**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 № 4 з дисципліни

«Проектування алгоритмів»

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

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-12 Шоман Данило*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов Олексій Олександрович*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc51260917)

[2 Завдання 4](#_Toc51260918)

[3 Виконання 10](#_Toc51260919)

[3.1 Програмна реалізація алгоритму 10](#_Toc51260920)

[3.1.1 Вихідний код 10](#_Toc51260921)

[3.1.2 Приклади роботи 10](#_Toc51260922)

[3.2 Тестування алгоритму 11](#_Toc51260923)

[3.2.1 Значення цільової функції зі збільшенням кількості ітерацій 11](#_Toc51260924)

[3.2.2 Графіки залежності розв'язку від числа ітерацій 11](#_Toc51260925)

[Висновок 12](#_Toc51260926)

[Критерії оцінювання 13](#_Toc51260927)

# Мета лабораторної роботи

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

# 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 (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 (α = 3, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 (α = 2, β = 4, ρ = 0,6, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 (α = 4, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 (α = 2, β = 4, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 (α = 3, β = 2, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 (α = 3, β = 2, ρ = 0,6, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |

# Виконання

## Програмна реалізація алгоритму

### Вихідний код

import random  
import numpy as np  
import matplotlib.pyplot as plt  
  
  
class Item:  
 def \_\_init\_\_(self, cost, weight):  
 self.cost = cost  
 self.weight = weight  
  
  
def init(population\_size, lenth\_of\_chromosome):  
 population = []  
 for i in range(population\_size):  
 population\_piece = '0' \* i  
 population\_piece += '1'  
 population\_piece += '0' \* (lenth\_of\_chromosome - i)  
 population.append(population\_piece)  
 return population  
  
  
def decode(chromosome, length, stuff, restriction):  
 taken\_stuff = []  
 total\_weight = 0  
 total\_cost = 0  
 for i in range(length):  
 if chromosome[i] == '1':  
 if total\_weight+stuff[i].weight <= restriction:  
 total\_weight += stuff[i].weight  
 total\_cost += stuff[i].cost  
 taken\_stuff.append(i)  
 else:  
 break  
 return total\_cost, taken\_stuff  
  
  
def fitness(population, length, stuff, restriction):  
 stuff\_cost = []  
 taken\_stuff = []  
 for i in range(len(population)):  
 total\_cost, taken\_stuff = decode(population[i], length, stuff, restriction)  
 stuff\_cost.append(total\_cost)  
 return stuff\_cost, taken\_stuff  
  
  
def randomization(population, stuff\_cost, size\_of\_population):  
 fitness\_sum = []  
 stuff\_cost\_sum = sum(stuff\_cost)  
 fitness = [i/stuff\_cost\_sum for i in stuff\_cost]  
 for i in range(len(population)):  
 if i == 0:  
 fitness\_sum.append(fitness[i])  
 else:  
 fitness\_sum.append(fitness\_sum[i-1]+fitness[i])  
 new\_population = []  
 for j in range(size\_of\_population):  
 r = np.random.uniform(0, 1)  
 for i in range(len(fitness\_sum)):  
 if i == 0:  
 if 0 <= r <= fitness\_sum[i]:  
 new\_population.append(population[i])  
 else:  
 if fitness\_sum[i-1] <= r <= fitness\_sum[i]:  
 new\_population.append(population[i])  
 return new\_population  
  
  
def crossover(population):  
 mother = population[:int(len(population)/2)]  
 father = population[int(len(population)/2):]  
 np.random.shuffle(mother)  
 np.random.shuffle(father)  
 offspring = []  
 for i in range(int(len(population)/2)):  
 r = np.random.uniform(0, 1)  
 mother\_offspring = mother[i][:50]+father[i][50:]  
 father\_offspring = father[i][:50]+mother[i][50:]  
 offspring.append(mother\_offspring)  
 offspring.append(father\_offspring)  
 return offspring  
  
  
def mutation(offspring, mutation\_chance):  
 for i in range(len(offspring)):  
 r = np.random.uniform(0, 1)  
 if r <= mutation\_chance:  
 point = np.random.randint(0, len(offspring[i]))  
 if not point:  
 if offspring[i][point] == '1':  
 offspring[i] = '0'+offspring[i][1:]  
 else:  
 offspring[i] = '1'+offspring[i][1:]  
 else:  
 if offspring[i][point] == '1':  
 offspring[i] = offspring[i][:(point-1)]+'0'+offspring[i][point:]  
 else:  
 offspring[i] = offspring[i][:(point-1)]+'1'+offspring[i][point:]  
 return offspring  
  
  
def local\_improve(offspring, stuff, capacity, arranged\_stuff):  
 for gen in range(len(offspring)):  
 weight = 0  
 for chromosome in range(len(offspring[gen])):  
 if offspring[gen][chromosome] == '1':  
 weight += stuff[chromosome].weight  
 if weight <= capacity:  
 for item in range(len(arranged\_stuff)):  
 if offspring[gen][arranged\_stuff[item][1]] == '0':  
 if weight + stuff[arranged\_stuff[item][1]].weight <= capacity:  
 weight += stuff[arranged\_stuff[item][1]].weight  
 if not arranged\_stuff[item][1]:  
 offspring[gen] = '1' + offspring[gen][1:]  
 else:  
 offspring[gen] = offspring[gen][:(arranged\_stuff[item][1] - 1)] + '1' + offspring[gen][arranged\_stuff[item][1]:]  
 break  
 return offspring  
  
  
def arrange\_stuff(stuff):  
 arranged\_stuff = []  
 for i in range(len(stuff)):  
 arranged\_stuff.append((stuff[i].cost/stuff[i].weight, i))  
 arranged\_stuff.sort(reverse=True)  
 return arranged\_stuff  
  
  
def main():  
 num\_of\_iterations = 1000  
 mutation\_chance = 0.1  
 population\_size = 100  
 length\_of\_chromosome = 100  
 stuff = []  
 for item in range(length\_of\_chromosome):  
 stuff.append(Item(random.randint(2, 20), random.randint(1, 10)))  
 arranged\_stuff = arrange\_stuff(stuff)  
 capacity = 250  
 population = init(population\_size, length\_of\_chromosome)  
 t = []  
 best = []  
 stuff\_cost\_temp = []  
 for i in range(num\_of\_iterations):  
 if i % 20 == 0 and i > 0:  
 print(f"Best value at the moment - {max(stuff\_cost\_temp)}, on {i} iteration")  
 offspring = crossover(population)  
 offspring = mutation(offspring, mutation\_chance)  
 offspring = local\_improve(offspring, stuff, capacity, arranged\_stuff)  
 union = population+offspring  
 stuff\_cost, taken\_union\_stuff = fitness(union, length\_of\_chromosome, stuff, capacity)  
 population = randomization(union, stuff\_cost, population\_size)  
 stuff\_cost\_temp, taken\_stuff = fitness(population, length\_of\_chromosome, stuff, capacity)  
 h = stuff\_cost\_temp.index(max(stuff\_cost\_temp))  
 t.append(max(stuff\_cost\_temp))  
 best.append(population[h])  
 total\_cost, taken\_stuff\_1 = decode(best[t.index(max(t))], length\_of\_chromosome, stuff, capacity)  
 cost, weight = 0, 0  
 for i in taken\_stuff\_1:  
 cost += stuff[i].cost  
 weight += stuff[i].weight  
 print(f"Best combination:\n{taken\_stuff\_1}\nCost: {cost}, Weight: {weight}")  
 print(f"First appeared on {t.index(max(t))} iteration")  
 plt.plot(t)  
 plt.xlabel('Iterations')  
 plt.ylabel('Quality')  
 plt.title('Dependence of quality on amount of iterations')  
 plt.show()  
  
  
main()

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

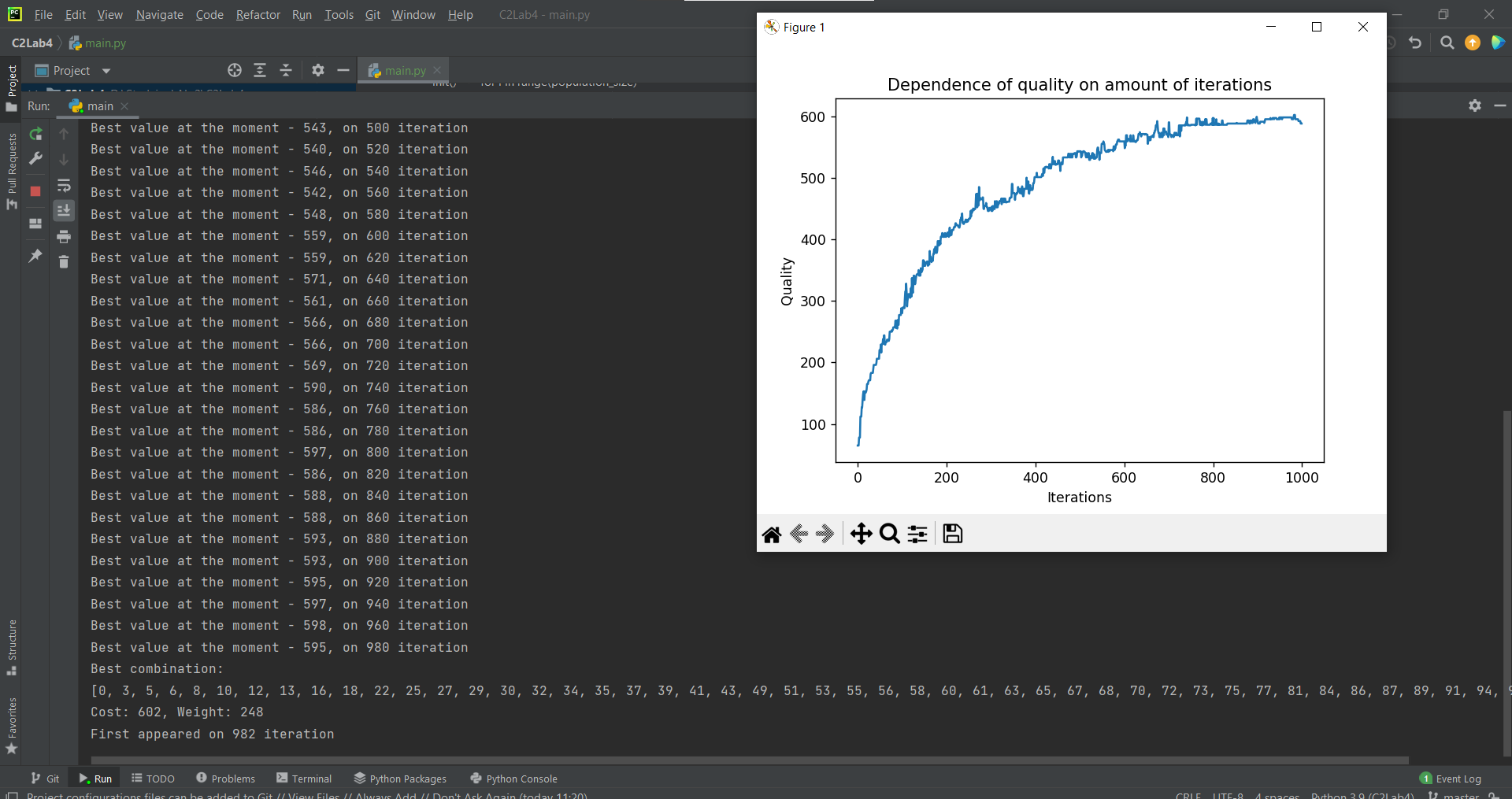


Рисунок 3.1 –

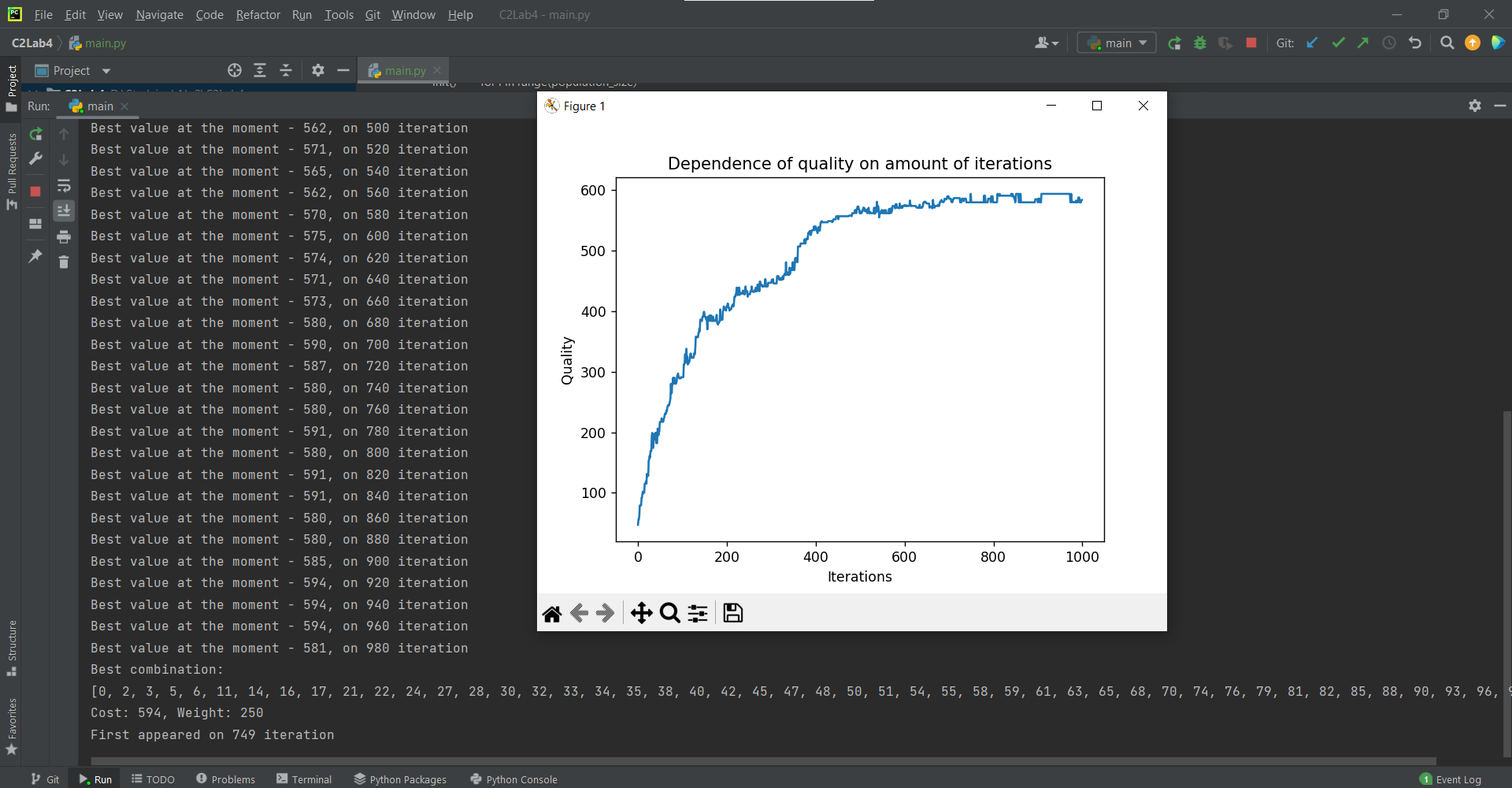


Рисунок 3.2 –

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

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

Значення цільової функції - 115, на 20 ітерації

Значення цільової функції - 199, на 40 ітерації

Значення цільової функції - 224, на 60 ітерації

Значення цільової функції - 290, на 80 ітерації

Значення цільової функції - 291, на 100 ітерації

Значення цільової функції - 330, на 120 ітерації

Значення цільової функції - 367, на 140 ітерації

Значення цільової функції - 391, на 160 ітерації

Значення цільової функції - 381, на 180 ітерації

Значення цільової функції - 408, на 200 ітерації

Значення цільової функції - 424, на 220 ітерації

Значення цільової функції - 428, на 240 ітерації

Значення цільової функції - 432, на 260 ітерації

Значення цільової функції - 441, на 280 ітерації

Значення цільової функції - 447, на 300 ітерації

Значення цільової функції - 452, на 320 ітерації

Значення цільової функції - 464, на 340 ітерації

Значення цільової функції - 481, на 360 ітерації

Значення цільової функції - 519, на 380 ітерації

Значення цільової функції - 540, на 400 ітерації

Значення цільової функції - 547, на 420 ітерації

Значення цільової функції - 549, на 440 ітерації

Значення цільової функції - 557, на 460 ітерації

Значення цільової функції - 558, на 480 ітерації

Значення цільової функції - 562, на 500 ітерації

Значення цільової функції - 571, на 520 ітерації

Значення цільової функції - 565, на 540 ітерації

Значення цільової функції - 562, на 560 ітерації

Значення цільової функції - 570, на 580 ітерації

Значення цільової функції - 575, на 600 ітерації

Значення цільової функції - 574, на 620 ітерації

Значення цільової функції - 571, на 640 ітерації

Значення цільової функції - 573, на 660 ітерації

Значення цільової функції - 580, на 680 ітерації

Значення цільової функції - 590, на 700 ітерації

Значення цільової функції - 587, на 720 ітерації

Значення цільової функції - 580, на 740 ітерації

Значення цільової функції - 580, на 760 ітерації

Значення цільової функції - 591, на 780 ітерації

Значення цільової функції - 580, на 800 ітерації

Значення цільової функції - 591, на 820 ітерації

Значення цільової функції - 591, на 840 ітерації

Значення цільової функції - 580, на 860 ітерації

Значення цільової функції - 580, на 880 ітерації

Значення цільової функції - 585, на 900 ітерації

Значення цільової функції - 594, на 920 ітерації

Значення цільової функції - 594, на 940 ітерації

Значення цільової функції - 594, на 960 ітерації

Значення цільової функції - 581, на 980 ітерації

### Графіки залежності розв'язку від числа ітерацій

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

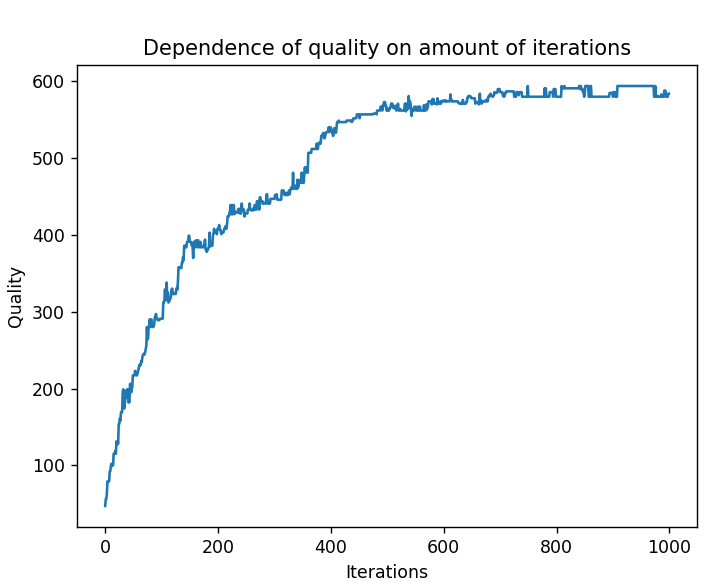


Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

Висновок

В рамках даної лабораторної роботи ми вивчали основні підходи формалізації метаеврестичних алгоритмів і вирішення типових задач з їхньою допомогою, а саме задачу про рюкзак. Також ми дослідили залежність цільової функції від кількості операцій.

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%.