Genetic Algorithm

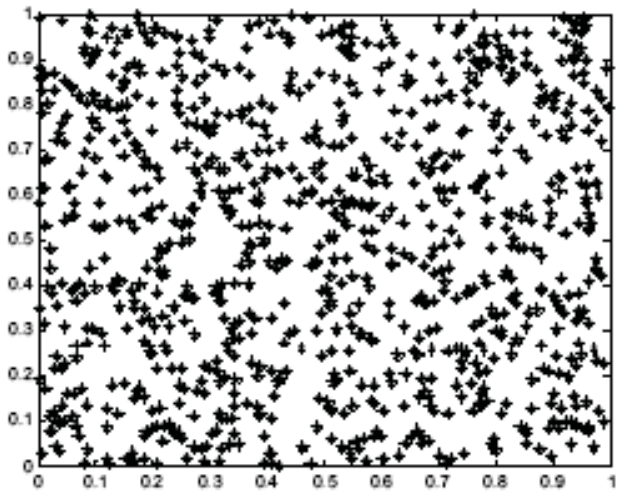
An Evolutionary Approach

Genetic Algorithm

- Were formally introduced in the US in the 1970s by John Holland at University of Michigan
- A class of probabilistic optimization algorithms
- Uses concepts of “Natural Selection" and “Genetic Inheritance" (Darwin 1859)
- In nature, competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones.

Elements of a Genetic Algorithm

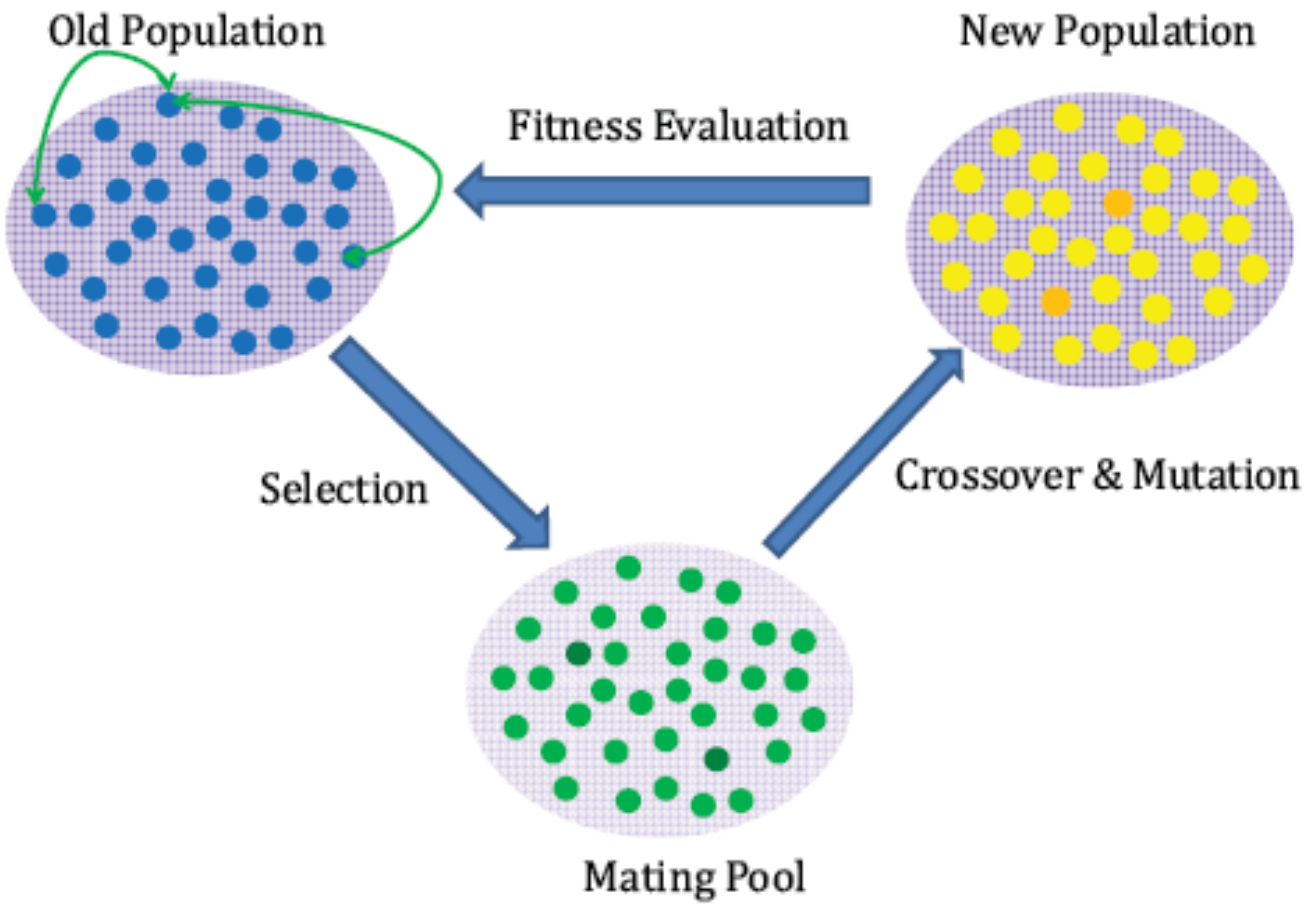
- Population Initialization



- Fitness Evaluation:

$$F(x) = -f(x), \frac{1}{1+f(x)} \; etc.$$

Elements of a Genetic Algorithm



Problem 1

A cylindrical 'can' is to be designed with two parameters — diameter (d) and height (h). Let us consider that the can needs to have a volume of at least 300 ml and the objective of the design is to minimize the cost of 'can' material.

Contd … Problem

Corresponding Nonlinear Programming Problem (NLP):

$$
\begin{aligned}
\text{Minimize} \quad & f(d,h) = c\left(\frac{\pi d^2}{2} + \pi dh\right), \\
\text{Subject to} \quad & g_1(d,h) \equiv \frac{\pi d^2 h}{4} \geq 300, \\
\text{Variable bounds} \quad & d_{\min} \leq d \leq d_{\max}, \\
& h_{\min} \leq h \leq h_{\max}.
\end{aligned}
$$

The parameter '*c*' is the cost of 'can' material per squared cm, and diameter '*d*' and height '*h*' are allowed to vary in $[d_{min}, d_{max}]$ and $[h_{min}, h_{max}]$ cm, respectively.

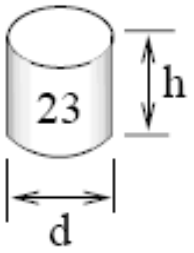
Representing a solution

To find the optimal parameter values of '*d*' and '*h*' which satisfies the constraint 'g_1' and minimizes 'f', we first need to represent the parameter values in binary strings.

Let us assume that we shall use five bits to code each of the two design parameters '*d*' and '*h*', thereby making the overall string length equal to 10.

The following string represents a can of diameter 8 cm and height 10 cm:

$$\underbrace{01000}_{d}\ \underbrace{01010}_{h}$$



(d, h) = (8, 10) cm

(Chromosome) = 0 1 0 0 0 0 1 0 1 0

Contd … Representing a solution

- With five bits to represent a parameter, exactly 32 different solutions are possible

- Choosing the lower and upper bounds like that allows GAs to consider only integer values in the range [0, 31].

- However, GAs are not restricted to use only integer values in the above range, in fact GAs can be assigned to use any other integer or non-integer values just by changing the string length and lower and upper bounds:

$$x_i = x_i^{\min} + \frac{x_i^{\max} - x_i^{\min}}{2^{\ell_i} - 1} \mathrm{DV}(s_i),$$

Where 'l_i' is the string length used to code *i*-th parameter and DV(s_i) is the decoded value of the string s_i.

Contd … Representing a solution

The above mapping function allows following properties to achieve in the parameter values:

1. Any arbitrary precision can be achieved in the parameter values by using along enough string.
2. Different parameters can have different precision by simply using different string lengths.
3. Parameters are allowed to take positive and negative values.

Assigning fitness to a solution

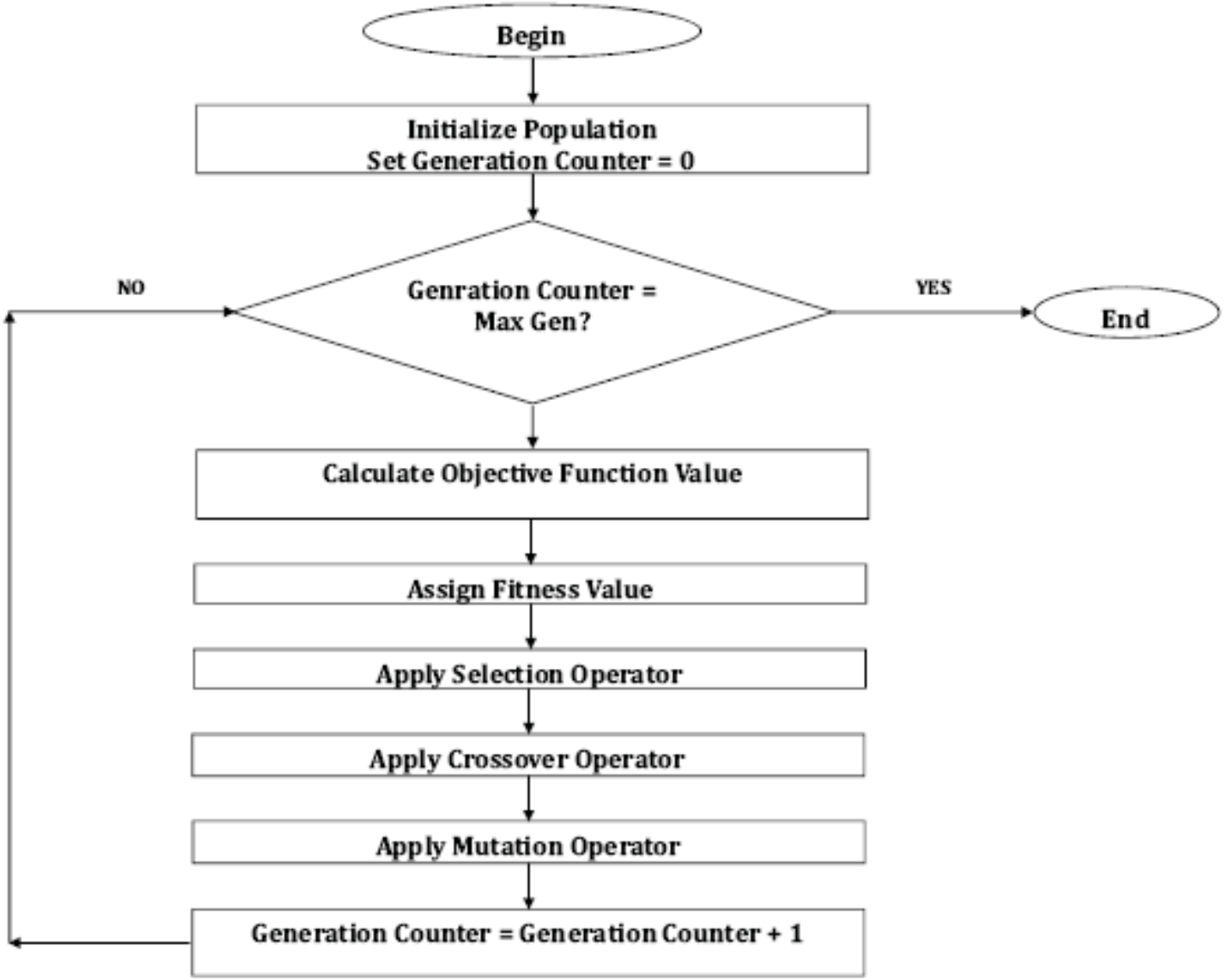
- GAs work with strings representing design parameters, instead of parameters themselves
- Once a string (or a solution) is created by genetic operators, it is necessary to evaluate the solution, particularly in the context of the underlying objective and constraint functions
- In the absence of constraints, the fitness of a string is assigned a value which is a function of the solution's objective function value
- In most cases, however, the fitness is made equal to the objective function value. For example, the fitness of the above can represented by the10-bit string is

$$\begin{aligned} F(s) &= 0.0654\left(\pi\,(8)^2/2 + \pi\,(8)\,(10)\right), \\ &= 23, \end{aligned}$$

Contd …Assigning fitness to a solution

Since the objective of the optimization is to minimize the objective function, it is to be noted that a solution with a smaller fitness value compared to another solution is better.

Genetic Algorithm



Initial Population

The figure below shows a random population of six ‘cans’. The fitness of each ‘can’ is marked on the ‘can’. It is interesting to note that two solutions do not have 300ml volume inside and thus have been penalized by adding an extra artificial cost.

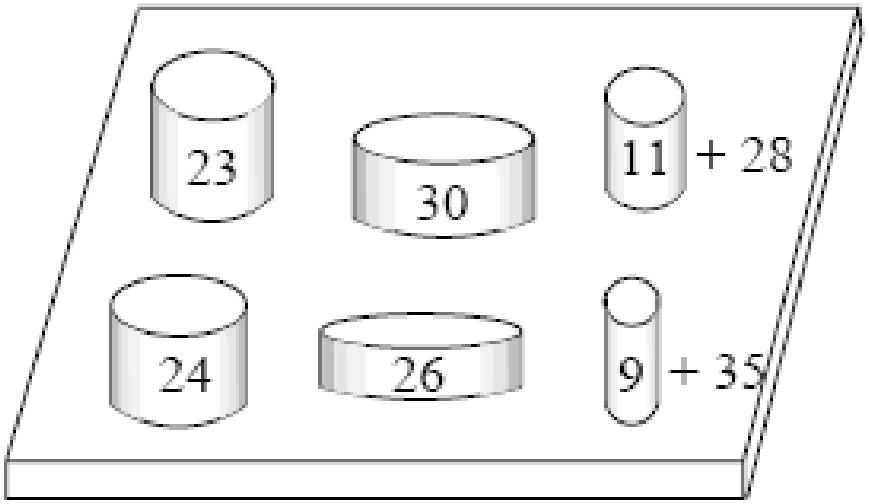


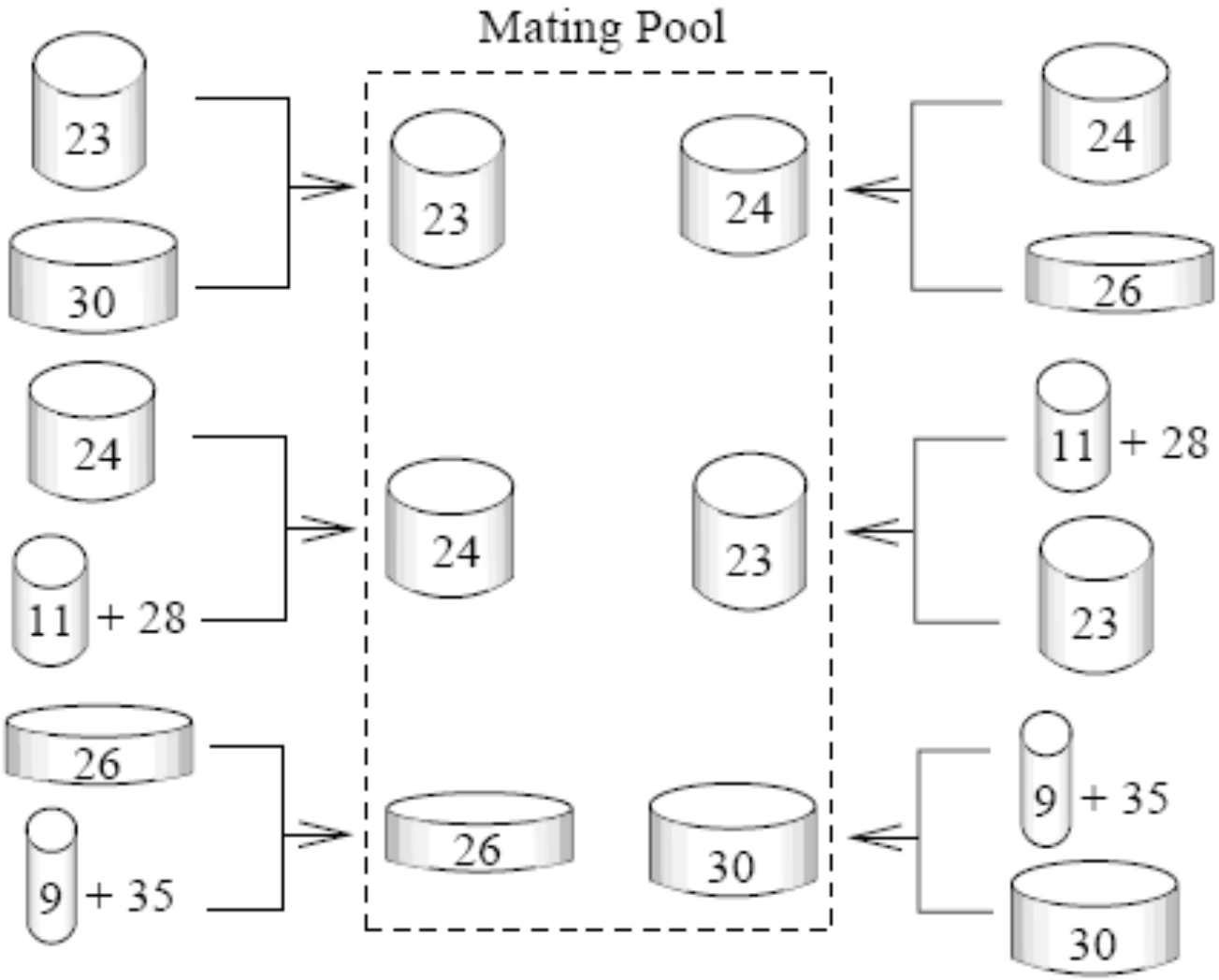
Figure 3: A random population of six cans is created.

Reproduction or Selection Operator

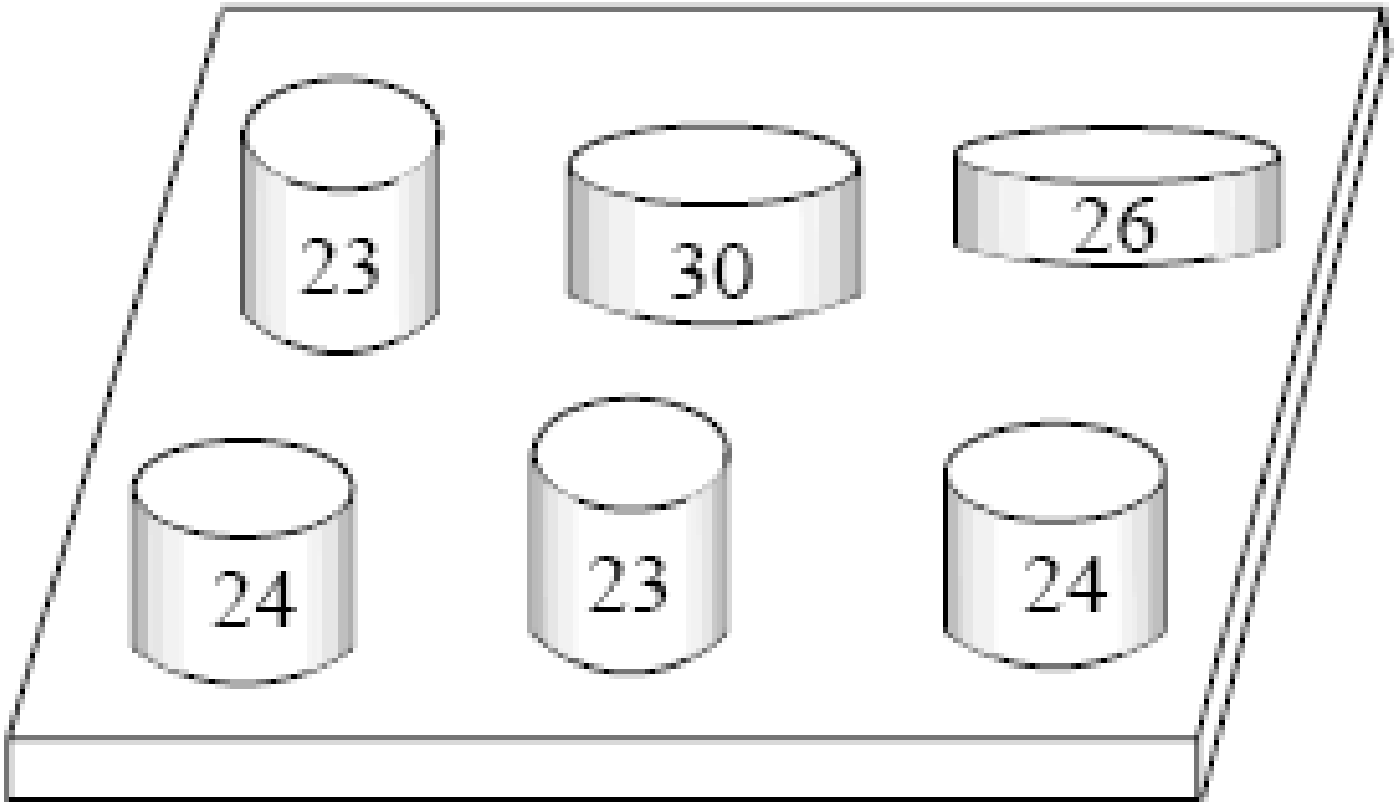
- The primary objective of the reproduction operator
  - To emphasize good solutions and eliminate bad solutions in a population, while keeping the population size constant

- This is achieved by performing the following tasks:
  1. Identify good (usually above-average) solutions in a population.
  2. Make multiple copies of good solutions.
  3. Eliminate bad solutions from the population so that multiple copies of good solutions can be placed in the population.
- There exist a number of ways to achieve the above tasks
  - Tournament selection
  - Ranking selection etc.

Reproduction or Selection Operator



Reproduction or Selection Operator



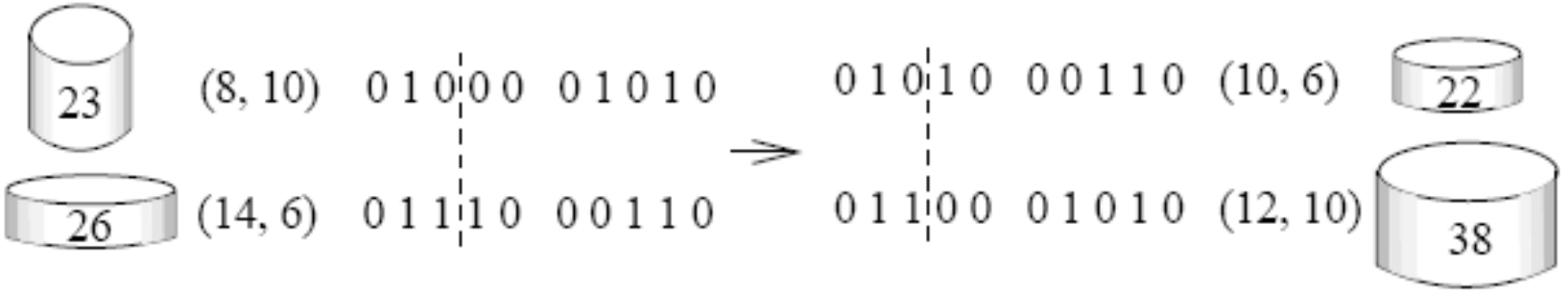
Crossover Operator

• Crossover operator is applied next to the strings of the mating pool

• A little thought will indicate that the reproduction operator can not create any new solutions in the population

• It only made more copies of good solutions at the expense of not-so-good solutions

• Creation of new solutions is performed in crossover and mutation operators.



Contd … Crossover operator

• It is true that every crossover between any two solutions from the new population is not likely to find children solutions better than parent solutions

• But the chance of creating better solutions is far better than random

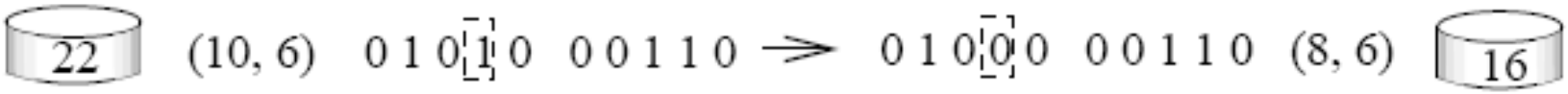
• It is true because parent strings being crossed are not any two arbitrary random solutions, they have survived during the reproduction phase

• Thus, they are expected to have some good bit combinations in their string representations

•Though, every crossover may not create better solutions, but we do not worry about it too much. If bad solutions are created, they will get eliminated in the next reproduction operator and hence will have a short life - **Elitism**

Mutation

- The need for mutation is to keep diversity in the population
- Figure below shows how a string obtained after reproduction and crossover operators has been mutated to another string, representing a slightly different can.



- Once again, the solution obtained is better than that the original solution. Although, it may not happen all the times

- Mutating a string with a small probability is not a random operation since the process has a bias for creating a few solutions in the neighborhood of the original solution

Problem 2

$$\begin{aligned} \textbf{Maximize} \quad & \sin(x) \\ \textbf{Variable bound} \quad & 0 \leq x \leq \pi. \end{aligned}$$

We use five-bit strings to represent the variable x in the range [0, π], so that the string (00000) represents x = 0 solution and the string (11111) represents x = π solution. Other 30 strings are mapped in the range [0, π].

Contd . . . Problem 2

Initial population								New population			
String	DV[a]	x	$f(x)$	$f_i/\bar{f}$	AC[b]	Mating pool	CS[c]	String	DV	x	$f(x)$
01001	9	0.912	0.791	1.39	1	01001	3	01000	8	0.811	0.725
10100	20	2.027	0.898	1.58	2	10100	3	10101	21	2.128	0.849
00001	1	0.101	0.101	0.18	0	10100	2	11100	28	2.838	0.299
11010	26	2.635	0.485	0.85	1	11010	2	10010	18	1.824	0.968
		Average, f	0.569							Average, f	0.711

[a] DV stands for decoded value of the string.

[b] AC stands for actual count of strings in the population.

[c] CS stands for cross site.

Books

- D. E. Goldberg, "*Genetic algorithms in search, optimization, and machine learning*", New York: Addison-Wesley, 1989.
- J. H. Holland, "*Adaptation in natural and artificial systems*", Ann Arbor: University of Michigan Press, 1975.
- Z. Michalewicz, "*Genetic Algorithms Data Structures Evolution Programs*", Berlin: Springer-Verlag, 1992.
- M. Mitchell, "*Introduction to Genetic Algorithms*", Ann Arbor: MIT Press, 1996 (Also New Delhi: Prentice-Hall).
- M. Gen and R. Cheng, "*Genetic Algorithms and Engineering Design*" New York: Wiley, 1997.
- T. Back, D. Fogel and Z. Michalewicz (Eds.), "*Handbook of Evolutionary Computation*", Institute of Physics Publishing and Oxford University Press, New York. 1997.