Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної технікиКафедра 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-24 Ротань Олександр Євгенович

Перевірив

Ахаладзе І. Е.

Київ 2023

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.

Варіант завдання

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.

Виконання

Код програми

```
package org.example.lab4.populations;


import lombok.EqualsAndHashCode;
import lombok.NonNull;
import lombok.ToString;
import org.example.lab4.backpack.BackpackPacking;
import org.example.lab4.populations.individuals.AbstractIndividual;
import org.example.lab4.populations.individuals.Individual;


import java.util.Arrays;
import java.util.TreeMap;
import java.util.stream.IntStream;
```

```
@ToString
@EqualsAndHashCode
public abstract class AbstractPopulation implements Population, ObservablePopulation {
    private Individual[] individuals;
    private final double maxWeight;
    private final double[] values;
    private final double[] weights;

    protected AbstractPopulation(@NonNull Individual[] individuals, double maxWeight, double[] values, double[] weights) {
        if (individuals.length == 0) {
            throw new IllegalArgumentException("Individuals array must not be empty");
        }
        this.individuals = individuals;
        this.maxWeight = maxWeight;
        this.values = values;
        this.weights = weights;
    }

    @Override
    public Individual getFittest() {
        return Arrays.stream(individuals).max((a, b) -> {
            var aFitness = a.getFitnessFor(values, weights, maxWeight);
            var bFitness = b.getFitnessFor(values, weights, maxWeight);
            return Double.compare(aFitness, bFitness);
        }).orElseThrow();
    }

    @Override
    public void evolve() {
        var fittest = getFittest();
        var newIndividuals = new Individual[individuals.length];
        newIndividuals[0] = fittest;
        for (int i = 1; i < individuals.length; i++) {
            var randomIndex = (int) (Math.random() * individuals.length);
            var randomIndividual = individuals[randomIndex];
            var newIndividual = fittest.crossover(randomIndividual);
            if (newIndividual.isDead(weights, maxWeight)) {
                newIndividual = fittest.clone();
            }
            newIndividual.mutate();
            if (newIndividual.isDead(weights, maxWeight)) {
                newIndividual = fittest.clone();
            }
            newIndividual.improve(values, weights, maxWeight);
            newIndividuals[i] = newIndividual;
        }
        individuals = newIndividuals;
    }

    @Override
    public void evolve(int generations) {
        for (int i = 0; i < generations; i++) {
            evolve();
        }
    }

    @Override
    public EvolutionRecap evolveObservable(int generations, int interval) {
        var recap = new EvolutionRecap(new TreeMap<>());
        for (int i = 0; i < generations; i++) {
            if ((i+1) % interval == 0) {
                var fittest = getFittest();
                recap.evolutionPath().put(i+1,
                        new BackpackPacking(IntStream.range(0, fittest.getGenes().length)
                                .filter(fittest::isGeneActive)
                                .mapToObj(j -> new BackpackPacking.BackpackItem(j, weights[j], values[j]))
                                .toList()));
            }
            evolve();
        }
        return recap;
    }
}

package org.example.lab4.populations;
```

```
import lombok.NonNull;
import org.example.lab4.populations.individuals.AbstractIndividual;
import org.example.lab4.populations.individuals.DiscreteIndividual;
import org.example.lab4.populations.individuals.Individual;

public class DiscreteGenesPopulation extends AbstractPopulation {
    public DiscreteGenesPopulation(@NonNull Individual[] individuals, double maxWeight, double[] values, double[] weights) {
        super(individuals, maxWeight, values, weights);
    }

    public static DiscreteGenesPopulation random(int size, int genesCount, double maxWeight, double[] values, double[] weights) {
        if (size <= 0) {
            throw new IllegalArgumentException("Size must be > 0");
        }
        var individuals = new Individual[size];
        for (int i = 0; i < size; i++) {
            var genes = new double[genesCount];
            var randomIndex = (int) (Math.random() * genesCount);
            genes[randomIndex] = 1;
            individuals[i] = new DiscreteIndividual(genes);
        }
        return new DiscreteGenesPopulation(individuals, maxWeight, values, weights);
    }
}
```

```
package org.example.lab4.populations.individuals;

import lombok.EqualsAndHashCode;

import java.util.Arrays;
import java.util.stream.IntStream;
@EqualsAndHashCode
public abstract class AbstractIndividual implements Individual {

    public abstract Individual clone();
```

```
private static final float ACTIVE_VALUE = 0.02f;

@Override
public boolean isGeneActive(int index) {
    return Math.abs(getGenes()[index] - 1) < ACTIVE_VALUE;
}

@Override
public double getFitnessFor(double[] values, double[] weights, double maxWeight) {
    var totalWeight = IntStream.range(0, getGenes().length).mapToDouble(i -> {
        if (Math.abs(getGenes()[i] - 1) < 0.000001) {
            return weights[i];
        }
        return 0;
    }).sum();
    var totalValue = IntStream.range(0, getGenes().length).mapToDouble(i -> {
        if (Math.abs(getGenes()[i] - 1) < 0.000001) {
            return values[i];
        }
        return 0;
    }).sum();

    if (totalWeight > maxWeight) {
        return -1;
    }
    return totalValue;
}

@Override
public void mutate() {
    if ((int)(Math.random() * 20) == 0) {
        var randomIndex = (int) (Math.random() * getGenes().length);
        var randomIndex2 = (int) (Math.random() * getGenes().length);
        while (randomIndex2 == randomIndex) {
            randomIndex2 = (int) (Math.random() * getGenes().length);
        }
        var temp = getGenes()[randomIndex];
```

```
            setGene(randomIndex, getGenes()[randomIndex2]);
            setGene(randomIndex2, temp);
        }
    }

    private void copyPartOfGenes(Individual other, int left, int right) {
        for (int i = left; i < right; i++) {
            setGene(i, other.getGenes()[i]);
        }
    }

    @Override
    public Individual crossover(Individual other) {
        var newIndividual = (AbstractIndividual) this.clone();
        var pivots = IntStream.range(0, 3).map(i -> (int) (Math.random() * getGenes().length)).sorted().toArray();
        newIndividual.copyPartOfGenes(other, 0, pivots[0]);
        newIndividual.copyPartOfGenes(other, pivots[1], pivots[2]);
        return newIndividual;
    }

    @Override
    public boolean isDead(double[] weights, double maxWeight) {
        var sum = 0d;
        for (int i = 0; i < getGenes().length; i++) {
            if (Math.abs(getGenes()[i] - 1) < ACTIVE_VALUE) {
                sum += weights[i];
            }
        }
        return sum > maxWeight;
    }

    @Override
    public void improve(double[] values, double[] weights, double maxWeight) {
        var totalWeight = IntStream.range(0, getGenes().length).mapToDouble(i -> {
            if (Math.abs(getGenes()[i] - 1) < 0.000001) {
                return weights[i];
            }
            return 0;
        }).sum();

        var min = Double.MAX_VALUE;
        var minIndex = -1;
        for (int i = 0; i < getGenes().length; i++) {
            if (Math.abs(getGenes()[i] - 1) < 0.000001) {
                continue;
            }
            var newWeight = totalWeight + weights[i];
            if (newWeight > maxWeight) {
                continue;
            }
            var newMin = maxWeight - newWeight;
            if (newMin < min) {
                min = newMin;
                minIndex = i;
            }
        }
        if (minIndex != -1) {
            setGene(minIndex, 1);
        }
    }
}

package org.example.lab4.populations.individuals;

import lombok.EqualsAndHashCode;
import lombok.ToString;

@ToString
@EqualsAndHashCode(callSuper = false)
public class DiscreteIndividual extends AbstractIndividual {
    private final boolean[] genes;
    public double[] getGenes() {
        var genes = new double[this.genes.length];
        for (int i = 0; i < genes.length; i++) {
            genes[i] = this.genes[i] ? 1d : 0d;
        }
        return genes;
    }

    @Override
    public void setGene(int index, double value) {
        this.genes[index] = Math.abs(Math.round(value) - 1d) < 0.000001;
    }

    @Override
    public Individual clone() {
        return new DiscreteIndividual(this.genes);
    }

    protected DiscreteIndividual(int size) {
        this.genes = new boolean[size];
    }

    protected DiscreteIndividual(boolean[] genes) {
        this(genes.length);
        System.arraycopy(genes, 0, this.genes, 0, genes.length);
    }

    public DiscreteIndividual(double[] genes) {
        this(toBooleanGenes(genes));
    }

    private static boolean[] toBooleanGenes(double[] genes) {
        var booleanGenes = new boolean[genes.length];
        for (int i = 0; i < genes.length; i++) {
            booleanGenes[i] = Math.abs(Math.round(genes[i]) - 1d) < 0.000001;
        }
        return booleanGenes;
    }
}

package org.example.lab4.backpack;
```

```
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.RequiredArgsConstructor;
import lombok.ToString;
import org.example.lab4.populations.DiscreteGenesPopulation;
import org.example.lab4.populations.EvolutionRecap;

import java.util.ArrayList;
import java.util.List;

public record Backpack(double[] values, double[] weights, double capacity) {

    public BackpackPacking getBestPacking() {
        var population = DiscreteGenesPopulation.random(100, values.length, capacity, values, weights);
        population.evolve(1000);
        var fittest = population.getFittest();
        var result = new ArrayList<Integer>();
        for (int i = 0; i < fittest.getGenes().length; i++) {
            if (fittest.getGenes()[i] > 0.9) {
                result.add(i);
            }
        }
        return new BackpackPacking(result.stream().map(i ->
                new BackpackPacking.BackpackItem(i, weights[i], values[i])).toList());
    }

    public EvolutionRecap getBestPackingEvolution() {
        var population = DiscreteGenesPopulation.random(100, values.length, capacity, values, weights);
        return population.evolveObservable(1000, 20);
    }

    public static Backpack random(int itemsCount, double capacity, double minValue, double maxValue,
double minWeight, double maxWeight) {
        var values = new double[itemsCount];
        var weights = new double[itemsCount];
        for (int i = 0; i < itemsCount; i++) {
            values[i] = Math.random() * maxValue + minValue;
            weights[i] = Math.random() * maxWeight + minWeight;
        }
        return new Backpack(values, weights, capacity);
    }
}
```

Приклади роботи

1. Backpack:

```
Item 0: value=4.914693246743952, weight=9.254244964361757
Item 1: value=8.232670008403565, weight=15.529011891852262
Item 2: value=31.62854232344185, weight=2.1821368748921204
Item 3: value=19.71358228101507, weight=15.308425146559516
Item 4: value=13.323540508989504, weight=8.093775170029412
Item 5: value=16.55130568560048, weight=14.51848512758395
Item 6: value=4.6253421514088595, weight=7.935687303998386
Item 7: value=22.76013059952173, weight=20.621496000725294
Item 8: value=27.868208831866397, weight=16.655614679421944
Item 9: value=11.378531311400835, weight=7.43107709294034
Item 10: value=5.350685821440492, weight=1.538654059583668
Item 11: value=19.075813474210108, weight=4.569915035205046
Item 12: value=10.848511193253335, weight=17.6433752113429
Item 13: value=16.483740049613964, weight=17.806557721789158
Item 14: value=2.3119361325728756, weight=15.294733116664373
Item 15: value=26.15590245644076, weight=14.341842681071784
Item 16: value=23.485197633128205, weight=20.513446412914696
Item 17: value=31.33689849121181, weight=6.5540212116978624
Item 18: value=6.433915967413309, weight=23.46560322576967
Item 19: value=3.440194524189764, weight=17.931363763293575
Item 20: value=29.651745398568035, weight=7.501632280246442
Item 21: value=14.287821223344245, weight=15.922022336464874
Item 22: value=17.540646434316816, weight=12.120346492649382
Item 23: value=30.75686947497051, weight=23.21313818397222
Item 24: value=26.6197063781322, weight=22.241645878271576
Item 25: value=24.507040257515396, weight=21.51466288981878
Item 26: value=21.815468199857794, weight=13.500553738512963
Item 27: value=17.93029385318062, weight=15.268095692050828
Item 28: value=12.622285448340437, weight=8.8856128311899
Item 29: value=19.367961407759253, weight=1.8263025037286487
Item 30: value=3.794254782422212, weight=15.93627354070324
Item 31: value=5.9824494769941925, weight=24.55782276620307
Item 32: value=28.540849394736394, weight=15.88675314706114
Item 33: value=11.449990433679925, weight=16.01753576787535
Item 34: value=3.946386009334619, weight=16.59795649626044
Item 35: value=27.85767681161899, weight=19.795294946155433
Item 36: value=8.519044724114103, weight=9.854382348494324
Item 37: value=6.842314110154783, weight=11.905100021707723
Item 38: value=20.398180350061605, weight=15.61955975500915
Item 39: value=4.10042533563602, weight=25.60962368704539
Item 40: value=21.80921576025359, weight=1.1413997965995062
Item 41: value=23.928038484581236, weight=17.388367107768858
Item 42: value=10.853471985781955, weight=6.377954982720564
Item 43: value=6.415688853783152, weight=21.352457847286292
Item 44: value=25.65580871581783, weight=14.074416542281531
Item 45: value=19.328350484484904, weight=24.78507544834227
Item 46: value=18.26601948144209, weight=22.963811320240726
Item 47: value=29.54182483194022, weight=8.64371093345159
Item 48: value=26.52045693255334, weight=15.986674014500542
Item 49: value=5.186406700991763, weight=8.798961135774066
Item 50: value=31.318427226721102, weight=12.129098920575736
Item 51: value=9.365432801950735, weight=16.504257379145848
Item 52: value=24.45341711271001, weight=25.46392059447559
Item 53: value=19.47063361589297, weight=25.09193676167586
Item 54: value=8.262282381760794, weight=3.246367307224817
Item 55: value=22.70799113555519, weight=19.173262462117226
Item 56: value=22.916068050593875, weight=1.46339012113997
Item 57: value=8.47711515812768, weight=17.68348915508701
Item 58: value=20.174175753592152, weight=6.064969281461701
Item 59: value=11.71393014079741, weight=22.957804224004562
Item 60: value=3.859136546267861, weight=21.216050680676265
Item 61: value=18.210777561915563, weight=19.64993190254933
Item 62: value=27.18231045670071, weight=22.2993695344295
Item 63: value=3.269834745208689, weight=9.712431321612582
Item 64: value=15.361354187204682, weight=11.07226679411714
Item 65: value=18.82503243669742, weight=12.846187731727802
Item 66: value=8.71837728898096, weight=13.334704780194732
Item 67: value=31.38446611851951, weight=5.796325791185743
Item 68: value=16.633222867434363, weight=25.70104577727854
Item 69: value=13.799591373876812, weight=18.622747248607684
Item 70: value=6.349383465681039, weight=22.58474928052235
Item 71: value=27.959279182432027, weight=1.6057241540644187
Item 72: value=2.690583118741767, weight=2.30301008315975
Item 73: value=13.03611724735207, weight=22.76045955212747
Item 74: value=5.244129959359343, weight=7.28743860767878
Item 75: value=14.879133630347585, weight=3.0292429346587038
Item 76: value=8.502666923252686, weight=22.747263607109254
Item 77: value=26.55141502751298, weight=19.09581593117489
Item 78: value=7.82901403017051, weight=15.77311389792807
Item 79: value=7.852888145075302, weight=2.4237834395426976
Item 80: value=19.919326623272198, weight=22.792302519323037
Item 81: value=2.9478440294977752, weight=12.559382354069767
Item 82: value=30.68785599429856, weight=2.4906370205893915
Item 83: value=9.933550626982598, weight=12.958653486701376
Item 84: value=6.182128526753866, weight=18.801123517548554
Item 85: value=4.340637204543632, weight=3.600029103209489
Item 86: value=3.260428826511318, weight=4.326538729056475
Item 87: value=30.147688202210976, weight=16.425575532062
Item 88: value=26.323990977840324, weight=3.6106988772770428
Item 89: value=16.92181900647524, weight=7.217625744723435
Item 90: value=4.394380033405021, weight=8.00234891479569
Item 91: value=10.163774964318595, weight=5.862228754361442
Item 92: value=9.262063313963468, weight=19.465954614755546
Item 93: value=3.8404063245775584, weight=16.076851834721644
Item 94: value=20.305239368980182, weight=19.585271589516825
Item 95: value=29.429168478480122, weight=10.58698616357823
Item 96: value=14.888530619845332, weight=22.178429149475473
Item 97: value=4.57858200324055, weight=23.42477311179725
Item 98: value=4.999189982694462, weight=3.397358000617164
Item 99: value=26.705402165819546, weight=22.492506401174126
Capacity: 250.0

Best fit: BackpackPacking[
items=[BackpackItem[index=2, weight=2.1821368748921204, value=31.62854232344185],
BackpackItem[index=4, weight=8.093775170029412, value=13.323540508989504],
BackpackItem[index=8, weight=16.655614679421944, value=27.868208831866397],
BackpackItem[index=9, weight=7.43107709294034, value=11.378531311400835],
BackpackItem[index=10, weight=1.538654059583668, value=5.350685821440492],
BackpackItem[index=11, weight=4.569915035205046, value=19.075813474210108],
BackpackItem[index=15, weight=14.341842681071784, value=26.15590245644076],
BackpackItem[index=17, weight=6.5540212116978624, value=31.33689849121181],
BackpackItem[index=20, weight=7.501632280246442, value=29.651745398568035],
BackpackItem[index=26, weight=13.500553738512963, value=21.815468199857794],
BackpackItem[index=28, weight=8.8856128311899, value=12.622285448340437],
BackpackItem[index=29, weight=1.8263025037286487, value=19.367961407759253],
BackpackItem[index=32, weight=15.88675314706114, value=28.540849394736394],
BackpackItem[index=40, weight=1.1413997965995062, value=21.80921576025359],
BackpackItem[index=42, weight=6.377954982720564, value=10.853471985781955],
BackpackItem[index=47, weight=8.64371093345159, value=29.54182483194022],
BackpackItem[index=48, weight=15.986674014500542, value=26.52045693255334],
BackpackItem[index=50, weight=12.129098920575736, value=31.318427226721102],
BackpackItem[index=54, weight=3.246367307224817, value=8.262282381760794],
BackpackItem[index=56, weight=1.46339012113997, value=22.916068050593875],
BackpackItem[index=58, weight=6.064969281461701, value=20.174175753592152],
BackpackItem[index=64, weight=11.07226679411714, value=15.361354187204682],
BackpackItem[index=67, weight=5.796325791185743, value=31.38446611851951],
BackpackItem[index=71, weight=1.6057241540644187, value=27.959279182432027],
BackpackItem[index=72, weight=2.30301008315975, value=2.690583118741767],
BackpackItem[index=75, weight=3.0292429346587038, value=14.879133630347585],
BackpackItem[index=79, weight=2.4237834395426976, value=7.852888145075302],
BackpackItem[index=82, weight=2.4906370205893915, value=30.68785599429856],
BackpackItem[index=85, weight=3.600029103209489, value=4.340637204543632],
BackpackItem[index=86, weight=4.326538729056475, value=3.260428826511318],
BackpackItem[index=87, weight=16.425575532062, value=30.147688202210976],
BackpackItem[index=88, weight=3.6106988772770428, value=26.323990977840324],
BackpackItem[index=89, weight=7.217625744723435, value=16.92181900647524],
BackpackItem[index=90, weight=8.00234891479569, value=4.394380033405021],
BackpackItem[index=95, weight=10.58698616357823, value=29.429168478480122],
BackpackItem[index=98, weight=3.397358000617164, value=4.999189982694462]],
value=700.1452190802412, weight=249.90960794589307]
```

2. Bacpack:

```
Item 0: value=2.5887647702077103, weight=2.4881566412925844
Item 1: value=11.529646161837283, weight=12.920426288881494
Item 2: value=13.056958997241942, weight=24.839860152071868
Item 3: value=27.87593444039335, weight=20.574861821355817
Item 4: value=16.613988064137935, weight=3.200142746480055
Item 5: value=13.763450072818248, weight=5.699974371308838
Item 6: value=21.1491709123256, weight=22.291455653539522
Item 7: value=24.455279864998918, weight=23.468015939227868
Item 8: value=7.355021885454532, weight=25.223131759745826
Item 9: value=9.297110964568676, weight=1.9596112770496408
Item 10: value=10.040719446475679, weight=14.344186307677575
Item 11: value=24.597142785584794, weight=5.159701800651624
Item 12: value=14.796099893938495, weight=1.3826034930570155
Item 13: value=24.684129651722643, weight=24.88208584000368
Item 14: value=28.285898598900733, weight=11.425088372821644
Item 15: value=24.78949189973055, weight=6.969987278476505
Item 16: value=23.29575763366941, weight=6.448508739924388
Item 17: value=30.209962973467153, weight=25.542949689658215
Item 18: value=25.930821341749812, weight=6.052700076419561
Item 19: value=15.45532242386309, weight=15.764456926089695
Item 20: value=2.0845307353416325, weight=20.764063418099827
Item 21: value=24.771274541857416, weight=10.590021514418897
Item 22: value=23.449530871613085, weight=1.78033733017199
Item 23: value=21.550597888753817, weight=25.22208859538772
Item 24: value=29.934399753027563, weight=25.309170296918495
Item 25: value=28.706244749262517, weight=19.54596508398912
Item 26: value=5.379202654960407, weight=19.716581182948588
Item 27: value=15.078090506103116, weight=18.65006358963516
Item 28: value=26.929676576073355, weight=9.546134124259513
Item 29: value=4.380612763583555, weight=9.644126659354923
Item 30: value=25.33413919312257, weight=17.095627413064967
Item 31: value=23.07179713756442, weight=9.380674839049629
Item 32: value=3.7690225511878555, weight=4.645313336960496
Item 33: value=31.834138997863942, weight=12.357796138516285
Item 34: value=26.578228251739915, weight=8.620161048095998
Item 35: value=12.106629019956035, weight=6.035477190987966
Item 36: value=6.528276938638509, weight=10.938378453743669
Item 37: value=24.29219095778527, weight=7.03304147807662
Item 38: value=29.83556670380046, weight=12.549408262720343
Item 39: value=7.888054289669638, weight=1.1918489591855836
Item 40: value=22.313791425550583, weight=19.696822383398548
Item 41: value=15.847117484339714, weight=13.285279807508804
Item 42: value=27.51892900133665, weight=24.755022668571844
```

Item 43: value=15.97638497801853, weight=13.23710789260133

Item 44: value=28.735954014089177, weight=10.5833866005499

Item 45: value=21.11411854200313, weight=10.069734962474115

Item 46: value=11.055766108131433, weight=5.21746198561837

Item 47: value=7.200394896609366, weight=17.44535089458441

Item 48: value=17.63259107848041, weight=12.374053450897884

Item 49: value=4.16628562798375, weight=20.58594131498654

Item 50: value=25.49143290748327, weight=17.685511550250258

Item 51: value=28.814957625788846, weight=8.562161211709196

Item 52: value=10.971934708447343, weight=20.798265371762312

Item 53: value=21.553504083133962, weight=8.426448655940721

Item 54: value=3.049317623922751, weight=19.471613443566678

Item 55: value=29.95738722545297, weight=18.432350746613842

Item 56: value=4.3648439094996645, weight=23.5950237112567

Item 57: value=11.709909143906321, weight=11.629887821531609

Item 58: value=4.8275755197614725, weight=4.605650058196891

Item 59: value=17.457176469205763, weight=18.455884753214352

Item 60: value=21.115964318961062, weight=14.132527000601371

Item 61: value=30.554575616553073, weight=22.548545299825964

Item 62: value=7.312138840743776, weight=9.819850014106024

Item 63: value=25.553812762463572, weight=17.907150049371985

Item 64: value=17.243517466864866, weight=6.18900378434001

Item 65: value=30.59013278943287, weight=2.690613028271759

Item 66: value=27.88570549519957, weight=24.087458722464262

Item 67: value=20.57736408680013, weight=25.402722759817802

Item 68: value=15.397404911154252, weight=18.742390363478748

Item 69: value=28.3039940758729, weight=2.947262922257381

Item 70: value=11.012381346695172, weight=23.687845419583933

Item 71: value=19.14652161313367, weight=22.05278871503816

Item 72: value=13.105314923556195, weight=5.86652215362163

Item 73: value=12.176882051096724, weight=15.86470396360339

Item 74: value=14.277672122575275, weight=11.020685430977561

Item 75: value=15.93878439701133, weight=1.0813760579507161

Item 76: value=28.27651996129198, weight=13.081482121334608

Item 77: value=31.10334753988008, weight=17.871378653414823

Item 78: value=10.963510770746547, weight=16.64189153645107

Item 79: value=14.587477045423027, weight=9.950556350571473

Item 80: value=22.393055059685032, weight=22.755770716728

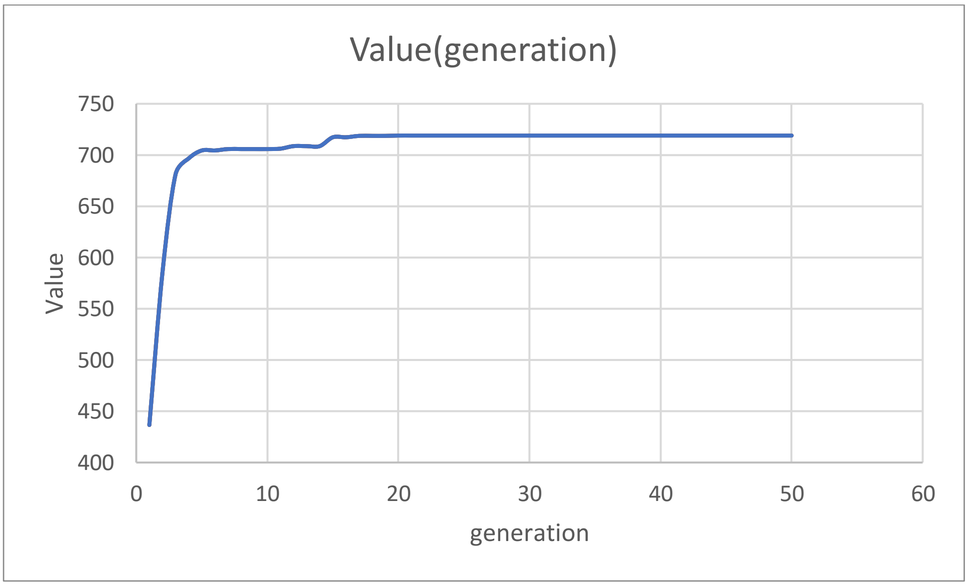
```
Item 81: value=10.528955595335365, weight=15.62045609938332
Item 82: value=13.33800307956816, weight=5.24393710783999
Item 83: value=12.760283309708523, weight=12.273766840715421
Item 84: value=19.317952640336046, weight=7.407949519921788
Item 85: value=21.108839484178667, weight=16.857773885354135
Item 86: value=28.339583875205026, weight=7.553234273225519
Item 87: value=9.973988505646549, weight=18.053413719957085
Item 88: value=25.15024286890669, weight=16.00541836277083
Item 89: value=22.192302894954803, weight=2.4753925125053553
Item 90: value=24.835269964070203, weight=6.367222258960799
Item 91: value=24.2001134917723, weight=12.027911826861194
Item 92: value=22.056382763723896, weight=24.60797683966271
Item 93: value=29.93815829704662, weight=7.66090953081245
Item 94: value=9.132700123055535, weight=11.203818918795902
Item 95: value=20.81613109535546, weight=16.163316342626057
Item 96: value=16.507822518693786, weight=13.953832390204079
Item 97: value=17.365492172262652, weight=1.3026943561778102
Item 98: value=12.925640482067342, weight=12.285896748160917
Item 99: value=24.95694838763727, weight=3.574701374356186
Capacity: 250.0

Best fit: BackpackPacking[
items=[BackpackItem[index=0, weight=2.4881566412925844, value=2.5887647702077103],
BackpackItem[index=4, weight=3.200142746480055, value=16.613988064137935],
BackpackItem[index=5, weight=5.699974371308838, value=13.763450072818248],
BackpackItem[index=9, weight=1.9596112770496408, value=9.297110964568676],
BackpackItem[index=11, weight=5.159701800651624, value=24.597142785584794],
BackpackItem[index=12, weight=1.3826034930570155, value=14.796099893938495],
BackpackItem[index=14, weight=11.425088372821644, value=28.285898598900733],
BackpackItem[index=15, weight=6.969987278476505, value=24.78949189973055],
BackpackItem[index=16, weight=6.448508739924388, value=23.29575763366941],
BackpackItem[index=18, weight=6.052700076419561, value=25.930821341749812],
BackpackItem[index=21, weight=10.590021514418897, value=24.771274541857416],
BackpackItem[index=22, weight=1.78033733017199, value=23.449530871613085],
BackpackItem[index=28, weight=9.546134124259513, value=26.929676576073355],
BackpackItem[index=31, weight=9.380674839049629, value=23.07179713756442],
BackpackItem[index=33, weight=12.357796138516285, value=31.834138997863942],
BackpackItem[index=34, weight=8.620161048095998, value=26.578228251739915],
BackpackItem[index=35, weight=6.035477190987966, value=12.106629019956035],
BackpackItem[index=37, weight=7.03304147807662, value=24.29219095778527],
BackpackItem[index=38, weight=12.549408262720343, value=29.83556670380046],
BackpackItem[index=39, weight=1.1918489591855836, value=7.888054289669638],
BackpackItem[index=44, weight=10.5833866005499, value=28.735954014089177],
BackpackItem[index=45, weight=10.069734962474115, value=21.11411854200313],
BackpackItem[index=46, weight=5.21746198561837, value=11.055766108131433],
BackpackItem[index=51, weight=8.562161211709196, value=28.814957625788846],
BackpackItem[index=53, weight=8.426448655940721, value=21.553504083133962],
BackpackItem[index=58, weight=4.605650058196891, value=4.8275755197614725],
BackpackItem[index=64, weight=6.18900378434001, value=17.243517466864866],
BackpackItem[index=65, weight=2.690613028271759, value=30.59013278943287],
BackpackItem[index=69, weight=2.947262922257381, value=28.3039940758729],
BackpackItem[index=72, weight=5.86652215362163, value=13.105314923556195],
BackpackItem[index=75, weight=1.0813760579507161, value=15.93878439701133],
BackpackItem[index=82, weight=5.24393710783999, value=13.33800307956816],
BackpackItem[index=84, weight=7.407949519921788, value=19.317952640336046],
BackpackItem[index=86, weight=7.553234273225519, value=28.339583875205026],
BackpackItem[index=89, weight=2.4753925125053553, value=22.192302894954803],
BackpackItem[index=90, weight=6.367222258960799, value=24.835269964070203],
BackpackItem[index=93, weight=7.66090953081245, value=29.93815829704662],
BackpackItem[index=97, weight=1.3026943561778102, value=17.365492172262652],
BackpackItem[index=98, weight=12.285896748160917, value=12.925640482067342],
BackpackItem[index=99, weight=3.574701374356186, value=24.95694838763727]],
value=829.2085847120242, weight=249.98293478585617]
```

Тестування алгоритму

```
Generation 20:value=436.590743032673, weight=249.74148051481944
Generation 40:value=585.2648090845257, weight=249.98574448317694
Generation 60:value=681.2229474182017, weight=249.76508851178014
Generation 80:value=696.6945722690975, weight=249.66067561986102
Generation 100:value=699.0204895056906, weight=245.0649381010132
Generation 120:value=704.463178947671, weight=247.22908879428573
Generation 140:value=704.463178947671, weight=247.22908879428573
Generation 160:value=705.8422739211848, weight=249.77726675040518
Generation 180:value=705.8422739211848, weight=249.77726675040518
Generation 200:value=705.8422739211848, weight=249.77726675040518
Generation 220:value=705.8422739211848, weight=249.77726675040518
Generation 240:value=706.2020349498653, weight=246.36423225574083
Generation 260:value=708.7011548126559, weight=247.45798506755543
Generation 280:value=708.7011548126559, weight=247.45798506755543
Generation 300:value=708.7011548126559, weight=247.45798506755543
Generation 320:value=717.3277009598855, weight=249.73855365595892
Generation 340:value=717.3277009598855, weight=249.73855365595892
Generation 360:value=718.6742492106459, weight=248.70971917638272
Generation 380:value=718.6742492106459, weight=248.70971917638272
Generation 400:value=718.6742492106459, weight=248.70971917638272
Generation 420:value=718.9903899561464, weight=249.89238836998922
Generation 440:value=718.9903899561464, weight=249.89238836998922
Generation 460:value=718.9903899561464, weight=249.89238836998922
Generation 480:value=718.9903899561464, weight=249.89238836998922
Generation 500:value=718.9903899561464, weight=249.89238836998922
Generation 520:value=718.9903899561464, weight=249.89238836998922
Generation 540:value=718.9903899561464, weight=249.89238836998922
Generation 560:value=718.9903899561464, weight=249.89238836998922
Generation 580:value=718.9903899561464, weight=249.89238836998922
Generation 600:value=718.9903899561464, weight=249.89238836998922
Generation 620:value=718.9903899561464, weight=249.89238836998922
Generation 640:value=718.9903899561464, weight=249.89238836998922
Generation 660:value=718.9903899561464, weight=249.89238836998922
Generation 680:value=718.9903899561464, weight=249.89238836998922
Generation 700:value=718.9903899561464, weight=249.89238836998922
Generation 720:value=718.9903899561464, weight=249.89238836998922
Generation 740:value=718.9903899561464, weight=249.89238836998922
Generation 760:value=718.9903899561464, weight=249.89238836998922
Generation 780:value=718.9903899561464, weight=249.89238836998922
Generation 800:value=718.9903899561464, weight=249.89238836998922
Generation 820:value=718.9903899561464, weight=249.89238836998922
Generation 840:value=718.9903899561464, weight=249.89238836998922
Generation 860:value=718.9903899561464, weight=249.89238836998922
Generation 880:value=718.9903899561464, weight=249.89238836998922
Generation 900:value=718.9903899561464, weight=249.89238836998922
Generation 920:value=718.9903899561464, weight=249.89238836998922
Generation 940:value=718.9903899561464, weight=249.89238836998922
Generation 960:value=718.9903899561464, weight=249.89238836998922
Generation 980:value=718.9903899561464, weight=249.89238836998922
Generation 1000:value=718.9903899561464, weight=249.89238836998922
```

Графік залежності



Висновок

Виконання цієї лабораторної роботи дозволило глибше зрозуміти основні принципи формалізації метаевристичних алгоритмів, зокрема, через розробку та програмну реалізацію генетичного алгоритму для розв'язання задачі про рюкзак. Використання генетичного алгоритму з початковою популяцією 100 осіб, триточковим оператором схрещування та мутацією з ймовірністю 5% ефективно демонструє, як метаевристичні підходи можуть оптимізувати пошук рішень у складних задачах. Особливо цінним є розробка власного оператора локального покращення, що додало унікальності до стандартного процесу генетичного алгоритму. Аналіз якості розв'язків на різних етапах ітерацій і побудова графіка залежності якості розв'язку від числа ітерацій дозволили візуально оцінити ефективність алгоритму та його здатність до оптимізації. Такий підхід підкреслює важливість ітеративного покращення та налаштування параметрів алгоритму для досягнення оптимальних результатів.