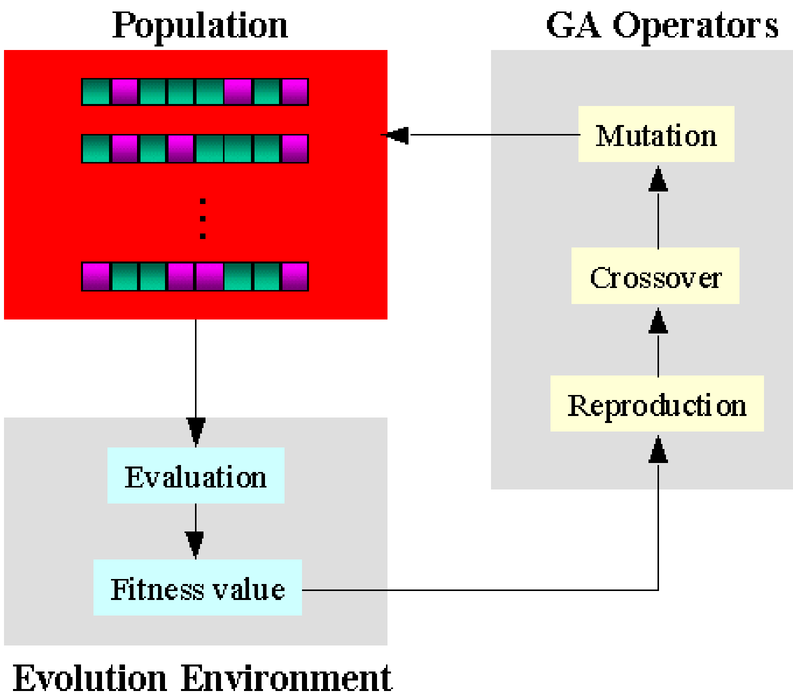
# Lecture 12
# Artificial Intelligence

Khola Naseem
khola.naseem@uet.edu.pk




Genetic Algorithm Evolution Flow

# Genetic Algorithm

- Genetic algorithms are defined as a type of computational optimization technique inspired by the principles of natural selection and genetics
- They are used to solve complex problems by mimicking the process of evolution to improve a population of potential solutions iteratively.
- Charles Darwinian Evolution – 1859
- **Theory of natural selection**
  - It proposes that the plants and animals that exist today are the result of millions of years of adaptation to the demands of the environment
- Over time, the entire population of the ecosystem is said to evolve to contain organisms that, on average, are fitter for environment than those of previous generations of the population

# Genetic Algorithm

- Genetic algorithms

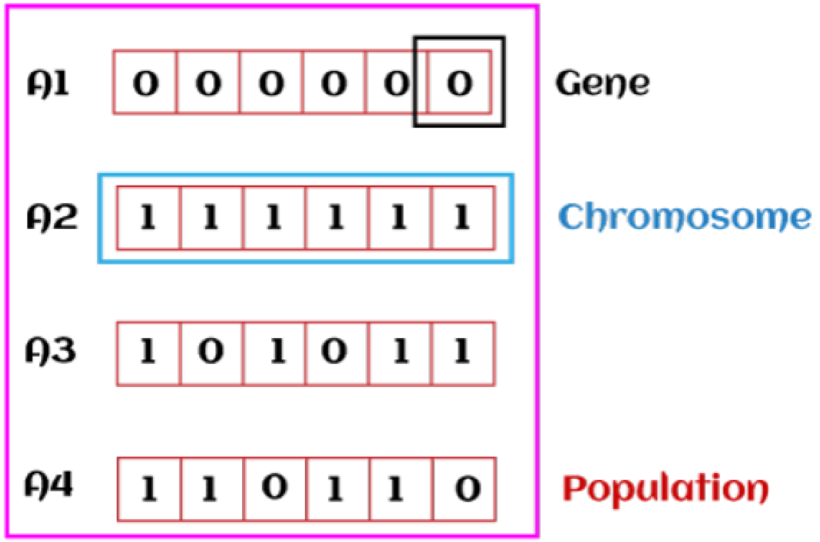
**Charles Darwin** – Evolution by *Descent with Modification* (1859)

Long-necked giraffes are randomly born and have more offspring due to their competitive advantage

- Focus on optimization

# Genetic Algorithm

- The concepts of GAs are directly derived from natural evolution.
- In the early 70's John Holland introduced this concept (Holland, 1975).
- GAs emulate ideas from genetics and natural selection and can search potentially large spaces
- Based on: survival of the most fittest individual
- Two key steps: reproduction, survive
- Try to simulate life.
  - Individual = solution
  - Environment = problem



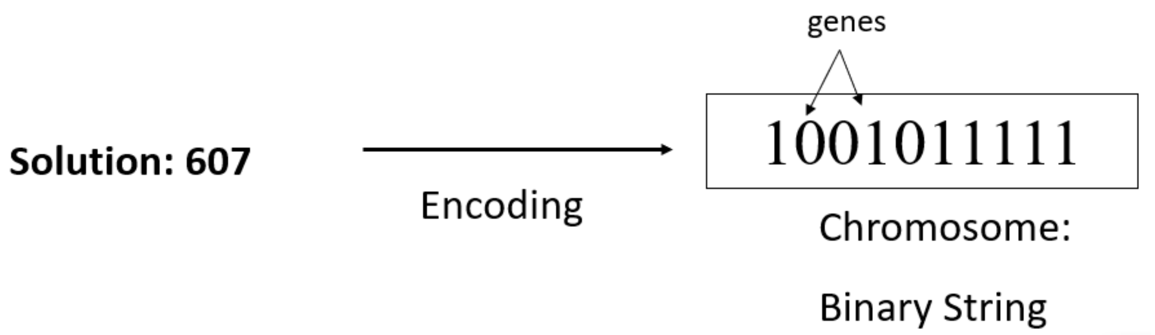

Credit: Khola Naseem

# Genetic Algorithm

- Before we can apply Genetic Algorithm to a problem, we need to answer:
  - How can an individual be represented?
  - What is the fitness function?
  - How are individuals selected?
  - How do individuals reproduce?

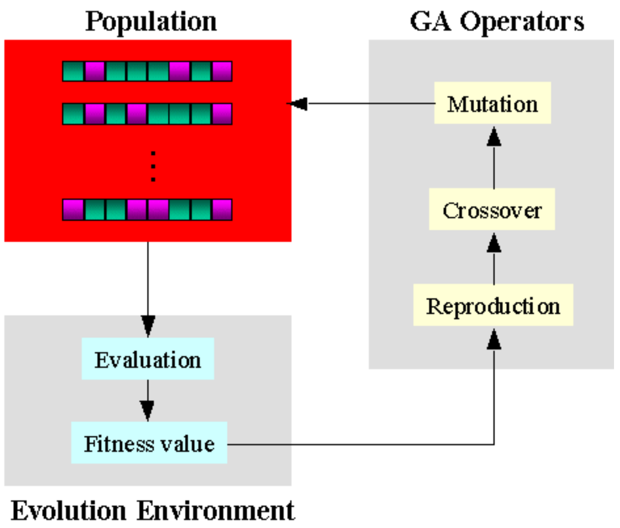
# Representation of states (solutions)

- State as sequence of strings
- Each state or individual is represented as a string over a finite alphabet {0,1}. It is also called chromosome which contains genes.



# Reproduction: building new states

- After representing States (solutions)
- Build a population of random solutions
- Let them reproduce using genetic operators
  - At each generation: apply ”survival of the fittest”
  - Hopefully better and better solutions evolve over time
  - The best solutions are more likely to survive and more likely to produce even better solutions




Genetic Algorithm Evolution Flow

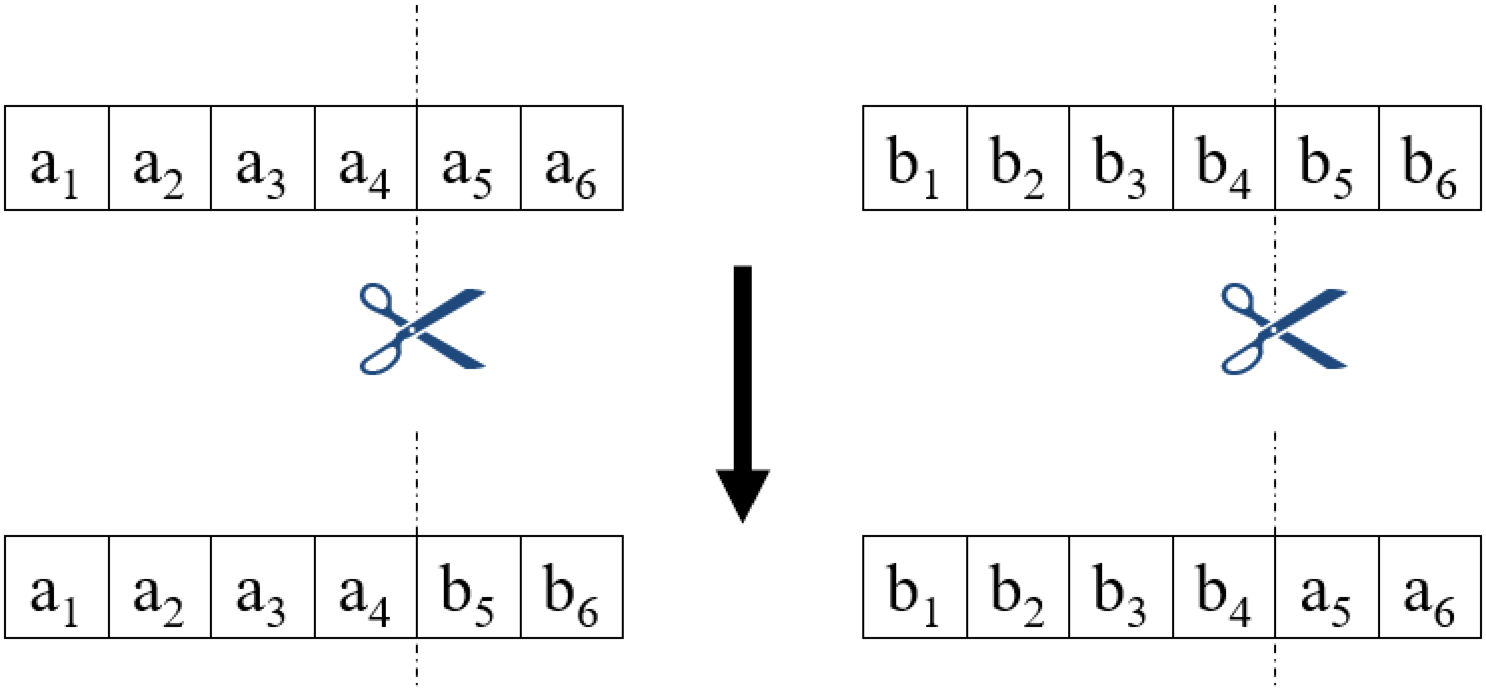
Credit: Khola Naseem

# Genetic Operators

- Crossover
- Mutation
- These operators mimic what happens to our genetic material when we reproduce

# Genetic Operators

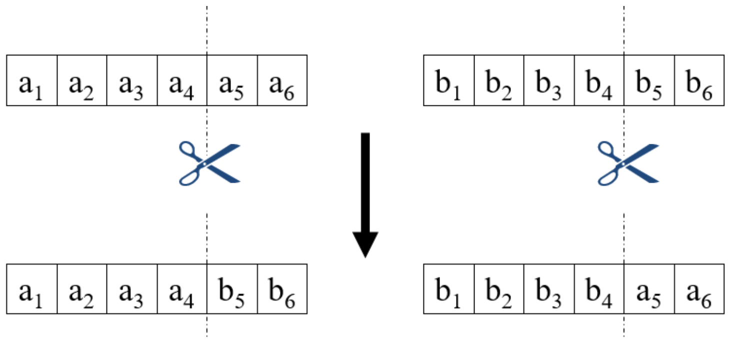
- Crossover
  - Cut two solutions at a random point and switch the respective parts
  - Typically a value of 0.7 for crossover probability gives good results.



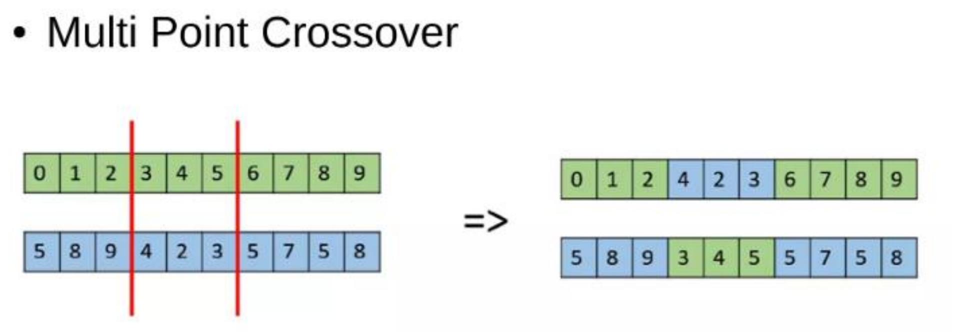

Credit: Khola Naseem

# Genetic Operators

- Crossover
  - Single point



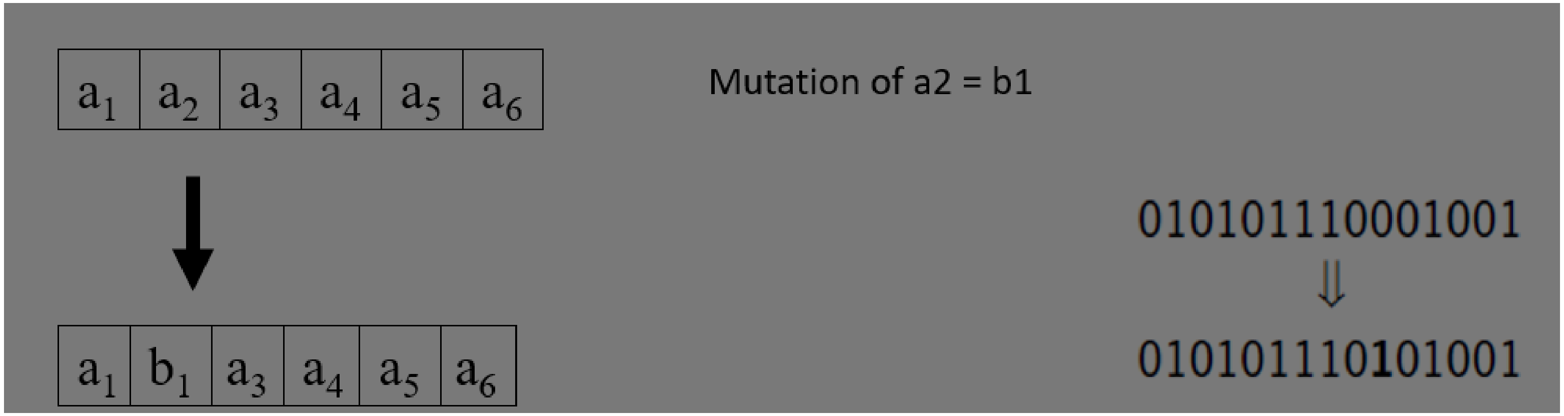

  - Multi point:




  - Uniform point

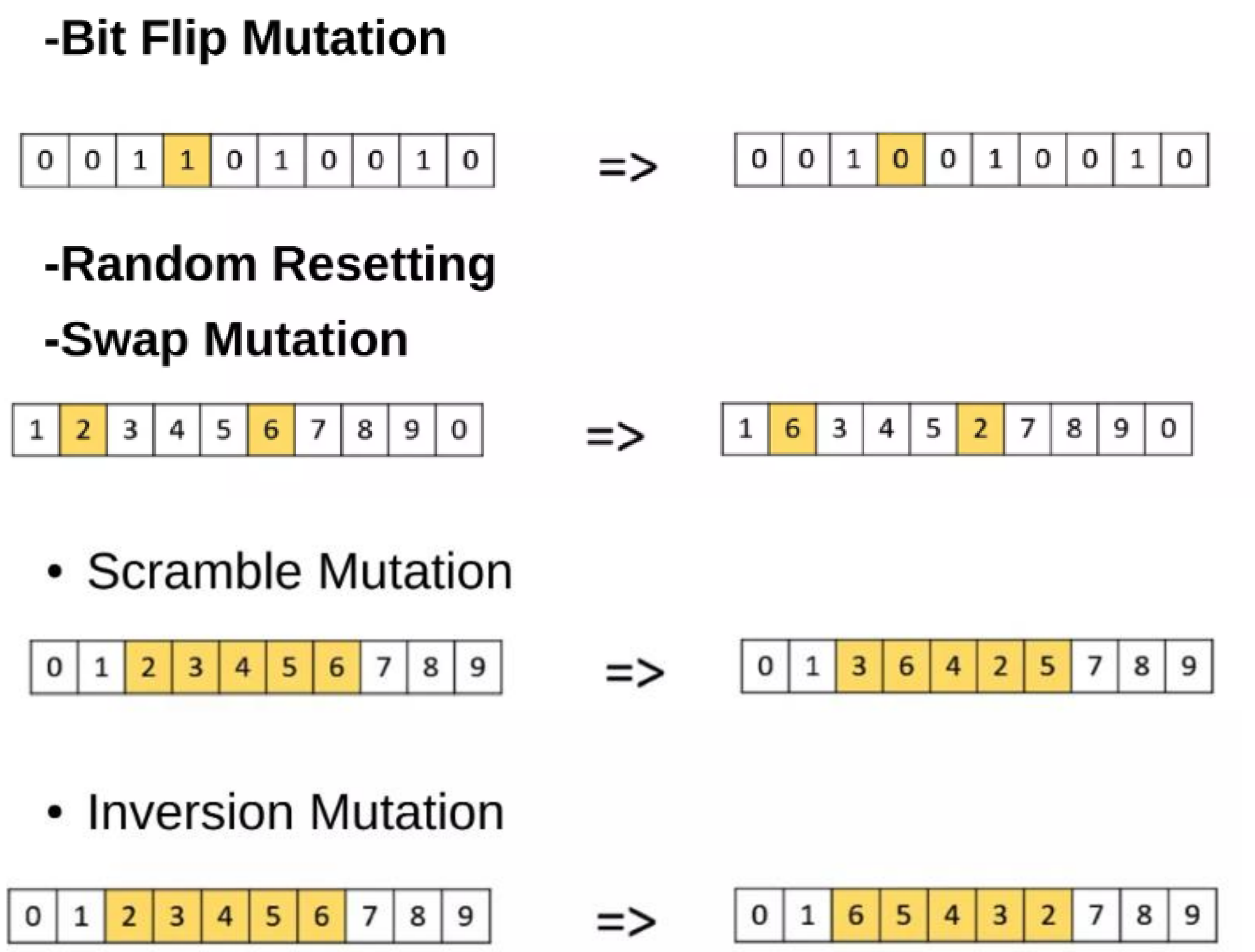
# Genetic Operators:Mutation operator

- Randomly change one bit in the solution
- Mutation is a unary operator (i.e., applied to just one argument—a single gene)
- Occasional mutation makes the method much less sensitive to the original population and also allows ”new” solutions to emerge



# Genetic Operators:Mutation operator

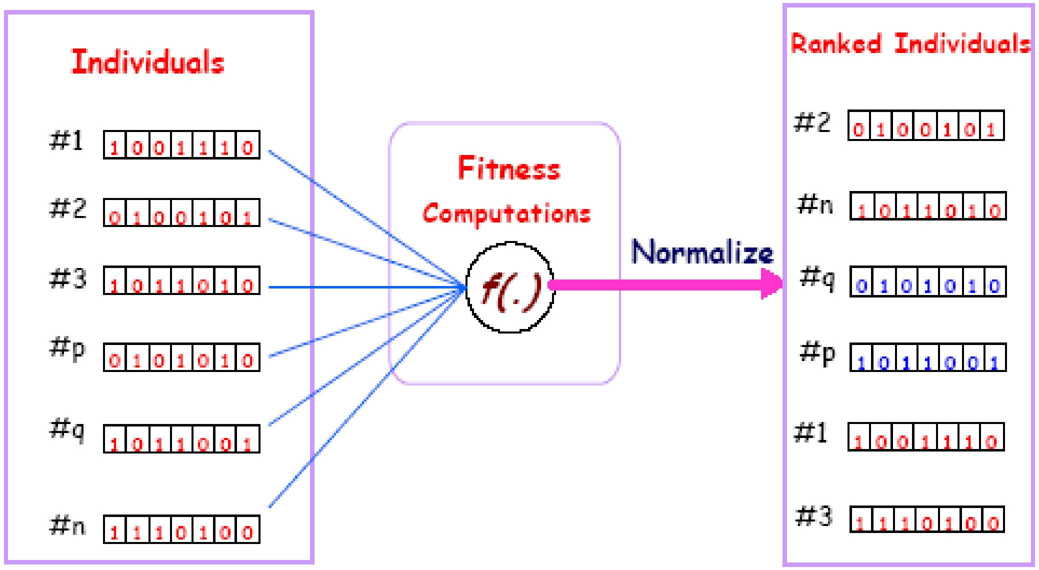
- Different types




Credit: Khola Naseem

# Evaluation and Selection

- We then see how good the solutions are, using an evaluation function (recall f(n) in informed search)
- Often it is a heuristic, especially if it is computationally expensive to do a complete evaluation
- The final population can then be evaluated more deeply to decide on the best solution




Credit: Khola Naseem

# Survival of the Fittest

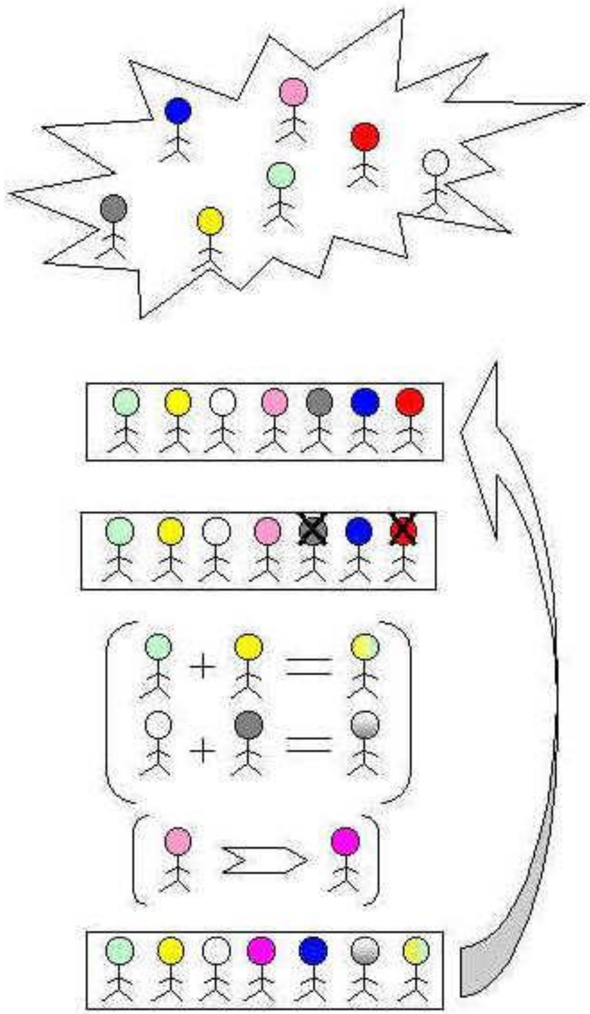
- Select the surviving population
- Likelihood of survival is related in some way to your score on the fitness function
  - The most common technique is roulette wheel selection
  - In roulette wheel selection, individuals are given a probability of being selected that is directly proportionate to their fitness.
- Note we always keep the best solution so far
- Remember: Its local search

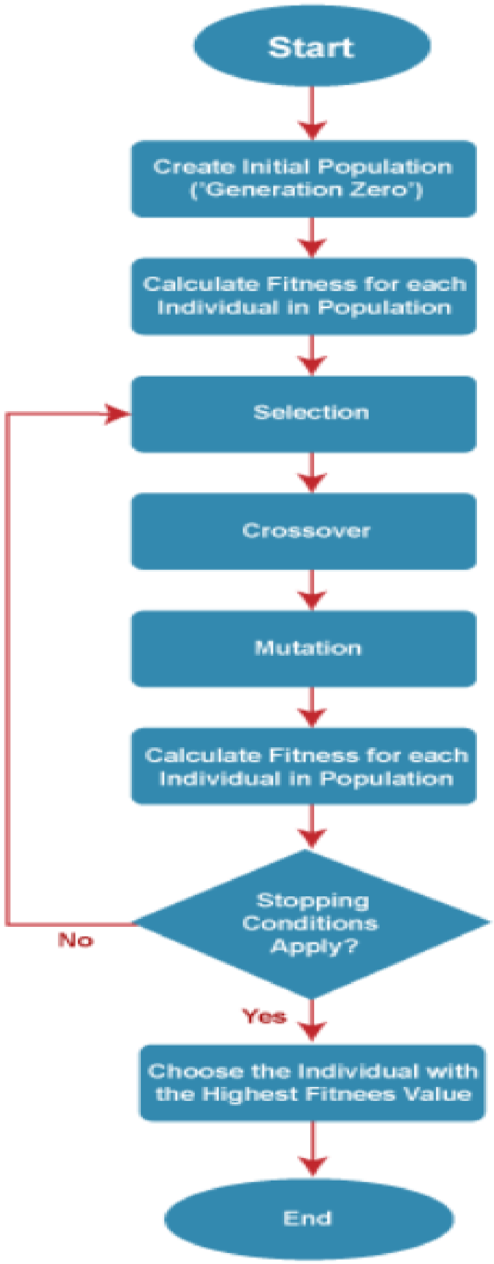
# Process:

- The most common type of genetic algorithm works like this:
- a population is created with a group of individuals created randomly.
- The individuals in the population are then evaluated.
- The evaluation function is provided by the programmer and gives the individuals a score based on how well they perform at the given task.
- Two individuals are then selected based on their fitness, the higher the fitness, the higher the chance of being selected.
- These individuals then "reproduce" to create one or more offspring, after which the offspring are mutated randomly.
- This continues until a suitable solution has been found or a certain number of generations have passed, depending on the needs of the programmer.

# Process:

- Genetic Algorithm





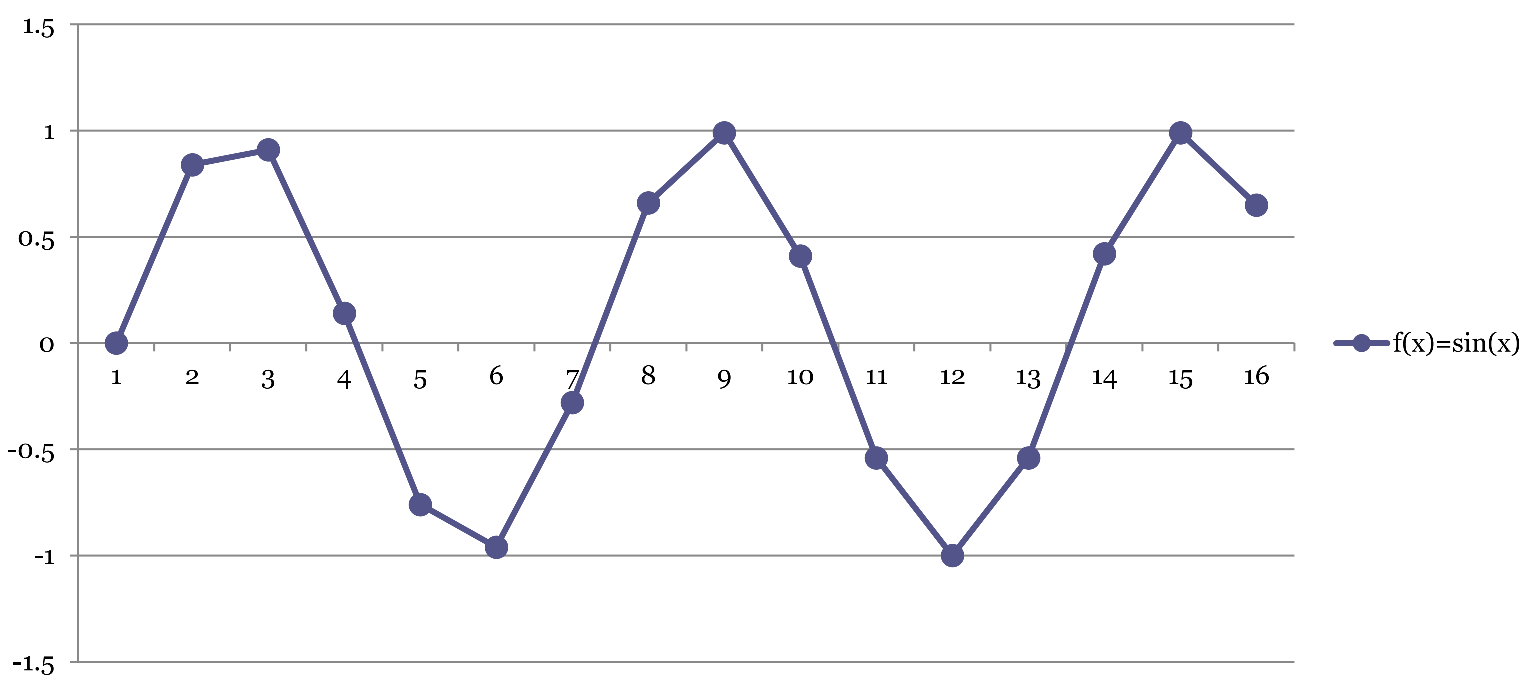

Credit: Khola Naseem

# Termination:

- This generational process is repeated until a termination condition has been reached.
- Common terminating conditions are:
  - A solution is found that satisfies minimum criteria
  - Fixed number of generations reached
  - Allocated budget (computation time/money) reached
  - The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
  - Manual inspection
  - Any Combinations of the above

# Fitness Function (Optimization):

- Fitness Function for a mathematic function
- Ex. attempt to maximize the function:
  - $f(x) = \sin(x)$ in range $0 \leq x \leq 15$




Credit: Khola Naseem

# Fitness Function (Optimization):

- Using population size of 4 chromosomes
- First Generation: Generate a random population:

| | | | |
|---|---|---|---|
| c1 = 1001 | (9) | c3 = 1010 | (10) |
| c2 = 0011 | (3) | c4 = 0101 | (5) |

- To calculate fitness of a chromosome, we calculate $f(x)$ for its decimal value
- Assign fitness as a numeric value from 0 to 100
  - 0 is the least fit
  - 100 is the most fit.

# Fitness Function (Optimization):

**Fitness of $x$, $f'(x)$:**

$$f'(x) = 50(f(x) + 1)$$
$$f'(x) = 50(\sin(x) + 1)$$

- 1 generation

**Table 14.1 Generation 1**

| Chromosome | Genes | Integer value | $f(x)$ | Fitness $f'(x)$ | Fitness ratio |
|---|---|---|---|---|---|
| c1 | 1001 | 9 | 0.41 | 70.61 | 46.3% |
| c2 | 0011 | 3 | 0.14 | 57.06 | 37.4% |
| c3 | 1010 | 10 | −0.54 | 22.80 | 14.9% |
| c4 | 0101 | 5 | −0.96 | 2.05 | 1.34% |

# Fitness Function (Optimization):

- The range of real numbers from 0 to 100 is divided up between the chromosomes proportionally to each chromosome's fitness.
- In first generation:
- c1 has 46.3% of the range (i.e., from 0 to 46.3)
- c2 37.4% of the range (i.e., from 46.4 to 83.7)

## 1 generation

**Table 14.1 Generation 1**

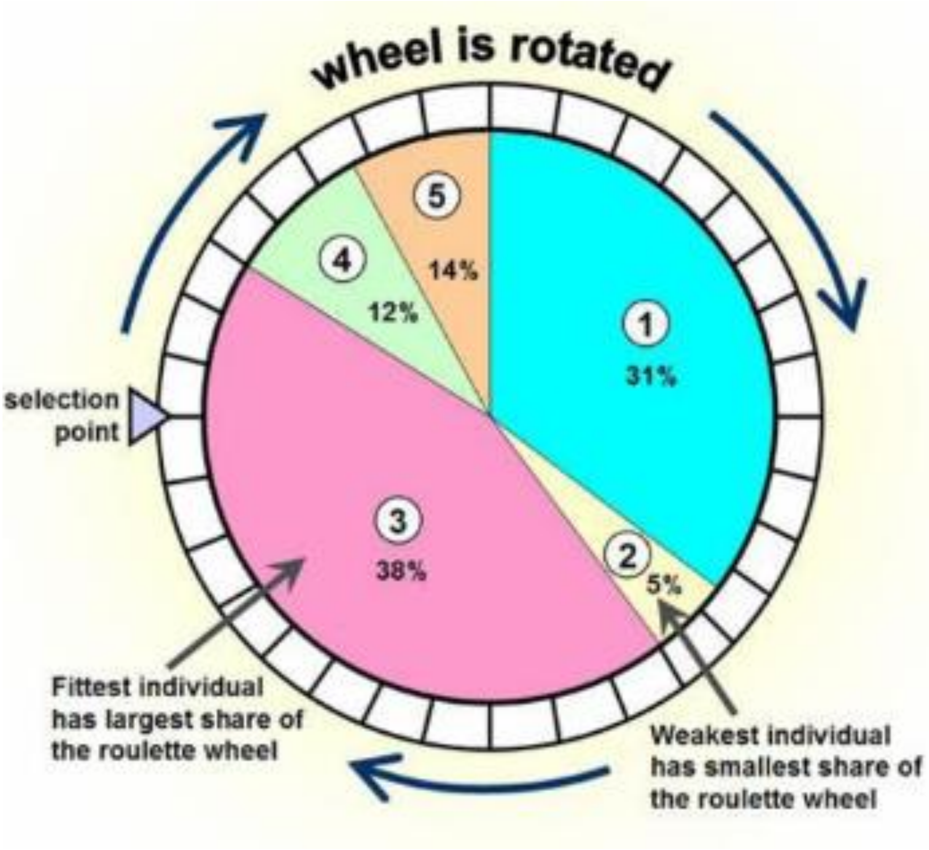
| Chromosome | Genes | Integer value | $f(x)$ | Fitness $f'(x)$ | Fitness ratio |
|---|---|---|---|---|---|
| c1 | 1001 | 9 | 0.41 | 70.61 | 46.3% |
| c2 | 0011 | 3 | 0.14 | 57.06 | 37.4% |
| c3 | 1010 | 10 | −0.54 | 22.80 | 14.9% |
| c4 | 0101 | 5 | −0.96 | 2.05 | 1.34% |

# Fitness Function (Optimization):

- Fitness ratio of c1:
  - $70.61/(70.61+57.06+22.8+2.05)\times100=46.29$
- Fitness ratio of c2:
  - $57.06/(70.61+57.06+22.8+2.05)\times100=37.4$

# Idea behind Roulette Selection

- Generate 4 random number



# Next Generation

- Need 4 random numbers for next generation
- Assume that:
  - First random number is 56.7 (c2 is chosen)
  - Second random number is 38.2 (c1 is chosen)
  - Third random number is 20 (c1 is chosen)
  - Fourth random number is 85 (c3 is chosen)

# Next Generation

- Combine first two to produce two new offspring:
  - Crossover point

| | | | | |
|---|---|---|---|---|
| c1 = 1001 | (9) | | c2 = 0011 | (3) |
| **c5 = 1011** | **(11)** | ↓ | **c6 = 0001** | **(1)** |

- Combine last two to produce two new offspring:
  - Crossover point

| | | | | |
|---|---|---|---|---|
| c1 = 1001 | (9) | | c3 = 1010 | (10) |
| **c8 = 1011** | **(11)** | ↓ | **c7 = 1000** | **(8)** |

# Second generation: c5 to c8

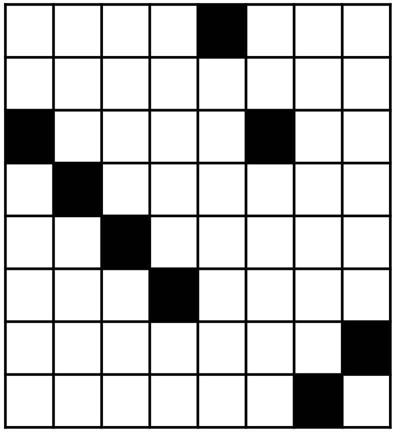
- c4 did not have a chance to reproduce (its genes will be lost)
- Fittest chromosome in the first generation (c1), able to reproduce twice
- Passing on its highly fit genes to all members of the next generation

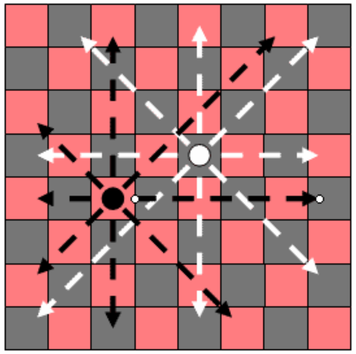
**Table 14.2 Generation 2**

| Chromosome | Genes | Integer value | $f(x)$ | Fitness $f'(x)$ | Fitness ratio |
|---|---|---|---|---|---|
| c5 | 1011 | 11 | −1 | 0 | 0% |
| c6 | 0001 | 1 | 0.84 | 92.07 | 48.1% |
| c7 | 1000 | 8 | 0.99 | 99.47 | 51.9% |
| c8 | 1011 | 11 | −1 | 0 | 0% |

# 8-Queens Problem

- State = position of 8 queens each in a column
  - 3 4 5 6 1 3 8 7
- Start with k randomly generated states (population)
- Evaluation function (fitness function).
- Higher values for better states.
- Produce the next generation of states
- Random selection
  - Crossover
  - Random mutation

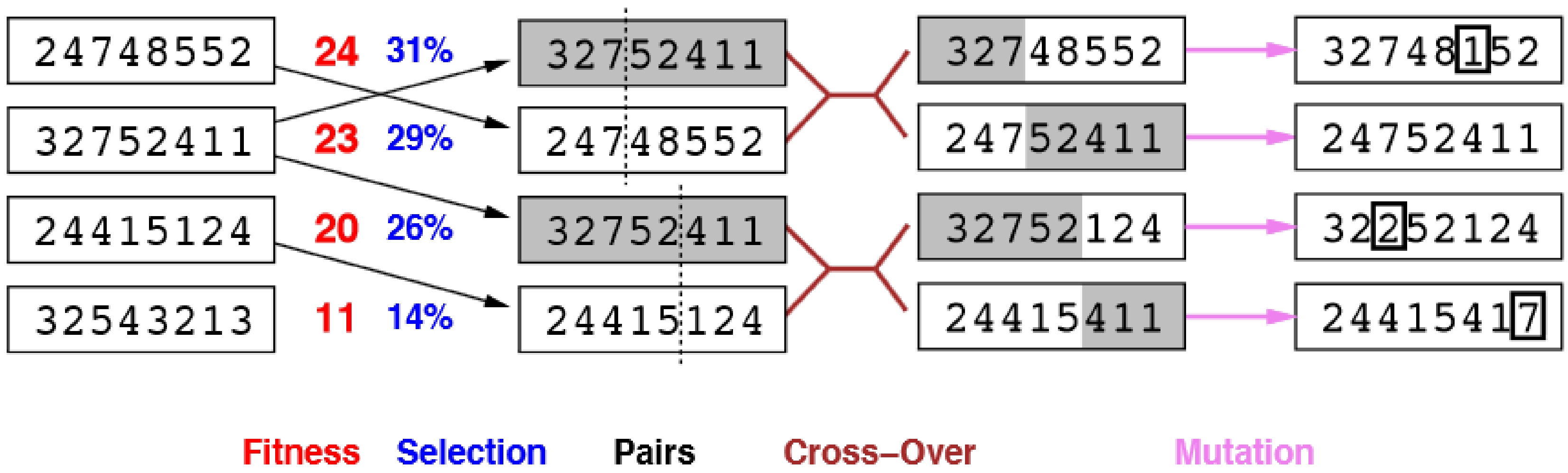




Credit: Khola Naseem

# 8-Queens Problem

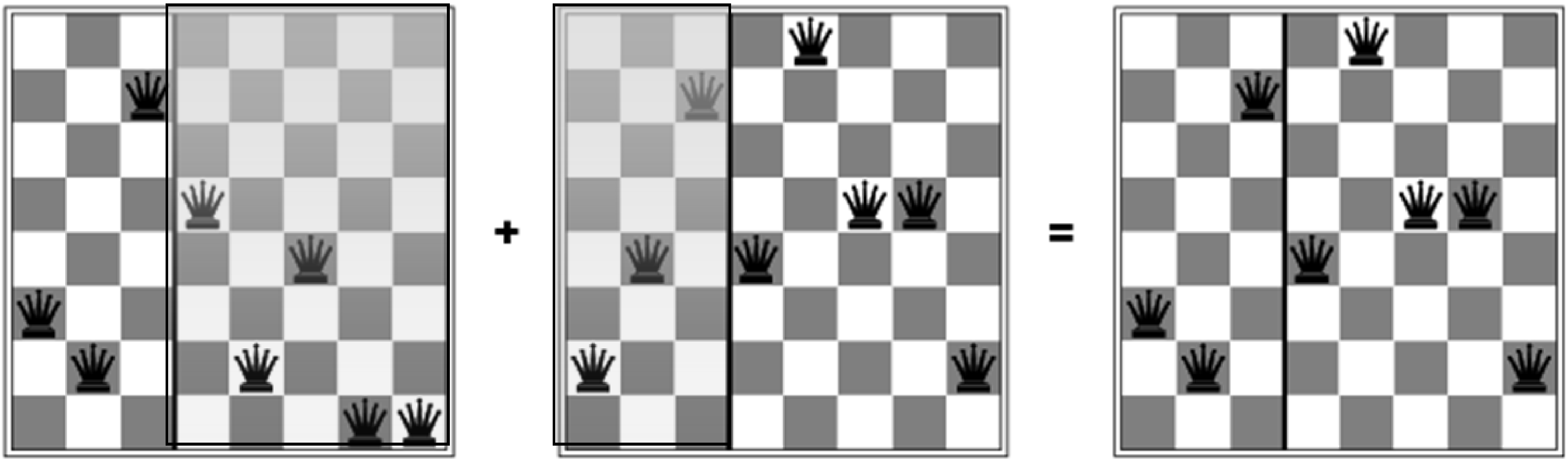
- State = position of 8 queens each in a column




Note: $24 / (24+23+20+11) = 31\%$

# 8-Queens Problem

- Effect of Crossover on 8-Queens



Has the effect of "jumping" to a completely different new part of the search space

# Advantages:

- They are Robust
- Provide optimization over large space state.
- Unlike traditional AI, they do not break on slight change in input or presence of noise

GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP
QRSTUVWXYZ 1234567890, .-;:_!"#%&/()=?@${[]}'''



# Example:

- Example

- Input:

```
GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP
QRSTUVWXYZ 1234567890, .-;:_!"#%&/()=?@${[]}'‘’

# Target string to be generated
TARGET = "I love GeeksforGeeks"
```

```
Generation: 1     String: tO{"-?=jH[k8=B4]Oe@}     Fitness: 18
Generation: 2     String: tO{"-?=jH[k8=B4]Oe@}     Fitness: 18
Generation: 3     String: .#lRWf9k_Ifslw #O$k_     Fitness: 17
Generation: 4     String: .-1Rq?9mHqk3Wo]3rek_     Fitness: 16
Generation: 5     String: .-1Rq?9mHqk3Wo]3rek_     Fitness: 16
Generation: 6     String: A#ldW) #lIkslw cVek)     Fitness: 14
Generation: 7     String: A#ldW) #lIkslw cVek)     Fitness: 14
Generation: 8     String: (, o x _x%Rs=, 6Peek3    Fitness: 13
                          .
                          .
                          .
Generation: 29    String: I lope Geeks#o, Geeks    Fitness: 3
Generation: 30    String: I loMe GeeksfoBGeeks     Fitness: 2
Generation: 31    String: I love Geeksfo0Geeks     Fitness: 1
Generation: 32    String: I love Geeksfo0Geeks     Fitness: 1
Generation: 33    String: I love Geeksfo0Geeks     Fitness: 1
Generation: 34    String: I love GeeksforGeeks     Fitness: 0
```

- https://www.geeksforgeeks.org/genetic-algorithms/

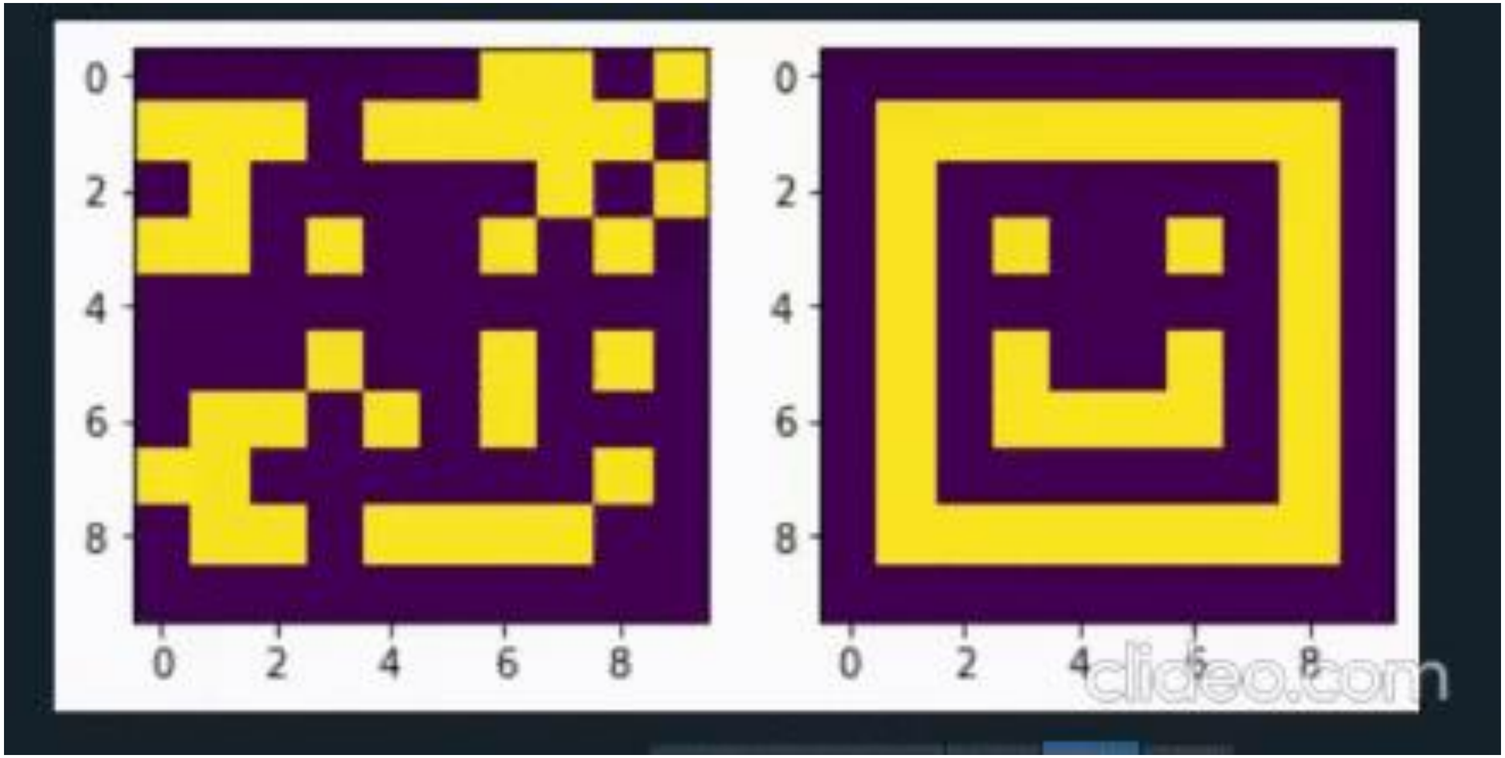
Credit: Khola Naseem

```
GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP
QRSTUVWXYZ 1234567890, .-;:_!"#%&/()=?@${[]}'''
```



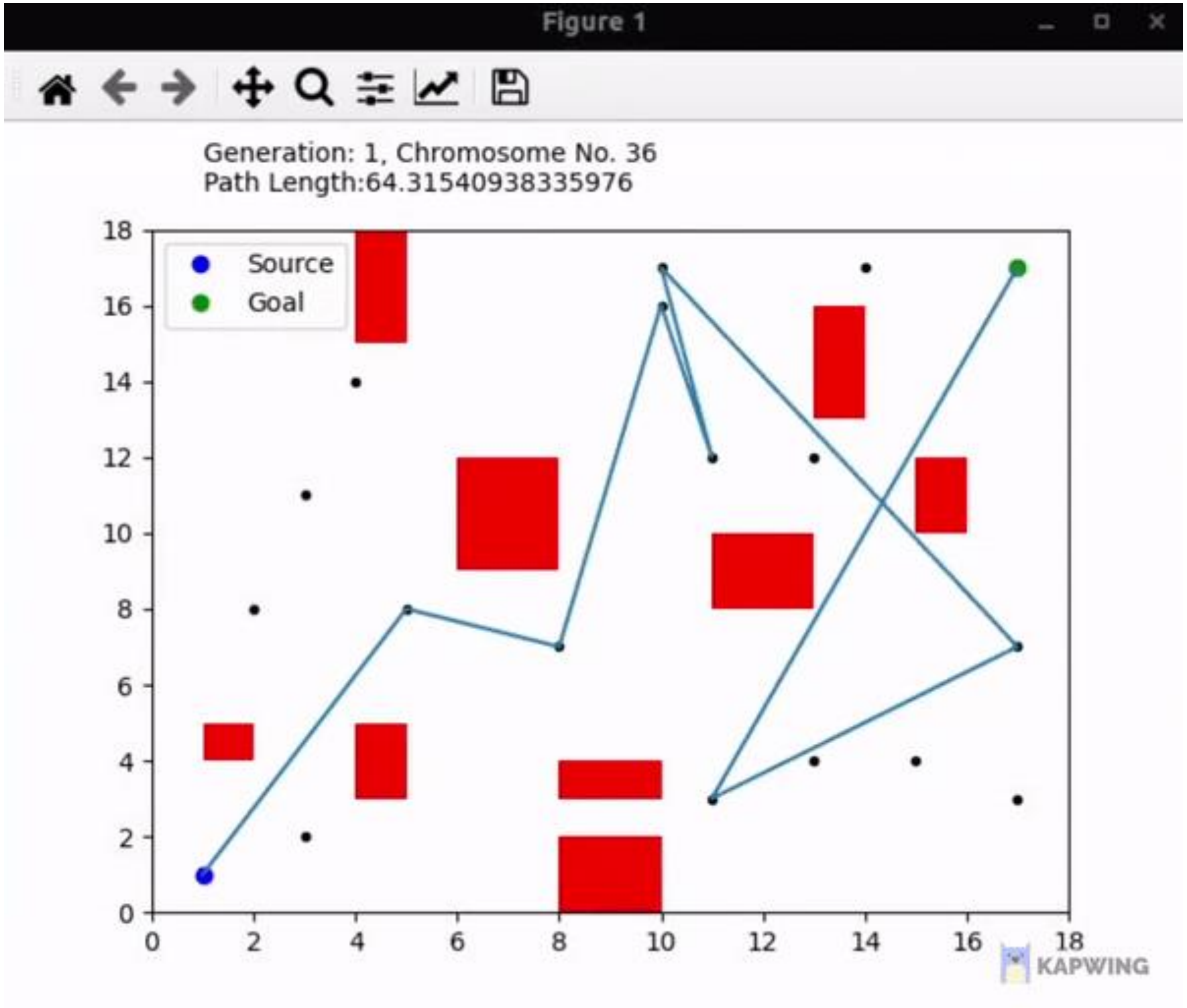
# Example:

- Example



# Example:

- Example




Credit: Khola Naseem

```
GENES = '''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP
QRSTUVWXYZ 1234567890, .-;:_!"#%&/()=?@${[]}'''
```



# Example:

- Example