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

Д.С. Чивилихин

Отчет по лабораторной работе «Использование генетических алгоритмов для построения управляющих автоматов»

Вариант №1

Санкт-Петербург
2009

Оглавление

1. Введение

Цель данной лабораторной работы – применение генетического алгоритма для построения конечного автомата Мили, решающего задачу об «Умном муравье».

Для выполнения работы использовалась программа «Виртуальная лаборатория», написанная студентами кафедры «Компьютерные технологии» СПбГУ ИТМО, и позволяющая реализовывать генетические алгоритмы их особи для них в виде подключаемых модулей (плагинов).

1.1. Постановка задачи

Задача данной лабораторной работы – построение автомата Мили, решающего задачу об «Умном муравье». При этом, муравей, управляемый построенным автоматом, имеющим фиксированное число состояний, должен съедать всю еду на поле за фиксированное число шагов.

1.2. Автомат Мили

Автомат Мили – конечный автомат, генерирующий выходные воздействия в зависимости от текущего состояния и входного воздействия. На рис.1 показан пример представления автомата Мили в виде графа переходов.

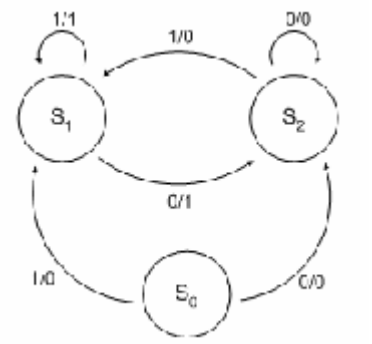

Рис. 1

Над каждой дугой показана пара значений – входное воздействие и выходное воздействие. Выходное воздействие зависит как от входного воздействия, так и от текущего состояния автомата.

1.3. Задача об «Умном муравье»

Игра происходит на поверхности тора размером 32x32 клетки. В некоторых клетках находится еда. Еда расположена не случайно, а лежит на некоторой ломаной. На поле находится 89 клеток с едой. Муравей может видеть только клетку перед собой. Муравей начинает движение из клетки, помеченной «Start».

За один ход муравей может выполнить следующие действия (в зависимости от входных параметров):

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

Игра длится 200 ходов. Цель игры – создание муравья с минимальным числом состояний, который за минимальное число шагов съест всю еду на поле.

2. Реализация

Виртуальная лаборатория представляет собой программу, предназначенную для разработки и изучения генетических алгоритмов. Лаборатория состоит из ядра и подключаемых к нему модулей (плагинов).

Ядро обеспечивает интерфейс пользователя, вывод графиков, и связывает плагины для формирования исследуемого алгоритма. Функциональность приложения обеспечивается за счет подключения к ядру модулей.

Для решения поставленной задачи необходимо создать два модуля. Первый модуль реализует канонический генетический алгоритм и метод рулетки для генерации очередного поколения. Второй модуль реализует «особь» – конечный автомат Мили, а так же операторы мутации и скрещивания [1].

2.1. Представление автомата

Автомат представляется в виде битовой строки, реализованной на основе класса *ArrayList* стандартной библиотеки *Java*.

На рисунке показано представления автомата с *N* состояниями [2] (рис.2).

Стартовое состояние	Состояние 1	Состояние 2	…	Состояние N

Рис. 2

Каждое состояние кодируется следующим образом: для каждого входного воздействия («нет еды», «есть еда») указывается номер состояния, в которое перейдет автомат, а также действие, которое он совершит при переходе (*L*, *R*, *M*). Действие *N* (ничего не делать) не рассматривается. На рис. 3 показана схема кодирования состояния автомата.

Состояние 5	R	Состояние 2	M

Рис. 3

При входном воздействии «нет еды», автомат перейдет в пятое состояние, совершив поворот направо, а при воздействии «есть еда», перейдет во второе состояние, совершив шаг вперед.

Входное воздействие – состояние клетки, которую муравей видит впереди себя.

Выходные воздействия автомата кодируются следующим образом:

- L = 01;
- R = 10;
- M = 11.

Отдельно хранится число состояний автомата. Длина кода каждого состояние выравнивается под длину состояние с максимальным номером.

2.2. Метод скрещивания

Оператор скрещивания получает на вход две особи, и выдает на выходе две особи. Скрещивание происходит следующим образом:

- случайным образом выбирается точка кроссовера (в пределах длины битовой строки);
- строки обмениваются хвостами после точки кроссовера.

2.3. Метод мутации

С вероятностью 100% инвертируется ровно один бит в битовом представлении автомата. Номер бита выбирается случайно.

2.4. Метод генерации очередного поколения

Начальное поколение состоит из некоторого фиксированного числа (в данном случае, 200) случайно сгенерированных автоматов с фиксированным числом состояний. В рамках одной задачи число состояний автоматов остается неизменным.

Для генерации следующего поколения используется канонический генетический алгоритм и метод рулетки. Шаг алгоритма разбит на три стадии: генерация промежуточного поколения путем отбора из текущего поколения, скрещивание особей из промежуточной популяции и мутация.

На этапе отбора в промежуточную популяцию добавляются особи, получившие право размножаться. Лучшие особи могут попасть туда несколько раз. В каноническом алгоритме используется метод пропорционального отбора, реализованный здесь с помощью метода рулетки: расположим все особи на колесе, причем размер сектора для каждой особи пропорционален ее приспособленности. Запуская рулетку N раз, получим требуемое число особей для промежуточной популяции.

На этапе скрещивания к особям в популяции применяется оператор скрещивания, и результирующие особи отправляются в следующее

поколение. Для достижения стабильности работы генетического алгоритма, некоторое фиксированное число лучших особей сразу переходит в следующее поколение.

На этапе мутации ко всем особям применяется оператор мутации, описанный выше.

На втором и третьем этапах, автоматы после скрещивания и мутации могут оказаться «плохими» – могут появиться состояния с номерами больше максимального, а также неопределенные выходные воздействия. Это связано с битовым представлением автоматов. В случае появления таких «плохих» особей, они удаляются из промежуточной популяции и заменяются новыми случайно сгенерированными автоматами.

Для достижения более быстрой и стабильной работы алгоритма, с некоторым заданным периодом времени применяется большая мутация: у всех особей в популяции меняется 30% битов.

2.5. Способ вычисления функции приспособленности

Будем эмулировать действия муравья до тех пор, пока он либо не съест всю еду, либо превысит максимально допустимое число шагов.

Значение функции приспособленности (Fitness) вычисляется по формуле:

$$\mathbf{Fitness = ApplesCount - MaxStep / 200,}$$

где **ApplesCount** – число съеденных муравьем яблок за 200 шагов, **MaxStep** – ход, на котором муравей последний раз съел яблоко.

3. Результаты работы

В результате работы описанного генетического алгоритма был получен автомат Мили, решающий задачу об «Умном муравье».

Автомат состоит из одиннадцати состояний, значение функции приспособленности равно 88.03. Муравей, управляемый данным автоматом, съедает все 89 яблок за 195 шагов. Автомат был получен в 129-м поколении.

3.1. Графики значений функции приспособленности

На рис. 4 приведены графики зависимости среднего (отмечено зеленым) и максимального (отмечено синим) значения функции приспособленности.

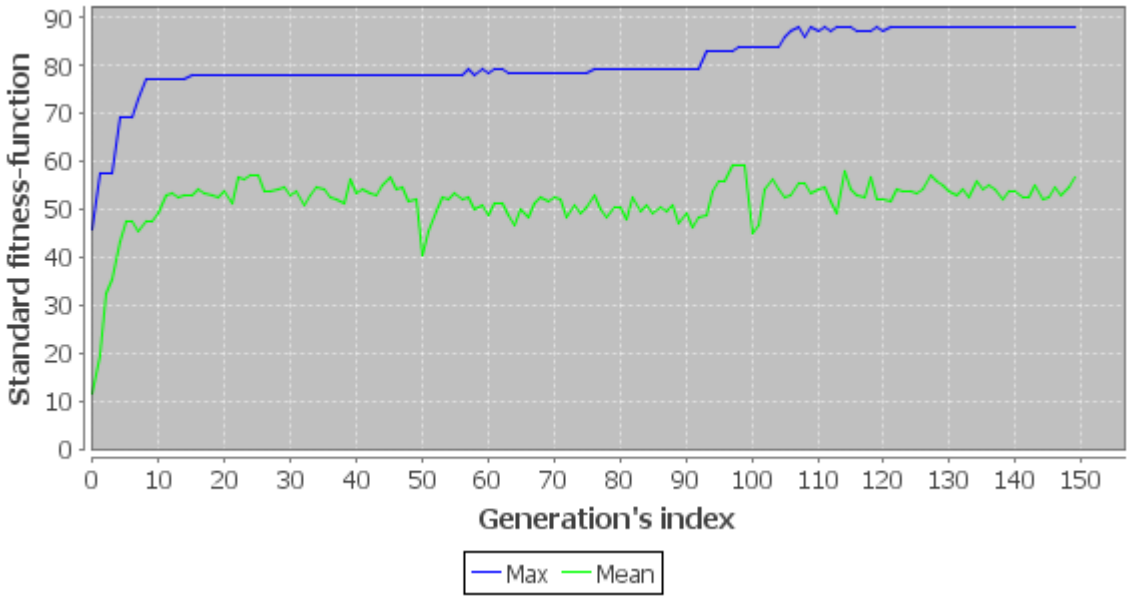

Рис. 4

3.2. Таблица переходов полученного автомата

Полученный автомат Мили представлен в табл. 1 в виде таблицы переходов. Входное воздействие может быть: T – «впереди есть еда», F – «впереди нет еды» . Выходное действие может быть: L – повернуть налево, R – повернуть направо, M – сделать шаг вперед.

Начало	Конец	Вход	Выход
1	3	F	L
1	4	T	M
2	2	T	M
2	7	F	L
3	2	T	M
3	11	F	M
4	1	F	R
4	6	T	M
5	4	T	M
5	9	F	R
6	11	T	M
6	9	F	L
7	5	T	M
7	4	F	L
8	3	T	M
8	11	F	R
9	10	F	R
9	3	T	M
10	3	T	L
10	8	F	R
11	4	T	M
11	5	F	M

Табл. 1

Заключение

Результаты лабораторной работы показали, что с помощью генетического алгоритма можно сгенерировать автомат Мили с одиннадцатью состояниями, эффективно решающий задачу об «Умном муравье». Полученный муравей съедает все яблоки за 195 шагов. Известен автомат, построенный вручную, управляющий муравьем, съедающим 81 яблоко [2]. Также, в работе [2] приведен автомат Мили, съедающий всю еду, и имеющий 10 состояний, однако для его получения потребовалось значительно больше времени.

Источники

1. Инструкция по созданию plugin'ов к виртуальной лаборатории http://svn2.assembla.com/svn/not_instrumental_tool/docs/pdf/interface_manual.pdf

2. *Бедный Ю.Д., Шалыто А.А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей» http://is.ifmo.ru/works/_ant.pdf

3. *Яминов Б.* Генетические алгоритмы http://rain.ifmo.ru/cat/view.php/theory/unsorted/genetic-2005

Приложение. Конфигурационные файлы

Для сборки модулей виртуальной лаборатории использовался ant-скрипт, находящийся в `build.xml`. В файле `build.properties` содержатся параметры для этого скрипта.

Модуль «особь»:

1. `individual.properties` – конфигурация задания;
2. `build.properties` – конфигурация модуля.

Модуль генетического алгоритма:

1. `algorithm.properties` – конфигурация генетического алгоритма;
   - `size.generation` – размер популяции;
   - `size.elite` – размер фиксированной части популяции;
2. `plugin.properties` – конфигурация модуля.

Приложение. Исходные тексты программы

Модуль особи:

1. `MealyIndividualFactoryLoader.java` – класс, реализующий загрузку загрузку модуля в лабораторию.
2. `MealyAutomaton.java` – класс, реализующий конечный автомат Мили.
3. `MealyAutomatonFactory.java` – класс, реализующий создание произвольной особи.
4. `StandardCrossover.java` – класс, реализующий оператор кроссовера.
5. `StandardMutation.java` – класс, реализующий оператор мутации.

Модуль генетического алгоритма:

`SimpleGA.java` – класс, реализующий канонический генетический алгоритм.

Листинг *MealyIndividualFactoryLoader.java*

```
package laboratory.plugin.individual.mealy;

import laboratory.plugin.individual.mealy.factory.MealyAutomatonFactory;
import laboratory.plugin.individual.mealy.gui.ConfigDialog;
import laboratory.common.genetic.IndividualFactory;
import laboratory.common.genetic.operator.Fitness;
import laboratory.util.loader.AbstractIndividualLoader;
import laboratory.util.loader.JarReader;
import laboratory.util.StandardFitness;

import javax.swing.*;
import java.util.jar.JarFile;
import java.util.List;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Properties;
import java.io.File;

public class MealyIndividualFactoryLoader extends
AbstractIndividualLoader<MealyAutomaton>{

    public MealyIndividualFactoryLoader(JarFile jar, File dir){
        super(jar, dir);
               Config.getInstance().setEx(Integer.parseInt(getProperty("external")));
        Config.getInstance().setIn(Integer.parseInt(getProperty("internal")));
    }
```

```
    @Override
    public List<IndividualFactory<MealyAutomaton>> loadFactories(){
        return Arrays.asList((IndividualFactory<MealyAutomaton>)new
MealyAutomatonFactory(Config.getInstance().getEx(),
                Config.getInstance().getIn()));
    }

    @Override
    public List<Fitness<MealyAutomaton>> loadFunctions(){
        List<Fitness<MealyAutomaton>> res = new ArrayList<Fitness<MealyAutomaton>>();
        res.add(new StandardFitness<MealyAutomaton>());
        return res;
    }

    @Override
    public JDialog getConfigDialog(JFrame owner){
        return new ConfigDialog(owner, JarReader.getProperties(getJar(),
"frame.config.properties"), this);
    }
}
```

Листинг *MealyAutomaton.java*

```
package laboratory.plugin.individual.mealy;

import laboratory.plugin.task.ant.individual.AbstractAutomaton;
import laboratory.plugin.task.ant.individual.Automaton;
import laboratory.plugin.task.ant.Ant;
import java.util.Arrays;
import java.util.Collection;
import java.util.ArrayList;

import java.io.File;
import java.io.PrintWriter;


public class MealyAutomaton extends AbstractAutomaton{

    public MealyAutomaton(int is, Automaton.Transition[][] tr, MealyAutomaton na){
        super(is, na, tr);
    }

    public String getStateString(int i){
        return (i + 1) + "";
    }

      public Automaton.Transition[][] getAutomatonTransition(ArrayList<Integer>
bitString, int stateCnt) throws Exception{
            Automaton.Transition[][] t = new Automaton.Transition[stateCnt][2];

            boolean bad = false;

            int localCnt = 0;
            int transitionCnt = 0;
            int stateLength = (Integer.toBinaryString(stateCnt)).length();
            ArrayList<Integer> transitions = new ArrayList<Integer>();

            for (int i = 0; i < 2 * (stateLength + 2); i++) {
                  transitions.add(0);
            }

            ArrayList<Integer> transition0 = new ArrayList<Integer>(stateLength + 2);
            ArrayList<Integer> transition1 = new ArrayList<Integer>(stateLength + 2);

            for (int i = 0; i < stateLength + 2; i++) {
                  transition0.add(0);
                  transition1.add(0);
```

```
		}

		int missedChar = 0;
		for (int i = stateLength; i <= bitString.size(); i++) {
			if (localCnt < 2 * (stateLength + 2) && i != bitString.size()) {
				if (i != stateLength && localCnt == 0) {
					localCnt = 1;
					transitions.set(0, missedChar);
				}
				transitions.set(localCnt, bitString.get(i));
				++localCnt;
			} else {
				if (i != bitString.size()) {
					missedChar = bitString.get(i);
				}
				// разбиваем полученную строчку transitions на 2 строки:
			// переход по 0 и по 1
				localCnt = 0;
				for (int j = 0; j < transition0.size(); j++) {
					transition0.set(j, transitions.get(j));
				}

				for (int j = 0; j < transition1.size(); j++) {
					transition1.set(j, transitions.get(transition1.size()
+ j));
				}

				ArrayList<Integer> endState0 = new ArrayList<Integer>();
				ArrayList<Integer> endState1 = new ArrayList<Integer>();

				for (int j = 0; j < stateLength; ++j) {
					endState0.add(0);
					endState1.add(0);
				}

				String strAction0, strAction1;
				char charAction0 = 'N';
				char charAction1 = 'N';
				for (int j = 0; j < stateLength; j++) {
					endState0.set(j, transition0.get(j));
					endState1.set(j, transition1.get(j));
				}

				strAction0 = String.valueOf(transition0.get(stateLength))
						+ String.valueOf(transition0.get(stateLength +
1));

				strAction1 = String.valueOf(transition1.get(stateLength))
						+ String.valueOf(transition1.get(stateLength +
1));


				if (strAction0.equalsIgnoreCase("01")) {
					charAction0 = 'L';
				}

				if (strAction0.equalsIgnoreCase("10")) {
					charAction0 = 'R';
				}

				if (strAction0.equalsIgnoreCase("11")) {
					charAction0 = 'M';
				}

				//Проверяем на вшивость=)
				if (charAction0 == 'N') {
					bad = true;
				}


				if (strAction1.equalsIgnoreCase("01")) {
```

```
                                charAction1 = 'L';
                            }

                            if (strAction1.equalsIgnoreCase("10")) {
                                charAction1 = 'R';
                            }

                            if (strAction1.equalsIgnoreCase("11")) {
                                charAction1 = 'M';
                            }

                            //Проверяем на вшивость=)
                            if (charAction1 == 'N') {
                                bad = true;
                            }

                            int eS0 = 0;
                            int eS1 = 0;

                            for (int j = endState0.size() - 1; j >= 0; j--) {
                                if (endState0.get(j) == 1) {
                                    eS0 += (int) Math.pow(2, endState0.size() - j -
1);
                                }

                                if (endState1.get(j) == 1) {
                                    eS1 += (int) Math.pow(2, endState1.size() - j -
1);
                                }
                            }

                            if (eS0 > stateCnt - 1 || eS1 > stateCnt - 1) {
                                bad = true;
                            }

                            if (transitionCnt < stateCnt) {
                                t[transitionCnt][0] = new
                                MealyAutomaton.Transition(eS0, charAction0);
                                t[transitionCnt][1] = new
                                MealyAutomaton.Transition(eS1, charAction1);
                                ++transitionCnt;
                            }
                        }
                    }
                    if (bad) {
                        throw new Exception("Bad individual!");
                    }

                    return t;
                }


    public Automaton setInitialState(int newIS) {
        return new MealyAutomaton(newIS, getTransition(),
(MealyAutomaton)getNestedAutomaton());
    }

    public Automaton setTransitions(Automaton.Transition[][] transitions) {
        return new MealyAutomaton(getInitialState(), transitions,
(MealyAutomaton)getNestedAutomaton());
    }

    public Automaton setNestedAutomaton(Automaton a) {
        return new MealyAutomaton(getInitialState(), getTransition(),
(MealyAutomaton)a);
    }


        public ArrayList<Integer> toBitString() {
            ArrayList<Integer> bitstring = new ArrayList<Integer>();
            String initialStateNumber = Integer.toBinaryString(getInitialState());
```

```
        int stateCntlength = Integer.toBinaryString(getNumberStates()).length();

        while (initialStateNumber.length() < stateCntlength) {
               initialStateNumber = "0" + initialStateNumber;
        }

        // Добавляем в битовую строку номер начального состояния
        for (int i = 0; i < initialStateNumber.length(); i++) {

bitstring.add(Integer.parseInt(String.valueOf(initialStateNumber.charAt(i))));
        }

        for (int i = 0; i < getNumberStates(); i++) {
               String chromosome0, chromosome1;
               String endState0 = Integer.toBinaryString(getTransition(i, 0)
                             .getEndState());
               String endState1 = Integer.toBinaryString(getTransition(i, 1)
                             .getEndState());

               String action0 = "";
               String action1 = "";

               char act0 = getTransition(i, 0).getAction();
               switch (act0) {
               case 'L':
                      action0 = "01";
                      break;
               case 'R':
                      action0 = "10";
                      break;
               case 'M':
                      action0 = "11";
                      break;
               }

               char act1 = getTransition(i, 1).getAction();
               switch (act1) {
               case 'L':
                      action1 = "01";
                      break;
               case 'R':
                      action1 = "10";
                      break;
               case 'M':
                      action1 = "11";
                      break;
               }

               // Выравниваем длину хромосомы
               String zero = String.valueOf('0');
               while (endState0.length() < stateCntlength) {
                      endState0 = zero + endState0;
               }

               while (endState1.length() < stateCntlength) {
                      endState1 = zero + endState1;
               }

               chromosome0 = endState0 + action0;
               chromosome1 = endState1 + action1;

               for (int j = 0; j < chromosome0.length(); j++) {
                      bitstring.add(Integer.parseInt(String.valueOf(chromosome0
                                    .charAt(j))));
               }

               for (int j = 0; j < chromosome1.length(); j++) {
                      bitstring.add(Integer.parseInt(String.valueOf(chromosome1
                                    .charAt(j))));
               }
        }
```

```
            return bitstring;

       }


    public static class Transition extends AbstractAutomaton.Transition{

        private final char action;

        public Transition(int endState, char action){
               super(endState);
            this.action = action;
        }

        public Automaton.Transition setEndState(int newEnd){
            return new Transition(newEnd, action);
        }

        public char getAction(){
            return action;
        }

        public String toString(){
            return "" + action;
        }
    }
}
```

Листинг *MealyAutomatonFactory.java*

```
package laboratory.plugin.individual.mealy.factory;

import laboratory.plugin.individual.mealy.MealyAutomaton;
import laboratory.plugin.task.ant.Ant;
import laboratory.plugin.task.ant.individual.factory.AutomatonFactory;


public class MealyAutomatonFactory extends AutomatonFactory<MealyAutomaton>{

    public MealyAutomatonFactory(int ns, int nns){
        super(ns, nns);
    }

    protected MealyAutomaton fullRandomAutomaton(int ns){
        MealyAutomaton.Transition[][] tr = new MealyAutomaton.Transition[ns][2];
        for(int i = 0;i < ns;i++){
            tr[i][1] = new MealyAutomaton.Transition(r.nextInt(ns),
Ant.ACTION_VALUES[r.nextInt(3)]);
            tr[i][0] = new MealyAutomaton.Transition(r.nextInt(ns),
Ant.ACTION_VALUES[r.nextInt(3)]);
        }
        return new MealyAutomaton(r.nextInt(ns), tr, null);
    }

    protected MealyAutomaton randomAutomatonWN(int ns, MealyAutomaton na){
        MealyAutomaton.Transition[][] tr = new MealyAutomaton.Transition[ns][2];
        for(int i = 0;i < ns;i ++){
            tr[i][1] = new MealyAutomaton.Transition(r.nextInt(ns + 1) - 1,
Ant.ACTION_VALUES[r.nextInt(3)]);
            tr[i][0] = new MealyAutomaton.Transition(r.nextInt(ns + 1) - 1,
Ant.ACTION_VALUES[r.nextInt(3)]);
        }
        return new MealyAutomaton(r.nextInt(ns), tr, na);
    }
}
```

Листинг *StandardCrossover.java*

```
package laboratory.plugin.individual.mealy.operator.crossover.standard;

import laboratory.plugin.individual.mealy.MealyAutomaton;
import laboratory.plugin.task.ant.individual.operator.AbstractAutomatonCrossover;

import java.util.List;
import java.util.ArrayList;
import java.util.Random;
import java.io.File;
import java.io.PrintWriter;


public class StandardCrossover extends AbstractAutomatonCrossover<MealyAutomaton>{

    private static final Random random = new Random();

    public StandardCrossover(){
        super(random);
    }

    public List<MealyAutomaton> apply(List<MealyAutomaton> parents) {

             MealyAutomaton parent1 = parents.get(0);
             MealyAutomaton parent2 = parents.get(1);

             ArrayList<Integer> bitParent1 = parent1.toBitString();
             ArrayList<Integer> bitParent2 = parent2.toBitString();

             int size = bitParent1.size();

             int stateLength =
(Integer.toBinaryString(parent1.getNumberStates())).length();

             int point = random.nextInt(size);

             for (int i = point;  i < size; i++) {
                    int tmp = bitParent1.get(i);
                    bitParent1.set(i, bitParent2.get(i));
                    bitParent2.set(i, tmp);
             }

             int initialState1 = 0;
             int initialState2 = 0;

             for (int j = stateLength - 1; j >= 0; j--) {
                    if (bitParent1.get(j) == 1) {
                           initialState1 += (int) Math.pow(2, stateLength - j - 1);
                    }

                    if (bitParent2.get(j) == 1) {
                           initialState2 += (int) Math.pow(2, stateLength - j - 1);
                    }

             }

             ArrayList<MealyAutomaton> result = new ArrayList<MealyAutomaton>();

             try {
                    result.add(new MealyAutomaton(initialState1,
                                  parent1.getAutomatonTransition(bitParent1,
                                  parent1.getNumberStates()),
                                        (MealyAutomaton)parent1.getNestedAutomaton()));

                    result.add(new MealyAutomaton(initialState2,
                                  parent2.getAutomatonTransition(bitParent2,
                                  parent2.getNumberStates()),
                                        (MealyAutomaton)parent2.getNestedAutomaton()));
             } catch (Exception e) {
                    return null;
```

```
            }
            return result;

      }
}
```

Листинг *StandardMutation.java*

```
package laboratory.plugin.individual.mealy.operator.mutation.standard;

import laboratory.plugin.task.ant.individual.operator.AbstractAutomatonMutation;
import laboratory.plugin.individual.mealy.MealyAutomaton;
import laboratory.plugin.task.ant.individual.Automaton;

import java.util.Random;
import java.util.Arrays;
import java.util.Collection;
import java.util.ArrayList;


import java.io.File;
import java.io.PrintWriter;

public class StandardMutation extends AbstractAutomatonMutation<MealyAutomaton>{

    private static final Random random = new Random();

    public StandardMutation(){
        super(random);
    }

       public MealyAutomaton apply(MealyAutomaton individual){

             MealyAutomaton result = individual;
             ArrayList<Integer> bitString = new ArrayList<Integer>();
             bitString = result.toBitString();
             int stateCnt = result.getNumberStates();
             int stateLength = (Integer.toBinaryString(stateCnt)).length();

             int i = 0;

             i =  random.nextInt(bitString.size());

             if (bitString.get(i) == 0) {
                    bitString.set(i, 1);
             } else {
                    bitString.set(i, 0);
             }

             int initialState = 0;
             for (int j = stateLength - 1; j >= 0; j--) {
                    if (bitString.get(j) == 1) {
                           initialState += (int) Math.pow(2, stateLength - j - 1);
                    }
             }

        try {
                    return new MealyAutomaton(initialState,
result.getAutomatonTransition(bitString, stateCnt),

(MealyAutomaton)result.getNestedAutomaton());
             } catch (Exception e) {
                    return null;
             }
       }
}
```

Листинг *SimpleGA.java*

```
package laboratory.plugin.algorithm.simple;

import laboratory.common.genetic.Algorithm;
import laboratory.common.genetic.Individual;
import laboratory.common.genetic.IndividualFactory;
import laboratory.common.genetic.FitIndividual;
import laboratory.common.genetic.operator.Mutation;
import laboratory.common.genetic.operator.Crossover;
import laboratory.common.genetic.operator.Selection;
import laboratory.common.genetic.operator.Fitness;
import laboratory.util.functional.Util;
import laboratory.util.functional.Functor1;
import laboratory.util.functional.Functor0;

import java.util.*;

public class SimpleGA<I extends Individual> implements Algorithm<I>{

    private List<FitIndividual<I>> generation;

    private final double probabilityMutation;

    private final IndividualFactory<I> factory;

    private final Random r;

    private final Mutation<I> mut;
    private final Crossover<I> cross;
    private final Selection<I> sel;
    private final Fitness<I> fitness;

    private final int fixedPart;

       private int generationIndex;
       private static int bigMutationPeriod = 50;

    public SimpleGA(final int sizeGeneration, final double probabilityMutation, final
IndividualFactory<I> factory,
                    final Mutation<I> mut, final Crossover<I> cross, final
Selection<I> sel, final Fitness<I> fitness){

        this.probabilityMutation = probabilityMutation;

        this.factory = factory;

        this.mut = mut;
        this.cross = cross;
        this.sel = sel;
        this.fitness = fitness;

        fixedPart = sizeGeneration / 2;

        generation = Util.listFromFunctor(new Functor0<FitIndividual<I>>(){
            public FitIndividual<I> apply(){
                return cons(factory.getIndividual());
            }
        }, sizeGeneration);
        Collections.sort(generation);

        r = new Random();
             this.generationIndex = 0;
    }

    private I winner(FitIndividual<I> a1, FitIndividual<I> a2){
        return ((a1.compareTo(a2) < 0) ? a1 : a2).ind;
    }

    private FitIndividual<I> cons(I i){
        return new FitIndividual<I>(i, fitness.apply(i));
```

```
    }

    private FitIndividual<I> randomI(){
        return generation.get(r.nextInt(generation.size()));
    }

    public void nextGeneration(){
        int size = generation.size();
        List<FitIndividual<I>> newGeneration = new ArrayList<FitIndividual<I>>(size);
        newGeneration.addAll(sel.apply(generation, fixedPart));

        while(newGeneration.size() + 1 <= generation.size()) {
                    List<I> s = new ArrayList<I>();
             try {
                         s = cross.apply(Arrays.asList(winner(randomI(), randomI()),
winner(randomI(), randomI())));
                         for(I ind : s){
                newGeneration.add(cons(ind));
                         }
                    } catch (Exception e) {
                        s = Arrays.asList(winner(randomI(), randomI()),
winner(randomI(), randomI()));
                         for(I ind : s){
                newGeneration.add(new FitIndividual<I>(ind,
Double.NEGATIVE_INFINITY));
                         }
                    }
        }

        if(newGeneration.size() < size) {
            newGeneration.add(cons(mut.apply(randomI().ind)));
        }

        for(int i = 0; i < size; i++) {
                         try {
                    newGeneration.set(i, cons(mut.apply(newGeneration.get(i).ind)));
                         } catch (Exception e) {
                                FitIndividual<I> individ = new
FitIndividual(newGeneration.get(i).ind, Double.NEGATIVE_INFINITY);
                                newGeneration.set(i, individ);
                         }
        }

        generation = newGeneration;

             if (generationIndex % bigMutationPeriod == 0) {
                    bigMutation();
             }

        Collections.sort(generation);
             generationIndex++;
    }


    public List<I> getGeneration(){
        return Util.map(generation, new Functor1<FitIndividual<I>, I>(){
            public I apply(FitIndividual<I> i){
                return i.ind;
            }
        });
    }

    public void stop(){
    }


    public void bigMutation(){
        for(int i = 0;i < generation.size();i++){

                    I ind = generation.get(i).ind;
                    I ind_bkp = ind;
```

```
                    for (int j = 0; j < 30; j++) {
                          try {
                                ind = mut.apply(ind);
                          } catch (Exception e) {
                                ind = ind_bkp;
                          }
                    }

                    try {
                          generation.set(i, new FitIndividual<I>(ind,
fitness.apply(ind)));
                    } catch (Exception e) {
                          generation.set(i, new FitIndividual<I>(ind_bkp,
fitness.apply(ind_bkp)));
                    }
        }
        Collections.sort(generation);
    }

}
```