

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 №5

з дисципліни

«Проектування алгоритмів»

Варіант 1

„Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.2”

Виконав(ла)

III-13 Ал Хадам М.Р.

(шифр, прізвище, ім'я, по батькові)

Перевірив

Сопов О. О.

(прізвище, ім'я, по батькові)

Київ 2023

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ	3
2	ЗАВДАННЯ	4
3	ВИКОНАННЯ	11
3.1	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ	11
3.1.1	<i>Вихідний код</i>	<i>11</i>
3.2	ПРИКЛАДИ РОБОТИ	15
3.3	ТЕСТУВАННЯ АЛГОРИТМУ	16
	ВИСНОВОК	22
	КРИТЕРІЇ ОЦІНЮВАННЯ	23

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи розробки метаевристичних алгоритмів для типових прикладних задач. Опрацювати методологію підбору прийнятних параметрів алгоритму.

2 ЗАВДАННЯ

Згідно варіанту, формалізувати алгоритм вирішення задачі відповідно загальної методології.

Записати розроблений алгоритм у покроковому вигляді. З достатнім ступенем деталізації.

Виконати його програмну реалізацію на будь-якій мові програмування.

Перелік задач наведено у таблиці 2.1.

Перелік алгоритмів і досліджуваних параметрів у таблиці 2.2.

Задача і алгоритм наведені в таблиці 2.3.

Змінюючи параметри алгоритму, визначити кращі вхідні параметри алгоритму. Для цього необхідно:

- обрати критерій зупинки алгоритму (кількість ітерацій або значення ЦФ);
- зафіксувати усі параметри крім одного і змінювати цей параметр, поки не буде досягнуто пікової ефективності;
- після цього параметр фіксується і змінюються інші параметри;
- далі повторюємо процедуру спочатку, з першого зафіксованого параметру;
- зупиняємось коли будуть знайдені оптимальні параметри для даної задачі або встановлена залежність одних параметрів від інших.

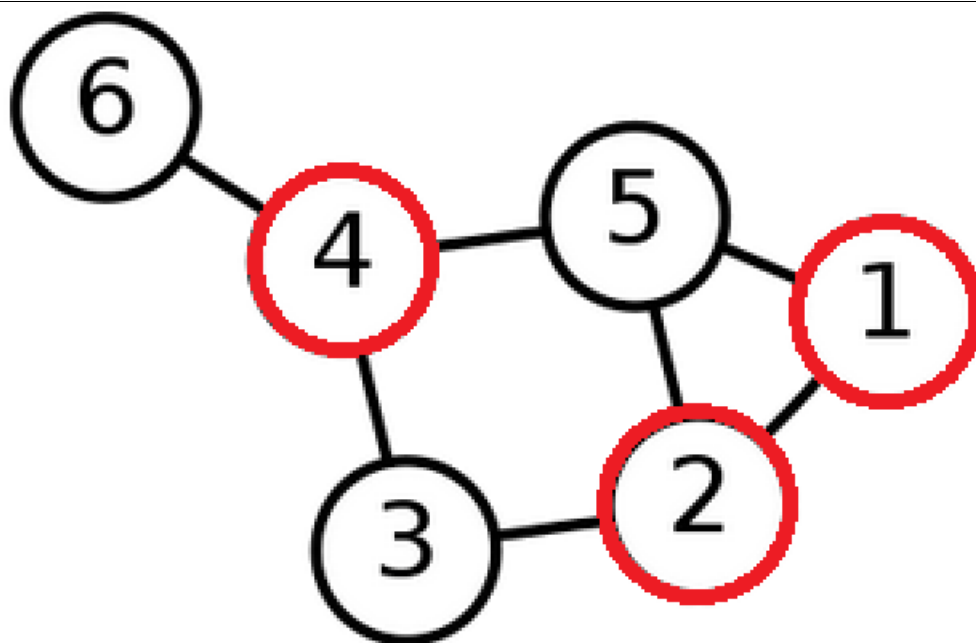
Зробити узагальнений висновок в якому обов'язково описати залежність якості розв'язку від вхідних параметрів.

Таблиця 2.1 – Прикладні задачі

№	Задача
1	Задача про рюкзак (місткість $P=500$, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 20 (випадкова)). Для заданої множини предметів, кожен з яких має вагу і цінність, визначити яку кількість кожного з предметів слід взяти, так, щоб сумарна вага не

	<p>перевищувала задану, а сумарна цінність була максимальною.</p> <p>Задача часто виникає при розподілі ресурсів, коли наявні фінансові обмеження, і вивчається в таких областях, як комбінаторика, інформатика, теорія складності, криптографія, прикладна математика.</p>
2	<p>Задача комівояжера (300 вершин, відстань між вершинами випадкова від 5 до 150) полягає у знаходженні найвигіднішого маршруту, що проходить через вказані міста хоча б по одному разу. В умовах завдання вказуються критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій тощо) і відповідні матриці відстаней, вартості тощо. Зазвичай задано, що маршрут повинен проходити через кожне місто тільки один раз, в такому випадку розв'язок знаходиться серед гамільтонових циклів.</p> <p>Розглядається симетричний, асиметричний та змішаний варіанти.</p> <p>В загальному випадку, асиметрична задача комівояжера відрізняється тим, що ребра між вершинами можуть мати різну вагу в залежності від напрямку, тобто, задача моделюється орієнтованим графом. Таким чином, окрім ваги ребер графа, слід також зважати і на те, в якому напрямку знаходяться ребра.</p> <p>У випадку симетричної задачі всі пари ребер між одними й тими самими вершинами мають однакову вагу.</p> <p>У випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів і напрямку руху.</p> <p>Застосування:</p> <ul style="list-style-type: none"> – доставка товарів (в цьому випадку може бути більш доречна постановка транспортної задачі - доставка в кілька магазинів з декількох складів); – доставка води; – моніторинг об'єктів;

	<ul style="list-style-type: none"> – поповнення банкоматів готівкою; – збір співробітників для доставки вахтовим методом.
3	<p>Розфарбовування графа (300 вершин, степінь вершини не більше 30, але не менше 2) – називають таке приписування кольорів (або натуральних чисел) його вершинам, що ніякі дві суміжні вершини не набувають однакового кольору. Найменшу можливу кількість кольорів у розфарбуванні називають хроматичне число.</p> <p>Застосування:</p> <ul style="list-style-type: none"> – розкладу для освітніх установ; – розкладу в спорті; – планування зустрічей, зборів, інтерв'ю; – розклади транспорту, в тому числі - авіатранспорту; – розкладу для комунальних служб;
4	<p>Задача вершинного покриття (300 вершин, степінь вершини не більше 30, але не менше 2). Вершинне покриття для неорієнтованого графа $G = (V, E)$ - це множина його вершин S, така, що, у кожного ребра графа хоча б один з кінців входить в вершину з S.</p> <p>Задача вершинного покриття полягає в пошуку вершинного покриття найменшого розміру для заданого графа (цей розмір називається числом вершинного покриття графа).</p> <p>На вході: Граф $G = (V, E)$.</p> <p>Результат: множина $C \subseteq V$ - найменше вершинне покриття графа G.</p>



Застосування:

- розміщення пунктів обслуговування;
- призначення екіпажів на транспорт;
- проектування інтегральних схем і конвеєрних ліній.

5	<p>Задача про кліку (300 вершин, степінь вершини не більше 30, але не менше 2). Клікою в неорієнтованому графі називається підмножина вершин, кожні дві з яких з'єднані ребром графа. Іншими словами, це повний підграф первісного графа. Розмір кліки визначається як число вершин в ній.</p> <p>Задача про кліку існує у двох варіантах: у задачі розпізнавання потрібно визначити, чи існує в заданому графі G кліка розміру k, тоді як в обчислювальному варіанті потрібно знайти в заданому графі G кліку максимального розміру або всі максимальні кліки (такі, що не можна збільшити).</p> <p>Застосування:</p> <ul style="list-style-type: none"> – біоінформатика; – електротехніка;
6	<p>Задача про найкоротший шлях (300 вершин, відстань між вершинами випадкова від 5 до 150, степінь вершини не більше 10, але не менше 1) -</p>

	<p>задача пошуку найкоротшого шляху (ланцюга) між двома точками (вершинами) на графі, в якій мінімізується сума ваг ребер, що складають шлях.</p> <p>Важливість задачі визначається її різними практичними застосуваннями. Наприклад, в GPS-навігаторах здійснюється пошук найкоротшого шляху між точкою відправлення і точкою призначення. Як вершин виступають перехрестя, а дороги є ребрами, які лежать між ними. Якщо сума довжин доріг між перехрестями мінімальна, тоді знайдений шлях найкоротший.</p>
--	--

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

№	Алгоритми і досліджувані параметри
1	<p>Генетичний алгоритм:</p> <ul style="list-style-type: none"> - оператор схрещування (мінімум 3); - мутація (мінімум 2); - оператор локального покращення (мінімум 2).
2	<p>Мурашиний алгоритм:</p> <ul style="list-style-type: none"> - α; - β; - ρ; - L_{min}; - кількість мурах M і їх типи (елітні, тощо...); - маршрути з однієї чи різних вершин.
3	<p>Бджолиний алгоритм:</p> <ul style="list-style-type: none"> - кількість ділянок; - кількість бджіл (фуражирів і розвідників).

Таблиця 2.3 – Варіанти задач і алгоритмів

№	Задачі і алгоритми
1	Задача про рюкзак + Генетичний алгоритм
2	Задача про рюкзак + Бджолиний алгоритм
3	Задача комівояжера (асиметрична мережа) + Генетичний алгоритм
4	Задача комівояжера (симетрична мережа) + Генетичний алгоритм
5	Задача комівояжера (змішана мережа) + Генетичний алгоритм
6	Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм
7	Задача комівояжера (симетрична мережа) + Мурашиний алгоритм
8	Задача комівояжера (змішана мережа) + Мурашиний алгоритм
9	Задача вершинного покриття + Генетичний алгоритм
10	Задача вершинного покриття + Бджолиний алгоритм
11	Задача комівояжера (асиметрична мережа) + Бджолиний алгоритм
12	Задача комівояжера (симетрична мережа) + Бджолиний алгоритм
13	Задача комівояжера (змішана мережа) + Бджолиний алгоритм
14	Розфарбовування графа + Генетичний алгоритм
15	Розфарбовування графа + Бджолиний алгоритм
16	Задача про кліку (задача розпізнавання) + Генетичний алгоритм
17	Задача про кліку (задача розпізнавання) + Бджолиний алгоритм
18	Задача про кліку (обчислювальна задача) + Генетичний алгоритм
19	Задача про кліку (обчислювальна задача) + Бджолиний алгоритм
20	Задача про найкоротший шлях + Генетичний алгоритм
21	Задача про найкоротший шлях + Мурашиний алгоритм
22	Задача про найкоротший шлях + Бджолиний алгоритм
23	Задача про рюкзак + Генетичний алгоритм
24	Задача про рюкзак + Бджолиний алгоритм
25	Задача комівояжера (асиметрична мережа) + Генетичний алгоритм
26	Задача комівояжера (симетрична мережа) + Генетичний алгоритм
27	Задача комівояжера (змішана мережа) + Генетичний алгоритм

28	Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм
29	Задача комівояжера (симетрична мережа) + Мурашиний алгоритм
30	Задача комівояжера (змішана мережа) + Мурашиний алгоритм

3 ВИКОНАННЯ

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

crossover_rate.py

```
# list of different crossover rates to test
crossover_rates = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
fitness_history_list = []

for rate in crossover_rates:
    _, fitness_history = optimize_with_param(weight, value, initial_population,
    pop_size, num_iterations, knapsack_threshold,
    rate)
    fitness_history_list.append(fitness_history)

# plot the mean fitness of each generation for each crossover rate
for i in range(len(crossover_rates)):
    mean_fitness = [np.mean(fitness_history_list[i][j]) for j in
    range(num_iterations)]
    plt.plot(list(range(num_iterations)), mean_fitness,
    label='Crossover rate = {}'.format(crossover_rates[i]))
    print(crossover_rates[i], '----->', np.mean(mean_fitness), sep=' ')

plt.legend()
plt.title('Effect of Crossover Rate on Mean Fitness')
plt.xlabel('Generation')
plt.ylabel('Mean Fitness')
plt.show()

plt.show()
```

mutation_rate.py

```
# list of different mutation rates to test
mutation_rates = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
fitness_history_list = []
num_iterations = 1000

for rate in mutation_rates:
    _, fitness_history = optimize(weight, value, initial_population, pop_size,
    num_iterations, knapsack_threshold,
    rate)
    fitness_history_list.append(fitness_history)

# plot the mean fitness of each generation for each mutation rate
for i in range(len(mutation_rates)):
    mean_fitness = [np.mean(fitness_history_list[i][j]) for j in
    range(num_iterations)]
    plt.plot(list(range(num_iterations)), mean_fitness, label='Mutation rate =
    {}'.format(mutation_rates[i]))
    print(mutation_rates[i], '----->', np.mean(mean_fitness), sep=' ')

plt.legend()
plt.title('Effect of Mutation Rate on Mean Fitness')
plt.xlabel('Generation')
```

```
plt.ylabel('Mean Fitness')
plt.show()
```

main.py

```
import numpy as np
import random as rd

from random import randint
import matplotlib.pyplot as plt

def cal_fitness(weight, value, population, threshold):
    fitness = np.empty(population.shape[0])
    for i in range(population.shape[0]):
        s1 = np.sum(population[i] * value)
        s2 = np.sum(population[i] * weight)
        if s2 <= threshold:
            fitness[i] = s1
        else:
            fitness[i] = 0
    return fitness.astype(int)

def selection(fitness, num_parents, population):
    fitness = list(fitness)
    parents = np.empty((num_parents, population.shape[1]))
    for i in range(num_parents):
        max_fitness_idx = np.where(fitness == np.max(fitness))
        parents[i, :] = population[max_fitness_idx[0][0], :]
        fitness[max_fitness_idx[0][0]] = -999999
    return parents

def crossover(parents, num_offsprings):
    offsprings = np.empty((num_offsprings, parents.shape[1]))
    crossover_point = int(parents.shape[1] / 2)
    crossover_rate = 0.7
    i = 0
    while parents.shape[0] < num_offsprings:
        parent1_index = i % parents.shape[0]
        parent2_index = (i + 1) % parents.shape[0]
        x = rd.random()
        if x > crossover_rate:
            continue
        parent1_index = i % parents.shape[0]
        parent2_index = (i + 1) % parents.shape[0]
        offsprings[i, 0:crossover_point] = parents[parent1_index,
0:crossover_point]
        offsprings[i, crossover_point:] = parents[parent2_index,
crossover_point:]
        i += 1
    return offsprings

def mutation(offsprings):
    mutants = np.empty((offsprings.shape))
    mutation_rate = 0.05
    for i in range(mutants.shape[0]):
        random_value = rd.random()
        mutants[i, :] = offsprings[i, :]
        if random_value > mutation_rate:
            continue
        int_random_value = randint(0, offsprings.shape[1] - 1)
```

```

        if mutants[i, int_random_value] == 0:
            mutants[i, int_random_value] = 1
        else:
            mutants[i, int_random_value] = 0
    return mutants

def optimize(weight, value, population, pop_size, num_generations, threshold):
    parameters, fitness_history = [], []
    num_parents = int(pop_size[0] / 2)
    num_offsprings = pop_size[0] - num_parents

    for _ in range(num_generations):
        fitness = cal_fitness(weight, value, population, threshold)
        fitness_history.append(fitness)
        parents = selection(fitness, num_parents, population)
        offsprings = crossover(parents, num_offsprings)
        mutants = mutation(offsprings)
        population[0:parents.shape[0], :] = parents
        population[parents.shape[0]:, :] = mutants

    print('Last generation: \n{}\n'.format(population))

    fitness_last_gen = cal_fitness(weight, value, population, threshold)
    print('Fitness of the last generation: \n{}\n'.format(fitness_last_gen))

    max_fitness = np.where(fitness_last_gen == np.max(fitness_last_gen))
    parameters.append(population[max_fitness[0][0], :])

    return parameters, fitness_history

if __name__ == '__main__':
    items = 100
    knapsack_threshold = 500

    item_index = np.arange(1, items + 1)
    weight = np.random.randint(1, 20, size=items)
    value = np.random.randint(2, 30, size=items)

    print('Generated items with random weight and value:')
    print('Item No.   Weight   Value')
    for i in range(item_index.shape[0]):
        print(f'{item_index[i]}           {weight[i]}           {value[i]}')

    solutions_per_pop = 80
    pop_size = (solutions_per_pop, item_index.shape[0])
    print('Population size = {}'.format(pop_size))

    initial_population = np.random.randint(2, size=pop_size)
    initial_population = initial_population.astype(int)

    num_iterations = 1000
    print('Initial population: \n{}'.format(initial_population))

    parameters, fitness_history = optimize(weight, value, initial_population,
pop_size, num_iterations,
                                         knapsack_threshold)
    print('The optimized parameters for the given inputs are:
\n{}'.format(parameters))
    selected = item_index * parameters # chromosomes

    print('\nSelected items that will maximize the knapsack:')
    total_weight = 0

```

```

total_value = 0
counter = 0
for i in range(selected.shape[1]):
    if selected[0][i] != 0:
        print('{}'.format(selected[0][i]), end=" ")
        total_weight += weight[i]
        total_value += value[i]
        counter += 1

print("\nAmount of selected items: ", counter)
print("Total weight is:", total_weight)
print("Total value is:", total_value)
print("Avg value of selected items is:", total_value / counter)
print(f"Avg value out of {items} generated:", sum(value) / items)

# plot
fitness_history_mean = [np.mean(fitness_history[i]) for i in
range(len(fitness_history)) if i % 20 == 0]
print("Fitness mean is", fitness_history_mean)

fitness_history_max = [np.max(fitness_history[i]) for i in
range(len(fitness_history)) if i % 20 == 0]
print("Fitness max is", fitness_history_max)

fitness_history_min = [np.min(fitness_history[i]) for i in
range(len(fitness_history)) if i % 20 == 0]
print("Fitness min is", fitness_history_min)

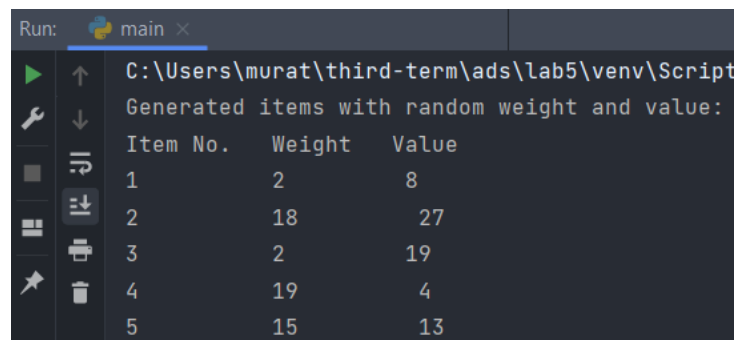
plt.plot(list(range(0, num_iterations, 20)), fitness_history_mean,
label='Mean Fitness')
plt.plot(list(range(0, num_iterations, 20)), fitness_history_max, label='Max
Fitness')
plt.plot(list(range(0, num_iterations, 20)), fitness_history_min, label='Min
Fitness')

plt.legend()
plt.title('Fitness through the iterations')
plt.xlabel('Iteration')
plt.ylabel('Fitness')

```

3.2 Приклади роботи

На рисунку 3.1 показаний вигляд згенерованих випадковим чином 100 одиниць рюкзаку.

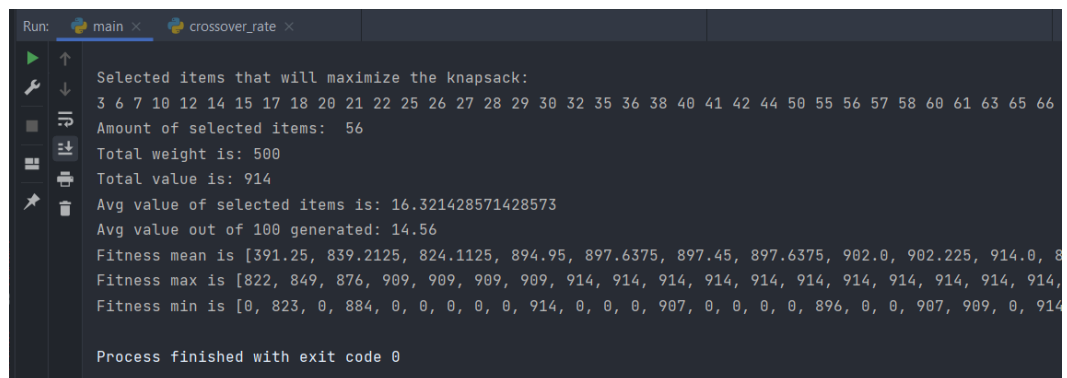


The screenshot shows a terminal window with a dark background. The title bar indicates the file path: C:\Users\murat\third-term\ads\lab5\venv\Script. The output text reads: "Generated items with random weight and value:". Below this, a table lists 5 items with their respective numbers, weights, and values.

Item No.	Weight	Value
1	2	8
2	18	27
3	2	19
4	19	4
5	15	13

Рисунок 3.1 – згенеровані дані

На рисунку 3.2 показано приклад роботи програми.



The screenshot shows a terminal window with a dark background. The title bar indicates the file path: C:\Users\murat\third-term\ads\lab5\venv\Script. The output text reads: "Selected items that will maximize the knapsack:". Below this, a list of selected item numbers is shown, followed by summary statistics and fitness values.

Selected items that will maximize the knapsack:
3 6 7 10 12 14 15 17 18 20 21 22 25 26 27 28 29 30 32 35 36 38 40 41 42 44 50 55 56 57 58 60 61 63 65 66
Amount of selected items: 56
Total weight is: 500
Total value is: 914
Avg value of selected items is: 16.321428571428573
Avg value out of 100 generated: 14.56
Fitness mean is [391.25, 839.2125, 824.1125, 894.95, 897.6375, 897.45, 897.6375, 902.0, 902.225, 914.0, 8
Fitness max is [822, 849, 876, 909, 909, 909, 909, 914, 914, 914, 914, 914, 914, 914, 914, 914, 914, 914,
Fitness min is [0, 823, 0, 884, 0, 0, 0, 0, 0, 914, 0, 0, 0, 907, 0, 0, 0, 896, 0, 0, 907, 909, 0, 914,
Process finished with exit code 0

Рисунок 3.2 – виконання алгоритму при 1000 ітерацій

3.3 Тестування алгоритму

3.3.1 Порівняння відсотку мутації.

Важко сказати, яка швидкість мутації є найкращою, виходячи лише з загальної постановки питання, оскільки найкраща швидкість мутації залежатиме від специфіки проблеми та популяції, яку ви використовуєте. Однак, запустивши роботу алгоритму з різними сценаріями і проаналізувавши результати, ви можете визначити, яка швидкість мутації найкраще підходить для вашого конкретного випадку використання.

Сценарій запускає генетичний алгоритм кілька разів з різною частотою мутації, а потім для кожного циклу будує графік середньої придатності популяції в часі. Порівнюючи графіки, ви можете шукати пробіжку з найкрутішим збільшенням середньої пристосованості, що вказуватиме на найшвидшу швидкість конвергенції.

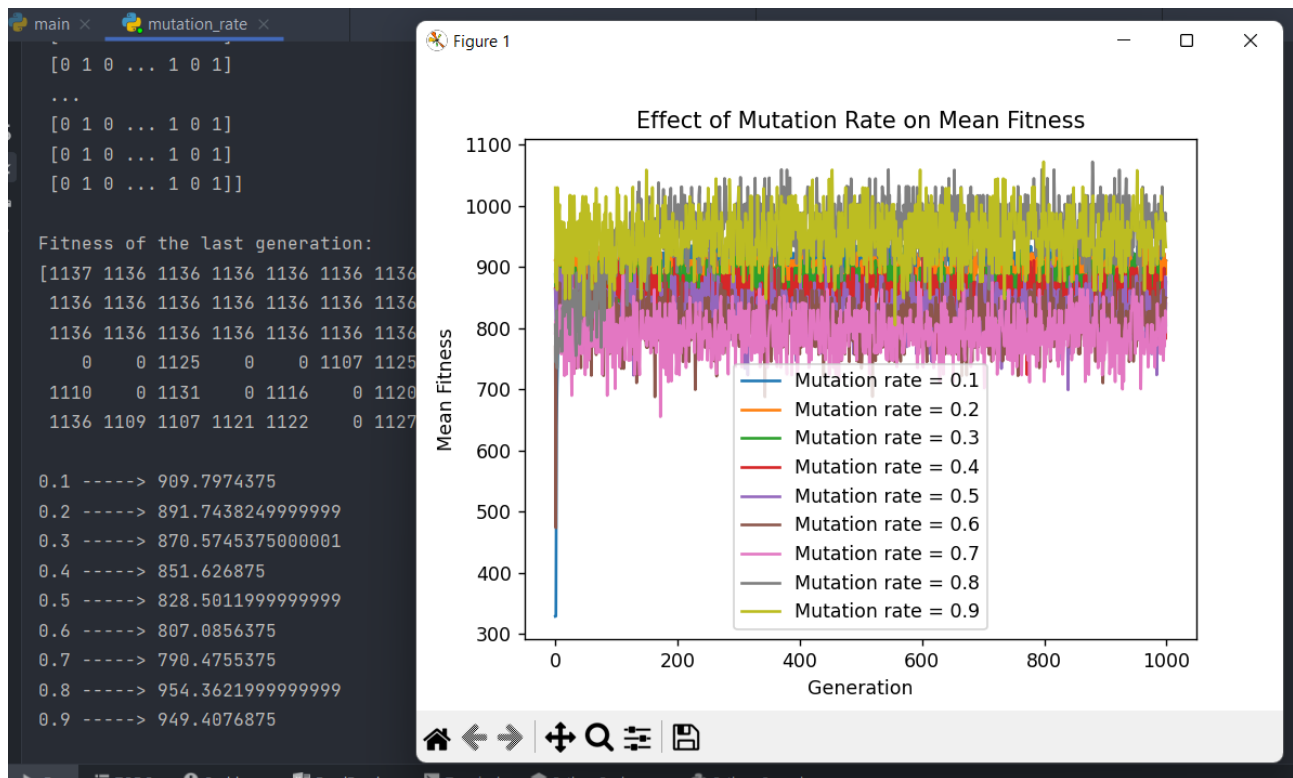
Ми також можемо спостерігати за графіками та бачити, як швидкість мутації впливає на придатність популяції з часом, що може допомогти вам зрозуміти, як різні частоти мутацій впливають на продуктивність алгоритму.

Загалом, нижча частота мутацій може бути кращою, якщо у вас хороша популяція, і вона вже близька до оптимального рішення. Навпаки, вищий рівень мутації може бути кращим, якщо у вас бідна популяція та далека від оптимального рішення.

Коротше кажучи, ви повинні вибрати швидкість мутації, яка забезпечує найкращий баланс між розвідкою та експлуатацією, висока швидкість мутації призведе до кращого дослідження простору рішень, тоді як низька швидкість мутації призведе до кращого використання вже знайдених хороших рішень.

Проведемо декілька тест-виконань нашого алгоритму для знайдення оптимального рівня.


```
Run: main × mutation_rate ×  
0.1 -----> 953.141975  
0.2 -----> 944.1456999999999  
0.3 -----> -25919.972862499995  
0.4 -----> 903.3060375  
0.5 -----> 883.0504249999999  
0.6 -----> 909.290625  
0.7 -----> 933.67655  
0.8 -----> 912.5268374999999  
0.9 -----> 892.9959  
  
Process finished with exit code 0  
Git Run TODO Problems ExcelReader Terminal Python Packages Python Console  
Externally added files can be added to Git // View Files // Always Add // Don't Ask Again (29 minutes ago)
```



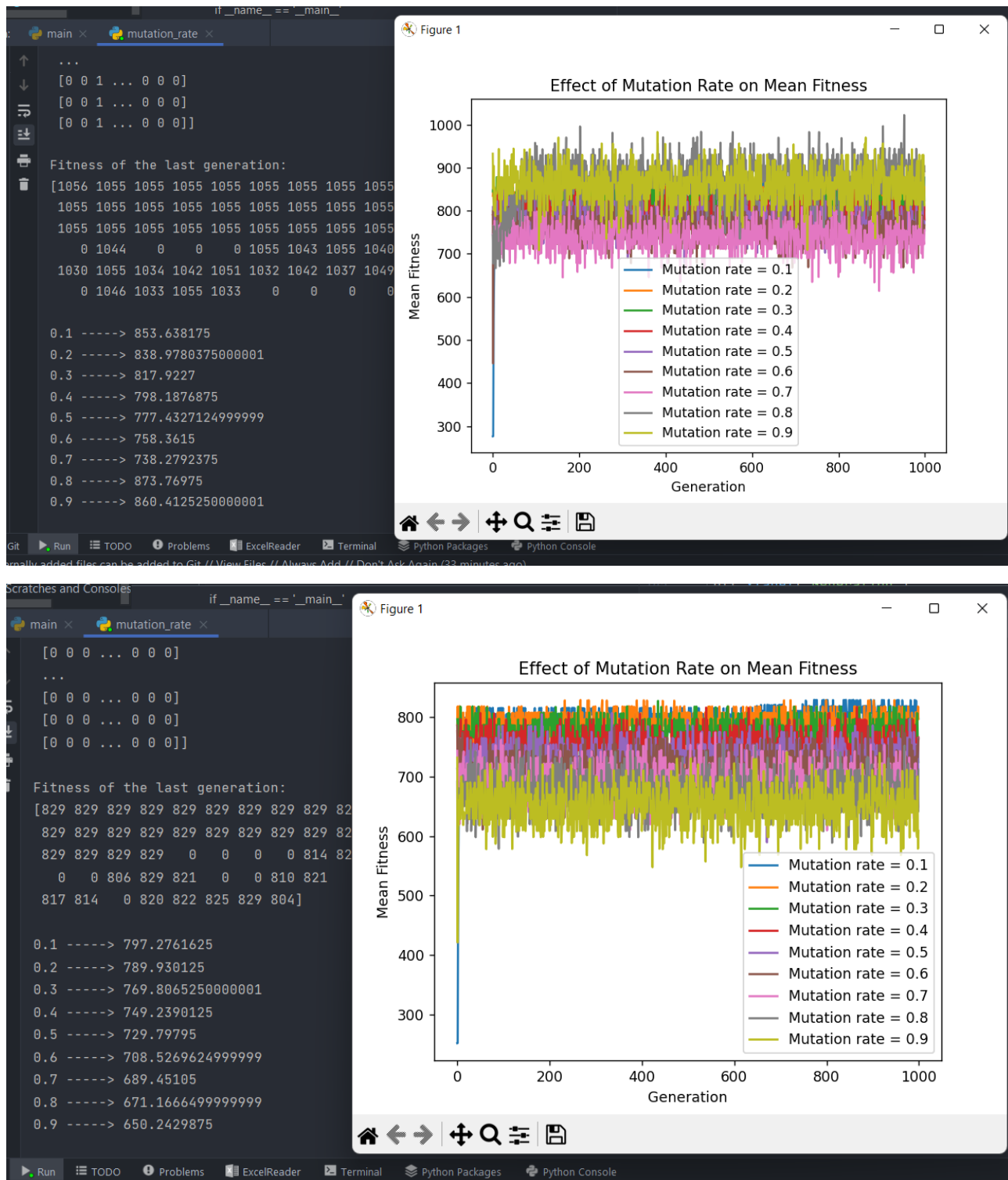


Рисунок 3.3.1 – відповідність частоти мутації та середньому значенню фітнес функції

Отже, звичайно при частоті мутації в +/- 10 відсотків ми буде мати в рази частіше оптимальніший результат, але як було показано на малюнках оптимальні результати нам давали також дуже високі ознаки мутації (80-90 %).

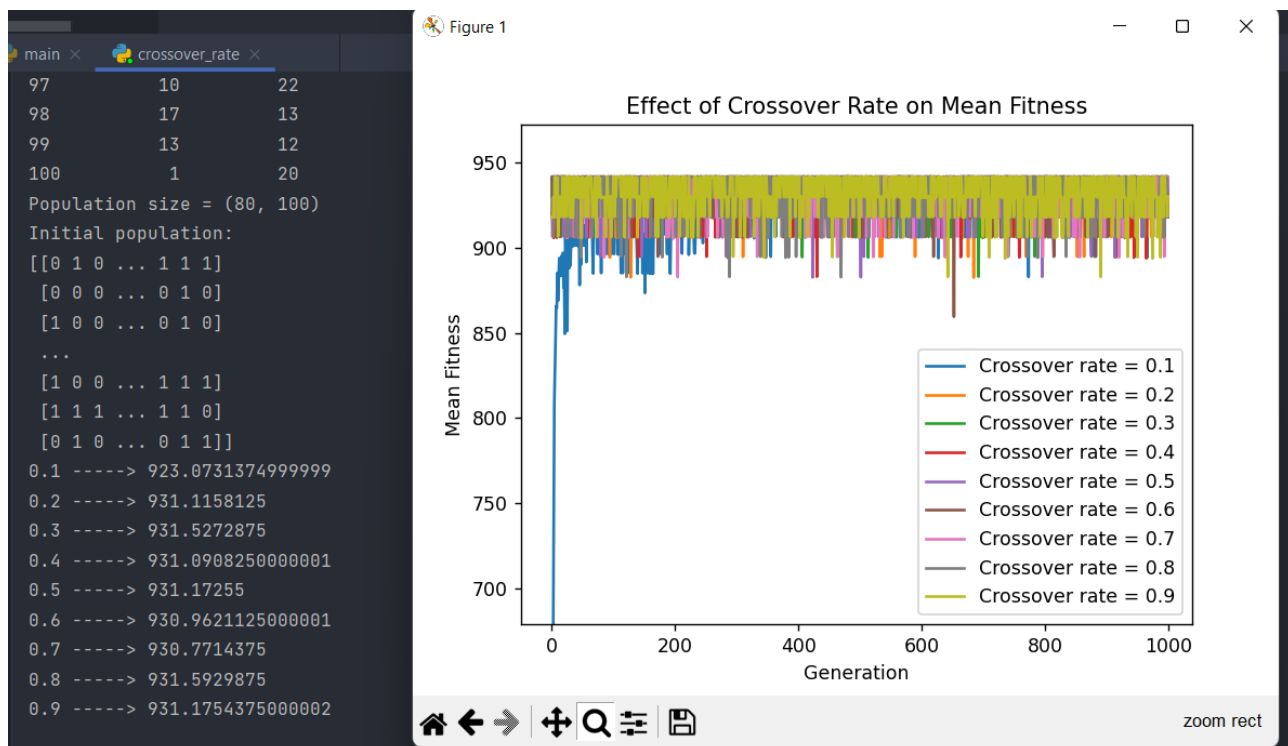
3.3.2 Порівняння відсотку кросинговеру.

Найкращий коефіцієнт кросинговеру для вирішення задачі про рюкзак залежатиме від специфіки проблеми, але хорошою відправною точкою було б спробувати коефіцієнт перехресного переходу приблизно від 0,7 до 0,9. Це пов'язано з тим, що високий рівень кросинговеру сприятиме більшій генетичній різноманітності в популяції, що може допомогти більш ефективно досліджувати простір рішень і знаходити кращі рішення.

У задачі про рюкзак, де мета полягає в тому, щоб максимізувати загальну вартість предметів, які можна включити в рюкзак, залишаючись у межах порогу ваги, наявність високої частоти кросоверу дозволить алгоритму досліджувати різні комбінації предметів і знаходити найкраще рішення .

Швидкість кросинговеру також можна регулювати на основі різноманітності популяції. Якщо популяція досить різноманітна, ви можете спробувати знизити частоту кросинговеру, щоб уникнути втрати хороших рішень.

Варто також зазначити, що оптимальна швидкість кросинговеру може змінюватися залежно від специфіки проблеми, чисельності населення та інших параметрів. Тому поекспериментуємо з різними частотами кросинговеру, щоб знайти найкращу для певного незалежного випадку використання.



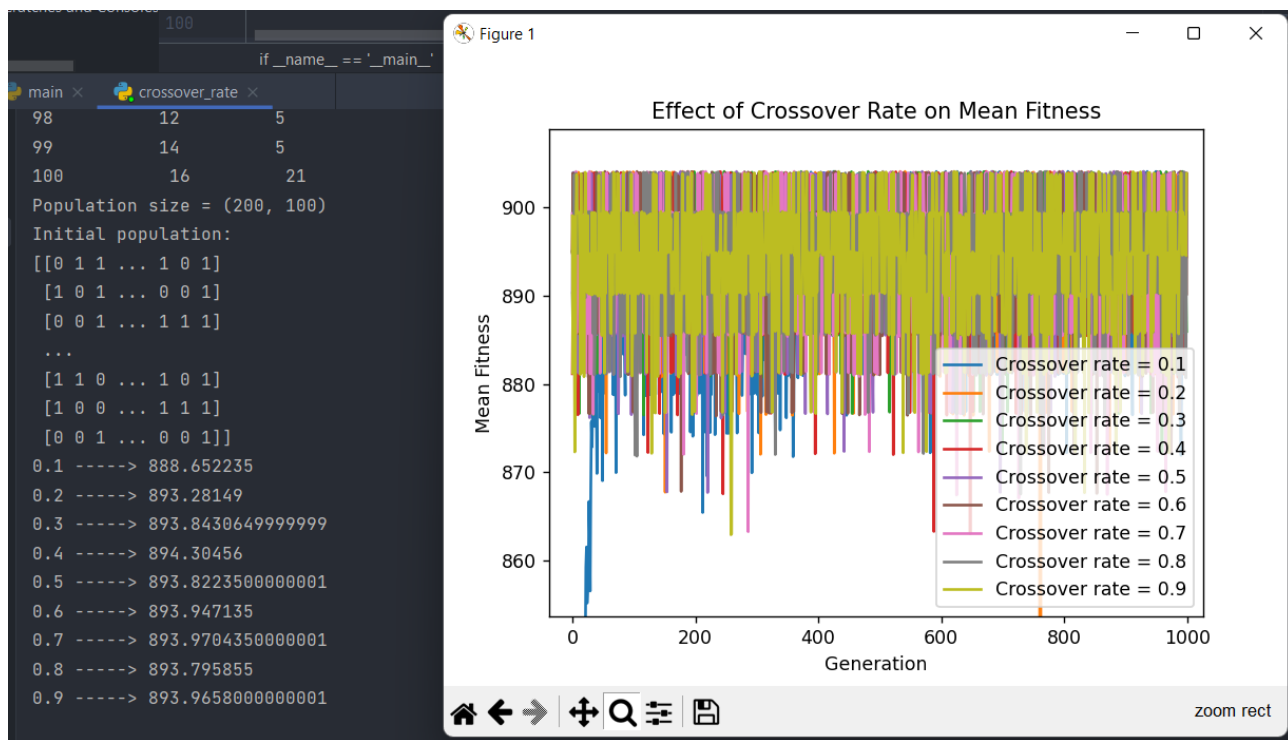


Рисунок 3.3.2 – відповідність частоти мутації та середньому значенню фітнес функції

При виконанні повторних виконань алгоритму, безумовно, все залежало від створенної популяції, але саме максимальне значення нашої цільової функції в більшості випадків було знайдено при **частоті кросингвера від 0.5 до 0.7**. При малих значень кросоверу популяція була недостатньо різноманітна, що не давала їй можливості досягнути максимуму цільової функції. Під час виконання роботи були побудовані статистичні залежності цільової функції від вхідних параметрів, присутній змістовний опис досліджень.

ВИСНОВОК

В рамках даної лабораторної роботи я проаналізував залежність знаходження максимального значення цільової функції проблеми про рюкзак. Було досліджено на двох основних параметрах: `crossover_rate`, `mutation_rate`. Треба розуміти що на якість знайденого розв'язку впливає ще й безліч інших параметрів та факторів.

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 11.12.2022 включно максимальний бал дорівнює – 5. Після 11.12.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- покроковий алгоритм – 15%;
- програмна реалізація алгоритму – 50%;
- тестування алгоритму – 30%;
- висновок – 5%.