Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №6
По дисциплине: «Методы решения задач в интеллектуальных системах»
Тема: «Решение задач с помощью генетических алгоритмов»

Выполнили:
Студены 2 курса
Группы ИИ-17(1)
Курский Н.В.
Мельничук Д.А.
Лицкевич Д.С.
Проверил:
Анфилец С.В.

Брест 2021

Цель работы: ознакомиться с подходом к решению оптимизационных задач с помощью генетических алгоритмов (ГА).

Задания:

1). Составить генетический алгоритм на основе методов, заданных по варианту.

2). Разработать компьютерную программу, которая осуществляет поиск кратчайшего пути для информационного пакета (сообщения) в компьютерной сети с помощью генетического алгоритма.

3). Для проведения серии экспериментов программа должна позволять пользователю задавать топологию сети (пропускные способности каналов связи), содержащей не менее 10 компьютеров (серверов), а также указывать компьютер-отправитель и компьютер-получатель. На экране должны отображаться все представители (хромосомы) одного поколения до и после применения каждого оператора (скрещивания, селекции, редукции и мутации). Переход к следующему поколению должен осуществляться при нажатии соответствующей кнопки или в автоматическом режиме.

Метод выбора родителей	Метод скрещивания	Метод мутации	Метод отбора
Аутбридинг	Одноточечный кроссинговер	Обмен	Отбор усечением

Код программы:

```
using System;

namespace MRZvIS_6
{
    class Program
    {
        static void Main(string[] args)
        {
            Genetic g = new Genetic(@"..\..\..\..\matrix.txt");
            Console.Write("Enter the first server: ");
            int startServer = int.Parse(Console.ReadLine());
            Console.Write("Enter the final server: ");
            int finalServer = int.Parse(Console.ReadLine());
            g.Run(startServer, finalServer);
            Console.Read();
        }
    }
}
```

Genetic.cs

```
using System;
using System.IO;

namespace MRZvIS_6

{
    class Genetic
    {
        public int generationCount { get; set; } // Кол-во итераций
        public int populationSize { get; set; } // Размер популяции
        public double mutationPercent { get; set; } // Процент мутации

        private Random rng = new Random();
        private int start; // Стартовый сервер
        private int end; // Конечный сервер
        private int[][] Individuals; // Особи
        private int[][] Offsprings; // Родители
```

```
        private int individualSize; // Размер особи
        private int[,] Matrix; // Матрица весов
        private int serverCount; // Количество серверов
        private double T; // Коэффициент для отбора усечением

        public Genetic(string address)
        {
            // Считывание кол-ва серверов и матрицы из файла
            using (StreamReader fileReader = new StreamReader(address))
            {
                serverCount = Convert.ToInt32(fileReader.ReadLine()); // Считали кол-во
серверов
                Matrix = new int[serverCount, serverCount];
                for (int i = 0; i < serverCount; i++)
                {
                    string tmp = fileReader.ReadLine(); // Считали очередную строку из
файла
                    string[] buf = tmp.Split(' '); // Разделили ее на подстроки по
пробелам. Например, была строка: "123 124 125", из нее получится три строки: "123",
"124", "125"
                    for (int j = 0; j < serverCount; j++)
                    {
                        Matrix[i, j] = Convert.ToInt32(buf[j]); // Запись в матрицу
                    }
                }
            }
            // Установка параметров ГА
            mutationPercent = 0.2d;
            populationSize = 40;
            generationCount = 4000;
            individualSize = serverCount - 2; // Размер особи = количество серверов (N) -
2, т.к. первая и конечная точка не должны меняться
            T = 0.8d;
        }

        // Главный метод, который ищет маршрут
        public void Run(int s, int e)
        {
            start = s;
            end = e;
            Individuals = new int[populationSize][];
            Offsprings = new int[populationSize][];
            for (int i = 0; i < populationSize; i++)
            {
                Individuals[i] = new int[individualSize];
                Offsprings[i] = new int[individualSize];
            }
            FirstGeneration(); // Первая генерация особей. Забивает их случайными числами
от 0 до количества серверов
            for (int i = 0; i < generationCount; i++)
            {
                Outbreeding(); // Выбираем родителей

                SinglePointCrossover(); // Скрещивание одноточечным кроссинговером

                Mutate(); // Мутация

                TruncationSelection(); // Выборка лучших из родителей в новую популяцию
            }
            DisplayBest(); // Вывод лучшего результата
        }

        private void FirstGeneration()
        {
```

```
            for (int i = 0; i < populationSize; i++)
            {
                for (int j = 0; j < individualSize; j++)
                {
                    Individuals[i][j] = rng.Next(serverCount);
                }
            }
        }
        // Находит лучший результат и выводит его
        private void DisplayBest()
        {
            // Сначала считаем путь для каждой особи
            int[] length = new int[populationSize];
            for (int j = 0; j < populationSize; j++)
            {
                length[j] = GetLength(Individuals[j]);
            }
            // Находим наикратчайший путь
            int bestResult = length[0];
            int bestResultIndex = 0;
            for (int i = 1; i < populationSize; i++)
            {
                if (length[i] < bestResult)
                {
                    bestResult = length[i];
                    bestResultIndex = i;
                }
            }
            /*Выводим его. Стартовая вершина, потом все остальные. Early используем для
того, чтобы не выводить повторяющиеся значения.
                    Путь вполне может быть: 1 1 1 1 1. Из 1 в 1 длина пути составит 0.
Поэтому выводим 1 только один раз.*/
            Console.Write("Best path: {0} ", start);
            int early = start;
            for (int i = 0; i < individualSize; i++)
            {
                if (Individuals[bestResultIndex][i] == early) continue;
                Console.Write("{0} ", Individuals[bestResultIndex][i]);
                early = Individuals[bestResultIndex][i];
                if (Individuals[bestResultIndex][i] == end) break; // Если в пути
встретилась конечная вершина, то перестать выводить его
            }
            if (end != early) Console.WriteLine(end); // Если не было конечной вершины,
то вывести ее
            else Console.WriteLine();
            Console.WriteLine("Length: {0}", bestResult); // Вывод длины пути
        }
        // Выборка родителей. Аутбридинг
        private void Outbreeding()
        {
            int hemmingCurr = 0;
            int hemmingMax = 0;
            int hemmingMaxID = 0;
            for (int i = 0; i < populationSize; i += 2)
            {
                int randomIndivid = rng.Next(populationSize); // Выборка случайной особи
                for (int j = 0; j < populationSize; j++)
                {
                    // Считывание для особи максимального хэммингового расстояния. Оно
считается как разница между каждым геном.
                    for (int k = 0; k < individualSize; k++)
                    {
                        hemmingCurr += Math.Abs(Individuals[j][k] -
Individuals[randomIndivid][k]);
```

```
                    }
                    if (hemmingCurr > hemmingMax)
                    {
                        hemmingMax = hemmingCurr;
                        hemmingMaxID = j;
                    }
                    hemmingCurr = 0;
                }
                /* Добавляем в родителей случайно выбранную особь */
                Offsprings[i] = Individuals[randomIndivid];
                /* Добавляем в родителей вторую особь, у которой хеммингово расстояние
максимально */
                Offsprings[i + 1] = Individuals[hemmingMaxID];
                hemmingMax = 0;
            }
        }
        /* Скрещивание. Одноточечный кроссинговер */
        private void SinglePointCrossover()
        {
            for (int i = 0; i < populationSize; i += 2)
            {
                int point = rng.Next(individualSize - 1); // Выбираем случайную точку, по
которой будем обмениваться генами
                for (int j = point; j < individualSize; j++) // Меняем все гены от этой
точки до конца
                {
                    int tmp = Offsprings[i][j];
                    Offsprings[i][j] = Offsprings[i + 1][j];
                    Offsprings[i + 1][j] = tmp;
                }
            }
        }

        private void Mutate()
        {
            for (int i = 0; i < populationSize; i++)
            {
                /* NextDouble возвращает случайное число от 0 до 1. Если оно меньше
нашего процента мутации, то особь мутирует */
                if (rng.NextDouble() <= mutationPercent)
                {
                    int numberGen = rng.Next(1, individualSize - 1); // Выбираем
случайное место, где будет мутация
                    int tmp = Offsprings[i][numberGen - 1]; // Меняем местами соседние
гены
                    Offsprings[i][numberGen - 1] = Offsprings[i][numberGen + 1];
                    Offsprings[i][numberGen + 1] = tmp;
                }
            }
        }
        // Отбор усечением
        private void TruncationSelection()
        {
            int[][] bestIndividuals = new int[populationSize * 2][]; // Массив особей,
состоящий из самих особей и их родителей
            int[] bestIndividualsResult = new int[populationSize * 2]; // Массив со
значениями длины пути в каждой особи
            int i = 0;
            /* Сначала добавляем самих особей */
            for (i = 0; i < populationSize; i++)
            {
                bestIndividuals[i] = Individuals[i];
                bestIndividualsResult[i] = GetLength(Individuals[i]);
            }
```

```
            /* Добавление их родителей */
            for (; i < populationSize * 2; i++)
            {
                bestIndividuals[i] = Offsprings[i - populationSize];
                bestIndividualsResult[i] = GetLength(Offsprings[i - populationSize]);
            }
            /* Сортировка пузырьком по возрастанию */
            for (i = 0; i < populationSize * 2; i++)
            {
                for (int j = 0; j < populationSize * 2; j++)
                {
                    if (bestIndividualsResult[i] < bestIndividualsResult[j])
                    {
                        int tmp = bestIndividualsResult[j];
                        bestIndividualsResult[j] = bestIndividualsResult[i];
                        bestIndividualsResult[i] = tmp;
                        int[] temp = bestIndividuals[j];
                        bestIndividuals[j] = bestIndividuals[i];
                        bestIndividuals[i] = temp;
                    }
                }
            }
            int n = Convert.ToInt32(T * (populationSize * 2)); // Количество лучших
особей, среди которых будет выбираться новая популяция
            for (i = 0; i < populationSize; i++)
            {
                Individuals[i] = bestIndividuals[rng.Next(n + 1)];
            }
        }
        /* Метод класса, считывающий длину пути */
        public int GetLength(int[] path)
        {
            int result = Matrix[start, path[0]]; // Считывание сначала от стартовой точки
до первой в особи
            for (int i = 0; i < path.Length - 1; i++)
            {
                if (path[i] == end) return result; // Если конечная вершина, то путь
посчитан
                result += Matrix[path[i], path[i + 1]];
            }
            result += Matrix[path[path.Length - 1], end]; // К полученному результату
добавляем путь до конечной вершины
            return result;
        }
    }
}
```

```
Select C:\Users\Crazy_pro\Documents\Visual Studio 2017\Projects\MRZvIS_6\MRZvIS_6\bin\Debug\MRZvIS_6.exe
Enter the first server: 0
Enter the final server: 9
Best path: 0 3 7 9
Length: 348
```

Вывод: в ходе данной лабораторной работы ознакомились с подходом к решению оптимизационных задач с помощью генетических алгоритмов (ГА).