

Учебна дисциплина: *Основи на инженерното проектиране*

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 8

- Тема:** *Оптимизация в инженерното проектиране. Решаване на оптимизационна задача с метода на Генетичните алгоритми*
- Продължителност:** 3 учебни часа
- Задание:**
1. Дефиниране на точните изисквания и целева функция.
 2. Имплементиране на функции за оценка.
 3. Избор и настройка на операторите за селекция, кръстосване и мутация, които съответстват на поставената задача.
 4. Определяне на началната популация и параметрите на генетичния алгоритъм.
 5. Разработване на функции за замяна на старите индивиди с новите и управление на генерациите.
 6. Тест и оптимизация на кода.
- Цел:** Запознаване с основните етапи на генетичния алгоритъм - инициализация на популацията, оценка на приспособеността, селекция, кръстосване, мутация и замяна на индивидите. Програмна реализация на решаване на оптимизационна задача с метода на генетичните алгоритми посредством език за програмиране Python.

I. Теоретична постановка

1. Теория

1.1. Оптимизация

Оптимизацията се занимава с проблеми на минимизиране или максимиране на функция с няколко променливи, обикновено подложени на ограничения за равенство и/или неравенство. Тя е ключов елемент при операционни изследвания, мениджмънт и инженерния дизайн. Голяма част от проблемите, с които тези области се сблъскват, са много сложни и трудни за решаване с помощта на конвенционални техники за оптимизация. Въз основа на тяхната лекота на работа, минимални изисквания и паралелна и глобална перспектива, генетичните алгоритми се прилагат широко при разрешаването на различни проблеми. Техниките за генетична оптимизация се разделят обикновено на три основни категории в зависимост от характера на оптимационния проблем, който решават:

❖ Размити (Continuous) оптимизации:

- Този вид оптимизации се прилага, когато пространството на решенията е непрекъснато или размито, което означава, че решенията могат да бъдат представени като набор от реални числа или стойности в интервал. Генетичните алгоритми за размита оптимизация се използват за намиране на най-добрите числови решения в непрекъснатото пространство.

❖ Комбинаторни (Combinatorial) оптимизации:

- В този случай, решенията са комбинаторни или дискретни по природа. Това означава, че решенията се състоят от комбинации от обекти, елементи или действия. Примери за комбинаторни оптимационни задачи включват съставянето на разписания, маршрутизация на мрежи и други. Генетичните алгоритми се използват за търсене на оптимални комбинации в дискретното пространство.

❖ Многоцелеви (Multi-objective) оптимизации:

- В многоцелевите оптимизации има повече от една целева функция, които трябва да се оптимизират едновременно. Това се нарича множествена целева оптимизация и често води до компромиси между различни цели. Генетичните алгоритми за многоцелева оптимизация се използват, за да намерят множество от недоминиращи оптимални решения.

1.2. Оптимизация с генетичен алгоритъм

Оптимизацията с генетични алгоритми е техника, вдъхновена от процеса на естествения подбор и генетиката. Генетичните алгоритми (Genetic algorithms (GA)) са клас еволюционни алгоритми, използвани за намиране на приблизителни решения на проблеми с оптимизацията и търсенето. Те са особено полезни, когато пространството

за търсене е голямо и сложно. Оптимизация с генетичен алгоритъм може да се прилага и към трите категории оптимизации (многоцелеви, комбинаторни и размити), в зависимост от характеристиките на конкретната задача.

1.3. Проектиране на надеждността на мрежата

Надеждността на системата се определя като вероятността системата да работи по най-добрания начин за определен интервал от време при дадени условия. Проблемите при проектирането на надеждността включват анализ на надеждността, тестване на надеждността, анализ на данните за надеждност, нарастване на надеждността и т.н.

Проектите за надеждност на мрежата се основават на споделяне на хардуерни и софтуерни ресурси и осигуряват достъп до основната сървърна система от отдалечени места. Важна стъпка в процеса на проектиране на мрежата е да се намери най-доброто разположение на компонентите за оптимизиране на определени критерии за ефективност като цена, надеждност, забавяне на предаването или пропускателна способност. Надеждността на мрежовия дизайн е както следва:

- Надеждност на цялата мрежа – вероятността всеки възел в мрежата да е свързан един с друг
- Надеждност на мрежата от източника до приемника – вероятността източникът да е свързан с приемника, така че възелът източник в мрежата да може да комуникира с възела приемник за определено време.

Генетичният алгоритъм предоставя подходи за решение за оптимален мрежови дизайн, като се вземат предвид горепосочените надеждности.

Основната идея зад генетичния алгоритъм е създаването и еволюцията на популация от потенциални решения чрез прилагане на оператори като селекция, кросоувър и мутация. Във всяка итерация на алгоритъма се оценяват качествата на индивидите в популацията, след което се избират най-добрите решения, които участват в процеса на създаване на следващото поколение.

1.4. Основните етапи в генетичния алгоритъм:

1.4.1. Инициализация: Създаване на начална популация от индивиди (решения).

1.4.2. Оценка (Фитнес функция): Оценка на качествата на всеки индивид в популацията чрез фитнес функция. Тази функция определя степента, в която индивидът отговаря на изискванията на проблема, който се опитваме да решим.

1.4.3. Селекция: Избиране на индивидите, които ще участват в създаването на следващото поколение. По-добрите индивиди имат по-голям шанс да бъдат избрани.

1.4.4. Кръстосване: Създаване на нови индивиди чрез комбиниране на характеристики от два или повече избрани родителя.

1.4.5. Мутация: Въвеждане на случайни промени в индивидите, които допринасят за разнообразието и предотвратяване на стагнацията на алгоритъма.

1.4.6. Повторение: Повтаряне на стъпките от 1.4.2. до 1.4.5. за определен брой итерации или докато не бъде достигнато желаното условие за спиране.

2. Задачи за изпълнение

Изисквания

Инсталиран Python 3.7 или по висока версия, примерна работна среда Visual Studio Code, библиотеките numpy, matplotlib, scipy и random.

Заб.: Направете нов файл на Python (със стандартно разширение .py). За всяка от описаните по-долу задачи записвайте кода в него, като не забравяте да запазите файла.

Напишете код за генетичен алгоритъм, който да бъде използван за оптимизация на разположението на точките за достъп в учебна зала с цел максимизиране на обхвата на Wi-Fi сигнала. Изпълнете поетапно стъпките:

1. Импортирайте библиотеки numpy, matplotlib, scipy и random

```
import random
import scipy.io
import matplotlib.pyplot as plt
import numpy as np
```

2. Дефинирайте константи и променливи. Задайте :

- **classroom_width, classroom_length:** Ширина и дължина на учебната зала в метри, съответно 7 и 5.
- **num_app:** Брой точки за достъп до Wi-Fi, съответно да е 5.
- **population_size:** Големина на популацията от различни разпределения на точките за достъп, да е равно на 12.
- **generations:** Брой поколения, през които ще се извършва оптимизацията, да е равен на 10.

- **mutation_rate:** Скорост на мутация - вероятността за случайно изменение на позицията на точка при генериране на ново поколение, да е равна на 0.1 (съответства на 10%).

3. Дефинирайте функция за изчисляване на силата на сигнала:

- **calculate_signal_strength:** Използва модел на разстоянието, за да изчисли силата на Wi-Fi сигнала в дадена точка в стаята, базирана на разстоянието до точките за достъп.

```
# Функция за изчисляване на силата на сигнала в дадена точка (модел на разстоянието)
def calculate_signal_strength(ap_x, ap_y, point_x, point_y):
    distance = np.sqrt((ap_x - point_x)**2 + (ap_y - point_y)**2)
    return 1 / (distance + 1) # Обратен модел на разстоянието
```

4. Дефинирайте функция за визуализация на начални и крайни позиции на точките за достъп:

- **visualize_initial_and_final:** Използва библиотеката **matplotlib** за визуализация на началното и крайното разпределение на точките за достъп.

```
# функция за визуализация на началните случаини точки и крайното разпределение на точките за достъп
def visualize_initial_and_final(classroom_width, classroom_length, initial_population, final_population):
    plt.figure(figsize=(15, 6))

    # Изобразяване на началните случаини точки с номера в черно
    for i, (x, y) in enumerate(initial_population):
        plt.text(x, y, str(i + 1), color='black', fontsize=8, ha='center', va='center')
    plt.scatter(*zip(*initial_population), color='red', marker='o', label='Начална позиция на точките за достъп до Wi-Fi')

    # Изобразяване на крайното разпределение на точките за достъп с номера в черно
    for i, (x, y) in enumerate(final_population):
        plt.text(x, y, str(i + 1), color='black', fontsize=8, ha='center', va='center')
    plt.scatter(*zip(*final_population), color='blue', marker='o', label='Финална позиция на точките за достъп до Wi-Fi')

    plt.xlabel('Ширина на учебна зала (м)')
    plt.ylabel('Дължина на учебна зала (м)')
    plt.title('Начално и крайно разпределение на точките за достъп')
    plt.legend()
    plt.show()
```

5. Дефинирайте функция за динамична визуализация на промените:

- **visualize_dynamic_changes:** За визуализация на промените в разпределението на точките за достъп и резултатите от фитнес през различните поколения.

```
# Функция за динамично визуализиране на промените в разположението на точките за достъп и фитнес резултатите през поколенията
def visualize_dynamic_changes(classroom_width, classroom_length, population, fitness_scores):
    for generation in range(generations):
        plt.figure(figsize=(15, 6))

        # Начертаване на топлинната карта за текущото поколение
        plt.subplot(1, 2, 1)
        ap_placements = population[generation]
        x = np.linspace(0, classroom_width, 100)
        y = np.linspace(0, classroom_length, 100)
        X, Y = np.meshgrid(x, y)
        Z = np.zeros_like(X)
        for ap_x, ap_y in ap_placements:
            Z += calculate_signal_strength(ap_x, ap_y, X, Y)
        plt.pcolormesh(X, Y, Z, shading='auto', cmap='viridis')
        plt.colorbar(label='Сила на сигнала')
        for i, (x, y) in enumerate(ap_placements):
            plt.text(x, y, str(i + 1), color='black', fontsize=8, ha='center', va='center')
        plt.scatter(*zip(*ap_placements), color='red', marker='o', label='Разпределение на точките за достъп')
        plt.xlabel('Ширина на учебна зала (м)')
        plt.ylabel('Дължина на учебна зала (м)')
        plt.title(f'Сила на Wi-Fi сигнала и разпределение на точките за достъп (Поколение {generation + 1})')
        plt.legend()

        # Изобразяване на графиката на фитнеса за всички поколения
        plt.subplot(1, 2, 2)
        plt.plot(range(1, generation + 2), fitness_scores[:generation + 1], marker='o')
        plt.xlabel('Поколение')
        plt.ylabel('Резултат от фитнеса')
        plt.title('Резултат от фитнеса за различните поколения')

    plt.tight_layout()
    plt.show()
```

6. Инициализирайте популацията:

- Генерира се първоначална популация от случайни позиции на точките за достъп.

```
# Инициализация на популацията със случайни разпределения на точките за достъп
initial_population = [(random.uniform(0, classroom_width), random.uniform(0, classroom_length)) for _ in range(population_size)]
population = [initial_population.copy()]
```

- Инициализират се променливи за проследяване на резултатите

```
# Променливи за проследяване на най-доброто решение и резултатите от фитнеса
best_solution = None
best_fitness = float('-inf')
all_fitness_scores = []
```

7. Главен оптимационен цикъл - Генетичен алгоритъм:

- През всяко поколение се изчислява фитнеса за всеки индивид в популацията.

```
# Оптимизационен цикъл на генетичния алгоритъм
for generation in range(generations):
    # Оценка на фитнеса на всеки индивид в популацията
    fitness_scores = []
    for ap_x, ap_y in population[generation]:
        coverage = 0
        for point_x in range(classroom_width):
            for point_y in range(classroom_length):
                coverage += calculate_signal_strength(ap_x, ap_y, point_x, point_y)
        fitness_scores.append(coverage)
```

- Най-доброто решение се запазва, идентифицирано по най-голямата стойност на фитнеса.

```
# Актуализация на най-доброто решение, ако е намерено
if max(fitness_scores) > best_fitness:
    best_solution = population[generation][fitness_scores.index(max(fitness_scores))]
    best_fitness = max(fitness_scores)

# Запазване на всички резултати от фитнеса
all_fitness_scores.append(max(fitness_scores))
```

- Родителите се избират с вероятност, пропорционална на техния фитнес.

```
# Избор на родители на база на резултатите от фитнеса
selected_parents = random.choices(population[generation], weights=fitness_scores, k=population_size)
```

- Потомството се генерира чрез кръстосване между двата родителя и се прилага мутация.

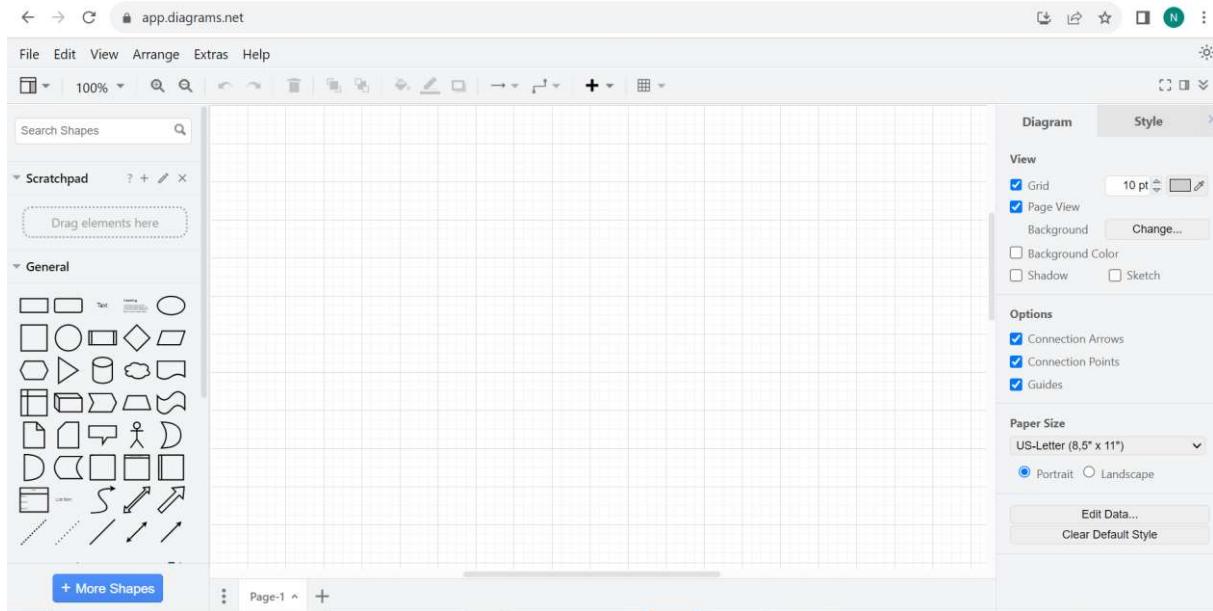
```
# Създаване на потомство чрез кръстосване
offspring = []
for _ in range(population_size):
    parent1, parent2 = random.sample(selected_parents, 2)
    crossover_x = random.uniform(parent1[0], parent2[0])
    crossover_y = random.uniform(parent1[1], parent2[1])
    offspring.append((crossover_x, crossover_y))
```

- Новото поколение се използва за следващата итерация.

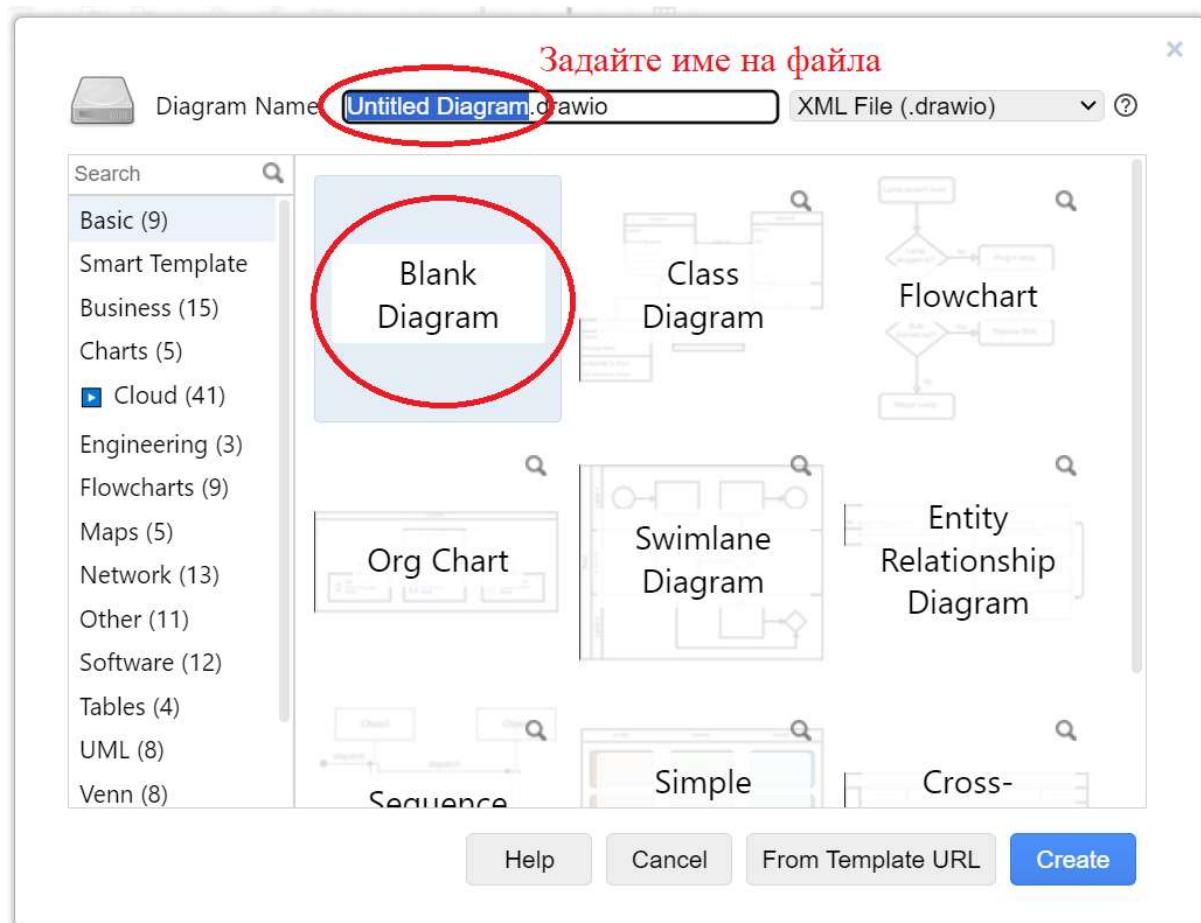
```
# Прилагане на мутация
for i in range(len(offspring)):
    if random.random() < mutation_rate:
        offspring[i] = (random.uniform(0, classroom_width), random.uniform(0, classroom_length))

# Замяна на старото население с ново поколение
population.append(offspring)
```

- Използвайте програмата <https://app.diagrams.net/> за да начертаете блоковата схема на генетичния алгоритъм. Това означава да вземете основните стъпки на алгоритъма, а не детайлно да правите детайлно описание на логиката на кода.



От File -> New ще се отвори:



Задавате име на файла и избирате Blank Diagram, а след това кликвате върху синият бутон „Create“.

- В протокола поставете еcranна снимка на начертаната блокова схема на генетичния алгоритъм.

8. Изведете резултатите в конзолата за:

- Най-добро разпределение на точките за достъп (best_solution)
- Най-добро покритие (best_fitness)
- В протокола представете еcranна снимка на координатите на точката за достъп и най-доброто покритие

пр:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
\adapter/.../debugpy\launcher' '59179' '--' 'C:\Users\nicol\Downloads\8_oip.py'  
Най-добро разпределение на точките за достъп: (3.828159109829729, 1.905131773111191)  
Най-добро покритие: 11.541606536229835
```

9. Визуализирайте:

- Началното и крайното разпределение на точките за достъп:

```
# Визуализация на началното и крайното разпределение на точките за достъп  
visualize_initial_and_final(classroom_width, classroom_length, initial_population, population[-1])
```

- В протокола представете еcranна снимка на визуализацията на началното и крайното разпределение на точките за достъп
- Динамичните промени в разпределението и резултатите от фитнес през различните поколения:

```
# Визуализация на динамичните промени в разпределението на точките за достъп и резултатите от фитнеса по поколения  
visualize_dynamic_changes(classroom_width, classroom_length, population, all_fitness_scores)
```

Забележка: За да се появи следващата графика трябва да се затвори прозореца на вече съществуващото изображение. Това важи и за допуописаните визуализации.

- В протокола представете еcranна снимка на визуализациите на промените в разпределението и резултатите от фитнес функцията през различните поколения. *Забележка: Броят на изображенията трябва да е равен на зададените параметри на generations (в нашият случай са 10).*
- Траекторията на популацията

```
# Траектория на популацията
trajectory_scores = []
for generation in range(generations + 1):
    distances = [np.linalg.norm(np.array(point) - np.array(best_solution)) for point in population[generation]]
    trajectory_scores.append(np.mean(distances))

plt.figure(figsize=(10, 6))
plt.plot(range(generations + 1), trajectory_scores, marker='o')
plt.xlabel('Поколение')
plt.ylabel('Средно разстояние до най-доброто решение')
plt.title('Траектория на популацията')
plt.show()
```

- В протокола представете екранна снимка на визуализацията на траекторията на популацията
- Графика на сходство:

```
# Графика на сходство
plt.figure()
plt.plot(range(1, generations + 1), all_fitness_scores, marker='o')
plt.xlabel('Поколение')
plt.ylabel('Най-добър фитнес резултат')
plt.title('Графика на сходство')
plt.show()
```

- В протокола представете екранна снимка на графиката за сходство