**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 № 3 з дисципліни

«Проектування алгоритмів»

„**Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.1**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-01 Пашковський Євгеній Сергійович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Вєчерковська А. С.*

Київ 2021

Зміст

[1 Мета лабораторної роботи 3](#_Toc51260917)

[2 Завдання 4](#_Toc51260918)

[3 Виконання 9](#_Toc51260919)

[3.1 Програмна реалізація алгоритму 9](#_Toc51260920)

[3.1.1 Вихідний код 9](#_Toc51260921)

[3.1.2 Приклади роботи 9](#_Toc51260922)

[3.2 Тестування алгоритму 10](#_Toc51260923)

[3.2.1 Значення цільової функції зі збільшенням кількості ітерацій 10](#_Toc51260924)

[3.2.2 Графіки залежності розв'язку від числа ітерацій 10](#_Toc51260925)

[Висновок 11](#_Toc51260926)

[Критерії оцінювання 12](#_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** |
| 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

## Програмна реалізація алгоритму

### Вихідний код

'use strict';

const generateRandom = require('./generateRandom.js');

const Population = require('./Population.js');

const fs = require('fs');

const P = 250;

const initialPopulationCount = 100;

const crossbreedingOperator = 0.25;

const mutationProbability = 0.05;

const iterations = 1000;

class Lab3 {

start() {

const population = new Population(P, initialPopulationCount);

fs.writeFileSync('./log.log', '');

let bestEntityEver = null;

for (let i = 0; i < iterations; i++) {

if (population.entities.length === 0) break;

const bestEntity = population.bestEntity;

if (!bestEntityEver || bestEntity.value > bestEntityEver.value) {

bestEntityEver = bestEntity;

}

const father1 = bestEntity;

const randomIndex = Math.round(

generateRandom(0, population.entities.length - 1)

);

const father2 = population.entities[randomIndex];

population.doCrossbreeding(

father1,

father2,

crossbreedingOperator,

mutationProbability

);

population.killWorst();

if (i % 20 === 0)

fs.appendFileSync(

'./log.log',

bestEntityEver.items

.map(el => el.value)

.reduce((acc, el) => acc + el) +

' ' +

bestEntityEver.weight +

' ' +

population.entities.length +

'\n'

);

}

console.log('Population left: ' + population.entities.length);

console.log('Value: ' + bestEntityEver.value);

console.log('Weight: ' + bestEntityEver.weight);

console.log(bestEntityEver);

process.exit();

}

}

module.exports = Lab3;

'use strict';

class Item {

constructor(weight, value) {

this.weight = weight;

this.value = value;

}

}

module.exports = Item;

'use strict';

module.exports = (min = 0, max = 1) => Math.random() \* (max - min) + min;

'use strict';

const generateRandom = require('./generateRandom.js');

const Entity = require('./Entity.js');

const Item = require('./Item.js');

const itemsCount = 100;

const weight = { min: 1, max: 25 };

const value = { min: 2, max: 30 };

class Population {

#mapping = [];

constructor(maxWeight, initialCount) {

for (let i = 0; i < itemsCount; i++) {

this.#mapping.push(

new Item(

Math.round(generateRandom(weight.min, weight.max)),

Math.round(generateRandom(value.min, value.max))

)

);

}

this.maxWeight = maxWeight;

this.entities = [];

for (let i = 0; i < initialCount; i++) {

const genes = [];

for (let j = 0; j < initialCount; j++) {

genes.push(i === j ? 1 : 0);

}

this.entities.push(new Entity(genes, this.#getItems(genes)));

}

}

#getItems(genes) {

return genes

.map((el, i) => (el === 1 ? this.#mapping[i] : null))

.filter(el => el !== null);

}

#useImprovementOperator(genes) {

const lightWeightItemToAdd = genes

.map((el, i) => (el === 0 ? this.#mapping[i] : null))

.filter(el => el !== null)

.reduce((acc, el) => (acc.weight < el.weight ? acc : el));

genes[this.#mapping.indexOf(lightWeightItemToAdd)] = 1;

if (new Entity(genes, this.#getItems(genes)).weight > this.maxWeight) {

return false;

}

return genes;

}

doCrossbreeding(

father1,

father2,

crossbreedingOperator,

mutationProbability = 0

) {

const crossbreedingSize = itemsCount \* crossbreedingOperator;

const crossbreedingPieces = 1 / crossbreedingOperator;

const genes1 = new Array(...father1.genes);

const genes2 = new Array(...father2.genes);

let newGenes = [];

let flag = false;

let offset = 0;

for (let i = 0; i < crossbreedingPieces; i++) {

const gene = flag ? genes2 : genes1;

newGenes.push(...gene.slice(offset, offset + crossbreedingSize));

offset += crossbreedingSize;

flag = !flag;

}

const possibleGene = this.#useImprovementOperator(newGenes);

if (!possibleGene) {

if (Math.random() < mutationProbability) {

const i = Math.round(generateRandom(0, itemsCount - 1));

const j = Math.round(generateRandom(0, itemsCount - 1));

const buf = newGenes[i];

newGenes[i] = newGenes[j];

newGenes[j] = buf;

}

} else {

newGenes = possibleGene;

}

const newEntity = new Entity(newGenes, this.#getItems(newGenes));

if (newEntity.weight > this.maxWeight) {

return false;

}

this.entities.push(newEntity);

return newEntity;

}

killWorst() {

this.entities.splice(this.entities.indexOf(this.worstEntity), 1);

}

get bestEntity() {

return this.entities.reduce((acc, el) => (el.value > acc.value ? el : acc));

}

get worstEntity() {

return this.entities.reduce((acc, el) => (el.value < acc.value ? el : acc));

}

}

module.exports = Population;

'use strict';

class Entity {

constructor(genes, items) {

this.genes = genes;

this.items = items;

}

get weight() {

let w = 0;

for (const item of this.items) {

w += item.weight;

}

return w;

}

get value() {

let v = 0;

for (const item of this.items) {

v += item.value;

}

return v;

}

}

module.exports = Entity;

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

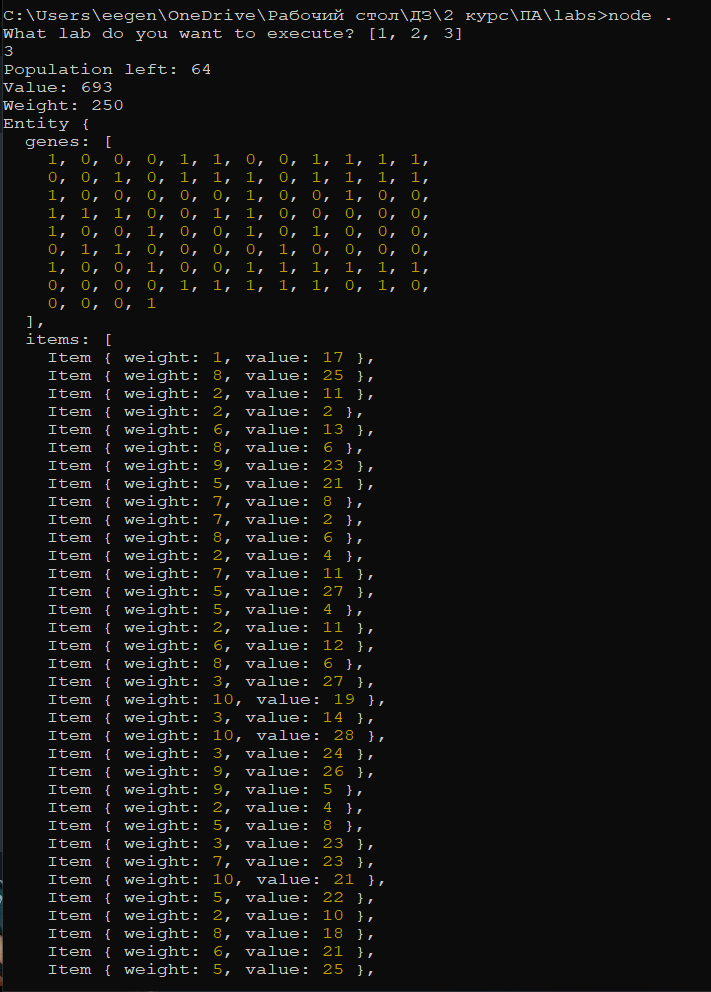


Рисунок 3.1

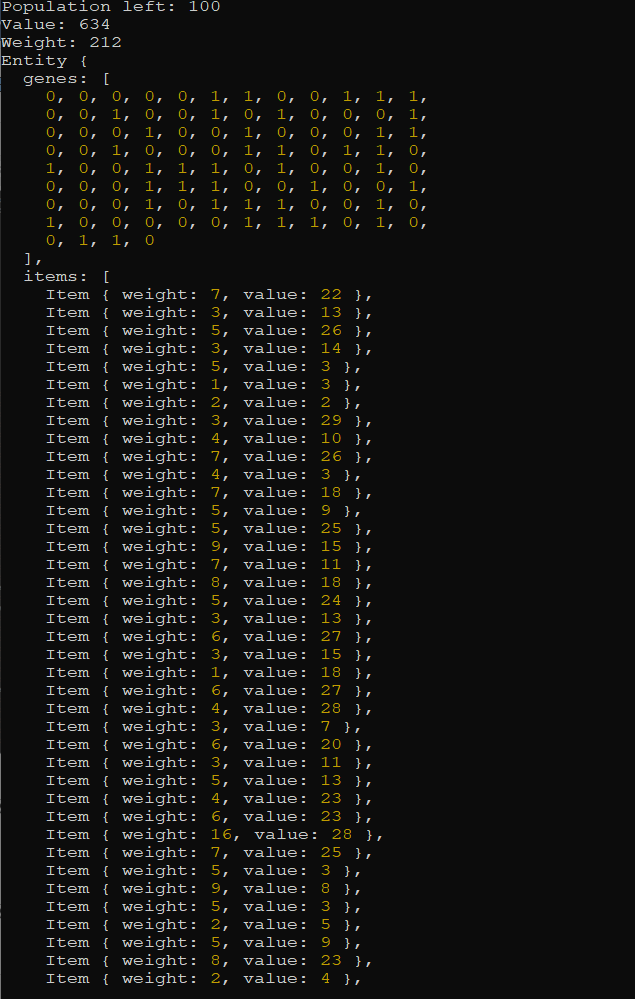


Рисунок 3.2

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

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

|  |  |
| --- | --- |
| Кількість ітерацій | Значення функції (загальна цінність) |
| 20 | 30 |
| 40 | 53 |
| 60 | 55 |
| 80 | 60 |
| 100 | 60 |
| 120 | 60 |
| 140 | 60 |
| 160 | 60 |
| 180 | 60 |
| 200 | 60 |
| 220 | 60 |
| 240 | 60 |
| 260 | 106 |
| 280 | 162 |
| 300 | 218 |
| 320 | 243 |
| 340 | 243 |
| 360 | 250 |
| 380 | 253 |
| 400 | 253 |
| 420 | 256 |
| 440 | 256 |
| 460 | 266 |
| 480 | 293 |
| 500 | 318 |
| 520 | 318 |
| 540 | 322 |
| 560 | 327 |
| 580 | 327 |
| 600 | 327 |
| 620 | 356 |
| 640 | 379 |
| 660 | 399 |
| 680 | 478 |
| 700 | 489 |
| 720 | 507 |
| 740 | 507 |
| 760 | 507 |
| 780 | 507 |
| 800 | 507 |
| 820 | 507 |
| 840 | 555 |
| 860 | 555 |
| 880 | 593 |
| 900 | 593 |
| 920 | 611 |
| 940 | 619 |
| 960 | 619 |
| 980 | 619 |
| 1000 | 634 |

### 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

В рамках даної лабораторної роботи…

Критерії оцінювання

При здачі лабораторної роботи до 5.11.2021 включно максимальний бал дорівнює – 5. Після 5.11.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* програмна реалізація алгоритму – 75%;
* тестування алгоритму– 20%;
* висновок – 5%.