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.

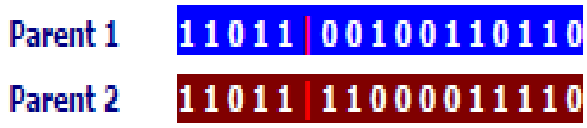
The Crossover operators are of many types. one simple way is, One-Point crossover. The others are: Two Point, Uniform, Arithmetic, and Heuristic crossovers.

The operators are selected based on the way chromosomes are encoded.

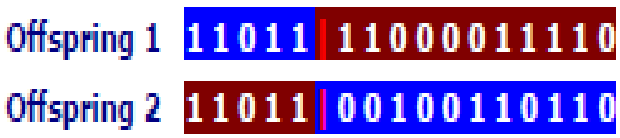
One-Point Crossover:

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.



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



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 | 0 0 1 0 0 1 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 0 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	1_1 1_2 0_2 1_1 1_1 1_2 1_2 0_2 0_1 0_1 0_2 1_1 1_2 1_1 1_1 0_2
Offspring 2	1_2 1_1 0_1 1_2 1_2 0_1 0_1 1_1 0_2 0_2 1_1 1_2 0_1 1_2 1_2 0_1

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

Arithmetic Crossover:

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

Offspring1 = a * Parent1 + (1- a) * Parent2

Offspring2 = (1 - a) * Parent1 + a * 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)**

Assis. Prof. Dr. Ziyad Tariq Al-Ta'i

Heuristic Crossover:

Heuristic crossoer operator uses the fitness values of the two parent chromosomes to determine the direction of the search.

The offspring are created according to the equations:

Offspring1 = BestParent + r * (BestParent - WorstParent)

Offspring2 = BestParent

where r is a random number between 0 and 1.

It is possible that offspring1 will not be feasible. It can happen if r is chosen such that one or more of its genes fall outside of the allowable upper or lower bounds. For this reason, heuristic crossover has a user defined parameter n for the number of

times to try and find an r that results in a feasible chromosome. If a feasible chromosome is not produced after n tries, the worst parent is returned as offspring1.

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.
- the Others are Boundary, Non-Uniform, Uniform, and Gaussian.
The operators are selected based on the way chromosomes are encoded .

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

Permutation Encoding

Crossover

Single point crossover - one crossover point is selected, till this point 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
Note: there are more ways how to produce the rest after crossover point

(1 2 3 4 5 6 7 8 9) + (4 5 3 6 8 9 7 2 1) = (1 2 3 4 5 6 8 9 7)

Mutation

Order changing - two numbers are selected and exchanged

(1 2 3 4 5 6 8 9 7) => (1 8 3 4 5 6 2 9 7)

Value Encoding

Crossover

All crossovers from binary encoding can be used

Mutation

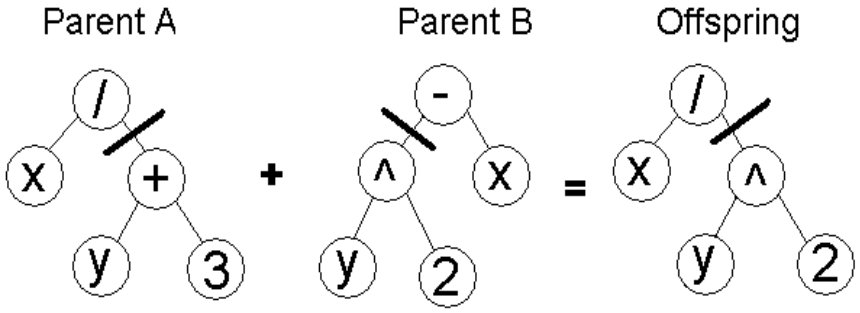
Adding a small number (for real value encoding) - to selected values is added (or subtracted) a small number

(1.29 5.68 2.86 4.11 5.55) => (1.29 5.68 2.73 4.22 5.55)

Tree Encoding

Crossover

Tree crossover - in both parent one crossover point is selected, parents are divided in that point and exchange part below crossover point to produce new offspring



Mutation

Changing operator, number - selected nodes are changed

Crossover and Mutation Probability

There are two basic parameters of GA - crossover probability and mutation probability.

Crossover probability says how often will be crossover performed. If there is no crossover, offspring is exact

copy of parents. If there is a crossover, offspring is made from parts of parents' chromosome. If crossover probability is 100%, then all offspring is made by crossover. If it is 0%, whole new generation is made from exact copies of chromosomes from old population (but this does not mean that the new generation is the same!). Crossover is made in hope that new chromosomes will have good parts of old chromosomes and maybe the new chromosomes will be better. However it is good to leave some part of population survive to next generation.

Mutation probability says how often will be parts of chromosome mutated. If there is no mutation, offspring is taken after crossover (or copy) without any change. If mutation is performed, part of chromosome is changed. If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed. Mutation is made to prevent falling GA into local extreme, but it should not occur very often, because then GA will in fact change to random search.

Other Parameters

There are also some other parameters of GA. One also important parameter is population size.

Population size says how many chromosomes are in population (in one generation). If there are too few chromosomes, GA have a few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, GA slows down. Research shows that after some limit (which depends mainly on encoding and the problem) it is not useful to increase population size, because it does not make solving the problem faster.

This chapter should give you some basic recommendations if you have decided to implement your genetic algorithm. These recommendations are very general. Probably you will want to experiment with your own GA for specific problem, because today there is no general theory which would describe parameters of GA for any problem.

Recommendations:

Recommendations are often results of some empiric studies of GAs, which were often performed only on binary encoding.

- **Crossover rate**
  Crossover rate generally should be high, about 80%-95%. (However some results show that for some problems crossover rate about 60% is the best.)
- **Mutation rate**
  On the other side, mutation rate should be very low. Best rates reported are about 0.5%-1%.
- **Population size**
  It may be surprising, that very big population size usually does not improve performance of GA (in meaning of speed of finding solution). Good population size is about 20-30, however sometimes sizes 50-100 are reported as best. Some research also shows, that best population size depends on encoding, on size of encoded string. It means, if you have chromosome with 32 bits, the population should be say 32, but surely two times more than the best population size for chromosome with 16 bits.
- **Selection**
  Basic roulette wheel selection can be used, but sometimes rank selection can be better. Check chapter about selection for advantages and disadvantages. There are also some more sophisticated method, which

changes parameters of selection during run of GA. Basically they behaves like simulated annealing. But surely elitism should be used (if you do not use other method for saving the best found solution). You can also try steady state selection.

- **Encoding**
  Encoding depends on the problem and also on the size of instance of the problem. Check chapter about encoding for some suggestions or look to other resources.
- **Crossover and mutation type**
  Operators depend on encoding and on the problem. Check chapter about operators for some suggestions. You can also check other sites.

- **Example 1 :**

  Maximize the function $f(x) = x^2$ over the range of integers from 0 . . . 31.

  Note : This function could be solved by a variety of traditional methods such as a hill-climbing algorithm which uses the derivative.

  One way is to :

  - Start from any integer x in the domain of f
  - Evaluate at this point x the derivative f'
  - Observing that the derivative is +ve, pick a new x which is at a small distance in the +ve direction from current x
  - Repeat until x = 31

  See, how a genetic algorithm would approach this problem ?

1. Devise a means to represent a solution to the problem :

   Assume, we represent **x** with five-digit unsigned binary integers.

2. Devise a heuristic for evaluating the fitness of any particular solution :

   The function **f(x)** is simple, so it is easy to use the **f(x)** value itself to rate the fitness of a solution; else we might have considered a more simpler heuristic that would more or less serve the same purpose.

3. Coding - Binary and the String length :

   GAs often process binary representations of solutions. This works well, because crossover and mutation can be clearly defined for binary solutions. A Binary string of length **5** can represents 32 numbers (0 to 31).

4. Randomly generate a set of solutions :

   Here, considered a population of four solutions. However, larger populations are used in real applications to explore a larger part of the search. Assume, four randomly generated solutions as : **01101, 11000, 01000, 10011.** These are chromosomes or genotypes.

5. Evaluate the fitness of each member of the population :

   The calculated fitness values for each individual are -

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

   01101 → 13; 11000 → 24; 01000 → 8; 10011 → 19;

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

   13 → 169; 24 → 576; 8 → 64; 19 → 361.

   (c) Expected count = **N * Prob i** , where **N** is the number of individuals in the population called population size, here **N = 4.**

Thus the evaluation of the initial population summarized in table below .

String No i	Initial Population (chromosome)	X value (Pheno types)	Fitness $f(x) = x^2$	Prob i (fraction of total)	Expected count N * Prob i
1	0 1 1 0 1	13	169	0.14	0.58
2	1 1 0 0 0	24	576	0.49	1.97
3	0 1 0 0 0	8	64	0.06	0.22
4	1 0 0 1 1	19	361	0.31	1.23
Total (sum)			1170	1.00	4.00
Average			293	0.25	1.00
Max			576	0.49	1.97

Thus, the string no 2 has maximum chance of selection.

6. Produce a new generation of solutions by picking from the existing pool of solutions with a preference for solutions which are better suited than others:

We divide the range into four bins, sized according to the relative fitness of the solutions which they represent.

Strings	*Prob i*	*Associated Bin*
0 1 1 0 1	0.14	0.0 ... 0.14
1 1 0 0 0	0.49	0.14 ... 0.63
0 1 0 0 0	0.06	0.63 ... 0.69
1 0 0 1 1	0.31	0.69 ... 1.00

By generating 4 uniform (0, 1) random values and seeing which bin they fall into we pick the four strings that will form the basis for the next generation.

Random No	*Falls into bin*	*Chosen string*
0.08	0.0 ... 0.14	0 1 1 0 1
0.24	0.14 ... 0.63	1 1 0 0 0
0.52	0.14 ... 0.63	1 1 0 0 0
0.87	0.69 ... 1.00	1 0 0 1 1

7. Randomly pair the members of the new generation

Random number generator decides for us to mate the first two strings together and the second two strings together.

8. Within each pair swap parts of the members solutions to create offspring which are a mixture of the parents :

For the first pair of strings: 0 1 1 0 1 , 1 1 0 0 0

- We randomly select the crossover point to be after the fourth digit. Crossing these two strings at that point yields:

  0 1 1 0 1 ⇒ 0 1 1 0 |1 ⇒ 0 1 1 0 0

  1 1 0 0 0 ⇒ 1 1 0 0 |0 ⇒ 1 1 0 0 1

For the second pair of strings: 1 1 0 0 0 , 1 0 0 1 1

- We randomly select the crossover point to be after the second digit. Crossing these two strings at that point yields:

  1 1 0 0 0 ⇒ 1 1 |0 0 0 ⇒ 1 1 0 1 1

  1 0 0 1 1 ⇒ 1 0 |0 1 1 ⇒ 1 0 0 0 0

9. Randomly mutate a very small fraction of genes in the population :

With a typical mutation probability of per bit it happens that none of the bits in our population are mutated.

10. Go back and re-evaluate fitness of the population (new generation) :

This would be the first step in generating a new generation of solutions. However it is also useful in showing the way that a single iteration of the genetic algorithm has improved this sample.

String No	*Initial Population (chromosome)*	*X value (Pheno types)*	*Fitness $f(x) = x^2$*	*Prob i (fraction of total)*	*Expected count*
1	0 1 1 0 0	12	144	0.082	0.328
2	1 1 0 0 1	25	625	0.356	1.424
3	1 1 0 1 1	27	729	0.415	1.660
4	1 0 0 0 0	16	256	0.145	0.580
Total (sum)			1754	1.000	4.000
Average			439	0.250	1.000
Max			729	0.415	1.660

Observe that :

1. Initial populations : At start step 5 were

   0 1 1 0 1 , 1 1 0 0 0 , 0 1 0 0 0 , 1 0 0 1 1

   After one cycle, new populations, at step 10 to act as initial population

   0 1 1 0 0 , 1 1 0 0 1 , 1 1 0 1 1 , 1 0 0 0 0

2. The total fitness has gone from 1170 to 1754 in a single generation.
3. The algorithm has already come up with the string 11011 (i.e x = 27) as a possible solution.