•*Chap. 8 Evolutionary Algorithms*

~ Evolutionary algorithms (EAs) mimic natural evolutionary principles to constitute search and optimization procedures.

~ EAs encompass several major branches, i.e., evolutionary strategies (ES), evolutionary programming (EP), genetic algorithms (GAs), and genetic programming (GP), due largely to historical reasons.

~ All EAs have two prominent features,

(1) They are all population-based.

(2) There are communications and information exchange among individuals in a populations, which are the result of selection and/or combination.

~ A general framework of EAs can be summarized as follows, where the search operators are also called genetic operators for GAs. They are used to generate offspring (new individuals) from parents (existing individuals).

1. Set I = 0;
2. Generate the initial population P(i) at random;
3. REPEAT
   (a) Evaluate the fitness (performance) of each individual in P(i);
   (b) Select parents from P(i) based on their fitness;
   (c) Apply search operators to the parents and produce generation P(i+1)
4. UNTIL the population converges or the maximum time is reached.

~Different representations of individuals and different schemes for implementing selection and search operators define different algorithms.

• Genetic algorithms

~Genetic Algorithms (GAs) are general-purpose search algorithms that use principles inspired by natural population genetics to evolve solutions to problems.

~They were first proposed by Holland (1975).

~GAs have been employed primarily in two major areas: optimization and machine learning.

• *Basics of Genetic Algorithms*

~The evolution of living beings is a process that operates on chromosomes- organic devices for encoding the structure of living beings.

~Natural selection is the link between chromosomes and the performance of their decoded structures.

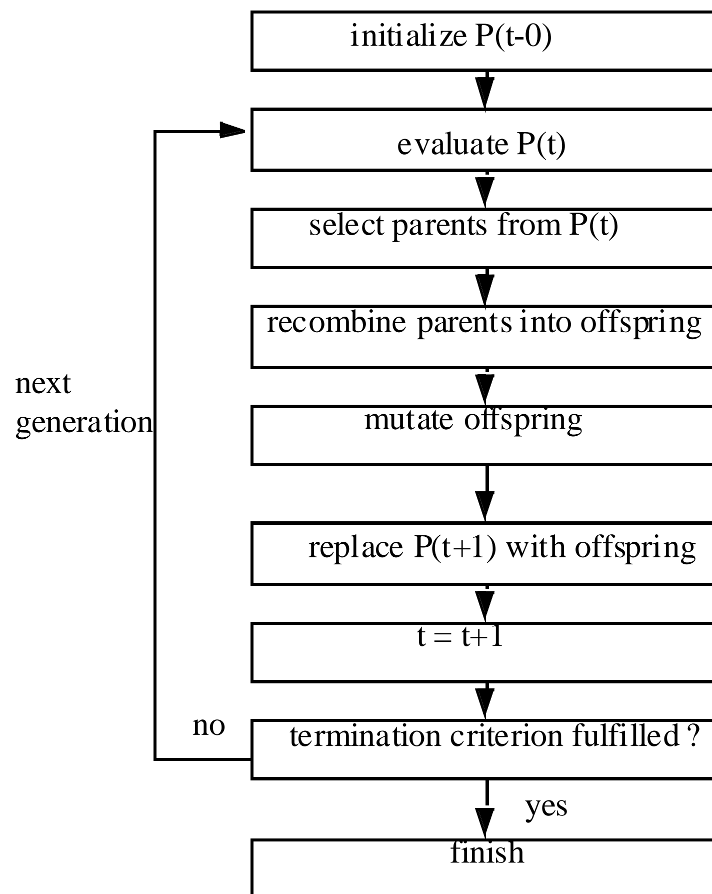
~Nature evolution process contains:

(1) *reproduction*: chromosomes that encode successful structures to reproduce more often than those do not.

(2) *recombination (crossover)*: create quite different chromosomes in children by combining material from the chromosomes of their two parents.

(3) *mutation*: cause the chromosomes of children to be different from those from those of their biological parents.

• Principal structure of GAs.



~ Roughly speaking, through a proper encoding mechanism, GAs manipulate strings of binary digits (ls and 0s) called chromosomes, which represent multiple points in the search space.

~ Like nature, GAs solve the problem of finding good chromosomes by manipulating the material in the chromosomes blindly without any knowledge about the type of problem they are solving.

~ The only information they are given is an evaluation of each chromosomes they produce.

• ***Terminology:***

~ chromosome (solution)-string of encoded parameters.

~ gene- variable.

~ alleles- the possible values of a variable.

~ locus- the position of a variable in a string.

~ genotype- the coded string which is processed by the algorithm.

~ phenotype- the decoded set of parameters.

~ Basic steps of genetic algorithm

step1: Establish a base population of chromosomes.

step2: Determine the fitness value of each chromosome.

step3: Duplicate the chromosomes according to their fitness values and create new chromosomes by mating current chromosomes. (e.g. mutation, recombination)

step4: Delete undesirable members of the population.

step5: Insert the new chromosomes into the population to form a new population. Go to Step 2.

continue until the predetermined condition is achieved.

- ~ The encoding mechanisms and the evaluation function form the links between the GA and the specific problem to be solved.
- ~ The technique for encoding solutions may vary from problem to problem and from GA to GA.
- ~ An evaluation function takes a chromosome as input and returns a number or a list of numbers that are a measure of the chromosome's performance on the problem to be solved. Evaluation functions play the same role in GAs as the environment plays in natural evolution.
- ~ The GA is a general-purpose stochastic optimization method for solving search problems.

- Advantages of A GA:
  - Optimizes with continuous or discrete variables
  - Does not require derivative variables
  - Simultaneously searches from a wide sampling of the cost surface
  - Deals with a large number of variables
  - Is well suited for parallel computers
  - Optimizes variables with extremely complex cost surfaces (they can jump out of a local minimum)
  - Provides a list optimum variables, not just a single solution
  - May encode the variables so that the optimization is done with the encoded variables
  - Works with numerically generated data, experimental data, or analytical functions.

~ Remarks:

1. Of course, the GA is not the best way to solve every problem. For instance, the traditional methods have been studied to quickly find the solution of a well-behaved convex analytical function of only a few variables.
2. The large population of solutions that gives the GA its power is also its bane when its comes to speed on a serial computer.

~ A GA in its simplest form uses three operations:
Reproduction, Crossover, and Mutation.

• ***Reproduction:***

~ Reproduction is a process in which individual strings are copied according to their fitness value.

~ A fitness f(i) is assigned to each individual in the population, where high numbers denote good fit.

~ The fitness function can be any nonlinear, positive, discontinuous function because the algorithm only needs a fitness assigned to each string.

~ The reproduction (parent selection) process:

① Roulette-wheel parent selection:

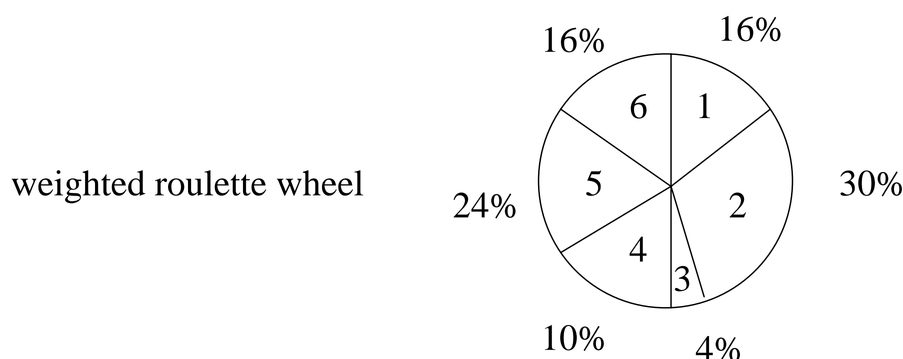
~ conducted by spinning a simulated biased roulette wheel whose slots have different sizes proportional to the fitness values of the individuals.

~ steps:

1. Sum the fitness of all the population members and call this result the total fitness.
2. Generate n, a random number between 1 and total fitness.
3. Return the first population member whose fitness, added to the fitness of the preceding population members (running total), is greater than or equal to n.

Ex: Consider a population of six chromosomes (strings) with a set of fitness values totaling 50.

No	string (Chromosome)	Fitness	% of Total	Running Total
1	01110	8	16	8
2	11000	15	30	23
3	00100	2	4	25
4	10010	5	10	30
5	01100	12	24	42
6	00011	8	16	50



~ generate numbers randomly from the interval 1 and 50.

~ the roulette-wheel parent selection technique chooses the first chromosome for which the running total of fitness is greater than or equal to the random number.

e.g.:

Random number	26	2	49	15	40	36	9
Chromosome chosen	4	1	6	2	5	5	2

② Tournament selection:

In tournament selection, two or more members of the population are selected at random and their fitness compared. The member with the highest fitness is selected.

~ once a chromosome has been selected for reproduction, an exact replica of it is made. This chromosome is then entered into a mating pool for further genetic operator action.

• ***Crossover:***

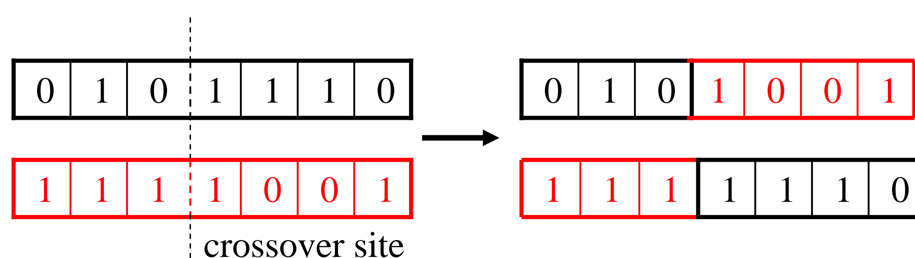
~ Reproduction directs the search toward the best existing individuals but does not create any new individuals.

~ In nature, an offspring has two parents and inherits genes from both.

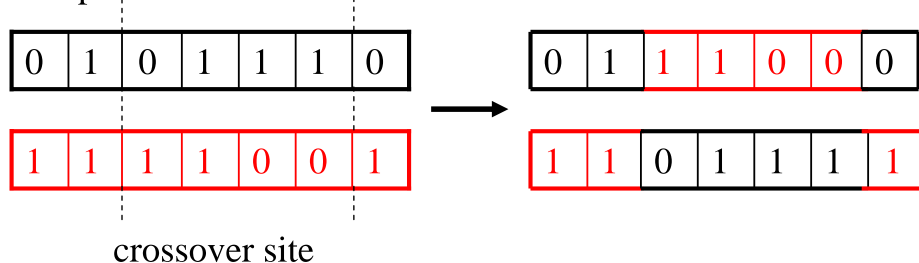
~ The main operator working on the parents is crossover, which happens for a selected pair with a crossover probability Pc.

~ At first, two chromosomes from the reproduced population are mated at random, and a crossover site (a bit position) is randomly selected. Then the chromosomes are crossed and separated at the site. This process produces two new chromosomes.

(1)one-point crossover:



(2) two-point crossover:



• ***Mutation:***

~ Although reproduction and crossover produce many new strings, they do not introduce any new information into the population at the gene level.

~ As a source of new genes, mutation is introduced and is applied with a low probability Pm.

~ Mutation should be used sparingly because it is a random search operator; otherwise, with high mutation rates, the algorithm will become little more than a random search.

• These three operators are applied repeatedly until the offspring take over the entire population. The next generation is thus made up of offspring of three types: mutated after crossover, crossed over but not mutated, and neither crossed over nor mutated, but just selected.

• In a simple GA, we need to specify the following parameters:

n : population size.
Pc: crossover probability.
Pm : mutation probability.

Ex: maximize $f(x)= x^2$, $x\in[0,31]$. $x\in Z$

① code the variable x as a binary unsigned integer of length 5.
e.g. "11000" represent integer 24.

② the fitness function is simply defined as the function f(x).

③ An initial population of size 4 is randomly selected.

④ Pc= 1.0, Pm= 0.001

(a) reproduction process:

the mating pool of the next generation is chosen by spinning the weighted roulette wheel four times.

No.	Initial Population	x	Fitness $f(x)= x^2$	P_{select_i} $f_i/\sum f$	No. of copies from Roulette Wheel
1	01001	9	81	0.08	1
2	11000	24	576	0.55	2
3	00100	4	16	0.02	0
4	10011	19	361	0.35	1
		sum	1034		
		Average	259		

(b) Crossover process:

Mating Pool after Reproduction	Mate	Crossover site	New Population	x	x^2
01001	2	4	01000	8	64
11000	1	4	11001	25	625
11000	4	2	11011	27	729
10011	2	2	10000	16	256
				sum	1674
				Average	419

Ex: maximize

$$f(x,y) = 0.5 - \frac{\sin^2\sqrt{x^2+y^2} - 0.5}{[1.0 + 0.001(x^2+y^2)]^2}$$

$x, y \in [-100, 100]$

~ *f(x,y)* is positive and to be maximized, it is used as the fitness function directly.

~ chromosome: a string of 44 bits

the initial 22 bits: an integer x in base-2 notation.

the last 22 bits: an integer y in base-2 notation.

decode: x and y are multiplied by $200/(2^{22}-1)$ to map the values of x and y from the range $[0, 2^{22}-1]$ to the range [0, 200]. Finally, 100 is subtracted from x and y.

e.g.,

0000101000001100000000011000101010001110111011

represent x=165,377 and y= 2,270,139

decode: x= -92.11 y= 8.25 $\Rightarrow f$= 0.495

~ operators: roulette- wheel parent selection, simple crossover with random mating, and simple mutation.

parameters: n (population size) = 100

P_c (crossover probability) = 0.65

P_m (mutation probability) = 0.008

~ At the 14th generation, the top five chromosomes are very similar and the fitness value are:

0.99304112 0.99261288 0.99254826 0.99254438 0.99229856

GAs: How do they work

- Objective: maximize $f(x_1, \cdots, x_k): \mathbb{R} \to \mathbb{R}$

  $x_i \in [a_i, b_i] \subseteq \Re \quad f(x_1, \cdots, x_k) > 0 \;\; \forall x_i \in [a_i, b_i]$

  required precsion: six decimal places for x_i
- representation of each chromosome

  ~cut each domain $[a_i, b_i]$ into $(b_i - a_i) \cdot 10^6$ equal ranges.

  ~let m_i be the smallest integer s.t.

  each variable x_i can be coded as a binary string of length m_i

  $(b_i - a_i) \cdot 10^6 \le 2^{m_i} - 1$

  $x_i = a_i + \text{decimal}(1001\cdots001) \cdot \frac{b_i - a_i}{2^{m_i} - 1}$

~ each chromosome is represented by a binary string of length

$$m = \sum_{i=1}^{k} m_i$$

the first m_1 bits $\to x_1 \in [a_1, b_1]$

next group of m_2 bits $\to x_2 \in [a_2, b_2]$

$\vdots$

last group of m_k bits $\to x_k \in [a_k, b_k]$

- Initialization:

  set some *pop-size* number of chromosomes randomly in a bitwise fashion.
- Calculate the fitness value $eval(v_i)$ for each chromosome v_i $(i = 1, \cdots, pop\text{-}size)$
- Find the total fitness of the population

$$F = \sum_{i=1}^{pop\text{-}size} eval(v_i)$$

- Calculate the probability of a selection p_i for each chromosome v_i

$$p_i = eval(v_i) / F$$

- Calculate a cumulative probability q_i for each chromosome v_i

$$q_i = \sum_{i=1}^{i} p_i$$

- Selection: spin the roulette wheel $pop\text{-}size$ times; each time we select a single chromosome for a new population in the following way.
  ~ Generate a random (float) number r from the range [0,1]
  ~ If $r < q_1$ then select v_1; otherwise select the i-th chromosome v_i $(2 \le i \le pop\text{-}size)$ s.t. $q_{i\text{-}1} < r < q_i$

Remark:The best chromosomes get more copies and the worst die off

- Crossover: expected number of chromosomes undergo the crossover operation is

$$p_c \cdot pop\text{-}size$$

  ~ Generate a random (float) number r from the range [0, 1]
  ~ If $r < p_c$, select given chromosome for crossover
  ~ mate selected chromosomes randomly: generate a rondom number pos from the range [1, m-1] (one point crossover).

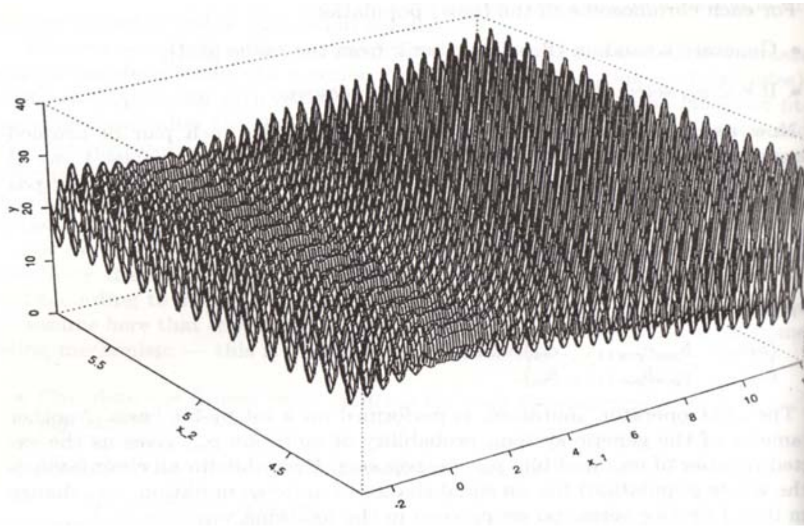
- Mutation: performed on a bit-by-bit basis
  ~ If mutation is performed on all chromosomes then the expected number of mutated bits is $p_m \cdot m \cdot pop\text{-}size.$
  ~ Each bit has an equal chance to undergo mutation, i.e., change from 0 to 1 or vice versa.
  ~ For each bit:
  Generate a rondom (float) number r from the range [0,1] ; if $r < p_m$, mutate the bit

Example:

maximize $f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2)$

$-3.0 \le x_1 \le 12.1,\ 4.1 \le x_2 \le 5.8$

reguired precision: 4 decimal places.



Assume *pop-size*=20，p_c=0.25, p_m=0.01

~ x_1 : domain length 15.1

$\Rightarrow$ at least $15.1 \cdot 10000$ equal size ranges

$2^{17} < 151000 \le 2^{18} \Rightarrow 18$ bits are required

x_2 : domain length 1.7

$\Rightarrow$ at least $1.7 \cdot 10000$ equal size ranges

$2^{14} < 17000 \le 2^{15} \Rightarrow 15$ bits are required

~ The total length of a chromosome is $m = 18 + 15 = 33$.

e.g. , (010001001011010000111110010100010)

$$x_1 = -3 + \text{decimal}(010001001011010000) \cdot \frac{12.1-(-3.0)}{2^{18}-1}$$

$$= -3 + 70352 \cdot \frac{15.1}{262143} = 1.052426$$

$$x_2 = 4.1 + \text{decimal}(111110010100010) \cdot \frac{5.8-4.1}{2^{15}-1}$$

$$= 4.1 + 31906 \cdot \frac{1.7}{32767} = 5.755330$$

fitness value: $f(x_1, x_2) = 20.252640$

~ Initialization : all 33 bits are initialized randomly

v_1 = (100110100000001111111010011011111)
v_2 = (111000100100110111001010100011010)
v_3 = (000010000011001000001010111011101)
v_4 = (100011000101101001111000001110010)
v_5 = (000111011001010011010111111000101)
v_6 = (000101000010010101001010111111011)
v_7 = (001000100000110101111011011111011)
v_8 = (100001100001110100010110101100111)
v_9 = (010000000101100010110000001111100)
v_{10} = (000001111000110000011010000111011)
v_{11} = (011001111110110101100001101111000)
v_{12} = (110100010111101101000101010000000)
v_{13} = (111011111010001000110000001000110),
v_{14} = (010010011000001010100111100101001)
v_{15} = (111011101101110000100011111011110)
v_{16} = (110011110000011111100001101001011)
v_{17} = (011010111111001111010001101111101)
v_{18} = (011101000000001110100111110101101)
v_{19} = (000101010011111111110000110001100)
v_{20} = (101110010110011110011000101111110)

Fitness values：

$$
\begin{aligned}
eval(\boldsymbol{v}_1) &= f(6.084492, 5.652242) = 26.019600\\
eval(\boldsymbol{v}_2) &= f(10.348434, 4.380264) = 7.580015\\
eval(\boldsymbol{v}_3) &= f(-2.516603, 4.390381) = 19.526329\\
eval(\boldsymbol{v}_4) &= f(5.278638, 5.593460) = 17.406725\\
eval(\boldsymbol{v}_5) &= f(-1.255173, 4.734458) = 25.341160\\
eval(\boldsymbol{v}_6) &= f(-1.811725, 4.391937) = 18.100417\\
eval(\boldsymbol{v}_7) &= f(-0.991471, 5.680258) = 16.020812\\
eval(\boldsymbol{v}_8) &= f(4.910618, 4.703018) = 17.959701\\
eval(\boldsymbol{v}_9) &= f(0.795406, 5.381472) = 16.127799\\
eval(\boldsymbol{v}_{10}) &= f(-2.554851, 4.793707) = 21.278435\\
eval(\boldsymbol{v}_{11}) &= f(3.130078, 4.996097) = 23.410669\\
eval(\boldsymbol{v}_{12}) &= f(9.356179, 4.239457) = 15.011619\\
eval(\boldsymbol{v}_{13}) &= f(11.134646, 5.378671) = 27.316702\\
eval(\boldsymbol{v}_{14}) &= f(1.335944, 5.151378) = 19.876294\\
eval(\boldsymbol{v}_{15}) &= f(11.089025, 5.054515) = 30.060205\\
eval(\boldsymbol{v}_{16}) &= f(9.211598, 4.993762) = 23.867227\\
eval(\boldsymbol{v}_{17}) &= f(3.367514, 4.571343) = 13.696165\\
eval(\boldsymbol{v}_{18}) &= f(3.843020, 5.158226) = 15.414128\\
eval(\boldsymbol{v}_{19}) &= f(-1.746635, 5.395584) = 20.095903\\
eval(\boldsymbol{v}_{20}) &= f(7.935998, 4.757338) = 13.666916
\end{aligned}
$$

$$\text{total fitness}: F = \sum_{i=1}^{20} eval(v_i) = 387.776822.$$

~ reproduction

According to roulette wheel selection, the probability of a selection p_i for each chromosome v_i is:

$$
\begin{aligned}
p_1 &= eval(\boldsymbol{v}_1)/F = 0.067099 & p_2 &= eval(\boldsymbol{v}_2)/F = 0.019547\\
p_3 &= eval(\boldsymbol{v}_3)/F = 0.050355 & p_4 &= eval(\boldsymbol{v}_4)/F = 0.044889\\
p_5 &= eval(\boldsymbol{v}_5)/F = 0.065350 & p_6 &= eval(\boldsymbol{v}_6)/F = 0.046677\\
p_7 &= eval(\boldsymbol{v}_7)/F = 0.041315 & p_8 &= eval(\boldsymbol{v}_8)/F = 0.046315\\
p_9 &= eval(\boldsymbol{v}_9)/F = 0.041590 & p_{10} &= eval(\boldsymbol{v}_{10})/F = 0.054873\\
p_{11} &= eval(\boldsymbol{v}_{11})/F = 0.060372 & p_{12} &= eval(\boldsymbol{v}_{12})/F = 0.038712\\
p_{13} &= eval(\boldsymbol{v}_{13})/F = 0.070444 & p_{14} &= eval(\boldsymbol{v}_{14})/F = 0.051257\\
p_{15} &= eval(\boldsymbol{v}_{15})/F = 0.077519 & p_{16} &= eval(\boldsymbol{v}_{16})/F = 0.061549\\
p_{17} &= eval(\boldsymbol{v}_{17})/F = 0.035320 & p_{18} &= eval(\boldsymbol{v}_{18})/F = 0.039750\\
p_{19} &= eval(\boldsymbol{v}_{19})/F = 0.051823 & p_{20} &= eval(\boldsymbol{v}_{20})/F = 0.035244
\end{aligned}
$$

The cumulative probilities q_i for each chromosome v_i are:

$q_1 = 0.067099$	$q_2 = 0.086647$	$q_3 = 0.137001$	$q_4 = 0.181890$
$q_5 = 0.247240$	$q_6 = 0.293917$	$q_7 = 0.335232$	$q_8 = 0.381546$
$q_9 = 0.423137$	$q_{10} = 0.478009$	$q_{11} = 0.538381$	$q_{12} = 0.577093$
$q_{13} = 0.647537$	$q_{14} = 0.698794$	$q_{15} = 0.776314$	$q_{16} = 0.837863$
$q_{17} = 0.873182$	$q_{18} = 0.912932$	$q_{19} = 0.964756$	$q_{20} = 1.000000$

Spin the roulette wheel 20 times.
Assume a (random) sequence of 20 numbers from the range [0, 1] is

0.513870	0.175741	0.308652	0.534534	0.947628
0.171736	0.702231	0.226431	0.494773	0.424720
0.703899	0.389647	0.277226	0.368071	0.983437
0.005398	0.765682	0.646473	0.767139	0.780237

$q_{10} < 0.513870 < q_{11} \rightarrow v_{11}$ is selected
$q_3 < 0.175741 < q_4 \rightarrow v_4$ is selected

temporary population (in mating pool)

v'_1 = (011001111110110101100001101111000) (v_{11})
v'_2 = (100011000101101001111000001110010) (v_4)
v'_3 = (001000100000110101111011011111011) (v_7)
v'_4 = (011001111110110101100001101111000) (v_{11})
v'_5 = (000101010011111111110000110001100) (v_{19})
v'_6 = (100011000101101001111000001110010) (v_4)
v'_7 = (111011101101110000100011111011110) (v_{15})
v'_8 = (000111011001010011010111111000101) (v_5)
v'_9 = (011001111110110101100001101111000) (v_{11})
v'_{10} = (000010000011001000001010111011101) (v_3)
v'_{11} = (111011101101110000100011111011110) (v_{15})
v'_{12} = (010000000101100010110000001111100) (v_9)
v'_{13} = (000101000010010101001010111111011) (v_6)
v'_{14} = (100001100001110100010110101100111) (v_8)
v'_{15} = (101110010110011110011000101111110) (v_{20})
v'_{16} = (100110100000001111111010011011111) (v_1)
v'_{17} = (000001111000110000011010000111011) (v_{10})
v'_{18} = (111011111010001000110000001000110) (v_{13})
v'_{19} = (111011101101110000100011111011110) (v_{15})
v'_{20} = (110011110000011111100001101001011) (v_{16})

~Crossover：
generate a random number r from [0,1], if r <0.25
→ select a chromosome for crossover.

A sequence of random numbers:

0.822951	0.151932	0.625477	0.314685	0.346901
0.917204	0.519760	0.401154	0.606758	0.785402
0.031523	0.869921	0.166525	0.674520	0.758400
0.581893	0.389248	0.200232	0.355635	0.826927

~v_2', v_{11}', v_{13}', and v_{18}' were selected for crossover.
~Mate selected chromosome randomly say (v_2' and v_{11}') , (v_{13}' and v_{18}') for each of the two pairs, generate a random integer number *pos* from the range [1,32]

first pair, *pos*=9

$$v'_2 = (100011000|101101001111000001110010)$$
$$v'_{11} = (111011101|101110000100011111011110)$$

Offspring

$$v''_2 = (100011000|101110000100011111011110)$$
$$v''_{11} = (111011101|101101001111000001110010).$$

Second pair, *pos*=20

$$v'_{13} = (00010100001001010100|1010111111011)$$
$$v'_{18} = (11101111101000100011|0000001000110)$$

Offspring:

$$v''_{13} = (00010100001001010100|0000001000110)$$
$$v''_{18} = (11101111101000100011|1010111111011).$$

Current version of the population

$v'_1 = (011001111110110101100001101111000)$
$v''_2 = (100011000101110000100011111011110)$
$v'_3 = (001000100000110101111011011111011)$
$v'_4 = (011001111110110101100001101111000)$
$v'_5 = (000101010011111111110000110001100)$
$v'_6 = (100011000101101001111000001110010)$
$v'_7 = (111011101101110000100011111011110)$
$v'_8 = (000111011001010011010111111000101)$
$v'_9 = (011001111110110101100001101111000)$
$v'_{10} = (000010000011001000001010111011101)$
$v''_{11} = (111011101101101001111000001110010)$
$v'_{12} = (010000000101100010110000001111100)$
$v''_{13} = (000101000010010101000000001000110)$
$v'_{14} = (100001100001110100010110101100111)$
$v'_{15} = (101110010110011110011000101111110)$
$v'_{16} = (100110100000001111111010011011111)$
$v'_{17} = (000001111000110000011010000111011)$
$v''_{18} = (111011111010001000111010111111011)$
$v'_{19} = (111011101101110000100011111011110)$
$v'_{20} = (110011110000011111100001101001011)$

~Mutation:
There are $m \times pop_size$=33 ×20=660 bits in the whole population.
We expect 660 ×p_m=660 ×0.01=6.6 mutations per generation operate a random number r from [0,1], if $r < 0.01$, mutate the bit.
660 random numbers are generated and 5 of these numbers were smaller than 0.01.

Bit position	Random number	Chromosome number	Bit number with in chromosome
112	0.000213	4	13
349	0.009945	11	19
418	0.008809	13	22
429	0.005425	13	33
602	0.002836	19	8

~ Final population

$$
\begin{aligned}
v_1 &= (011001111110110101100001101111000)\\
v_2 &= (100011000101110000100011111011110)\\
v_3 &= (001000100000110101111011011111011)\\
v_4 &= (011001111110010101100001101111000)\\
v_5 &= (000101010011111111110000110001100)\\
v_6 &= (100011000101101001111000001110010)\\
v_7 &= (111011101101110000100011111011110)\\
v_8 &= (000111011001010011010111111000101)\\
v_9 &= (011001111110110101100001101111000)\\
v_{10} &= (000010000011001000001010111011101)\\
v_{11} &= (111011101101101001011000001110010)\\
v_{12} &= (010000000101100010110000001111100)\\
v_{13} &= (000101000010010101000100001000111)\\
v_{14} &= (100001100001110100010110101100111)\\
v_{15} &= (101110010110011110011000101111110)\\
v_{16} &= (100110100000001111111010011011111)\\
v_{17} &= (000001111000110000011010000111011)\\
v_{18} &= (111011111010001000111010111111011)\\
v_{19} &= (111011100101110000100011111011110)\\
v_{20} &= (110011110000011111100001101001011)
\end{aligned}
$$

decode each chromosome and calculate the fitness values from $f(x_1,x_2)$

$$
\begin{aligned}
eval(v_1) &= f(3.130078, 4.996097) = 23.410669\\
eval(v_2) &= f(5.279042, 5.054515) = 18.201083\\
eval(v_3) &= f(-0.991471, 5.680258) = 16.020812\\
eval(v_4) &= f(3.128235, 4.996097) = 23.412613\\
eval(v_5) &= f(-1.746635, 5.395584) = 20.095903\\
eval(v_6) &= f(5.278638, 5.593460) = 17.406725\\
eval(v_7) &= f(11.089025, 5.054515) = 30.060205\\
eval(v_8) &= f(-1.255173, 4.734458) = 25.341160\\
eval(v_9) &= f(3.130078, 4.996097) = 23.410669\\
eval(v_{10}) &= f(-2.516603, 4.390381) = 19.526329\\
eval(v_{11}) &= f(11.088621, 4.743434) = 33.351874\\
eval(v_{12}) &= f(0.795406, 5.381472) = 16.127799\\
eval(v_{13}) &= f(-1.811725, 4.209937) = 22.692462\\
eval(v_{14}) &= f(4.910618, 4.703018) = 17.959701\\
eval(v_{15}) &= f(7.935998, 4.757338) = 13.666916\\
eval(v_{16}) &= f(6.084492, 5.652242) = 26.019600\\
eval(v_{17}) &= f(-2.554851, 4.793707) = 21.278435\\
eval(v_{18}) &= f(11.134646, 5.666976) = 27.591064\\
eval(v_{19}) &= f(11.059532, 5.054515) = 27.608441\\
eval(v_{20}) &= f(9.211598, 4.993762) = 23.867227
\end{aligned}
$$

total fitness=447.049688 (higher than previous population
best $eval(v_{11})$=33.351874 > best $eval(v_{15})$=30.060205 in previous population.
~ After 1000 generations, the population is

v_1 = (111011110110011011100101010111011)
v_2 = (111001100110000100010101010111000)
v_3 = (111011110111011011100101010111011)
v_4 = (111001100010000110000101010111001)
v_5 = (111011110111011011100101010111011)
v_6 = (111001100110000100000100010100001)
v_7 = (110101100010010010001100010110000)
v_8 = (111101100010001010001101010010001)
v_9 = (111001100010010010001100010110001)
v_{10} = (111011110111011011100101010111011)
v_{11} = (110101100000010010001100010110000)
v_{12} = (110101100010010010001100010110001)
v_{13} = (111011110111011011100101010111011)
v_{14} = (111001100110000100000101010111011)
v_{15} = (111001101010111001010100110110001)
v_{16} = (111001100110000101000100010100001)
v_{17} = (111001100110000100000101010111011)
v_{18} = (111001100110000100000101010111001)
v_{19} = (111101100010001010001110000010001)
v_{20} = (111001100110000100000101010111001)

The fitness values:

$$
\begin{aligned}
eval(\boldsymbol{v}_1) &= f(11.120940, 5.092514) = 30.298543\\
eval(\boldsymbol{v}_2) &= f(10.588756, 4.667358) = 26.869724\\
eval(\boldsymbol{v}_3) &= f(11.124627, 5.092514) = 30.316575\\
eval(\boldsymbol{v}_4) &= f(10.574125, 4.242410) = 31.933120\\
eval(\boldsymbol{v}_5) &= f(11.124627, 5.092514) = 30.316575\\
eval(\boldsymbol{v}_6) &= f(10.588756, 4.214603) = 34.356125\\
eval(\boldsymbol{v}_7) &= f(9.631066, 4.427881) = 35.458636\\
eval(\boldsymbol{v}_8) &= f(11.518106, 4.452835) = 23.309078\\
eval(\boldsymbol{v}_9) &= f(10.574816, 4.427933) = 34.393820\\
eval(\boldsymbol{v}_{10}) &= f(11.124627, 5.092514) = 30.316575\\
eval(\boldsymbol{v}_{11}) &= f(9.623693, 4.427881) = 35.477938\\
eval(\boldsymbol{v}_{12}) &= f(9.631066, 4.427933) = 35.456066\\
eval(\boldsymbol{v}_{13}) &= f(11.124627, 5.092514) = 30.316575\\
eval(\boldsymbol{v}_{14}) &= f(10.588756, 4.242514) = 32.932098\\
eval(\boldsymbol{v}_{15}) &= f(10.606555, 4.653714) = 30.746768\\
eval(\boldsymbol{v}_{16}) &= f(10.588814, 4.214603) = 34.359545
\end{aligned}
$$

$$
\begin{aligned}
eval(\boldsymbol{v}_{17}) &= f(10.588756, 4.242514) = 32.932098\\
eval(\boldsymbol{v}_{18}) &= f(10.588756, 4.242410) = 32.956664\\
eval(\boldsymbol{v}_{19}) &= f(11.518106, 4.472757) = 19.669670\\
eval(\boldsymbol{v}_{20}) &= f(10.588756, 4.242410) = 32.956664
\end{aligned}
$$

Remark:

It maybe that in earlier generations the fitness values of some chromosomes were better than the value 35.477938 of the best chromosome after 1000 generations.

• ***Mapping objective function values to fitness:***

~ Since a fitness function must be a nonnegative figure of merit, it is often necessary to map the underlying natural objective function to a fitness function form through one or more mappings

~ If the optimization problem is to minimize a cost function $g(x)$, then the following cost-to-fitness transformation is commonly used with GAs:

$$f(x) = \begin{cases} C_{max} - g(x) & \text{where } g(x) < C_{max} \\ 0 & \text{otherwise} \end{cases}$$

where C_{max} may be taken as input coefficient, for examples, as the largest g value observed thus far, the largest g value in the current population, or the largest of the last k generations.

~ When the original objective function is a profit or utility function $u(x)$ that is to be maximized, we simply transform fitness according to the equation

$$f(x)= \begin{cases} u(x)+ C_{min} & \text{when } u(x)+ C_{min} > 0 \\ 0 & \text{otherwise} \end{cases}$$

where C_{min} can be chosen as input coefficient such as the absolute value of the worse u value in the current or last k generations.

• ***Fitness scaling:***

~ To avoid that a few “super” individuals can potentially take over a large part of the population, thereby reducing its diversity and leading to premature convergence, especially in the first few generations.

~ Fitness scaling can help with this problem.

~ One useful scaling procedure is linear scaling.

~ Let us denote the raw fitness as f and the scaled fitness as f'. Linear scaling requires a linear relationship between f' and f as follows:

$$f' = af + b$$

where the coefficients a and b are chosen such that

$$f'_{avg} = f_{avg}$$

and $$f'_{max} = C_{mult} f_{ave}, \quad ....(*)$$

where the best fitness f'_{max} is increased by a desired multiple (C_{mult}) of the average fitness. For typically small populations (n= 50 to 100), C_{mult}= 1.2 to 2 has been used successfully.

~ The linear scaling function above may cause the low fitness values to go negative after scaling, violating the nonnegativity requirement. One solution is to replace the condition in Eq. (*) by the condition f'_{min}=0.

• *Floating point GA:*

~ *Coding:*

each gene is represented by a floating point number

Example (genetic fuzzy system) :

Rule j : IF x_1 is A_1^j AND x_2 is A_2^j AND · · · AND x_n is A_n^j , THEN y is w_j , $j = 1, \ldots, r$.

A_i^j : Gaussian membership function with center m_1^j and width σ_1^j

Chromosome:

$$| m_1^1 | \sigma_1^1 | m_2^1 | \sigma_2^1 | \cdots | m_n^1 | \sigma_n^1 | w_1 | \cdots\cdots | m_1^r | \sigma_1^r | \cdots | m_n^r | \sigma_n^r | w_r |$$

~ *Crossover*

(1) Arithmetical crossover:

if x_1 and x_2 are to be crossed,

offspring: $x_1' = a \cdot x_1 + (1-a) \cdot x_2$

$x_2' = a \cdot x_2 + (1-a) \cdot x_1$, a: a random value in [0, 1]

such a crossover was called a flat crossover; guaranteed average crossover (when a=1/2); intermediate crossover; and linear crossover.

(2) Simple crossover:

if $X_1 = (x_1, \cdots, x_q)$ and $X_2 = (y_1, \cdots, y_q)$ are crossed after the ith position,

offspring:

$X_1' = (x_1, \cdots, x_i, y_{i+1}, \cdots, y_q)$, $X_2' = (y_1, \cdots, y_i, x_{i+1}, \cdots, x_q)$

(3) k-point crossover:

k crossover sites are selected randomly within the range of an individual and swapping occurs.

~ *Mutation*

(1) uniform mutation:

$x_i^t = (v_1, \cdots, v_n)$ is a chromosome, suppose gene v_k is mutated,

result: $x_i^{t+1} = (v_1, \cdots, v_k', \cdots, v_n)$, t:generation number

v_k' : a random value from the domain of the corresponding parameter

(2) non-uniform mutation:

$s_i^t = (v_1, \cdots, v_m)$, gene v_k is mutated (domain of v_k is $[\ell_k, u_k]$)

result: $s_i^{t+1} = (v_1, \cdots, v_k', \cdots, v_m)$

$$v_k' = \begin{cases} v_k + \Delta(t, u_k - v_k) & \text{if a random digit is 0,} \\ v_k - \Delta(t, v_k - \ell_k) & \text{if a random digit is 1,} \end{cases}$$

where the function $\Delta(t, y)$ returns a value in the range $[0, y]$ such that the probability of $\Delta(t, y)$ being close to 0 as t increases.

e.g. $\Delta(t, y) = y \cdot (1 - r^{(1-\frac{t}{T})^b})$,

r: a random number from [0, 1],

T: the maximal generation number,

b: determining the degree of non-uniformity.

• Genetic fuzzy systems

~ Consider the TSK-type fuzzy systems

Rule j : IF x_1 is A_1^j AND x_2 is A_2^j AND …AND x_n is A_n^j,

THEN y is $a_0^j + \sum_{i=1}^{n} a_i^j x_i$, $\quad j = 1, \ldots, r.$

A_i^j: Gaussian membership function with center m_i^j and width σ_i^j

~ Chromosome:

$$| m_1^1 | \sigma_1^1 | \cdots | m_n^1 | \sigma_n^1 | a_0^1 | m_1^2 | \sigma_1^2 | \cdots\cdots | m_1^r | \sigma_1^r | \cdots | m_n^r | \sigma_n^r | a_0^r | a_1^1 | \cdots | a_1^n | \cdots | a_1^r | \cdots | a_n^r |$$

~ Example: fuzzy controller design

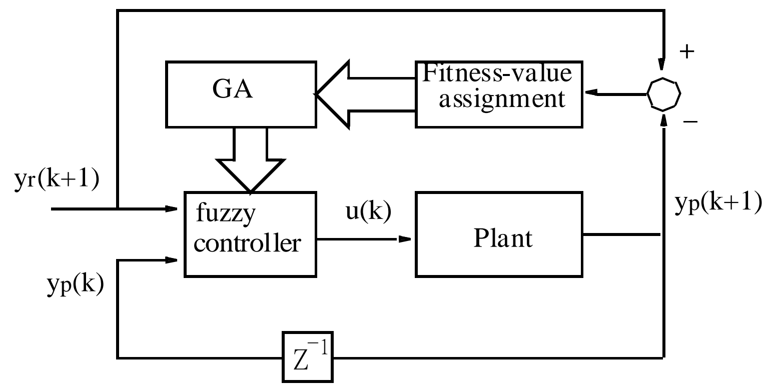
The plant to be controlled is given by

$$y_p(k+1) = \frac{y_p(k)}{1 + y_p^2(k)} + u^3(k)$$

reference output

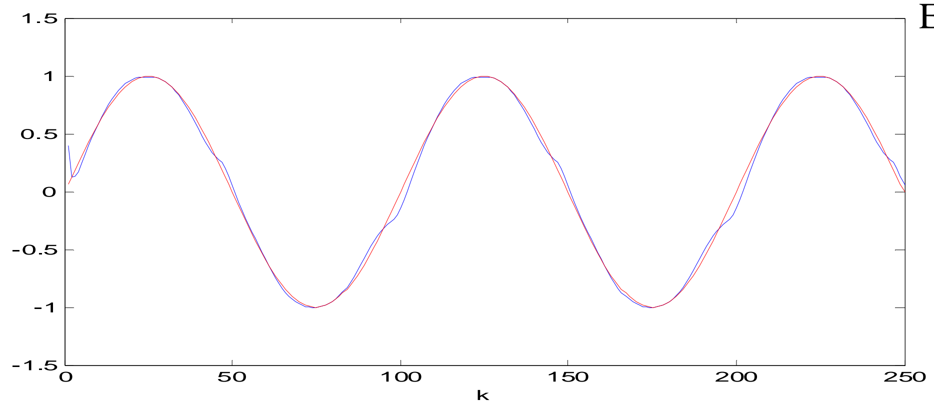
$$y_r(k) = \sin(2\pi k / 100), 1 \le k \le 250$$

~Control configuration

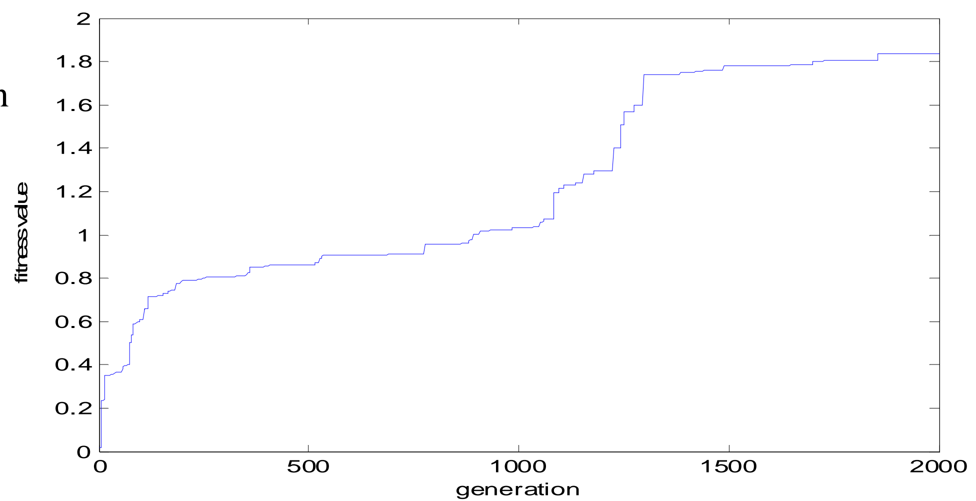


~ Fitness value: $$Fit = \frac{1}{\sum_{k=1}^{250} (y_r(k) - y_p(k))^2}$$

~control result



~fitness values
for each generation



```
//  Simple Genetic Algorithm
//  binary coded
//  roulette wheel method
//  function f(x) = exp^(-3.125*ln(2)*(x-0.1)^2)*sin(5*pi*x)^6 ; ;x:[0,1]

#include  <iostream.h>
#include  <conio.h>
#include  <stdlib.h>
#include  <math.h>
#include  <stdio.h>
#include  <string.h>
#include  <time.h>
#define    _pi         3.14159
#define   _popsize    40
#define   _lchrom     12     // chromosome length
#define   _maxgen     600      // max generation
#define   _pcross        0.6      // crossover probability
#define   _pmutation  0.01   // mutation probability
inline float max(float a , float b)
{ return ( ( a>b) ? a :b); }
```



```
 class  Population
{
  friend void  generation(void) ;
  friend void  crossover(int ,int ,int );
  friend void  report(void) ;
  float   chrom[_lchrom+1] ;
  float   x ;
  public :
  float   fitness ;
  void    initpop(void) ;
  void    decode(void) ;
  void    objfunc(void) ;
  void    save(void) ;
}  ;
   void  initialize(void) ;
   void  statistics(void) ;
   void  generation(void) ;
   void  report(void) ;

   Population   _oldpop[_popsize+1],_newpop[_popsize+1]  ;
   int     _gen,_nmutation=0,_nncross=0  ;
   float   _sumfitness, _avg,_max,_min ;
```

```
  int main(void)
{ int   i ;

  _gen = 0 ;
  initialize() ;

 while ( _gen < _maxgen )
 {
 _gen = _gen + 1 ;
 generation() ;
 for(i=1;i<=_popsize;i++)
  _oldpop[i] = _newpop[i] ;
  statistics() ;
  report() ;
 }

for(i=1;i<=_popsize;i++)
  _oldpop[i].Save() ;

 getch() ;
 return(0);
}
```



```
void initialize()
{ int j ;
 void initreport(void) ;
 time_t t;

 cout << "\n ****** Simple Genetic Algorithm ****** " << endl ;
srand((unsigned) time(&t));
for(j=1;j<=_popsize;j++)
     { _oldpop[j].initpop();
        _oldpop[j].decode();
        _oldpop[j].objfunc() ;
      }
  statistics() ;
  report() ;
}

void Population :: initpop()
{ int j,j1 ;

 for(j1=1;j1<=_lchrom;j1++)
 chrom[j1] = rand()%2 ;
 }
```

```
void Population :: decode()
{
  int j ;
  float powerof2,coef ;
     x = 0 ;
    powerof2 = 1 ;
    for(j=1;j<=_lchrom;j++)
    {
     if (chrom[j]) x += powerof2 ;
     powerof2 = powerof2 * 2. ;
    }
 coef = pow(2,_lchrom) -1;
 x = x /coef ;
 // return accum ;
 }

void Population :: objfunc()
{
  fitness = exp(-3.125*log(2)*pow(x-0.1,2))*pow(sin(5*_pi*x),6) ;
}
```



```
void statistics()
{   int j ;
    _sumfitness = _oldpop[1].fitness ;
    _min = _oldpop[1].fitness ;
    _max = _oldpop[1].fitness ;
    for(j=2;j<=_popsize;j++)
    {
     _sumfitness += _oldpop[j].fitness ;
     if ( _oldpop[j].fitness > _max ) _max = _oldpop[j].fitness;
     if ( _oldpop[j].fitness < _min ) _min = _oldpop[j].fitness ;
     }
 _avg = _sumfitness/_popsize ;
}

int flip(float prob)
{   int i,j ;
  i = rand()%1000 ;
  if ( i  < 1000*prob )
  j = 1 ;
  else
  j = 0 ;
  return j ; }
```

```
void generation()
{
  int   i,mate1,mate2 ;
  int   select(void) ;
  void  crossover(int ,int ,int );
 i = 1 ;
  while(i<= _popsize)
  {
  mate1 = select() ;
  mate2 = select() ;
  crossover(i,mate1,mate2) ;
  _newpop[i].decode() ;
  _newpop[i].objfunc();
  _newpop[i+1].decode() ;
  _newpop[i+1].objfunc();
  i= i+2 ;
  }
 }
```



```
int select()
{ float ran,partsum=0 ;
  int j=0 ;
 ran=_sumfitness*(rand()%10000)/9999.;
 while((partsum<ran)&&(j !=_popsize))
 {  j=j+1;
    partsum=partsum+_oldpop[j].fitness;
  }
 return j;
}

void  crossover(int i,int mate1,int mate2)
{
int j,jcross ;
int mutation(int) ;
if (flip(_pcross))
{jcross = rand()%(_lchrom-1)+1 ;
 _nncross = _nncross + 1 ;
}
 else
  jcross = _lchrom ;
```

```
for(j=1;j<=jcross;j++)
{ newpop[i].chrom[j]   = mutation(_oldpop[mate1].chrom[j]);
 _newpop[i+1].chrom[j] = mutation(_oldpop[mate2].chrom[j]);
}
if ( jcross != _lchrom )
 for (j=jcross+1;j<=_lchrom;j++)
 {_newpop[i].chrom[j]   = mutation(_oldpop[mate2].chrom[j]);
   _newpop[i+1].chrom[j] = mutation(_oldpop[mate1].chrom[j]);
 }
}

int mutation(int ge)
{  int muta ;
  if (flip(_pmutation))
 {_nmutation = _nmutation + 1 ;
   if (ge == 0 ) muta = 1 ;
   if (ge == 1 ) muta = 0 ;
 }
else
 muta = ge ;
return muta ;
}
```



```
void report()
{
int i,j ;
cout << “ \n Generation is ------ " << _gen ;

cout << "\n ------------------------------------------------- " ;
cout << "\n max --- " << _max << " , min --- " << _min << " , avg -- " <<
     _avg ;
cout << "\n sum_fitness " << _sumfitness <<" , nmutation " <<
     _nmutation  ;
cout <<  ", ncross ----- " << _nncross ;
_nmutation = 0 ;
 _nncross = 0 ;
}

void Population :: Save()
{ FILE *fy  ;
  if( (fy=fopen("y.dat","a"))==NULL) exit(1) ;
  fprintf(fy,"%f %f \n",x,fitness);
  fclose(fy) ;
 }
```

- Program for tournament selection

```
const int _number=2 ;
int tournament_select()
{  int ran[_number+1]  ;
     int max,j=0 ;

   for(j=1;j<=_number;j++)
   ran[j] = rand()%_popsize + 1  ;

  max = ran[1] ;

  for(j=2;j<=_number;j++)
  {
     if ( _oldpop[ran[j]].fitness > _oldpop[max].fitness )
       { max = ran[j] ; }
  }

  return max ;

}
```