**Санкт-Петербургский государственный университет информационных технологий, механики и оптики**
**Факультет информационных технологий и программирования**
**Кафедра «Компьютерные Технологии»**

**А. С. Афанасьева**

# Отчет по лабораторной работе

# «Построение управляющих автоматов с помощью генетических алгоритмов»

**Вариант №64**

**Санкт-Петербург**

**2010**

# Оглавление

# 1. Введение

## 1.1 Общая постановка задачи

Цель лабораторной работы – применение генетического алгоритма для построения конечного автомата, решающего задачу об «умном муравье-3» [1].

Решение создается в рамках виртуальной лаборатории «GlOpt» («Global Optimization»), написанной на языке C# [2]. Требуется добавить плагин, который реализует конкретный тип генетического алгоритма и конкретное представление особей-автоматов.

## 1.2 Задача об «умном муравье-3»

Действие задачи происходит на поверхности тора размером 32×32 клетки. В некоторых клетках находится еда. Муравей начинает движение из фиксированной стартовой клетки.

За ход муравей может выполнить следующие действия:

- повернуть налево;
- повернуть направо;
- сделать шаг вперед и, если в новой клетке есть еда, съесть ее;

Муравей видит перед собой восемь клеток (см. рис. 1):

Рис. 1. Фрагмент визуализатора к задаче «умный муравей – 3»

Максимальное число ходов, которое может совершить муравей, равно двумстам. Целью решения задачи является создание конечного автомата с минимальным числом состояний, управляющего муравьем, который за минимальное число ходов съест как можно больше яблок. В качестве **функции приспособленности** выступает число съеденных муравьем яблок минус отношение числа ходов до последнего съеденного яблока к допустимому числу ходов.

Особи задачи об «умном муравье-3» представлены конечными автоматами Мили. Выходное воздействие в этих автоматах определяется текущим состоянием и входным воздействием. Число состояний автомата является настраиваемым параметром задачи, а входным воздействием является наличие или отсутствие еды впереди муравья.

# 2. Алгоритм решения

## 2.1 Представление автомата

В качестве метода представления автомата используется **метод сокращенных таблиц [3].** Этот метод основан на том, что в каждом состоянии значимым является лишь определенное, небольшое подмножество предикатов. Это свойство позволяет существенно сократить размер описания состояний.

Реализация представления автомата выглядит следующим образом. Число значимых предикатов ограничивается некоторой константой **r.** Множество значимых предикатов описывается битовым вектором, в котором r бит равны единицам, а остальные биты равны нулям.

| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | $X_7$ | $X_8$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

Рис. 2. Вектор, описывающий множество значимых предикатов, r = 2

| {X} / S | 00 | 01 | 10 | 11 |
|---|---|---|---|---|
| 1 | $(Z_m, S_k)$ | … | | |
| 2 | ... | | | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |
| 8 | | | | |

Рис. 3. Схема таблицы переходов. Число значимых предикатов r = 2. Число состояний автомата s = 8

Автомат представлен в виде таблицы переходов. Индекс строки соответствует номеру начального состояния, а индекс столбца соответствует значению предикатов. Число столбцов равно $2^r$. В ячейках таблицы хранятся пары «действие – целевое состояние».

## 2.2 Метод мутации

**Мутация множества значимых предикатов.** Каждый из значимых предикатов с некоторой вероятностью заменяется другим, который не принадлежит множеству.

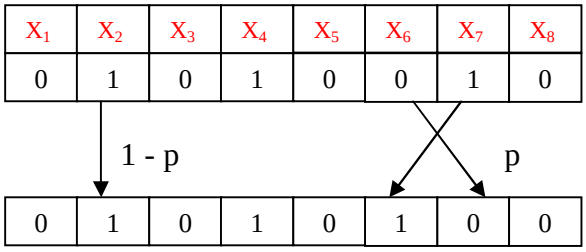


Рис. 4. Пример мутации множества значимых предикатов

**Мутация таблицы** аналогична мутации в случае полных таблиц. С некоторой вероятностью может мутировать каждый элемент таблицы. При этом номер целевого состояния и тип действия заменяются на любые из допустимых.

## 2.3 Метод скрещивания

Реализован метод, подробное описание которого можно найти в работе [3].

**Выбор значимых предикатов для потомков.** Предикаты, значимые для обоих родителей, наследуются обоими потомками, а каждый из тех предикатов, которые были значимы лишь для одной родительской особи, равновероятно достаются любому из двух потомков.

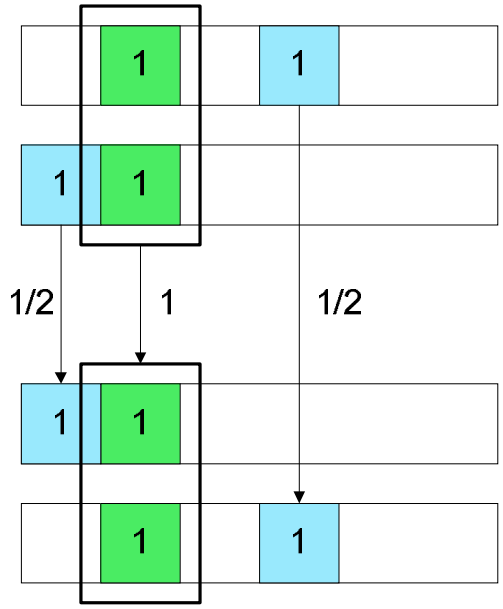


Рис. 5. Пример выбора значимых предикатов потомков

**Заполнение таблиц потомков.** Целевые состояния для каждого начального состояния при фиксированном значении предикатов (т.е. при фиксированном столбце) выбираются случайно с распределением вероятностей, задаваемым значениями в соответствующих столбцах родительских таблиц.

## 2.4 Метод отбора

В качестве метода отбора используется **элитизм [4]**. Фиксированная доля лучших особей переходит в следующее поколение напрямую. Остальные особи с некоторой вероятностью скрещиваются или мутируют, и их потомки переходят в следующее поколение.

## 2.5 Тип генетического алгоритма

Реализован **клеточный генетический алгоритм [4].** Особи расположены в ячейках тора. Скрещивание производится с лучшей особью из тех, что находятся в соседних клетках (соседними считаются клетки, расположенные сверху, снизу, слева и справа от данной). Полученный потомок помещается в свою ячейку вместо родителя.

# 3. Результаты работы алгоритма

Построен конечный автомат с одним состоянием. Муравей съедает все яблоки за 147 шагов, величина функции приспособленности равна 88,27. Скорость построения автомата составляет в среднем две секунды.

Лучшая особь была выведена при следующих значениях настраиваемых параметров, указанных в таблице:

| **Параметр** | **Значение** |
|---|---|
| Число состояний | 1 |
| Число значимых предикатов | 3 |
| Период большой мутации | 500 |
| Вероятность большой мутации | 0.05 |
| Доля лучших особей в методе элитизма | 0.1 |
| Размер поколения | 100 |
| Вероятность мутации | 0.1 |

Из графика функции приспособленности видно, что реализованный клеточный генетический алгоритм имеет несомненные преимущества в скорости нахождения оптимального решения по сравнению с поиском «в лоб» (см. рис. 6).

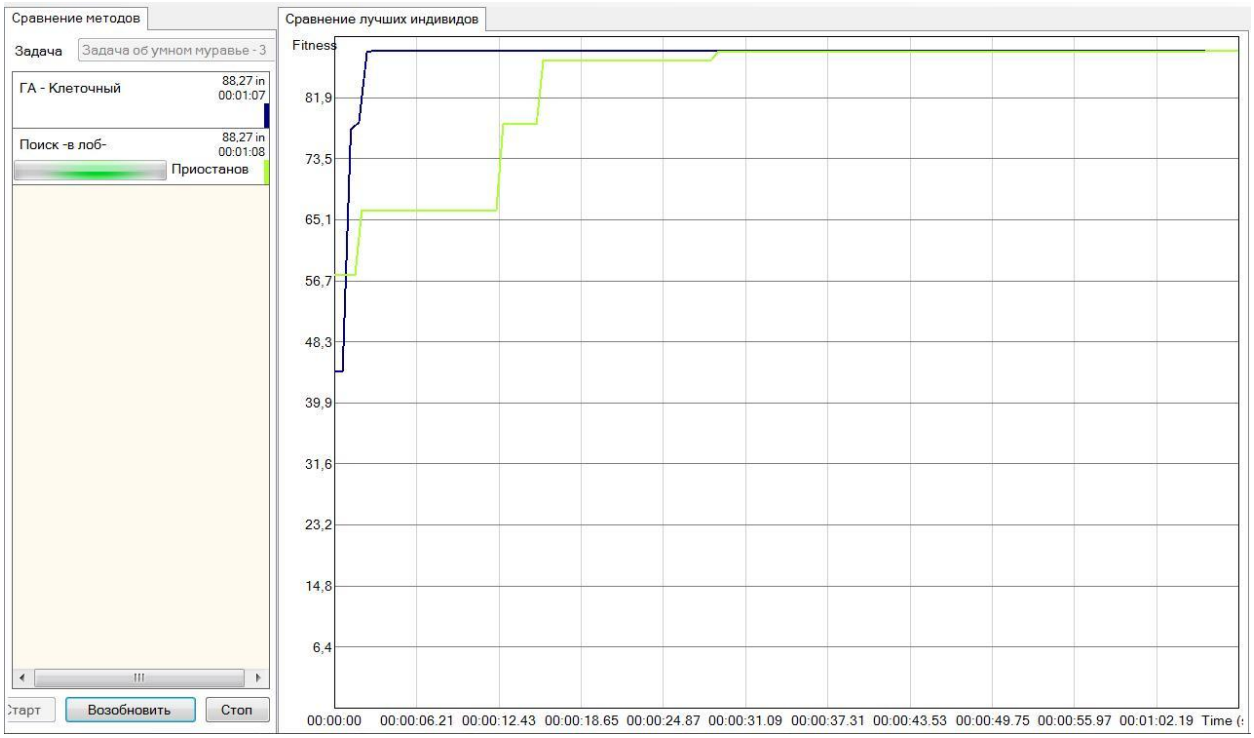


Рис. 6. График значений функции приспособленности

# 4. Заключение

Применение метода сокращенных таблиц позволяет успешно решить задачу об «умном муравье-3» в условиях большего, чем в задаче об «умном муравье» [5], числа входных переменных. Клеточный генетический алгоритм позволяет построить автомат, который эффективно и быстро решает задачу об «умном муравье-3».

# Источники


1. Инструкция по работе с виртуальной лабораторией GlOpt. Описание задач
   http://is.ifmo.ru/genalg/labs_2010-2011/GlOpt_problems.doc
2. Инструкция по работе с виртуальной лабораторией GlOpt. Описание виртуальной лаборатории
   http://is.ifmo.ru/genalg/labs_2010-2011/GlOpt_instruction.doc
3. Точилин В.Н. Метод сокращенных таблиц для генерации автоматов с большим числом входных воздействий на основе генетического программирования
   http://is.ifmo.ru/papers/tochilin/doc.pdf
4. Яминов Б. Генетические алгоритмы.
   http://rain.ifmo.ru/cat/view.php/theory/unsorted/genetic-2005
5. А. А. Давыдов, Д. О. Соколов, Ф. Н. Царев. Применение генетических алгоритмов для построения автоматов Мура и систем взаимодействующих автоматов Мили на примере задачи об «умном муравье».
   http://is.ifmo.ru/works/_2009_08_12_davydov.pdf

# Приложение. Исходный код

## 1. Файл Automation.cs

```
#region Usings

using System;
using System.Collections;
using System.Collections.Generic;
using ArtificialAnt3.ProblemInfo;
using NewBrain;

#endregion

namespace ArtificialAnt3.IndividualInfo
{
    /// <summary>
    /// Автомат к задаче об "умном муравье-3" (по методу сокращенных таблиц)
    /// </summary>
    public class Automation : Individual
    {
       public const int PRED_COUNT = 8;
        /// <summary>
        /// Новый автомат
        /// </summary>
        /// <param name="initialState">Начальное состояние</param>
        /// <param name="statesCount">Число состояний</param>
        /// <param name="predCount">Число предикатов</param>
        public Automation(int initialState, int statesCount, int predCount)
        {
            InitialState = initialState;
            StatesCount = statesCount;
            ActualPredCount = predCount;

            Predicates = new BitArray(PRED_COUNT);
            InputCount = (int)Math.Pow(2, ActualPredCount);

            Table = new Transition[StatesCount, InputCount];

        }
        /// <summary>
        /// Начальное состояние автомата
        /// </summary>
        public int InitialState { get; set; }

        /// <summary>
        /// Число состояний автомата
        /// </summary>
        public int StatesCount { get; protected set; }

        /// <summary>
        /// Число возможных комбинаций значений значимых предикатов
        /// </summary>
        public int InputCount { get; private set; }

        /// <summary>
        /// Число значимых предикатов
        /// </summary>
        public int ActualPredCount { get; protected set; }

        public Transition[,] Table { get; private set; }

        /// <summary>
```

```
/// Вектор, описывающий множество предикатов
/// </summary>
public BitArray Predicates { get; private set; }

public List<int> getActualPreds()
{
    List<int> list = new List<int>();
    for (int i = 0; i < PRED_COUNT; i++)
        if (Predicates[i])
            list.Add(i);
    return list;
}
#region Nested type: Transitions

/// <summary>
/// Переход, состоящий из действия и конечного состояния по выполнению этого действия.
/// </summary>
public class Transition
{
    public Transition(int endState, SimpleAntProblem.AntActions action)
    {
        EndState = endState;
        Action = action;
    }

    /// <summary>
    /// Конечное состояние
    /// </summary>
    public int EndState { get; private set; }

    /// <summary>
    /// Действие
    /// </summary>
    public SimpleAntProblem.AntActions Action { get; protected set; }

    public override string ToString()
    {
        return String.Format("-{0}->{1}", Action, EndState);
    }
}

#endregion

public Transition GetTransition(int state, BitArray input)
{

    return Table[state, BitArrayToInt(input)];
}

public Transition GetTransition(int state, int input)
{
    return Table[state, input];
}

public void SetTransition(int state, BitArray input, Transition transition)
{
    Table[state, BitArrayToInt(input)] = transition;
}

public void SetTransition(int state, int input, Transition transition)
{
    Table[state, input] = transition;
}

private int BitArrayToInt(BitArray b)
{
```

```
            int size = b.Length;
            int result = 0;
            int k = 1;
            for (int i = size - 1; i >= 0; i--)
            {
                int digit = b[i] ? 1 : 0;
                result += digit * k;
                k *= 2;
            }
            return result;
        }
    }
}
```

## 2. Файл SimpleAntProblem.cs

```
#region Usings

using System;
using System.Windows.Forms;
using ArtificialAnt3._Internal.Mover;
using ArtificialAnt3._Internal.Phenotype;
using ArtificialAnt3.IndividualInfo;
using ArtificialAnt3.Properties;
using ArtificialAnt3.AntViewer;
using NewBrain;
using NewBrain.ComponentModel;

#endregion

namespace ArtificialAnt3.ProblemInfo
{
    [IndividualType(typeof (Automation))]
    [IndividualViewer(typeof(AntViewer))]
     public class SimpleAntProblem : Problem
    {
        #region AntActions enum
        /// <summary>
        /// Возможные типы действий муравья
        /// </summary>
        public enum AntActions : byte
        {
            Left = 0,
            Right,
            Move
        }

        #endregion

        /// <summary>
        /// Направление оптимизации - максимизация функции
        /// </summary>
        public override OptimizationDirection OptimizationDirection
        {
            get { return OptimizationDirection.Maximize; }
        }

        public override string HtmlDescription
        {
            get
            {
                return Resources.artAnt;
            }
        }

        public override string Title
        {
            get { return Properties.Resources.ArtAnt3Problem; }
        }

        public override string Description
        {
            get { return ""; }
        }
```

```
        /// <summary>
        /// Вычисление fitness-функции текущей особи –
        ///числа съеденных яблок на торе за ограниченное число ходов.
        /// </summary>
        /// <param name="individual">Особь для подсчета значения fitness-функции</param>
        /// <returns>Значение fitness-функции</returns>
        public override double EvaluateIndividual(Individual individual)
        {
            var a = individual as Automation;
            if (a == null)
                return double.NaN;

            var mover = new SimpleMover(a);
            mover.Reset();
            int apples = 0;
            int lastStep = 0;
            for (int i = 0; i < Ant.MaxStepsCount; ++i)
            {
                if (mover.Move())
                {
                    apples++;
                    lastStep = i;
                }
                if (apples == Ant.FoodCount)
                    break;
            }
            return apples - 1.0*lastStep/Ant.MaxStepsCount;
        }
    }

    internal class AntViewer : IIndividualViewer
    {
        /// <summary>
        /// Визуализатор особи - муравья
        /// </summary>
        /// <param name="individual">Особь для визуализации</param>
        public void ViewIndividual(Individual individual)
        {
            Automation a = individual as Automation;
            if (a == null)
                throw new ArgumentException();

            using (var form = new AntViewerForm())
            {
                form.fieldControl.Mover = new SimpleMover(a);
                form.ShowDialog();
            }
        }
    }
}
```

## 3. Файл GeneticSearchOperatorOnAutomation.cs

```
#region Usings

using System;
using System.ComponentModel;
using System.Collections.Generic;
using System.Collections;
using ArtificialAnt3.IndividualInfo;
using ArtificialAnt3.ProblemInfo;
using Genetic.Operators;
using NewBrain;
using NewBrain.Attributes;

#endregion

namespace Genetic
{
    [IndividualType(typeof(Automation))]
    public abstract class SearchOperatorOnAutomation3 : SearchOperator
    {
        public SearchOperatorOnAutomation3()
        {
            StatesCount = 8;
            ActualPredCount = 3;
        }

        [Configurable]
        [Description("Число состояний")]
        public int StatesCount { get; set; }

        [Configurable]
        [Description("Число значимых предикатов")]
        public int ActualPredCount { get; set; }  //

        public override Individual Create(Random random)
        {
            Automation automation = new Automation(0, StatesCount, ActualPredCount);
            for (int i = 0; i < StatesCount; i++)
                for (int j = 0; j < automation.InputCount; j++)
                    automation.SetTransition(i, j, new Automation.Transition(random.Next(StatesCount),
GetRandomAction(random)));
            for (int i = 0; i < ActualPredCount; i++)
            {
                int index;
                do
                    index = random.Next(automation.Predicates.Length);
                while (automation.Predicates[index]);

                automation.Predicates[index] = true;
            }

            return automation;
        }

        protected SimpleAntProblem.AntActions GetRandomAction(Random random)
        {
            SimpleAntProblem.AntActions action = SimpleAntProblem.AntActions.Left;
            switch (random.Next(3))
            {
                case 0:
                    action = SimpleAntProblem.AntActions.Left;
                    break;
                case 1:
                    action = SimpleAntProblem.AntActions.Move;
                    break;
```

```
                case 2:
                    action = SimpleAntProblem.AntActions.Right;
                    break;
            }
            return action;
        }
    }

    [IndividualType(typeof(Automation))]
    public class GeneticSearchOperatorOnAutomation3 : SearchOperatorOnAutomation3,
IGeneticSearchOperator
    {
        public override string Title
        {
            get { return Properties.Resources.GeneticOperOnAuto; }
        }

        public override string Description
        {
            get { return Properties.Resources.GeneticOperOnAuto; }
        }

        #region IGeneticSearchOperator Members

        public Individual Mutate(Individual oldIndividual, Random random)
        {

            Automation aa = oldIndividual as Automation;
            bool alreadyMutate = false;

            if (aa == null)
                return oldIndividual;

            int p = 3;  // 1\(mutation probability)

            int initState = aa.InitialState;
            if (random.Next(p) == 1)
            {
                initState = random.Next(aa.StatesCount);
                alreadyMutate = true;
            }

            Automation res = new Automation(initState, aa.StatesCount, aa.ActualPredCount);

            for (int i = 0; i < Automation.PRED_COUNT; i++)
            {
                res.Predicates[i] = aa.Predicates[i];
            }

            if (random.Next(p) == 1)
            {
                int from, to;
                do
                {
                    from = random.Next(Automation.PRED_COUNT - 1);
                    to = random.Next(Automation.PRED_COUNT - 1);
                }
                while (res.Predicates[to] == res.Predicates[from]);

                res.Predicates[to] = !res.Predicates[to];
                res.Predicates[from] = !res.Predicates[from];

                alreadyMutate = true;
            }

            for (int i = 0; i < aa.InputCount; i++)
```

```
                for (int j = 0; j < aa.StatesCount; j++)
                {
                    if (random.Next(p) == 1)
                    {
                        res.Table[j, i] =
                            new Automation.Transition(random.Next(aa.StatesCount),
GetRandomAction(random));
                        alreadyMutate = true;
                    }
                    else
                        res.Table[j, i] = aa.Table[j, i];
                }


            if (!alreadyMutate)
            {
                int state = random.Next(aa.StatesCount);
                int c = random.Next(aa.InputCount - 1);
                int es = random.Next(res.StatesCount);
                res.SetTransition(state, c, new Automation.Transition(es, GetRandomAction(random)));
            }

            return res;
        }

        private Automation[] CreateAndChoosePredicates(Automation aa1, Automation aa2, Random random)
        {
            Automation[] s = new Automation[2];
            for (int i = 0; i < 2; i++)
                s[i] = new Automation(0, aa1.StatesCount, aa1.ActualPredCount);

            s[0].Predicates.SetAll(false);
            s[1].Predicates.SetAll(false);

            if (random.Next(2) == 1)
            {
                s[0].InitialState = aa2.InitialState;
                s[1].InitialState = aa1.InitialState;
            }
            else
            {
                s[0].InitialState = aa1.InitialState;
                s[1].InitialState = aa2.InitialState;
            }

            int count0 = s[0].ActualPredCount;
            int count1 = s[1].ActualPredCount;

            for (int i = 0; i < Automation.PRED_COUNT; i++)
            {
                if (aa1.Predicates[i] && aa2.Predicates[i])
                {
                    s[0].Predicates[i] = true;
                    s[1].Predicates[i] = true;
                    count0--;
                    count1--;
                }
            }

            for (int i = 0; i < Automation.PRED_COUNT; i++)
            {
                if (aa1.Predicates[i] != aa2.Predicates[i])
                {
                    if (count0 > 0 && count1 > 0)
                    {
                        if (random.Next(2) == 1)
```

```
                {
                    s[0].Predicates[i] = true;
                    count0--;
                }
                else
                {
                    s[1].Predicates[i] = true;
                    count1--;
                }
            }
            else
            {
                if (count0 > 0)
                {
                    s[0].Predicates[i] = true;
                    count0--;
                }

                if (count1 > 0)
                {
                    s[1].Predicates[i] = true;
                    count1--;
                }
            }
        }
    }

    return s;
}

public Individual[] Crossover(Individual[] individuals, Random random)
{
    Automation aa1 = individuals[0] as Automation;
    Automation aa2 = individuals[1] as Automation;

    if (aa1 == null || aa2 == null)
        return individuals;

    Automation[] s = CreateAndChoosePredicates(aa1, aa2, random);
    int crossPoint = random.Next(aa1.InputCount);

    FillChildTable(aa1, aa2, ref s[0], crossPoint, random);
    FillChildTable(aa1, aa2, ref s[1], crossPoint, random);

    return s;
}

private BitArray IntegerToBitArray(int input, int size)
{
    String str = Convert.ToString(input, 2);
    BitArray res = new BitArray(size);
    int len = str.Length;
    for (int i = len - 1; i >= 0; i--)
    {
        res[i] = (str[i] == '1');
    }
    return res;
}

private List<int> getLines(Automation parent, Automation parent2,
                           Automation child, int inputCh, int crossPoint, Boolean larger)
{
    List<int> lines = new List<int>();
    List<int> actualForChild = child.getActualPreds();
    List<int> actualForParent = parent.getActualPreds();
    List<int> actualForParent2 = parent2.getActualPreds();
```

```
            Dictionary<int, int> indexToBeTheSame = new Dictionary<int, int>();

            for (int i = 0; i < parent.ActualPredCount; i++)
                if (larger && !(actualForParent2.Contains(actualForParent[i]) && inputCh >= crossPoint)
||!larger && !(actualForParent2.Contains(actualForParent[i]) && inputCh < crossPoint))
                    if (actualForChild.Contains(actualForParent[i]))
                        indexToBeTheSame.Add(i, actualForChild.IndexOf(actualForParent[i]));


            BitArray inputChArr = IntegerToBitArray(inputCh, child.ActualPredCount);

            for (int i = 0; i < parent.InputCount; i++)
            {
                BitArray inputParArr = IntegerToBitArray(i, parent.ActualPredCount);

                Boolean flag = true;

                foreach (int j in indexToBeTheSame.Keys)
                    if (inputParArr[j] != inputChArr[indexToBeTheSame[j]])
                        flag = false;
                if (flag)
                    lines.Add(i);
            }


            return lines;
        }

        private List<float> CalculateProbability(Automation parent1, List<int> lines1)
        {
            List<float> p1 = new List<float>(parent1.StatesCount);

            for (int i = 0; i < parent1.StatesCount; i++)
            {
                p1.Add(0);
            }

            foreach (int input in lines1)
            {
                for (int k = 0; k < parent1.StatesCount; k++)
                    p1[parent1.GetTransition(k, input).EndState] += 1;
            }

            int linesSize = lines1.Count;
            int pSize = p1.Count;

            for (int i = 0; i < pSize; i++)
                p1[i] = p1[i] / (parent1.StatesCount*lines1.Count);

            return p1;
        }

        private void FillChildTable(Automation parent1, Automation parent2, ref Automation child, int
crossPoint, Random random)
        {
            for (int i = 0; i < child.InputCount; i++)
            {
                List<int> lines1 = getLines(parent1, parent2, child, i, crossPoint, true);
                List<int> lines2 = getLines(parent1, parent2, child, i, crossPoint, true);

                List<float> p1 = CalculateProbability(parent1, lines1);
                List<float> p2 = CalculateProbability(parent2, lines2);
                List<float> p = new List<float>();

                for (int j = 0, len = child.StatesCount; j < len; j++)
                {
```

```
                    p.Add(p1[j] + p2[j]);
                }

                for (int j = 0, len = child.StatesCount; j < len; j++)
                    if (p[j] != 0)
                        child.SetTransition(j, i,
                            new Automation.Transition(random.Next((int)(1/p[j]) % child.StatesCount),
GetRandomAction(random)));
                    else
                        child.SetTransition(j, i,
                            new Automation.Transition(random.Next(child.StatesCount),
GetRandomAction(random)));
            }
        }

        #endregion
    }
}
```

## 4. Файл CelluarGeneticOptimizationAlgorithm

```
#region Usings

using System.Collections.Generic;
using System.ComponentModel;
using System;
using Genetic;
using NewBrain;
using NewBrain.Attributes;
using SimpleGeneticOptimizationAlgorithm.Properties;

#endregion

namespace SimpleGeneticOptimizationAlgorithm
{
    /// <summary>
    /// Клеточный генетический алгоритм
    /// </summary>
    public class CelluarGeneticOptimizationAlgorithm : GeneticOptimizationAlgorithm
    {
        public CelluarGeneticOptimizationAlgorithm()
        {
            ElitePart = 0.1;
            MutationProbability = 0.1;
            BigMutationProbability = 0.05;
        }

        [Configurable]
        [Description("Процент элитных особей")]
        public double ElitePart { get; set; }

        [Configurable]
        [Description("Вероятность мутации")]
        public double MutationProbability { get; set; }

        [Configurable]
        [Description("Вероятность большой мутации")]
        public double BigMutationProbability { get; set; }

        public override string Title
        {
            get { return Properties.Resources.CelluarGenetic; }
        }

        public override string Description
        {
            get { return ""; }
        }

        public override string HtmlDescription
        {
            get
            {
                return Resources.simple;
            }
        }

        /// <summary>
        /// Направление оптимизации - максимизировать функцию.
        /// </summary>
        protected override OptimizationDirection OptimizationDirection
        {
            get { return OptimizationDirection.Maximize; }
        }
```

```
protected Individual[,] Torus;
protected int TorusSize;

protected override void Initialize()
{
    Random = new Random();
    Generation = new List<Individual>(GenerationSize);
    for (int i = 0; i < GenerationSize; i++)
        Generation.Add(SearchOperator.Create(Random));

    TorusSize = (int)Math.Sqrt(GenerationSize);
    Torus = new Individual[TorusSize, TorusSize];

    FillTorus();
}

private void FillTorus()
{
    Torus = new Individual[TorusSize, TorusSize];
    for (int i = 0; i < TorusSize; i++)
        for (int j = 0; j < TorusSize; j++)
        {
            Torus[i, j] = Generation[j * TorusSize + i];
        }
}

private void FillGeneration()
{
    Generation.Clear();
    for (int i = 0; i < TorusSize; i++)
        for (int j = 0; j < TorusSize; j++)
        {
            Generation[j * TorusSize + i] = Torus[i, j];
        }
}

private Individual getBestNeighbour(int i, int j)
{
    List<Individual> neighbours = new List<Individual>();

    int[] top, right, bottom, left; //top[0] = i, top[1] = j
    top = new int[2];
    right = new int[2];
    bottom = new int[2];
    left = new int[2];

    top[0] = i - 1;
    bottom[0] = i + 1;
    right[0] = i;
    left[0] = i;

    if (i == 0)
    {
        top[0] = TorusSize - 1;
    }

    if (i == TorusSize - 1)
    {
        bottom[0] = 0;
    }

    left[1] = j - 1;
    right[1] = j + 1;
    bottom[1] = j;
    top[1] = j;
```

```
    if (j == 0)
    {
        left[1] = TorusSize - 1;
    }

    if (j == TorusSize - 1)
    {
        right[1] = 0;
    }

    neighbours.Add(Torus[top[0], top[1]]);
    neighbours.Add(Torus[right[0], right[1]]);
    neighbours.Add(Torus[bottom[0], bottom[1]]);
    neighbours.Add(Torus[left[0], left[1]]);

    neighbours.Sort();

    return neighbours[neighbours.Count - 1];
}

/// <summary>
/// Генерация нового поколения: рассматривается очередной индивидуум,
/// выбирается лучшая особь среди его соседей на торе,
/// текущий индивидуум скрещивается с ней,
/// и один полученный потомок помещается в текущую ячейку вместо родителя.
/// Далее с вероятносью MutationProbability осуществляется мутация.
/// </summary>
protected override void NextGeneration()
{
    int gSize = Generation.Count;
    var newGeneration = new List<Individual>(gSize);
    int eliteSize = (int) (ElitePart*GenerationSize);

    //Generation отсортирован по убыванию.
    newGeneration.AddRange(Generation.GetRange(0, eliteSize));

    //Добавляем в новое поколение eliteSize лучших из старого
    for (int i = 0; i < TorusSize; i++)
        for (int j = 0; j < TorusSize; j++)
        {
            if (j * TorusSize + i >= eliteSize)
            {
                Individual a1 = getBestNeighbour(i, j);
                Individual a2 = Torus[i, j];
                Individual[] children = GeneticSearchOperator.Crossover(new[] {a1, a2},Random);
                newGeneration.Add(children[Random.Next(2)]);
            }
        }


    for (int i = 0; i < newGeneration.Count; i++)
    {
        if (Random.NextDouble() < MutationProbability)
            newGeneration[i] = GeneticSearchOperator.Mutate(newGeneration[i], Random);
    }

    newGeneration.Sort();
    newGeneration.Reverse();
    Generation = newGeneration;
    CurrentIndividuals = Generation;
    if (BestIndividual == null || BestIndividual.CompareTo(Generation[0]) < 0)
        BestIndividual = Generation[0];

    FillTorus();
}
```

```
        protected override void BigMutation()
        {
            lock (Generation)
            {
                for (int i = 0; i < Generation.Count; i++)
                    if (Random.NextDouble() > 1-BigMutationProbability)
                    Generation[i] = GeneticSearchOperator.Create(Random);
                Generation.Sort();
                Generation.Reverse();
                FillTorus();
            }
        }
    }
}
```