Міністерство освіти і науки України
НаціональнийтехнічнийуніверситетУкраїни«Київськийполітехнічнийінст
итутіменіІгоряСікорського"
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 Макарчук Лідія Олександрівна* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов Олексій Олександрович* _____
(прізвище, ім'я, по батькові)

Київ2022

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 одноточковийпо 50 генів, мутація зймовірністю5% 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 ($\alpha = 2$, $\beta = 4$, $\rho = 0{,}4$, Lminзнайти 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

Варіант 16

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.

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

//Модуль Generation

```
using System.Reflection.Metadata;
using System.Security.Cryptography;

namespace laba4
{
   public class Generation
   {
      public enum SelectionMethod
      {
         BestAndRandom,
         Tournament,
         Proportionate
      }
      public enum LocalImprovementMethod
      {
         Superset,
         Subtitute,
         Hybrid
      }
      private SortedList<int, Creature> _currPopulation;
      private Creature bestCreature;
      private int n;
      private int MaxWeight;
      private int iterationNumber;
      public List<int> F_ValuesAfter20Iterations;

      private int genNum1;
      private int genNum2;
      private int genNum3;

      private double mutationPossibility;
      SelectionMethod selectMethod;
      LocalImprovementMethod imprMethod;
      int setNumber;//var for Tournament selection
      int sumFitness;//var for Proportionate selection
```

```
int itemNumberLimit;//var for Hybrid improvement

public Generation(int n, int P, int iterations, int selectMethod, int imprMethod=2)
{
   // initialization
   this.selectMethod = (SelectionMethod)selectMethod;
   this.imprMethod = (LocalImprovementMethod)imprMethod;
   this.n = n;
   this.MaxWeight = P;
   this.iterationNumber = iterations;
   F_ValuesAfter20Iterations = new List<int>();

   genNum1 = (int)(0.3 * n); //30%
   genNum2 = (int)(0.4 * n); //40%
   genNum3 = n - genNum1 - genNum2;    //30%
   mutationPossibility = 0.1;  //10%

   _currPopulation = new SortedList<int, Creature>(new DuplicateKeyComparer<int>());
   CreateInitialPopulation(n);
   bestCreature = _currPopulation.Last().Value;

   if (this.selectMethod == SelectionMethod.Tournament)
   {
      switch (n)
      {
         case < 5: setNumber = 2; break;
         case < 20: setNumber = (int)(n * 0.4); break;
         default: setNumber = (int)(n * 0.2); break;
      }
   }
   else if(this.selectMethod == SelectionMethod.Proportionate)
      sumFitness = CalcFinessSum();
   itemNumberLimit = (int)(n / 3);
}
private void CreateInitialPopulation(int n)
{
   bool[] chromosome;
   for (int i=0; i<n; i++)
   {
      chromosome = new bool[n];
      chromosome[i] = true;
      Creature creature = new Creature(chromosome);
      _currPopulation.Add(creature.F, creature);
   }
}
public void GeneticAlgorithm()
{
   int count20Iterations = 0;
   for (int i=0; i< this.iterationNumber; i++)
   {
      if (count20Iterations == 20)
      {
         F_ValuesAfter20Iterations.Add(bestCreature.F);
         count20Iterations = 0;
      }
      count20Iterations++;
      Creature parent1, parent2;
      Creature child1, child2;
      Selection(out parent1, out parent2);
      Crossover(parent1, parent2, out child1, out child2);

      // for the first child
      if (child1.P <= MaxWeight)//check if alive
      {
         child1 = Mutation(child1);
```

```
            child1 = LocalImprovement(child1);
            AddChildToPopulation(child1);
        }
        // for the second child
        if (child2.P <= MaxWeight)//check if alive
        {
            child2 = Mutation(child2);
            child2 = LocalImprovement(child2);
            AddChildToPopulation(child2);
        }
    }
}
private void Selection(out Creature parent1, out Creature parent2)
{
    switch (selectMethod)
    {
        case SelectionMethod.Tournament:
            S_Tournament(out parent1, out parent2);
                break;
        case SelectionMethod.BestAndRandom:
                S_BestAndRandom(out parent1, out parent2);
                break;
        case SelectionMethod.Proportionate:
                S_Proportionate(out parent1, out parent2);
                break;
        default:
            {
                parent1 = _currPopulation.ElementAt(0).Value;
                parent2 = _currPopulation.ElementAt(1).Value;
                break;
            }
    }
}
private void S_BestAndRandom(out Creature parent1, out Creature parent2)
{
    Random rnd = new Random();
    parent1 = bestCreature;
    do
    {
        parent2 = _currPopulation.ElementAt(rnd.Next(0, _currPopulation.Count)).Value;
    } while (parent1 == parent2);
}
private void S_Tournament(out Creature parent1, out Creature parent2)
{

    SortedList<int, Creature> subList = ChooseSublist(setNumber);
    parent1 = subList.Last().Value;
    subList = ChooseSublist(setNumber);
    int indexOfParent1 = subList.IndexOfValue(parent1);
    if (indexOfParent1 != -1)
        subList.RemoveAt(indexOfParent1);//avoiding choosing the same parent twice
    parent2 = subList.Last().Value;
}
private SortedList<int, Creature> ChooseSublist(int setNumber)
{
    Random rnd = new Random();
    SortedList<int, Creature> subList = new SortedList<int, Creature>(new DuplicateKeyComparer<int>());
    for (int i = 0; i < setNumber; i++)
    {
        int randNumb = rnd.Next(0, _currPopulation.Count - 1);
        if (!subList.ContainsValue(_currPopulation.ElementAt(randNumb).Value))
            subList.Add(_currPopulation.ElementAt(randNumb).Value.F, _currPopulation.ElementAt(randNumb).Value);
        else
            i--;
    }
    return subList;
```

```
        }
        private void S_Proportionate(out Creature parent1, out Creature parent2)
        {
            int p1Index = ProportionalGetIndex();
            int p2Index;
            do
            {
                p2Index = ProportionalGetIndex();
            } while (p1Index == p2Index);
            parent1 = _currPopulation.ElementAt(p1Index).Value;
            parent2 = _currPopulation.ElementAt(p2Index).Value;
        }
        private int ProportionalGetIndex()
        {
            Random rnd = new Random();
            int randNumber;
            if (sumFitness > 0)
                randNumber = rnd.Next(0, sumFitness);
            else
                randNumber = rnd.Next(0, _currPopulation.Count - 1);
            int i = 0;
            int currSum = 0;
            while (currSum < randNumber && i < _currPopulation.Count)
            {
                currSum += _currPopulation.ElementAt(i).Value.F;
                i++;
            }
            if (i >= _currPopulation.Count)
                i = _currPopulation.Count - 1;

            return i;
        }
    private void Crossover(Creature parent1, Creature parent2, out Creature child1, out Creature child2)
        {
            //  creating 2 children
            bool[] childChromosome1 = new bool[n];
            bool[] childChromosome2 = new bool[n];
            Array.Copy(parent1.Chromosome, 0, childChromosome1, 0, genNum1);
            Array.Copy(parent2.Chromosome, genNum1, childChromosome1, genNum1, genNum2);
            Array.Copy(parent1.Chromosome, genNum1 + genNum2, childChromosome1, genNum1 + genNum2, genNum3);

            Array.Copy(parent2.Chromosome, 0, childChromosome2, 0, genNum1);
            Array.Copy(parent1.Chromosome, genNum1, childChromosome2, genNum1, genNum2);
            Array.Copy(parent2.Chromosome, genNum1 + genNum2, childChromosome2, genNum1 + genNum2, genNum3);

            child1 = new Creature(childChromosome1);
            child2 = new Creature(childChromosome2);
        }
        private Creature Mutation(Creature child)
        {
            Random rnd = new Random();
            if (rnd.NextDouble() <= mutationPossibility)
            {
                bool[] newChromosome = new bool[n];
                Array.Copy(child.Chromosome, newChromosome, n);
                int gen1, gen2;
                gen1 = rnd.Next(0, n);
                do
                {
                    gen2 = rnd.Next(0, n);
                } while (gen1 == gen2);
                newChromosome[gen1] = child.Chromosome[gen2];
                newChromosome[gen2] = child.Chromosome[gen1];//
                Creature mutatedChild = new Creature(newChromosome);
                if (mutatedChild.P > MaxWeight) //if mutatedChild is dead
                    return child;
```

```
            else
                return mutatedChild;
        }
        return child;
    }
    private Creature LocalImprovement(Creature child)
    {
      switch (imprMethod)
        {
            case LocalImprovementMethod.Superset:
                return LI_Superset(child);
            case LocalImprovementMethod.Subtitute:
                return LI_Subtitute(child);
            case LocalImprovementMethod.Hybrid:
                return LI_Hybrid(child);
            default:
                return child;
        }
    }
    private Creature LI_Superset(Creature child)    //local improvement method
    {
        Creature childImproved = child;
        bool[] newChromosome = new bool[n];
        int currF = child.F;
        int currP = child.P;
        Array.Copy(child.Chromosome, newChromosome, n);
        Random rnd = new Random();
            for (int i = 0; i < n; i++)
            {
                if (newChromosome[i] == false)
                {
                    if (currP + Creature.allItems[i].Weight <= MaxWeight && currF + Creature.allItems[i].Value > currF)// if
alive and have better F
                    {
                        newChromosome[i] = true;
                        Creature ch = new Creature(newChromosome);
                        if (childImproved.F < ch.F)
                            childImproved = ch;
                        newChromosome[i] = false;
                    }
                }
            }
        return childImproved;
    }
    private Creature LI_Subtitute(Creature child)   //local improvement method
    {
        Creature childImproved = child;
        bool[] newChromosome = new bool[n];
        int currF = child.F;
        int currP = child.P;
        Array.Copy(child.Chromosome, newChromosome, n);
        for (int i = 0; i < n; i++)
        {
            if (child.Chromosome[i] == false)
            {
                for (int j=0; j < n; j++)
                {
                    if (child.Chromosome[j] == true)
                    {
                        int tempP = currP + Creature.allItems[i].Weight - Creature.allItems[j].Weight;
                        int tempF = currF + Creature.allItems[i].Value - Creature.allItems[j].Value;
                        if (tempP <= MaxWeight && tempF > currF)
                        {
                            newChromosome[i] = true;
                            newChromosome[j] = false;
                            Creature ch = new Creature(newChromosome);
```

```
                        if (ch.F > childImproved.F) childImproved = ch;
                    newChromosome[i] = false;
                    newChromosome[j] = true;
                }
            }
        }
    }
  }
  return childImproved;
}
private Creature LI_Hybrid(Creature child)
{
  double LIMIT;
  if (child.ItemNumber < itemNumberLimit)
    LIMIT = 0.85;
  else
    LIMIT = 0.15;
  Random rnd= new Random();
  if (rnd.NextDouble() < LIMIT)
    return LI_Superset(child);
  else return LI_Subtitute(child);
}
private void AddChildToPopulation(Creature child)   //add child and remove the worst
{
  if (this.selectMethod == SelectionMethod.Proportionate)
  {
    sumFitness += child.F;
    sumFitness -= _currPopulation.ElementAt(0).Value.F;
  }
  if (bestCreature.F < child.F)
    bestCreature = child;
  _currPopulation.Add(child.F, child);
  _currPopulation.RemoveAt(0);
}

private int CalcFinessSum()
{
  int sum = 0;
  for(int i=0; i< _currPopulation.Count; i++)
    sum += _currPopulation.ElementAt(i).Value.F;
  return sum;
}
public Creature GetBest() => bestCreature;
}
      }
```

3.1.2 Приклади роботи

На рисунках 3.1, 3.2, 3.3 показані приклади роботи програми.

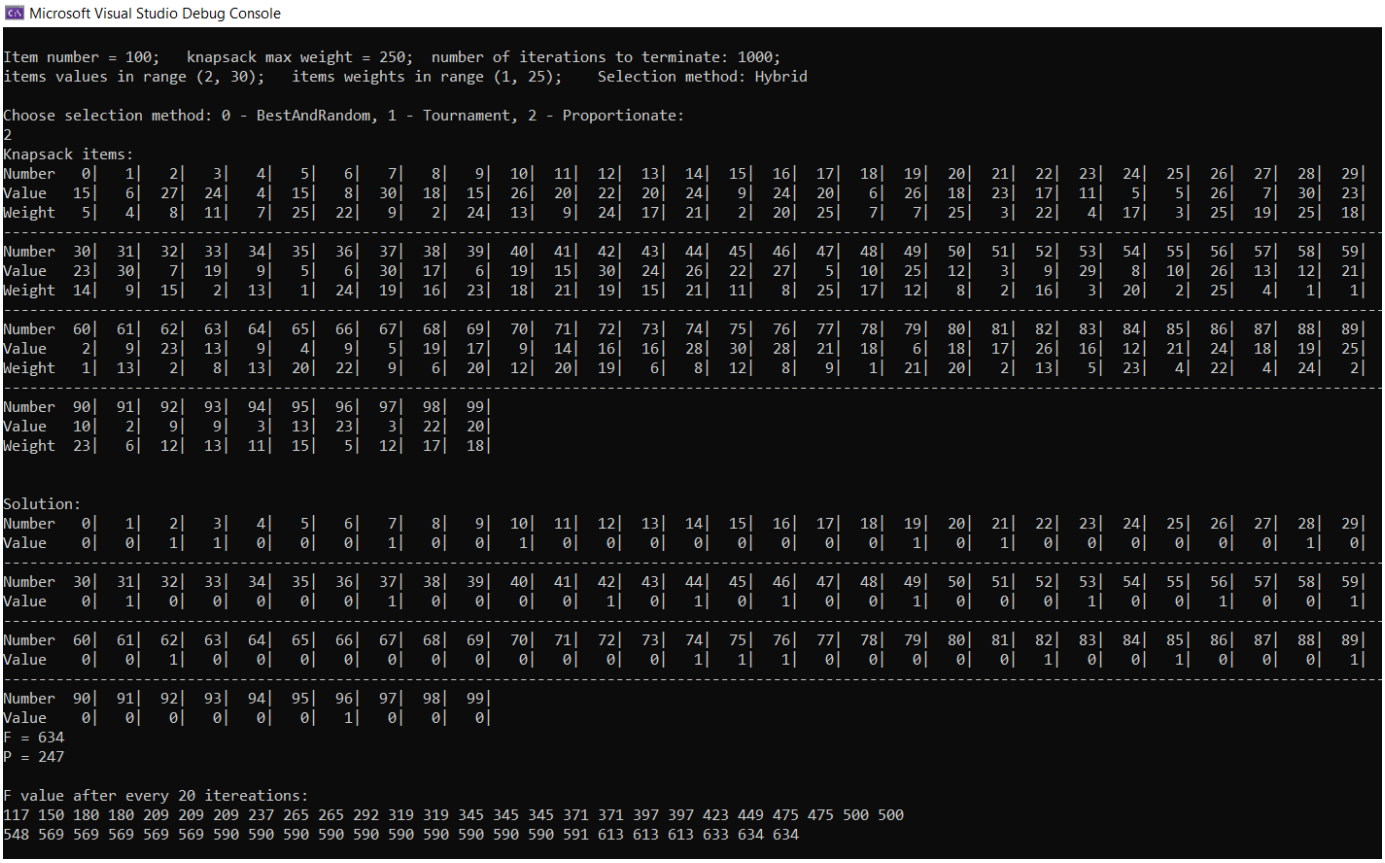

Рисунок 3.1 – Робота програми на розмірності 100 предметів та місткістю 250

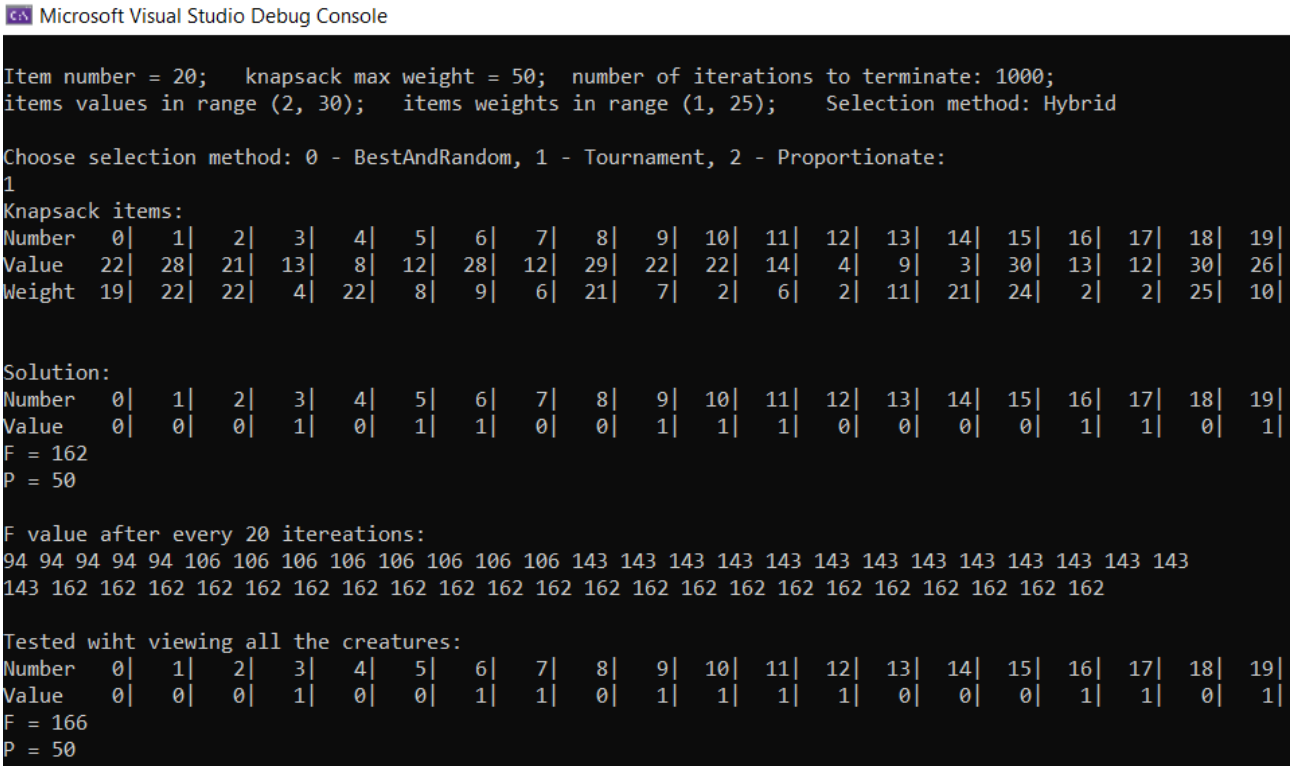

Рисунок 3.2 – Робота програми для 20 предметів, 1000 ітерацій та місткістю 50 з перевіркою шляхом перебору усіх розв’язків

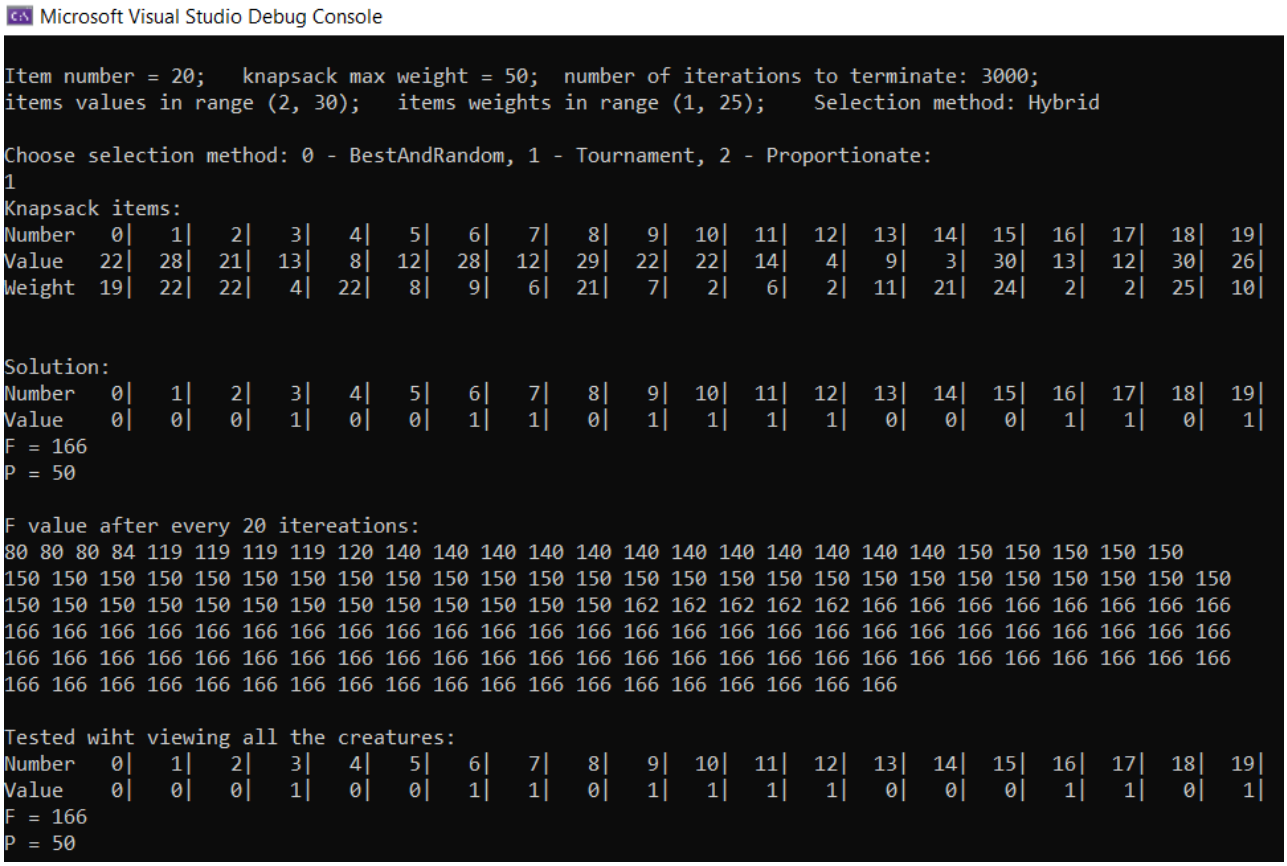

Рисунок 3.3 – Робота програми для 20 предметів, 3000 ітерацій та місткістю 50 з перевіркою шляхом перебору усіх розв’язків

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.

Таблиця 3.1

Номер ітерації	20	40	60	80	100	120	140	160	180	200	220	240	260
Значення цільової функції	117	150	180	180	209	209	209	237	265	292	319	319	345
Номер ітерації	280	300	320	340	360	380	400	420	440	460	480	500	520
Значення цільової функції	345	345	371	371	397	397	397	423	449	475	500	500	548
Номер ітерації	540	560	580	600	620	640	660	680	700	720	740	760	780
Значення цільової функції	569	569	569	569	569	569	590	590	590	590	590	590	590
Номер ітерації	800	820	840	860	880	900	920	940	960	980	1000		
Значення цільової функції	590	590	590	591	613	613	613	633	633	634	634		

У таблиці 3.2 представлено значення точності цільової функції найкращого нащадка відносно найкращого рішення, знайденого прямим перебором, за умови, що к-сть предметів = 20, місткість = 50. Способи селекції та локального покращення – Tournament та Hybrid відповідно. Для найкращого рішення, знайденого прямим перебором, F=166, P=50 Результати декількох експериментів можна побачити на рисунках 3.2 та 3.3

Таблиця 3.2

Номер експерименту /кількість ітерацій	1000 ітерацій		2000 ітерацій		3000 ітерацій	
	F	P	F	P	F	P
1	162	50	162	50	166	50
2	162	50	166	50	162	50
3	137	44	150	42	166	50
4	166	50	136	48	133	49
5	162	50	162	50	166	50
6	140	50	162	50	149	49
7	133	49	138	40	166	50
8	133	49	166	50	166	50
9	132	50	162	50	143	49
10	150	48	162	50	166	50

У таблиці 3.3 порівняно значення цільової функції найкращого нащадка з цільовою функцією найкращого рішення, знайденого прямим перебором. Кількість ітерацій – 1000, місткість – 50, кількість предметів – 20, спосіб селекції – Proportionate, спосіб локального покращення – Hybrid.

Таблиця 3.3

Номер експерименту /значення функцій	Генетичний алгоритм		Прямий перебір	
	F	P	F	P
1	206	48	206	48
2	135	50	142	50
3	148	48	148	48
4	215	49	215	49
5	134	50	134	50
6	138	50	142	50

7	107	50	107	50
8	155	50	157	48
9	127	49	127	49
10	180	49	180	49

3.2.2 Графіки залежності розв'язку від числа ітерацій

На рисунку 3.4 наведений графік, який показує якість отриманого розв'язку.

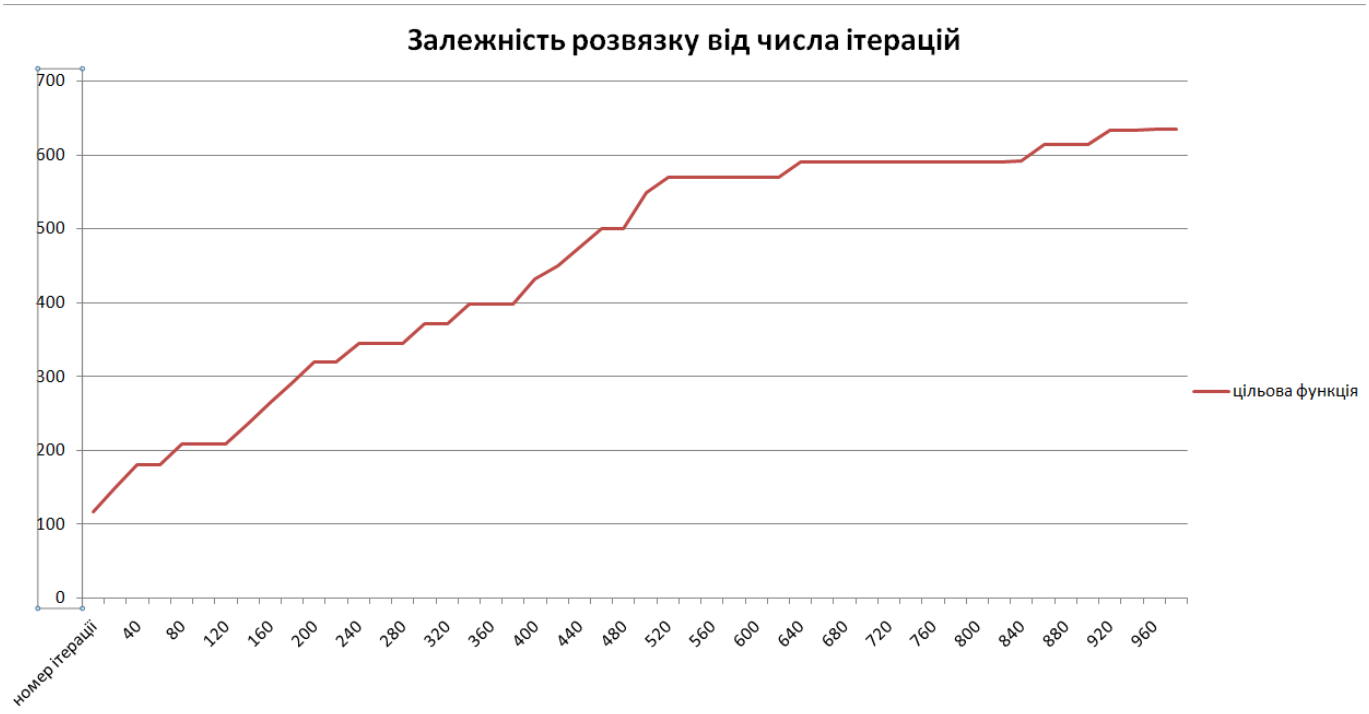

Рисунок 3.4 – Графіки залежності розв'язку від числа ітерацій

ВИСНОВОК

В рамках даної лабораторної роботи я реалізувала генетичний алгоритм для вирішення задачі про рюкзак. Виконавши реалізацію, я провела ряд досліджень для визначення ефективності та точності даного алгоритму. Спочатку я визначила зміну значення цільової функції для поточного нащадка через кожні 20 ітерацій і на основі отриманих значень побудувала графік залежності (рисунок 3.4). Далі я спробувала порівняти розв'язок задачі за допомогою генетичного алгоритму та шляхом перебору всіх можливих розв'язків задачі. Виявилося, що генетичний алгоритм не завжди знаходить найкраще рішення. З іншого боку, рішення, знайдене генетичним алгоритмом, у більшості випадків є близьким до рішення, що було отримано за допомогою прямого перебору (див. таблицю 3.3). Також розв'язок покращується зі збільшенням кількості ітерацій, тому при відносно великому значенні ітерацій рішення, знайдене генетичним алгоритмом, можна вважати якщо не найкращим, то оптимальним (дуже близьким до найкращого) (див. таблицю 3.2).

Отже, можна зробити висновок, що генетичний алгоритм варто використовувати на великих розмінностях вхідних даних, де прямий перебір всіх розв'язків є надто повільним. Якщо ж маємо справу з даними невеликої розмірності – варто застосувати прямий перебір (або інший метод пошуку в залежності для задачі), щоб отримане рішення було гарантовано найкращим.

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 27.11.2021 включно максимальний балдорівнює – 5. Після 27.11.2021максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

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