

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 № 5 з дисципліни
«Проектування алгоритмів»

„Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.2”

Виконав(ла)

ІП-22, Андрєєва Уляна Андріївна
(шифр, прізвище, ім'я, по батькові)

Перевірив

Ахаладзе Ілля Елдарійович
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ.....	3
2	ЗАВДАННЯ.....	4
3	ВИКОНАННЯ	10
3.1	ПОКРОКОВИЙ АЛГОРИТМ	10
3.2	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ	12
3.2.1	<i>Вихідний код.....</i>	<i>12</i>
3.2.2	<i>Приклади роботи.....</i>	<i>19</i>
3.3	ТЕСТУВАННЯ АЛГОРИТМУ	21
	ВИСНОВОК.....	23
	КРИТЕРІЇ ОЦІНЮВАННЯ.....	24

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи розробки метасвистичних алгоритмів для типових прикладних задач. Опрацювати методологію підбору прийнятних параметрів алгоритму.

2 ЗАВДАННЯ

Зробити узагальнений висновок в якому обов'язково описати залежність якості розв'язку від вхідних параметрів.

Таблиця 2.1 – Прикладні задачі

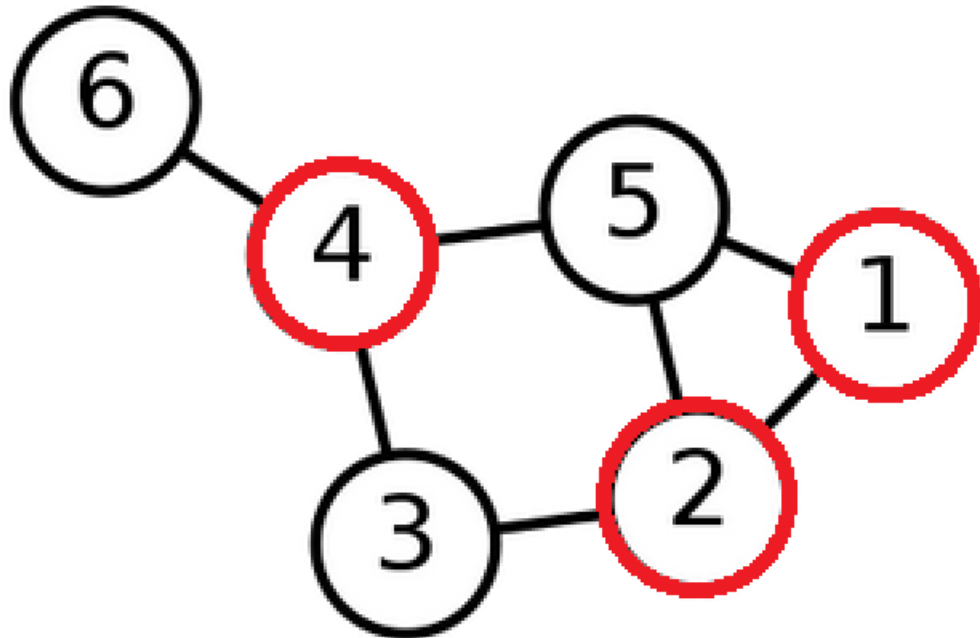
№	Задача
1	<p>Задача про рюкзак (місткість $P=500$, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 20 (випадкова)). Для заданої множини предметів, кожен з яких має вагу і цінність, визначити яку кількість кожного з предметів слід взяти, так, щоб сумарна вага не перевищувала задану, а сумарна цінність була максимальною.</p> <p>Задача часто виникає при розподілі ресурсів, коли наявні фінансові обмеження, і вивчається в таких областях, як комбінаторика, інформатика, теорія складності, криптографія, прикладна математика.</p>
2	<p>Задача комівояжера (300 вершин, відстань між вершинами випадкова від 5 до 150) полягає у знаходженні найвигіднішого маршруту, що проходить через вказані міста хоча б по одному разу. В умовах завдання вказуються критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій тощо) і відповідні матриці відстаней, вартості тощо. Зазвичай задано, що маршрут повинен проходити через кожне місто тільки один раз, в такому випадку розв'язок знаходиться серед гамільтонових циклів.</p> <p>Розглядається симетричний, асиметричний та змішаний варіанти.</p> <p>В загальному випадку, асиметрична задача комівояжера відрізняється тим, що ребра між вершинами можуть мати різну вагу в залежності від напрямку, тобто, задача моделюється орієнтованим графом. Таким чином, окрім ваги ребер графа, слід також зважати і на те, в якому напрямку знаходяться ребра.</p>

	<p>У випадку симетричної задачі всі пари ребер між одними й тими самими вершинами мають однакову вагу.</p> <p>У випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів і напрямку руху.</p> <p>Застосування:</p> <ul style="list-style-type: none"> – доставка товарів (в цьому випадку може бути більш доречна постановка транспортної задачі - доставка в кілька магазинів з декількох складів); – доставка води; – моніторинг об'єктів; – поповнення банкоматів готівкою; – збір співробітників для доставки вахтовим методом.
3	<p>Розфарбовування графа (300 вершин, степінь вершини не більше 30, але не менше 2) – називають таке приписування кольорів (або натуральних чисел) його вершинам, що ніякі дві суміжні вершини не набувають однакового кольору. Найменшу можливу кількість кольорів у розфарбуванні називають хроматичне число.</p> <p>Застосування:</p> <ul style="list-style-type: none"> – розкладу для освітніх установ; – розкладу в спорті; – планування зустрічей, зборів, інтерв'ю; – розклади транспорту, в тому числі - авіатранспорту; – розкладу для комунальних служб;
4	<p>Задача вершинного покриття (300 вершин, степінь вершини не більше 30, але не менше 2). Вершинне покриття для неорієнтованого графа $G = (V, E)$ - це множина його вершин S, така, що, у кожного ребра графа хоча б один з кінців входить в вершину з S.</p> <p>Задача вершинного покриття полягає в пошуку вершинного покриття</p>

найменшого розміру для заданого графа (цей розмір називається числом вершинного покриття графа).

На вході: Граф $G = (V, E)$.

Результат: множина $C \subseteq V$ - найменше вершинне покриття графа G .



Застосування:

- розміщення пунктів обслуговування;
- призначення екіпажів на транспорт;
- проектування інтегральних схем і конвеєрних ліній.

5

Задача про кліку (300 вершин, степінь вершини не більше 30, але не менше 2). Клікою в неорієнтованому графі називається підмножина вершин, кожні дві з яких з'єднані ребром графа. Іншими словами, це повний підграф первісного графа. Розмір кліки визначається як число вершин в ній.

Задача про кліку існує у двох варіантах: у **задачі розпізнавання** потрібно визначити, чи існує в заданому графі G кліка розміру k , тоді як в **обчислювальному варіанті** потрібно знайти в заданому графі G кліку максимального розміру або всі максимальні кліки (такі, що не можна збільшити).

Застосування:

	<ul style="list-style-type: none"> – біоінформатика; – електротехніка;
6	<p>Задача про найкоротший шлях (300 вершин, відстань між вершинами випадкова від 5 до 150, степінь вершини не більше 10, але не менше 1) - задача пошуку найкоротшого шляху (ланцюга) між двома точками (вершинами) на графі, в якій мінімізується сума ваг ребер, що складають шлях.</p> <p>Важливість задачі визначається її різними практичними застосуваннями. Наприклад, в GPS-навігаторах здійснюється пошук найкоротшого шляху між точкою відправлення і точкою призначення. Як вершин виступають перехрестя, а дороги є ребрами, які лежать між ними. Якщо сума довжин доріг між перехрестями мінімальна, тоді знайдений шлях найкоротший.</p>

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

№	Алгоритми і досліджувані параметри
1	<p>Генетичний алгоритм:</p> <ul style="list-style-type: none"> - оператор схрещування (мінімум 3); - мутація (мінімум 2); - оператор локального покращення (мінімум 2).
2	<p>Мурашиний алгоритм:</p> <ul style="list-style-type: none"> – α; – β; – ρ; – L_{min}; – кількість мурах M і їх типи (елітні, тощо...); – маршрути з однієї чи різних вершин.
3	<p>Бджолиний алгоритм:</p> <ul style="list-style-type: none"> – кількість ділянок;

	– кількість бджіл (фуражирів і розвідників).
--	--

Таблиця 2.3 – Варіанти задач і алгоритмів

№	Задачі і алгоритми
1	Задача про рюкзак + Генетичний алгоритм
2	Задача про рюкзак + Бджолиний алгоритм
3	Задача комівояжера (асиметрична мережа) + Генетичний алгоритм
4	Задача комівояжера (симетрична мережа) + Генетичний алгоритм
5	Задача комівояжера (змішана мережа) + Генетичний алгоритм
6	Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм
7	Задача комівояжера (симетрична мережа) + Мурашиний алгоритм
8	Задача комівояжера (змішана мережа) + Мурашиний алгоритм
9	Задача вершинного покриття + Генетичний алгоритм
10	Задача вершинного покриття + Бджолиний алгоритм
11	Задача комівояжера (асиметрична мережа) + Бджолиний алгоритм
12	Задача комівояжера (симетрична мережа) + Бджолиний алгоритм
13	Задача комівояжера (змішана мережа) + Бджолиний алгоритм
14	Розфарбовування графа + Генетичний алгоритм
15	Розфарбовування графа + Бджолиний алгоритм
16	Задача про кліку (задача розпізнавання) + Генетичний алгоритм
17	Задача про кліку (задача розпізнавання) + Бджолиний алгоритм
18	Задача про кліку (обчислювальна задача) + Генетичний алгоритм
19	Задача про кліку (обчислювальна задача) + Бджолиний алгоритм
20	Задача про найкоротший шлях + Генетичний алгоритм
21	Задача про найкоротший шлях + Мурашиний алгоритм
22	Задача про найкоротший шлях + Бджолиний алгоритм
23	Задача про рюкзак + Генетичний алгоритм

24	Задача про рюкзак + Бджолиний алгоритм
25	Задача комівояжера (асиметрична мережа) + Генетичний алгоритм
26	Задача комівояжера (симетрична мережа) + Генетичний алгоритм
27	Задача комівояжера (змішана мережа) + Генетичний алгоритм
28	Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм
29	Задача комівояжера (симетрична мережа) + Мурашиний алгоритм
30	Задача комівояжера (змішана мережа) + Мурашиний алгоритм

3.1 Покроковий алгоритм

```

function crossover(parent1, parent2, random):
    parent1Items = parent1.getItems()
    parent2Items = parent2.getItems()

    if parent1Items.isEmpty() or parent2Items.isEmpty():
        child = Individual()
        child.setWeightLimit(parent1.getWeightLimit())
        child.setVolumeLimit(parent1.getVolumeLimit())
        return child

    crossoverPoint1 = random.nextInt(parent1Items.size())
    crossoverPoint2 = random.nextInt(parent1Items.size())
    start = min(crossoverPoint1, crossoverPoint2)
    end = max(crossoverPoint1, crossoverPoint2)

    child = Individual()
    childItems = []

    childItems.addAll(parent1Items.subList(start, end))

    for item in parent2Items:
        if item not in childItems:
            childItems.add(item)

    child.setItems(childItems)
    child.setWeightLimit(parent1.getWeightLimit())

```

```
child.setVolumeLimit(parent1.getVolumeLimit())
```

```
return child
```

```
function crossover(parent1, parent2, random):
```

```
    parent1Items = parent1.getItems()
```

```
    parent2Items = parent2.getItems()
```

```
    if parent1Items.isEmpty() or parent2Items.isEmpty():
```

```
        child = Individual()
```

```
        child.setWeightLimit(parent1.getWeightLimit())
```

```
        child.setVolumeLimit(parent1.getVolumeLimit())
```

```
        return child
```

```
    crossoverPoint1 = random.nextInt(parent1Items.size())
```

```
    crossoverPoint2 = random.nextInt(parent1Items.size())
```

```
    start = min(crossoverPoint1, crossoverPoint2)
```

```
    end = max(crossoverPoint1, crossoverPoint2)
```

```
    child = Individual()
```

```
    childItems = []
```

```
    childItems.addAll(parent1Items.subList(start, end))
```

```
    for item in parent2Items:
```

```
        if item not in childItems:
```

```
            childItems.add(item)
```

```
    child.setItems(childItems)
```

```
    child.setWeightLimit(parent1.getWeightLimit())
```

```
child.setVolumeLimit(parent1.getVolumeLimit())
```

```
return child
```

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
import java.util.*;

public class GeneticAlgorithm {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int populationSize = getPositiveIntegerFromUser("Enter the population size: ", scanner);
        int numGenerations = getPositiveIntegerFromUser("Enter the number of generations: ", scanner);
        double mutationRate = getProbabilityFromUser("Enter the mutation rate: ", scanner);
        double localImprovementRate = getProbabilityFromUser("Enter the local improvement rate: ", scanner);

        List<Individual> population = initializePopulation(populationSize, new Random());

        for (int generation = 0; generation < numGenerations; generation++) {
            evaluatePopulation(population);
            System.out.println("Generation: " + generation);

            for (int i = 0; i < population.size(); i++) {
                System.out.println("Individual " + i + " Fitness: " + population.get(i).getFitness());
            }

            List<Individual> offspring = new ArrayList<>();
            while (offspring.size() < population.size()) {
                Individual parent1 = selectParent(population, new Random());
                Individual parent2 = selectParent(population, new Random());
                Individual child = crossover(parent1, parent2, new Random());
                if (new Random().nextDouble() < mutationRate) {
                    mutate(child, new Random());
                }
                if (new Random().nextDouble() < localImprovementRate) {
                    localImprovement(child, new Random());
                }
                offspring.add(child);
            }
            population = offspring;
        }

        scanner.close();
    }

    protected static int getPositiveIntegerFromUser(String message, Scanner scanner) {
        int value = 0;
```

```

while (true) {
    System.out.print(message);
    String input = scanner.next();

    try {
        value = Integer.parseInt(input);

        if (value > 0 && !input.matches("^0+[1-9][0-9]*$")) {
            break;
        } else {
            System.out.println("Please enter a positive integer without leading zeros.");
        }
    } catch (NumberFormatException e) {
        System.out.println("Invalid input. Please enter a valid positive integer.");
    }
}

return value;
}

static double getProbabilityFromUser(String message, Scanner scanner) {
    double probability = 0;

    while (true) {
        System.out.print(message);
        String input = scanner.next();

        try {
            probability = Double.parseDouble(input);

            if (probability >= 0 && probability <= 1 && !input.matches("^0[0-9].*")) {
                break;
            } else {
                System.out.println("Please enter a number between 0 and 1 without leading zeros.");
            }
        } catch (NumberFormatException e) {
            System.out.println("Invalid input. Please enter a valid number.");
        }
    }

    return probability;
}

```

```

protected static List<Individual> initializePopulation(int populationSize, Random random) {
    List<Individual> population = new ArrayList<>();
    for (int i = 0; i < populationSize; i++) {
        Individual individual = new Individual();

        for (int j = 0; j < 5; j++) {
            double weight = random.nextDouble() * 10;

```

```

        double volume = random.nextDouble() * 5;
        double value = random.nextDouble() * 15;
        Individual.Item item = new Individual.Item(weight, volume, value);
        individual.getItems().add(item);
    }

    individual.setWeightLimit(random.nextDouble() * 100);
    individual.setVolumeLimit(random.nextDouble() * 50);

    population.add(individual);
}
return population;
}

protected static void evaluatePopulation(List < Individual > population) {
    for (Individual individual : population) {
        individual.evaluateFitness();
    }
}

protected static Individual selectParent(List < Individual > population, Random random) {
    int tournamentSize = 3;
    List<Individual> tournament = new ArrayList<>();

    for (int i = 0; i < tournamentSize; i++) {
        Individual randomIndividual = population.get(random.nextInt(population.size()));
        tournament.add(randomIndividual);
    }

    return Collections.max(tournament, Comparator.comparingDouble(Individual::getFitness));
}

protected static Individual crossover(Individual parent1, Individual parent2, Random random) {
    List<Individual.Item> parent1Items = parent1.getItems();
    List<Individual.Item> parent2Items = parent2.getItems();

    if (parent1Items.isEmpty() || parent2Items.isEmpty()) {
        Individual child = new Individual();
        child.setWeightLimit(parent1.getWeightLimit());
        child.setVolumeLimit(parent1.getVolumeLimit());
        return child;
    }

    int crossoverPoint1 = random.nextInt(parent1Items.size());
    int crossoverPoint2 = random.nextInt(parent2Items.size());
    int start = Math.min(crossoverPoint1, crossoverPoint2);
    int end = Math.max(crossoverPoint1, crossoverPoint2);

    Individual child = new Individual();
    List<Individual.Item> childItems = new ArrayList<>();

    childItems.addAll(parent1Items.subList(start, end));

```

```

    for (Individual.Item item : parent2Items) {
        if (!childItems.contains(item)) {
            childItems.add(item);
        }
    }

    child.setItems(childItems);
    child.setWeightLimit(parent1.getWeightLimit());
    child.setVolumeLimit(parent1.getVolumeLimit());

    return child;
}

private static void mutate(Individual individual, Random random) {
    List<Individual.Item> items = individual.getItems();

    if (items.size() >= 2) {
        int mutationIndex1 = random.nextInt(items.size());
        int mutationIndex2 = random.nextInt(items.size());

        Collections.swap(items, mutationIndex1, mutationIndex2);
    }
}

private static void localImprovement(Individual individual, Random random) {
    if (random.nextBoolean()) {
        double adjustment = (random.nextDouble() - 0.5) * 5;
        double newWeightLimit = individual.getWeightLimit() + adjustment;
        individual.setWeightLimit(newWeightLimit);
    } else {
        double adjustment = (random.nextDouble() - 0.5) * 2;
        double newVolumeLimit = individual.getVolumeLimit() + adjustment;
        individual.setVolumeLimit(newVolumeLimit);
    }
}

protected static class Individual {
    private List<Item> items;
    private double weightLimit;
    private double volumeLimit;
    private double fitness;

    public Individual() {
        this.items = new ArrayList<>();
    }

    public List<Item> getItems() {
        return items;
    }
}

```

```

public void setItems(List<Item> items) {
    this.items = items;
}

public double getWeightLimit() {
    return weightLimit;
}

public void setWeightLimit(double weightLimit) {
    this.weightLimit = weightLimit;
}

public double getVolumeLimit() {
    return volumeLimit;
}

public void setVolumeLimit(double volumeLimit) {
    this.volumeLimit = volumeLimit;
}

public double getFitness() {
    return fitness;
}

public void setFitness(double fitness) {
    this.fitness = fitness;
}

public void evaluateFitness() {
    double totalWeight = 0;
    double totalVolume = 0;
    double totalValue = 0;

    if (items == null) {
        System.out.println("Error: items is null");
        return;
    }

    for (Item item : items) {
        if (item == null) {
            System.out.println("Error: item is null");
            continue;
        }

        totalWeight += item.getWeight();
        totalVolume += item.getVolume();
        totalValue += item.getValue();
    }

    double weightPenalty = Math.max(0, totalWeight - weightLimit);
    double volumePenalty = Math.max(0, totalVolume - volumeLimit);

```



```

        fitness = totalValue - (weightPenalty + volumePenalty);

        System.out.println("Fitness calculated: " + fitness);
    }

    protected static class Item {
        private double weight;
        private double volume;
        private double value;

        public Item(double weight, double volume, double value) {
            this.weight = weight;
            this.volume = volume;
            this.value = value;
        }

        public double getWeight() {
            return weight;
        }
        public double getVolume() {
            return volume;
        }
        public double getValue() {
            return value;
        }
    }
}

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.util.List;
import java.util.Scanner;

public class GeneticAlgorithmTests {

    @Test
    public void testGetProbabilityFromUserWithValidInput() {
        String input = "0.75";
        InputStream in = new ByteArrayInputStream(input.getBytes());
        System.setIn(in);

        GeneticAlgorithm algorithm = new GeneticAlgorithm();
        Scanner scanner = new Scanner(System.in);

        double result = algorithm.getProbabilityFromUser("Enter a probability: ", scanner);
    }
}

```

```

        assertEquals(0.75, result, 0.0001);
    }

    @Test
    public void testGetProbabilityFromUserWithInvalidInput() {
        String input = "abc\n1.5\n-0.2\n0.01";
        InputStream in = new ByteArrayInputStream(input.getBytes());
        System.setIn(in);

        GeneticAlgorithm algorithm = new GeneticAlgorithm();
        Scanner scanner = new Scanner(System.in);

        double result = algorithm.getProbabilityFromUser("Enter a probability: ", scanner);
        assertEquals(0.01, result, 0.0001);
    }

    @Test
    public void testInitializePopulation() {
        int populationSize = 10;
        List<GeneticAlgorithm.Individual> population = GeneticAlgorithm.initializePopulation(populationSize, new java.util.Random());

        assertEquals(populationSize, population.size());
        for (GeneticAlgorithm.Individual individual : population) {
            assertEquals(5, individual.getItems().size());
        }
    }

    @Test
    public void testEvaluatePopulation() {
        GeneticAlgorithm.Individual individual1 = new GeneticAlgorithm.Individual();
        individual1.getItems().add(new GeneticAlgorithm.Individual.Item(2.0, 3.0, 5.0));
        individual1.setWeightLimit(10.0);
        individual1.setVolumeLimit(5.0);

        GeneticAlgorithm.Individual individual2 = new GeneticAlgorithm.Individual();
        individual2.getItems().add(new GeneticAlgorithm.Individual.Item(1.0, 2.0, 3.0));
        individual2.setWeightLimit(5.0);
        individual2.setVolumeLimit(3.0);

        List<GeneticAlgorithm.Individual> population = List.of(individual1, individual2);

        GeneticAlgorithm.evaluatePopulation(population);

        assertEquals(5.0, individual1.getFitness(), 0.0001);
        assertEquals(3.0, individual2.getFitness(), 0.0001);
    }

    @Test
    public void testSelectParent() {
        GeneticAlgorithm.Individual individual1 = new GeneticAlgorithm.Individual();
        individual1.setFitness(5.0);

        GeneticAlgorithm.Individual individual2 = new GeneticAlgorithm.Individual();
        individual2.setFitness(3.0);
    }

```

```

GeneticAlgorithm.Individual individual3 = new GeneticAlgorithm.Individual();
individual3.setFitness(7.0);

List<GeneticAlgorithm.Individual> population = List.of(individual1, individual2, individual3);

GeneticAlgorithm.Individual selectedParent = GeneticAlgorithm.selectParent(population, new java.util.Random());

assertEquals(individual3, selectedParent);
}
@Test
public void testCrossover() {
    GeneticAlgorithm.Individual parent1 = new GeneticAlgorithm.Individual();
    parent1.getItems().add(new GeneticAlgorithm.Individual.Item(2.0, 3.0, 5.0));
    parent1.getItems().add(new GeneticAlgorithm.Individual.Item(1.0, 2.0, 3.0));
    parent1.setWeightLimit(10.0);
    parent1.setVolumeLimit(5.0);

    GeneticAlgorithm.Individual parent2 = new GeneticAlgorithm.Individual();
    parent2.getItems().add(new GeneticAlgorithm.Individual.Item(4.0, 5.0, 8.0));
    parent2.getItems().add(new GeneticAlgorithm.Individual.Item(3.0, 4.0, 6.0));
    parent2.setWeightLimit(15.0);
    parent2.setVolumeLimit(8.0);

    GeneticAlgorithm.Individual child = GeneticAlgorithm.crossover(parent1, parent2, new java.util.Random());

    assertTrue(child.getItems().containsAll(parent1.getItems()) || child.getItems().containsAll(parent2.getItems()));
}

```

3.2.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

Рисунок 3.1 – Unit tests

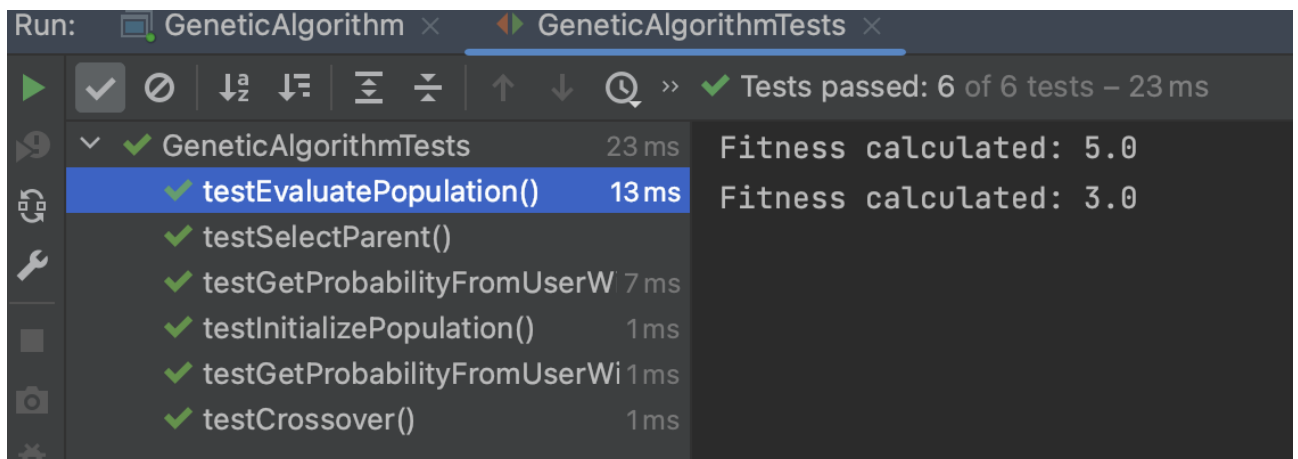
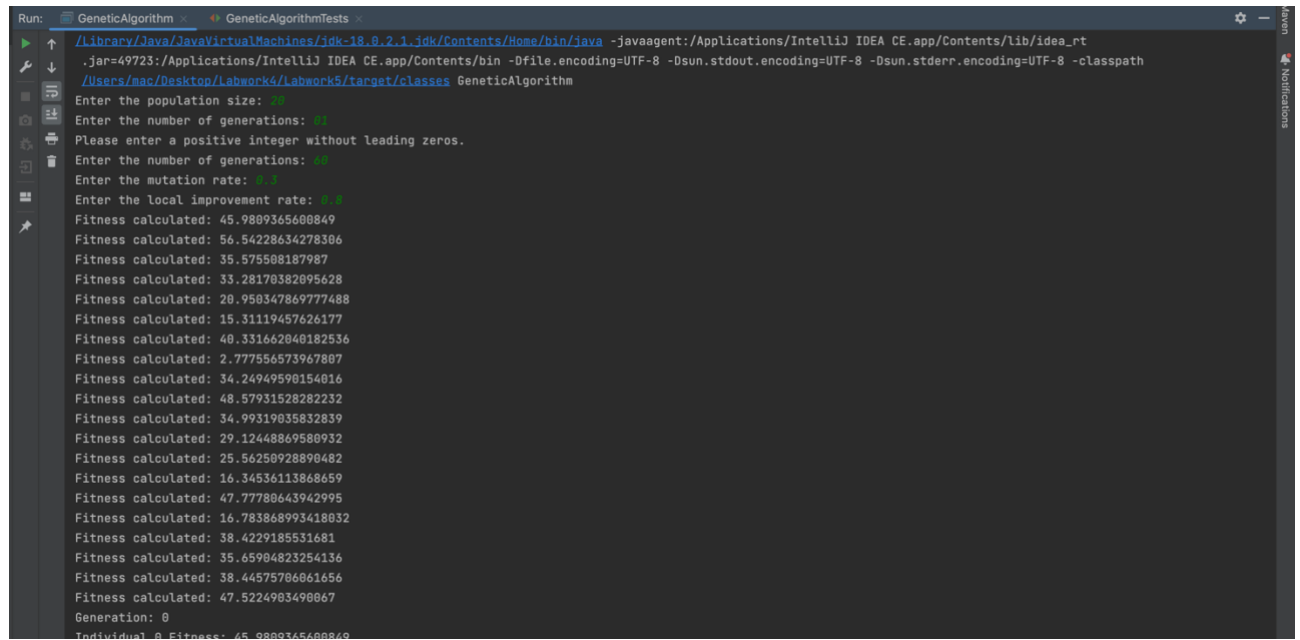


Рисунок 3.2 – Program work



```
Run: GeneticAlgorithm GeneticAlgorithmTests
/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt
.jar=49723:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath
/Users/mac/Desktop/Labwork4/Labwork5/target/classes GeneticAlgorithm
Enter the population size: 20
Enter the number of generations: 41
Please enter a positive integer without leading zeros.
Enter the number of generations: 40
Enter the mutation rate: 0.1
Enter the local improvement rate: 0.4
Fitness calculated: 45.9809365600849
Fitness calculated: 56.54228634278306
Fitness calculated: 35.575508187987
Fitness calculated: 33.28170382095628
Fitness calculated: 20.950347869777488
Fitness calculated: 15.31119457626177
Fitness calculated: 40.331662040182536
Fitness calculated: 2.777556573967807
Fitness calculated: 34.24949590154016
Fitness calculated: 48.57931528282232
Fitness calculated: 34.99319035832839
Fitness calculated: 29.12448869580932
Fitness calculated: 25.56250928890402
Fitness calculated: 16.34536113868659
Fitness calculated: 47.77780643942995
Fitness calculated: 16.783868993418032
Fitness calculated: 38.4229185531681
Fitness calculated: 35.65904823254136
Fitness calculated: 38.44575706061656
Fitness calculated: 47.5224903490067
Generation: 0
Individual 0 Fitness: 45.9809365600849
```

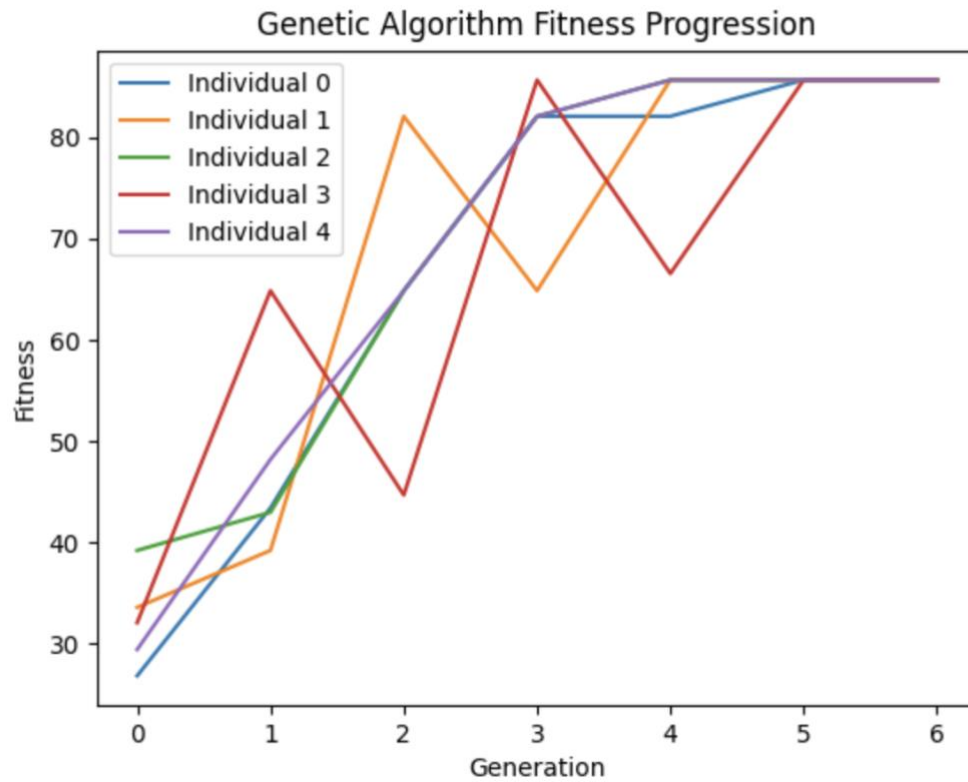
3.3 Тестування алгоритму

In each generation, the algorithm is creating a population of individuals, calculating their fitness, and evolving them through crossover and mutation.

From the plot, you can observe how the fitness values change over generations. Ideally, you would want to see an increasing trend, as this would indicate that the algorithm is converging towards better solutions. The metrics to analyze might include the maximum fitness achieved, the average fitness of the population, and the diversity of the population.

In your case, it looks like the fitness values are consistently increasing and eventually plateau, indicating that the algorithm may have found a good solution. However, further analysis and fine-tuning of parameters may be needed for a more in-depth understanding.

Рисунок 3.3 – Metrics plot



```
/Users/mac/PycharmProjects/pythonProject2/venv/bin/python /Users/mac/PycharmProjects/pythonProject2/main.py
```

Generation	Individual 0	Individual 1	Individual 2	Individual 3	Individual 4
0	26.7934	33.556	39.1892	32.0352	29.3859
1	43.3877	39.1892	42.9562	64.8669	48.1857
2	64.8669	82.1248	64.8669	44.6684	64.8669
3	82.1248	64.8669	82.1248	85.7115	82.1248
4	82.1248	85.7115	85.7115	66.5748	85.7115
5	85.7115	85.7115	85.7115	85.7115	85.7115
6	85.7115	85.7115	85.7115	85.7115	85.7115

ВИСНОВОК

В рамках даної лабораторної роботи я вивчила алгоритм генетичного пошуку та його застосування для оптимізації проблеми. Під час виконання завдання з оптимізації генетичним пошуком для певного набору параметрів, я спостерігала за динамікою поколінь та зміною значень фітнес-функцій у кожному поколінні.

Відображення цих результатів за допомогою графіків та таблиць стало ефективним інструментом для аналізу та визначення ефективності алгоритму. Я врахувала важливі параметри, такі як розмір популяції, кількість поколінь, шанс мутації та локального поліпшення. Аналізуючи отримані результати, я зрозуміла, як ці параметри впливають на швидкість збіжності алгоритму та точність знаходження оптимального рішення.

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 24.12.2023 включно максимальний бал дорівнює – 5. Після 24.12.2023 максимальний бал дорівнює – 4,5.

Критерії оцінювання у відсотках від максимального балу:

- покроковий алгоритм – 10%;
- програмна реалізація алгоритму – 45%;
- робота з гіт – 20%;
- тестування алгоритму – 20%;
- висновок – 5%.

+1 додатковий бал можна отримати за виконання та захист роботи до 17.12.2023