**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-з21 Єфремов Юрій Володимирович

**Перевірив**

(прізвище, ім'я, по батькові)

Головченко М.М.

Київ 2023

Зміст

[1 Мета лабораторної роботи 3](#_Toc109342184)

[2 ЗаВдання 4](#_Toc109342185)

[3 Виконання](#_Toc109342186) 5

[3.1 Псевдокод алгоритму 6](#_Toc109342187)

[3.2 Програмна реалізація алгоритму 6](#_Toc109342188)

[3.2.1 Вихідний код 6](#_Toc109342189)

[Висновок 7](#_Toc109342190)

# 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 - 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.

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.

# Виконання

## Псевдокод алгоритму

**Алгоритм: Генетичний алгоритм для задачі про рюкзак з локальним покращенням**

**Параметри:**

**- Місткість рюкзака P = 250**

**- Кількість предметів n = 100**

**- Початкова популяція розміром 100 осіб**

**- Оператор схрещування: одноточковий по 50 генів**

**- Ймовірність мутації: 5%, змінюємо 1 випадковий ген**

**- Локальне покращення: Власний оператор локального покращення**

**Початок:**

**1. Ініціалізувати початкову популяцію розміром 100 осіб, кожна із яких містить лише один випадковий предмет.**

**2. Для кожного індивіда визначити його цільову функцію (вартість рюкзака) та його вагу.**

**3. Повторювати до досягнення 1000 ітерацій або збільшення якості розв'язку до максимально можливого значення:**

**a. Здійснювати селекцію і створювати нову популяцію, використовуючи оператор схрещування та мутації.**

**b. Застосовувати локальне покращення до кращих особин нової популяції.**

**c. Запам'ятовувати якість кращої особини після кожних 20 ітерацій.**

**4. Побудувати графік залежності якості розв'язку від числа ітерацій.**

**Оператор схрещування:**

**- Вибрати випадковий пункт схрещування (точка переривання).**

**- Обміняти гени між батьками перед точкою переривання.**

**- Створити двох нащадків з отриманих батьків.**

**Оператор мутації:**

**- Вибрати випадкового індивіда.**

**- Вибрати випадковий ген цього індивіда.**

**- Замінити значення гену на нове випадкове значення відповідно до обмежень задачі.**

**Локальне покращення:**

**- Для кожного індивіда в популяції:**

**a. Спробувати додати або видалити один предмет, якщо це покращить вартість рюкзака.**

**b. Повторювати цей процес, доки є можливість покращення.**

**Завершення.**

## Програмна реалізація алгоритму

### Вихідний код

import random  
import matplotlib.pyplot as plt  
import numpy as np  
  
# Параметри задачі про рюкзак  
capacity = 250  
num\_items = 100  
items = [(random.randint(2, 20), random.randint(1, 10)) for \_ in range(num\_items)]  
  
# Параметри генетичного алгоритму  
population\_size = 100  
crossover\_point = 50  
mutation\_probability = 0.05  
local\_improvement\_probability = 0.2  
iterations = 1000  
local\_improvement\_iterations = 10  
  
  
def fitness(individual):  
 total\_value = sum(item[0] for i, item in enumerate(items) if individual[i] == 1)  
 total\_weight = sum(item[1] for i, item in enumerate(items) if individual[i] == 1)  
 return total\_value if total\_weight <= capacity else 0  
  
  
def crossover(parent1, parent2):  
 point = random.randint(0, len(parent1) - 1)  
 child1 = parent1[:point] + parent2[point:]  
 child2 = parent2[:point] + parent1[point:]  
 return child1, child2  
  
  
def mutate(individual):  
 index = random.randint(0, len(individual) - 1)  
 individual[index] = 1 - individual[index]  
  
  
def local\_improvement(individual):  
 for \_ in range(local\_improvement\_iterations):  
 if random.random() < local\_improvement\_probability:  
 mutate(individual)  
  
  
def genetic\_algorithm():  
 population = [  
 [random.randint(0, 1) for \_ in range(num\_items)] for \_ in range(population\_size)  
 ]  
 best\_fitness\_history = []  
  
 for generation in range(iterations):  
 population.sort(key=lambda ind: -fitness(ind))  
 best\_individual = population[0]  
 best\_fitness\_history.append(fitness(best\_individual))  
  
 new\_population = []  
  
 for \_ in range(population\_size // 2):  
 parent1, parent2 = random.choices(population[:10], k=2)  
 child1, child2 = crossover(parent1, parent2)  
  
 if random.random() < mutation\_probability:  
 mutate(child1)  
 if random.random() < mutation\_probability:  
 mutate(child2)  
  
 local\_improvement(child1)  
 local\_improvement(child2)  
  
 new\_population.extend([child1, child2])  
  
 population = new\_population  
  
 return best\_individual, best\_fitness\_history  
  
  
best\_solution, fitness\_history = genetic\_algorithm()  
  
  
def main():  
 best\_solution, fitness\_history = genetic\_algorithm()  
  
 # Графік залежності якості розв'язку від числа ітерацій  
 iterations = len(fitness\_history)  
 iterations\_range = np.arange(20, iterations + 1, 20)  
  
 plt.plot(iterations\_range, fitness\_history[: len(iterations\_range)], marker="o")  
 plt.xlabel("Number of Iterations")  
 plt.ylabel("Fitness")  
 plt.title("Genetic Algorithm for Knapsack Problem")  
 plt.show()  
  
 print("Best Solution:", best\_solution)  
 print("Best Fitness:", fitness(best\_solution))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Висновок

В ході лабораторної роботи було розроблено і реалізовано генетичний алгоритм для вирішення задачі про рюкзак. Задача полягає в тому, щоб знайти такий набір предметів з визначеною вагою та цінністю, який максимізує сумарну цінність і при цьому не перевищує встановлену місткість рюкзака.

Генетичний алгоритм був реалізований з використанням початкової популяції, одноточкового схрещування, мутації та оператора локального покращення. В процесі алгоритму фіксувалась якість отриманого розв'язку після кожних 20 ітерацій, і побудований графік залежності якості розв'язку від числа ітерацій.

Висновок:

1. Генетичний алгоритм успішно вирішує задачу про рюкзак, знаходячи оптимальні розв'язки.

2. Оператор локального покращення сприяє збільшенню якості розв'язку.

3. З побудованих графіків видно, якість розв'язку збільшується з кожною ітерацією, але при цьому спостерігається деяка збіжність, коли приріст якості уповільнюється.