Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра ІПІ

Звіт

з лабораторної роботи № 8 з дисципліни
«Методи та технології штучного інтелекту»

„Застосування генетичних алгоритмів в задачах оптимізації”

Виконав(ла) *ІП-14 Шляхтун Денис Михайлович*
(шифр, прізвище, ім'я, по батькові)

Перевірив *Шимкович Володимир Миколайович*
(прізвище, ім'я, по батькові)

Київ 2023

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

Постановка задачі.

Для студентів з парними номерами в журналі групи необхідно написати програму, на будь-якій мові високого рівня, що реалізує рішення сформульованої задачі оптимізації обчислювальної мережі з використанням описаного генетичного алгоритму.

86	Шляхтун Денис Михайлович

Обґрунтувати вибір всіх параметрів і критеріїв зупинки роботи генетичного алгоритму.

Оформіть звіт по лабораторній роботі.

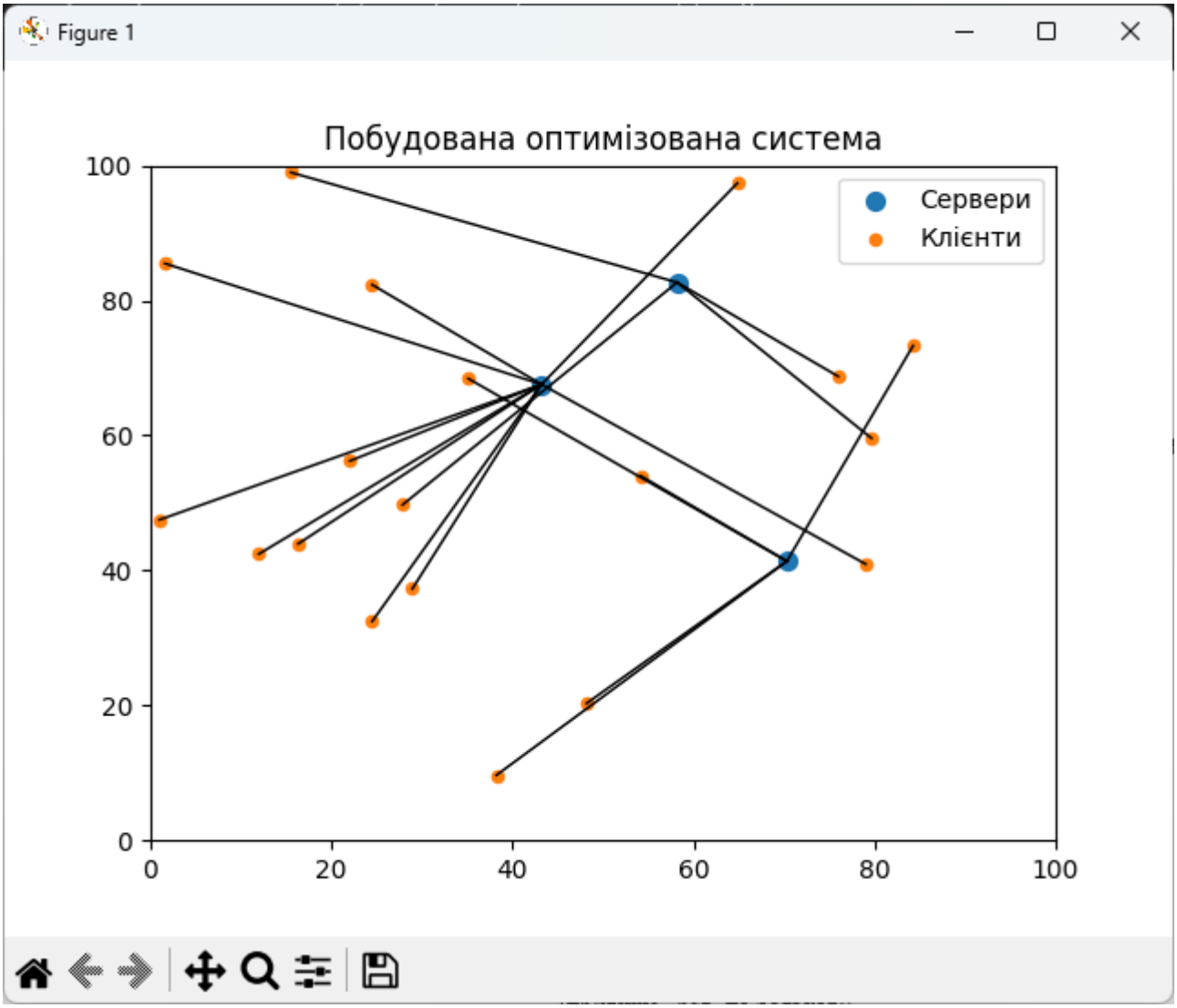
Виконання завдання.

Робота виконана мовою високого рівня Python.

Значення кількості серверів – 3, кількість клієнтів – 70% від максимального навантаження на сервер. Зупинка алгоритму відбувається після підряд 50 ітерацій без змін, загальний ліміт ітерацій встановлено на значення 1000, шанс мутацій встановлено на рівні 20%, розмір популяцій встановлено на рівні 1000. Виведемо вхідні дані:

```
Сервери:
[[70.31, 41.37 -> 7] [43.20, 67.56 -> 12] [58.18, 82.67 -> 8]]
Клієнти:
[(79.02, 40.93) (16.31, 43.94) (64.83, 97.38) (35.11, 68.42)
 (11.96, 42.42) (22.02, 56.22) (84.25, 73.29) (24.48, 82.33)
 (54.18, 53.99) (1.59, 85.47) (75.99, 68.73) (0.99, 47.50) (15.51, 98.97)
 (79.66, 59.58) (48.17, 20.30) (27.85, 49.72) (28.92, 37.26)
 (24.48, 32.44) (38.21, 9.65)]
```

Результат оптимізації:




```
Найкращий результат:
Хромосома: [1 1 1 0 1 1 0 1 0 1 2 1 2 2 0 2 1 1 0]
Навантаження серверів: [2 2 4]
Затримки: [1992.17485604 1281.09233701 1357.30709175 1970.3237507  1607.79071199
  577.16249489 1213.74388783  568.54070884  419.24617055 2052.26473989
  511.29605922 2184.32829368 2086.85406954  994.06755058  933.78867667
 2005.9278388  1122.1360028  1583.88358683 2036.20705171]
Найбільша затримка: 2184
```

Висновок.

При виконанні лабораторної роботи було застосовано генетичний алгоритм для оптимізації обчислювальної мережі. Було отримано та закріплено знання, практичні навички застосування генетичних алгоритмів до завдання оптимізації.

Код програми:

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import random
import copy
matplotlib.use('TkAgg')

worst_arr = []
class Chromosome:
    def __init__(self):
        self.gene = np.zeros(AMOUNT_OF_CLIENTS, dtype=int)

    def init_with_gene(self, gene):
        self.gene = gene
        self._calculate_capacities()
        self._recalculate_full()

    def generate_random(self):
        self.capacities = copy.deepcopy(max_capacities)
        self.clients_delays = np.array([np.Infinity for _ in
range(AMOUNT_OF_CLIENTS)])
        self.worst_delay = np.Infinity
        for i in range(AMOUNT_OF_CLIENTS):
            server = random.randrange(0, AMOUNT_OF_SERVERS)
            self.gene[i] = server
            self.capacities[server] -= 1
        self._recalculate_full()

    def generate_greedy(self):
        self.capacities = copy.deepcopy(max_capacities)
        self.clients_delays = np.array([np.Infinity for _ in
range(AMOUNT_OF_CLIENTS)])
        self.worst_delay = np.Infinity
        order = np.random.permutation(AMOUNT_OF_CLIENTS)
        for client in order:
            available_servers = np.where(self.capacities > 0)[0]
            if len(available_servers) == 0:
                raise ValueError('No avaliable servers')
            distances_avaliable = distances[available_servers, client]
            closest_server = available_servers[np.argmin(distances_avaliable)]
            self.gene[client] = closest_server
            self.capacities[closest_server] -= 1
        self._recalculate_full()

    def generate_greedy_optimal(self):
        self.capacities = copy.deepcopy(max_capacities)
        self.clients_delays = np.full(AMOUNT_OF_CLIENTS, np.inf)
        self.worst_delay = np.inf
        copied_distances = copy.deepcopy(distances)
```

```
        average_distances = np.mean(distances, axis=0)
        for _ in range(AMOUNT_OF_CLIENTS):
            client = np.argmax(average_distances)
            available_servers = np.where(self.capacities > 0)[0]
            if len(available_servers) == 0:
                raise ValueError('No avaliable servers')
            distances_available = copied_distances[available_servers, client]
            closest_server = available_servers[np.argmin(distances_available)]
            self.gene[client] = closest_server
            self.capacities[closest_server] -= 1
            if self.capacities[closest_server] == 0:
                copied_distances[closest_server, :] = np.inf
            copied_distances[:, client] = np.inf
            average_distances[client] = 0
        self._recalculate_full()

    def modify_client(self, client_id, server_id):
        self.capacities[self.gene[client_id]] += 1
        self.capacities[server_id] -= 1
        self.gene[client_id] = server_id
        self._recalculate_full()

    def mutate(self, chance):
        if get_rand_bool(chance):
            self._mutate()
            return True
        else:
            return False

    def show_iteration_info(self, number):
        worst_arr.append(self.worst_delay)
        print(f'Ітерація {number}, кращий результат: {self.worst_delay:.0f}')

    def info(self):
        print(f'Хромосома: {self.gene}')
        print(f'Навантаження серверів: {self.capacities}')
        print(f'Затримки: {self.clients_delays}')
        print(f'Найбільша затримка: {self.worst_delay:.0f}\n')

    def _calculate_capacities(self):
        self.capacities = copy.deepcopy(max_capacities)
        for i in self.gene:
            self.capacities[i] -= 1

    def _recalculate_full(self):
        server_indices = self.gene
        client_indices = np.arange(len(self.gene))
        self.clients_delays = evaluate_client_delay(server_indices, client_indices,
self.capacities)
        self.worst_delay = np.max(self.clients_delays)
```

```
    def _mutate(self):
        mutated_server = random.randrange(0, AMOUNT_OF_SERVERS)
        mutaded_client = random.randrange(0, AMOUNT_OF_CLIENTS)
        self.modify_client(mutaded_client, mutated_server)

    def __str__(self):
        return f'Найкращий результат = {self.worst_delay:.0f}'

    def __repr__(self):
        return f'Найкращий результат = {self.worst_delay:.0f}'

    def __lt__(self, other):
        return self.worst_delay < other.worst_delay

    def __le__(self, other):
        return self.worst_delay <= other.worst_delay

class Client:
    def __init__(self, x, y):
        self.x = np.float64(x)
        self.y = np.float64(y)

    def __str__(self):
        return f'Клієнт({self.x:.2f}, {self.y:.2f})'

    def __repr__(self):
        return f'({self.x:.2f}, {self.y:.2f})'

class Server:
    def __init__(self, x, y, capacity):
        self.x = np.float64(x)
        self.y = np.float64(y)
        self.capacity = np.int64(capacity)
    def __str__(self):
        return f'Сервер({self.x:.2f}, {self.y:.2f}) - {self.capacity} завантаження'
    def __repr__(self):
        return f'[{self.x:.2f}, {self.y:.2f} -> {self.capacity}]'

def get_rand_coord():
    return random.random() * 100

def create_clients(amount):
    return np.array([Client(get_rand_coord(), get_rand_coord()) for _ in
range(amount)])

def create_servers(amount, capacity_mean, capacity_dev, min_capacity = 1):
    counter = 0
    servers = []
    while counter < amount:
```

```
        generated_capacity = round(random.gauss(capacity_mean, capacity_dev))
        if generated_capacity < min_capacity:
            continue
        servers.append(Server(get_rand_coord(), get_rand_coord(),
generated_capacity))
        counter += 1
    return np.array(servers)

def sum_capacity(servers):
    return sum(i.capacity for i in servers)

def get_distance(server, clients):
    server_coords = np.array([server.x, server.y])
    client_coords = np.array([[client.x, client.y] for client in clients])
    return np.sqrt(np.sum((server_coords - client_coords) ** 2, axis=1))

def evaluate_client_delay(server_indices, client_indices, capacities,
capacity_punishment=3000):
    return distances_squared[server_indices, client_indices] + capacity_punishment
* np.maximum(0, 1 - capacities[server_indices])

def get_best(population):
    return np.min(population)

def get_best_index(population):
    return np.argmin(population)

def get_worst_index(population):
    return np.argmax(population)

def get_rand_bool(chance):
    return random.random() < chance

def except_rand_element(variables, exception):
    result = random.randint(0, len(variables) - 2)
    if result >= exception:
       result += 1
    return variables[result]

def selection(variables):
    best_index = get_best_index(variables)
    best = variables[best_index]
    random = except_rand_element(variables, best_index)
    return best, random

def crossing(parent1, parent2):
    crossover_point = np.random.randint(1, AMOUNT_OF_CLIENTS - 1)
    child1_gene = np.concatenate((parent1.gene[:crossover_point],
parent2.gene[crossover_point:]))
```

```
    child2_gene = np.concatenate((parent2.gene[:crossover_point],
parent1.gene[crossover_point:]))
    children1 = Chromosome()
    children1.init_with_gene(child1_gene)
    children2 = Chromosome()
    children2.init_with_gene(child2_gene)
    return get_best((children1, children2))

def init_population(size, greedy_percent):
    potential_optimum = [Chromosome()]
    potential_optimum[0].generate_greedy_optimal()
    size -= 1

    num_greedy = round(size * greedy_percent)
    num_random = size - num_greedy

    greedy_population = [Chromosome() for _ in range(num_greedy)]
    for i in greedy_population:
        i.generate_greedy()

    random_population = [Chromosome() for _ in range(num_random)]
    for i in random_population:
        i.generate_random()

    return np.array(potential_optimum + greedy_population + random_population)

def genetic(stop_iterations, iterations_limit, mutation_chance, population_size,
greedy_percent=0.01):
    population = init_population(population_size, greedy_percent)
    print('\nВхідна популяція:')
    for i in population:
        print(i)
    print()
    best = get_best(population)
    same_best_counter = 1
    total_counter = 1
    best.show_iteration_info(total_counter)
    while same_best_counter < stop_iterations and total_counter < iterations_limit:
        parents = selection(population)
        children = crossing(parents[0], parents[1])
        children.mutate(mutation_chance)
        worst_index = get_worst_index(population)
        if children.worst_delay < population[worst_index].worst_delay:
            population[worst_index] = children
        if children.worst_delay < best.worst_delay:
            best = children
            same_best_counter = 0
        total_counter += 1
        same_best_counter += 1
        best.show_iteration_info(total_counter)
```

```
    return population
# Вхідні дані
AMOUNT_OF_SERVERS = 3
servers = create_servers(AMOUNT_OF_SERVERS, 10, 5)
print(f'Сервери:\n{servers}')
AMOUNT_OF_CLIENTS = round(sum_capacity(servers) * 0.7)
clients = create_clients(AMOUNT_OF_CLIENTS)
print(f'Клієнти:\n{clients}')
# Необхідні змінні
distances = np.array([get_distance(server, clients) for server in servers])
distances_squared = np.square(distances)
max_capacities = np.array([i.capacity for i in servers])
# Створення та навчання нейронної мережі
genSer = genetic(stop_iterations=50, iterations_limit=1000, mutation_chance=0.2,
population_size=1000)
# Вивід результатів
print('Остання популяція:')
for i in genSer:
    print(i)
best = get_best(genSer)
print('Найкращий результат:')
best.info()

# Вивід результатів графічно
server_x = [server.x for server in servers]
server_y = [server.y for server in servers]
server_capacity = [server.capacity * 3 for server in servers]
client_x = [client.x for client in clients]
client_y = [client.y for client in clients]
for client_index, server_index in enumerate(best.gene):
    plt.plot([clients[client_index].x, servers[server_index].x],
[clients[client_index].y, servers[server_index].y], color='black', linewidth=1)
plt.scatter(server_x, server_y, marker='o', label='Сервери', s=50)
plt.scatter(client_x, client_y, marker='o', label='Клієнти', s=20)
plt.xlim(0, 100)
plt.ylim(0, 100)
plt.title('Побудована оптимізована система')
plt.legend()
plt.show()
```