Continuous Cellular Automata and Gene Evolution

IPHD Yang, YuanFu
IPHD Tom Li

Agenda

Introduction

- **Continuous cellular automata** (**CCA**) is a form of artificial life. It was derived from Conway's Game of Life by making everything smooth, continuous and generalized. These digital creatures show lifelike features like self-organization, self-repair, bilateral and radial symmetries, locomotive dynamics, and sometimes chaotic nature.
- In 2020, further extention of CCA led to more emergent phenomena, like interesting 3D/4D patterns, self-replication, pattern emission, self-boundary/individuality, aggregated patterns, polymorphism, intercommunicating colonies, etc.
- **Gene evolution** is the process by which a gene changes in structure and sequence over time. Genetic algorithm is a search heuristic method that imitates genetic evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

Research Directions

- Artificial Life
- Artificial Intelligence
- Theoretical Biology
- Computer Science
- Mathematics & Physics
- Digital Art

Method

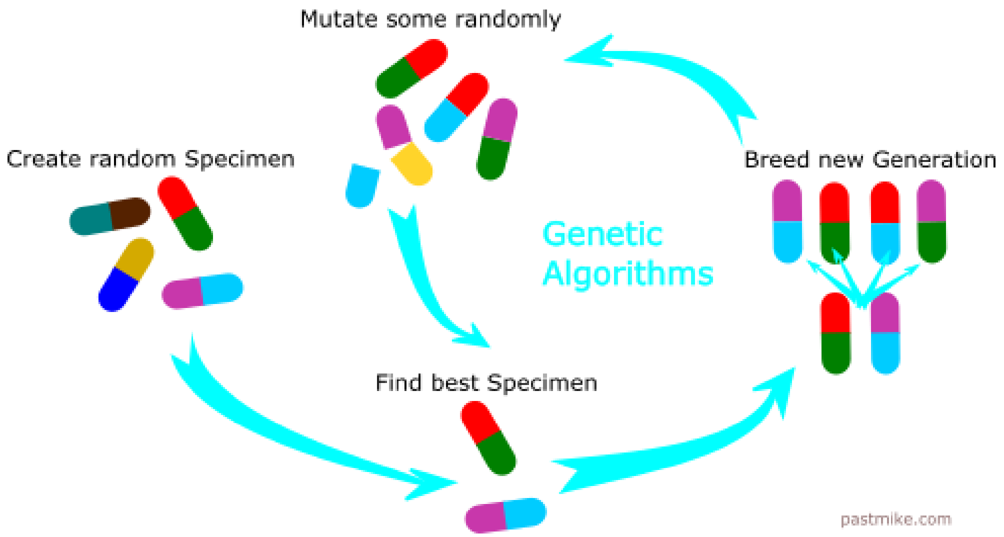
The process of Continuous Cellular Automata:

1. Take a 2D array (world A) of real values between 0 and 1, initialize with an initial pattern A0.
2. Every cell A0 interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:
3. Any live cell with fewer than two live neighbours dies, as if by underpopulation.
4. Any live cell with two or three live neighbours lives on to the next generation.
5. Any live cell with more than three live neighbours dies, as if by overpopulation.
6. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.
7. Repeat steps 2-7 for each time-step.

Method

Five phases are considered in a genetic algorithm:

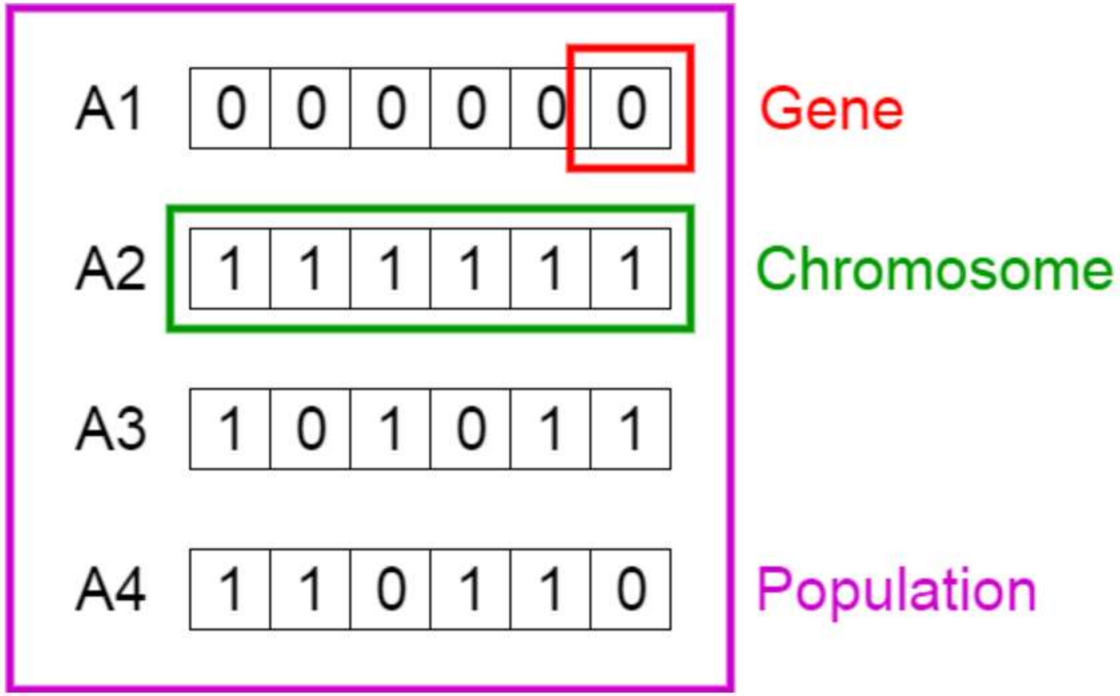
1. Initial population
2. Fitness function
3. Selection
4. Crossover
5. Mutation



Method

Five phases are considered in a genetic algorithm:

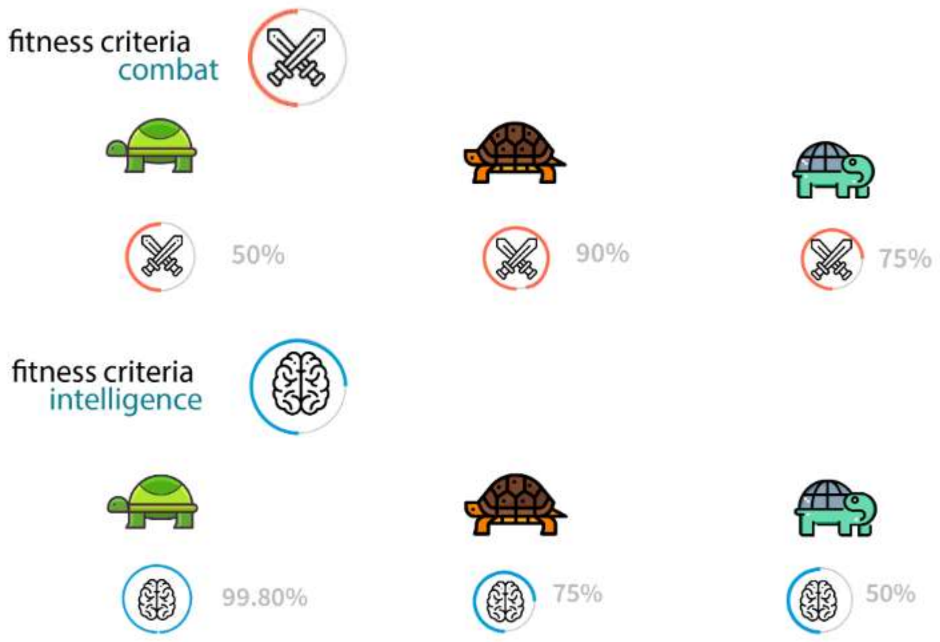
1. **Initial population**
2. Fitness function
3. Selection
4. Crossover
5. Mutation



Method

Five phases are considered in a genetic algorithm:

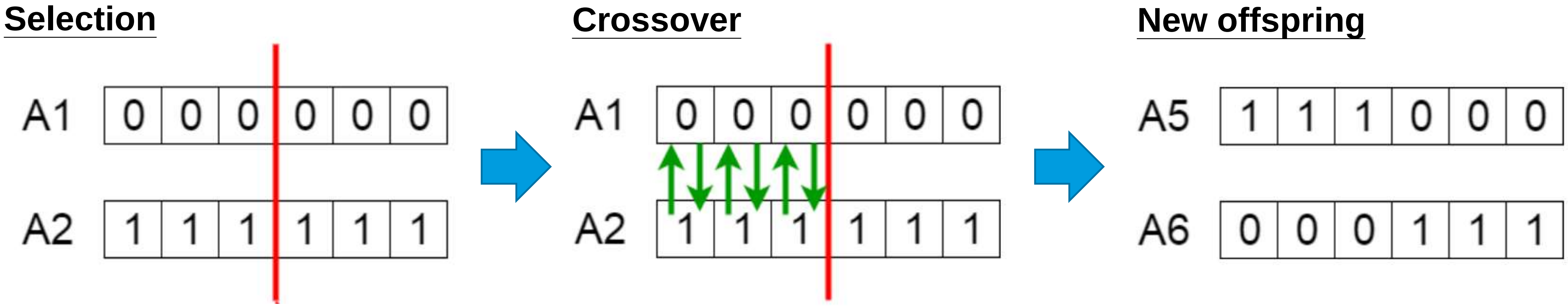
1. Initial population
2. **Fitness function**
3. Selection
4. Crossover
5. Mutation



Method

Five phases are considered in a genetic algorithm:

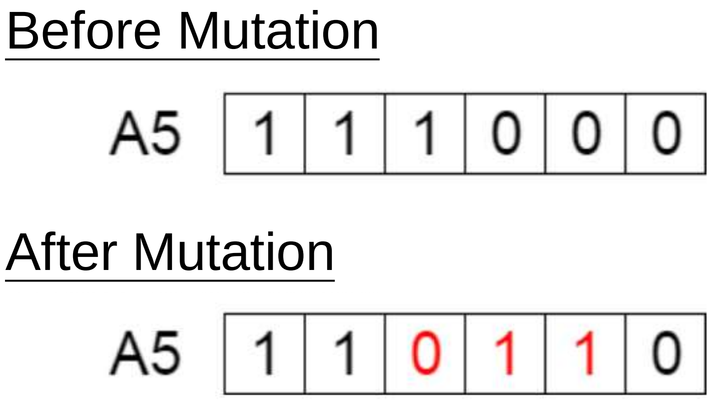
1. Initial population
2. Fitness function
3. **Selection**
4. **Crossover**
5. Mutation



Method

Five phases are considered in a genetic algorithm:

1. Initial population
2. Fitness function
3. Selection
4. Crossover
5. **Mutation**



Mutation: Before and After

Modeling & Result - Continuous Cellular Automata

Step-1: define the neighbours

```
def alive(x,i,j):
    if(i>=0):
        if(i<=n-1):
            if(j>=0):
                if(j<=m-1):
                    return x[i,j]
    if(i<0):
        return 0
    if(j<0):
        return 0
    if(i>n-1):
        return 0
    if(j>m-1):
        return 0
```

For Example: A0 = Array(3X5)

0	0	0	0	0	0	0
0						0
0		1	1	1		0
0						0
0						0
0	0	0	0	0	0	0

Modeling & Result - Continuous Cellular Automata

```python
def play(x,n,m):
    #x2=np.zeros((11,11))
    #n,m=x.shape[0],x.shape[1]
```

Step2

```python
    for i in range(n):
        for j in range(m):
            s = alive(x,i+1,j)+alive(x,i+1,j+1)+alive(x,i+1,j-1)
                +alive(x,i,j-1)+alive(x,i,j+1)
                +alive(x,i-1,j)+alive(x,i-1,j-1)+alive(x,i-1,j+1)
            s = int(s)
            x2[i,j] = s
```

Step3

```python
    for i in range(n):
        for j in range(m):
            if(x[i,j]==1):
                if(x2[i,j]!=2):
                    if(x2[i,j]!=3):
                        x[i,j]=0
```

Step4

```python
    for i in range(n):
        for j in range(m):
            if(x[i,j]==0):
                if(x2[i,j]==3):
                    x[i,j]=1
    return(x)
```

Step-2

0	0	0	0	0	0	0
0						0
0		1	1	1		0
0						0
0						0
0	0	0	0	0	0	0

0	0	0	0	0	0	0
0	1	2	3	2	1	0
0	1	1	2	1	1	0
0	1	2	3	2	1	0
0						0
0	0	0	0	0	0	0

Step-3

0	0	0	0	0	0	0
0						0
0		0	1	0		0
0						0
0						0
0	0	0	0	0	0	0

Stpe-4

0	0	0	0	0	0	0
0			1			0
0		0	1	0		0
0			1			0
0						0
0	0	0	0	0	0	0

Modeling & Result - Continuous Cellular Automata

- Results:

```
for i in range(10):
    print("Generation: ",i)
    print(x)
    x=play(x,n,m)
```

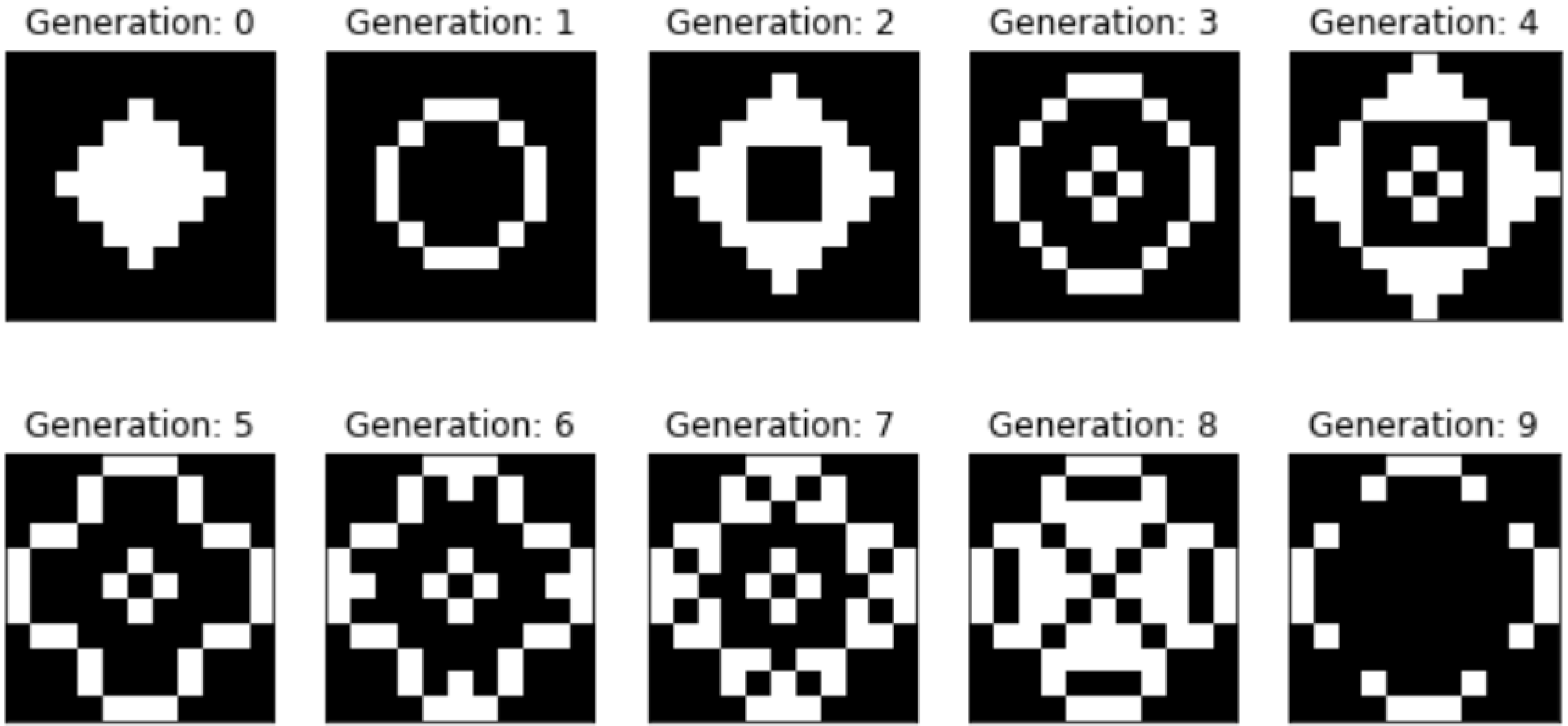
```
Generation:  0
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0.]
 [0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 0.]
 [0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
Generation:  1
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

```
Generation:  2
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0.]
 [0. 0. 1. 1. 0. 0. 0. 1. 1. 0. 0.]
 [0. 1. 1. 1. 0. 0. 0. 1. 1. 1. 0.]
 [0. 0. 1. 1. 0. 0. 0. 1. 1. 0. 0.]
 [0. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
Generation:  3
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0.]
 [0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

```
Generation:  4
[[0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]]
Generation:  5
[[0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1.]
 [1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1.]
 [1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0.]]
```

Modeling & Result - Continuous Cellular Automata

- Results:



Modeling & Result - Gene Evolution

Five phases are considered in a genetic algorithm:

1. **Initial population**
2. Fitness function
3. Selection
4. Crossover
5. Mutation

```
pop = np.random.randint(2, size=(POP_SIZE, DNA_SIZE))
pop
```

```
array([[0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [1, 1, 1, 0, 1, 0, 0, 0, 1, 0],
       [1, 1, 0, 1, 1, 1, 1, 1, 0, 0],
       [1, 0, 0, 1, 1, 1, 1, 1, 1, 0],
       [0, 0, 1, 0, 1, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
       [0, 0, 0, 0, 1, 1, 1, 0, 0, 1],
       [1, 0, 1, 0, 0, 1, 0, 1, 1, 0],
       [1, 0, 0, 0, 0, 1, 1, 0, 0, 1],
       [0, 1, 1, 0, 1, 1, 0, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 1, 0, 1, 0],
```

```
pop.shape
```

```
(100, 10)
```

Modeling & Result - Gene Evolution

Five phases are considered in a genetic algorithm:

1. Initial population
2. **Fitness function**
3. Selection
4. Crossover
5. Mutation

```
def F(x):
    return np.sin(10*x)*x + np.cos(2*x)*x
```

```
def translateDNA(pop):
    return pop.dot(2 ** np.arange(DNA_SIZE)[::-1]) / float(2**DNA_SIZE-1) * X_BOUND[1]
```

```
F_values = F(translateDNA(pop))
```

```
F_values.min(),F_values.max()
```

```
(-9.480744870638143, 6.421816483484863)
```

```
def get_fitness(pred):
    return pred + 1e-3 - np.min(pred)
```

```
fitness = get_fitness(F_values)
```

```
fitness.min(),fitness.max()
```

```
(0.0009999999999994458, 15.903561354123006)
```

Modeling & Result - Gene Evolution

Five phases are considered in a genetic algorithm:

1. Initial population
2. Fitness function
3. **Selection**
4. Crossover
5. Mutation

```
def select(pop, fitness):
    idx = np.random.choice(np.arange(POP_SIZE), size=POP_SIZE, replace=True,
                           p=fitness/fitness.sum())
    return pop[idx]
```

5.1 Convert the fitness value content of Array 100X1 (0~15) into probability p

5.2 According to 100 probability values p, from 0~99 (np.arange(POP_SIZE)), randomly select 100 (POP_SIZE=100), and then put it back (replace=True)

Modeling & Result - Gene Evolution

Five phases are considered in a genetic algorithm:

1. Initial population
2. Fitness function
3. Selection
4. **Crossover**
5. Mutation

```
CROSS_RATE = 0.8
def crossover(parent, pop):
    if np.random.rand() < CROSS_RATE:
        i_ = np.random.randint(0, POP_SIZE, size=1)
        cross_points = np.random.randint(0, 2, size=DNA_SIZE).astype(np.bool)
        parent[cross_points] = pop[i_, cross_points]
    return parent
```

Randomly select n from the 10 genes in the original DNA for exchange. Cross rate is 80%.

Modeling & Result - Gene Evolution

Five phases are considered in a genetic algorithm:

1. Initial population
2. Fitness function
3. Selection
4. Crossover
5. **Mutation**

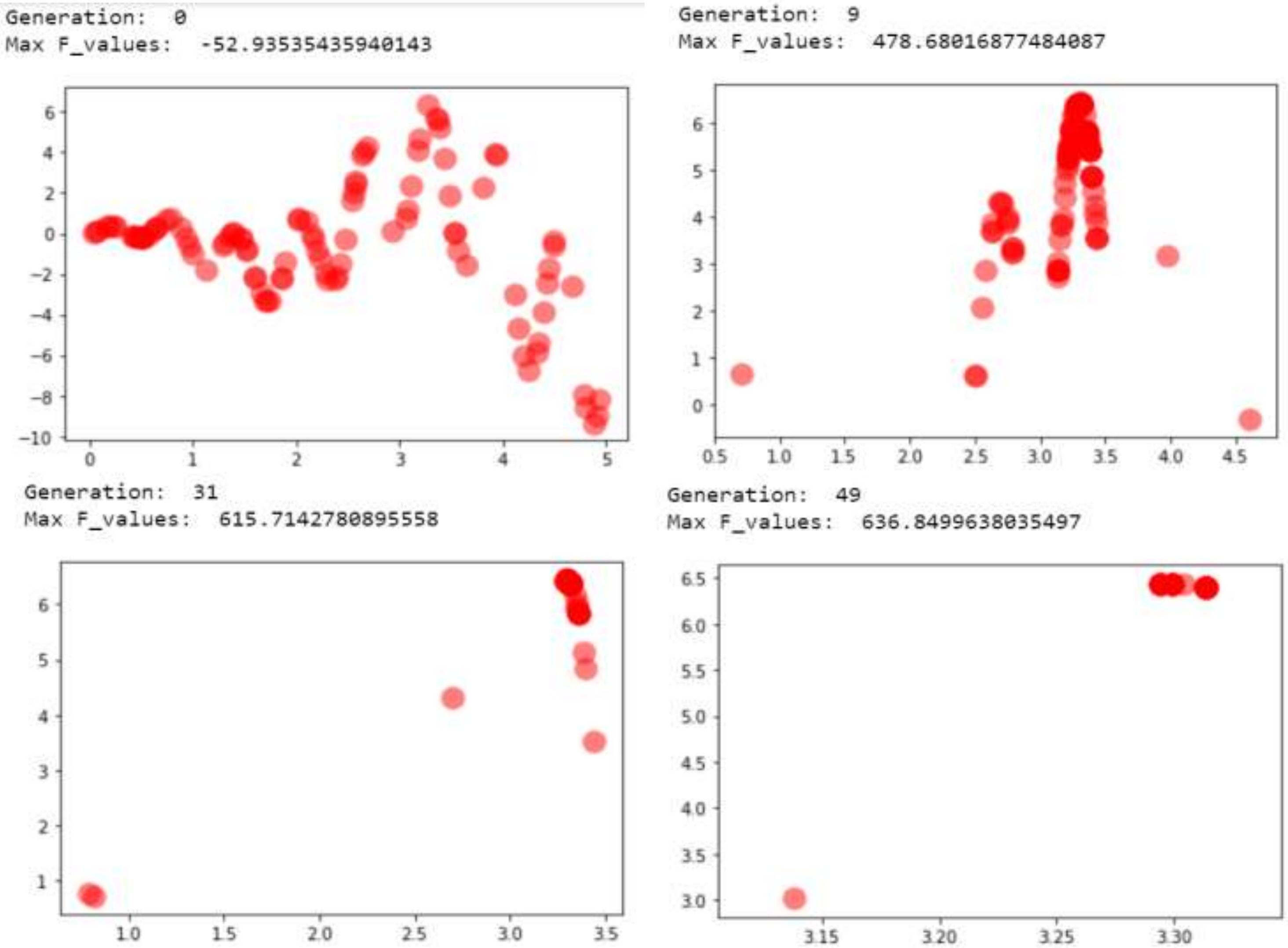
```
MUTATION_RATE = 0.003
def mutate(child):
    for point in range(DNA_SIZE):
        if np.random.rand() < MUTATION_RATE:
            child[point] = 1 if child[point] == 0 else 0
    return child
```

Randomly take offspring (probability: 0.003) for mutation, and exchange 0 and 1 if they are taken.

Modeling & Result - Gene Evolution

- Results:

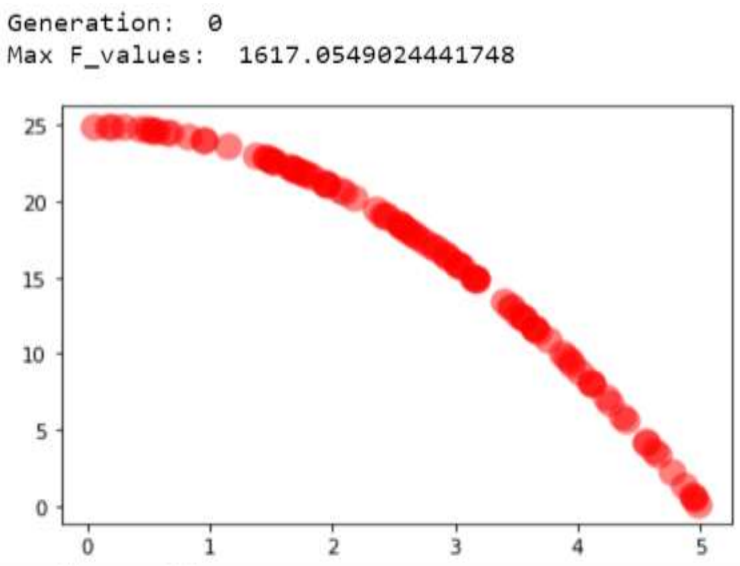
```python
def F(x):
    return np.sin(10*x)*x + np.cos(2*x)*x
```

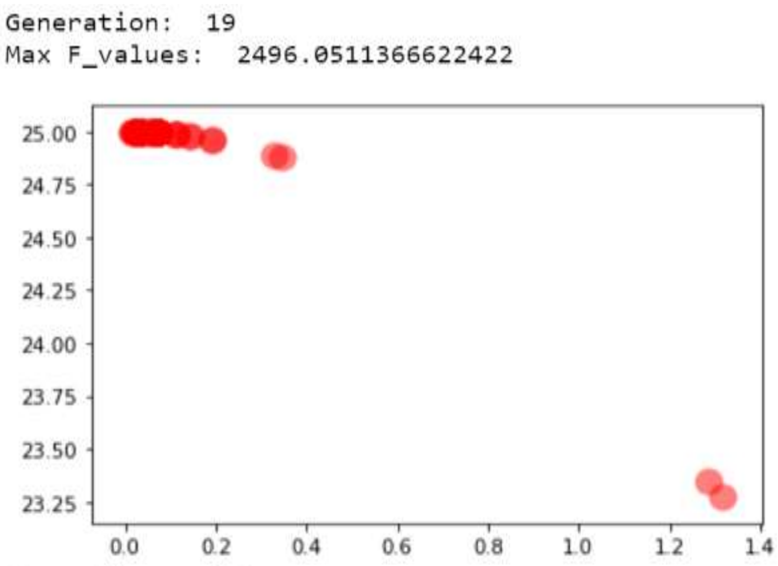


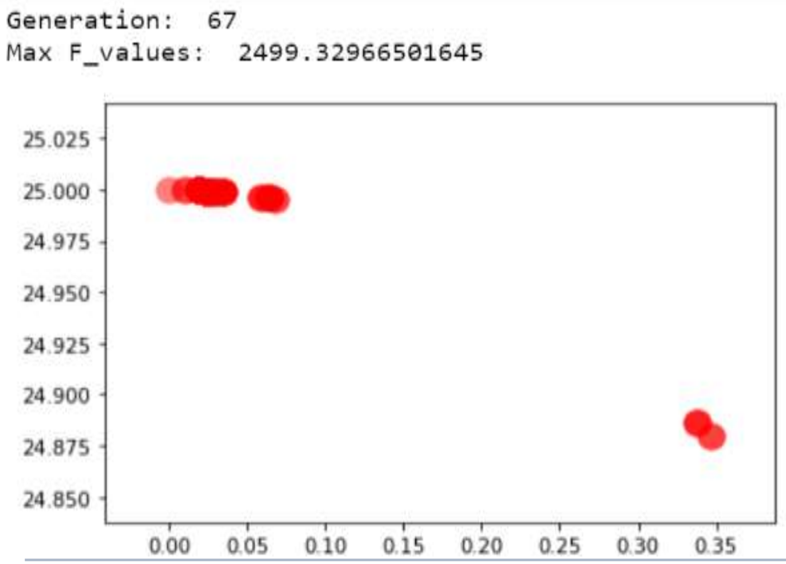
Modeling & Result - Gene Evolution

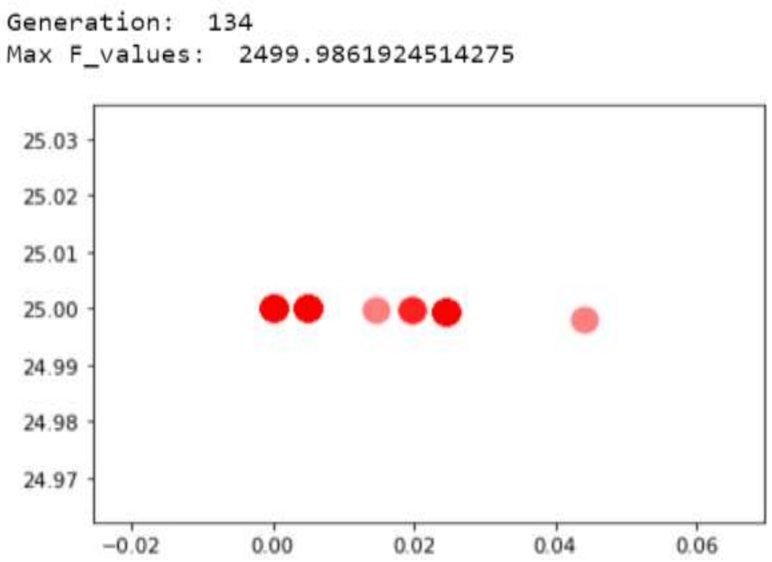
- Results:

```python
def F(x): return -x**2+25
```





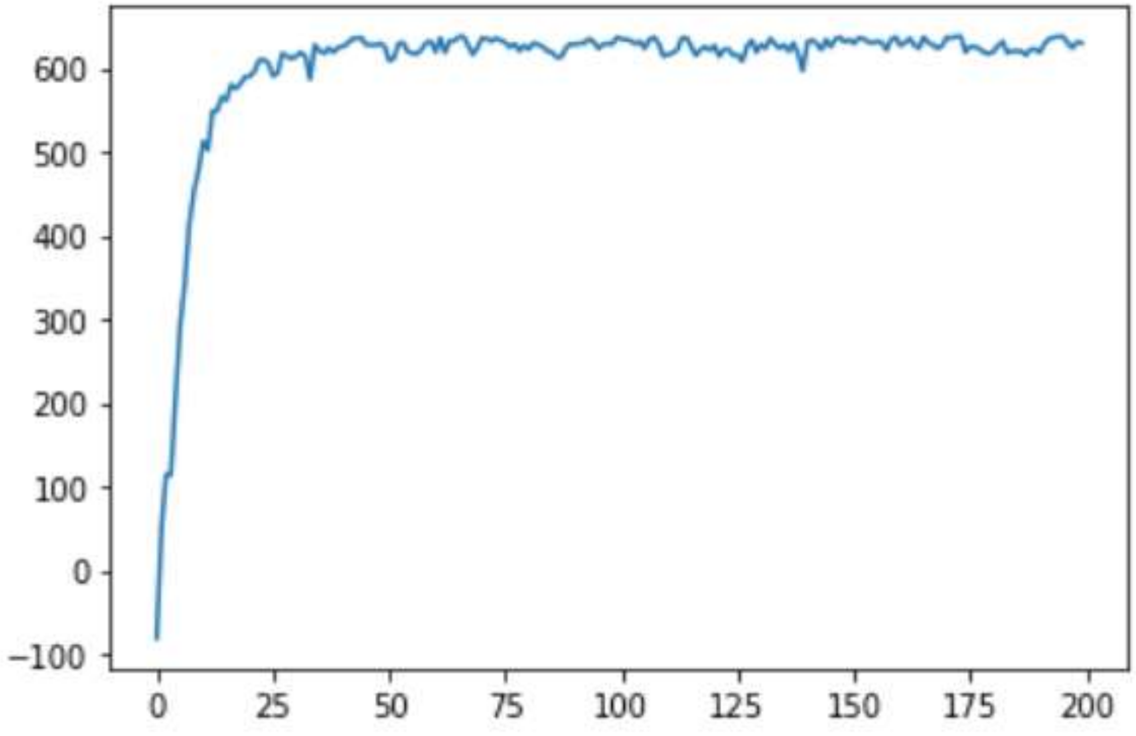




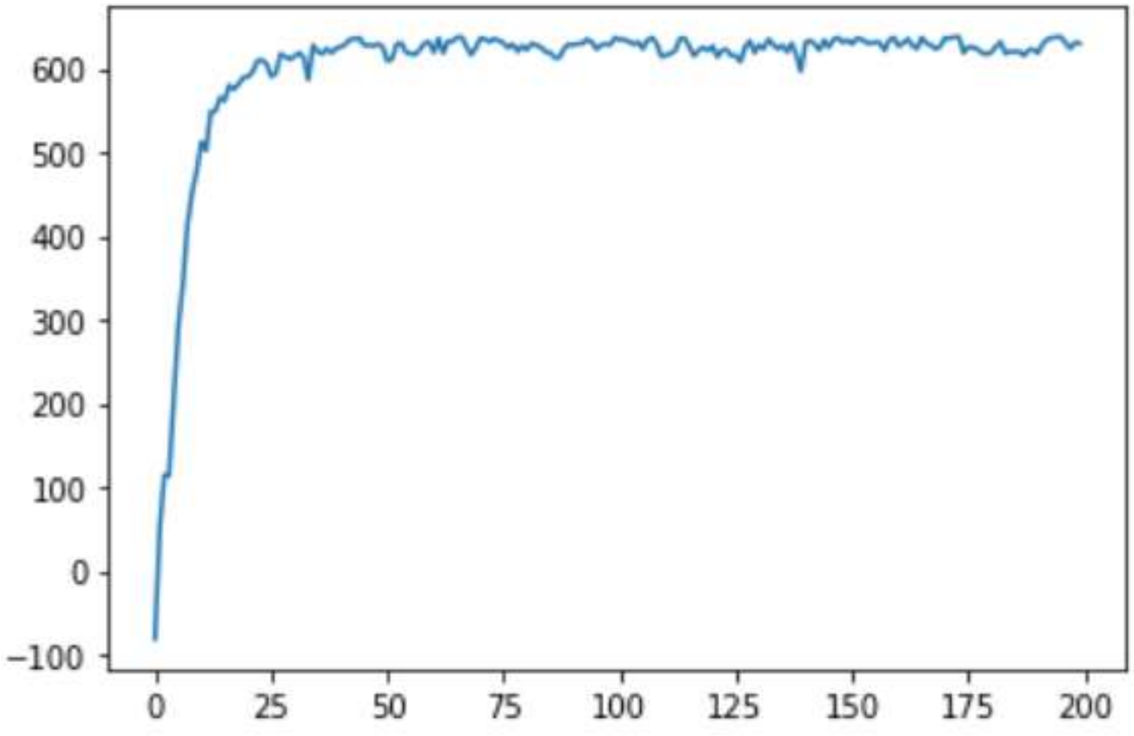
Modeling & Result - Gene Evolution

- Results:

```python
def F(x):
    return np.sin(10*x)*x + np.cos(2*x)*x
```




```python
def F(x): return -x**2+25
```

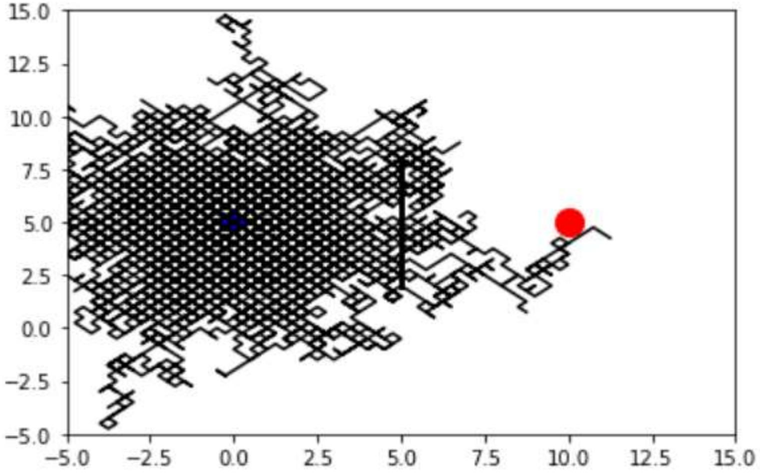


Modeling & Result - Gene Evolution

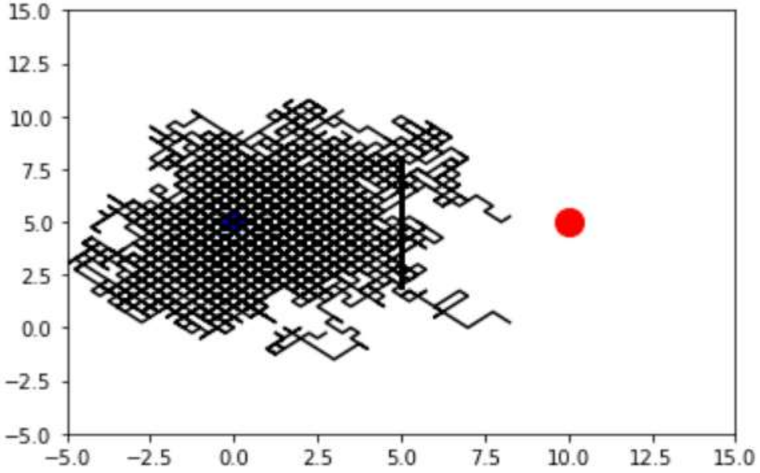
- Results:

Find the Path

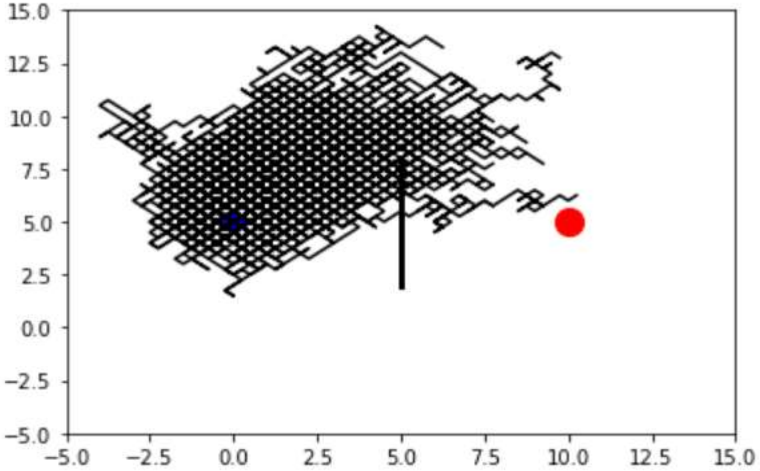
Gen: 0 | best fit: 0.02249173111529981



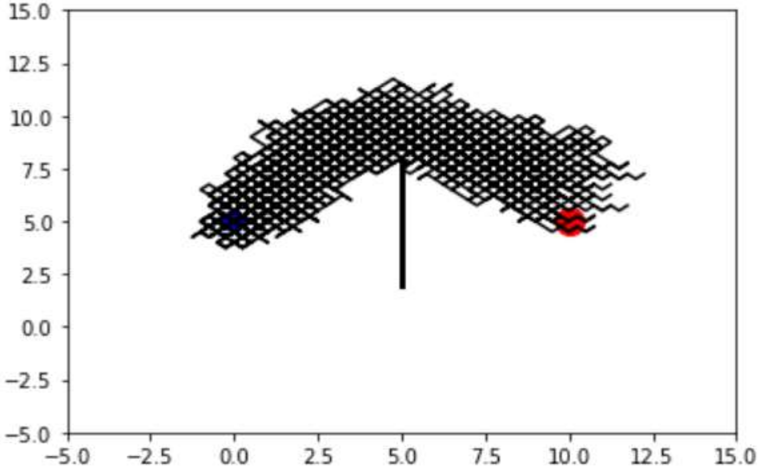
Gen: 9 | best fit: 0.021912590293173776



Gen: 31 | best fit: 0.06744963495484023



Gen: 39 | best fit: 0.3119016527346512



Gen: 99 | best fit: 0.545819714368591