Санкт-Петербургский государственный университет информационных технологий, механики и оптики
Факультет информационных технологий и программирования
Кафедра компьютерных технологий

А. П. Данильченко

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

Вариант № 3

Санкт-Петербург
2010

Оглавление

1. Введение

Цель данной лабораторной работы — применение генетических алгоритмов для построения конечных автоматов. В качестве примера рассматривается построение конечного автомата Мили, решающего задачу об «Умном муравье».

При выполнении лабораторной работы использовалась программа «Виртуальная лаборатория», написанная студентами кафедры компьютерных технологий СПбГУ ИТМО. Данная программа позволяет реализовывать особь и генетический алгоритм в качестве подключаемых модулей (plugin'ов) [1].

2. Постановка задачи

Задача данной лабораторной работы — построение автомата Мили, решающего задачу об «Умном муравье». При построении следует стремиться к наибольшей функции приспособленности. Иными словами, муравей, управляемый полученным автоматом, должен съедать как можно больше еды за как можно меньшее время.

2.1. Автомат Мили

Автомат типа Мили — это конечный автомат, который генерирует выходные воздействия в зависимости от текущего состояния и входного воздействия. Пример автомата Мили в виде диаграммы переходов приведен на рис. 1.

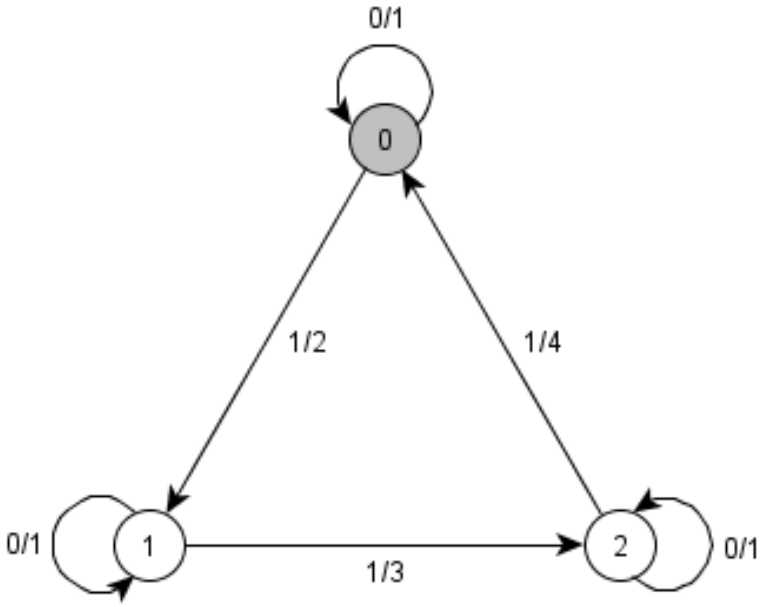


Рис. 1. Пример автомата Мили, заданного графом

Над каждым ребром расположено два значения — соответственно входное и выходное воздействия. При этом выходное воздействие зависит только от состояния и входного воздействия.

2.2. Задача об «Умном муравье»

Игра происходит на поверхности тора размером 32x32 клетки. В некоторых клетках тора находится еда. Муравей начинает движение из клетки, помеченной «Start».

Муравей умеет определять, находится ли непосредственно перед ним еда. За один ход муравей может совершить одно из следующих действий:

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

Съеденная еда не восстанавливается, муравей жив на протяжении всей игры, а расположение еды фиксировано. Игра длится 200 ходов. Цель игры — создать муравья, который за минимальное число ходов съест как можно большее количество еды.

3. Реализация

«Виртуальная лаборатория» состоит из ядра и подключаемых модулей [1]. Для решения поставленной задачи необходимо создать два модуля. Модуль «особи» — конечный автомат Мили, решающий задачу об «Умном муравье», и модуль, реализующий генетический алгоритм.

3.1. Представление автоматов

Каждый автомат представляется в виде битовой строки, реализованной на основе класса `ArrayList` из стандартной библиотеки *Java*. Представление автомата из *N* состояний приведено на рис. 2.

Стартовое состояние	Состояние 1	Состояние 2	Состояние 3	...	Состояние N

Рис. 2 Представление автомата

В представлении автомата хранится номер стартового состояния, затем друг за другом состояния. В описании состояния для каждого из двух возможных входных воздействий («есть еда», «нет еды») закодировано сначала состояние, в которое будет произведен переход, потом выходное воздействие. При кодировании состояния, куда будет совершен переход, записывается его номер в двоичной системе счисления. Число бит в представлении номера состояния равно числу бит в двоичной записи максимального номера состояния. Выходное воздействие кодируется двумя битами:

- 01 — поворот налево (L);
- 10 — поворот направо (R);
- 11 — шаг вперед (M).

Значение числа состояний хранится отдельно.

3.2. Метод скрещивания

Оператор скрещивания («кроссовера») получает на вход две особи и выдает также две особи. Скрещивание происходит по следующей схеме:

- случайным образом выбирается точка кроссовера;
- битовые строчки обмениваются хвостами после точки кроссовера.

3.3. Метод мутации

Оператор мутации инвертирует ровно один случайный бит в битовой строчке, которая представляет автомат.

3.4. Модель генетического алгоритма

Островной генетический алгоритм реализует модель параллельного генетического алгоритма [3]. Суть островного алгоритма заключается в следующем. Все особи разбиваются на некоторое количество популяций («островов»), которые развиваются независимо друг от друга. Время от времени происходит обмен хорошими особями между островами. Этот процесс носит название миграция. Она позволяет островам обмениваться генетическим материалом.

Так как населенность островов обычно невелика, подпопуляции будут иметь тенденцию к преждевременной сходимости. Поэтому важно правильно установить частоту миграции. Чересчур частая миграция приведет к смешению подпопуляций и тогда результат работы островного генетического алгоритма будет мало, чем отличаться от результатов работы канонического алгоритма. Если же миграция будет слишком редкой, то она не сможет предотвратить преждевременного схождения подпопуляций.

3.5. Метод генерации очередного поколения

Начальное поколение состоит из некоторого фиксированного числа случайно сгенерированных особей. Число состояний управляющих автоматов неизменно в рамках одной задачи.

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

На этапе отбора в промежуточную популяцию добавляются особи, получившие право размножаться. Отбор происходит по методу рулетки. Суть метода состоит в следующем: расположим особи на колесе, причем размер сектора для каждой особи пропорционален ее приспособленности. Запуская рулетку N раз, получим N особей, которые и сформируют промежуточное поколение. Заметим, что при таком способе генерации промежуточного поколения одна особь может быть отобрана несколько раз.

Затем к особям из промежуточного поколения применяются операторы скрещивания и мутации. При этом могут получиться «плохие» особи — битовая строчка представляет некорректный автомат. Могут появиться переходы в состояния, большие максимального или

неопределенные выходные воздействия. «Плохие» особи не включаются в новое поколение, они заменяются новыми случайно сгенерированными автоматами.

Для достижения более быстрой работы алгоритма с некоторой периодичностью применяется большая мутация — у всех особей меняется некоторое фиксированное число битов (например, 20%).

3.6. Способ вычисления функции приспособленности

Будем эмулировать поведение муравья до тех пор, пока он не съест всю еду или не превысит допустимое число шагов. Далее значение функции приспособленности вычисляется по формуле

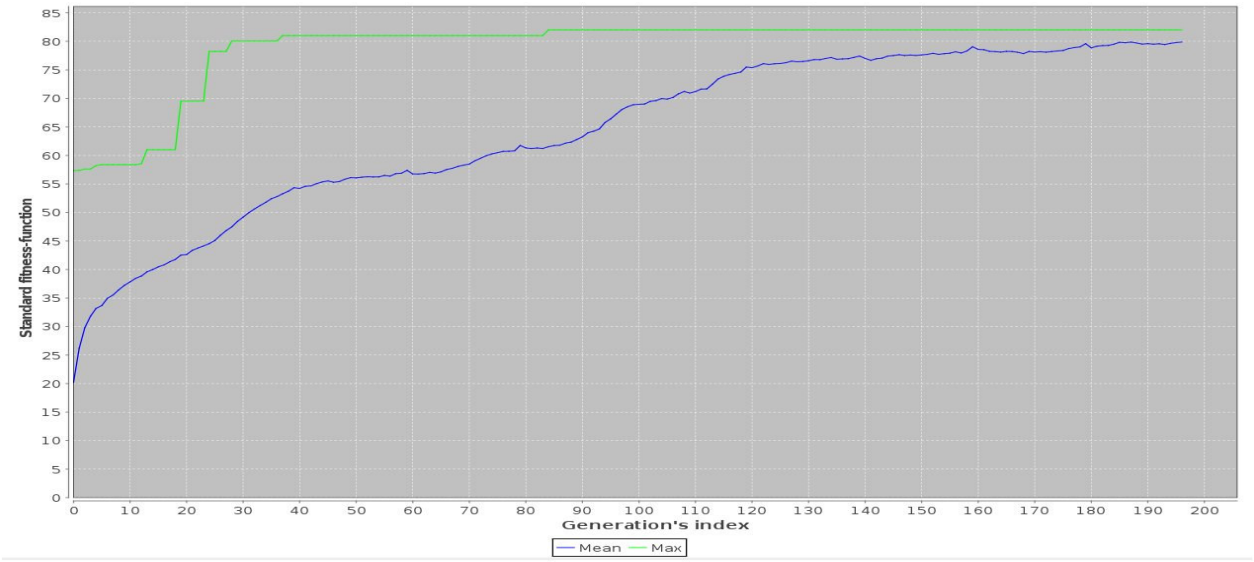
$$Fitness = Food - Steps / 200,$$

где ***Fitness*** – функция приспособленности, ***Food*** – количество съеденной еды, ***Steps*** – максимальный номер шага, на котором была съедена еда.

4. Результаты работы

В результате работы алгоритма за 195 ходов был получен автомат из пяти состояний, имеющий значение функции приспособленности 82,01. Муравей, управляемый данным автоматом съедает 83 яблока за 198 ходов.

4.1. График функции приспособленности



На рис. 3 приведен график максимального (max) и среднего (mean) значения функции приспособленности.**Рис. 3** График функции приспособленности

4.2 Диаграмма переходов полученного автомата

Полученный автомат Мили представлен на рис. 4 в виде диаграммы переходов. Входное воздействие описано символами T — «есть еда» и F — «нет еды». Выходное воздействие описывается символами R — «поворот направо», L — «поворот налево» и M — «шаг вперед».

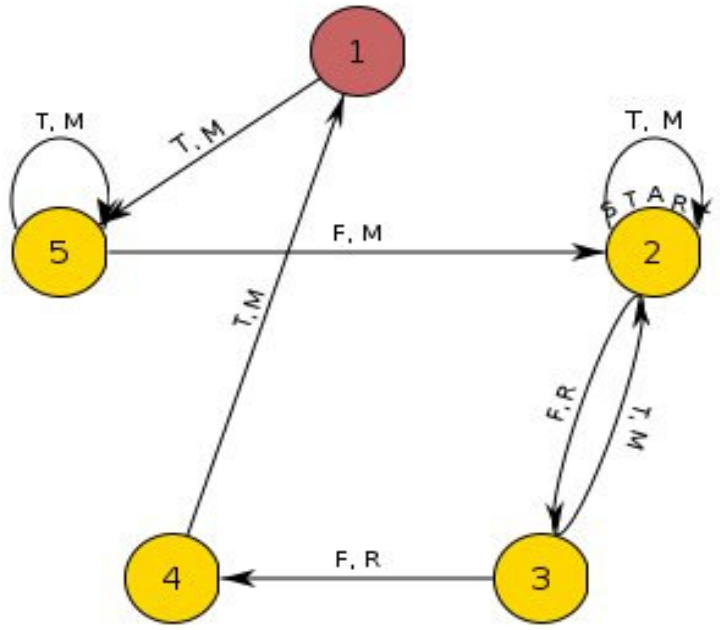


Рис. 4 Диаграмма переходов полученного автомата

Заключение

Результаты лабораторной работы показали, что используемые методы эффективны для построения управляющего автомата с пятью состояниями, который решает задачу об «Умном муравье», съедая 83 яблока, так как полный перебор позволяет строить автомат с таким же результатом. Помимо этого график функции приспособленности показывает, что неплохие результаты получаются уже на первых поколениях особей, что говорит об эффективности представления автомата и островного генетического алгоритма. При этом стоит заметить, что известен автомат, построенный для этой задачи вручную, который съедает 81 яблоко [2]. К недостаткам использованного подхода можно отнести большое время генерации очередного поколения, но это компенсируется тем, что в результате получен автомат, который как раз наиболее быстро решает данную задачу.

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

3. *Яминов Б.* Генетические алгоритмы

   http://rain.ifmo.ru/cat/view.php/theory/unsorted/genetic-2005

Приложение. Конфигурационные файлы

Листинг *Individual.properties*

```
external=5
internal=0
name=Mealy Automaton
task.name=Ant
description.file=description.html
name=Madagascar
```

Листинг *Algorithm.properties*

```
number.island=30
size.island=200
mutation.probability=0.1
size.elite=0.01
number.swap=0.015
period.swap=20
period.big.mutation=40
percent.kill.island=0.5
label.number.island=Number of islands
label.size.island=Island size
label.mutation.probability=Mutation probability
label.size.elite=Elite size
label.number.swap=Number of swap
label.period.swap=Period of swap
label.period.big.mutation=Period of big mutation
label.percent.kill.island=Percent of killed
title = Algorithm properties
width = 400
height = 400
title.operators = Select operator
operators.width = 300
operators.height = 250
description.file=description.html
```

Приложение. Исходные тексты программ

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);
                }

                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';
}


if (charAction0 == 'N') {
        bad = true;
}


if (strAction1.equalsIgnoreCase("01")) {
        charAction1 = 'L';
}

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

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


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;
```

```
}


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;
	}
```

```
                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 *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 *ConfigDialog.java*

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


import laboratory.util.gui.config.TextFieldsPanel;

import laboratory.util.gui.config.OkCancelPanel;

import laboratory.util.gui.config.Util;

import laboratory.util.loader.AbstractIndividualLoader;

import laboratory.plugin.individual.mealy.Config;

import laboratory.plugin.individual.mealy.MealyAutomaton;

import javax.swing.*;

import java.awt.*;

import java.awt.event.ActionListener;

import java.awt.event.ActionEvent;

import java.util.Properties;

public class ConfigDialog extends JDialog {


    public ConfigDialog(JFrame owner, final Properties p, final
AbstractIndividualLoader<MealyAutomaton> l) {

        super(owner, p.getProperty("title"));

        final TextFieldsPanel fieldsPanel = new TextFieldsPanel(new String[]{

                p.getProperty("label.external")/*, p.getProperty("label.internal")*/},

                new String[]{p.getProperty("external")/*, p.getProperty("internal")*/},

                new String[]{Integer.toString(Config.getInstance().getEx())/*,

                        Integer.toString(Config.getInstance().getIn())*/},

                new int[]{3/*, 3*/}, 1);


        getContentPane().add(fieldsPanel, BorderLayout.CENTER);

        getContentPane().add(new OkCancelPanel(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                String[] value = fieldsPanel.getValue();

                Config.getInstance().setEx(Integer.parseInt(value[0]));

                Config.getInstance().setIn(/*Integer.parseInt(value[1])*/0);

                showOperatorChooserDialog(p, l);

                dispose();

            }

        }, this), BorderLayout.SOUTH);

        Util.setSize(this, new Dimension(Integer.parseInt(p.getProperty("width")),
Integer.parseInt(p.getProperty("height"))));

    }

    private void showOperatorChooserDialog(Properties p, AbstractIndividualLoader<MealyAutomaton> l)
{

        final JDialog dialog = l.getOperatorChooser(this, p.getProperty("title.operators"));

        Util.setSize(dialog, new Dimension(Integer.parseInt(p.getProperty("operators.width")),
Integer.parseInt(p.getProperty("operators.height"))));

        Util.showModal(dialog);

    }

}
```

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 *IslandGA.java*

```
package laboratory.plugin.algorithm.madagascar;

import laboratory.common.genetic.Algorithm;
import laboratory.common.genetic.Individual;
import laboratory.common.genetic.IndividualFactory;
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.common.genetic.FitIndividual;
import laboratory.util.functional.Util;
import laboratory.util.functional.Functor1;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

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

    private final SimpleGA<I>[] islands;

    private int indexGeneration;

    private final int exchangeNumber;
    private final int exchangeTime;

    private final int bigMutationTime;
    private final double percentKilledIslands;
    private final double probabilityMutations;
    private final int eliteSize;

    private final Fitness<I> fitness;
    private final IndividualFactory<I> factory;
    private final Mutation<I> mutation;
    private final Crossover<I> crossover;
    private final Selection<I> selection;


    private Random r;

    public IslandGA(int numberIslands, int sizeIsland, double probabilityMutations,
                    double numberSwap, int periodSwap, int periodBigMutation,
                    double percentKilledIslands, double eliteSize,
```

```
                    IndividualFactory<I> factory, Mutation<I> mutation,
                    Crossover<I> crossover, Selection<I> selection, Fitness<I> fitness) {
        indexGeneration = 0;
        islands = new SimpleGA[numberIslands];
        this.eliteSize = (int)Math.round(sizeIsland*eliteSize);
        for(int i = 0;i < numberIslands;i++){
            islands[i] = new SimpleGA<I>(sizeIsland, probabilityMutations, factory, mutation,
crossover, selection, fitness, this.eliteSize);
        }
        this.exchangeNumber = (int)(numberSwap * sizeIsland);
        this.exchangeTime = periodSwap;
        this.bigMutationTime = periodBigMutation;
        this.percentKilledIslands = percentKilledIslands;
        this.fitness = fitness;
        this.probabilityMutations = probabilityMutations;
        this.factory = factory;
        this.mutation = mutation;
        this.crossover = crossover;
        this.selection = selection;

        r = new Random();
    }



    public List<I> getGeneration(){
        List<FitIndividual<I>> gen = new ArrayList<FitIndividual<I>>();
        for(Algorithm<I> ga : islands){
            gen.addAll(Util.map(ga.getGeneration(), new Functor1<I, FitIndividual<I>>() {
                public FitIndividual<I> apply(I i) {
                    return new FitIndividual<I>(i, fitness.apply(i));
                }
            }));
        }
        Collections.sort(gen);
        return Util.map(gen, new Functor1<FitIndividual<I>, I>(){
            public I apply(FitIndividual<I> i){
                return i.ind;
            }
        });
    }

    public void stop(){}



    public void nextGeneration(){
        indexGeneration++;
        for(Algorithm<I> ga : islands){
```

```
            ga.nextGeneration();
        }
        if(indexGeneration % exchangeTime == 0){
            int ni = islands.length;
            List<I>[] generation = new List[ni];
            for(int i = 0;i < ni;i++){
                generation[i] = islands[i].getGeneration();
            }
            int size = generation[0].size();
            for(int i = 0;i < ni;i++){
                for(int j = 0;j < exchangeNumber;j++){
                    generation[i].set(size - j - 1,
generation[r.nextInt(ni)].get(r.nextInt(generation[r.nextInt(ni)].size())));
                }
            }
            for(int i = 0;i < ni;i++){
                islands[i] = new SimpleGA<I>(generation[i], probabilityMutations, factory, mutation,
crossover, selection, fitness, eliteSize);
            }
        } else if(indexGeneration % bigMutationTime == 0){
            bigMutation();
        }
    }

    public void bigMutation(){
        for(SimpleGA<I> ga : islands){
            if(r.nextDouble() < percentKilledIslands){
                ga.bigMutation();
            }
        }
    }

}
```

Листинг *SimpleGA.java*

```
package laboratory.plugin.algorithm.madagascar;

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 final int eliteSize;

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

        this.probabilityMutation = probabilityMutation;
```

```
        this.factory = factory;

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

        this.generation = Util.map(generation, new Functor1<I, FitIndividual<I>>(){
            public FitIndividual<I> apply(I i){
                return cons(i);
            }
        });
        Collections.sort(this.generation);

        //ToDo: Rewrite this!!
        fixedPart = generation.size() / 2;
        this.eliteSize = eliteSize;




        r = new Random();
    }

    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,
                    final int eliteSize){
        this(Util.listFromFunctor(new Functor0<I>(){
                     public I apply(){
                         return factory.getIndividual();
                     }
                }, sizeGeneration),
             probabilityMutation, factory, mut, cross, sel, fitness, eliteSize);
    }

    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(){
        // System.err.println(generation.get(0).fitness);
        int size = generation.size();
        List<FitIndividual<I>> newGeneration = new ArrayList<FitIndividual<I>>(size);
        for (int i = 0; i < eliteSize; ++i)
            newGeneration.add(generation.get(i));
        // System.err.println(newGeneration.get(0).fitness);
        newGeneration.addAll(sel.apply(generation, fixedPart));
        while(newGeneration.size() + 1 <= generation.size()){
            List<I> s = cross.apply(Arrays.asList(winner(randomI(), randomI()), winner(randomI(),
randomI())));
            for(I ind : s){
                newGeneration.add(cons(ind));
            }
        }
        if(newGeneration.size() < size){
            newGeneration.add(cons(mut.apply(randomI().ind)));
        }
        for(int i = 0;i < size;i ++){
            if(r.nextDouble() < probabilityMutation){
                // newGeneration.set(i, cons(mut.apply(newGeneration.get(i).ind)));
            }
        }
        generation = newGeneration;
        Collections.sort(generation);
        // System.err.println(generation.get(0).fitness);
        // System.err.println();
    }
    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++){
            final I ind = factory.getIndividual();
            generation.set(i, new FitIndividual<I>(ind, fitness.apply(ind)));
        }
        Collections.sort(generation);
    }
}
```

Листинг *Config.java*

```
package laboratory.plugin.algorithm.madagascar;

import laboratory.util.loader.JarReader;
import laboratory.util.Parser;

import java.util.jar.JarFile;

public class Config {
    static int islandsNumber, islandSize, exchangeTime, bigMutationTime;
    static double mutationProbability, exchangeNumber, islandPercentKill, eliteSize;

    public static double getEliteSize(){
        return eliteSize;
    }

    public static void setEliteSize(double eliteSize){
        Config.eliteSize = eliteSize;
    }

    public static int getIslandsNumber(){
        return islandsNumber;
    }

    public static void setIslandsNumber(int islandsNumber){
        Config.islandsNumber = islandsNumber;
    }

    public static int getIslandSize(){
        return islandSize;
    }

    public static void setIslandSize(int islandSize){
        Config.islandSize = islandSize;
    }

    public static int getExchangeTime(){
        return exchangeTime;
    }

    public static void setExchangeTime(int time){
        Config.exchangeTime = time;
    }

    public static int getBigMutationTime(){
```

```
        return bigMutationTime;
    }

    public static void setBigMutationTime(int time){
        Config.bigMutationTime = time;
    }

    public static double getMutationProbability(){
        return mutationProbability;
    }

    public static void setMutationProbability(double mutationProbability){
        Config.mutationProbability = mutationProbability;
    }

    public static double getExchangeNumber(){
        return exchangeNumber;
    }

    public static void setExchangeNumber(double number){
        Config.exchangeNumber = number;
    }

    public static double getIslandPercentKill(){
        return islandPercentKill;
    }

    public static void setIslandPercentKill(double islandPercentKill){
        Config.islandPercentKill = islandPercentKill;
    }

    public static void setJar(JarFile jar) {
        Parser p = new Parser(JarReader.getProperties(jar, "algorithm.properties"));
        setIslandsNumber(p.getInt("number.island"));
        setIslandSize(p.getInt("size.island"));
        setMutationProbability(p.getDouble("mutation.probability"));
        setExchangeNumber(p.getDouble("number.swap"));
        setExchangeTime(p.getInt("period.swap"));
        setBigMutationTime(p.getInt("period.big.mutation"));
        setIslandPercentKill(p.getDouble("percent.kill.island"));
        setEliteSize(p.getDouble("size.elite"));
    }

}
```

Листинг *IslandConfigDialog.java*

```
package laboratory.plugin.algorithm.madagascar;


import laboratory.util.gui.config.TextFieldsPanel;
import laboratory.util.gui.config.OkCancelPanel;
import laboratory.util.loader.AbstractAlgorithmLoader;
//import laboratory.util.loader.JarReader;
import laboratory.util.gui.config.Util;


import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.util.Properties;


public class IslandConfigDialog extends JDialog {
    JFrame owner;


    public IslandConfigDialog(JFrame owner, final Properties p, final IslandGALoader<?> l) {
        super(owner, p.getProperty("title"));
       this.owner = owner;


        final TextFieldsPanel fieldsPanel = new TextFieldsPanel(new String[]{
                p.getProperty("label.number.island"), p.getProperty("label.size.island"),
p.getProperty("label.mutation.probability"), p.getProperty("label.size.elite"),
p.getProperty("label.number.swap"), p.getProperty("label.period.swap"),
p.getProperty("label.period.big.mutation"), p.getProperty("label.percent.kill.island") },
                new String[]{p.getProperty("number.island"), p.getProperty("size.island"),
p.getProperty("mutation.probability"), p.getProperty("size.elite"), p.getProperty("number.swap"),
p.getProperty("period.swap"), p.getProperty("period.big.mutation"),
p.getProperty("percent.kill.island")},
                new String[]{Integer.toString(Config.getIslandsNumber()),
                        Integer.toString(Config.getIslandSize()),
Double.toString(Config.getMutationProbability()), Double.toString(Config.getEliteSize()),
Double.toString(Config.getExchangeNumber()), Integer.toString(Config.getExchangeTime()),
Integer.toString(Config.getBigMutationTime()), Double.toString(Config.getIslandPercentKill())},
                new int[]{3, 3, 3, 3, 3, 3, 3, 3}, 8);


        getContentPane().add(fieldsPanel, BorderLayout.CENTER);


        getContentPane().add(new OkCancelPanel(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String[] value = fieldsPanel.getValue();
                Config.setIslandsNumber(Integer.parseInt(value[0]));
                Config.setIslandSize(Integer.parseInt(value[1]));
                Config.setMutationProbability(Double.parseDouble(value[2]));
               Config.setEliteSize(Double.parseDouble(value[3]));
               Config.setExchangeNumber(Double.parseDouble(value[4]));
               Config.setExchangeTime(Integer.parseInt(value[5]));
```

```
                Config.setBigMutationTime(Integer.parseInt(value[6]));
                Config.setIslandPercentKill(Double.parseDouble(value[7]));
                 showOperatorChooserDialog(p, l);
                 dispose();
            }
        }, this), BorderLayout.SOUTH);
        Util.setSize(this, new Dimension(Integer.parseInt(p.getProperty("width")),
Integer.parseInt(p.getProperty("height"))));
    }

    private void showOperatorChooserDialog(Properties p, AbstractAlgorithmLoader<?> l) {
        final JDialog dialog = l.getSelectionChooser(owner, p.getProperty("title.operators"));
        Util.setSize(dialog, new Dimension(Integer.parseInt(p.getProperty("operators.width")),
Integer.parseInt(p.getProperty("operators.height"))));
        Util.showModal(dialog);
    }

}
```

Листинг *IslandGALoader.java*

```
package laboratory.plugin.algorithm.madagascar;

import laboratory.common.genetic.Individual;
import laboratory.common.genetic.Algorithm;
import laboratory.common.genetic.IndividualFactory;
import laboratory.common.genetic.operator.Crossover;
import laboratory.common.genetic.operator.Mutation;
import laboratory.common.genetic.operator.Selection;
import laboratory.common.genetic.operator.Fitness;
import laboratory.util.loader.AbstractAlgorithmLoader;
import laboratory.util.loader.JarReader;
import laboratory.util.gui.config.Util;

import java.util.List;
import javax.swing.*;
import java.util.jar.JarFile;
import java.awt.*;
import java.io.File;


public class IslandGALoader<I extends Individual>
        extends AbstractAlgorithmLoader<I>{

    public IslandGALoader(JarFile file, File dir) {
        super(file, dir);
        Config.setJar(file);
    }

    @Override
    public String getMessage() {
        if (getSelections().isEmpty()) {
            return "Please, select only one selection strategy!";
        } else {
            return "OK";
        }
    }


    @Override
    public Algorithm<I> loadAlgorithm(List<IndividualFactory<I>> individualFactories
                                ,List<Crossover<I>> crossovers
                                ,List<Mutation<I>> mutations
                                ,List<Selection<I>> selections
                                ,List<Fitness<I>> functions) {
```

```
        return new IslandGA<I>(Config.getIslandsNumber()
                               ,Config.getIslandSize()
                               ,Config.getMutationProbability()
                               ,Config.getExchangeNumber()
                               ,Config.getExchangeTime()
                               ,Config.getBigMutationTime()
                               ,Config.getIslandPercentKill()
                               ,Config.getEliteSize()
                               ,individualFactories.get(0)
                               ,mutations.get(0)
                               ,crossovers.get(0)
                               ,getSelections().get(0)
                               ,functions.get(0));
    }

    public JDialog getConfigDialog(){
        //Todo: Implement this method.
        return null;
    }

    @Override
    public JDialog getConfigDialog(JFrame owner) {
       final JDialog d = new IslandConfigDialog(owner, JarReader.getProperties(getJar(),
"algorithm.properties"), this);
        return d;
    }
}
```