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) *ІТ-01Бардін В. Д.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Камінська П. А.* _____
(прізвище, ім'я, по батькові)

Київ 2021

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 одно точковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.

3 ВИКОНАННЯ

3.1 Програмна реалізація алгоритму

```
internal static Population ChooseBestLocalUpgrade(
    this Population population,
    Backpack backpack,
    int iterationNumber,
    int dropSelected)
{
    if (!backpack.Items.Select(i => i.Selected).Contains(false))
        return population;
    var bestItem = backpack.Items.FirstOrDefault(i => i.Selected == false &&
i.Weight <= i.Value) ?? backpack.Items.First(i => i.Selected == false);

    var index = Array.IndexOf(backpack.Items, bestItem);
    backpack.Items[index].Selected = true;
    population.SelectedItems[index] = true;

    if (iterationNumber % dropSelected != 0) return population;
        foreach (var item in backpack.Items)
            item.Selected = false;

    return population;
}
```

3.1.1 Вихідний код

```
public class Item
{
    public int Value { get; }
    public int Weight { get; }
    public bool Selected { get; set; }
    public Item(int value, int weight)
    {
        Value = value;
        Weight = weight;
    }
}
```

```
public class Backpack
{
    public int Capacity { get; }
    public Item[] Items { get; }
    public Backpack(IEnumerable<Item> items, int capacity)
    {
        Capacity = capacity;
        Items = items
            .OrderBy(i => i.Weight)
            .ThenByDescending(i => i.Value)
            .ToArray();
    }
}


public class Population
{
    public bool[] SelectedItems { get; }
    public int TotalWeight { get; }
    public int Worth { get; }
    public double WorthPercentage { get; }
    public int Iteration { get; set; }
    public Population(Backpack backpack, bool[] selectedItems)
    {
        SelectedItems = selectedItems;
        Worth = backpack.Items.Where((_, i) => selectedItems[i]).Sum(t =>
t.Value);
        for (var i = 0; i < backpack.Items.Length; i++)
            if (selectedItems[i])
                TotalWeight += backpack.Items[i].Weight;
        if (TotalWeight > backpack.Capacity)
            Worth = 0;
        WorthPercentage = (double)TotalWeight / Worth * 100;
    }
    public static void AddAndDelete(List<Population> populations, Population
added)
    {
        populations.Add(added);
        var minWorth = populations.Select(p => p.Worth).Min();
        for (var i = 0; i < populations.Count; i++)
            if (populations[i].Worth == minWorth)
                populations.RemoveAt(i);
    }
}
```

```
internal static class GaOperators
{
    private static readonly Random Randomizer = new();
    internal static Population Selection(this IEnumerable<Population>
populations)
    {
        var bestPopulation = populations
            .OrderByDescending(p => p.Worth)
            .ThenBy(p => p.TotalWeight)
            .First();
        return bestPopulation;
    }
    internal static Population SinglePointCrossbreeding(this Population lhs,
Population rhs, Backpack backpack)
    {
        var halfElementsAmount = lhs.SelectedItems.Length / 2;
        var firstCross = new Population(
            backpack,
            lhs.SelectedItems
                .Skip(halfElementsAmount)
                .Concat(rhs.SelectedItems.Take(halfElementsAmount))
                .ToArray());
        var secondCross = new Population(
            backpack,
            lhs.SelectedItems
                .Take(halfElementsAmount)
                .Concat(rhs.SelectedItems.Skip(halfElementsAmount))
                .ToArray());
        return firstCross.Worth > secondCross.Worth ? firstCross : secondCross;
    }
    internal static Population ProbabilisticMutation(
        this Population population,
        Backpack backpack,
        int mutationChance)
    {
        if (Randomizer.Next(0, 100) > mutationChance) return population;
        var n = Randomizer.Next(0, population.SelectedItems.Length);
        population.SelectedItems[n] = population.SelectedItems[n] == false;
        population = new Population(backpack, population.SelectedItems);
        if (population.Worth == 0)
        {
            population.SelectedItems[n] = population.SelectedItems[n] == false;
        }
        return population;
    }
```

```
internal static Population ChooseBestLocalUpgrade(
        this Population population,
        Backpack backpack,
        int iterationNumber,
        int dropSelected)
    {
        if (!backpack.Items.Select(i => i.Selected).Contains(false))
            return population;
        var bestItem = backpack.Items.FirstOrDefault(i => i.Selected == false &&
i.Weight <= i.Value)
                        ?? backpack.Items.First(i => i.Selected == false);
        var index = Array.IndexOf(backpack.Items, bestItem);
        backpack.Items[index].Selected = true;
        population.SelectedItems[index] = true;
        if (iterationNumber % dropSelected != 0) return population;
        foreach (var item in backpack.Items)
            item.Selected = false;
        return population;
    }
}


internal static class Program
{
    private static readonly Random Randomizer = new();
    private const int KnapsackCapacity = 250;
    private const int ItemsAmount = 100;
    private const int MinItemsWorth = 2;
    private const int MaxItemsWorth = 20;
    private const int MinItemsWeight = 1;
    private const int MaxItemsWeight = 10;
    private const int GeneticIterations = 1000;
```

```
    public static void Main()
    {
        var knapsack = new Backpack(GenerateItems(), KnapsackCapacity);
        List<Population> bestPopulations = new();
        for (var iterationNumber = 1; iterationNumber <= GeneticIterations;
iterationNumber++)
        {
            var geneticProcessor = new GeneticAlgorithmProcessor(knapsack, 100,
5, 71)
            {
                CurrentPopulation =
                {
                    Iteration = iterationNumber
                }
            };
            bestPopulations.Add(geneticProcessor.CurrentPopulation);
        }
        foreach (var population in bestPopulations)
        {
            Console.WriteLine("Iteration: " + population.Iteration);
            Console.WriteLine("Worth: " + population.Worth);
            Console.WriteLine("Weigh: " + population.TotalWeight);
            Console.WriteLine("Worth Percentage: {0:####.####}%\n",
population.WorthPercentage);
        }
        var bestPopulation = bestPopulations.OrderByDescending(p =>
p.WorthPercentage).First();
        Console.WriteLine("Best Iteration: " + bestPopulation.Iteration +
                          "\nTotal weight: {0}" +
                          "\nTotal worth: {1}" +
                          "\nWorth Percentage: {2:####.####}%",
            bestPopulation.TotalWeight,
            bestPopulation.Worth,
            bestPopulation.WorthPercentage);
        Console.WriteLine("\nAverage Worth Percentage: {0:####.####}%",
            bestPopulations.Average(p => p.WorthPercentage));
    }
    private static IEnumerable<Item> GenerateItems() =>
        Enumerable.Range(0, ItemsAmount)
            .Select(_ => new Item(
                Randomizer.Next(MinItemsWorth, MaxItemsWorth),
                Randomizer.Next(MinItemsWeight, MaxItemsWeight)));
}
```

3.1.2 Приклади роботи

На рисунку 3.1 показані приклади роботи програми.

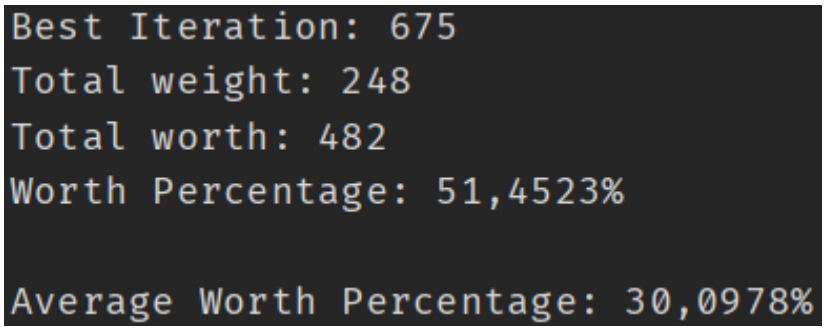

Рисунок 3.1 – Результат виконання алгоритму для 1000 ітерацій

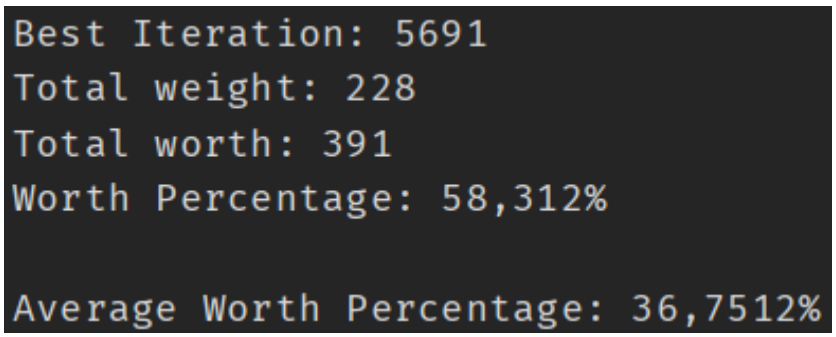

Рисунок 3.2 – Результат виконання алгоритму для 10000 ітерацій

3.2 Тестування алгоритму

Для тестування алгоритму задачу було запущено декілька разів з різною кількістю ітерацій. Всього при тестуванні розглядалося 5 варіантів: 1 ітерація, 10, 100, 1000, а також 10000 ітерацій. Далі буде наведено графіки залежності розв’язків від кількості ітерацій.

3.2.1 Графіки залежності розв'язку від числа ітерацій

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

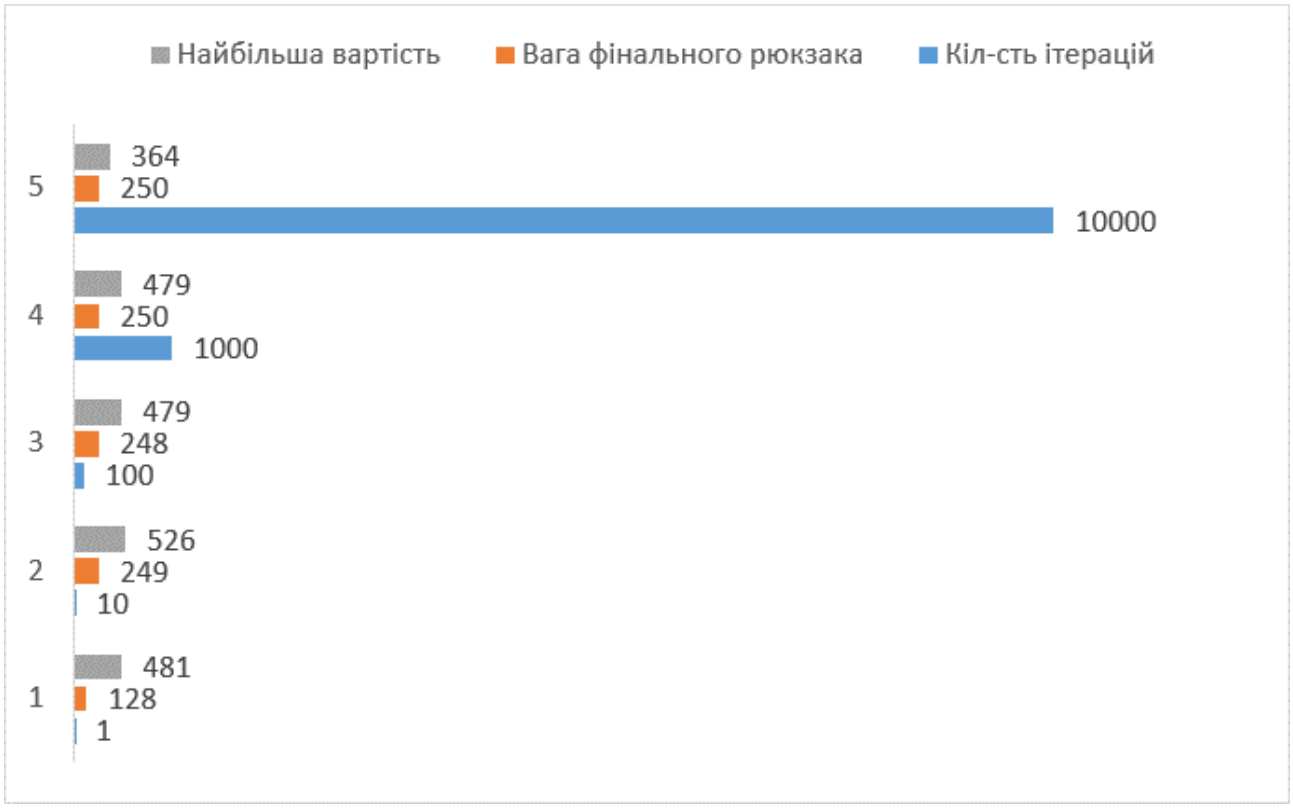

Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

ВИСНОВОК

В рамках даної лабораторної роботи я за допомогою метаевристичного алгоритму, а саме генетичного алгоритму розв’язав типову задачу про «Пакування рюкзака». В ході виконання я розібрався як працюють метаевристичні алгоритми, на прикладі генетичного алгоритму, а також написав його власну імплементацію. Також дослідив вплив кількість ітерацій на ефективність пошуку оптимального рішення за випадкових вхідних даних.