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) *ІП-13 Замковий Д.В.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов О.О.* _____
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи формалізації метаевристичних алгоритмів і вирішення типових задач з їхньою допомогою.

2 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**
1	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.
2	Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0{,}4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 30, починають маршрут в різних випадкових вершинах).
3	Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники).
4	Задача про рюкзак (місткість P=200, 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 двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.

5	Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах M = 35, починають маршрут в різних випадкових вершинах).
6	Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники).
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.
8	Задача комівояжера (200 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 3, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах M = 45, починають маршрут в різних випадкових вершинах).
9	Задача розфарбовування графу (150 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 25 із них 3 розвідники).
10	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 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
11	Задача комівояжера (250 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 2, β = 4, ρ

	= 0,6, Lmin знайти жадібним алгоритмом, кількість мурах M = 45, починають маршрут в різних випадкових вершинах).
12	Задача розфарбовування графу (300 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 60 із них 5 розвідники).
13	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 одноточковий 30% і 70%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
14	Задача комівояжера (250 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 4, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах M = 45 (10 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах).
15	Задача розфарбовування графу (100 вершин, степінь вершини не більше 20, але не менше 1), класичний бджолиний алгоритм (число бджіл 30 із них 3 розвідники).
16	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 двоточковий 30%, 40% і 30%, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
17	Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 2, β = 4, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах M = 45 (15 з них дикі,

	обирають випадкові напрямки), починають маршрут в різних випадкових вершинах).
18	Задача розфарбовування графу (300 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 60 із них 5 розвідники).
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.
20	Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм ($\alpha = 3$, $\beta = 2$, $\rho = 0,7$, Lmin знайти жадібним алгоритмом, кількість мурах M = 45 (10 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах).
21	Задача розфарбовування графу (200 вершин, степінь вершини не більше 30, але не менше 1), класичний бджолиний алгоритм (число бджіл 40 із них 2 розвідники).
22	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% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
23	Задача комівояжера (300 вершин, відстань між вершинами випадкова від 1 до 60), мурашиний алгоритм ($\alpha = 3$, $\beta = 2$, $\rho = 0,6$, Lmin знайти жадібним алгоритмом, кількість мурах M = 45 (15 з них елітні,

	подвійний феромон), починають маршрут в різних випадкових вершинах).
24	Задача розфарбовування графу (400 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 70 із них 10 розвідники).
25	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.
26	Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 30, починають маршрут в різних випадкових вершинах).
27	Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники).
28	Задача про рюкзак (місткість P=200, 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 двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
29	Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 3$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 35, починають маршрут в різних випадкових вершинах).

30	Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники).
31	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.
32	Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0{,}4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 30, починають маршрут в різних випадкових вершинах).
33	Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники).
34	Задача про рюкзак (місткість P=200, 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 двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
35	Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 3$, $\rho = 0{,}4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 35, починають маршрут в різних випадкових вершинах).

3 ВИКОНАННЯ

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApp4{
    class backpack {
        public bool[] items;
        public backpack(int amount) {
            items = new bool[amount];
        }
        public backpack(backpack bp) {
            this.items = bp.items;
        }

        public void print() {
            Console.Write("{");
            for (int i = 0; i < this.items.Length; ++i) {
                if (this.items[i]) Console.Write("1");
                else Console.Write(".");
            }
            Console.WriteLine("}\nCost: " + Program.get_total_cost(this) + "\nWeight: " +
Program.get_total_weight(this));
        }

        public static bool operator ==(backpack bp1, backpack bp2) {
            for (int i = 0; i < bp1.items.Length; ++i) {
                if (bp1.items[i] != bp2.items[i]) return false;
            }
            return true;
        }

        public static bool operator !=(backpack bp1, backpack bp2) {
            return !(bp1 == bp2);
        }
    }

    class item {
        public int weight { get; set; }
        public int cost { get; set; }

        public item() {
            var rand = new Random();
            weight = rand.Next(2, 11);
            cost = rand.Next(1, 6);
        }
    }

    class Program {
        public const int CAPACITY = 150;
        public const int AMOUNT = 100;
        public const int NUMBER_OF_ITERATIONS = 1000;

        public static item[] item_list = create_items(AMOUNT);

        static void Main(string[] args) {
```

```
            var backpacks = init_backpacks(AMOUNT);
            print_items(item_list);
            backpacks = genetic_algo(backpacks);
            Console.WriteLine("\nBest item:");
            backpacks[get_best_index(backpacks)].print();
        }

        static backpack[] genetic_algo(backpack[] backpacks) {
            for (int i = 0; i < NUMBER_OF_ITERATIONS; ++i) {
                backpacks = genetic_iteration(backpacks);
            }
            return backpacks;
        }

        static backpack[] genetic_iteration(backpack[] backpacks) {
            (var best_bp, var random_bp) = choose_ancestors(backpacks);

            var child = crossing_genes(best_bp, random_bp);
            var mutated = mutate(child);

            if (get_total_weight(child) > CAPACITY && get_total_weight(mutated) > CAPACITY)
return backpacks;
            var improved = local_improvement(child, mutated);

            var best = choose_best(child, mutated, improved);
            int worst_index = get_worst_index(backpacks);
            backpacks[worst_index] = best;
            return backpacks;
        }

        static int get_worst_index(backpack[] backpacks) {
            int worst_cost = int.MaxValue;
            int worst_index = -1;
            for (int i = 0; i < AMOUNT; ++i) {
                if (get_total_cost(backpacks[i]) < worst_cost) {
                    worst_cost = get_total_cost(backpacks[i]);
                    worst_index = i;
                }
            }
            return worst_index;
        }

        static backpack choose_best(backpack child, backpack mutated, backpack improved) {
            if (get_total_weight(child) > CAPACITY) {
                if (get_total_weight(improved) > CAPACITY) return mutated;
                if (get_total_cost(mutated) > get_total_cost(improved)) return mutated;
                else return improved;
            }
            else if (get_total_weight(mutated) > CAPACITY) {
                if (get_total_weight(improved) > CAPACITY) {
                    return child;
                }
                if (get_total_cost(child) > get_total_cost(improved)) return child;
                else return improved;
            }
            else if (get_total_weight(improved) > CAPACITY) {
                if (get_total_cost(mutated) > get_total_cost(child)) return mutated;
                else return child;
            }
            else {
                int maxCost = Math.Max(get_total_cost(child),
Math.Max(get_total_cost(mutated), get_total_cost(improved)));
                if (maxCost == get_total_cost(improved)) return improved;
                else if (maxCost == get_total_cost(mutated)) return mutated;
                else return child;
```

```
        }
    }

    static backpack local_improvement(backpack child, backpack mutated) {
        if (get_total_weight(mutated) > CAPACITY) {
            if (get_total_weight(child) <= CAPACITY) {
                return improve(child);
            }
        }
        else{
            if (get_total_weight(child) <= CAPACITY) {
                if (get_total_cost(child) >= get_total_cost(mutated))
{
                    return improve(child);
                }
                else
{
                    return improve(mutated);
                }
            }
            else{
                return improve(mutated);
            }
        }
        return null;
    }

    static backpack improve(backpack bp) {
        var result = new backpack(bp);
        if (result.items[get_best_item_index()] == false) {
            result.items[get_best_item_index()] = true;
        }
        return result;
    }

    static int get_best_item_index() {
        double best_benefit = 0;
        int best_index = -1;
        for (int i = 0; i < AMOUNT; ++i) {
            if (item_list[i].cost / item_list[i].weight > best_benefit) {
                best_benefit = item_list[i].cost / item_list[i].weight;
                best_index = i;
            }
        }
        return best_index;
    }

    static backpack mutate(backpack bp) {
        var rand = new Random();
        if (rand.NextDouble() > 0.1) {
            int rand_index1 = rand.Next(0, bp.items.Length);
            int rand_index2 = rand.Next(0, bp.items.Length);
            while (rand_index1 == rand_index2) {
                rand_index2 = rand.Next(0, bp.items.Length);
            }

            bool tmp = bp.items[rand_index1];
            bp.items[rand_index1] = bp.items[rand_index2];
            bp.items[rand_index2] = tmp;
        }
        return bp;
    }

    static backpack crossing_genes(backpack bp1, backpack bp2) {
        var rand = new Random();
```

```
    var backpack_res = new backpack(AMOUNT);
    for (int i = 0; i < bp1.items.Length; ++i) {
        if (rand.NextDouble() < 0.5) {
            backpack_res.items[i] = bp1.items[i];
        }
        else{
            backpack_res.items[i] = bp2.items[i];
        }
    }
    return backpack_res;
}

static (backpack, backpack) choose_ancestors(backpack[] backpacks) {
    int best_index = get_best_index(backpacks);
    var best_bp = backpacks[best_index];

    var rand = new Random();
    int random_index = rand.Next(0, backpacks.Length);
    while (random_index == best_index) {
        random_index = rand.Next(0, backpacks.Length);
    }
    var random_bp = backpacks[random_index];

    return (best_bp, random_bp);
}

static int get_best_index(backpack[] bps) {
    int best_index = -1;
    int best_cost = 0;
    for (int i = 0; i < bps.Length; ++i) {
        if (get_total_cost(bps[i]) > best_cost) {
            best_cost = get_total_cost(bps[i]);
            best_index = i;
        }
    }
    return best_index;
}

static item[] create_items(int amount) {
    item[] items = new item[amount];
    for (int i = 0; i < items.Length; ++i) {
        items[i] = new item();
    }
    return items;
}

static backpack[] init_backpacks(int amount) {
    backpack[] backpacks = new backpack[amount];
    for (int i = 0; i < backpacks.Length; ++i) {
        backpacks[i] = new backpack(AMOUNT);
        backpacks[i].items[i] = true;
    }
    return backpacks;
}

static void print_items(item[] items) {
    Console.Write("Weights:\n");
    foreach (var item in items) {
        Console.Write(item.weight + " ");
    }
    Console.Write("\n\nCosts:\n");
    foreach (var item in items) {
        Console.Write(item.cost + " ");
    }
    Console.WriteLine();
```

```
        }

        public static int get_total_cost(backpack bp) {
            int totalCost = 0;
            for (int i = 0; i < bp.items.Length; ++i) {
                if (bp.items[i]) totalCost += item_list[i].cost;
            }
            return totalCost;
        }

        public static int get_total_weight(backpack bp) {
            int totalWeight = 0;
            for (int i = 0; i < bp.items.Length; ++i) {
                if (bp.items[i]) totalWeight += item_list[i].weight;
            }
            return totalWeight;
        }

    }
}
```

3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

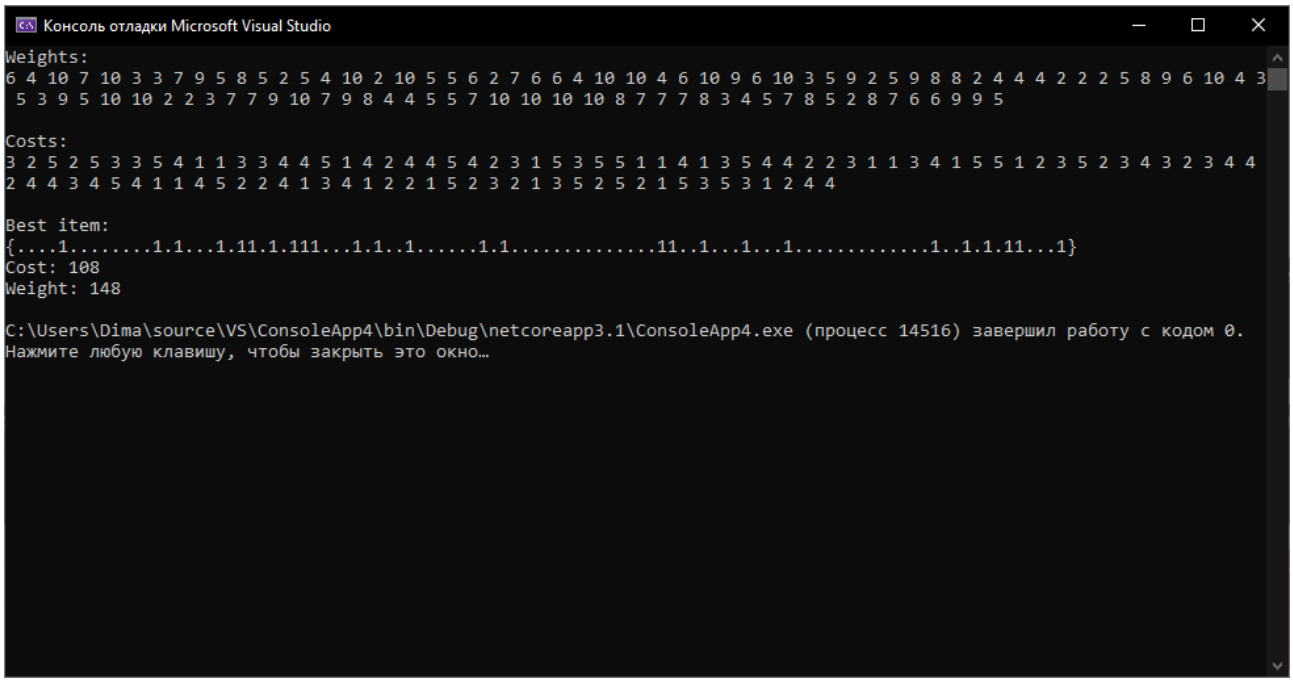

Рисунок 3.1 – приклад результату роботи

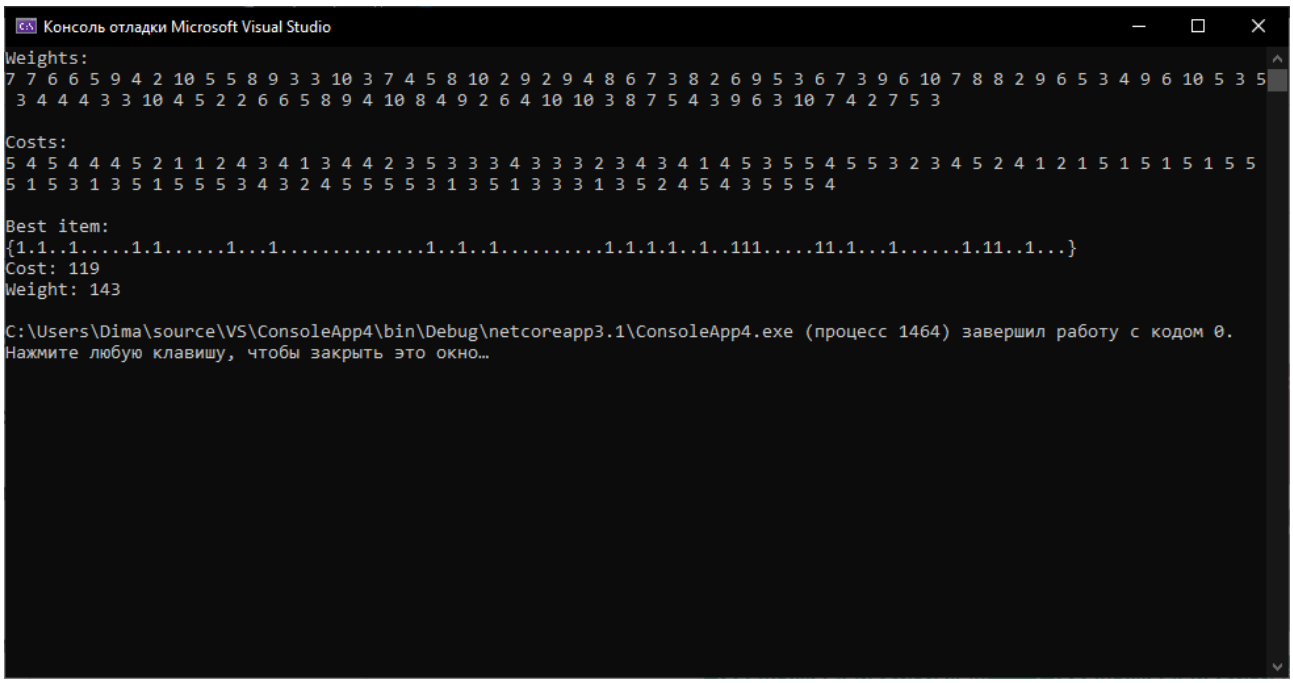

Рисунок 3.2 – приклад результату роботи

3.2 Тестування алгоритму

3.2.1 Значення цільової функції зі збільшенням кількості ітерацій

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

№	Кількість ітерацій	Значення цільової функції
1	0	5
2	20	21
3	40	21
4	60	23
5	80	28
6	100	35
7	120	35
8	140	35
9	160	35
10	180	35
11	200	39
12	220	41
13	240	42
14	260	48
15	280	49
16	300	52
17	320	59
18	340	59
19	360	59
20	380	60
21	400	65
22	420	67
23	440	70

Продовження таблиці 3.1

24	460	71
25	480	76
26	500	81
27	520	83
28	540	83
29	560	85
30	580	86
31	600	86
32	620	97
33	640	99
34	680	103
35	700	108
36	720	108
37	740	108
38	760	108
39	780	108
40	800	112
41	820	113
42	840	115
43	860	115
44	880	115
45	900	116
46	920	119
47	940	119
48	960	122
49	980	122
50	1000	122

3.2.2 Графіки залежності розв'язку від числа ітерацій

На рисунку 3.3 наведений графік, який показує якість отриманого розв'язку.

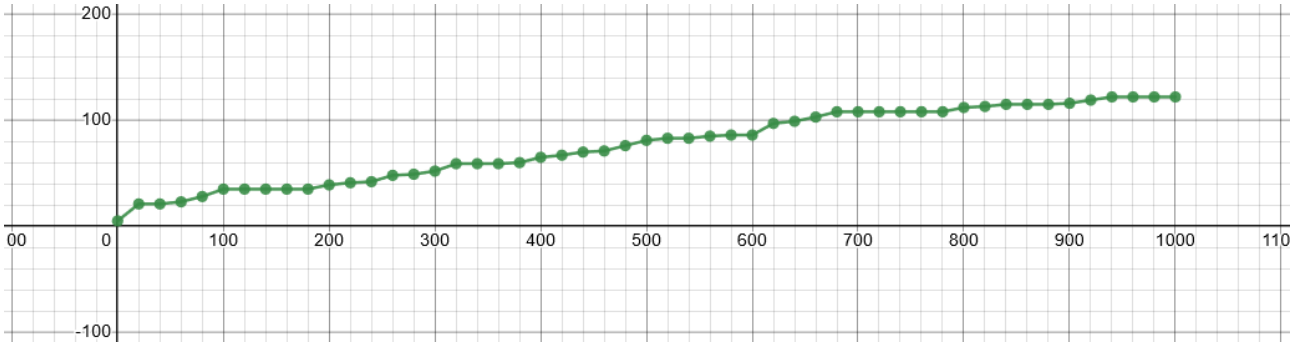

Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

ВИСНОВОК

В рамках даної лабораторної роботи я дослідив метаевристичні алгоритми, а також набув навичок їх реалізації на прикладі задачі про рюкзак та генетичний алгоритм.

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%.