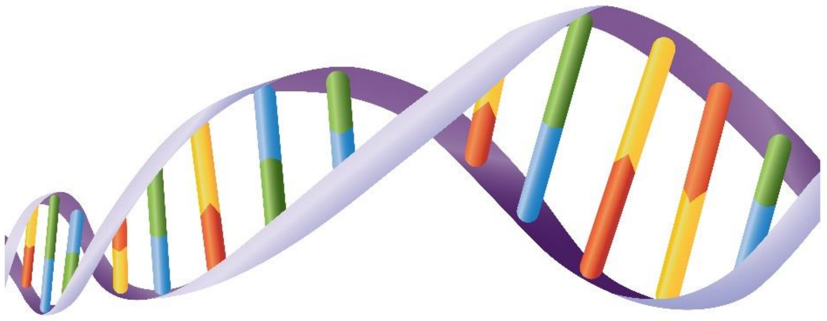
# Genetic Algorithms

## Textbook Reading



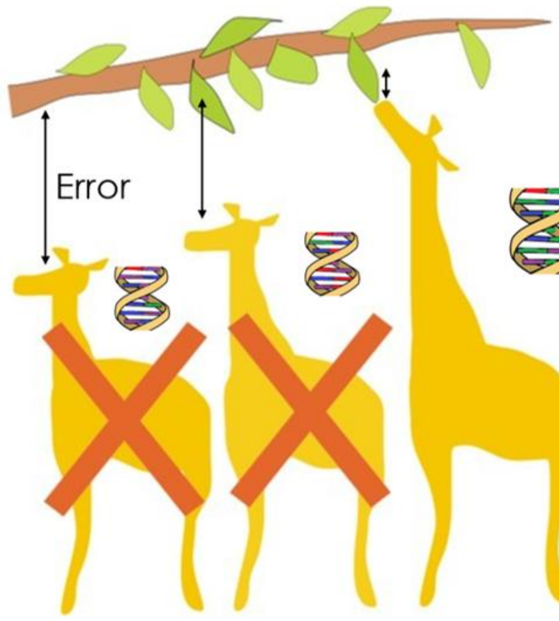
Algorithms: *Special Topics*:

Chapter 9

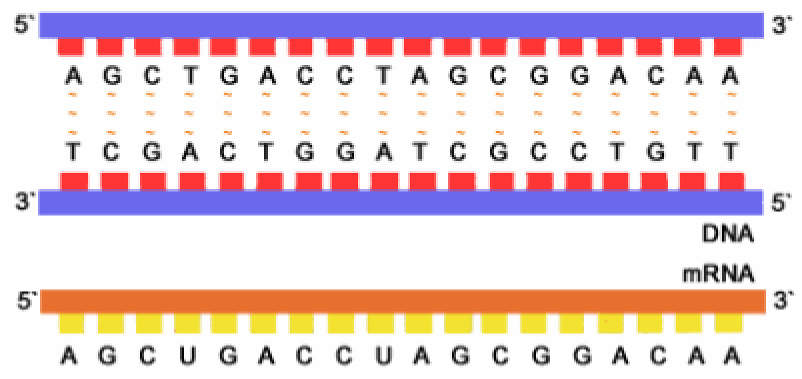
- 9.1 Reproduction, Crossover, and Mutation
- 9.2 Reproduction: Survival of the Fittest
- 9.3 Crossover Operations
- 9.4 Mutation
- 9.5 An SGA in Action: Reproduction, Crossover, Mutation

## Genetic Algorithms: Optimization based on Natural Selection

- The genetic algorithm (GA) design strategy is based on *natural selection* and *survival of the fittest* laws of nature popularized by Darwin. In analogy to genetic DNA molecules, inputs to a GA are strings drawn from a given alphabet.
- The genetic algorithm design strategy, introduced by Holand in the 1970s.
- Genetic algorithms have been utilized with success in a wide variety of optimization applications.



# String Representation


5'
3'
A G C T G A C C T A G C G G A C A A
T C G A C T G G A T C G C C T G T T
3'
5'
DNA
mRNA
5'
3'
A G C U G A C C U A G C G G A C A A

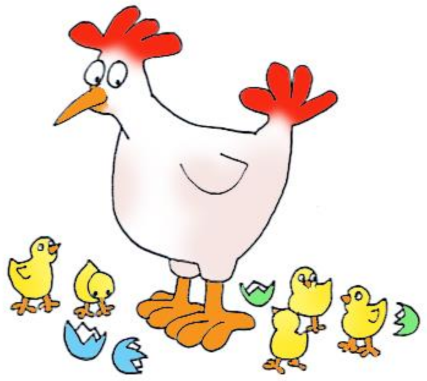
- In analogy to genetic DNA molecules, inputs to a GA are strings drawn from a given alphabet.
- Through appropriate encoding, there is no loss of generality in assuming that the alphabet is simply a two element set of 0's and 1's.
- Thus, we assume that solving a problem using GA requires that we suitably encode problems instances as binary strings whose length depends only on the input size to the algorithm.

# Population and Fitness Function

- Through appropriate encoding, there is no loss of generality in assuming that the alphabet is simply a two element set of 0’s and 1’s.
- A GA requires that there be a suitable fitness function *f* defined on each input string to the algorithms.
- A GA seeks to find a string having optimal fitness, starting with a randomly selected set (**population**) of *n* strings (**individuals**), called the **initial (first) generation***.*
- Each successive generation consists of *n* strings and is generated from the previous generation by selecting *n*/2 pairs of strings and “mating” each pair to produce another pair. We assume *n* is even.
- The choice on *n* should be large enough to yield a good sampling of the set of all binary strings encoding the inputs to the original problem.

## Simple Genetic Algorithm (SGA)

- We limit our discussion to what’s known as the simple genetic algorithm (SGA).
- Emulating natural processes, an SGA computes successive generations by utilizing three basic operations:
  - **reproduction** (**selection**)
  - **crossover**
  - **mutation**



- Reproduction, crossover and mutation each are performed according to various probabilistic rules.

## Selecting fittest for next generation

- SGA selects pairs of individuals for the $(i+1)^{st}$ generation from the $i^{th}$ generation according to a probability distribution determined by the relative fitness of the individuals making up the $i^{th}$ generation.
- The more fit the individual, the more likely it is to be selected for inclusion in the next generation.
- We select individuals with replacement so, in particular, a highly fit individual has a good chance of being selected more than once for inclusion in the next generation.

## Mating

- After a pair of individuals is selected, we mate the two individuals.
- The mating of **two strings** produces **two offspring**, which replace the two parents in the population for the next generation.
- The mating procedure is a combination of a crossover operation  together with a possible mutation of individual bits (changing 0 to 1 or 1 to 0, as appropriate).
- Both crossover and mutation occur according to predetermined probabilities.
- In particular, often neither crossover nor mutation occurs and the parents become part of the next generations without change.

## Crossover and Mutation

- The aim of crossover is to produce an offspring having a high potential to inherit good fitness qualities from both parents and thereby be even more fit than either parent individually.
- Mutation is relatively rare operations and usually many generations pass in between generations where a mutation occurs.
- Mutations are performed as an insurance measure against the possibility that an important fitness trait has been lost (or was never present) in the entire population corresponding to a given generation.

## SGA Process

- An SGA repeats the process of computing a new generation for a predetermined maximum number of times.
- At termination an SGA might output the individual in the final generation having the best fitness, or alternatively, it might maintain and output the fittest individual ever generated.
- In certain situations an SGA might terminate before the predetermined maximum number of generations if an individual has been generated whose fitness can be shown to be optimal (or nearly optimal). The latter termination conditions requires and efficient test for optimality, which is often not available.
- Another termination condition might involve measuring the largest deviation of an individual's fitness from the average. The SGA might stop whenever the latter deviation is sufficiently small.

## Example

Consider the problem of finding the maximum value of the function $f(x) = 100x - x^2$,

where $x$ is an integer between 0 and 63.

Since 63 = $2^6 - 1$, inputs to an SGA for this problem are precisely all binary strings of length 6.

For convenience, we chose our input interval to be of the form $[0,2^k - 1]$ , so that inputs correspond to all binary strings of length $k$.

# maximize $f(x) = 100x - x^2$

Assume that the initial population (first generation) is given by the four randomly generated strings: 000110, 011110, 010010, 110111 encoding the decimal numbers 6, 30, 18, and 55, respectively.

$f_i = f(x_i)$ is the **fitness** of the $i^{\text{th}}$ string (number) $x_i, \; i = 1,2,3,4.$

$f_i/F$ is the **relative fitness** of the $x_i$ where $F = \sum_{j=1}^{4} f_j$

$\bar{f} = F/4$ is the **average fitness of the population**

$E_i = 4f_i/F = f_i/\bar{f}$ is the **expected number of times** that $x_i$ is chosen for the next generation.

| **string** | **000110** | **011110** | **010010** | **110111** |
|---|---|---|---|---|
| **fitness** | 564 | 2100 | 1476 | 2475 |
| **relative fitness** | 0.085 | 0.317 | 0.223 | 0.374 |
| ***E*** | 0.341 | 1.270 | 0.893 | 1.500 |

## Next Generation

- A new generation also involves $n$ strings and is obtained by selecting $n/2$ pairs for mating from the old generation.
- Individuals are selected using the following reproduction rule that models the Darwinian survival of the fittest hypothesis of nature.
- Each time a selection is made from the old generation, ***the survival of the fittest rule*** states that the $i^{\text{th}}$ individual has a probability of being selected for the next generation equal to its relative fitness, i.e.,

**probability $\boldsymbol{x_i}$ is selected is** $\boldsymbol{f_i/(\sum_{j=1}^{n} f_j) = f_i/F}$.

- So that the expected number of times that an individual is selected for the next generation is given by $E_i = nf_i/(\sum_{j=1}^{n} f_j) = f_i/\bar{f}$.
- In our example maximizing $f(x) = 100x - x^2$ following the survival of the fittest rule the expected number of times that individual 1,2,3,4 is selected from the old generation is 0.341, 1.27, 0.892 and 1.5, respectively. Hence, a likely scenario would be individual 1,2,3,4 would be selected 0,1,1,2 times, respectively.

# Selecting $x_i$ for next generation

- Divide up the line interval $[0, F]$ into the $n$ segments

$[0, f_1]$, $(f_{1,} f_1 + f_2]$, $(f_1 + f_{2,} , f_1 + f_{2,} + f_3]$ ,…, $(f_1 + f_2 + f_{3+} \dots + f_{n-1}, f_1 + f_2 + f_{3+} \dots + f_n]$.

- Then invoke *Random*(0,*F*) that returns a random number *m* between 0 and *F*, and select $x_i$ if $m$ belongs to the $i^{\text{th}}$ segment $(f_1 + f_2 + f_{3+} \dots + f_{i-1}, f_1 + f_2 + f_{3+} \dots + f_i]$.

# Psuedocode for *Select*

```
Function Select(Fitness[0:n])
Input: Fitness[0, n] (an array with $Fitness[i] = f(x_i) = f_i$,   $i = 1,2, \ldots , n$)
Output:  an index i subject to the condition  $P(x_i is\ selected) = f_i/(\sum_{j=1}^{n} f_j)$,
                                                                          $i = 1, \ldots, n$.

    $F \leftarrow Fitness[1]$
    for $i \leftarrow 2$ to $n$ do
         $F \leftarrow F + Fitness[i]$
    endfor
    $m \leftarrow Random(0, F)$            //$m$ is a randomly chosen value  in $[0, F]$
    $fit \leftarrow 0$
    for $i \leftarrow 1$ to $n$ do
         $fit \leftarrow fit + Fitness[i]$
        if  $m \le fit$ then
             return($i$)
        endif
    endfor
end SelectFitness
```

## Complexity of application of *Select*

After *n* invocations of *Select*, *n* individuals have been selected  so complexity of *Select* belongs to *O*(*n*).

## Crossover Operation

After selecting a pair of individuals, i.e., binary strings of length $l$
$a = a_1a_2 \dots a_l$ and $b = b_1b_2 \dots b_l$

for inclusion in the next generation, a **crossover** operation is probabilistically performed on the pair.

There are various types of crossover operations.

## 1-Point Crossover

In the 1-point crossover method, a crossover point *CrossPoint* between 1 and $l - 1$ is randomly selected, two (offspring) strings *a'*, *b'* are produced as follows:

> *a'* agrees with *a* in bit positions 1 through *CrossPoint* and agrees with *b* in bit positions *CrossPoint* + 1 through $l$,
>
> *b'* agrees with *b* in bit positions 1 through *CrossPoint* and agrees with *a* in bit positions *CrossPoint* + 1 through $l$.

For example

001|1010011 → 0011011100
110|1011100 → 1101010011

crossover point 3

00110100|11 → 0011010000
11010111|00 → 1101011111

crossover point 8

# 2-Point Crossover

In a **2-point crossover**, points *CrossPoint1* < *CrossPoint2* between 1 and $l$ are randomly selected, two (offspring) strings *a′*, *b′* are produced as follows:

*a′* agrees with *a* in bit positions 1 through *CrossPoint1* and *CrossPoint2* through $l$ and agrees with *b* in bit positions *CrossPoint1* + 1 through *CrossPoint2* – 1,

*b′* agrees with *b* in bit positions 1 through *CrossPoint1* and *CrossPoint2* through $l$ and agrees with *a* in bit positions *CrossPoint1* + 1 through *CrossPoint2* – 1.

2-point crossover operation applied to the same strings *a* = 00110100111, *b* = 11010111000 of length 11 for **crossover points 2 and 9**.

00|1101001|11 → 00010111011

11|0101110|00 → 11110100100

# PSN

a) Give 1-point crossover for 11010100110 and 00001110111 at crossover point 5.

b) Repeat for 2-point crossover at crossover points 3 and 6.

# Uniform Crossover

In both 1-point and 2-point crossovers, entire segments are simply exchanged between the two strings. In a uniform crossover, the exchange in a segment is made point by point based on coin flips.

Consider the 2-point crossover in our example, i.e.

00|1101001|11 → 00010111011

11|0101110|00 → 11110100100

Instead of exchanging the whole segment between position 3 and 8, a coin flip would be done at each position in this segment to see if an exchange is made (heads) or not (tails) at that position in the offspring.

For example, suppose the coin flips result in HHTTTTH. Then the resulting offsprings would be 00010111111, and 11110100000, respectively.

00|1101001|11 → 00010100011

11|0101110|00 → 11110111100

## Mutation

- The mutation operation consists of simply changing the parity of a single bit.
- For example, given the bit string 0011010011, the mutation operation applied to the 4th bit results in the bit string 0010010011.
- $f(x) = 100x - x^2$ on the interval of integers [0,63]. Then without mutation the best solution we could possibly find (even allowing crossovers) would be 011111, having suboptimal value $f(31) = 2139$. Thus, one would hope that a mutation of one of the offspring would happen in the leading coefficient changing it to 1.

# SGA in Action

- An SGA begins with an initial generation of $n$ randomly generated bit string of length $l$.
- Then the SGA produces successive generations where a new generation is obtained from its predecessor (old generation) by selecting $n/2$ pairs from the old generation using *Select* and mating each of the $n/2$ pairs using crossover and mutation.
- After a pair of individuals is selected for mating by *Select*, a (biased) coin flip is performed which comes up heads with probability $p_c$.
- A crossover occurs if and only if the coin comes up heads.
- If a crossover is called for, then a uniformly random crossover position *CrossPoint* is chosen between  and 1 and $l - 1$ (recall $l$ is the fixed length of the individual bit strings in the population).
- In both cases (crossover versus no crossover) the offspring bit strings of the two parents is computed in a simple bit-by-bit copy.
- The 2-point and Uniform Crossover operations are performed similarly.

# SGA in Action cont'd

- Mutation can be accomplished simultaneously with the crossover operations in a bit-by-bit fashion.
- As each bit is copied to a particular offspring, a coin flip occurs to see whether or not the bit will be altered.
- However, since mutation is a rare event, in practice the multiple calls to a random number generator to implement mutation are usually avoided by making a single coin flip (after the crossover operation has been completed) for each offspring to see whether a mutation will occur.
- If the coin flip calls for a mutation, then a call to $Random([1, l])$ determines the location of the bit to be altered.

# Illustrating computing next generation with earlier example maximize $f(x) = 100x - x^2$

- The relative fitness of the individuals in the first generation makes it reasonable that *Select* selects the 2nd string 011110 and the 4th string 110111 for mating and selects the 3rd string 010010 and the 4th string 110111 (again) for mating.
- Suppose that no 1-point Crossover occurs in the mating of the first pair and that a 1-point Crossover occurs at crossover point 3 in the mating of the second pair. Suppose also that no mutation occurs.
- Then, the mating of the first pair yields the offspring 011110 and 110111, whereas the mating of the second pair yields the offspring and 110010 and 010111.

## Table from previous slide

| string | 000110 | 011110 | 010010 | 110111 |
|---|---|---|---|---|
| fitness | 564 | 2100 | 1476 | 2475 |
| relative fitness | 0.085 | 0.317 | 0.223 | 0.374 |
| *E* | 0.341 | 1.270 | 0.893 | 1.500 |

## New generation

| string | 011110 | 110111 | 110010 | 010111 |
|---|---|---|---|---|
| fitness | 2100 | 2475 | 2500 | 1771 |
| relative fitness | 0.237 | 0.280 | 0.283 | 0.200 |
| *E* | 0.950 | 1.120 | 1.130 | 0.200 |

## What did one DNA say to the other DNA?

Do these genes make me look fat?