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) *ІП-15 Богун Д.О* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Ахаладзе І.Е* _____
(прізвище, ім'я, по батькові)

Київ 2022

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.

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.

№	**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.

3 Виконання

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

Main.py

```
from Functions import *
backpack, items_num, items = initialize()
print("Backpack: ")
backpack.show_info()
solutions_per_pop = len(items_num)
population_size = (solutions_per_pop, items_num.shape[0])
population = (np.random.randint(2, size = population_size)).astype(int)
iterations = input("Enter iterations amount [20-1000]:")
while not iterations.isdigit() or int(iterations) < 20 or
int(iterations)>1000:
    iterations = input("Enter iterations amount [20-1000]!!!:")
iterations = int(iterations)
print("Running...")
parameters, function_history = local_optimization(backpack, population,
population_size, iterations)
selected_items = items_num * parameters
print('\nSelected items that will maximize the knapsack without breaking
it:')
print_result(selected_items,items)
```

Functions.py

```
import numpy as np
import matplotlib.pyplot as plt
from random import randint
import random as rd
from Classes import *
def initialize():
    items_num = np.arange(1, 101)
    items = []
    for num in items_num:
        item_price = np.random.randint(2, 21)
        item_weight = np.random.randint(1, 11)
        item = Item(num, item_weight, item_price)
        items.append(item)
    backpack = Backpack(250,items)
    return backpack, items_num, items


def calculate_cost(backpack, population, capacity):
    cost = np.empty(population.shape[0])
    for i in range(population.shape[0]):
        sum1 = np.sum(population[i] * backpack.get_price_list())
        sum2 = np.sum(population[i] * backpack.get_weight_list())
        if sum2 <= capacity:
            cost[i] = sum1
        else :
            cost[i] = 0
    return cost.astype(int)

def selection(cost, num_parents, population):
    cost = list(cost)
```

```
    parents = np.empty((num_parents, population.shape[1]))
    for i in range(num_parents):
        max_function_idx = np.where(cost == np.max(cost))
        parents[i,:] = population[max_function_idx[0][0], :]
        cost[max_function_idx[0][0]] = -999999
    return parents

def crossing_over(parents, offsprings_amount):
    offsprings = np.empty((offsprings_amount, parents.shape[1]))
    point = 50
    crossover_rate = 0.7
    i=0
    while (parents.shape[0] < offsprings_amount):
        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:point] = parents[parent1_index,0:point]
        offsprings[i,point:] = parents[parent2_index,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 local_optimization(backpack, population, pop_size, iterations):
    threshold = backpack.get_capacity()
    parameters, function_history = [], []
    num_parents = int(pop_size[0]/2)
    num_offsprings = pop_size[0] - num_parents
    for i in range(iterations):
        function = calculate_cost(backpack, population, threshold)
        function_history.append(function)
        parents = selection(function, num_parents, population)
        offsprings = crossing_over(parents, num_offsprings)
        mutants = mutation(offsprings)
        population[0:parents.shape[0], :] = parents
        np.seterr(divide='ignore', invalid='ignore')
        population[parents.shape[0]:, :] = mutants
    function_last_gen = calculate_cost(backpack, population, threshold)
    max_function = np.where(function_last_gen == np.max(function_last_gen))
    parameters.append(population[max_function[0][0],:])
    return parameters, function_history
def print_result(selected_items, items):
    j = 0
    total_weight = 0
    total_price = 0
    for i in range(0,selected_items.shape[1]):
      if selected_items[0][i] != 0:
```

```
            j+=1
            items[i].show_info()
            print( "Total weight:", total_weight,
"+",items[i].get_weight(),"=", total_weight+items[i].get_weight() )
            total_weight += items[i].get_weight()
            total_price += items[i].get_price()
            print("-----------------------------------------------------")
    print("\n\nAMOUNT OF ITEMS IN BACKPACK: ", j)
    print("TOTAL WEIGHT: ", total_weight)
    print("TOTAL PRICE: ", total_price)
```

Classes.py

```
class Item:
    def __init__(self,number,weight,cost):
        self.number = number
        self.weight = weight
        self.price = cost;
    def show_info(self):
        print("Number:", self.number ,", weight:", self.weight, "price:",
self.price)
    def get_weight(self):
        return self.weight
    def get_price(self):
        return self.price

class Backpack:
    def __init__(self,capacity,items):
        self.items = items
        self.capacity = capacity
    def get_capacity(self):
        return self.capacity
    def get_items(self):
        return self.items
    def show_info(self):
        print("Capacity:", self.capacity)
        price = 0
        weight = 0
        for item in self.items:
            price += item.get_price()
            weight += item.get_weight()
            item.show_info()
        print("Weight of ALL items:",weight)
        print("Price of ALL items:",price)

    def get_price_list(self):
        l = []
        for item in self.items:
            l.append(item.get_price())
        return l
    def get_weight_list(self):
        l = []
        for item in self.items:
            l.append(item.get_weight())
        return l
```

3.1.2 Приклади роботи

```
Items in backpack:
Number: 3 , weight: 3 price: 11
Total weight: 0 + 3 = 3
-------------------------------------------------
Number: 5 , weight: 10 price: 19
Total weight: 3 + 10 = 13
-------------------------------------------------
Number: 7 , weight: 5 price: 11
Total weight: 13 + 5 = 18
-------------------------------------------------
Number: 10 , weight: 2 price: 2
Total weight: 18 + 2 = 20
-------------------------------------------------
Number: 11 , weight: 5 price: 2
Total weight: 20 + 5 = 25
-------------------------------------------------
Number: 12 , weight: 8 price: 18
Total weight: 25 + 8 = 33
-------------------------------------------------
Number: 14 , weight: 4 price: 17
Total weight: 33 + 4 = 37
-------------------------------------------------
```

```
-------------------------------------------------
Number: 15 , weight: 2 price: 13
Total weight: 37 + 2 = 39
-------------------------------------------------
Number: 16 , weight: 4 price: 11
Total weight: 39 + 4 = 43
-------------------------------------------------
Number: 23 , weight: 3 price: 7
Total weight: 43 + 3 = 46
-------------------------------------------------
Number: 24 , weight: 4 price: 13
Total weight: 46 + 4 = 50
-------------------------------------------------
Number: 27 , weight: 1 price: 6
Total weight: 50 + 1 = 51
-------------------------------------------------
Number: 29 , weight: 3 price: 12
Total weight: 51 + 3 = 54
-------------------------------------------------
Number: 31 , weight: 8 price: 20
Total weight: 54 + 8 = 62
-------------------------------------------------
Number: 33 , weight: 4 price: 5
Total weight: 62 + 4 = 66
-------------------------------------------------
```

```
-------------------------------------------------
Number: 34 , weight: 6 price: 12
Total weight: 66 + 6 = 72
-------------------------------------------------
Number: 36 , weight: 9 price: 3
Total weight: 72 + 9 = 81
-------------------------------------------------
Number: 37 , weight: 6 price: 20
Total weight: 81 + 6 = 87
-------------------------------------------------
Number: 41 , weight: 2 price: 19
Total weight: 87 + 2 = 89
-------------------------------------------------
Number: 43 , weight: 10 price: 3
Total weight: 89 + 10 = 99
-------------------------------------------------
```

```
-------------------------------------------------
Number: 48 , weight: 7 price: 14
Total weight: 99 + 7 = 106
-------------------------------------------------
Number: 49 , weight: 9 price: 13
Total weight: 106 + 9 = 115
-------------------------------------------------
Number: 53 , weight: 3 price: 20
Total weight: 115 + 3 = 118
-------------------------------------------------
-------------------------------------------------
Number: 54 , weight: 6 price: 13
Total weight: 118 + 6 = 124
-------------------------------------------------
Number: 55 , weight: 8 price: 18
Total weight: 124 + 8 = 132
-------------------------------------------------
Number: 56 , weight: 4 price: 15
Total weight: 132 + 4 = 136
-------------------------------------------------
Number: 57 , weight: 1 price: 9
Total weight: 136 + 1 = 137
-------------------------------------------------
Number: 59 , weight: 9 price: 20
Total weight: 137 + 9 = 146
-------------------------------------------------
Number: 60 , weight: 6 price: 8
Total weight: 146 + 6 = 152
-------------------------------------------------
Number: 62 , weight: 6 price: 5
Total weight: 152 + 6 = 158
-------------------------------------------------
Number: 64 , weight: 1 price: 12
Total weight: 158 + 1 = 159
-------------------------------------------------
-------------------------------------------------
Number: 66 , weight: 1 price: 17
Total weight: 159 + 1 = 160
-------------------------------------------------
Number: 67 , weight: 1 price: 16
Total weight: 160 + 1 = 161
-------------------------------------------------
Number: 69 , weight: 2 price: 16
Total weight: 161 + 2 = 163
-------------------------------------------------
Number: 71 , weight: 9 price: 14
Total weight: 163 + 9 = 172
-------------------------------------------------
Number: 77 , weight: 9 price: 14
Total weight: 172 + 9 = 181
-------------------------------------------------
Number: 79 , weight: 2 price: 11
Total weight: 181 + 2 = 183
-------------------------------------------------
Number: 83 , weight: 8 price: 19
Total weight: 183 + 8 = 191
-------------------------------------------------
Number: 84 , weight: 10 price: 8
Total weight: 191 + 10 = 201
-------------------------------------------------
```

```
--------------------------------------------------
Number: 85 , weight: 4 price: 9
Total weight: 201 + 4 = 205
--------------------------------------------------
Number: 87 , weight: 3 price: 14
Total weight: 205 + 3 = 208
--------------------------------------------------
Number: 88 , weight: 9 price: 15
Total weight: 208 + 9 = 217
--------------------------------------------------
Number: 89 , weight: 2 price: 17
Total weight: 217 + 2 = 219
--------------------------------------------------
Number: 91 , weight: 9 price: 9
Total weight: 219 + 9 = 228
--------------------------------------------------
Number: 93 , weight: 3 price: 19
Total weight: 228 + 3 = 231
--------------------------------------------------
Number: 94 , weight: 1 price: 13
Total weight: 231 + 1 = 232
--------------------------------------------------
Number: 97 , weight: 9 price: 15
Total weight: 232 + 9 = 241
--------------------------------------------------
--------------------------------------------------
Number: 98 , weight: 2 price: 4
Total weight: 241 + 2 = 243
--------------------------------------------------
Number: 100 , weight: 7 price: 6
Total weight: 243 + 7 = 250
--------------------------------------------------


AMOUNT OF ITEMS IN BACKPACK:  49
TOTAL WEIGHT:  250
TOTAL PRICE:  607
Press 0 to exit or 1 to restart:0
```

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.

Таблиця 3.1

Номер ітерації	Значення функції
0	179
20	586
40	619
60	620
80	313
100	620
120	608
140	626
160	614
180	620
200	607
220	614
240	620
260	608
280	620
300	626
320	614
340	620
360	620
380	627
400	614
420	614
440	614
460	620
480	608
500	620
520	627
540	620
560	614
580	620
600	620
620	614
640	626
660	620
680	608
700	614

720	626
740	614
760	620
780	626
800	620
820	620
840	626
860	595
880	620
900	614
920	620
940	620
960	626
980	608
1000	607

3.2.2 Графіки залежності розв'язку від числа ітерацій

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

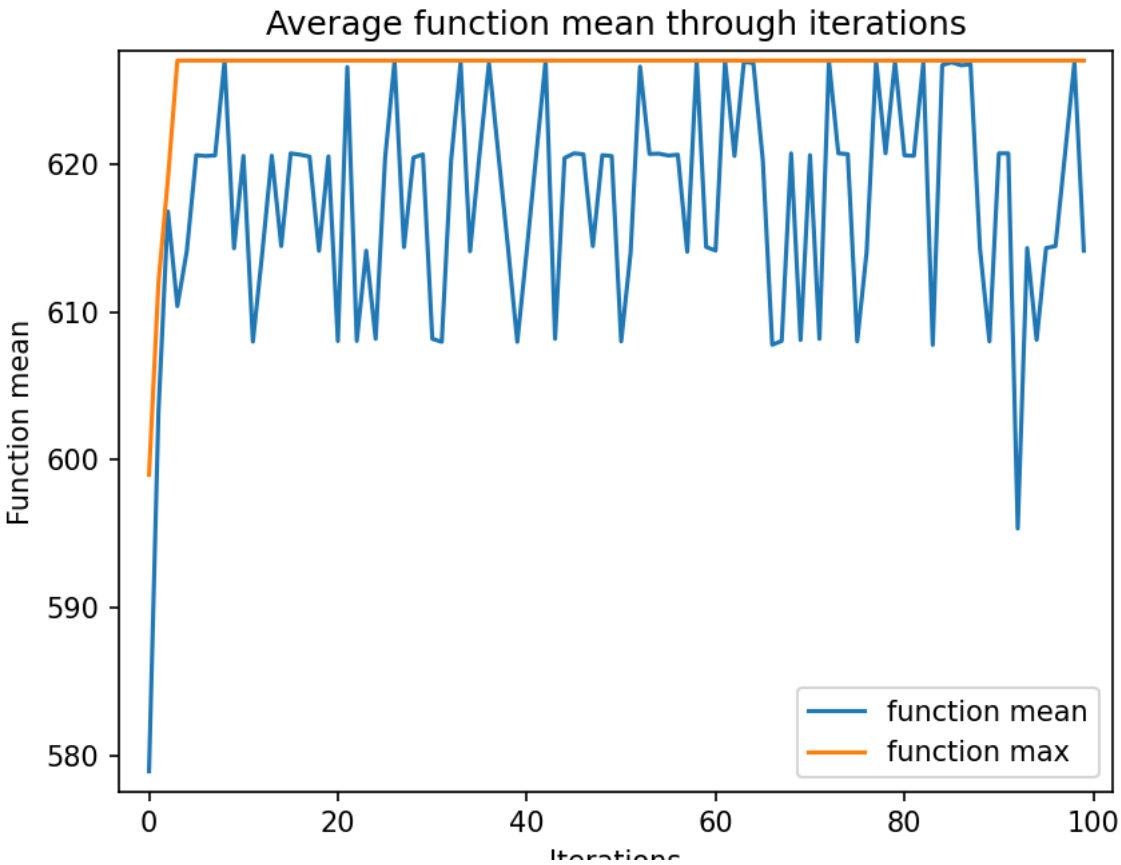

Рисунок 3.3 - Графік залежності розв'язку від числа ітерацій

Висновок

В рамках даної лабораторної роботи я дізнався про різноманітні метаевристичні алгоритми, навчився розв’язувати базові задачі за їх вимогою, а також реалізував задачу про рюкзак, використовуючи генетичний алгоритм та провів аналіз результатів для різної кількості ітерацій.