**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)

*Бобрик Максим Геннадійович ІП-12*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

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

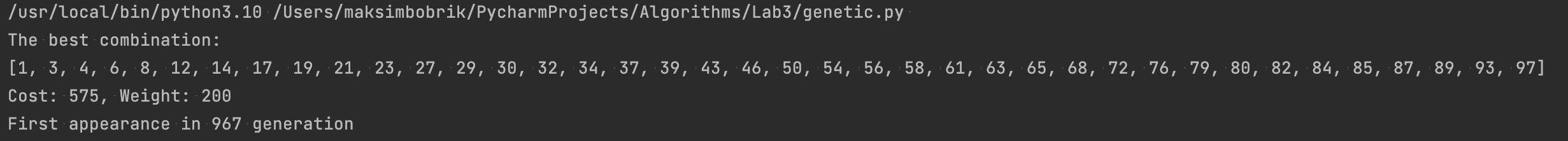
### Вихідний код

import random  
import numpy as np  
import matplotlib.pyplot as plt  
  
  
class Item:  
 num\_items = 0  
  
 def \_\_init\_\_(self, cost, weight):  
 self.cost = cost  
 self.weight = weight  
  
  
def init(popsize, n):  
 population = []  
 for i in range(popsize):  
 pop = '0' \* i  
 pop += '1'  
 pop += '0' \* (n - i)  
 population.append(pop)  
 return population  
  
  
def decode(chromosome, lenght, ITEMS, cap):  
 s = []  
 g = 0  
 f = 0  
 for i in range(lenght):  
 if chromosome[i] == '1':  
 if g+ITEMS[i].weight <= cap:  
 g += ITEMS[i].weight  
 f += ITEMS[i].cost  
 s.append(i)  
 else:  
 break  
 return f, s  
  
  
def fitness(population, lenght, ITEMS, cap):  
 value = []  
 ss = []  
 for i in range(len(population)):  
 f, s = decode(population[i], lenght, ITEMS, cap)  
 value.append(f)  
 ss.append(s)  
 return value, ss  
  
  
def roulettewheel(population, value, p\_size):  
 fitness\_sum = []  
 value\_sum = sum(value)  
 fitness = [i/value\_sum for i in value]  
 for i in range(len(population)):  
 if i == 0:  
 fitness\_sum.append(fitness[i])  
 else:  
 fitness\_sum.append(fitness\_sum[i-1]+fitness[i])  
 population\_new = []  
 for j in range(p\_size):  
 r = np.random.uniform(0, 1)  
 for i in range(len(fitness\_sum)):  
 if i == 0:  
 if 0 <= r <= fitness\_sum[i]:  
 population\_new.append(population[i])  
 else:  
 if fitness\_sum[i-1] <= r <= fitness\_sum[i]:  
 population\_new.append(population[i])  
 return population\_new  
  
  
def crossover(population, pc):  
 a = int(len(population)/2)  
 parents\_one = population[:a]  
 parents\_two = population[a:]  
 np.random.shuffle(parents\_one)  
 np.random.shuffle(parents\_two)  
 offspring = []  
 for i in range(a):  
 r = np.random.uniform(0, 1)  
 if r <= pc:  
 point1 = int(len(parents\_one[i])/3)  
 point2 = 2 \* point1  
 off\_one = parents\_one[i][:point1]+parents\_two[i][point1:point2]+parents\_one[i][point2:]  
 off\_two = parents\_two[i][:point1]+parents\_one[i][point1:point2]+parents\_two[i][point2:]  
 else:  
 off\_one = parents\_one[i]  
 off\_two = parents\_two[i]  
 offspring.append(off\_one)  
 offspring.append(off\_two)  
 return offspring  
  
  
def mutation(offspring, pm):  
 for i in range(len(offspring)):  
 r = np.random.uniform(0, 1)  
 if r <= pm:  
 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, ITEMS, CAPACITY, ITEM\_BENEFITS):  
 for gen in range(len(offspring)):  
 weight = 0  
 for chromosome in range(len(offspring[gen])):  
 if offspring[gen][chromosome] == '1':  
 weight += ITEMS[chromosome].weight  
  
 if weight <= CAPACITY:  
 for item in range(len(ITEM\_BENEFITS)):  
 if offspring[gen][ITEM\_BENEFITS[item][1]] == '0':  
 if weight + ITEMS[ITEM\_BENEFITS[item][1]].weight <= CAPACITY:  
 weight += ITEMS[ITEM\_BENEFITS[item][1]].weight  
 if not ITEM\_BENEFITS[item][1]:  
 offspring[gen] = '1' + offspring[gen][1:]  
 else:  
 offspring[gen] = offspring[gen][:(ITEM\_BENEFITS[item][1] - 1)] + '1' + offspring[gen][ITEM\_BENEFITS[item][1]:]  
 break  
 return offspring  
  
  
def fitness\_items(ITEMS):  
 items = []  
 for i in range(len(ITEMS)):  
 items.append((ITEMS[i].cost/ITEMS[i].weight, i))  
 items.sort(reverse=True)  
 return items  
  
  
def main():  
 GENERATIONS = 1000  
 P\_CROSSOVER = 1  
 P\_MUTATION = 0.1  
 P\_SIZE = 100  
 LENGTH\_CHROMOSOME = 100  
 ITEMS = []  
 for item in range(LENGTH\_CHROMOSOME):  
 ITEMS.append(Item(random.randint(2, 20), random.randint(1, 10)))  
 ITEM\_BENEFIT = fitness\_items(ITEMS)  
 CAPACITY = 200  
 IMPROVE = True  
  
 population = init(P\_SIZE, LENGTH\_CHROMOSOME)  
  
 t = []  
 best\_ind = []  
  
 for i in range(GENERATIONS):  
  
 offspring = crossover(population, P\_CROSSOVER)  
 offspring = mutation(offspring, P\_MUTATION)  
 if IMPROVE:  
 offspring = local\_improve(offspring, ITEMS, CAPACITY, ITEM\_BENEFIT)  
 mixpopulation = population+offspring  
  
 value, s = fitness(mixpopulation, LENGTH\_CHROMOSOME, ITEMS, CAPACITY)  
 population = roulettewheel(mixpopulation, value, P\_SIZE)  
  
 value1, s1 = fitness(population, LENGTH\_CHROMOSOME, ITEMS, CAPACITY)  
  
 h = value1.index(max(value1))  
 t.append(max(value1))  
 best\_ind.append(population[h])  
  
 hh = t.index(max(t))  
 f2, s2 = decode(best\_ind[hh], LENGTH\_CHROMOSOME, ITEMS, CAPACITY)  
 cost, weight = 0, 0  
 for i in s2:  
 cost += ITEMS[i].cost  
 weight += ITEMS[i].weight  
 print(f"The best combination:\n{s2}\nCost: {cost}, Weight: {weight}")  
 print(f"First appearance in {hh} generation")  
  
 plt.plot(t, color='red')  
 plt.xlabel('Generations')  
 plt.ylabel('Fitness')  
 plt.title('Dependence of quality on generation')  
 plt.show()  
  
  
main()

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

Рисунок 3.1



## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

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

Таблиця 3.1

|  |  |
| --- | --- |
| Кількість ітерацій | Значення цільової функції |
| 0 | 155 |
| 20 | 185 |
| 40 | 223 |
| 60 | 260 |
| 80 | 300 |
| 100 | 332 |
| 120 | 361 |
| 140 | 393 |
| 160 | 424 |
| 180 | 414 |
| 200 | 436 |
| 220 | 438 |
| 240 | 445 |
| 260 | 481 |
| 280 | 487 |
| 300 | 490 |
| 320 | 495 |
| 340 | 521 |
| 360 | 521 |
| 380 | 527 |
| 400 | 532 |
| 420 | 532 |
| 440 | 541 |
| 460 | 541 |
| 480 | 541 |
| 500 | 541 |
| 520 | 541 |
| 540 | 541 |
| 560 | 541 |
| 580 | 541 |
| 600 | 541 |
| 620 | 541 |
| 640 | 541 |
| 660 | 541 |
| 680 | 541 |
| 700 | 541 |
| 720 | 541 |
| 740 | 545 |
| 760 | 541 |
| 780 | 541 |
| 800 | 541 |
| 820 | 541 |
| 840 | 541 |
| 860 | 541 |
| 880 | 541 |
| 900 | 541 |
| 920 | 541 |
| 940 | 541 |
| 960 | 541 |
| 980 | 541 |
| 1000 | 541 |

### Графіки залежності розв'язку від числа ітерацій

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

Description automatically generated

Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

Висновок

Під час лобароторної роботи були розглянути метаевристичні алгоритми, розроблено та реалізовано генетичний алгоритм, що виріщує задачу про рюкзак.

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%.