**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-11 Головня О.Р.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

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** |
| 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. |

# Виконання

Варіант 7

## Програмна реалізація алгоритму

from copy import copy

from random import random

from random import randint

class Genetic:

BACKPACK\_CAPACITY = 150

NUMBER\_OF\_ITEMS = 100

ITERATION\_LIMIT = 1000

mutation\_probability = 0.05

PATH\_TO\_FILE = "my\_file.txt"

weight = [randint(1, 5) for \_ in range(NUMBER\_OF\_ITEMS)]

value = [randint(2, 10) for \_ in range(NUMBER\_OF\_ITEMS)]

population = []

probabilities = []

best = None

best\_ = None

def start(self):

data\_ = ""

self.CreatePopulation()

for i in range(self.ITERATION\_LIMIT):

if (i + 1) % 20 == 0:

data\_ += f"({i + 1}, {self.population[-1][-1]})\n"

self.Probabilities\_()

self.NewGeneration()

weight = 0

for i in range(self.NUMBER\_OF\_ITEMS):

if self.population[-1][0][i] == 1:

weight += self.weight[i]

print("Weight: ", weight)

print("Value: ", self.population[-1][-1])

self.in\_file(data\_)

@staticmethod

def mutation(chromosome):

mutated\_chromosome = copy(chromosome)

i1 = randint(0, len(chromosome) - 1)

i2 = randint(0, len(chromosome) - 1)

mutated\_chromosome[i1], mutated\_chromosome[i2] = mutated\_chromosome[i2], mutated\_chromosome[i1]

return mutated\_chromosome

@staticmethod

def crossover(parent\_1, parent\_2):

child\_chromosome = []

for i in range(len(parent\_1)):

if parent\_1[i] != parent\_2[i]:

new\_gene = randint(0, 1)

child\_chromosome.append(new\_gene)

else:

child\_chromosome.append(parent\_1[i])

return child\_chromosome

def CreatePopulation(self):

for i in range(self.NUMBER\_OF\_ITEMS):

init = [0] \* self.NUMBER\_OF\_ITEMS

init[i] = 1

self.population.append((init, self.fitness(init)))

return True

def fitness(self, chromosome):

sum\_of\_weights = 0

sum\_of\_values = 0

for i in range(len(chromosome)):

if chromosome[i] == 1:

sum\_of\_weights += self.weight[i]

sum\_of\_values += self.value[i]

if sum\_of\_weights > self.BACKPACK\_CAPACITY:

return 0

else:

return sum\_of\_values

def Probabilities\_(self):

self.probabilities = []

total\_sum = 0

for chromosome in self.population:

total\_sum += chromosome[1]

probability\_sum = 0

for chromosome in self.population:

if total\_sum == 0:

probability\_sum = 0

else:

probability\_sum += float(chromosome[1] / total\_sum)

self.probabilities.append(probability\_sum)

def NewGeneration(self):

parent\_1 = 0

parent\_2 = -1

r = random()

for parent in range(len(self.probabilities)):

if r < self.probabilities[parent]:

parent\_1 = parent

break

r = random()

for parent in range(len(self.probabilities)):

if r < self.probabilities[parent]:

parent\_2 = parent

break

child\_chromosome = self.crossover(self.population[parent\_1][0], self.population[parent\_2][0])

r = random()

if r < self.mutation\_probability:

mutated\_chromosome = self.mutation(child\_chromosome)

final\_chromosome = self.upgraded\_chromosome(mutated\_chromosome)

else:

final\_chromosome = self.upgraded\_chromosome(child\_chromosome)

self.population.append((final\_chromosome, self.fitness(final\_chromosome)))

self.population.sort(key=lambda x: x[-1])

self.population.pop(0)

def upgraded\_chromosome(self, chromosome):

upgraded\_chromosome = chromosome.copy()

while True:

i = randint(0, self.NUMBER\_OF\_ITEMS - 1)

if upgraded\_chromosome[i] == 0:

upgraded\_chromosome[i] = 1

break

return upgraded\_chromosome

def in\_file(self, data\_):

final\_weight = 0

result = ''

for i in range(self.NUMBER\_OF\_ITEMS):

result += f"item {i + 1}: ( weight {self.weight[i]}, values {self.value[i]})\n"

result += "---------------------------------\n"

for i in range(self.NUMBER\_OF\_ITEMS):

if self.population[-1][0][i] == 1:

result += f"item {i + 1}: ( weight {self.weight[i]}, values {self.value[i]})\n"

final\_weight += self.weight[i]

result += "---------------------------------\n"

result += data\_

with open(self.PATH\_TO\_FILE, "wt") as my\_file:

my\_file.write(result)

g = Genetic()

g.start()

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

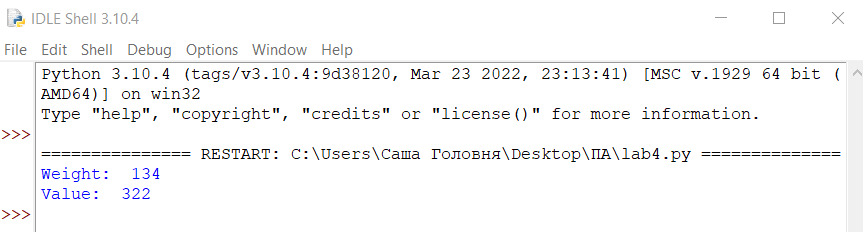
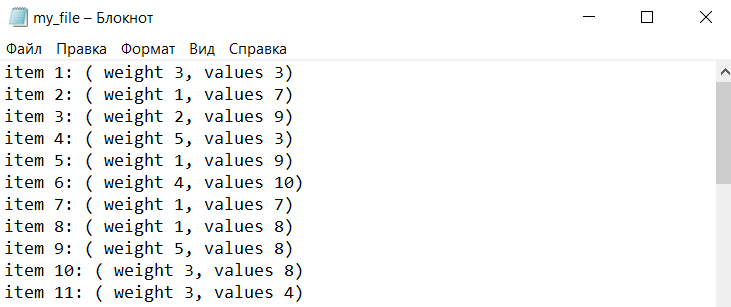
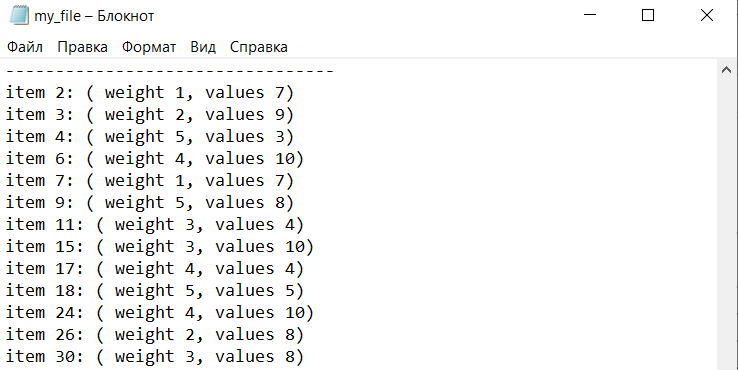


Рисунок 3.1 – Вивід кінцевих результатів у консоль

Рисунок 3.2 – Повні дані про розв’язок, записані у текстовий файл



- Початкові предмети



- Рішення

Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

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

Таблиця 3.1 – Характеристика роботи генетичного алгоритму

|  |  |  |  |
| --- | --- | --- | --- |
| Кількість ітерацій | Значення цільової функції | Кількість ітерацій | Значення цільової функції |
| 20 | 33 | 520 | 195 |
| 40 | 59 | 540 | 221 |
| 60 | 68 | 560 | 221 |
| 80 | 77 | 580 | 221 |
| 100 | 84 | 600 | 226 |
| 120 | 84 | 620 | 231 |
| 140 | 106 | 640 | 253 |
| 160 | 120 | 660 | 253 |
| 180 | 120 | 680 | 253 |
| 200 | 120 | 700 | 253 |
| 220 | 120 | 720 | 260 |
| 240 | 148 | 740 | 260 |
| 260 | 148 | 760 | 279 |
| 280 | 148 | 780 | 281 |
| 300 | 149 | 800 | 281 |
| 320 | 149 | 820 | 281 |
| 340 | 160 | 840 | 281 |
| 360 | 160 | 860 | 297 |
| 380 | 172 | 880 | 297 |
| 400 | 172 | 900 | 300 |
| 420 | 172 | 920 | 310 |
| 440 | 176 | 940 | 316 |
| 460 | 187 | 960 | 316 |
| 480 | 189 | 980 | 316 |
| 500 | 189 | 1000 | 316 |

### 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

Виконуючи дану лабораторну роботу, було розглянуто та досліджено генетичний алгоритм, його реалізація на прикладі задачі про рюкзак, використовуючи пропорційну селекцію, рівномірний оператор схрещування, імовірнісний оператор мутації у вигляді зміни місцями двох випадкових генів. Також, було побудовано графік значень 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%.