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

М.В. Костенко

Отчет по лабораторной работе «Применение генетических алгоритмов для построения конечных автоматов»

Вариант № 8

Санкт-Петербург
2009

Оглавление

Введение

В данной работе исследуется применение генетических алгоритмов для автоматического построения конечных управляющих автоматов. Рассматривается задача об «Умном муравье». С помощью генетического алгоритма генерируется конечный автомат Мура, который управляет действиями муравья.

Для эмуляций действий муравья студентами кафедры «Компьютерные технологии» СПбГУ ИТМО была создана программа «Виртуальная лаборатория обучения методам искусственного интеллекта для генерации управляющих конечных автоматов» [1]. Решением поставленной задачи является плагин для данной лаборатории, который описывает представление муравья в виде автомата Мура, а также реализует генетический алгоритм. «Лаборатория» и плагин написаны на языке программирования *C#*.

1. Постановка задачи

Построить с помощью генетических алгоритмов конечный автомат Мура, решающий задачу об «Умном муравье». Использовать представление автоматов с помощью графов переходов. Способ скрещивания выбрать самостоятельно. Использовать островной генетический алгоритм и метод «рулетки» для генерации очередного поколения.

1.1. Задача об «Умном муравье»

Игра происходит на поверхности тора размером 32x32 клетки. В некоторых клетках (обозначены на рис. 1 черным цветом) находятся яблоки. Всего на поле 89 клеток с яблоками.

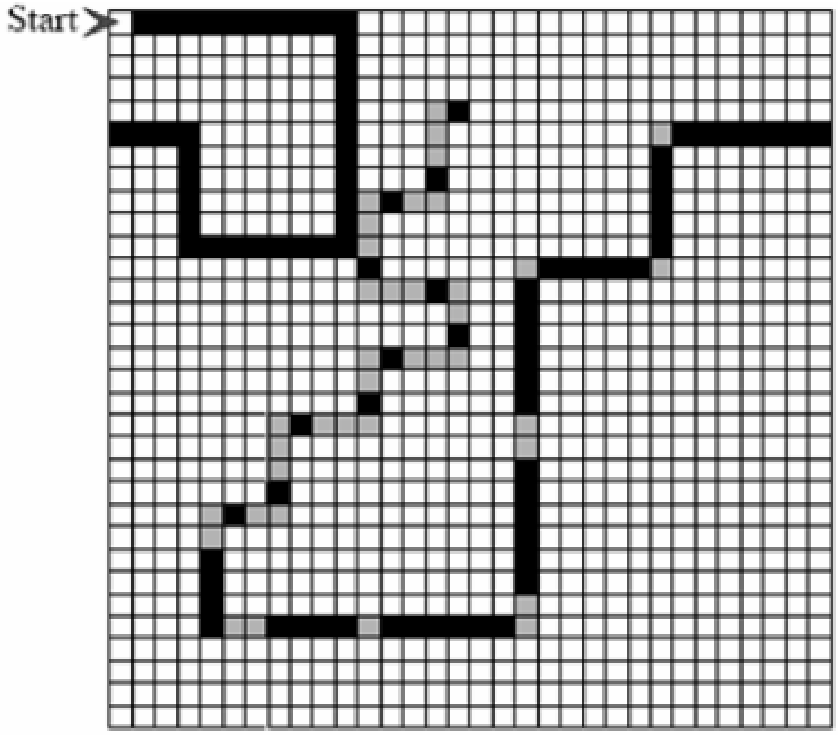

Рис. 1. Игровое поле

Муравей начинает движение из клетки, помеченной «`Start`». Он занимает одну клетку и смотрит в одном из четырех направлений (север, юг, запад, восток). В начале игры муравей смотрит на восток. За ход он может выполнить одно из следующих действий:

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

Съеденная муравьем еда не восполняется. Игра длится 200 ходов. Цель игры – создать муравья с минимальным числом состояний, который за минимальное число ходов съест как можно больше яблок.

2. Автомат Мура

Автомат Мура – конечный автомат, выходное воздействие которого зависит только от текущего состояния. Переход в новое состояние осуществляется с учетом текущего состояния и входного воздействия. Формально автомат Мура представляется совокупностью:

$$A = (S, S_0, X, Y, \delta, \mu),$$

где

- S – множество состояний;
- S_0 – начальное состояние;
- X – множество входных воздействий;
- Y – множество выходных воздействий;
- $\delta : S \times X \rightarrow S$ – таблица переходов;
- $\mu : S \rightarrow Y$ – выходные воздействия для каждого состояния.

2.1. Способ представления автомата Мура

В программе автомат Мура представлен классом `MooreMachine`, содержащим массив состояний `State`. Каждое состояние хранит действие и таблицу переходов в другие состояния.

```
class MooreMachine : Automation
    {
        public class State
          {
          public int[] transitions;
          public SimpleAntProblem.AntActions _action;
          }
        public State[] states;
    }
```

3. Генетический алгоритм

3.1. Островной генетический алгоритм

Особенностью островного генетического алгоритма является расселение всей популяции на несколько островов, на каждом из которых эволюция происходит независимо от других. Раз в несколько поколений происходит обмен особями между островами. Более подробно про островные генетические алгоритмы можно узнать в работе [2].

3.2. Метод «рулетки»

Метод «рулетки» выбирает из `n` особей одну следующим образом: отрезок [0,1] разбивается на n отрезков. Длина каждого отрезка пропорциональна функции приспособленности соответствующей особи. Затем на отрезок случайным образом ставится точка. Она попадает на один из `n` отрезков. Выбирается особь, соответствующая данному отрезку.

3.3. Скрещивание

Оператор скрещивания получает на вход две особи и возвращает тоже две особи. Скрещивание производится следующим образом. Стартовое состояние каждого ребенка полагается равным стартовому состоянию одного из родителей. Далее, для каждого состояния независимо выбирается родитель, от которого наследуются действие и таблица переходов. Выбор особей для скрещивания производится методом «рулетки».

3.4. Мутация

Оператор мутации получает на вход одну особь и возвращает тоже одну особь. Мутация происходит одним из четырех равновероятных способов:

- заменяется начальное состояние;
- заменяется действие в случайном состоянии;
- в случайном состоянии меняются местами направления переходов;
- в случайном состоянии заменяется направление одного из переходов.

Выбор особей для мутации производится методом «рулетки».

3.5. Генерация очередного поколения внутри острова

В следующее поколение добавляется `ElitePart` лучших особей из предыдущего поколения. Затем добавляется `MutatedPart` особей, полученных в результате мутации. Еще `NewPart` особей создаются случайным образом, а оставшиеся особи получаются путем скрещивания.

3.6. Генерация очередного поколения

Генерация очередного поколения происходит следующим образом. Для каждого острова определяется, произойдет ли «большая мутация». Если она происходит, то весь остров заселяется особями, созданными случайным образом. Если же «большая мутация» не происходит, то генерируется очередное поколение внутри острова.

После обработки всех островов определяется необходимость миграции. Если миграция необходима, то `MovedPart` особей с каждого острова, выбранных методом «рулетки», переселяются на соседний с ним остров.

3.7. Настраиваемые параметры алгоритма

Следующие параметры алгоритма могут быть изменены:

- `ElitePart` – часть лучших особей, переходящая в следующее поколение;
- `MutatedPart` – часть особей, получаемая мутацией;
- `NewPart` – часть особей, создаваемая случайным образом;
- `BigMutationProbability` – вероятность «большой мутации»;
- `MovedPart` – часть особей, перемещаемая с острова на остров во время миграции;
- `MovePeriod` – период миграций;
- `IslandPopulation` – население каждого острова;
- `IslandCount` – число островов.

4. Результаты работы модуля

В результате запуска модуля был получен автомат Мура с шестью состояниями. Муравей, управляемый данным автоматом, съедает 79 яблок за 200 ходов.

4.1. Граф переходов построенного автомата

На рис 2. изображен граф переходов построенного автомата. Стрелка в начальное состояние отмечена подписью «`Start`». Действие, совершаемое при попадании в состояние, обозначено в круге под номером состояния (`Forward` – шаг вперед, `Right` – поворот направо, `Left` – поворот налево). Условия переходов обозначены около стрелок переходов (`Yes` – перед муравьем есть еда, `No` – перед муравьем нет еды).

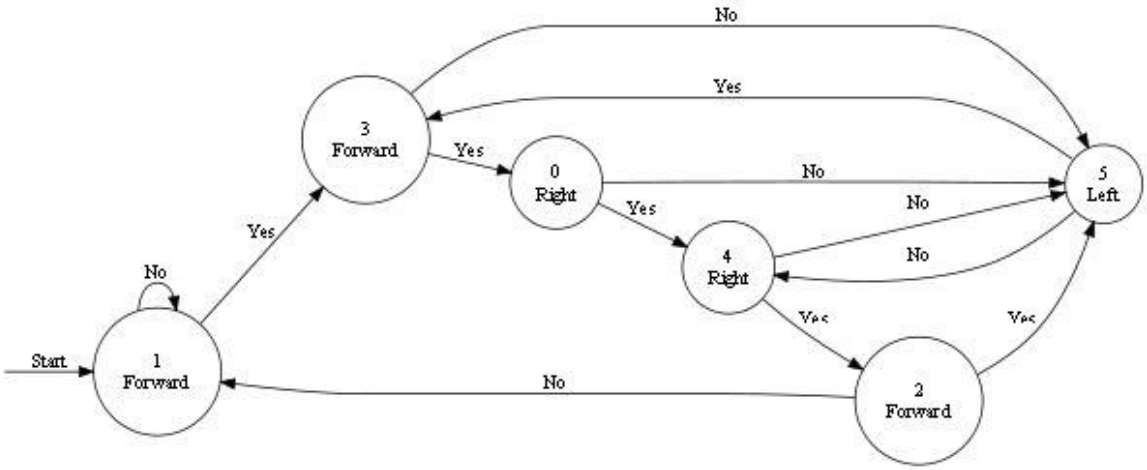

Рис. 2. Построенный автомат Мура

4.2. График максимального значения функции приспособленности

На рис. 3. изображен график зависимости максимального значения функции приспособленности от времени для запуска модуля, при котором был получен указанный автомат Мура. Время указано в секундах.

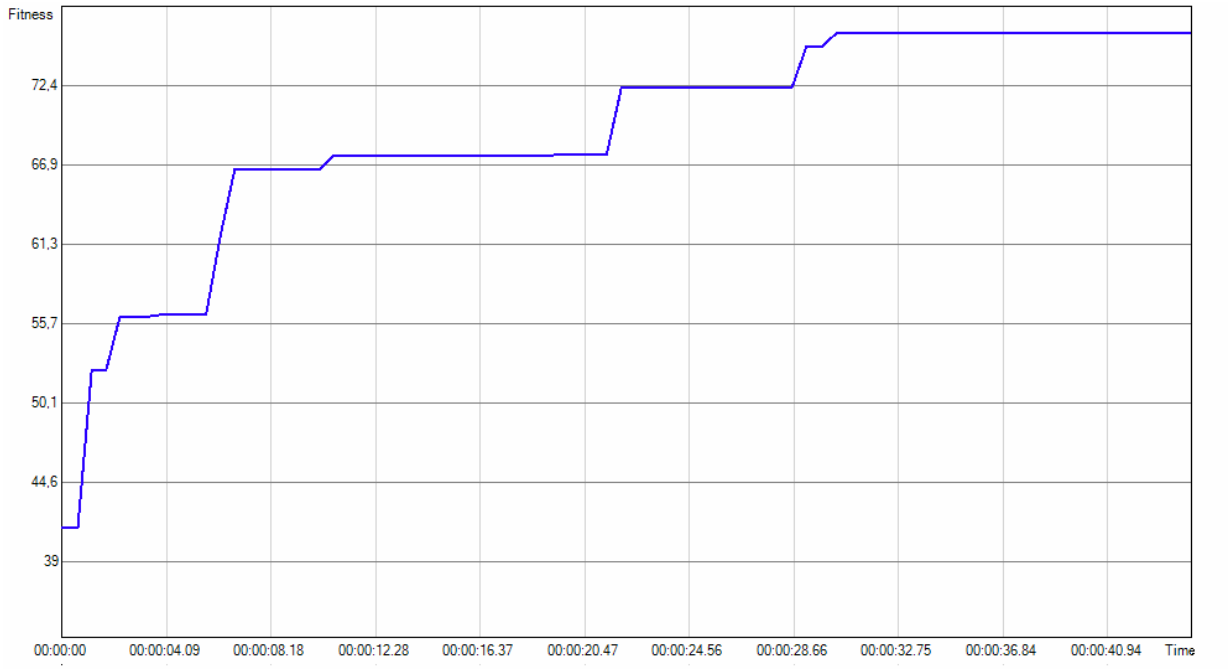

Рис. 3. График зависимости максимального значения функции приспособленности от времени

Заключение

Результаты лабораторной работы показали, что с помощью островного генетического алгоритма можно получить автомат Мура с шестью состояниями, который эффективно решает задачу об «Умном муравье». Получение автоматов с большим числом состояний затруднено, так как в процессе работы алгоритма число состояний у особи может только уменьшиться (при появлении изолированного состояния).

Источники

1. *Виртуальная лаборатория обучения методам искусственного интеллекта для генерации управляющих конечных автоматов (язык программирования C#)* http://is.ifmo.ru/courses/_giopt-src.rar
2. *Яминов Б.* Генетические алгоритмы http://rain.ifmo.ru/cat/view.php/theory/unsorted/genetic-2005

Приложение

Исходные тексты программ

Файл *IslandGeneticSearchAlgorithm.cs*

```
// Miron Kostenko 3539. November 2009.

using System;
using System.IO;
using System.Collections.Generic;
using System.ComponentModel;
using Genetic.Operators;
using NewBrain;
using NewBrain.Attributes;
using NewBrain.Plugins;

namespace IslandGeneticAlgorithm
{
    /// <summary>
    /// Островной генетический алгоритм, строящий автомат Мура и использующий
метод «рулетки».
    /// </summary>
    [SearchOperatorType(typeof(IGeneticSearchOperator))]
    public class IslandGeneticAlgorithm : Algorithm
    {

        public IslandGeneticAlgorithm()
        {
            ElitePart = 0.1;
            NewPart = 0.05;
            MutatedPart = 0.1;
            IslandCount = 10;
            IslandPopulation = 50;
            MovedPart = 0.1;
            MovePeriod = 10;
            BigMutationProbability = 0.01;
        }

        [Configurable]
        [Description("Период миграций")]
        public int MovePeriod { get; set; }

        [Configurable]
        [Description("Население острова")]
        public int IslandPopulation { get; set; }

        [Configurable]
        [Description("Количество островов")]
        public int IslandCount { get; set; }

        [Configurable]
        [Description("Процент переселяемых лучших особей")]
        public double MovedPart { get; set; }


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

```
        [Configurable]
        [Description("Процент мутирующих особей")]
        public double MutatedPart { get; set; }

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

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

        /// <summary>
        /// Генератор случайных чисел. Используется при создании индивидуума,
        /// мутации, скрещивании, а так же методом «рулетки».
        /// </summary>
        protected Random Random { get; private set; }

        /// <summary>
        /// Номер текущего поколения.
        /// </summary>
        public int GenerationNumber { get; private set; }

        /// <summary>
        /// Массив популяций островов.
        /// </summary>
        protected List<Individual>[] Islands { get; set; }

        /// <summary>
        /// Возвращает название, используемое лабораторией.
        /// </summary>
        public override string Title
        {
            get { return "Островной генетический алгоритм"; }
        }

        /// <summary>
        /// Возвращает описание, используемое лабораторией. Не реализовано.
        /// </summary>
        public override string Description
        {
            get { return ""; }
        }

        /// <summary>
        /// Возвращает описание в формате html, используемое лабораторией. Не
реализовано.
        /// </summary>
        public override string HtmlDescription
        {
            get
            {
                return "";
            }
        }

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

```
        /// <summary>
        /// В качестве SearchOperator используется GeneticSearchOperator.
        /// </summary>
        protected IGeneticSearchOperator GeneticSearchOperator
        {
           get { return SearchOperator as IGeneticSearchOperator; }
        }

        private int population;

        /// <summary>
        /// Инициализация начального состояния генетического алгоритма.
        /// Создается первое поколение особей, полученных случайным образом.
        /// </summary>
        protected override void Initialize()
        {
            Random = new Random();
            CurrentIndividuals  = new
List<Individual>(IslandPopulation*IslandCount);
            population = IslandPopulation * IslandCount;
            Islands = new List<Individual>[IslandCount];

            for (int i = 0; i < IslandCount; i++)
            {
                Islands[i] = new List<Individual>(IslandPopulation);
                for (int j = 0; j < IslandPopulation; j++)
                    Islands[i].Add(SearchOperator.Create(Random));
                CurrentIndividuals.AddRange(Islands[i]);
            }
        }

        /// <summary>
        /// Выполняет миграцию части особей с каждого острова на соседние.
        /// </summary>
        /// <param name="rand">Генератор случайных чисел.</param>
        private void migrateAll(Random rand)
        {
            for (int i = 0; i < IslandCount; i++)
            {
                Roulette roulette = new Roulette(Islands[0]);
                var moveCount = (int)(MovedPart * IslandPopulation);
                int next = (i+1)%IslandCount;
                var newIsland = new List<Individual>(Islands[next]);
                for (int j = IslandPopulation - moveCount-1; j <
IslandPopulation; j++)
                    newIsland[j] = roulette.getIndividual(rand);
                Islands[next] = newIsland;
            }
        }

        /// <summary>
        /// Очередная итерация генетического алгоритма.
        /// Происходит переход к следующему поколению особей, при необходимости
        /// производится миграция.
        /// </summary>
        protected override void NextIteration()
        {
            var newCurrentIndividuals = new List<Individual>(population);
            GenerationNumber++;
            for (int i = 0; i < IslandCount; i++)
            {
                if (Random.NextDouble() < BigMutationProbability)
                    Islands[i] = bigMutation(IslandPopulation);
```

```
                else
                    Islands[i] = NextIslandGeneration(i);
                newCurrentIndividuals.AddRange(Islands[i]);
            }
            CurrentIndividuals = newCurrentIndividuals;

            if (GenerationNumber > MovePeriod)
                migrateAll(Random);

            BestIndividual = CurrentIndividuals[0];
            if (BestIndividual.Fitness > 77.0)
            {
                StreamWriter sw = File.AppendText("smartAnt.log");
                sw.WriteLine(BestIndividual.ToString());
                sw.Close();
            }

            CurrentIndividuals.Sort();
            CurrentIndividuals.Reverse();
            BestIndividual = CurrentIndividuals[0];
        }

        /// <summary>
        /// Выполняет большую мутацию для одного острова.
        /// </summary>
        /// <param name="population">Население острова.</param>
        private List<Individual> bigMutation(int population)
        {
            List<Individual> newIsland = new List<Individual>(population);
                for (int j = 0; j < population; j++)
                    newIsland.Add(SearchOperator.Create(Random));
            return newIsland;
        }


        /// <summary>
        /// Класс, реализующий метод «рулетки».
        /// </summary>
        private class Roulette
        {

            private double[,] rouletteBounds;
            private double totalFitness;
            private double currentUpperBorder;
            private List<Individual> list;

            /// <summary>
            /// Создает экземпляр Roulette. Выбор особей будет осуществляться из
копии массива generation.
            /// </summary>
            /// <param name="generation">Массив особей, из которых будет
производиться выбор.</param>
            public Roulette(List<Individual> generation)
            {
                var size = generation.Count;

                list = new List<Individual>(generation);
                rouletteBounds = new double[2, size];
                totalFitness = 0;
                currentUpperBorder = 0;
                for (int i = 0; i < size; i++) totalFitness += list[i].Fitness;
                for (int i = 0; i < size; i++)
                {
                    rouletteBounds[0, i] = currentUpperBorder;
```

```
                    rouletteBounds[1, i] = (currentUpperBorder += list[i].Fitness
/ totalFitness);
                }
            }

            /// <summary>
            /// Возвращает особь, выбранную из исходных методом «рулетки».
            /// Вероятность быть выбранной прямо пропорциональна функции
приспособленности.
            /// </summary>
            /// <param name="rand">Генератор случайных чисел.</param>
            /// <returns>Индивидуум, выбранный методом «рулетки».</returns>
            public Individual getIndividual(Random rand)
            {
                var rnd = rand.NextDouble();
                var res = 0;
                for (int j = 0; j < rouletteBounds.GetLength(1); j++)
                    if (rnd > rouletteBounds[0, j] + 10e-6 && rnd <
rouletteBounds[1, j] + 10e-6)
                    {
                        res = j;
                        break;
                    }
                return list[res];
            }

            /// <summary>
            /// Возвращает массив особей требуемого размера, получаемый
многократными
            /// вызовами getIndividual(rand). Особи могут повторяться.
            /// </summary>
            /// <param name="count">Желаемое количество особей.</param>
            /// <param name="rand">Генератор случайных чисел.</param>
            /// <returns>Массив особей, выбранных методом «рулетки».</returns>
            public List<Individual> getRange(int count, Random rand)
            {
                var res = new List<Individual>(count);
                for (int i = 0; i < count; i++)
                    res.Add(getIndividual(rand));
                return res;
            }

        };

        /// <summary>
        /// Пересчет поколения для одного острова. Используется метод «рулетки».
        /// </summary>
        /// <param name="n">Номер острова, для которого пересчитывается
поколение.</param>
        private List<Individual> NextIslandGeneration(int n)
        {
            int size = Islands[n].Count;
            var newIslandGeneration = new List<Individual>(size);

            int eliteCount = (int)(size * ElitePart);
            newIslandGeneration.AddRange(Islands[n].GetRange(0, eliteCount));


            int newCount = (int)(size * NewPart);
            for (int i = 0; i < newCount; i++)
                newIslandGeneration.Add(SearchOperator.Create(Random));

            int mutatedCount = (int)(size * MutatedPart);
            for (int i = 0; i < mutatedCount; i++)
```

```
newIslandGeneration.Add(GeneticSearchOperator.Mutate(Islands[n][Random.Next(size)
],Random));

            int crossoverCount = (size - eliteCount - newCount - mutatedCount-1);

            Roulette roulette = new Roulette(Islands[n]);

            for (int i = 0; i < crossoverCount; i += 2)
            {
                Individual a1 = roulette.getIndividual(Random);
                Individual a2 = roulette.getIndividual(Random);
                Individual[] s = GeneticSearchOperator.Crossover(new[] { a1, a2
}, Random);
                newIslandGeneration.AddRange(s);
            }

            if (newIslandGeneration.Count < size)
                newIslandGeneration.Add(SearchOperator.Create(Random));

            if (newIslandGeneration.Count != Islands[n].Count)
                throw new Exception(newIslandGeneration.Count +"!="+
Islands[n].Count);

            newIslandGeneration.Sort();
            newIslandGeneration.Reverse();

            return newIslandGeneration;
        }
    }
}
```

Файл *MooreMachine.cs*

```
// Miron Kostenko 3539. November 2009.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NewBrain;
using ArtificialAnt.IndividualInfo;
using ArtificialAnt.ProblemInfo;

namespace IslandGeneticAlgorithm
{
    /// <summary>
    /// Автомат Мура для задачи об "умном муравье".
    /// В связи с особенностями среды пришлось наследовать автомат Мура от
автомата Мили (класс Automation).
    /// Соответственно, необходимо сделать метод GetTransition в Automation
сделать виртуальным.
    /// </summary>
    class MooreMachine : Automation
    {

        #region State class
        /// <summary>
        /// Внутренний класс для представления состояния.
        /// </summary>
        public class State
        {
```

```
            public int[] transitions;
            public SimpleAntProblem.AntActions _action;

            /// <summary>
            /// Метод, возвращающий действие, соответствующее данному состоянию.
            /// </summary>
            /// <returns>Действие.</returns>
            public SimpleAntProblem.AntActions getAction() { return _action; }

            /// <summary>
            /// Метод, возвращающий состояние, в которое переходит муравей в
случае отсутствия перед ним еды.
            /// </summary>
            /// <returns>Номер состояния.</returns>
            public int getIfYes() { return transitions[0]; }

            /// <summary>
            /// Метод, возвращающий состояние, в которое переходит муравей в
случае наличия перед ним еды.
            /// </summary>
            /// <returns>Номер состояния.</returns>
            public int getIfNo() { return transitions[1]; }

            public State(int ifYes, int ifNo, SimpleAntProblem.AntActions action)
            {
                transitions = new int[2];
                transitions[0] = ifYes;
                transitions[1] = ifNo;
                _action = action;
            }

            /// <summary>
            /// Копирующий конструктор для класса State.
            /// </summary>
            /// <param name="other">Объект, копия которого создается.</param>
            public State(State other)
            {
                transitions = (int[])other.transitions.Clone();
                _action = other._action;
            }

        }
        #endregion

        /// <summary>
        /// Массив состояний.
        /// </summary>
        public State[] states;

        /// <summary>
        /// Создает новый экземпляр MooreMachine.
        /// </summary>
        /// <param name="initialState">Номер начального состояния.</param>
        /// <param name="statesCount">Количество состояний.</param>
        public MooreMachine(int initialState, int statesCount)
        {
            InitialState = initialState;
            StatesCount = statesCount;
            states = new State[StatesCount];
        }

        /// <summary>
        /// Копирующий конструктор для класса MooreMachine.
        /// </summary>
```

```
        /// <param name="other">Объект, копия которого создается.</param>
        public MooreMachine(MooreMachine other)
        {
            InitialState = other.InitialState;
            StatesCount = other.StatesCount;
            states = (State[])other.states.Clone();
        }

        /// <summary>
        /// Возвращает переход из состояния current при условии food.
        /// </summary>
        /// <param name="current">Номер текущего состояния.</param>
        /// <param name="food">food == 0 => перед муравьем есть еда, food == 1 =>
еды перед ним нету.</param>
        /// <returns></returns>
        public override Transition GetTransition(int current, int food)
        {
            return new Transition(states[current].transitions[food],
states[current]._action);
        }

        /// <summary>
        /// Сохраняет автомат в виде строки.
        /// </summary>
        /// <returns>Строка, являющяяся текстовым представлением
автомата.</returns>
        public override String ToString()
        {
            String result = "Moore machine with fitness " + Fitness + "\n";
            result += "states: " + StatesCount + "; initial: " +
InitialState+"\n";
            for (int i = 0; i < StatesCount; i++)
            {
                result += "state#" + i + " action: " + states[i].getAction() + ";
ifYes: " + states[i].getIfYes() + "; ifNo: " + states[i].getIfNo() + "\n";
            }
            return result;
        }
    }

    /// <summary>
    /// Класс-расширение. Добавляет к классу Individual метод toAutomation.
    /// </summary>
    public static class Extention
    {
        /// <summary>
        /// Возвращает, при возможности представление Individual, являющегося
MooreMachine, в виде Automation.
        /// </summary>
        /// <param name="indiv">Явно передаваемый указатель на себя.</param>
        /// <returns>Объект Automation.</returns>
        public static Automation toAutomation(this Individual indiv)
        {
            MooreMachine ind = indiv as MooreMachine;
            if (ind == null) return null;
            Automation res = new Automation(ind.InitialState, ind.StatesCount);
            for (int i = 0; i < ind.StatesCount; i++)
            {
                MooreMachine.State st = ind.states[i];
                res.SetTransition(i,0,new
Automation.Transition(st.getIfNo(),st.getAction()));
                res.SetTransition(i, 1,new Automation.Transition(st.getIfYes(),
st.getAction()));
            }
```

```
            return res;
        }
    }
}
```

Файл *SearchOperatorOnMooreMachine.cs*

```
// Miron Kostenko 3539. November 2009.

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.IO;
using Genetic.Operators;
using ArtificialAnt.IndividualInfo;
using ArtificialAnt.ProblemInfo;
using NewBrain;
using NewBrain.Attributes;

namespace IslandGeneticAlgorithm
{
    /// <summary>
    /// Класс, инкапсулирующий работу с особями.
    /// </summary>
    [IndividualType(typeof (Automation))]
    public abstract class SearchOperatorOnMooreMachine : SearchOperator
    {
        public SearchOperatorOnMooreMachine()
        {
            StatesCount = 10;
        }

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

        /// <summary>
        /// Возвращает одно из трех действий, выбранное случайным образом.
        /// </summary>
        /// <param name="random">Генератор случайных чисел.</param>
        /// <returns>Случайное действие.</returns>
        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;
        }
```

```
        /// <summary>
        /// Случайным образом создает особь.
        /// </summary>
        /// <param name="random">Генератор случайных чисел.</param>
        /// <returns>Особь с StatesCount состояниями и случайно выбранными
действиями и переходами.</returns>
        public override Individual Create(Random random)
        {
            MooreMachine machine = new MooreMachine(0, StatesCount);
            for (int i = 0; i < StatesCount; i++)
                machine.states[i] = new
MooreMachine.State(random.Next(StatesCount), random.Next(StatesCount),
GetRandomAction(random));
            return machine;
        }
    }

    /// <summary>
    /// Класс, инкапсулирующий работу с особями (автоматами Мура).
    /// </summary>
    [IndividualType(typeof(Automation))]
    public class GeneticSearchOperatorOnMooreMachine :
SearchOperatorOnMooreMachine, IGeneticSearchOperator
    {
        /// <summary>
        /// Возвращает название, используемое лабораторией.
        /// </summary>
        public override string Title
        {
            get { return "Генетический оператор для автоматов Мура."; }
        }

        /// <summary>
        /// Возвращает описание, используемое лабораторией.
        /// Не реализовано.
        /// </summary>
        public override string Description
        {
            get { return ""; }
        }

        /// <summary>
        /// Производит мутацию особи.
        /// </summary>
        /// <param name="oldIndividual">Исходная особь.</param>
        /// <param name="random">Генератор случайных чисел.</param>
        /// <returns>Получившаяся в результате мутации особь.</returns>
        public Individual Mutate(Individual oldIndividual, Random random)
        {
            MooreMachine old = oldIndividual as MooreMachine;
            if (old == null) return oldIndividual;

            MooreMachine res = new MooreMachine(old);

            switch (random.Next(4))
            {
                case 0:
                    // Изменяем начальное состояние.
                    res.InitialState = random.Next(res.StatesCount);
                    break;
                case 1:
                    // В случайном состоянии изменяем внутреннее действие.
                    int stateNum = random.Next(res.StatesCount);
```

```
                    res.states[stateNum]._action = GetRandomAction(random);
                    break;
                case 2:
                    // В случайном состоянии меняем местами направления
переходов.
                    // (переход по "Нет еды"  --> переход по "Есть еда").
                    int stateNum2 = random.Next(res.StatesCount);
                    int tmpDir = res.states[stateNum2].transitions[0];
                    res.states[stateNum2].transitions[0] =
res.states[stateNum2].transitions[1];
                    res.states[stateNum2].transitions[1] = tmpDir;
                    break;
                case 3:
                    // В случайном состоянии меняем направление случайного
перехода на случайное.
                    int stateNum3 = random.Next(res.StatesCount);
                    int transNum = random.Next(2);
                    res.states[stateNum3].transitions[transNum] =
random.Next(StatesCount);
                    break;
            }
            return res;
        }

        /// <summary>
        /// Производит скрещивание особей.
        /// </summary>
        /// <param name="individuals">Массив, содержащий 2 особи.</param>
        /// <param name="random">Генератор случайных чисел.</param>
        /// <returns>Массив, содержащий результат скрещивания --- 2
особи.</returns>
        public Individual[] Crossover(Individual[] individuals, Random random)
        {

            MooreMachine[] p = new MooreMachine[individuals.Count()];
            for (int i = 0; i < individuals.Count(); i++)
            {
                p[i] = individuals[i] as MooreMachine;
                if (p[i] == null) return individuals;
            }

            MooreMachine[] res = new MooreMachine[2];

            int rnd = random.Next(2);
            res[0] = new MooreMachine(rnd == 1 ? p[0].InitialState :
p[1].InitialState, p[0].StatesCount);
            res[1] = new MooreMachine(rnd == 1 ? p[1].InitialState :
p[0].InitialState, p[0].StatesCount);

            for (int i = 0; i < p[0].StatesCount; i++)
            {
                int takeActionFrom = random.Next(2);
                int takeIfYesFrom = random.Next(2);
                int takeIfNoFrom = random.Next(2);
                res[0].states[i] = new
MooreMachine.State(p[takeIfYesFrom].states[i].getIfYes(),

p[takeIfNoFrom].states[i].getIfNo(),

p[takeActionFrom].states[i].getAction());
                res[1].states[i] = new MooreMachine.State(p[(3-
takeIfYesFrom)%2].states[i].getIfYes(),
                                                          p[(3-
takeIfNoFrom)%2].states[i].getIfNo(),
```

```
                                                                p[(3-
takeActionFrom)%2].states[i].getAction());
            }

            return res;
        }
    }
}
```