

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

Варіант 1

з лабораторної роботи №4 з дисципліни
«Проектування алгоритмів»

“Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.1”

Виконав(ла)

Ал Хадам Мурат Реззович ІП-13
(шифр, прізвище, ім'я, по батькові)

Перевірив

Сопов О. О.
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ	3
2	ЗАВДАННЯ	4
3	ВИКОНАННЯ	10
3.1	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ	10
3.1.1	<i>Вихідний код</i>	<i>10</i>
3.1.2	<i>Приклади роботи</i>	<i>12</i>
3.2	ТЕСТУВАННЯ АЛГОРИТМУ	16
3.2.1	<i>Значення цільової функції зі збільшенням кількості ітерацій..</i>	<i>16</i>
3.2.2	<i>Графіки залежності розв'язку від числа ітерацій</i>	<i>17</i>
	ВИСНОВОК	18
	КРИТЕРІЇ ОЦІНЮВАННЯ	19

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи формалізації метаевристичних алгоритмів і вирішення типових задач з їхньою допомогою.

2 ЗАВДАННЯ

Згідно варіанту, розробити алгоритм вирішення задачі і виконати його програмну реалізацію на будь-якій мові програмування.

Задача, алгоритм і його параметри наведені в таблиці 2.1.

Зафіксувати якість отриманого розв'язку (значення цільової функції) після кожних 20 ітерацій до 1000 і побудувати графік залежності якості розв'язку від числа ітерацій.

Зробити узагальнений висновок.

Таблиця 2.1 – Варіанти алгоритмів

№	Задача і алгоритм
1	Задача про рюкзак (місткість $P=250$, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
2	Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0,4$, L_{\min} знайти жадібним алгоритмом, кількість мурах $M = 30$, починають маршрут в різних випадкових вершинах).
3	Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники).
4	Задача про рюкзак (місткість $P=200$, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.

5	Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 3$, $\rho = 0,4$, L_{\min} знайти жадібним алгоритмом, кількість мурах $M = 35$, починають маршрут в різних випадкових вершинах).
6	Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники).
7	Задача про рюкзак (місткість $P=150$, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
8	Задача комівояжера (200 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм ($\alpha = 3$, $\beta = 2$, $\rho = 0,3$, L_{\min} знайти жадібним алгоритмом, кількість мурах $M = 45$, починають маршрут в різних випадкових вершинах).
9	Задача розфарбовування графу (150 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 25 із них 3 розвідники).
10	Задача про рюкзак (місткість $P=150$, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
11	Задача комівояжера (250 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, ρ

	$= 0,6$, L_{\min} знайти жадібним алгоритмом, кількість мурах $M = 45$, починають маршрут в різних випадкових вершинах).
12	Задача розфарбовування графу (300 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 60 із них 5 розвідники).
13	Задача про рюкзак (місткість $P=250$, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий 30% і 70%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
14	Задача комівояжера (250 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм ($\alpha = 4$, $\beta = 2$, $\rho = 0,3$, L_{\min} знайти жадібним алгоритмом, кількість мурах $M = 45$ (10 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах).
15	Задача розфарбовування графу (100 вершин, степінь вершини не більше 20, але не менше 1), класичний бджолиний алгоритм (число бджіл 30 із них 3 розвідники).
16	Задача про рюкзак (місткість $P=250$, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий 30%, 40% і 30%, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
17	Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0,7$, L_{\min} знайти жадібним алгоритмом, кількість мурах $M = 45$ (15 з них дикі,

	обирають випадкові напрямки), починають маршрут в різних випадкових вершинах).
18	Задача розфарбовування графу (300 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 60 із них 5 розвідники).
19	Задача про рюкзак (місткість $P=250$, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
20	Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм ($\alpha = 3$, $\beta = 2$, $\rho = 0,7$, L_{\min} знайти жадібним алгоритмом, кількість мурах $M = 45$ (10 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах).
21	Задача розфарбовування графу (200 вершин, степінь вершини не більше 30, але не менше 1), класичний бджолиний алгоритм (число бджіл 40 із них 2 розвідники).
22	Задача про рюкзак (місткість $P=250$, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
23	Задача комівояжера (300 вершин, відстань між вершинами випадкова від 1 до 60), мурашиний алгоритм ($\alpha = 3$, $\beta = 2$, $\rho = 0,6$, L_{\min} знайти жадібним алгоритмом, кількість мурах $M = 45$ (15 з них елітні,

	подвійний феромон), починають маршрут в різних випадкових вершинах).
24	Задача розфарбовування графу (400 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 70 із них 10 розвідники).
25	Задача про рюкзак (місткість $P=250$, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
26	Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0,4$, L_{\min} знайти жадібним алгоритмом, кількість мурах $M = 30$, починають маршрут в різних випадкових вершинах).
27	Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники).
28	Задача про рюкзак (місткість $P=200$, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
29	Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 3$, $\rho = 0,4$, L_{\min} знайти жадібним алгоритмом, кількість мурах $M = 35$, починають маршрут в різних випадкових вершинах).

30	Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники).
31	Задача про рюкзак (місткість $P=250$, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
32	Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0,4$, L_{\min} знайти жадібним алгоритмом, кількість мурах $M = 30$, починають маршрут в різних випадкових вершинах).
33	Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники).
34	Задача про рюкзак (місткість $P=200$, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
35	Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 3$, $\rho = 0,4$, L_{\min} знайти жадібним алгоритмом, кількість мурах $M = 35$, починають маршрут в різних випадкових вершинах).

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

```

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 = 0.8
    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
        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 = 0.1
    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 = 250

    item_index = np.arange(1, items + 1)
    weight = np.random.randint(1, 10, size=items)
    value = np.random.randint(1, 20, 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 = 100
    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)

    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')

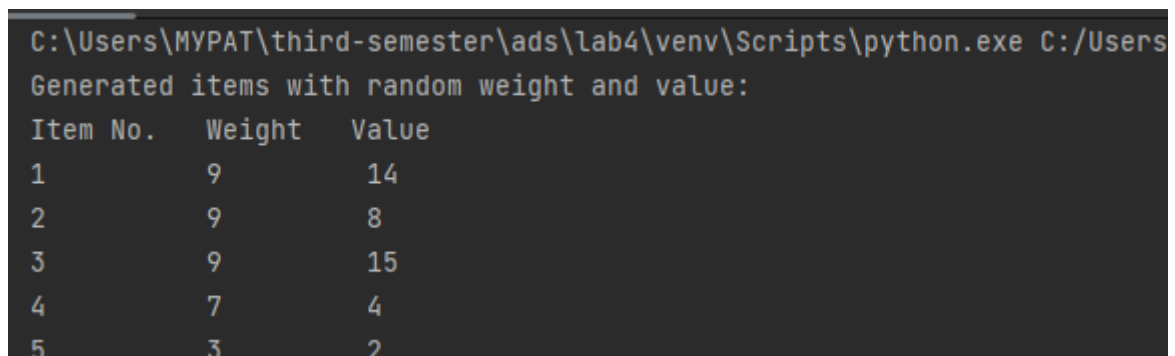
    plt.legend()
    plt.title('Fitness through the generations')
    plt.xlabel('Generations')
    plt.ylabel('Fitness')

    plt.show()

```

3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.



```

C:\Users\MYPAT\third-semester\ads\lab4\venv\Scripts\python.exe C:/Users
Generated items with random weight and value:

```

Item No.	Weight	Value
1	9	14
2	9	8
3	9	15
4	7	4
5	3	2

.....

```

97          1          8
98          5          5
99          4         11
100         7          2
Population size = (100, 100)
Initial population:
[[0 0 0 ... 0 0 0]
 [0 1 1 ... 1 0 1]
 [1 0 0 ... 0 1 1]
 ...
 [0 1 1 ... 0 0 1]
 [1 1 1 ... 1 1 0]
 [0 0 0 ... 1 1 0]]
Last generation:
[[0 0 1 ... 1 0 1]
 [0 0 1 ... 1 0 1]
 [0 0 1 ... 1 0 1]
 ...
 [0 0 1 ... 1 0 1]
 [0 0 1 ... 1 0 1]
 [0 0 1 ... 1 0 1]]
Fitness of the last generation:
[635 635 635 635 635 635 635 635 635 635 635 635 635 635 635 635 635 635
 635 635 635 635 635 635 635 635 635 635 635 635 635 635 635 635 635 635
 635 635 635 635 635 635 635 635 635 635 635 635 635 635 635 635 635 635
   0 635 635 635 635 635 635 635 635 635 635 635 635 635 635 635 635 635
 635 635 635 635 635 635 635 635 635 635 635 635   0 635 635 635 635 635
 635 635 635 635 616 618 635   0 635 635]
```

```

The optimized parameters for the given inputs are:
[array([0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0,
        0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0,
        1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0,
        0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1]))]

Selected items that will maximize the knapsack:
3 6 9 12 14 15 19 24 25 27 28 30 32 33 35 36 37 39 40 43 44 45 47 50 52 54 55 56 5
Amount of selected items: 53
Total weight is: 250
Total value is: 635
Avg value of selected items is: 11.981132075471699
Avg value out of 100 generated: 10.54
```

Рисунок 3.1 – робота програми для 100 ітерацій

```
main X
C:\Users\MYPAT\third-semester\ads\lab4\venv\Scripts\python.exe C:/U
Generated items with random weight and value:
Item No.  Weight  Value
1         6      14
2         5      19
3         2       1
4         9       7
```

```
97         6       6
98         3      11
99         7      13
100        1      16
Population size = (100, 100)
Initial population:
[[0 1 1 ... 1 0 0]
 [0 0 1 ... 0 0 0]
 [0 0 0 ... 0 1 0]
 ...
 [0 1 1 ... 1 0 0]
 [1 1 0 ... 1 1 1]
 [0 1 1 ... 0 1 0]]
Last generation:
[[0 1 1 ... 1 1 1]
 [0 0 1 ... 1 1 1]
 [0 0 1 ... 1 1 1]
 ...
 [0 0 1 ... 1 1 1]
 [0 0 1 ... 1 1 1]
 [0 0 1 ... 1 1 1]]
Fitness of the last generation:
[531 530 530 530 530 530 530 530 530 530 530 530 530 530 530 530 530 530 530 530
 530 530 530 530 530 530 530 530 530 530 530 530 530 530 530 530 530 530 530
 530 530 530 530 530 530 530 530 530 530 530 530 530 530 512 530 530 530
 530 525 530 530 530 530 530 530 522 530 530 530 530 530 530 530 530 530 530
 530 530 530 530 530 530 530 530 530 0 530 530 519 530 530 530 530 530
 530 0 530 530 530 530 521 530 530 530]
```


3.2 Тестування алгоритму

3.2.1 Значення цільової функції зі збільшенням кількості ітерацій

У таблиці 3.1 наведено значення цільової функції зі збільшенням кількості ітерацій.

Ітерації	Середнє значення цільової функції
0	323.52
20	501.4
40	522.58
60	517.19
80	522.32
100	502.15
120	517.96
140	512.08
160	512.83
180	512.8
200	522.96
.....	
860	507.67
880	513.67
900	518.68
920	518.57
940	524.11
960	513.8
980	513.53
1000	513.52

3.2.2 Графіки залежності розв'язку від числа ітерацій

На рисунку 3.3 наведений графік, який показує якість отриманого розв'язку.

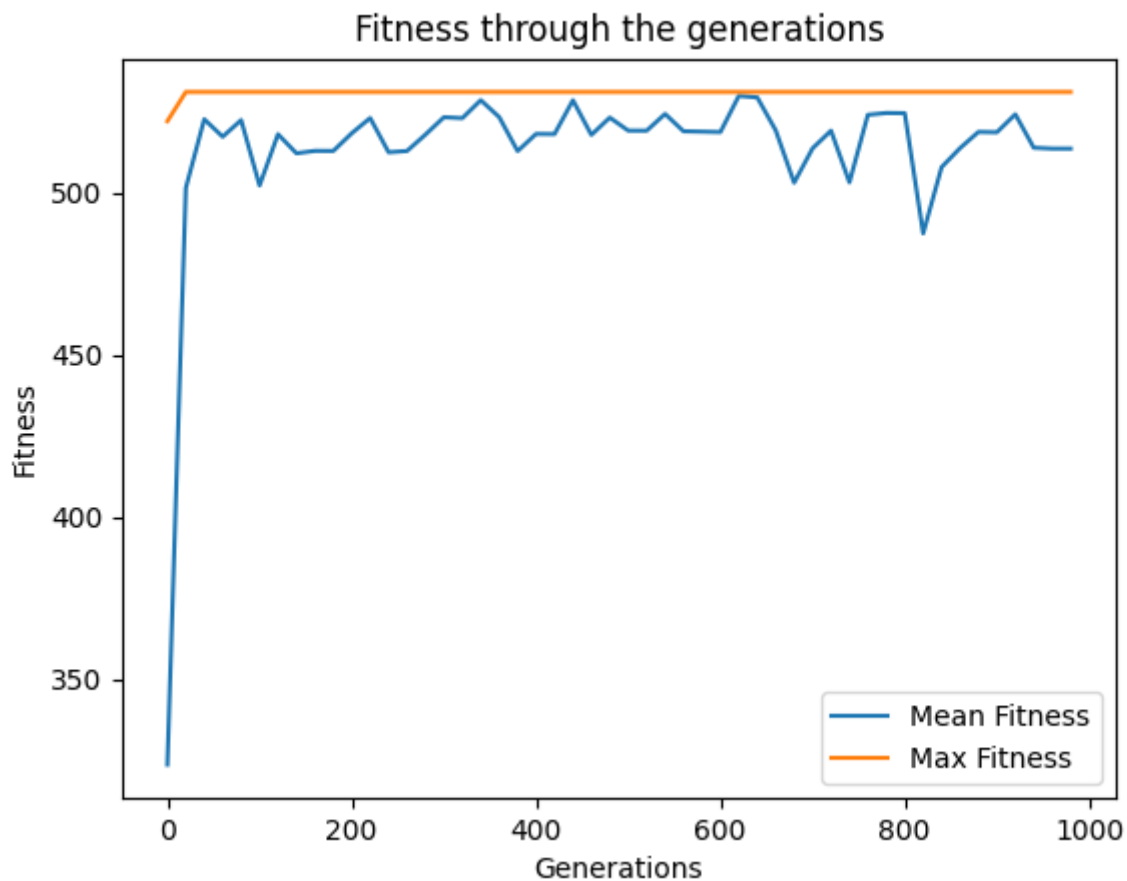


Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

ВИСНОВОК

В рамках даної лабораторної роботи я реалізував генетичний алгоритм на прикладі задачі про рюкзак. Генетичний алгоритм в умовах даної задачі складався з вирахування цільової функції, вибірки генів для подальшого кросинговеру, сам кросинговер безпосередньо та можливі мутації генів з ймовірністю 5 відсотків. Суть мутації полягає в тому, що значення хромосоми буде бінарно зміненим на протилежний ($1 \Rightarrow 0 \mid 0 \Rightarrow 1$). Було проаналізовано алгоритм при різних значеннях вхідних параметрів та побудовано графік залежності значення цільової функції від кількості ітерацій.

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 27.11.2021 включно максимальний бал дорівнює – 5. Після 27.11.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- програмна реалізація алгоритму – 75%;
- тестування алгоритму – 20%;
- висновок – 5%.