whole population is replaced by its offsprings, which is called next generation
  - **Steady-state model**
    - The entire population is not changed at once, but rather a part of it
    - In this case,  $\lambda$  ( $<\mu$ ) old individuals are replaced by the  $\lambda$  new offsprings.
    - The percentage of the population that is replaced is called the generational gap, and is equal to  $\lambda/\mu$



## Population Model

- There are two points in the evolutionary cycle at which fitness based competition can occur:
  - During selection to take part in mating (parent selection)
  - During the selection of individuals to survive into the next generation (survival selection)



## Fitness Proportional Selection (FPS)

- The probability that an individual  $f_i$  is selected for mating is, i.e., the selection probability depends on the absolute fitness value of the individual compared to the absolute fitness values of the rest of the population.
- There are two problems with it:
  - Outstanding individuals take over the entire population very quickly. It is known as premature convergence
  - When fitness values are very close together, there is almost no selection pressure, so the selection is almost uniformly random



## Ranking Selection

- It preserves a constant selection pressure by sorting the population on the basis of fitness
- Allocating selection probabilities to individuals according to their rank, rather than according to their actual fitness values
- The mapping from rank number to selection probability is done usually either linearly decreasing or exponentially decreasing such that the sum of the probabilities over the entire population is unity
- The usual formula for calculating the selection probability for linear ranking schemes uses a parameter  $s$  ( $1.0 < s \leq 2.0$ )

# Ranking Selection

- If the individual has rank  $\mu-1$ , and the worst has rank 0, than the selection probability for an individual of rank  $i$  is

$$P_{\text{lin\_rank}}(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$$

- The table below show how the selection probabilities differ for a population with fitness proportionate and rank-based selection with different values of  $s$

	Fitness	Rank	$P_{\text{selFP}}$	$P_{\text{selLR}} (s=2)$	$P_{\text{selLR}} (s=1.5)$
A	1	0	0.1	0	0.167
B	5	2	0.5	0.67	0.5
C	4	1	0.4	0.33	0.33
Sum	10		1.0	1.0	1.0



## Ranking Selection

- With linear selection probabilities, the applied selection pressure is limited because, on average, an individual of median fitness should have one chance to reproduce, which in turn imposes a maximum value of  $s=2.0$
- If a higher selection pressure is required, i.e., more emphasis on selecting individuals of above average fitness, an exponential ranking scheme is used, of the form:

$$P_{\text{exp\_rank}}(i) = \frac{1-e^{-i}}{c}$$

- The normalization factor  $c$  is chosen so that the sum of probabilities is unity in the entire population