INTRODUCTION TO GENETIC ALGORITHMS PART 2

JAVIER FERNANDEZ

UPNA

SURVIVORS SELECTION

- After having selected λ parents (from a total population of μ individuals) and having crossed them (we will study how to do the crossing later) we have a total of $\lambda + \mu$ (the original population and the descendants obtained) from which we must select the μ that will form the new generation.
- This process is carried out through replacement.
- Such replacement can be of various types.

1.- Replacement based on age.

- It is the most basic replacement strategy.
    - The oldest λ individuals are eliminated.
    - In particular, if $\lambda = \mu$, we recover the generational model that we had previously talked about.

- It is possible to refine this model.
    - For example, it is possible to create a single descendant at a time ($\lambda = 1$) by eliminating one of its parents.
    - However, this is not a recommended strategy.

2.- Fitness-based replacement.

• In this case, the fitness of the λ + μ individuals is taken into account when selecting survivors.

• On many occasions, in addition, it is combined with replacement based on age, in the sense that fitness is used which of the λ members of the previous population should be replaced by the λ descendants.

• In this case, the selection of these individuals can be carried out by any of the parent selection methods that we have seen previously (roulette, ranking, tournament).

In addition, it is also possible to consider other methods such as:

- GENITOR:

- The worst λ individuals from the previous population are selected to be replaced.

- Although it tends to lead to very rapid improvements in the average fitness of the population, it also tends to lead to premature convergence problems.

- ELITISM.

- Often used in combination with age-based replacement and selection methods such as roulette or tournament.

- The idea is to know which individual C is with the best fitness of the previous population.

- If this individual is going to disappear in the new population and if none of the descendants that are going to appear have a better fitness, then C is kept in the new population and one of the descendants is eliminated.

Within the fundamental scheme of genetic algorithms, we still have three fundamental elements to study:

1.- The coding of chromosomes

2.- The crossover operators.

3.- The mutation operators

Since these elements are closely related to each other, we are going to study them together.

1.- CHROMOSOMES WITH BINARY CODING

- We are going to analyze a genetic algorithm with binary coding of the data.
- In other words, we assume that the chromosomes are given as strings of bits (0 and 1).
- First, we must determine how we are going to use the chromosomes to represent the information.
    - o In this sense, we must take into account that:
        1-All chromosomes must be of the same length.

        2-However, it is not necessary that each gene corresponds to a bit.
    - o For example, if we have the individuals:

Crom1: 111011100110

Crom2: 000100001101

We can be representing chromosomes of 12 genes (that is, each bit corresponds to a gene, and therefore, each gene corresponds to 1 or 0), or chromosomes of 6 genes (each gene corresponds to two bits), chromosomes of 3 genes (each gene corresponds to 4 bits), etc ...

• In any case, it must be taken into account that:

1-The representation of the information for all chromosomes must be the same.

That is, if in Chrom1 a gene corresponds to two bits, then for all the chromosomes in this genetic algorithm one gene corresponds to two bits.

2-In general, using representations in which each gene corresponds to more than one bit is not adequate.

It is preferable in these cases to directly use a representation with integers, real numbers, etc.

• In what follows, we consider that we are working with chromosomes of length N and such that each gene corresponds to one bit.

A previous problem: Gray coding

Definition: Given two binary strings (chromosomes) of the same length N:

C1=p1p2...pN

C2=q1q2...qN

The **Hamming distance** between C1 and C2 is defined as:

d(C1,C2)=|p1-q1|+|p2-q2|+...+|pN-qN|

That is to say:

The **Hamming distance** between two chromosomes measures the number of mismatched positions on those chromosomes.

The Hamming distance is a metric and it measures the number of mutations that are required to convert the C1 chromosome into the C2 chromosome**.**

Consider the following three chromosomes (with N=4)

Crom1: 0110

Crom2: 0111

Crom3: 1000

If we decode the chromosomes to see what integer they represent, we get that:

Crom1=6; Crom2=7; Crom3=8

Nevertheless,

d(Crom1,Crom2)=1

d(Crom2,Crom3)=4

• This is an undesirable feature of the usual representation of binary numbers.

• In general, we would like the number of mutations required to add or subtract a unit (in decimal representation) to be the same.

• If other intrinsic considerations of the problem do not come into play, it should be just as likely to go from 7 to 8 as from 7 to 6

- To solve this problem, the so-called **Gray Coding** is introduced.
- This coding is built based on the lengths of the chromosomes, as follows.

N=1 (One bit encoding)

0=0; 1=1.

N=2

0=00; 1=01; 2=11; 3=10

N=3

0=000; 1=001; 2=011; 3=010; 4=110; 5=111; 6=101; 7=100

• In general, given the N-1 order encoding, the N-order encoding is constructed as follows.

1- We reflect the list of chains of length N-1.

For example, to go from N-1 = 2 to N = 3, we start by mirroring the strings of length 2 as follows:

00 01 11 10   10 11 01 00

2-We put a zero before the entries of the original list and a one in front of the reflected entries.

000 001 011 010   110 111 101 100

The result is Gray's encoding for N.

It is important to note the following:

• Gray's encoding is stable in the sense that if a number is ranked k in the N-order encoding, it remains k in any higher-order encoding.

• The code is cyclical. Each element differs by a single bit from the previous one and the last one also differs by a single bit from the first.

MUTATION FOR BINARY CODINGS

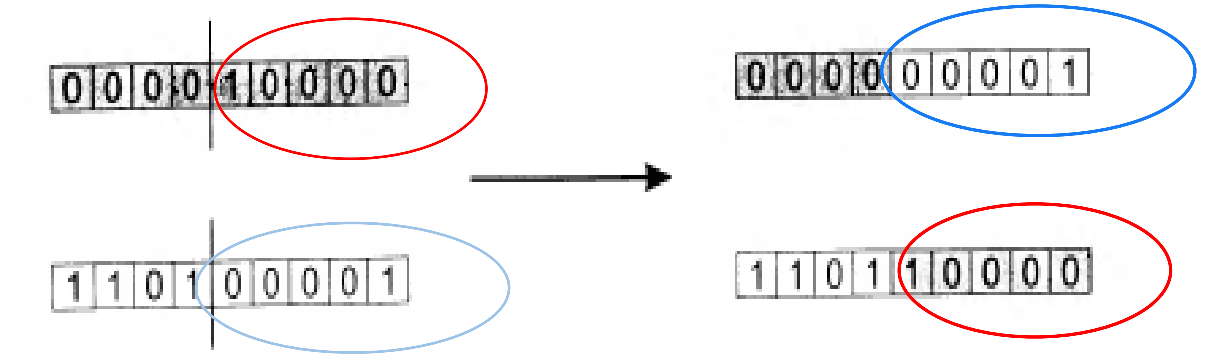
- In general, a mutation is an operator that starts from a single parent and gives rise to a single offspring.
- In the case of binary coding, each gene is usually considered separately.
    - o Each gene can change its value (from 0 to 1 or from 1 to 0) with probability $p > 0$.
    - o Therefore, the total number of mutations is not fixed a priori, but on average pN mutations will occur (on a chromosome of length N).
- In general, the probability p for genetic algorithms is taken very low.
    - o Setting this parameter is not easy, and depends on factors such as, for example, if it is enough to obtain a single chromosome with optimal fitness or if all chromosomes should reach very high fitness.
    - o But, in general, it is intended that, on average, it mutes between one gene per generation and one gene per offspring.
    - o That is, if we have a population of k chromosomes of length L, the mutation probability is usually a number between

      $1/(kL)$ and $1/L$.

CROSSING FOR BINARY CODINGS

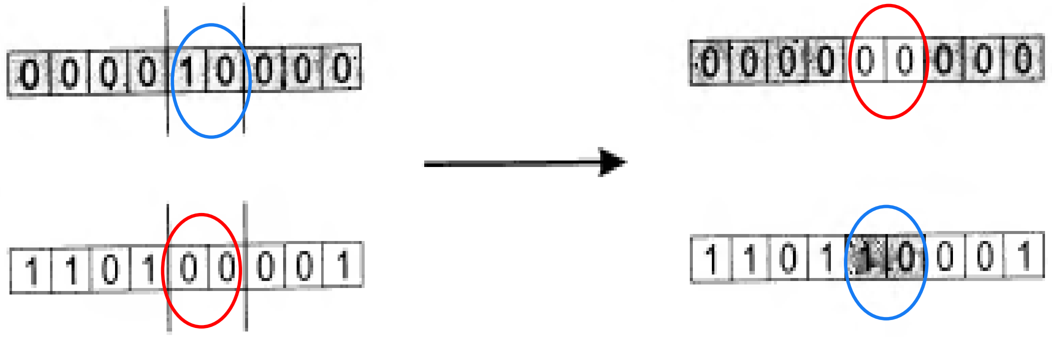
1-Crossover on a point (One point crossover, 1X)

- o Given two chromosomes of length N, choose an integer l between 0 and N-1 and mix:
  - -The first l genes of chromosome 1 with the N-l second genes of chromosome 2.
  - -The first l genes of chromosome 2 with the N-l second genes of chromosome 1.



2-Crossover over n-points (n-point crossover, nX)

It is the generalization of the previous one taking n points.



3- Uniform crossover (UX)

o In contrast to the previous cases, the uniform crossing considers each gene individually and decides randomly from which parent it will be inherited.

o To do this, we generate a string of N random values between 0 and 1.

- If the value falls below a preset parameter (normally 0.5), the value of the first parent is taken for that gene. If not, the value of the second parent.
- The second descendant is obtained by the inverse operation.

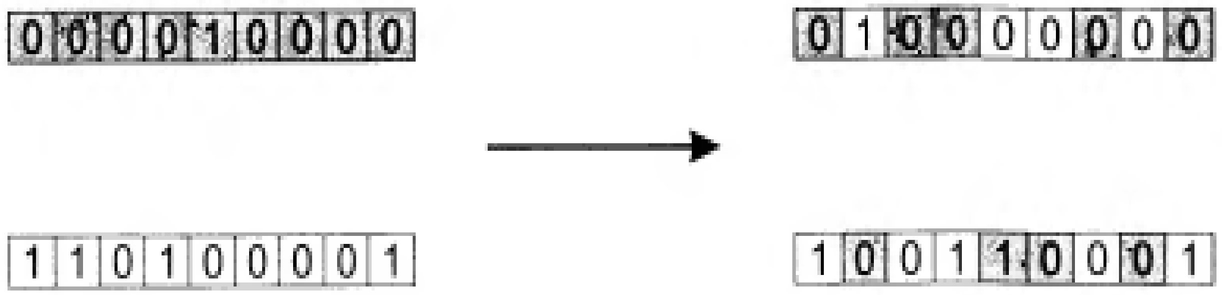


Fig. 3.8. Uniform crossover. In this example the array [0.35, 0.62, 0.18, 0.42, 0.83, 0.76, 0.39, 0.51, 0.36] of random variables drawn uniformly from [0,1) was used to decide inheritance

o It is important to note that in the nX (Crossing over n-points) there is a bias that points that were together on the parent chromosomes stay together on the descendant chromosomes.

o Furthermore, if n is odd, there is also a bias to separate genes located at opposite ends of the parent chromosomes.

o These biases are called positional biases.

o The UX (Uniform crossover) does not present any positional bias.

o However, due to the central limit theorem, there is a bias that each offspring inherits 50% of their genes from each of her parents.

o This is called distributional bias.

o The choice of one or the other of these crossover operators depends on the specific problem considered.

2.- CHROMOSOMES WITH INTEGER CODING

o Although most of the problems with genetic algorithms try to use a binary coding, there are problems where it is more reasonable and / or easier to use other types of representations.

o In this case, the chromosomes are given by strings of integers.

o As in the case of binary coding, the shape of each gene must be determined and fixed in advance and for the entire problem.

o Thus, the chromosome: 1045089806, can consist of:

- 10 genes of length 1: 1,0,4,5,0,8,9,8,0,6
- 5 genes of length 2: 10, 45, 08,98,06
- 2 genes of length 5: 10450, 89806

o In general, it is convenient to consider whether the different values of an attribute (encoded in a gene) are related to each other.

o For example, if we consider the problem of maximizing a function defined in terms of variables defined over sets of integers, we are considering ordinal attributes, and in this case, 134 is more different from 5 than 4 from 2.

o However, if we consider the problem of finding an optimal path in a mesh of squares, we can represent the movements in terms of the integers 1,2,3,4, being:

1=Norte; 2=Este; 3=Sur; 4= Oeste.

En este caso, los atributos tienen un carácter cardinal, sin relación obvia entre ellos.

MUTATION FOR INTEGER CODINGS

There are two basic mutation operators in this case.

- **Random reset:**
  - It is the analog of the mutation operator in binary encoding.
  - In this case, the value of a gene is changed by another value chosen randomly from all possible values, according to a probability p.
  - This is the most recommended mechanism when genes have a cardinal and non-ordinal character.

- **Creep mutation:**
  - It is based on adding a small value (positive or negative) to each gene with probability p.
  - These values to add or subtract change randomly for each position, and are chosen according to a symmetric probability distribution with respect to zero (that is, it is more likely to choose a number the closer it is to zero, and the probability of choosing x and -x is the same).
  - Therefore, various parameters are necessary to control said probability distribution.
  - As this may not be easy, on many occasions a random reset with a low p probability and the slip mutation are combined.

CROSSING FOR INTEGER CODINGS

The same operators are often used as in binary encoding.

3.- CHROMOSOMES WITH REAL CODING (FLOATING-POINT)

- It is assumed that genes can take value in an interval [L, U] (which can be different for each position of the chromosomes).

- Since the representation capacity of real numbers in a machine is limited, we actually consider floating point representations.

- However, when working with this representation we tend to ignore the discretization imposed by the computer and consider that we are working with continuous numbers.

MUTATION FOR FLOATING POINT CODINGS

1-Uniform mutation

- In this case, the value of a gene is changed to another random value in the interval [L, U], with probability p.
- It is the analogue to the mutation in the binary case, and the probability p can **change for each position of the chromosome.**

2-Non-uniform mutation according to a fixed distribution.

- It is the most common form of mutation in problems with floating point encodings.
- We set a Gaussian probability distribution with zero mean and deviation set by the user.
- For each gene, with probability p, we choose a random real value according to this Gaussian distribution.
- We add this value (which can be negative) to the value of the gene and we obtain the new value of said gene.
- If the sum is outside the domain of possible values of the gene [L, U], we take the corresponding end of this interval, depending on whether we have left the top or bottom.

CROSSING FOR FLOATING POINT CODINGS

o In this case, there are two possibilities:

-We can apply the same operators as in the binary or integer cases. In this way, however, we do not introduce new values into the chromosomes other than through mutation.

-We can create for each gene in the descendants a new value that does not correspond exactly to those that appear in the parents.

In particular, if this value is between the values of the corresponding genes in the parents, it is called arithmetic or intermediate crossing.

o We are going to focus on the latter case.

o There are three fundamental types of arithmetic crossover.

o In all of them, a parameter α is set at [0,1] and the value of the offspring's gene is calculated by taking:

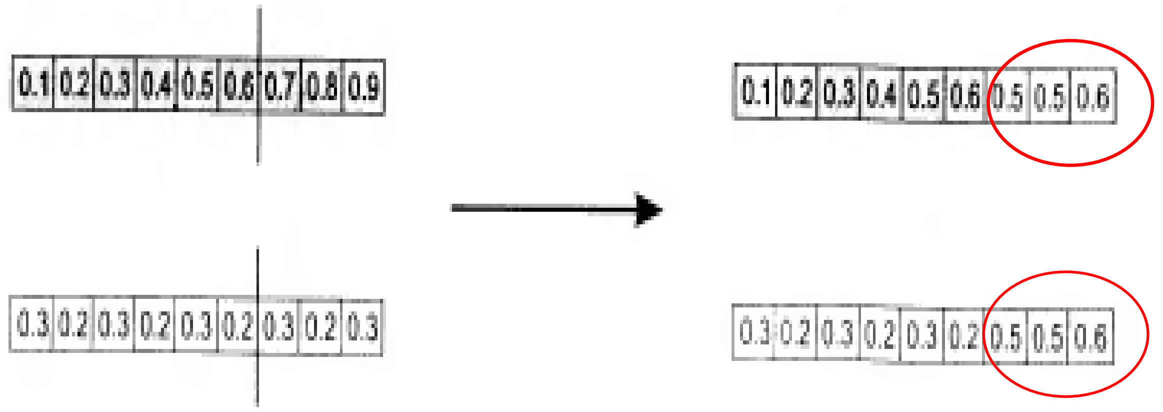
$$x'=\alpha x+(1-\alpha)y$$

where x and y are the values corresponding to that gene in the first and second parents, respectively.

o The arithmetic mean ($\alpha = 0.5$) of the corresponding genes in the parents is usually taken.

1.- Simple crossing.

- o We choose a point k (at random) on the parent chromosomes.
- o The offspring 1 has its first k genes equal to those of the first parent.
- o The rest of her genes are the average (according to the α parameter) of the corresponding genes of the parents.



- o Descendant 2 is obtained in the same way by exchanging the place of the parents.

2.- Individual arithmetic crossing.

- o We choose a k position at random on the parent chromosomes.
- o The genes in that position are combined arithmetically.
- o The rest of the genes of offspring 1 are those of parent 1, and those of offspring 2, those of parent 2.

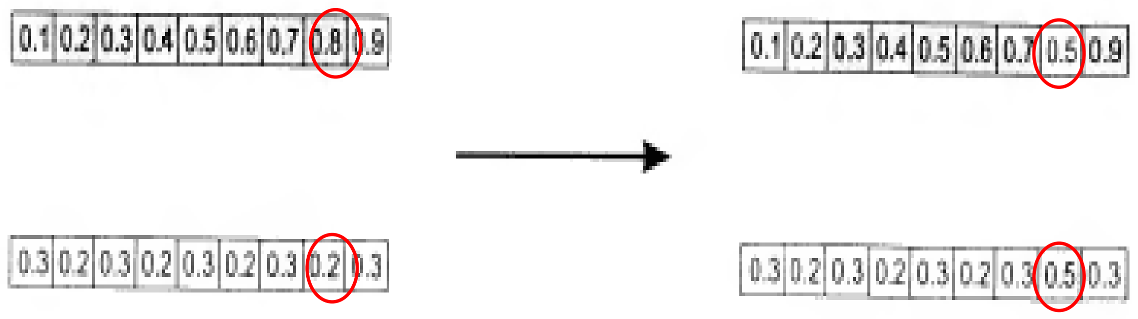


Fig. 3.10. Single arithmetic recombination: $k = 3, \alpha = 1/2$

3.- Total arithmetic crossover.

In this case, we use the expressions:

$$\text{Child 1} = \alpha \cdot \bar{x} + (1-\alpha) \cdot \bar{y}. \qquad \text{Child 2} = \alpha \cdot \bar{y} + (1-\alpha) \cdot \bar{x}.$$

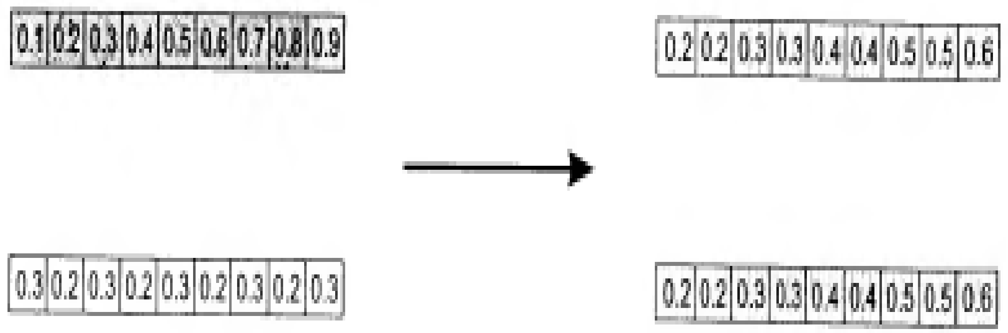


Fig. 3.11. Whole arithmetic recombination: $\alpha = 1/2$

o If we take α = 0.5, as can be seen, the two descendants are equal.

4.- CHROMOSOMES WITH PERMUTATION CODING

o In many problems it is natural to use a representation by means of permutations in the chromosomes.

o In particular, the most frequent is that these permutations are given by strings of integers.

MUTATION FOR PERMUTATION CODINGS

Exchange mutation:

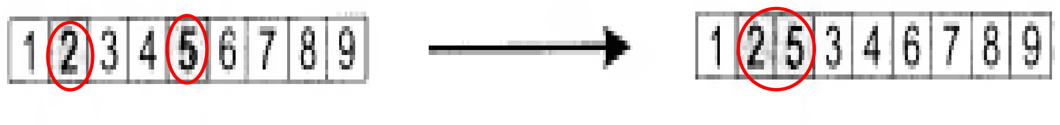
- The simplest procedure is to simply swap two positions. So:

$$(423156) \rightarrow (425136)$$

- The positions to be exchanged are chosen randomly.

Insertion mutation:

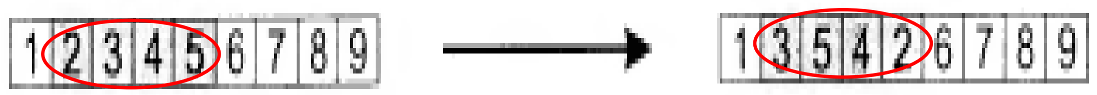
- In this case, we choose two genes at random and move one of them to join it with the other, moving the others to allow it.
- For example, in the following image we have chosen the second and fifth genes.



Shake mutation

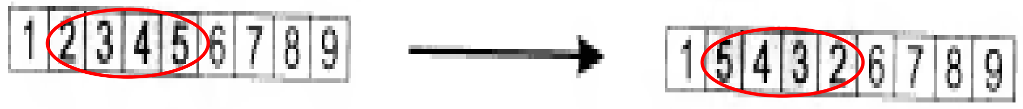
o The positions of all genes or a substring of them are randomly exchanged.

o For example, if we choose the substring between genes 2 and 5, we have



Inversion mutation

o Two positions of the chromosome are chosen at random and the order in which the genes appear between these two points is reversed.



CROSSING FOR PERMUTATION CODINGS

o In this case, the crossover is clearly more complex to obtain since it is not possible to simply exchange two substrings.

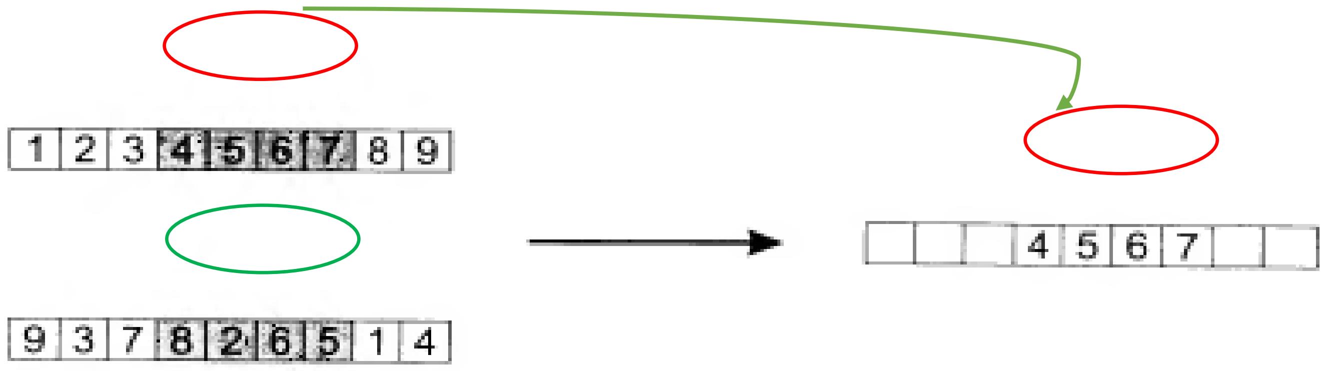
o For this reason, it is important to bear in mind that the representation by permutations is used when, either a given order is to be obtained or a chain of operators that pass from one state to another (adjacency).

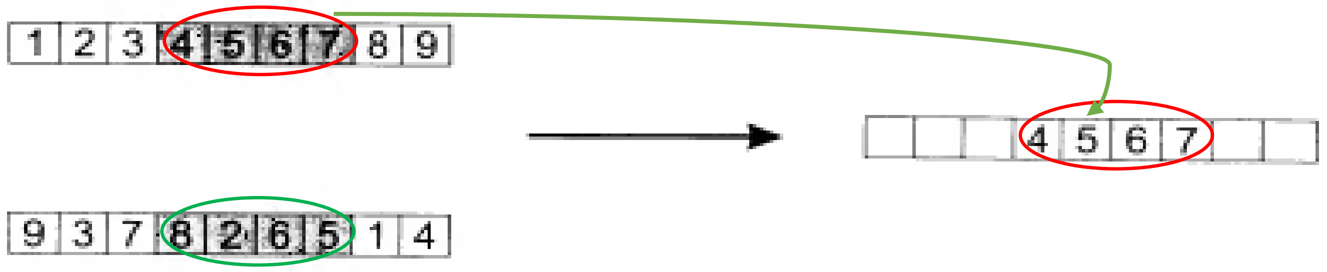
o We are going to study two operators, each one adapted to one of these situations.

Partially mapped crossover (PMX)

- This method is widely used in problems where the fundamental is adjacency.
- The procedure consists of the following stages, denoting by P1 and P2 the first and second parents, respectively.

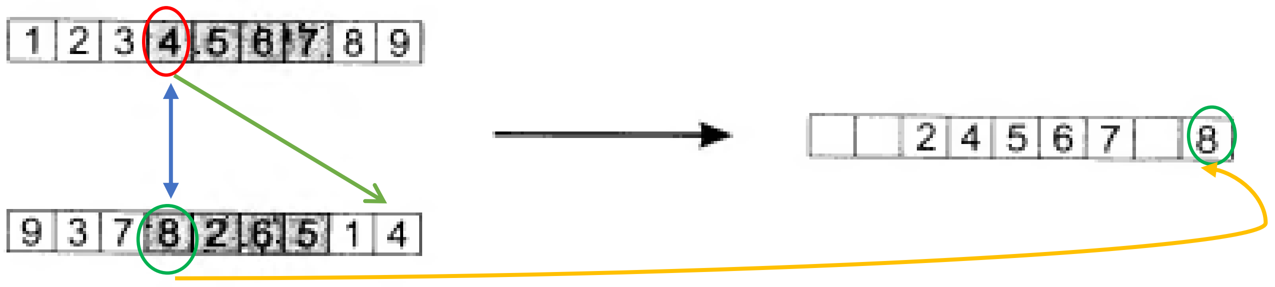
  1- We choose two random points in P1 and copy the elements between them in the first descendant





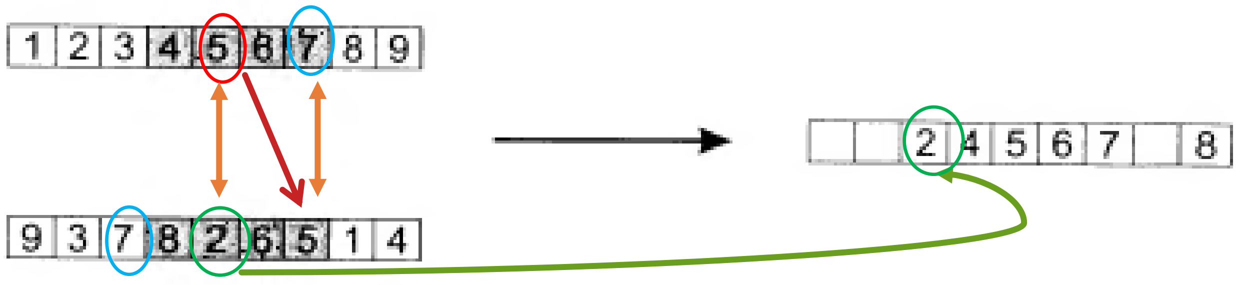
2- From the first crossing point chosen in the previous step, look for elements in P2 that have not been copied.

- o For each of them (say i), find which element j of P1 has been copied in its place.
- o Place i in the position occupied by j in P2.
- o If the position occupied by j in P2 has already been occupied by k, place i in the position occupied by k.
  - o In this case, the 8 in P2 occupies the position of the 4 in P1.
  - o Therefore, we place the 8 in the position occupied by the 4 in P2.

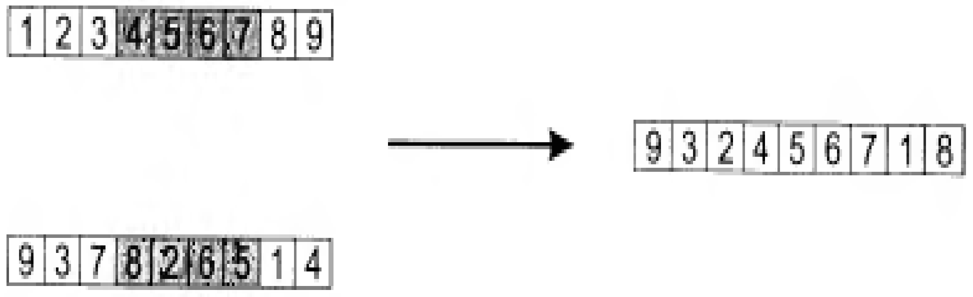


o The 2, for its part, occupies the position of 5 in P1.

- We look for the position of 5 in P2, which is already occupied by 7. Therefore, we place the 2 where the 7 was in P2.



3-The remaining positions are filled with the elements of P2.



The other offspring is generated by reversing the role of the parents.

o It is important to emphasize that in this case not all the information present in the parents is preserved.

o However, it is observed that 6 of the 9 adjacencies in the descendants were already present in the parents.

  o However, of the edges {5,6} and {7,8} that appeared in the parents, only {5.6} has been preserved.

o To try to avoid this loss of information, other operators have been introduced, such as the one we see below:

Edge crossing

The idea of this procedure is to try to preserve as many adjacencies (edges) as possible. For it:

1-We create an adjacency table.

2-We choose an element at random and include it in the descendant.

3-Make current_**element = entry**

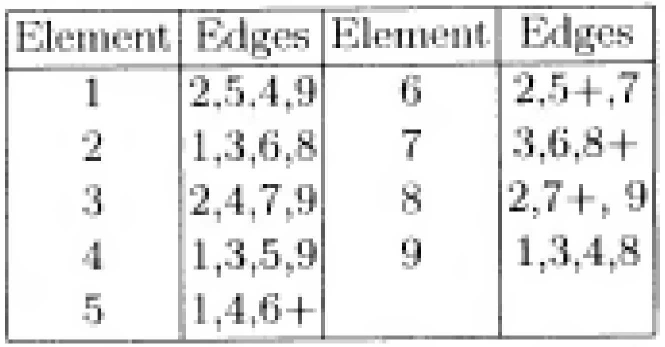
4-Remove all references to **current_element** from the table.

5-Examine the list of **current_element**

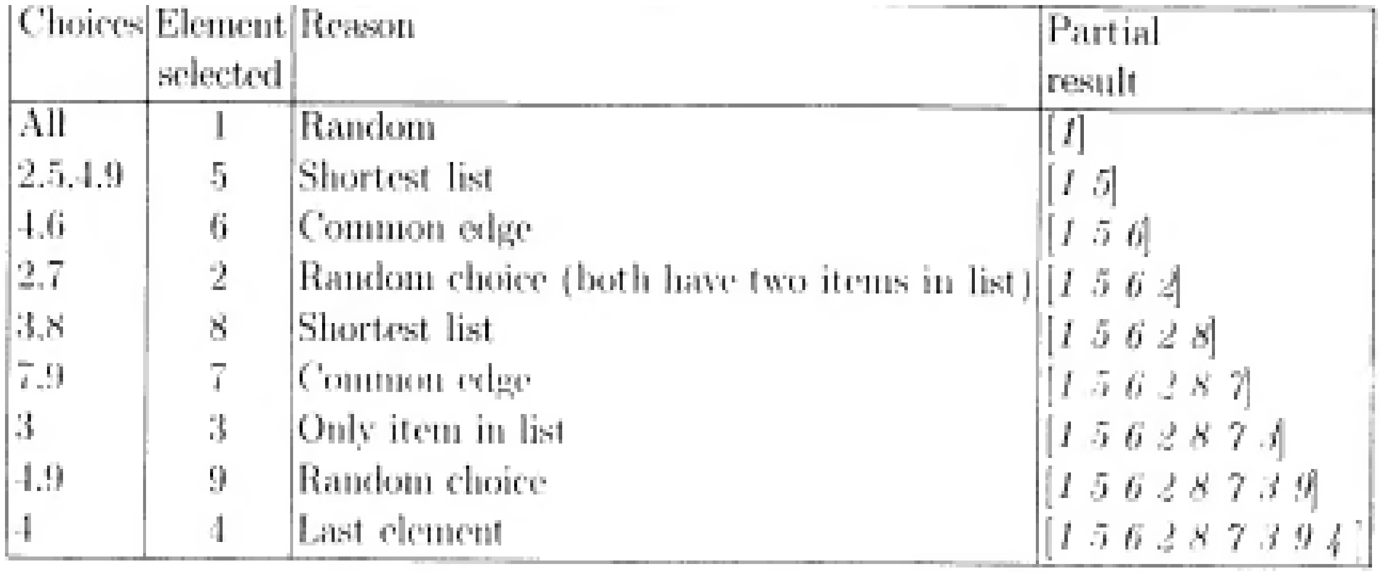
- If there is a common edge, choose it as the next element.
- If not, choose the entry with the shortest list.
- Ties are broken randomly

6-If an empty list is reached, the other end of the descendant is examined to continue. If no new item is chosen at random.

- For instance, consider the permutations [1 2 3 4 5 6 7 8 9] and [9 3 7 8 2 6 5 1 4].
- We have:



Element	Edges	Element	Edges
1	2,5,4,9	6	2,5+,7
2	1,3,6,8	7	3,6,8+
3	2,4,7,9	8	2,7+, 9
4	1,3,5,9	9	1,3,4,8
5	1,4,6+		



Choices	Element selected	Reason	Partial result
All	1	Random	[1]
2,5,4,9	5	Shortest list	[1 5]
4,6	6	Common edge	[1 5 6]
2,7	2	Random choice (both have two items in list)	[1 5 6 2]
3,8	8	Shortest list	[1 5 6 2 8]
7,9	7	Common edge	[1 5 6 2 8 7]
3	3	Only item in list	[1 5 6 2 8 7 3]
4,9	9	Random choice	[1 5 6 2 8 7 3 9]
4	4	Last element	[1 5 6 2 8 7 3 9 4]

IMPORTANT: This method only produces one descendant.

Order-based crossover

o This method is designed for those problems in which the important thing is the order and not the adjacency.

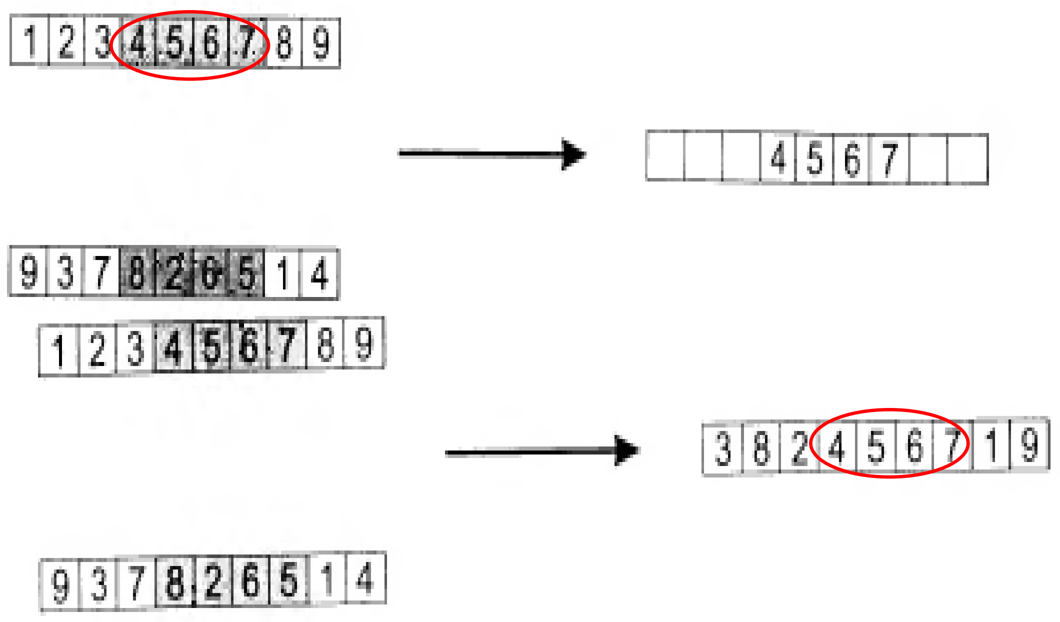
o Although it starts in a similar way to PMX, then it continues in a different way since it tries to convey information about the relative order of the elements in the second parent.

The procedure is the next.

1- Choose two crossing points at random and copy the segment between them from the first parent to the first offspring.

2-From the second crossing point, copy in order the values of the second parent that have not yet appeared.

3-Do the same with the roles of the reversed parents for the second offspring.



Cycle crossing

o This method aims to preserve information about the absolute position of the elements within each parent chromosome.

o The idea is to divide the chromosome into cycles, that is, subsets of elements such that each of its elements always appears paired with another element of the same cycle when the parents align.

The procedure consists of two parts:

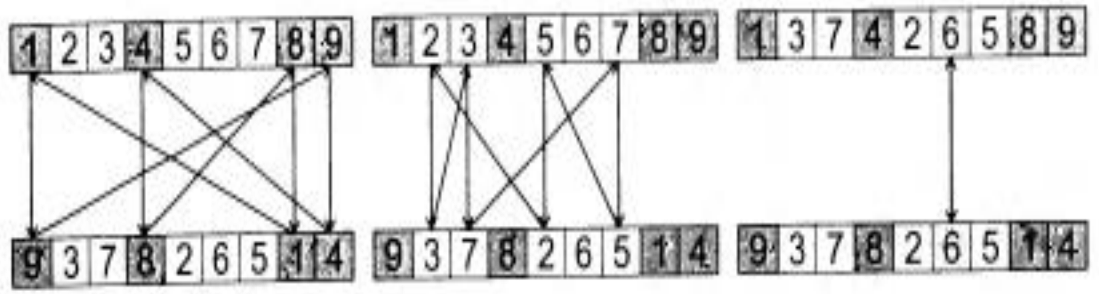
1-Build the cycles:

i) We start with the first unused position of P1.

ii) We look for the allele in the same position as P2.

iii) Let's go to the position with the same allele in P1.

iv) We add this allele to the cycle. We repeat steps 2-4 until we return to the allele considered in i).



Finally, it only remains to exchange the cycles with each other to generate the two descendants.

MULTIPARENTAL CROSSINGS

o So far we have seen operators that start from a single parent (mutations) and from two parents (crosses).

o It seems natural to consider whether we can continue to generalize these operators by taking 3, 4, ... parents.

o These more general operators are called multi-parental recombination operators

o These operators can be classified taking into account the way in which the information present in the parents is combined. Thus, we have:

- Operators based on frequency, which generalize the UX.
- Operators based on the segmentation and recombination of the parents, which generalize the nX
- Operators that are based on numerical operations, which generalize the arithmetic crossing operators.

o In any case, for now it has not been possible to prove that the use of these operators improves AG performance in general, although it does in some specific problems.

REFERENCES

The classic reference on genetic algorithms is:

D.E. Goldberg, *Genetic algorithms in search, optimization and machine learning,* Adison Wesley 1989.

Another good reference is:

M. Mitchell, *An introduction to genetic algorithms,* MIT press, 1996.