מעבדה מס' 4 בבינה מלאכותית

1.ציינו את הייצוג של הפרטים באוכלוסיה ואת ופונקציות פיטנס

- הייצוג של הפרטים לפי הפונקציה:

Coevolution init:

```
class Coevolution:
    def __init__(self, num_elements, population_size, generations):
        self.num_elements = num_elements
        self.population_size = population_size
        self.generations = generations
        self.network_population = []
        self.vector_population = []

    def generate_initial_population(self):
        for _ in range(self.population_size):
            network = SortingNetwork(self.num_elements)
            vector = np.random.randint(0, 2, size=self.num_elements)
            self.network_population.append(network)
            self.vector_population.append(vector)
```

- פונקצית הפיטנס:

```
def evaluate_fitness(self):
    network_fitness_scores = []
    vector_fitness_scores = []
    comparator_counts = []
    for network, vector in zip(self.network_population, self.vector_population):
        sorted_vector = network.sort(vector)
        network_fitness = sum(sorted_vector == np.sort(vector))
        vector_fitness = self.population_size - sum(sorted_vector == np.sort(vector))
        network_fitness_scores.append(network_fitness)
        vector_fitness_scores.append(vector_fitness)
        comparator_counts.append(network.num_comparators)
    return network_fitness_scores, vector_fitness_scores, comparator_counts
```

2.ציינו את היוריסטיקות הבניה והשיפור שבהן השתמשתם

- היורסטיקת הבניה:

וקטורים:

```python
class Coevolution:
    def __init__(self, num_elements, population_size, generations):
        self.num_elements = num_elements
        self.population_size = population_size
        self.generations = generations
        self.network_population = []
        self.vector_population = []

    def generate_initial_population(self):
        for _ in range(self.population_size):
            network = SortingNetwork(self.num_elements)
            vector = np.random.randint(1, 17, size=self.num_elements)
            self.network_population.append(network)
            self.vector_population.append(vector)
```

רשתות:

```python
import numpy as np

class SortingNetwork:
    def __init__(self, num_elements):
        self.num_elements = num_elements
        self.num_comparators = num_elements * (num_elements - 1) // 2
        self.comparators = np.zeros((self.num_comparators, 2), dtype=int)

    def crossover(self, other):
        child = SortingNetwork(self.num_elements)
        child.comparators = np.where(np.random.rand(*child.comparators.shape) < 0.5,
                                     self.comparators,
                                     other.comparators)
        return child

    def generate_random(self):
        self.comparators = np.random.randint(0, self.num_elements, size=(self.num_comparators, 2))

    def sort(self, input_vector):
        for i in range(self.num_comparators):
            a, b = self.comparators[i]
            if input_vector[a] > input_vector[b]:
                input_vector[a], input_vector[b] = input_vector[b], input_vector[a]
        return input_vector
```

- שיפור להיוריסטיקת הבנייה:
  רשתות ביונטיות :

```python
import numpy as np

class BionicSorting:
    def __init__(self, num_elements):
        self.num_elements = num_elements
        self.permutation = np.arange(num_elements)

    def crossover(self, other):
        child = BionicSorting(self.num_elements)
        child.permutation = self.permutation.copy()
        mask = np.random.choice([True, False], size=self.num_elements)
        child.permutation[mask] = other.permutation[mask]
        return child

    def generate_random(self):
        np.random.shuffle(self.permutation)

    def sort(self, input_vector):
        sorted_vector = input_vector[self.permutation]
        return sorted_vector
```

3.ציירו את גרף ההתכנסות של הפיטנס לאורך האיטרציות

- גרף התכנסות של הפיטנס:

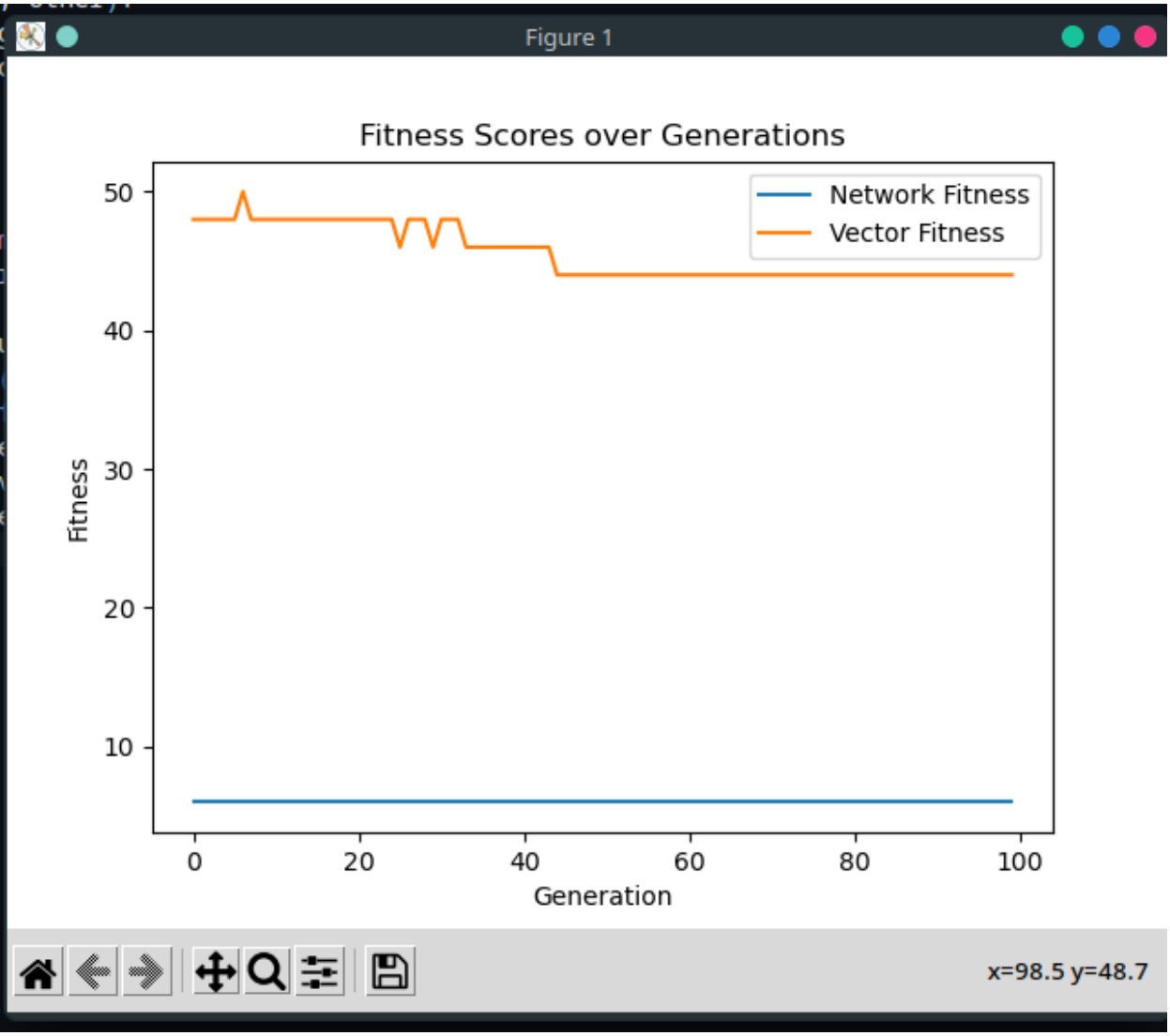
Figure 1
Fitness Scores over Generations
Network Fitness
Vector Fitness
Fitness
Generation
50
40
30
20
10
0
20
40
60
80
100
x=98.5 y=48.7

4.ציינו את הגנים המיטביים שמצאתם

- בכל גיל הצגנו את הגן הכי טוב:
  - הרשת הכי טובה:

```
Generation 100:
Best Sorting Network Fitness: 6
Best Network:
[[0 1]
 [2 0]
 [3 0]
 [3 5]
 [0 0]
 [0 0]
 [0 3]
 [4 0]
 [5 3]
 [0 0]
 [0 4]
 [0 0]
 [0 0]
 [4 0]
 [0 5]]
Best Vector Fitness: 50
```

- הוקטור הכי טוב:

```
Best Vector Fitness: 50
Best Vector:
[ 5 16 10 15  2 14]
Number of Comparators: 15
```

5.מה הייתה פרדיגמת בחירת האוכלוסיות של בחירת הטפילים (וקטורי המיון)

וקטורים עם פיטנס יותר גדול מועדפים על אלו עם פיטנס קטן ככל שהוקטור מכשיל יותר רשתות מיון, הפיטנס שלו עולה יותר כדי לבחור מאכלוסיה השתמשנו ברוליטה:

```python
def roulette_wheel_selection(population, fitness_scores):
    total_fitness = sum(fitness_scores)
    probabilities = [score / total_fitness for score in fitness_scores]
    selected_indices = np.random.choice(len(population), size=len(population), replace=True, p=probabilities)
    selected_population = [population[index] for index in selected_indices]
    return selected_population
```

6.תארו כיצד נעזרתם ברשתות ביטוניות

- הקוד של הרשת הביטונית

```python
import numpy as np

class BionicSorting:
    def __init__(self, num_elements):
        self.num_elements = num_elements
        self.permutation = np.arange(num_elements)

    def crossover(self, other):
        child = BionicSorting(self.num_elements)
        child.permutation = self.permutation.copy()
        mask = np.random.choice([True, False], size=self.num_elements)
        child.permutation[mask] = other.permutation[mask]
        return child

    def generate_random(self):
        np.random.shuffle(self.permutation)

    def sort(self, input_vector):
        sorted_vector = input_vector[self.permutation]
        return sorted_vector
```

7. ציינו כיצד התאמתם את המנוע הגנטי שלכם עבור הבעיה:תארו את אלגוריתם הבחירה והשרידות,האופרטורים הגנטיים,ואת הפרמטרים של האבולוציה

- בחירה:

```python
def selection(self, network_fitness_scores, vector_fitness_scores):
    # Select parents based on fitness scores and comparator counts
    combined_scores = [network_fitness_scores[i] + vector_fitness_scores[i] for i in range(self.population_size)]
    selected_network_population = roulette_wheel_selection(self.network_population, combined_scores)
    selected_vector_population = roulette_wheel_selection(self.vector_population, combined_scores)
    return selected_network_population, selected_vector_population
```

- מוטטציות וקרוסוביר:

```python
def crossover(self, selected_network_population, selected_vector_population):
    new_network_population = []
    new_vector_population = []
    for _ in range(self.population_size):
        parent1_network = random.choice(selected_network_population)
        parent2_network = random.choice(selected_network_population)
        child_network = SortingNetwork(self.num_elements)
        child_network.comparators = np.where(np.random.rand(*child_network.comparators.shape) < 0.5,
                                             parent1_network.comparators,
                                             parent2_network.comparators)
        if random.random() < MUTATION_RATE:  # Mutation probability
            for i in range(child_network.num_comparators):
                if random.random() < 0.1:  # Mutation probability
                    child_network.comparators[i] = np.random.randint(0, self.num_elements, size=2)
        new_network_population.append(child_network)
        new_vector_population.append(random.choice(selected_vector_population))
    return new_network_population, new_vector_population
```

- שלב האבולוציה:

```python
def evolve(self):
    self.generate_initial_population()
    best_network_fitness_scores = []
    best_vector_fitness_scores = []
    comparator_counts = []
    for generation in range(self.generations):
        network_fitness_scores, vector_fitness_scores, comparators = self.evaluate_fitness()
        best_network_fitness_scores.append(max(network_fitness_scores))
        best_vector_fitness_scores.append(max(vector_fitness_scores))
        comparator_counts.append(min(comparators))
        selected_network_population, selected_vector_population = self.selection(network_fitness_scores, vector_fitness_scores)
        new_network_population, new_vector_population = self.crossover(selected_network_population, selected_vector_population)
        self.network_population, self.vector_population = new_network_population, new_vector_population
    return best_network_fitness_scores, best_vector_fitness_scores, comparator_counts
```

8. לא פתרנו