Министерство образования и науки Российской Федерации
Санкт-Петербургский государственный университет информационных технологий, механики и оптики
Кафедра компьютерных технологий

Тяхти А. С., Чебатуркин А. А.

Программный комплекс для изучения методов искусственного интеллекта «Виртуальная лаборатория «GlOpt»

Научный руководитель – Царев Ф. Н.

Санкт-Петербург
2010 г.

Оглавление

ГЛАВА 1. ВОЗМОЖНОСТИ ВИРТУАЛЬНОЙ ЛАБОРАТОРИИ

Лаборатория реализована на языке C# под платформой Microsoft.NET.

В виртуальной лаборатории существует пять типов плагинов:

- рассматриваемые задачи (наследуются от абстрактного класса Problem);
- алгоритмы и методы искусственного интеллекта (наследуются от абстрактного класса Algorithm);
- методы, адаптирующие алгоритмы к конкретным задачам (наследуются от абстрактного класса SearchOperator);
- визуализаторы для задач;
- текстовые описания задач и алгоритмов.

Каждый из первых четырех типов плагинов представляет собой dll-библиотеку, описания задач и алгоритмов оформляются в формате html. Особо отметим, что виртуальная лаборатория спроектирована таким образом, что любую задачу можно решать, используя любой из реализованных методов искусственного интеллекта, для этого необходимо определить соответствующий «адаптер», наследуемый от абстрактного класса SearchOperator. Основным назначением плагинов, прежде всего, является предоставление возможности реализовывать новые задачи и методы оптимизации в ходе обучения генетическому программированию.

В настоящее время программный комплекс виртуальной лаборатории включает в себя следующие плагины.

- Задачи:
  - об «умном муравье»;
  - об «умном муравье - 3»;
  - о роботе, огибающем препятствия;
  - о расстановке N ферзей на шахматной доске.
- Визуализаторы для указанных задач.
- Графики, позволяющие оценивать эффективность методов применительно к конкретным задачам.
- HTML-описания решаемых задач и используемых для их решения методов.

Для наглядности проводимых сравнений методов в виртуальную лабораторию была добавлена поддержка одновременного запуска нескольких методов, решающих одну и ту же задачу. Результат работы алгоритмов можно наблюдать на одном сводном графике.

ГЛАВА 2. СТРУКТУРА ВИРТУАЛЬНОЙ ЛАБОРАТОРИИ

Ядро программной среды представлено классом `Brain`, который ответственен за загрузку основных сущностей программного комплекса: задач, методов искусственного интеллекта и визуализаторов, а также выполняет функции распределения ресурсов между работающими алгоритмами.

Для описания задач используются абстрактные классы `Problem` и `Individual`, от которых наследуются классы, описывающие конкретные задачи, например, задачу об «умном муравье». Алгоритмы решения описываются классами `Algorithm` и `SearchOperator`, от которых в свою очередь наследуются классы, реализующие конкретные методы искусственного интеллекта. Общая структура виртуальной лаборатории приведена на рис. 8.

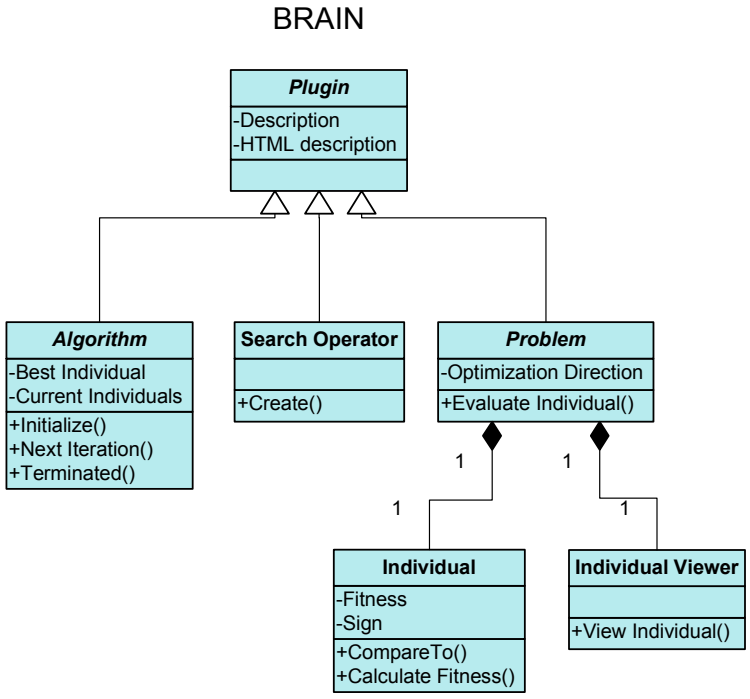

Рис. 8. Основные элементы структуры виртуальной лаборатории `GlOpt`: класс `Brain`.

2.1 Класс Problem

Абстрактный класс `Problem` содержит метод `OptimizationDirection`, возвращающий информацию о направления оптимизации (`min, max`) и метод `EvaluateIndividual`, служащий для вычисления целевой fitness-функции у конкретной особи. Наследники этого класса должны реализовывать эти методы, а также

методы, обеспечивающие доступ к базовой информации о задаче: ее названии и кратком описании.

```
public abstract class Problem : Plugin
{
      public abstract OptimizationDirection OptimizationDirection { get; }
      public abstract double EvaluateIndividual(Individual individual);
}
```

2.2 Класс Individual

Абстрактный класс `Individual` реализует представление индивидуальной для каждой задачи оценочной характеристики `Fitness`, описывает метод сравнения индивидов между собой.

```
public abstract class Individual : IComparable<Individual>
{
    public double Fitness
    { …  }
    public int CompareTo(Individual other)
    {
        return Fitness.CompareTo(other.Fitness);
    }
}
```

Класс `Individual` реализует работу со значениями функции приспособленности с учетом направления оптимизации конкретной задачи – максимизации или минимизации. Это обеспечивает независимость реализуемых алгоритмов оптимизации от конкретных условий той или иной задачи.

2.3 Класс SearchOperator

Класс `SearchOperator` необходим для организации требуемых в задаче операций над объектами класса `Individual`. Он обеспечивает универсальность в применении методов решения к различным задачам. Так, например, для применения уже реализованного алгоритма к задаче об «умном муравье» требуется определить методы мутации и скрещивания индивидов, а для метода имитации отжига – изменение уже найденного решения в соответствии с выбранным вероятностным распределением.

Классы наследники должны определять следующие методы.

- `Create` – создание новой особи.

- `Modify` – изменение особи каким-либо образом (например, случайно), с учетом интерфейса, который требует `Algorithm`.

2.4 Класс Algorithm

Класс `Algorithm` необходим для реализации конкретного метода решения.

Содержит следующие методы:

- `OptimizationDirection` – направление оптимизации, характеризующее сам алгоритм (направление по умолчанию).
- `SearchOperator` – набор методов для работы с особями, которые использует алгоритм.
- `Initialize` – установка необходимых для работы алгоритма параметров.
- `NextIteration` – метод, описывающий шаг итерации алгоритма.
- `Terminated` – метод, позволяющий определить - закончил ли алгоритм работу. Если метод возвращает true, то алгоритм автоматически перезапускается.

```
public abstract class Algorithm : Plugin
{
      protected internal abstract OptimizationDirection OptimizationDirection
{ get; }
      protected internal SearchOperator SearchOperator {get; internal set; }
      protected internal virtual List<Individual> CurrentIndividuals { get;
                                                              protected set; }
      protected internal virtual Individual BestIndividual { get; set; }
      protected internal virtual void Initialize(){}
      protected internal virtual void NextIteration(){}
      protected internal virtual bool Terminated() {}
}
```

2.5 Интерфейс IndividualVeiwer

Данный интерфейс необходим для создания визуализаторов, наглядно отображающих решения конкретной задачи. Для визуального отображения решения также должен быть подготовлен необходимый набор форм.

```
public interface IIndividualViewer
{
     void ViewIndividual(Individual individual);
}
```

ГЛАВА 3. ТЕХНОЛОГИЯ РАЗРАБОТКИ МОДУЛЕЙ GLOPT

В программную среду закладывалась возможность добавления новых алгоритмов и рассматриваемых задач как самими авторами на различных этапах разработки, так и другими студентами с помощью отдельных подключения модулей.

Проектирование модулей предполагает изучение структуры уже реализованных методов оптимизации, и построении на их основе собственных. Ниже приведены базовые рекомендации, которым необходимо следовать при разработке собственных модулей, определяющих алгоритм решения. В приложении 1 содержатся примеры реализации основых типов подключаемых модулей, в соответствии с которыми можно оценить общую трудоемкость реализации того или иного плагина.

Выполнение лабораторной предполагает создание собственного проекта в рамках существующего solution лаборатории.

1. Реализовать алгоритм поиска оптимального решения – класс, наследованный от абстрактного класса `Algorithm`. Данный класс должен определять следующие методы.
    - `Title` – возвращает стоку с названием реализуемого алгоритма.
    - `Description` – возвращает строку с кратким описанием алгоритма.
    - `OptimizationDirection` – возвращает одну из двух именованных констант: `OptimizationDirection.Minimize` или `OptimizationDirection.Maximize`.
    - `Initialize` – описывает начальное состояние в алгоритме поиска решения.
    - `NextIteration` – описывает действия, происходящие на очередной итерации алгоритма.
    - В переменной `BestIndividual` типа `Individual` должна содержаться актуальная информация об объекте типа `Individual` с лучшей на данной итерации алгоритма целевой функцией.

   Методы `Title` и `Description` в свою очередь являются абстрактными методами класса `Plugin`, от которого наследуется класс `Algorithm`.
2. Реализовать интерфейс `SearchOperator`, наследованный от интерфейса `ICreateOperator`. В классе, реализующем данный интерфейсе должен быть реализован метод `Create`, возвращающий объект типа Individual (абстрактный класс, определяется для конкретной задачи). Также, в нем реализуются все

необходимые методы интерфейса `SearchOperator` для работы с объектами `Individual` – например, операции мутации и кроссовера.

3. Необходимо удостовериться, что библиотека `*.dll` откомпилированного модуля находится в папке `plugins` проекта `Framework`.

Для реализации новой задачи в комплексе GlOpt необходимо сделать следующее.

1. Реализовать класс - наследник абстрактного класса `Individual`, определяющего объект к которому в дальнейшем применяется алгоритм оптимизации. Так, в задачи о ферзях таким объектом выступает шахматная доска с расставленными на ней ферзями.
2. Реализовать класс – наследник абстрактного класса `Problem`. В данном классе должны быть реализованы методы `Title`, `Description`, `OptimizationDirection`, аналогичные методам, описанным в разделе рекомендаций по реализации алгоритма. Также необходимо реализовать метод `EvaluateIndividual`, возвращающий значение типа `double` – значение целевой функции для данного индивида.
3. Для реализации компоненты визуализации текущего решения требуется определить класс `Viewer`, наследуемый от класса `IndividualViewer`, и в частности определить метод `ViewIndividual`, вызывающий графическую форму, либо представляющий данные о решении в любом другом удобном для пользователя виде.
4. Необходимо удостовериться что библиотека *.dll откомпилированного модуля находится в папке `plugins` проекта `Framework`.

ПРИЛОЖЕНИЕ. ПРИМЕР РЕАЛИЗАЦИИ ПОДКЮЧАЕМЫХ МОДУЛЕЙ ЛАБОРАТОРИИ

1. Файл `SimpleAntProblem.cs`

Содержит пример класса, наследованного от абстрактного класса Problem. В примере приведена реализация задачи об «умном муравье».

```
#region Usings

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

#endregion

namespace ArtificialAnt.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,
            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.ArtAntProblem; }
        }
```

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

        /// <summary>
        /// Вычисление fitness-функции текущей особи - количества 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();
                  }
            }
      }
}
```

2. Файл `Automation.cs`

Содержит пример класса, наследованного от абстрактного класса `Individual`. В примере приведена реализация индивида, построенного на базе управляющего конечного автомата для задачи об «умном муравье»

```
#region Usings

using ArtificialAnt.ProblemInfo;
using NewBrain;

#endregion

namespace ArtificialAnt.IndividualInfo
{
    /// <summary>
    /// Автомат в задаче об "умном муравье"
    /// </summary>
    public class Automation : Individual
    {
        /// <summary>
        /// Матрица состояний автомата
        /// </summary>
        private readonly Transition[,] _transitions;

        /// <summary>
        /// Новый автомат
        /// </summary>
        /// <param name="initialState">Начальное состояние</param>
        /// <param name="statesCount">Число состояний</param>
        public Automation(int initialState, int statesCount)
        {
            InitialState = initialState;
            StatesCount = statesCount;
            _transitions = new Transition[statesCount,2];
        }

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

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

        public Transition GetTransition(int index, int condition)
        {
            return _transitions[index, condition];
        }

        public void SetTransition(int index, int condition, Transition
transition)
        {
            _transitions[index, condition] = transition;
        }

        #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
    }
    }
```

3. Файл `GeneticAlgorithm.cs`

Данный файл содержит пример класса, наследованного от абстрактного класса `Algorithm`. В примере приведена реализация простого генетического алгоритма. Описание методов `NextGeneration` и `BigMutation` содержится в файл `SimpleGeneticAlgorithm.cs`, представленном в следующем разделе.

```
#region Usings

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

#endregion

namespace Genetic
{
    [SearchOperatorType(typeof (IGeneticSearchOperator))]
    public abstract class GeneticOptimizationAlgorithm : Algorithm
    {
            [Configurable]
            [Description("Размер поколения")]
        public int GenerationSize { get; set;}

        [Configurable]
        [Description("Количество поколений, после генерации которого большая
мутация")]
```

```
        public int BigMutationPeriod { get; set; }

        /// <summary>
        /// Генетический алгоритм оптимизации
        /// </summary>
        protected GeneticOptimizationAlgorithm()
        {
            GenerationSize = 100;
            BigMutationPeriod = 500;
            Generation = null;
        }

        protected Random Random { get; private set; }

        /// <summary>
        /// Текущее "поколение" особей для работы алгоритма
        /// </summary>
        protected List<Individual> Generation { get; set; }

        protected IGeneticSearchOperator GeneticSearchOperator
        {
            get { return SearchOperator as IGeneticSearchOperator; }
        }

        /// <summary>
        /// Порядковый номер текущего поколения
        /// </summary>
        public int GenerationNumber { get; private set; }

        /// <summary>
        /// Инициализация начального состояния генетического алгоритма.
        /// Создается первое поколение особей, полученное случайным образом.
        /// </summary>
        protected override void Initialize()
        {
            Random = new Random();
            Generation = new List<Individual>(GenerationSize);
            for (int i = 0; i < GenerationSize; i++)
                Generation.Add(SearchOperator.Create(Random));
        }

        /// <summary>
        /// Очередная итерация генетического алгоритма.
        /// Происходит переход к следующему поколению особей. Осуществляются
мутации.
        /// </summary>
        protected override void NextIteration()
        {
            GenerationNumber++;
            NextGeneration();
            if (GenerationNumber >= BigMutationPeriod)
            {
                GenerationNumber = 0;
                BigMutation();
            }
        }
        /// <summary>
        /// Метод получения нового поколения особей описывается в конкретных
реализациях
        /// генетического алгоритма.
        /// </summary>
        protected abstract void NextGeneration();

        /// <summary>
```

```
        /// Описывается в конкретных реализациях генетического алгоритма.
        /// </summary>
        protected abstract void BigMutation();
    }
    }
```

4. Файл SimpleGeneticAlgorithm.cs

```
#region Usings

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

#endregion

namespace SimpleGeneticOptimizationAlgorithm
{
    /// <summary>
    /// Простой генетический алгоритм
    /// </summary>
    public class SimpleGeneticOptimizationAlgorithm :
GeneticOptimizationAlgorithm
    {
            public SimpleGeneticOptimizationAlgorithm()
            {
                  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.SimpleGenetic; }
        }

        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; }
        }

        /// <summary>
        /// Генерация нового поколения заключается в выборе случайным образом
двух особей из старого поколения и получения
        /// на их основе двух новых особей. 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);
            double[] pr = new double[gSize];
            double fullFitness =0;
            for (int i = 0; i < gSize; i++)
                fullFitness += Generation[i].Fitness;
            for (int i = 0; i < gSize; i++)
                pr[i] = Generation[i].Fitness / fullFitness;

                newGeneration.AddRange(Generation.GetRange(0, eliteSize));
            for (int i = 0; i < gSize - eliteSize; i++)
            {
                double val = Random.NextDouble();
                int index = 0;
                while (val > 0 && index < (gSize - 1))
                    val -= pr[index++];
                newGeneration.Add(Generation[index]);
            }

            Generation.Clear();
            Generation.AddRange(newGeneration);
            newGeneration.Clear();
            newGeneration.AddRange(Generation.GetRange(0, eliteSize));

            for (int i = 0; i < (gSize - eliteSize)/2; i++)
            {
                Individual a1, a2;
                a1 = Generation[Random.Next(gSize)];
                a2 = Generation[Random.Next(gSize)];
                Individual[] s = GeneticSearchOperator.Crossover(new[] {a1, a2},
Random);
                newGeneration.AddRange(s);
            }
            if (newGeneration.Count < gSize)

newGeneration.Add(GeneticSearchOperator.Mutate(Generation[Random.Next(gSize)],
Random));

            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];
        }

        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();
            }
        }
    }
    }
```

5. Файл GeneticSearchOperatorOnAutomaton.cs

Содержит пример класса, наследованного от абстрактного класса `SearchOperator`. Данный файл содержит описание необходимых методов генетического алгоритма: create, crossover, mutate применительно к объектам на базе управляющих конечных автоматов, описанных в файле `Automation.cs`

```
#region Usings

using System;
using System.ComponentModel;
using ArtificialAnt.IndividualInfo;
using ArtificialAnt.ProblemInfo;
using Genetic.Operators;
using NewBrain;
using NewBrain.Attributes;

#endregion

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

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

      public override Individual Create(Random random)
        {
            Automation automation = new Automation(0, StatesCount);
            for (int i = 0; i < StatesCount; i++)
                for (int j = 0; j < 2; j++)
                    automation.SetTransition(i, j,
```

```
                                                new
Automation.Transition(random.Next(StatesCount), GetRandomAction(random)));

            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 GeneticSearchOperatorOnAutomation : SearchOperatorOnAutomation,
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 state = random.Next(aa.StatesCount);
            int c = random.Next(2);

            int iss = aa.InitialState;
            if (random.Next(2) == 1)
            {
                iss = random.Next(aa.StatesCount);
                alreadyMutate = true;
            }

            Automation res = new Automation(iss, aa.StatesCount);

            for (int i = 0; i < res.StatesCount; i++)
            {
                Automation.Transition t = aa.GetTransition(i, 1);
```

```
                res.SetTransition(i, 1, new Automation.Transition(t.EndState,
t.Action));
                t = aa.GetTransition(i, 0);
                res.SetTransition(i, 0, new Automation.Transition(t.EndState,
t.Action));
            }

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

            return res;
        }

        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 = new Automation[2];
            for (int i = 0; i < 2; i++)
                s[i] = new Automation(0, aa1.StatesCount);

            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;
            }

            for (int i = 0; i < aa1.StatesCount; i++)
            {
                int flag = random.Next(4);
                switch (flag)
                {
                    case 0:
                        s[0].SetTransition(i, 0, aa2.GetTransition(i, 0));
                        s[0].SetTransition(i, 1, aa2.GetTransition(i, 1));
                        s[1].SetTransition(i, 0, aa1.GetTransition(i, 0));
                        s[1].SetTransition(i, 1, aa1.GetTransition(i, 1));
                        break;
                    case 1:
                        s[0].SetTransition(i, 0, aa1.GetTransition(i, 0));
                        s[0].SetTransition(i, 1, aa1.GetTransition(i, 1));
                        s[1].SetTransition(i, 0, aa2.GetTransition(i, 0));
                        s[1].SetTransition(i, 1, aa2.GetTransition(i, 1));
                        break;
                    case 2:
                        s[0].SetTransition(i, 0, aa2.GetTransition(i, 0));
                        s[0].SetTransition(i, 1, aa1.GetTransition(i, 1));
                        s[1].SetTransition(i, 0, aa2.GetTransition(i, 0));
                        s[1].SetTransition(i, 1, aa1.GetTransition(i, 1));
                        break;
                    case 3:
```

```
                    s[0].SetTransition(i, 0, aa1.GetTransition(i, 0));
                    s[0].SetTransition(i, 1, aa2.GetTransition(i, 1));
                    s[1].SetTransition(i, 0, aa1.GetTransition(i, 0));
                    s[1].SetTransition(i, 1, aa2.GetTransition(i, 1));
                    break;
                }
            }

            return s;
        }

        #endregion
    }
    }
```