МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №8 з дисципліни

«Методи та технології штучного інтелекту»

«Застосування генетичних алгоритмів в задачах оптимізації »

Перевірив:
Шимкович В.М.

Виконав:
студент 3 курсу
групи ІП-11 ФІОТ
Прищепа В.С.

Київ-2023

Лабораторна робота №8

Застосування генетичних алгоритмів в задачах оптимізації

Мета роботи: отримання та закріплення знань, формування практичних навичок застосування генетичних алгоритмів до різних завдань оптимізації.

Задача: розмістити радіоелементи у корпусі пристрою.
Задамо наступні параметри пристрою: розмір — 50x50 у.о., кількість радіоелементів — 10, елементи довжиною та шириною між 3 та 7, з'єднання елементів відбувається з ймовірністю 0.25.
Нехай генетичний алгоритм має наступні параметри: розмір популяції 50, схрещування одно- та двоточкове, ймовірності мутацій (ген хромосоми змінюється на новий випадковий) 0.1 та 0.25, вагові коефіцієнти загальної довжини з'єднань 0.2 та 0.5, зупинка алгоритму після 100 генерацій або коли загальна площа перекриттів = 0.

Лістинг:

```
import random
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import patches

class Chromosome:
    def __init__(self, genes):
        self.genes = genes
        self.fitness = 0
        self.total_overlap_area = 0
        self.total_length = 0

class GeneticAlgorithm:
    def __init__(self, board_width, board_height, max_side, elements_number,
elements_sizes, connection_matrix, population_size, mutation_rate, connection_rate):
        self.board_width = board_width
        self.board_height = board_height
        self.max_side = max_side
        self.elements_number = elements_number
        self.elements_sizes = elements_sizes
        self.connection_matrix = connection_matrix
        self.population_size = population_size
        self.mutation_rate = mutation_rate
        self.connection_rate = connection_rate
```

```
        self.population = [Chromosome([(random.uniform(0, self.board_width - self.max_side), random.uniform(0, self.board_height - self.max_side)) for _ in range(self.elements_number)]) for _ in range(self.population_size)]

    def calculateOverlapArea(self, elem_1, elem_2, sizes_1, sizes_2):
        x1, y1 = elem_1
        x2, y2 = elem_2
        width1, height1 = sizes_1
        width2, height2 = sizes_2
        left = max(x1 - width1 / 2, x2 - width2 / 2)
        right = min(x1 + width1 / 2, x2 + width2 / 2)
        bottom = max(y1 - height1 / 2, y2 - height2 / 2)
        top = min(y1 + height1 / 2, y2 + height2 / 2)
        overlap_width = max(0, right - left)
        overlap_height = max(0, top - bottom)
        return overlap_width * overlap_height

    def calculateFitness(self, chromosome):
        total_overlap_area = 0
        for i in range(len(chromosome.genes)):
            for j in range(i + 1, len(chromosome.genes)):
                overlap_area = self.calculateOverlapArea(chromosome.genes[i], chromosome.genes[j], self.elements_sizes[i], self.elements_sizes[j])
                total_overlap_area += overlap_area
        chromosome.total_overlap_area = total_overlap_area
        P_total_overlap_area = total_overlap_area / (self.elements_number * self.board_width * self.board_height)
        total_length = 0
        for i in range(len(chromosome.genes)):
            for j in range(len(chromosome.genes)):
                if self.connection_matrix[i][j] == 1:
                    distance = np.linalg.norm(np.array(chromosome.genes[i]) - np.array(chromosome.genes[j]))
                    total_length += distance
        chromosome.total_length = total_length
        O_L_total = total_length / (self.elements_number**2 * np.sqrt(self.board_width**2 + self.board_height**2))
        chromosome.fitness = self.connection_rate * O_L_total + P_total_overlap_area
        return chromosome.fitness

    def chooseParents(self, population):
        parent1 = population[0]
        temp = population.copy()
        temp.pop(0)
        parent2 = random.choice(temp)
```

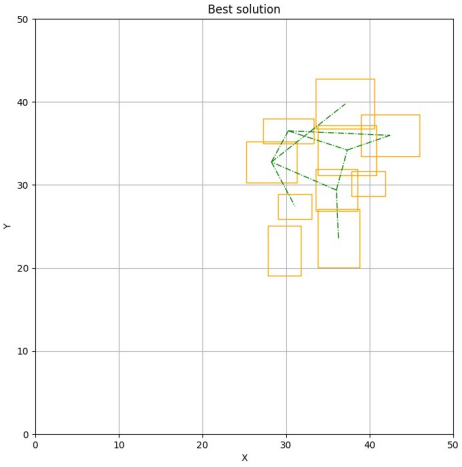
```
        return parent1, parent2

    def crossingOneDot(self, parent1, parent2):
        crossover_point = random.randint(1, len(parent1.genes) - 1)
        child1_genes = parent1.genes[:crossover_point] +
parent2.genes[crossover_point:]
        child2_genes = parent2.genes[:crossover_point] +
parent1.genes[crossover_point:]
        child1 = Chromosome(child1_genes)
        child2 = Chromosome(child2_genes)
        return child1, child2

    def crossingTwoDot(self, parent1, parent2):
        crossover_point_1 = random.randint(0, len(parent1.genes) - 1)
        crossover_point_2 = random.randint(crossover_point_1, len(parent1.genes) - 1)
        child1_genes = parent1.genes[:crossover_point_1] +
parent2.genes[crossover_point_1:crossover_point_2] +
parent1.genes[crossover_point_2:]
        child2_genes = parent1.genes[:crossover_point_1] +
parent2.genes[crossover_point_1:crossover_point_2] +
parent1.genes[crossover_point_2:]
        child1 = Chromosome(child1_genes)
        child2 = Chromosome(child2_genes)
        return child1, child2

    def mutation(self, chromosome):
        for i in range(len(chromosome.genes)):
            if random.random() < self.mutation_rate:
                chromosome.genes[i] = (random.uniform(0, self.board_width -
self.max_side), random.uniform(0, self.board_height - self.max_side))
        return chromosome

    def start(self, generations_number):
        best_fitness_per_generation = []
        for generation in range(generations_number):
            for chromosome in self.population:
                self.calculateFitness(chromosome)
            self.population.sort(key=lambda x: x.fitness)
            next_generation = self.population[:len(self.population) // 2]
            if self.population[0].total_overlap_area == 0:
                break
            best_fitness_per_generation.append(self.population[0].fitness)
            while len(next_generation) < len(self.population):
                parent1, parent2 = self.chooseParents(next_generation)
                child1, child2 = self.crossingTwoDot(parent1, parent2)
```

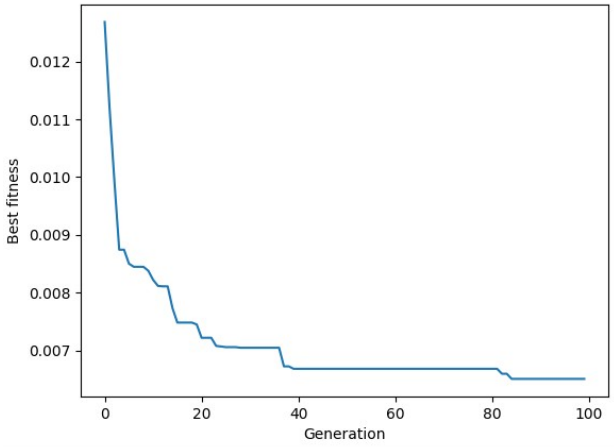
```
                child1 = self.mutation(child1)
                child2 = self.mutation(child2)
                next_generation.extend([child1, child2])
            self.population = next_generation
        best_solution = sorted(self.population, key=lambda x: x.fitness)[0]
        self.calculateFitness(best_solution)
        return best_solution, best_fitness_per_generation

board_width = 50
board_height = 50
elements_number = 10
min_side = 3
max_side = 7

elements_sizes = [(random.randint(min_side, max_side), random.randint(min_side,
max_side)) for _ in range(elements_number)]
connection_matrix = [[0 if random.random() < 0.75 else 1 for _ in
range(elements_number)] for _ in range(elements_number)]
genetic_algo = GeneticAlgorithm(board_width=board_width,
board_height=board_height, max_side=max_side,
elements_number=elements_number,
 elements_sizes=elements_sizes, connection_matrix=connection_matrix,
population_size=50, mutation_rate=0.25, connection_rate=0.5)
best_solution, best_fitness_per_generation =
genetic_algo.start(generations_number=100)
print(f'Total overlap area: {best_solution.total_overlap_area}')

fig, ax = plt.subplots(figsize=(8, 8))
for i, (x, y) in enumerate(best_solution.genes):
    ax.add_patch(patches.Rectangle((x, y), elements_sizes[i][0], elements_sizes[i][1],
fill=None, edgecolor='orange'))
for i in range(len(best_solution.genes)):
    for j in range(i + 1, len(best_solution.genes)):
        if connection_matrix[i][j] == 1:
            x1, y1 = best_solution.genes[i]
            x2, y2 = best_solution.genes[j]
            ax.plot([x1 + elements_sizes[i][0] / 2, x2 + elements_sizes[j][0] / 2], [y1 +
elements_sizes[i][1] / 2, y2 + elements_sizes[j][1] / 2], linestyle='-.', linewidth=1,
color='green')
plt.xlim(0, board_width)
plt.ylim(0, board_height)
plt.title('Best solution')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True)
```

```
plt.show()

plt.plot(best_fitness_per_generation)
plt.xlabel('Generation')
plt.ylabel('Best fitness')
plt.show()
```

Виконання:

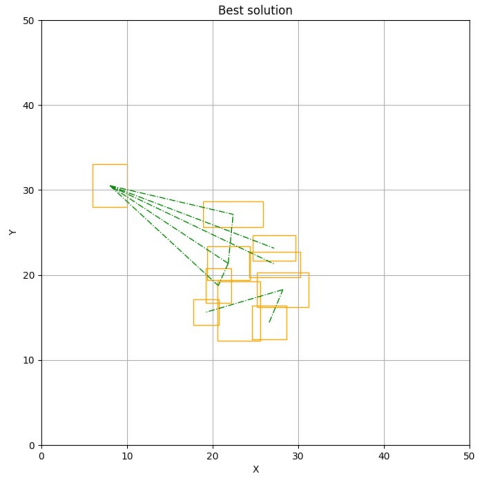
1) Схрещування одноточкове, ймовірності мутацій 0.1, вагові коефіцієнти загальної довжини з’єднань 0.2:

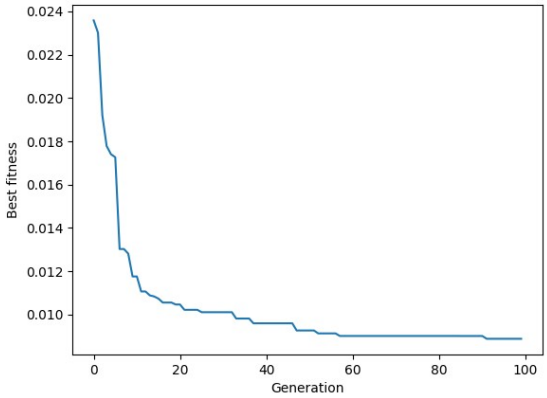





```
Total overlap area: 22.67921060795426
Press any key to continue . . .
```

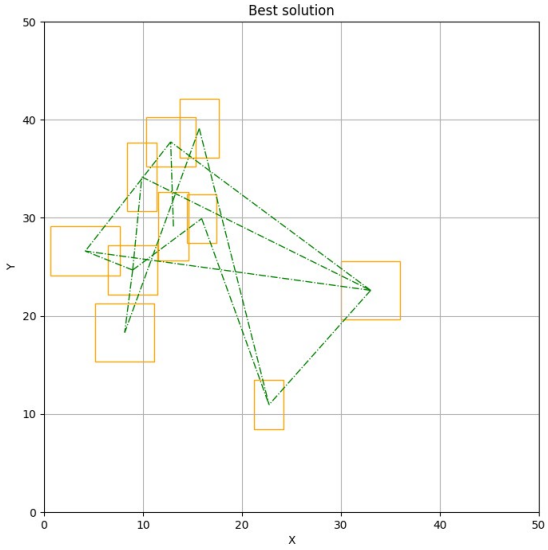
2) Схрещування одноточкове, ймовірності мутацій 0.1, вагові коефіцієнти загальної довжини з’єднань 0.5:

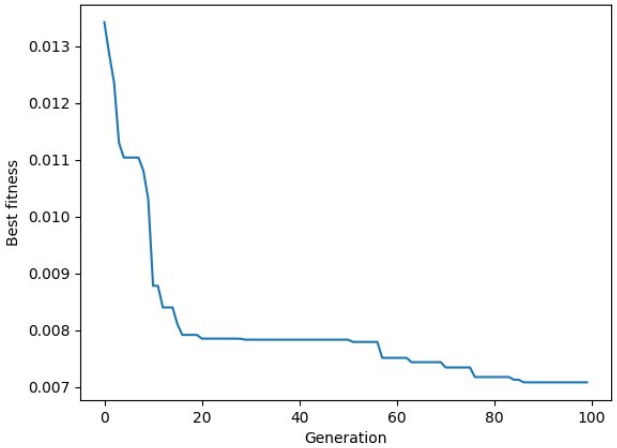





```
Total overlap area: 22.65875720284321
Press any key to continue . . .
```

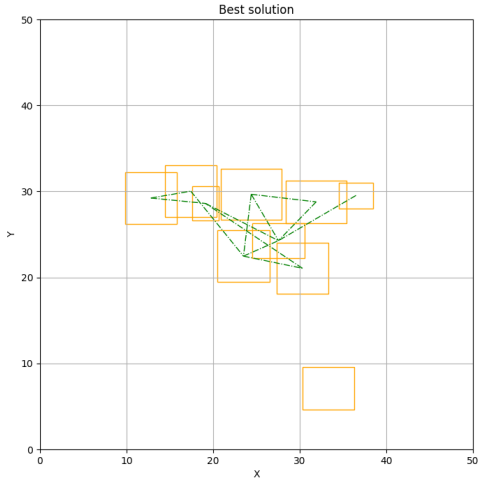
3) Схрещування одноточкове, ймовірності мутацій 0.25, вагові коефіцієнти загальної довжини з’єднань 0.2:

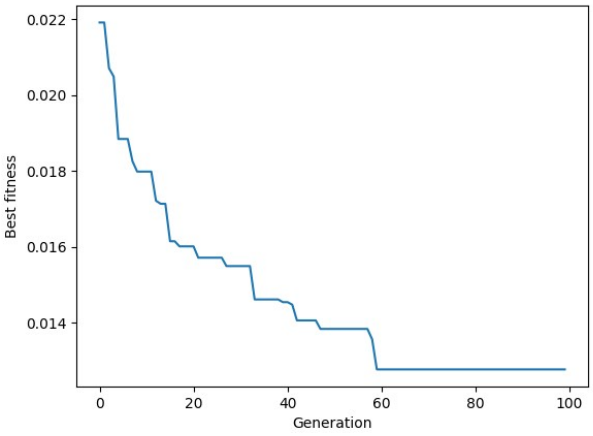





```
Total overlap area: 9.388036868502455
Press any key to continue . . .
```

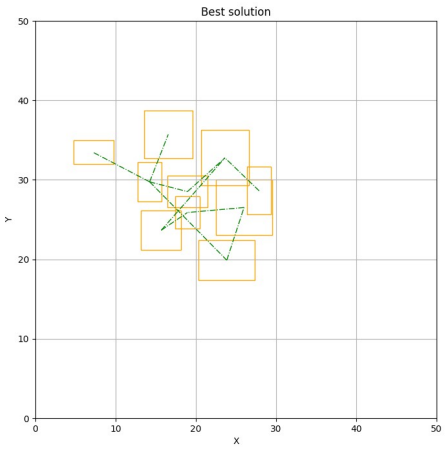
4) Схрещування одноточкове, ймовірності мутацій 0.25, вагові коефіцієнти загальної довжини з’єднань 0.5:

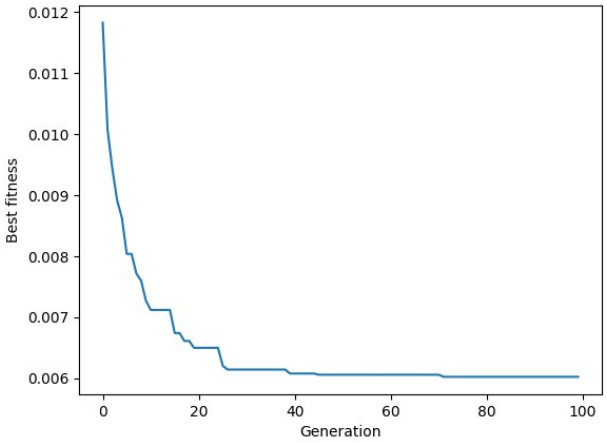





```
Total overlap area: 29.26070769068711
Press any key to continue . . .
```

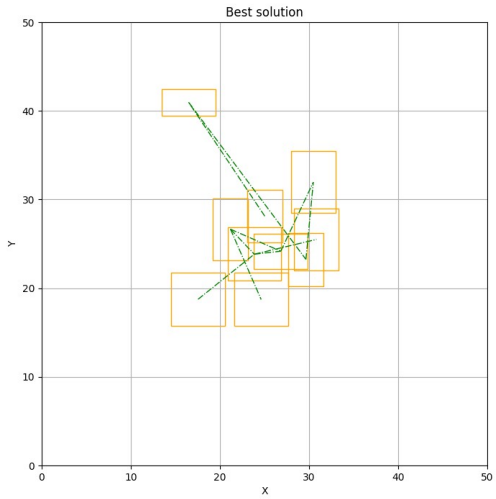
5) Схрещування двоточкове, ймовірності мутацій 0.1, вагові коефіцієнти загальної довжини з’єднань 0.2:

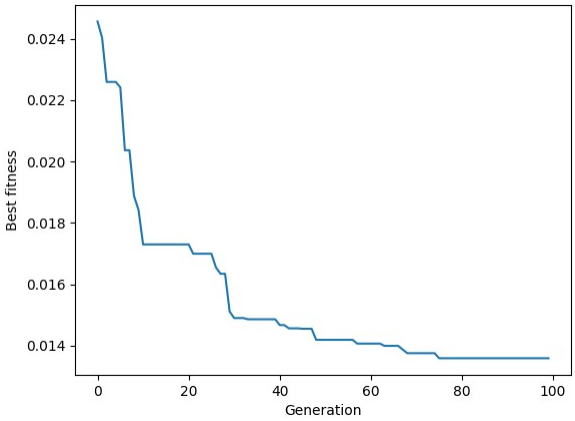





```
Total overlap area: 18.751676183073098
Press any key to continue . . .
```

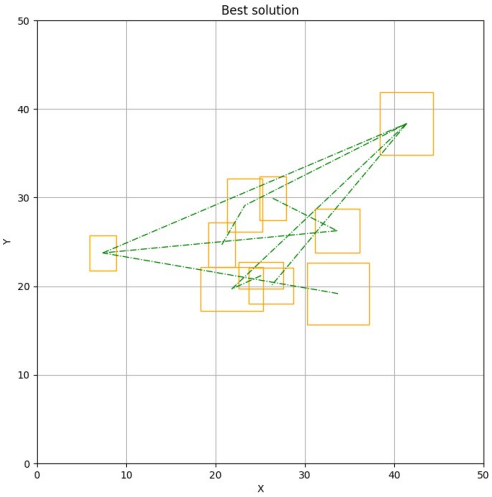
6) Схрещування двоточкове, ймовірності мутацій 0.1, вагові коефіцієнти загальної довжини з’єднань 0.5:

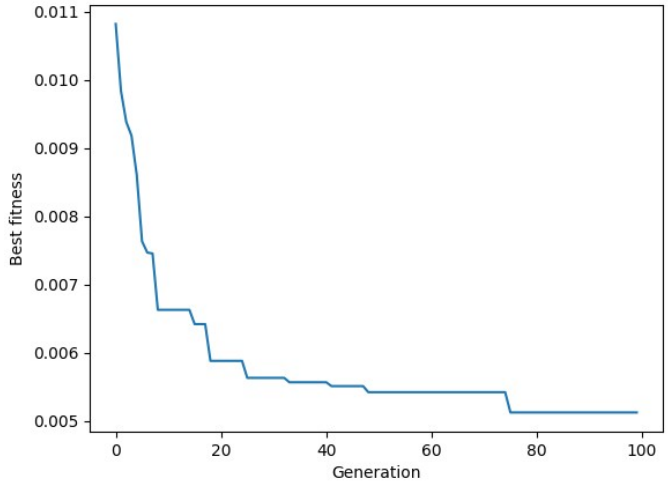





```
Total overlap area: 72.96454158032074
Press any key to continue . . .
```

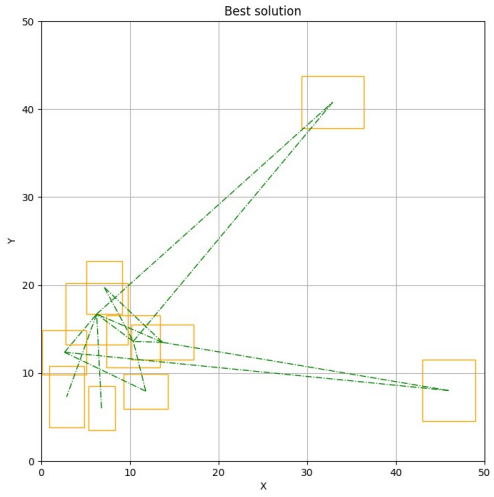
7) Схрещування двоточкове, ймовірності мутацій 0.25, вагові коефіцієнти загальної довжини з’єднань 0.2:

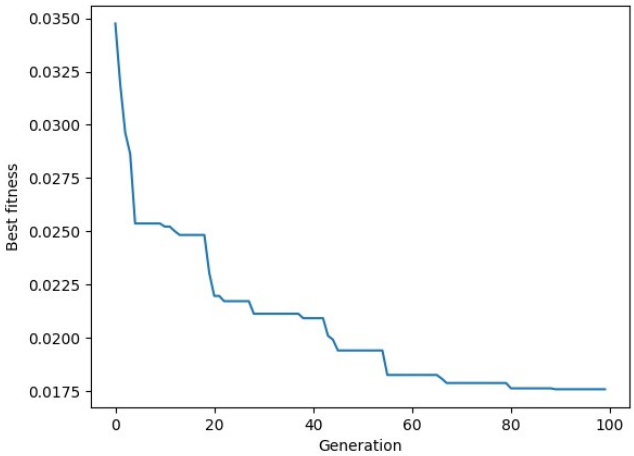





```
Total overlap area: 14.910981364225877
Press any key to continue . . .
```

8) Схрещування двоточкове, ймовірності мутацій 0.25, вагові коефіцієнти загальної довжини з’єднань 0.5:






```
Total overlap area: 41.62961522368425
Press any key to continue . . .
```

Висновок: Під час виконання лабораторної роботи я застосував генетичний алгоритм для задачі розміщення радіоелементів у корпусі пристрою. Код, результати виконання наведені вище. Найкращі параметри ГА: схрещування одноточкове, ймовірності мутацій 0.25, вагові коефіцієнти загальної довжини з’єднань 0.2.