

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА
ПРИРОДОКОРИСТУВАННЯ

Звіт

З лабораторної роботи №2

На тему «Генетичний алгоритм»

**З дисципліни «Основи проектування систем штучного інтелекту та
розпізнавання образів»**

Підготував: Коваль Ігор Леонідович

Студент навчально-наукового інституту
автоматики, кібернетики та
обчислювальної техніки

група КН-51м

Рівне 2022

Мета роботи: створити програмний код для реалізації поставленого завдання з використанням генетичного алгоритму.

Теоретичні відомості

Генетичний алгоритм (Genetic algorithm) являє собою техніку оптимізації, яка моделює феномен природної еволюції (вперше відкритий Чарльзом Дарвіном). При природній еволюції виживають і дають саму більшу чисельність нащадків особи, найбільш адаптовані до складних умов навколишнього середовища. Ступінь адаптації, в свою чергу, залежить від набору хромосом конкретної особи, отриманих від батьків. Це основа виживання сильнішого – не тільки процес виживання, але й участь у формуванні наступного покоління. У природі виживання є визначальною і основною функцією. Генетичний алгоритм виконується в три етапи (якщо не враховувати початкове створення популяції). Під час оцінки визначається здоров'я популяції. Далі виконується відбір підгрупи хромосом на підставі попередньо заданого критерію. Нарешті, вибрана підгрупа рекомбінує, в результаті чого виходить нова популяція. Алгоритм виконується заново з новою популяцією. Процес триває до тих пір, поки не буде досягнуто певну межу. Тоді робота алгоритму вважається завершеною.

Створення початкової популяції (Initialization) дозволяє сформувати відправну точку для роботи алгоритму. Зазвичай це виконується шляхом довільного створення хромосом, але також допускається додавання в популяцію «здорових» хромосом.

Етап оцінки (Evaluation) дає можливість визначити, як кожна хромосома (рішення) справляється з даною проблемою. Алгоритм декодує хромосому стосовно проблеми і перевіряє результат вирішення проблеми з використанням нових парамет-

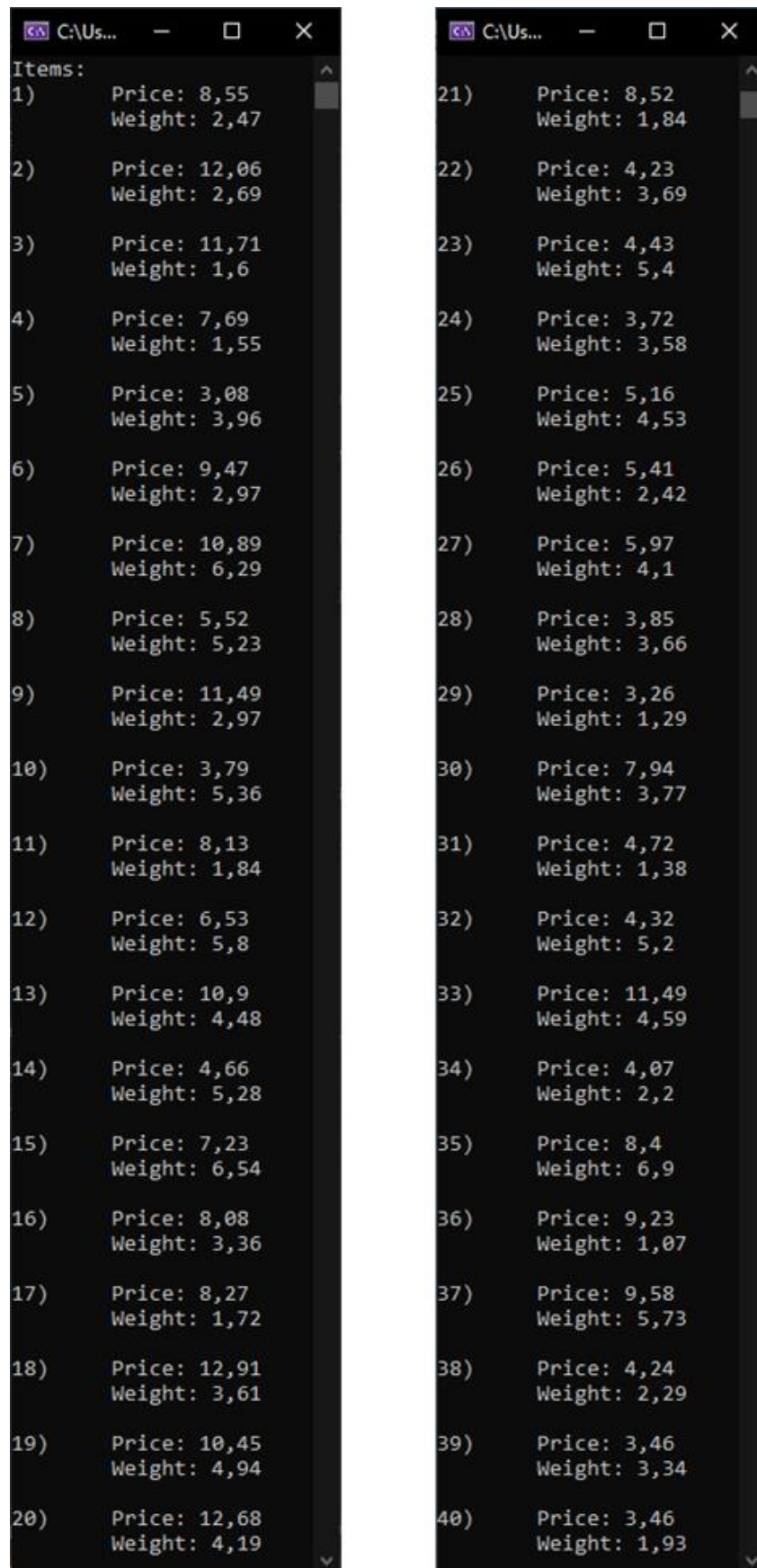
рів. Потім на підставі результату розраховується «здоров'я» хромосоми.

Відбір (Selection) є, ймовірно, найбільш важливим і найважчим етапом для розуміння генетичного алгоритму. На цьому етапі хромосоми вибираються для подальшого використання в іншій популяції. Відбір здійснюється на підставі здоров'я хромосом (тобто того, наскільки ефективно вони вирішують дану проблему). Цей процес є двозначним, тому що, якщо включити у вибір тільки дуже здорові хромосоми, то рішення стає обмеженим через недостатню різноманітність. Якщо вибір здійснюється довільно, то немає гарантій, що здоров'я наступних поколінь буде поліпшуватися. Тобто вибирається група хромосом, які будуть брати участь в рекомбінації (або схрещуванні).

При рекомбінації (Recombination) частини хромосом переміщуються, може бути, навіть змінюються, а отримані нові хромосоми повертаються назад в популяцію для формування наступного покоління. Перша група хромосом зазвичай називається батьками, а друга - дітьми. З однаковою ймовірністю можуть застосовуватися один або кілька генетичних операторів. Доступні оператори включають мутацію і перехресне схрещування, які являються аналогами однойменних генетичних процесів.

Результат роботи програми

Варіант 10 Розв'язати задачу наповнення рюкзака



Items:	
1)	Price: 8,55 Weight: 2,47
2)	Price: 12,06 Weight: 2,69
3)	Price: 11,71 Weight: 1,6
4)	Price: 7,69 Weight: 1,55
5)	Price: 3,08 Weight: 3,96
6)	Price: 9,47 Weight: 2,97
7)	Price: 10,89 Weight: 6,29
8)	Price: 5,52 Weight: 5,23
9)	Price: 11,49 Weight: 2,97
10)	Price: 3,79 Weight: 5,36
11)	Price: 8,13 Weight: 1,84
12)	Price: 6,53 Weight: 5,8
13)	Price: 10,9 Weight: 4,48
14)	Price: 4,66 Weight: 5,28
15)	Price: 7,23 Weight: 6,54
16)	Price: 8,08 Weight: 3,36
17)	Price: 8,27 Weight: 1,72
18)	Price: 12,91 Weight: 3,61
19)	Price: 10,45 Weight: 4,94
20)	Price: 12,68 Weight: 4,19
21)	Price: 8,52 Weight: 1,84
22)	Price: 4,23 Weight: 3,69
23)	Price: 4,43 Weight: 5,4
24)	Price: 3,72 Weight: 3,58
25)	Price: 5,16 Weight: 4,53
26)	Price: 5,41 Weight: 2,42
27)	Price: 5,97 Weight: 4,1
28)	Price: 3,85 Weight: 3,66
29)	Price: 3,26 Weight: 1,29
30)	Price: 7,94 Weight: 3,77
31)	Price: 4,72 Weight: 1,38
32)	Price: 4,32 Weight: 5,2
33)	Price: 11,49 Weight: 4,59
34)	Price: 4,07 Weight: 2,2
35)	Price: 8,4 Weight: 6,9
36)	Price: 9,23 Weight: 1,07
37)	Price: 9,58 Weight: 5,73
38)	Price: 4,24 Weight: 2,29
39)	Price: 3,46 Weight: 3,34
40)	Price: 3,46 Weight: 1,93

Рис. 1. Згенерований список предметів, їх вартість та вага

```
C:\Users\Goblin\Desktop\лабн 51\al\2sharp\bin\Debug\net6.0\2sharp.exe

Новий найкращий рюкзак з вартістю 49,85 на ітерації 0
0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1

Новий найкращий рюкзак з вартістю 52,73 на ітерації 2
1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1

Новий найкращий рюкзак з вартістю 61,43 на ітерації 4
1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1

Новий найкращий рюкзак з вартістю 64,76 на ітерації 6
0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1

Новий найкращий рюкзак з вартістю 65,15 на ітерації 9
1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1

Новий найкращий рюкзак з вартістю 72,28 на ітерації 10
0 1 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1

Новий найкращий рюкзак з вартістю 73,11 на ітерації 11
0 1 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
```

Рис. 2. Результат пошуку оптимального вибору предметів в рюкзаку

Код програми

```
using System.Collections;
using System.Diagnostics;
using System.Runtime.CompilerServices;

namespace MyProject;
class Program
{
    private const int NUMBER_OF_ITEMS = 40;
    private static Item[] items = new Item[NUMBER_OF_ITEMS];
    private const double MIN_ITEM_PRICE = 3.0;
    private const double MAX_ITEM_PRICE = 7.0;
    private const double MIN_ITEM_WEIGHT = 1.0;
    private const double MAX_ITEM_WEIGHT = 5.0;
    private const double MAX_WEIGHT = 20.0;
    private const int NUMBER_OF_BAGS = 10;
    private const int MAX_BAGS = 20;
    private const int MIN_BAGS = 3;
    private const int MAX_ITERATIONS = 1000000000;
    private const double MUTATION_CHANCE = 0.15;

    private struct Item
    {
        public double price;
        public double weight;

        public Item(double price, double weight)
        {
            this.price = price;
            this.weight = weight;
        }
    }

    public class BagsComparer : IComparer<bool[]>
    {
        public int Compare(bool[] x, bool[] y)
        {
            return f(x).CompareTo(f(y));
        }
    }

    static void Main(string[] args)
    {
        var rand = new Random();
        Console.WriteLine("Items: ");
        for(int i = 0; i < NUMBER_OF_ITEMS; ++i)
        {
            var price = MIN_ITEM_PRICE + (rand.NextDouble() * (MIN_ITEM_PRICE +
            MAX_ITEM_PRICE));
            var weight = MIN_ITEM_WEIGHT + (rand.NextDouble() * (MIN_ITEM_WEIGHT +
            MAX_ITEM_WEIGHT));
            price = Math.Round(price, 2);
            weight = Math.Round(weight, 2);
            items[i] = new Item(price, weight);

            Console.WriteLine($"{i + 1})\tPrice: {price}\n\tWeight: {weight}\n");
        }
        List<bool[]> bags = new List<bool[]>();
        for(int i = 0; i < NUMBER_OF_BAGS; ++i)
        {
            bags.Add(GenerateBag());
        }
    }
}
```

```

uint iteration = 0;

bags.Sort(new BagsComparer());

bool[] bestBag = new bool[NUMBER_OF_ITEMS];
for(int i = 0; i < NUMBER_OF_ITEMS; ++i)
{
    bestBag[i] = bags[bags.Count - 1][i];
}
Console.WriteLine($"Новий найкращий рюкзак з вартістю {f(bestBag)} на
ітерації {iteration}");
for(int i = 0; i < bestBag.Length; ++i)
{
    Console.Write(bestBag[i] ? "1 " : "0 ");
}
Console.WriteLine();

while (iteration < MAX_ITERATIONS)
{
    ++iteration;

    //int half = bags.Count / 2;

    //// Видалення поганих рюкзаків
    //for (int i = 0; i < half; ++i)
    //{
    //    bags.Remove(bags[0]);
    //}

    while (bags.Count > MAX_BAGS)
    {
        if (bags.Count <= MIN_BAGS)
        {
            break;
        }
        bags.Remove(bags[0]);
    }

    // Мутація старих рюкзаків з ймовірністю MUTATION_CHANCE
    var mutatedBags = MutateBags(bags);
    // Перехресне схрещування сусідніх рюкзаків
    var crossedBags = Cross(bags);

    bags.AddRange(mutatedBags);
    bags.AddRange(crossedBags);
    bags.Sort(new BagsComparer());

    if (f(bags[bags.Count - 1]) > f(bestBag)){
        //Console.WriteLine($"old - {f(bestBag)}    new - {f(bags[bags.Count
- 1])}");
        bestBag = new bool[NUMBER_OF_ITEMS];
        for (int i = 0; i < NUMBER_OF_ITEMS; ++i)
        {
            bestBag[i] = bags[bags.Count - 1][i];
        }
        Console.WriteLine($"
Новий найкращий рюкзак з вартістю {f(bestBag)}
на ітерації {iteration}");
        for (int i = 0; i < bestBag.Length; ++i)
        {
            Console.Write(bestBag[i] ? "1 " : "0 ");
        }
        Console.WriteLine();
    }
}

```

```

        Console.WriteLine($"\\nОбчислення завершено, за {MAX_ITERATIONS} ітерацій
знайдено рюкзак з вартістю {f(bestBag)}: ");
        for (int i = 0; i < bestBag.Length; ++i)
        {
            Console.Write(bestBag[i] ? "1 " : "0 ");
        }
        Console.WriteLine();
    }

    // Функція оцінки
    private static double f(bool[] selection)
    {
        double sumPrice = 0;
        double sumWeight = 0;
        for(int i = 0; i < selection.Length; ++i)
        {
            if (selection[i])
            {
                sumPrice += items[i].price;
                sumWeight += items[i].weight;
            }
        }

        if (sumWeight > MAX_WEIGHT)
        {
            sumPrice = 0.0;
        }

        return Math.Round(sumPrice, 2);
    }

    private static bool[] GenerateBag()
    {
        var rand = new Random();
        bool[] itemsInBag = new bool[NUMBER_OF_ITEMS];
        double sumWeight = 0;
        for (int i = 0; i < NUMBER_OF_ITEMS; ++i)
        {
            itemsInBag[i] = rand.Next() % 2 == 0;
            if (itemsInBag[i])
            {
                sumWeight += items[i].weight;
            }
        }

        while(sumWeight > MAX_WEIGHT)
        {
            var index = rand.Next() % (NUMBER_OF_ITEMS - 1);

            if (itemsInBag[index])
            {
                itemsInBag[index] = false;
                sumWeight -= items[index].weight;
            }
        }

        return itemsInBag;
    }

    private static List<bool[]> MutateBags(List<bool[]> oldBags)
    {
        List<bool[]> newBags = new List<bool[]>();
        var rand = new Random();
        foreach (bool[] bag in oldBags)

```



```

    {
        if(rand.NextDouble() <= MUTATION_CHANCE)
        {
            bool[] newBag = new bool[NUMBER_OF_ITEMS];
            for (int i = 0; i < NUMBER_OF_ITEMS; ++i)
            {
                newBag[i] = bag[i];
            }
            int index = rand.Next() % (bag.Length - 1);
            newBag[index] = !newBag[index];
            newBags.Add(newBag);
        }
    }

    return newBags;
}

private static List<bool[]> Cross(List<bool[]> oldBags)
{
    List<bool[]> newBags = new List<bool[]>();
    var rand = new Random();
    for(int i = 0; i < oldBags.Count - 1; ++i)
    {
        int crossingPoint = rand.Next() % (NUMBER_OF_ITEMS - 1);
        bool[] newBag = new bool[NUMBER_OF_ITEMS];
        for (int j = 0; j < NUMBER_OF_ITEMS; ++j)
        {
            if(j < crossingPoint)
            {
                newBag[j] = oldBags[i][j];
            }
            else
            {
                newBag[j] = oldBags[i + 1][j];
            }
        }
        newBags.Add(newBag);
    }

    return newBags;
}
}

```