Basics of Genetic Algorithm

History of GAs

- As early as 1962, John Holland's work on adaptive systems laid the foundation for later developments.

- By the 1975, the publication of the book *Adaptation in Natural and Artificial Systems*, by Holland and his students and colleagues.

History of GAs

- early to mid-1980s, genetic algorithms were being applied to a broad range of subjects.

- In 1992 John Koza has used genetic algorithm to evolve programs to perform certain tasks. He called his method "genetic programming" (GP).

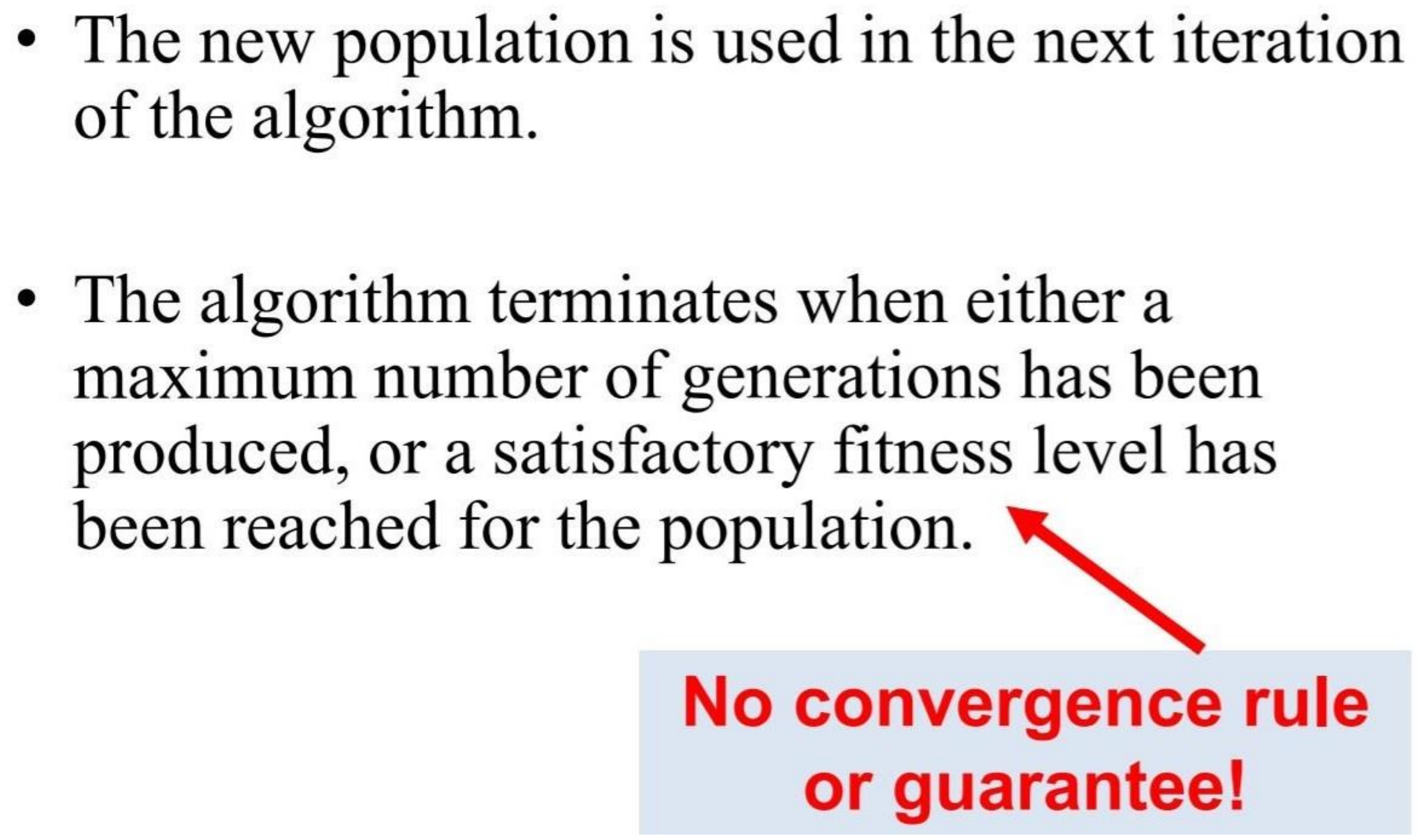
What is GA

- A genetic algorithm (or GA) is a search technique used in computing to find true or approximate solutions to optimization and search problems.
- (GA)s are categorized as global search heuristics.
- (GA)s are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

What is GA

- The evolution usually starts from a population of randomly generated individuals and happens in generations.
- In each generation, the fitness of every individual in the population is evaluated, multiple individuals are selected from the current population (based on their fitness), and modified to form a new population.

What is GA



- The new population is used in the next iteration of the algorithm.

- The algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

No convergence rule or guarantee!

Vocabulary

- **Individual** - Any possible solution
- **Population** - Group of all individuals
- **Fitness** – Target function that we are optimizing (each individual has a fitness)
- **Trait** - Possible aspect (features) of an individual
- **Genome** - Collection of all chromosomes (traits) for an individual.

Basic Genetic Algorithm

- Start with a large “population” of randomly generated “attempted solutions” to a problem
- Repeatedly do the following:
  - Evaluate each of the attempted solutions
  - (probabilistically) keep a subset of the best solutions
  - Use these solutions to generate a new population
- Quit when you have a satisfactory solution (or you run out of time)

Example:
the MAXONE problem

Suppose we want to maximize the number of ones in a string of l binary digits

Is it a trivial problem?

It may seem so because we know the answer in advance

However, we can think of it as maximizing the number of correct answers, each encoded by 1, to l yes/no difficult questions`

Example (cont)

- An individual is encoded (naturally) as a string of l binary digits
- The fitness f of a candidate solution to the MAXONE problem is the number of ones in its genetic code
- We start with a population of n random strings. Suppose that $l = 10$ and $n = 6$

Example (initialization)

We toss a fair coin 60 times and get the following initial population:

$s_1 = 1111010101 \quad f(s_1) = 7$

$s_2 = 0111000101 \quad f(s_2) = 5$

$s_3 = 1110110101 \quad f(s_3) = 7$

$s_4 = 0100010011 \quad f(s_4) = 4$

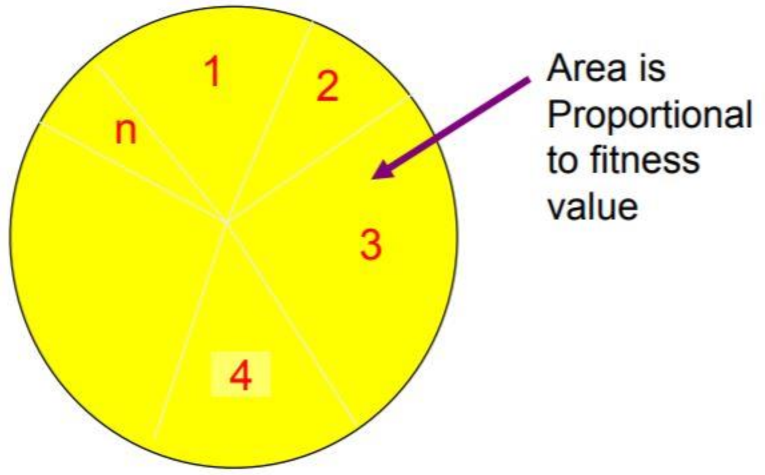
$s_5 = 1110111101 \quad f(s_5) = 8$

$s_6 = 0100110000 \quad f(s_6) = 3$

Step 1: Selection

We randomly (using a biased coin) select a subset of the individuals based on their fitness:

Individual *i* will have a probability to be chosen $\frac{f(i)}{\sum_i f(i)}$



Selected set

Suppose that, after performing selection, we get the following population:

s_1` = 1111010101 (s_1)

s_2` = 1110110101 (s_3)

s_3` = 1110111101 (s_5)

s_4` = 0111000101 (s_2)

s_5` = 0100010011 (s_4)

s_6` = 1110111101 (s_5)

Step 2: crossover

- Next we mate strings for crossover. For each couple we first decide (using some pre-defined probability, for instance 0.6) whether to actually perform the crossover or not
- If we decide to actually perform crossover, we randomly extract the crossover points, for instance 2 and 5

Crossover result

Before crossover:

s_1` = 1111010101 $s_2$` = 1110110101

After crossover:

s_1`` = 1110110101 $s_2$`` = 1111010101

Step 3: mutations

The final step is to apply random mutations: for each bit that we are to copy to the new population we allow a small probability of error (for instance 0.1)

Initial strings	After mutating
s_1'' = 1110110101	s_1''' = 1110100101
s_2'' = 1111010101	s_2''' = 1111110100
s_3'' = 1110111101	s_3''' = 1110101111
s_4'' = 0111000101	s_4''' = 0111000101
s_5'' = 0100011101	s_5''' = 0100011101
s_6'' = 1110110011	s_6''' = 1110110001

And now, iterate …

In one generation, the total population fitness changed from 34 to 37, thus improved by ~9%

At this point, we go through the same process all over again, until a stopping criterion is met