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.2"

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

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

Київ 2020

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

2 ЗАВДАННЯ

Згідно варіанту, формалізувати алгоритм вирішення задачі відповідно загальної методології.

Записати розроблений алгоритм у покроковому вигляді. З достатнім степенем деталізації.

Виконати його програмну реалізацію на будь-якій мові програмування.

Перелік задач наведено у таблиці 2.1.

Перелік алгоритмів і досліджуваних параметрів у таблиці 2.2.

Задача і алгоритм наведені в таблиці 2.3.

Змінюючи параметри алгоритму, визначити кращі вхідні параметри алгоритму. Для цього необхідно:

- обрати критерій зупинки алгоритму (кількість ітерацій або значення ЦФ);
- зафіксувати усі параметри крім одного і змінювати цей параметр, поки не буде досягнуто пікової ефективності;
- після цього параметр фіксується і змінюються інші параметри;
- далі повторюємо процедуру спочатку, з першого зафіксованого параметру;
- зупиняємось коли будуть знайдені оптимальні параметри для даної задачі або встановлена залежність одних параметрів від інших.

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

Таблиця 2.1 – Прикладні задачі

№	**Задача**
1	**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. Задача часто виникає при розподілі ресурсів, коли наявні фінансові обмеження, і вивчається в таких областях, як комбінаторика, інформатика, теорія складності, криптографія, прикладна математика.

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

№	**Алгоритми і досліджувані параметри**
1	**Генетичний алгоритм:** - оператор схрещування (мінімум 3); - мутація (мінімум 2); - оператор локального покращення (мінімум 2).

Таблиця 2.3 – Варіанти задач і алгоритмів

№	**Задачі і алгоритми**
1	Задача про рюкзак + Генетичний алгоритм

3 ВИКОНАННЯ

3.1 Покроковий алгоритм

Для знаходження оптимальних параметрів для алгоритму розв’язання задачі «Пакування рюкзака» розроблено простий алгоритм, який почергово шукає оптимальні параметри запуску(шанс мутації та кількість ітерацій). Потім зі знайденими параметрами алгоритм запускається 10 разів і серед отриманих значень обираються 20% найкращих показників і серед них отриманих значень береться середнє арифметичне з округленням до гори.

Алгоритм пошуку оптимальних параметрів

1. Знайти оптимальний шанс мутації;
2. Знайти оптимальну кількість ітерацій;
3. Зі знайденими параметрами запустити алгоритм 10 разів;
4. Обрати з отриманої вибірки 20% найкращих результати, та знайти середнє арифметичне округливши отримане значення до більшого.

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
internal static class Program
{
    private static readonly Random Randomizer = new();
    private const int KnapsackCapacity = 500;
    private const int ItemsAmount = 100;
    private const int MinItemsWorth = 2;
    private const int MaxItemsWorth = 30;
    private const int MinItemsWeight = 1;
    private const int MaxItemsWeight = 20;
    private const int MinIterations = 2;
    private const int IterationsIncreaseFactor = 2;
    private const int MaxIterations = 10000;
    public static void Main()
    {
        var items = GenerateItems().ToArray();
        var results = new List<(double, int, int)>();
        for (int k = 0; k < 10; k++)
        {
            var populations = new List<(Population, int)>();
            var bestMutation = 1;
            for (var mutationChance = 1; mutationChance <= 100;
mutationChance++)
```

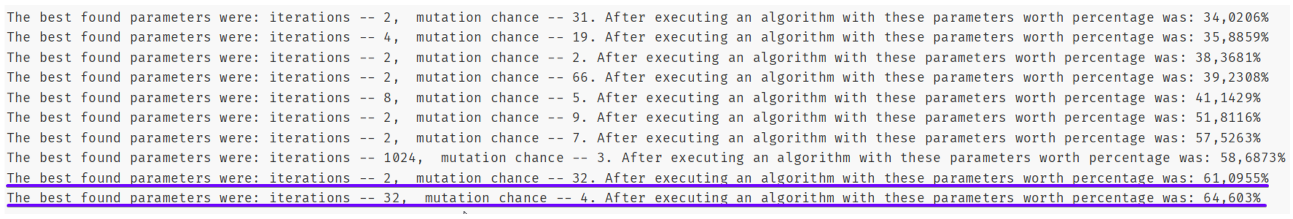
```
            {
                populations.Add((ExecuteWithIterations(MinIterations,
mutationChance, items), mutationChance))
            }
            var bestPopulationByMutation = populations.OrderBy(x =>
x.Item1.WorthPercentage).First();
            bestMutation = bestPopulationByMutation.Item2;
            for (var i = MinIterations; i < MaxIterations; i *=
IterationsIncreaseFactor)
            {
                populations.Add((ExecuteWithIterations(i, bestMutation, items),
i));
            }
            var bestIterations = populations.OrderBy(x =>
x.Item1.WorthPercentage).First().Item2;
            var bestPopulation = ExecuteWithIterations(bestIterations,
bestMutation, items);
            results.Add((bestPopulation.WorthPercentage, bestMutation,
bestIterations));
        }
        foreach (var result in results.OrderBy(x => x.Item1))
        {
            Console.WriteLine("The best found parameters were: iterations --
{0},  mutation chance -- {1}. " +
                                  "After executing an algorithm with these
parameters worth percentage was: {2:###
                result.Item3, result.Item2, result.Item3);
        }
    }
    private static Population ExecuteWithIterations(int iterations, int
mutation, IEnumerable<Item> items)
    {
        var knapsack = new Backpack(items, KnapsackCapacity);
        List<Population> bestPopulations = new();
        var geneticProcessor = new GeneticAlgorithmProcessor();
        for (var iterationNumber = 1; iterationNumber <= iterations;
iterationNumber++)
        {
            var population = geneticProcessor.Process(
                knapsack: knapsack,
                iterations: 100,
                mutationChance: mutation,
                dropSelected: 71);
            bestPopulations.Add(population);
        }
        return bestPopulations.OrderByDescending(p =>
p.WorthPercentage).First();
    }
    private static IEnumerable<Item> GenerateItems() =>
        Enumerable.Range(0, ItemsAmount)
            .Select(_ => new Item(
                Randomizer.Next(MinItemsWorth, MaxItemsWorth),
                Randomizer.Next(MinItemsWeight, MaxItemsWeight)));
}
```

```csharp
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 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 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 class GeneticAlgorithmProcessor
{
    private static readonly Random Randomizer = new();
    private Population _currentPopulation;
    public Population Process(
        Backpack knapsack,
        int iterations,
        int mutationChance,
        int dropSelected)
    {
        var bestPopulations = new List<Population>();
        var populations = GeneratePopulations(knapsack);
        for (var iterationNumber = 1; iterationNumber <= iterations;
iterationNumber++)
        {
            var newPopulation = populations
                .Selection()

.SinglePointCrossbreeding(populations[Randomizer.Next(populations.Count)],
knapsack)
                .ProbabilisticMutation(knapsack, mutationChance)
                .ChooseBestLocalUpgrade(knapsack, iterationNumber,
dropSelected);
            var upgradedPopulation = new Population(knapsack,
newPopulation.SelectedItems)
                { Iteration = iterationNumber };
            if (_currentPopulation is null ||upgradedPopulation.Worth >
_currentPopulation.Worth)
                _currentPopulation = upgradedPopulation;
            Population.AddAndDelete(populations, upgradedPopulation);
            if (upgradedPopulation.Worth != 0 &&
                !bestPopulations.Any(bp => bp.Worth == upgradedPopulation.Worth
&&
                                           bp.TotalWeight ==
upgradedPopulation.TotalWeight))
                // as an optimization we can store 2, 4 or 8 best populations
            {
                bestPopulations.Add(upgradedPopulation);
            }
        }
        return _currentPopulation;
    }
    private static List<Population> GeneratePopulations(Backpack knapsack)
    {
        var itemsAmount = knapsack.Items.Length;
        var populations = new List<Population>();
        for (var i = 0; i < itemsAmount; i++)
        {
            var selectedItems = new bool[itemsAmount];
            selectedItems[i] = true;
            populations.Add(new Population(knapsack, selectedItems));
        }
        return populations;
    }
}


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;
    }
    /// <summary>
    /// <b> Single point crossover operator. </b> <br/>
    /// The new vector X' gets the first 50 coordinates from vector X1,
    /// and the remaining 50 from vector X2.
    /// </summary>
    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;
    }
    /// <summary>
    /// <b> Uniform crossing operator. </b> <br/>
    /// A new solution in each coordinate is obtained with a probability of 0.5
the value of one of the parents.
    /// </summary>
    internal static Population UniformCrossbreeding(
        this Population first,
        Population second,
        Backpack knapsack)
    {
        var n = first.SelectedItems.Length;
        var array = new[] {first, second};
        var rnd = new Random();
        var firstItemsSelected = new bool[n];
        var secondItemsSelected = new bool[n];
        for (int i = 0; i < n; i++)
        {
            var select = rnd.Next(2);
            firstItemsSelected[i] = array[select].SelectedItems[i];
            select = select == 0 ? 1 : 0;
            secondItemsSelected[i] = array[select].SelectedItems[i];
        }
        var firstCross = new Population(knapsack, firstItemsSelected);
        var secondCross = new Population(knapsack, secondItemsSelected);
        return firstCross.Worth > secondCross.Worth ? firstCross : secondCross;
    }
    /// <summary>
    /// <b> Multipoint crossbreeding operator. </b> <br/>
    /// Divides populations by points into equal segments
    /// and in turn selects line segments. At the output, we have two
populations - choose the best one.
    /// </summary>
    internal static Population MultipointCrossbreeding(
        this Population first,
        Population second,
        Backpack knapsack,
        int pointNumbers)
    {
        var n = first.SelectedItems.Length / (pointNumbers + 1);
        var sum = 0;
        var partsOfFirstArray = new List<bool[]>();
        var partsOfSecondArray = new List<bool[]>();
        for (var i = 0; i < pointNumbers + 1; i++)
        {
            if (i % 2 == 0)
            {

partsOfFirstArray.Add(first.SelectedItems.Skip(sum).Take(n).ToArray());

partsOfSecondArray.Add(second.SelectedItems.Skip(sum).Take(n).ToArray());
            }
            else
            {

partsOfFirstArray.Add(second.SelectedItems.Skip(sum).Take(n).ToArray());

partsOfSecondArray.Add(first.SelectedItems.Skip(sum).Take(n).ToArray());
            }
            sum += n;
        }
        var firstFinalPop = new bool[first.SelectedItems.Length];
        var secondFinalPop = new bool[first.SelectedItems.Length];
        var num = 0;
        for (var i = 0; i < partsOfFirstArray.Count; i++)
        {
            for (var j = 0; j < partsOfFirstArray[i].Length; j++)
            {
                firstFinalPop[j + num] = partsOfFirstArray[i][j];
                secondFinalPop[j + num] = partsOfSecondArray[i][j];
            }
            num += partsOfFirstArray[0].Length;
        }
        var firstCross = new Population(knapsack, firstFinalPop);
        var secondCross = new Population(knapsack, secondFinalPop);
        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;
    }
    /// <summary>
    /// <b> Inverse mutation operator </b> <br/>
    /// Elements from both ends are swapped in pairs on a random segment
    /// </summary>
    internal static Population InversionMutation(
        this Population population,
        Backpack knapsack,
        int chance)
    {
        if (Randomizer.Next(0, 100) > chance) return population;
        var segment = new[]
        {
            Randomizer.Next(0, population.SelectedItems.Length),
            Randomizer.Next(0, population.SelectedItems.Length)
        };
        var start = segment.Min();
        var end = segment.Max();
        for (int i = start, j = end; i < j; i++, j--)
        {
            (population.SelectedItems[i], population.SelectedItems[j])
                = (population.SelectedItems[j], population.SelectedItems[i]);
        }
        population = new Population(knapsack, population.SelectedItems);
        return population;
    }
    internal static Population ChooseBestLocalUpgrade(
        this Population population,
        Backpack backpack,
        int iterationNumber,
        int dropSelected)
    {
        // Find and replace an item with a minimum weight, max value and value
>= weight that still wasn't added
        // Each dropSelected - 1 iterations this items will be changed
        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 Population ChooseBestLocalUpgrade2(
        this Population population,
        Backpack backpack,
        int iterationNumber,
        int dropSelected)
    {
        if (!backpack.Items.Select(i => i.Selected).Contains(false))
            return population;
        // Select an item that wasn't selected yet and has the minimum weight
and maximum value
        var bestItem = backpack.Items.Aggregate((lhs, rhs)
            => !lhs.Selected &&
                !rhs.Selected &&
                lhs.Weight < rhs.Weight &&
                lhs.Value > rhs.Value
                 ? lhs : rhs);
        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;
    }
}
```

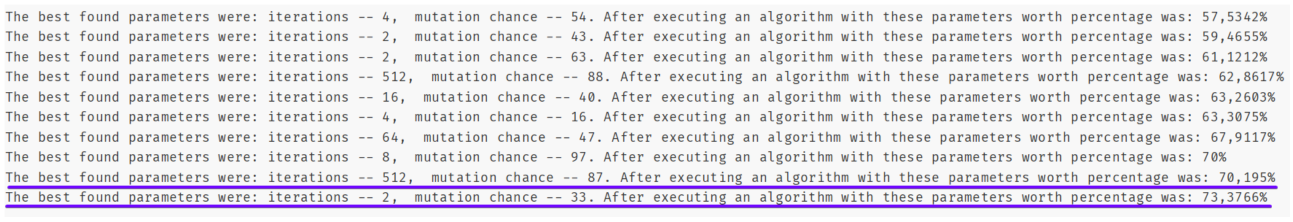
3.2.2 Приклади роботи

На рисунках 3.1 — 3.12 показані приклади роботи програми для різних комбінацій генетичних операторів.



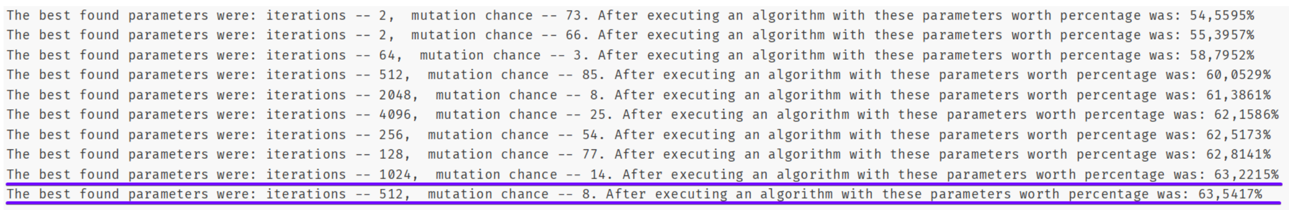
```
The best found parameters were: iterations -- 2,  mutation chance -- 31. After executing an algorithm with these parameters worth percentage was: 34,0206%
The best found parameters were: iterations -- 4,  mutation chance -- 19. After executing an algorithm with these parameters worth percentage was: 35,8859%
The best found parameters were: iterations -- 2,  mutation chance -- 2. After executing an algorithm with these parameters worth percentage was: 38,3681%
The best found parameters were: iterations -- 2,  mutation chance -- 66. After executing an algorithm with these parameters worth percentage was: 39,2308%
The best found parameters were: iterations -- 8,  mutation chance -- 5. After executing an algorithm with these parameters worth percentage was: 41,1429%
The best found parameters were: iterations -- 2,  mutation chance -- 9. After executing an algorithm with these parameters worth percentage was: 51,8116%
The best found parameters were: iterations -- 2,  mutation chance -- 7. After executing an algorithm with these parameters worth percentage was: 57,5263%
The best found parameters were: iterations -- 1024,  mutation chance -- 3. After executing an algorithm with these parameters worth percentage was: 58,6873%
The best found parameters were: iterations -- 2,  mutation chance -- 32. After executing an algorithm with these parameters worth percentage was: 61,0955%
The best found parameters were: iterations -- 32,  mutation chance -- 4. After executing an algorithm with these parameters worth percentage was: 64,603%
```

Рисунок 3.1 – Результати пошуку оптимальних параметрів для Single Point Crossbreeding, Probabilistic Mutation та Choose Best Local Upgrade



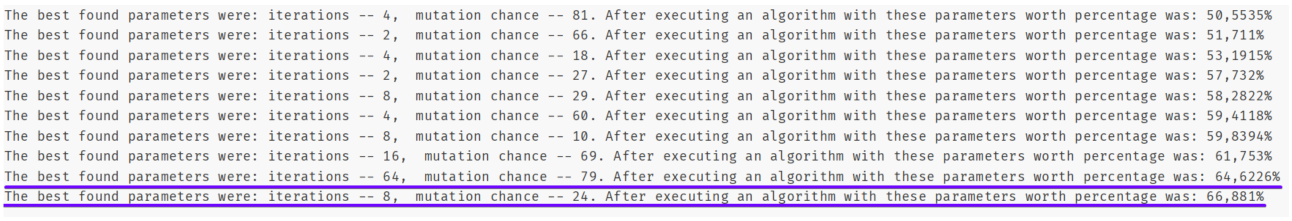
```
The best found parameters were: iterations -- 4,  mutation chance -- 54. After executing an algorithm with these parameters worth percentage was: 57,5342%
The best found parameters were: iterations -- 2,  mutation chance -- 43. After executing an algorithm with these parameters worth percentage was: 59,4655%
The best found parameters were: iterations -- 2,  mutation chance -- 63. After executing an algorithm with these parameters worth percentage was: 61,1212%
The best found parameters were: iterations -- 512,  mutation chance -- 88. After executing an algorithm with these parameters worth percentage was: 62,8617%
The best found parameters were: iterations -- 16,  mutation chance -- 40. After executing an algorithm with these parameters worth percentage was: 63,2603%
The best found parameters were: iterations -- 4,  mutation chance -- 16. After executing an algorithm with these parameters worth percentage was: 63,3075%
The best found parameters were: iterations -- 64,  mutation chance -- 47. After executing an algorithm with these parameters worth percentage was: 67,9117%
The best found parameters were: iterations -- 8,  mutation chance -- 97. After executing an algorithm with these parameters worth percentage was: 70%
The best found parameters were: iterations -- 512,  mutation chance -- 87. After executing an algorithm with these parameters worth percentage was: 70,195%
The best found parameters were: iterations -- 2,  mutation chance -- 33. After executing an algorithm with these parameters worth percentage was: 73,3766%
```

Рисунок 3.2 – Результати пошуку оптимальних параметрів для Single Point Crossbreeding, Probabilistic Mutation та Choose Best Local Upgrade 2



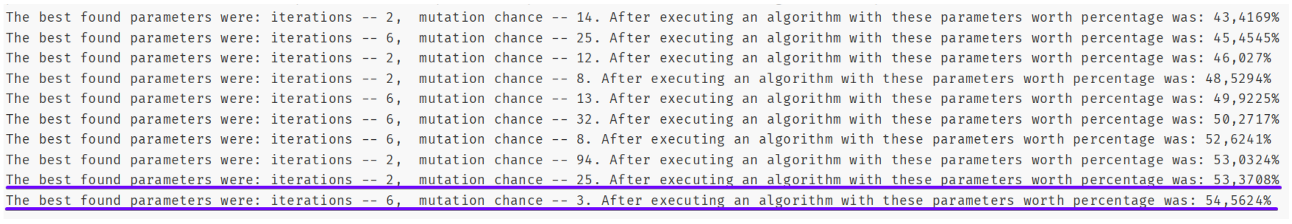
```
The best found parameters were: iterations -- 2,  mutation chance -- 73. After executing an algorithm with these parameters worth percentage was: 54,5595%
The best found parameters were: iterations -- 2,  mutation chance -- 66. After executing an algorithm with these parameters worth percentage was: 55,3957%
The best found parameters were: iterations -- 64,  mutation chance -- 3. After executing an algorithm with these parameters worth percentage was: 58,7952%
The best found parameters were: iterations -- 512,  mutation chance -- 85. After executing an algorithm with these parameters worth percentage was: 60,0529%
The best found parameters were: iterations -- 2048,  mutation chance -- 8. After executing an algorithm with these parameters worth percentage was: 61,3861%
The best found parameters were: iterations -- 4096,  mutation chance -- 25. After executing an algorithm with these parameters worth percentage was: 62,1586%
The best found parameters were: iterations -- 256,  mutation chance -- 54. After executing an algorithm with these parameters worth percentage was: 62,5173%
The best found parameters were: iterations -- 128,  mutation chance -- 77. After executing an algorithm with these parameters worth percentage was: 62,8141%
The best found parameters were: iterations -- 1024,  mutation chance -- 14. After executing an algorithm with these parameters worth percentage was: 63,2215%
The best found parameters were: iterations -- 512,  mutation chance -- 8. After executing an algorithm with these parameters worth percentage was: 63,5417%
```

Рисунок 3.3 – Результати пошуку оптимальних параметрів для Single Point Crossbreeding, Inversion Mutation та Choose Best Local Upgrade



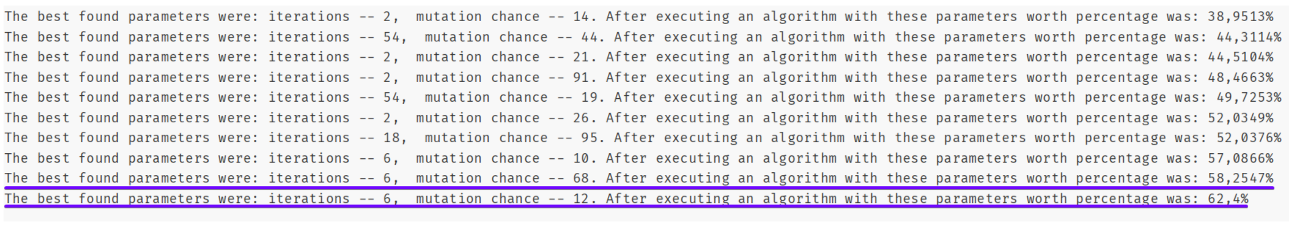
```
The best found parameters were: iterations -- 4,  mutation chance -- 81. After executing an algorithm with these parameters worth percentage was: 50,5535%
The best found parameters were: iterations -- 2,  mutation chance -- 66. After executing an algorithm with these parameters worth percentage was: 51,711%
The best found parameters were: iterations -- 4,  mutation chance -- 18. After executing an algorithm with these parameters worth percentage was: 53,1915%
The best found parameters were: iterations -- 2,  mutation chance -- 27. After executing an algorithm with these parameters worth percentage was: 57,732%
The best found parameters were: iterations -- 8,  mutation chance -- 29. After executing an algorithm with these parameters worth percentage was: 58,2822%
The best found parameters were: iterations -- 4,  mutation chance -- 60. After executing an algorithm with these parameters worth percentage was: 59,4118%
The best found parameters were: iterations -- 8,  mutation chance -- 10. After executing an algorithm with these parameters worth percentage was: 59,8394%
The best found parameters were: iterations -- 16,  mutation chance -- 69. After executing an algorithm with these parameters worth percentage was: 61,753%
The best found parameters were: iterations -- 64,  mutation chance -- 79. After executing an algorithm with these parameters worth percentage was: 64,6226%
The best found parameters were: iterations -- 8,  mutation chance -- 24. After executing an algorithm with these parameters worth percentage was: 66,881%
```

Рисунок 3.4 – Результати пошуку оптимальних параметрів для Single Point Crossbreeding, Inversion Mutation та Choose Best Local Upgrade 2



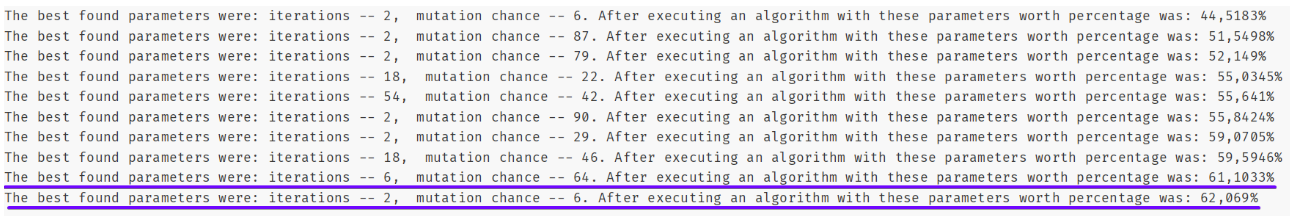
```
The best found parameters were: iterations -- 2,  mutation chance -- 14. After executing an algorithm with these parameters worth percentage was: 43,4169%
The best found parameters were: iterations -- 6,  mutation chance -- 25. After executing an algorithm with these parameters worth percentage was: 45,4545%
The best found parameters were: iterations -- 2,  mutation chance -- 12. After executing an algorithm with these parameters worth percentage was: 46,027%
The best found parameters were: iterations -- 2,  mutation chance -- 8. After executing an algorithm with these parameters worth percentage was: 48,5294%
The best found parameters were: iterations -- 6,  mutation chance -- 13. After executing an algorithm with these parameters worth percentage was: 49,9225%
The best found parameters were: iterations -- 6,  mutation chance -- 32. After executing an algorithm with these parameters worth percentage was: 50,2717%
The best found parameters were: iterations -- 6,  mutation chance -- 8. After executing an algorithm with these parameters worth percentage was: 52,6241%
The best found parameters were: iterations -- 2,  mutation chance -- 94. After executing an algorithm with these parameters worth percentage was: 53,0324%
The best found parameters were: iterations -- 2,  mutation chance -- 25. After executing an algorithm with these parameters worth percentage was: 53,3708%
The best found parameters were: iterations -- 6,  mutation chance -- 3. After executing an algorithm with these parameters worth percentage was: 54,5624%
```

Рисунок 3.5 – Результати пошуку оптимальних параметрів для Uniform Crossbreeding, Probabilistic Mutation та Choose Best Local Upgrade



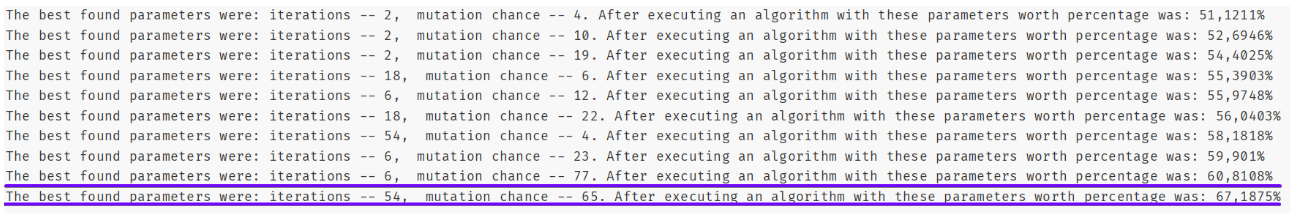
```
The best found parameters were: iterations -- 2,  mutation chance -- 14. After executing an algorithm with these parameters worth percentage was: 38,9513%
The best found parameters were: iterations -- 54,  mutation chance -- 44. After executing an algorithm with these parameters worth percentage was: 44,3114%
The best found parameters were: iterations -- 2,  mutation chance -- 21. After executing an algorithm with these parameters worth percentage was: 44,5104%
The best found parameters were: iterations -- 2,  mutation chance -- 91. After executing an algorithm with these parameters worth percentage was: 48,4663%
The best found parameters were: iterations -- 54,  mutation chance -- 19. After executing an algorithm with these parameters worth percentage was: 49,7253%
The best found parameters were: iterations -- 2,  mutation chance -- 26. After executing an algorithm with these parameters worth percentage was: 52,0349%
The best found parameters were: iterations -- 18,  mutation chance -- 95. After executing an algorithm with these parameters worth percentage was: 52,0376%
The best found parameters were: iterations -- 6,  mutation chance -- 10. After executing an algorithm with these parameters worth percentage was: 57,0866%
The best found parameters were: iterations -- 6,  mutation chance -- 68. After executing an algorithm with these parameters worth percentage was: 58,2547%
The best found parameters were: iterations -- 6,  mutation chance -- 12. After executing an algorithm with these parameters worth percentage was: 62,4%
```

Рисунок 3.6 – Результати пошуку оптимальних параметрів для Uniform Crossbreeding, Probabilistic Mutation та Choose Best Local Upgrade 2



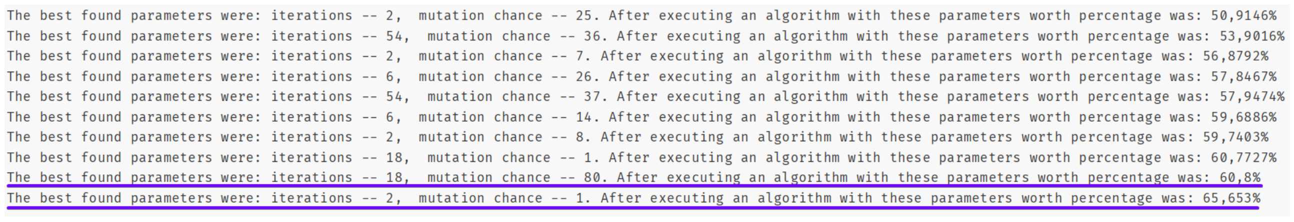
```
The best found parameters were: iterations -- 2,  mutation chance -- 6. After executing an algorithm with these parameters worth percentage was: 44,5183%
The best found parameters were: iterations -- 2,  mutation chance -- 87. After executing an algorithm with these parameters worth percentage was: 51,5498%
The best found parameters were: iterations -- 2,  mutation chance -- 79. After executing an algorithm with these parameters worth percentage was: 52,149%
The best found parameters were: iterations -- 18,  mutation chance -- 22. After executing an algorithm with these parameters worth percentage was: 55,0345%
The best found parameters were: iterations -- 54,  mutation chance -- 42. After executing an algorithm with these parameters worth percentage was: 55,641%
The best found parameters were: iterations -- 2,  mutation chance -- 90. After executing an algorithm with these parameters worth percentage was: 55,8424%
The best found parameters were: iterations -- 2,  mutation chance -- 29. After executing an algorithm with these parameters worth percentage was: 59,0705%
The best found parameters were: iterations -- 18,  mutation chance -- 46. After executing an algorithm with these parameters worth percentage was: 59,5946%
The best found parameters were: iterations -- 6,  mutation chance -- 64. After executing an algorithm with these parameters worth percentage was: 61,1033%
The best found parameters were: iterations -- 2,  mutation chance -- 6. After executing an algorithm with these parameters worth percentage was: 62,069%
```

Рисунок 3.7 – Результати пошуку оптимальних параметрів для Uniform Crossbreeding, Inversion Mutation та Choose Best Local Upgrade



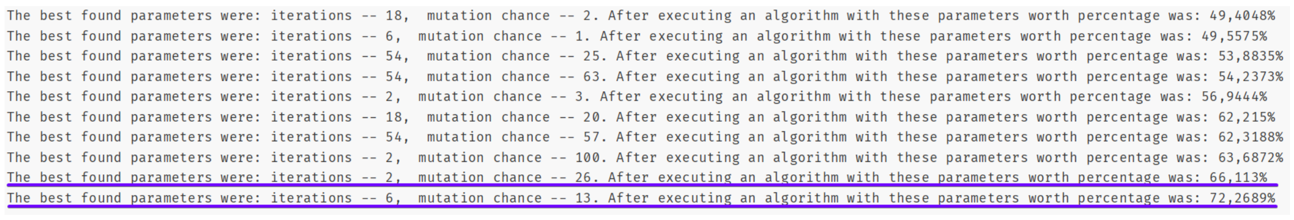
```
The best found parameters were: iterations -- 2,  mutation chance -- 4. After executing an algorithm with these parameters worth percentage was: 51,1211%
The best found parameters were: iterations -- 2,  mutation chance -- 10. After executing an algorithm with these parameters worth percentage was: 52,6946%
The best found parameters were: iterations -- 2,  mutation chance -- 19. After executing an algorithm with these parameters worth percentage was: 54,4025%
The best found parameters were: iterations -- 18,  mutation chance -- 6. After executing an algorithm with these parameters worth percentage was: 55,3903%
The best found parameters were: iterations -- 6,  mutation chance -- 12. After executing an algorithm with these parameters worth percentage was: 55,9748%
The best found parameters were: iterations -- 18,  mutation chance -- 22. After executing an algorithm with these parameters worth percentage was: 56,0403%
The best found parameters were: iterations -- 54,  mutation chance -- 4. After executing an algorithm with these parameters worth percentage was: 58,1818%
The best found parameters were: iterations -- 6,  mutation chance -- 23. After executing an algorithm with these parameters worth percentage was: 59,901%
The best found parameters were: iterations -- 6,  mutation chance -- 77. After executing an algorithm with these parameters worth percentage was: 60,8108%
The best found parameters were: iterations -- 54,  mutation chance -- 65. After executing an algorithm with these parameters worth percentage was: 67,1875%
```

Рисунок 3.8 – Результати пошуку оптимальних параметрів для Uniform Crossbreeding, Inversion Mutation та Choose Best Local Upgrade 2



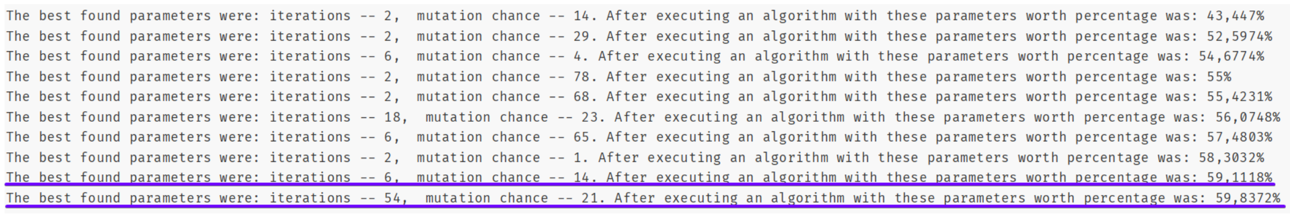
```
The best found parameters were: iterations -- 2,  mutation chance -- 25. After executing an algorithm with these parameters worth percentage was: 50,9146%
The best found parameters were: iterations -- 54,  mutation chance -- 36. After executing an algorithm with these parameters worth percentage was: 53,9016%
The best found parameters were: iterations -- 2,  mutation chance -- 7. After executing an algorithm with these parameters worth percentage was: 56,8792%
The best found parameters were: iterations -- 6,  mutation chance -- 26. After executing an algorithm with these parameters worth percentage was: 57,8467%
The best found parameters were: iterations -- 54,  mutation chance -- 37. After executing an algorithm with these parameters worth percentage was: 57,9474%
The best found parameters were: iterations -- 6,  mutation chance -- 14. After executing an algorithm with these parameters worth percentage was: 59,6886%
The best found parameters were: iterations -- 2,  mutation chance -- 8. After executing an algorithm with these parameters worth percentage was: 59,7403%
The best found parameters were: iterations -- 18,  mutation chance -- 1. After executing an algorithm with these parameters worth percentage was: 60,7727%
The best found parameters were: iterations -- 18,  mutation chance -- 80. After executing an algorithm with these parameters worth percentage was: 60,8%
The best found parameters were: iterations -- 2,  mutation chance -- 1. After executing an algorithm with these parameters worth percentage was: 65,653%
```

Рисунок 3.9 – Результати пошуку оптимальних параметрів для Multipoint Crossbreeding, Probabilistic Mutation та Choose Best Local Upgrade



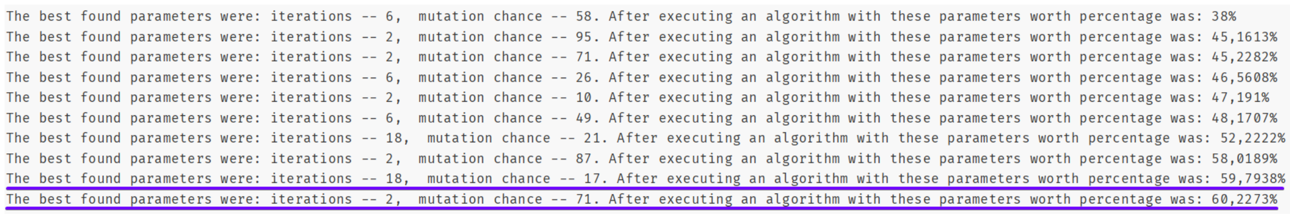
```
The best found parameters were: iterations -- 18,  mutation chance -- 2. After executing an algorithm with these parameters worth percentage was: 49,4048%
The best found parameters were: iterations -- 6,  mutation chance -- 1. After executing an algorithm with these parameters worth percentage was: 49,5575%
The best found parameters were: iterations -- 54,  mutation chance -- 25. After executing an algorithm with these parameters worth percentage was: 53,8835%
The best found parameters were: iterations -- 54,  mutation chance -- 63. After executing an algorithm with these parameters worth percentage was: 54,2373%
The best found parameters were: iterations -- 2,  mutation chance -- 3. After executing an algorithm with these parameters worth percentage was: 56,9444%
The best found parameters were: iterations -- 18,  mutation chance -- 20. After executing an algorithm with these parameters worth percentage was: 62,215%
The best found parameters were: iterations -- 54,  mutation chance -- 57. After executing an algorithm with these parameters worth percentage was: 62,3188%
The best found parameters were: iterations -- 2,  mutation chance -- 100. After executing an algorithm with these parameters worth percentage was: 63,6872%
The best found parameters were: iterations -- 2,  mutation chance -- 26. After executing an algorithm with these parameters worth percentage was: 66,113%
The best found parameters were: iterations -- 6,  mutation chance -- 13. After executing an algorithm with these parameters worth percentage was: 72,2689%
```

Рисунок 3.10 – Результати пошуку оптимальних параметрів для Multipoint Crossbreeding, Probabilistic Mutation та Choose Best Local Upgrade 2



```
The best found parameters were: iterations -- 2,  mutation chance -- 14. After executing an algorithm with these parameters worth percentage was: 43,447%
The best found parameters were: iterations -- 2,  mutation chance -- 29. After executing an algorithm with these parameters worth percentage was: 52,5974%
The best found parameters were: iterations -- 6,  mutation chance -- 4. After executing an algorithm with these parameters worth percentage was: 54,6774%
The best found parameters were: iterations -- 2,  mutation chance -- 78. After executing an algorithm with these parameters worth percentage was: 55%
The best found parameters were: iterations -- 2,  mutation chance -- 68. After executing an algorithm with these parameters worth percentage was: 55,4231%
The best found parameters were: iterations -- 18,  mutation chance -- 23. After executing an algorithm with these parameters worth percentage was: 56,0748%
The best found parameters were: iterations -- 6,  mutation chance -- 65. After executing an algorithm with these parameters worth percentage was: 57,4803%
The best found parameters were: iterations -- 2,  mutation chance -- 1. After executing an algorithm with these parameters worth percentage was: 58,3032%
The best found parameters were: iterations -- 6,  mutation chance -- 14. After executing an algorithm with these parameters worth percentage was: 59,1118%
The best found parameters were: iterations -- 54,  mutation chance -- 21. After executing an algorithm with these parameters worth percentage was: 59,8372%
```

Рисунок 3.11 – Результати пошуку оптимальних параметрів для Multipoint Crossbreeding, Inversion Mutation та Choose Best Local Upgrade



```
The best found parameters were: iterations -- 6,  mutation chance -- 58. After executing an algorithm with these parameters worth percentage was: 38%
The best found parameters were: iterations -- 2,  mutation chance -- 95. After executing an algorithm with these parameters worth percentage was: 45,1613%
The best found parameters were: iterations -- 2,  mutation chance -- 71. After executing an algorithm with these parameters worth percentage was: 45,2282%
The best found parameters were: iterations -- 6,  mutation chance -- 26. After executing an algorithm with these parameters worth percentage was: 46,5608%
The best found parameters were: iterations -- 2,  mutation chance -- 10. After executing an algorithm with these parameters worth percentage was: 47,191%
The best found parameters were: iterations -- 6,  mutation chance -- 49. After executing an algorithm with these parameters worth percentage was: 48,1707%
The best found parameters were: iterations -- 18,  mutation chance -- 21. After executing an algorithm with these parameters worth percentage was: 52,2222%
The best found parameters were: iterations -- 2,  mutation chance -- 87. After executing an algorithm with these parameters worth percentage was: 58,0189%
The best found parameters were: iterations -- 18,  mutation chance -- 17. After executing an algorithm with these parameters worth percentage was: 59,7938%
The best found parameters were: iterations -- 2,  mutation chance -- 71. After executing an algorithm with these parameters worth percentage was: 60,2273%
```

Рисунок 3.12 – Результати пошуку оптимальних параметрів для Multipoint Crossbreeding, Inversion Mutation та Choose Best Local Upgrade 2

3.3 Тестування алгоритму

№	Схрещування	Мутація	Шанс мутації	Оператор локального покрашення	Кіл-сть ітерацій
1	Single Point Crossbreeding	Probabilistic Mutation	18	Choose Best Local Upgrade	17
2	Single Point Crossbreeding	Probabilistic Mutation	60	Choose Best Local Upgrade 2	257
3	Single Point Crossbreeding	Inversion Mutation	11	Choose Best Local Upgrade	768
4	Single Point Crossbreeding	Inversion Mutation	52	Choose Best Local Upgrade 2	36
5	Uniform Crossbreeding	Probabilistic Mutation	4	Choose Best Local Upgrade	14
6	Uniform Crossbreeding	Probabilistic Mutation	6	Choose Best Local Upgrade 2	40

7	Uniform Crossbreeding	Inversion Mutation	4	Choose Best Local Upgrade	35
8	Uniform Crossbreeding	Inversion Mutation	30	Choose Best Local Upgrade 2	71
9	Multipoint Crossbreeding	Probabilistic Mutation	10	Choose Best Local Upgrade	41
10	Multipoint Crossbreeding	Probabilistic Mutation	4	Choose Best Local Upgrade 2	20
11	Multipoint Crossbreeding	Inversion Mutation	30	Choose Best Local Upgrade	18
12	Multipoint Crossbreeding	Inversion Mutation	10	Choose Best Local Upgrade 2	44

Табл. 3.1 — Оптимальні початкові дані в залежності від комбінацій генетичних операторів

ВИСНОВОК

В рамках даної лабораторної роботи я створив 2 додаткові методи схрещення для популяцій: Uniform Crossbreeding та Multipoint Crossbreeding, а також додатковий оператор мутації: Inversion Mutation та оператор локального покрашення: Choose Best Local Upgrade 2. Також дослідив вплив різних параметрів на результати виконання алгоритму. Для цього я розробив алгоритм для тестування мого розв’язку задачі «Пакування рюкзака» в залежності від різних вхідних даних.

За даними з табл. 3.1 видно, що в середньому Multipoint Crossbreeding потребує менше ітерацій для досягнення оптимального результату. І навіть при відносно не великих шансах мутації ми досить швидко досягаємо результату. Також з цієї ж таблиці можна помітити, що 2-га версія оператора локального покращення дає кращий результат, чим його 1-ша версія. Оператор мутації у цих експериментах впливу майже не мав, проте Inversion Mutation використовував більше пам’яті так, як він проводить більше операцій над масивами.

Отже, оптимальна комбінацію — Multipoint Crossbreeding, Probabilistic Mutation та 2-га версія оператора локального покрашення.