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

И. И. Чернявский

Отчёт по лабораторной работе «Построение управляющих автоматов с помощью генетических алгоритмов»

Вариант № 9

Санкт-Петербург
2009

Оглавление

Введение

В данной лабораторной работе изучается применение генетических алгоритмов для генерации конечных автоматов. В качестве примера взята задача об «Умном муравье». Результатом работы является автомат Мили, построенный с помощью генетического алгоритма и представляющий логику муравья.

При выполнении лабораторной работы использовалась программа «Виртуальная лаборатория» [1], написанная студентами кафедры «Компьютерные технологии» СПбГУ ИТМО и позволяющая реализовывать генетические алгоритмы и особи для них в виде плагинов.

1. Постановка задачи

Задача данной лабораторной работы – построить близкий к оптимальному автомат Мили, решающий задачу об «Умном муравье». Оптимальность заключается в том, что автомат должен иметь минимальное число состояний, и муравей, управляемый данным автоматом, должен съесть как можно больше еды, выполнив при этом как можно меньше шагов.

1.1. Автомат Мили

Автомат Мили – это конечный автомат, генерирующий свои выходные действия в зависимости от текущего состояния и входного сигнала. Пример диаграммы переходов автомата Мили приведён на рис. 1.

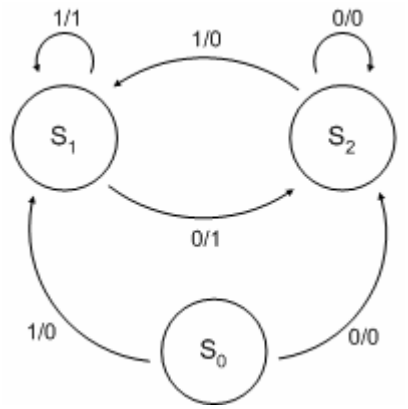

Рис. 1

Как видно из приведённой диаграммы, над каждой дугой расположена пара значений – входное и выходное воздействия, причём выходное действие зависит не только от состояния, в котором находится автомат, но и от входного воздействия.

1.2. Задача об «Умном муравье»

В задаче об «Умном муравье» [2] рассматривается поле, состоящее из клеток. Поле имеет размеры 32 на 32 клетки и располагается на поверхности тора. Некоторые клетки поля пусты, некоторые содержат по одному яблоку. Всего на поле 89 яблок. Муравей начинает своё движение из клетки, помеченной как «Start».

За один ход муравей может определить, есть ли в клетке перед ним яблоко, и выполнить одно из следующих действий:

- повернуть налево;
- повернуть направо;
- сделать шаг вперёд, и если в новой клетке есть яблоко, то съесть его;
- ничего не делать.

Максимальное число шагов – 200. Цель – создать муравья с фиксированным числом состояний, который за минимальное число шагов съест все яблоки.

2. Генетический алгоритм

Для задачи поиска оптимального автомата, управляющего муравьём, применяется генетический алгоритм [3]. Работа генетического алгоритма состоит из нескольких фаз. В начале происходит генерация первого поколения особей. Далее алгоритм начинает выполнение итеративного процесса – на каждой итерации алгоритм строит следующее поколение из предыдущего. При этом применяются операции:

- отбор – из предыдущего поколения выбирается часть особей, для сравнения особей между собой алгоритм использует функцию приспособленности – функцию, сопоставляющую каждой особи число, определяющее её приспособленность;
- скрещивание – по двум особям-родителям создаются две новые особи;
- мутация – случайным образом изменяется строение особи.

Опишем более подробно алгоритм, применяемый в данной работе.

2.1. Представление особи

Особями в данном алгоритме являются автоматы Мили. Автомат Мили представляется в виде графа переходов. Непосредственно в коде программы используется двумерный массив `Automaton.Transition[][] transitions`, который хранит переход (новое состояние и выходное действие) для каждой пары, состоящей из текущего состояния автомата и входного сигнала. В задаче об «Умном муравье» значениями входной переменной являются нуль и единица – присутствие или отсутствие еды в клетке перед муравьём.

2.2. Отбор особей и генерация нового поколения

Начальное поколение генерируется из особей, созданных случайным образом. Рассмотрим процесс генерации нового поколения из текущего. Для заполнения половины свободных мест в новом поколении алгоритм использует «метод рулетки»:

1. Для каждой особи рассчитывается вероятность её выбора по формуле

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j},$$

   где *fi* – значение функции приспособленности, *N* – число особей в поколении.
2. В соответствии с полученным распределением алгоритм случайным образом выбирает особь из текущего поколения и добавляет её в новое.

Для окончательного заполнения нового поколения особями алгоритм использует «турнирный метод» для выбора скрещиваемых особей:

1. Для турнира случайным образом выбираются две особи из текущего поколения и сравниваются значения функции приспособленности особей. Та особь, у которой значение больше, объявляется победителем.
2. Турнир проводится вновь, определяется еще один победитель.

3. Два победителя скрещиваются, полученные новые особи добавляются в новое поколение.

Используемый генетический алгоритм является «островным» – он работает не с одним поколением, а параллельно с несколькими поколениями, которые называются «островами». За один шаг алгоритм генерирует новое поколение на каждом острове. С заранее заданным периодом алгоритм выполняет дополнительный шаг – случайным образом переносит часть особей с одних «островов» на другие.

2.3. Метод скрещивания особей

При скрещивании алгоритм порождает две новые особи из двух особей-родителей. Состояния автоматов нумеруются числами от 1 до N (у всех автоматов одинаковое число состояний). Рассмотрим процедуру скрещивания – для всех i от 1 до N выполняются шаги:

1. Рассматривается состояние с номером i из первого родителя и состояние с тем же номером из второго; запоминаются переходы из этих состояний.
2. Рассматривается состояния с одинаковым номером i в автоматах-потомках; случайным образом переходы особей-родителей распределяются среди состояний особей-потомков.

2.4. Мутация особей

Мутация особи-автомата заключается в случайном выборе состояния и случайном изменении одного из переходов из данного состояния. Также с вероятностью 0.5 изменяется начальное состояние автомата на случайно выбранное.

2.5. Функция приспособленности

Функция приспособленности вычисляется по формуле

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

где *Apples* – число яблок, съедаемых муравьём за 200 шагов, *Steps* – номер шага, на котором муравей съедает последнее яблоко.

3. Построенный автомат

В результате работы генетического алгоритма был получен автомат Мили, который решает задачу об «Умном муравье». Автомат состоит из пяти состояний, значение функции приспособленности равно 81,01. Муравей, управляемый данным автоматом, съедает 82 яблока за 199 шагов.

3.1. Граф переходов

Для получения графа переходов использовался визуализатор «Виртуальной лаборатории»

(меню `Individual -> Show Best Individuals -> Ok`).

На рис. 2 представлен граф переходов полученного автомата. Обозначения на рисунке: T, F – присутствие и отсутствие еды, L, R, M – шаг влево, вправо, вперед.

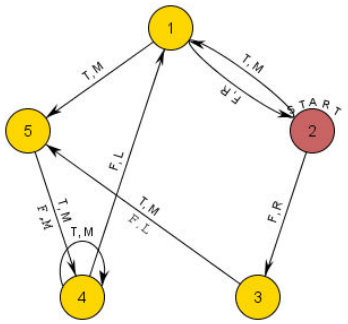

Рис. 2

3.2. График максимального значения функции приспособленности

График максимального значения функции приспособленности среди особей поколения в зависимости от номера поколения приведён на рис. 3.

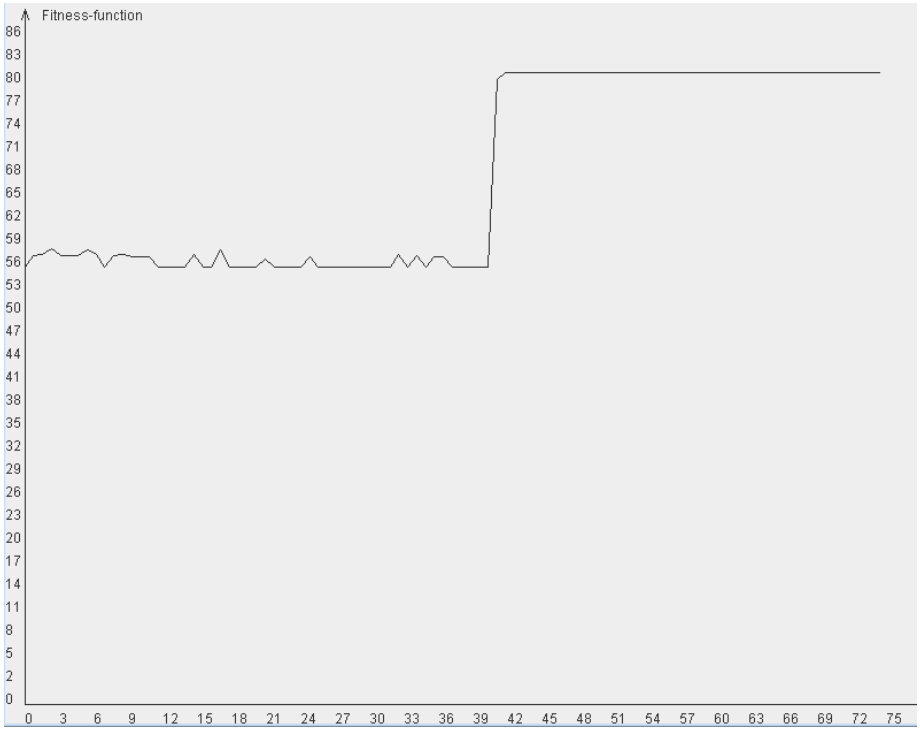

Рис. 3

3.3. График среднего значения функции приспособленности

График среднего значения функции приспособленности среди особей поколения в зависимости от номера поколения представлен на рис. 4.

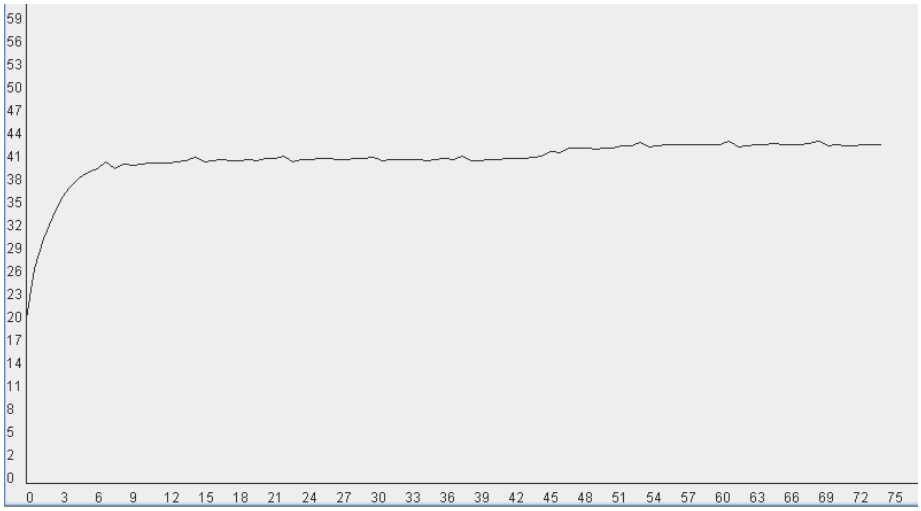

Рис. 4

Заключение

Результаты лабораторной работы показали, что используемые методы достаточно эффективны для построения автомата Мили с пятью состояниями, который решает задачу об «Умном муравье», съедая 82 яблока, так как полный перебор позволяет построить муравья, съедающего только на одно яблоко больше. При этом стоит заметить, что известен автомат, построенный для этой задачи вручную, который управляет муравьём, съедающим 81 яблоко [2].

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
4. *Царёв Ф. Н., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов с минимальным числом состояний для задачи об «Умном муравье» / Тезисы научно-технической конференции «Научно-программное обеспечение в образовании и научных исследованиях». СПбГУ ПУ. 2008, с. 209–215. http://is.ifmo.ru/download/2008-02-25_tsarev_shalyto.pdf

Приложение. Исходные коды

Файл MealyT9Automaton.java

```
package individual.mealyt9;

import laboratory.common.ga.Individual;
import task.ant.simple.individual.Automaton;
import task.ant.simple.Ant;
import task.ant.simple.SimpleAnt;
import task.ant.simple.individual.SimpleMover;

import java.util.Random;

public class MealyT9Automaton implements Automaton {

    private int initialState;
    private Automaton.Transition[][] transitions;
    private double fitness = Double.NEGATIVE_INFINITY;

    public MealyT9Automaton(int is, int ns) {
        initialState = is;
        transitions = new Transition[ns][2];
    }

    public MealyT9Automaton[] crossover(Individual parent, Random r) {
        MealyT9Automaton p = (MealyT9Automaton) parent;
        MealyT9Automaton[] res = new MealyT9Automaton[2];
        res[0] = new MealyT9Automaton(0, getNumberStates());
        res[1] = new MealyT9Automaton(0, getNumberStates());

        if (r.nextBoolean()) {
            res[0].initialState = p.initialState;
            res[1].initialState = initialState;
        } else {
            res[1].initialState = p.initialState;
            res[0].initialState = initialState;
        }
        for (int i = 0; i < transitions.length; i++) {
            int flag = r.nextInt(4);
            switch (flag) {
                case 0:
                    res[0].transitions[i][0] = p.transitions[i][0];
                    res[0].transitions[i][1] = transitions[i][1];
                    res[1].transitions[i][0] = transitions[i][0];
                    res[1].transitions[i][1] = p.transitions[i][1];
                    break;
                case 1:
                    res[0].transitions[i][0] = transitions[i][0];
                    res[0].transitions[i][1] = p.transitions[i][1];
                    res[1].transitions[i][0] = p.transitions[i][0];
                    res[1].transitions[i][1] = transitions[i][1];
                    break;
                case 2:
                    res[0].transitions[i][0] = p.transitions[i][0];
                    res[0].transitions[i][1] = p.transitions[i][1];
                    res[1].transitions[i][0] = transitions[i][0];
                    res[1].transitions[i][1] = transitions[i][1];
                    break;
```

```
                case 3:
                    res[0].transitions[i][0] = transitions[i][0];
                    res[0].transitions[i][1] = transitions[i][1];
                    res[1].transitions[i][0] = p.transitions[i][0];
                    res[1].transitions[i][1] = p.transitions[i][1];
                    break;
            }
        }
        return res;
    }

    public MealyT9Automaton mutate(Random r) {
        int ns = getNumberStates();
        MealyT9Automaton res;

        res = new MealyT9Automaton(getInitialState(), ns);
        for (int i = 0; i < ns; i++) {
            Automaton.Transition t;
            t = getTransition(i, 1);
            res.setTransition(i, 1, new Transition(t.getEndState(),
t.getAction()));
            t = getTransition(i, 0);
            res.setTransition(i, 0, new Transition(t.getEndState(),
t.getAction()));
        }

        for (int i = 0; i < ns / 3; i++) {
            if (r.nextBoolean()) {
                int state = r.nextInt(ns);
                int c = r.nextBoolean() ? 1 : 0;
                int nstate = r.nextInt(ns);
                char nc =
Ant.ACTION_VALUES[r.nextInt(Ant.ACTION_VALUES.length)];
                res.setTransition(state, c, new Transition(nstate, nc));
            }
        }

        if (r.nextBoolean())
            res.setInitialState(r.nextInt(ns));

        return res;
    }

    public double fitness() {
        if (fitness == Double.NEGATIVE_INFINITY) {
            SimpleMover mover = new SimpleMover(this);
            mover.restart(new SimpleAnt());
            int count = 0;
            int lem = 0;
            for (int i = 0; i < Ant.NUMBER_STEPS; i++) {
                if (mover.move()) {
                    count++;
                    lem = i;
                }
                if (count == Ant.NUMBER_FOOD) {
                    break;
                }
            }
            fitness = (count - lem * 1.0 / Ant.NUMBER_STEPS);
        }
        return fitness;
    }

    public int getInitialState() {
```

```
        return initialState;
    }

    public void setInitialState(int is) {
        initialState = is;
    }

    public int getNumberStates() {
        return transitions.length;
    }

    public Automaton.Transition getTransition(int i, int c){
        return transitions[i][c];
    }

    public void setTransition(int i, int c, Automaton.Transition t){
        transitions[i][c] = t;
    }

    public Automaton getNestedAutomaton() {
        return null;
    }

    public int compareTo(Individual i) {
        return Double.compare(i.fitness(), fitness());
    }

    public Object[] getAttributes() {
        return new Object[] {this};
    }

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

    public static class Transition implements Automaton.Transition {

        private final int endState;

        public int getEndState() {
            return endState;
        }

        private final char action;

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

        public char getAction() {
            return action;
        }

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

Файл MealyT9IndividualFactoryLoader.java

```
package individual.mealyt9;
```

```
import individual.mealyt9.factory.MealyT9AutomatonFactory;
import laboratory.common.Loader;
import laboratory.common.ga.IndividualFactory;
import laboratory.util.Parser;

import java.util.jar.JarFile;
import java.util.jar.JarEntry;
import java.util.Properties;
import java.io.IOException;

public class MealyT9IndividualFactoryLoader implements
Loader<IndividualFactory> {

    private final Parser properties;

    public MealyT9IndividualFactoryLoader(JarFile file) {
        Properties in = new Properties();
        try {
            JarEntry ent = new JarEntry("mealyT9Automaton.conf");
            in.load(file.getInputStream(ent));
        } catch (IOException e) {
            e.printStackTrace();
        }
        properties = new Parser(in);
    }

    public IndividualFactory load(Object... args){
        return new
MealyT9AutomatonFactory(properties.getInt("numberofstates"));
    }

    public Properties getProperties() {
        return properties.getProperties();
    }
}
```

Файл MealyT9AutomatonFactory.java

```
package individual.mealyt9.factory;

import individual.mealyt9.MealyT9Automaton;
import task.ant.simple.Ant;
import laboratory.common.ga.IndividualFactory;
import java.util.Random;

public class MealyT9AutomatonFactory implements IndividualFactory {
    private final int ns;
    private final Random r;

    public MealyT9AutomatonFactory(int ns) {
        this.ns = ns;
        r = new Random();
    }

    public MealyT9Automaton randomIndividual() {
        MealyT9Automaton a = new MealyT9Automaton(r.nextInt(ns), ns);
        for (int i = 0; i < ns; i++) {
            a.setTransition(i, 1, new
MealyT9Automaton.Transition(r.nextInt(ns), Ant.ACTION_VALUES[r.nextInt(3)]));
            a.setTransition(i, 0, new
MealyT9Automaton.Transition(r.nextInt(ns), Ant.ACTION_VALUES[r.nextInt(3)]));
        }
        return a;
    }
```

```
}
```

Файл SimpleFPSGA.java

```
package ga.simplefps;

import laboratory.common.ga.GA;
import laboratory.common.ga.Individual;
import laboratory.common.ga.IndividualFactory;

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

public class SimpleFPSGA implements GA {

    private List<Individual> generation;

    private final double probabilityMutation;

    private final IndividualFactory factory;

    private final Random r;

    public void nextGeneration() {
        int size = generation.size();

        double[] prs = new double[size];
        double[] sums = new double[size + 1];
        sums[0] = 0;
        double sum = 0;
        for (int i = 0; i < size; i++) {
            prs[i] = generation.get(i).fitness();
            sums[i+1] = sums[i] + prs[i];
            sum += prs[i];
        }
        for (int i = 0; i < size; i++) {
            prs[i] /= sum;
            sums[i] /= sum;
        }
        sums[size] = 1;

        List<Individual> tmpGeneration = new ArrayList<Individual>(size);

        for(int i = 0; i < size; i++) {
            double p = r.nextDouble();
            int left = 0;
            int right = size;

            while (right - left > 1) {
                int x = (left + right) / 2;

                if (p < sums[x]) {
                    right = x;
                } else {
                    left = x;
                }
            }

            tmpGeneration.add(generation.get(right - 1));
        }

        List<Individual> newGeneration = new ArrayList<Individual>(size);
```

```
        for (int i = 0; i < (size / 2); i++) {
            Individual a1, a2;
            a1 = tmpGeneration.get(r.nextInt(size));
            a2 = tmpGeneration.get(r.nextInt(size));
            Individual p = (a1.compareTo(a2) < 0) ? a1 : a2;
            a1 = tmpGeneration.get(r.nextInt(size));
            a2 = tmpGeneration.get(r.nextInt(size));
            Individual[] s = p.crossover((a1.compareTo(a2) < 0) ? a1 : a2,
r);
            newGeneration.add(s[0]);
            newGeneration.add(s[1]);
        }

        if (newGeneration.size() < size) {
            newGeneration.add(tmpGeneration.get(r.nextInt(size)).mutate(r));
        }

        for (int i = 0; i < size; i++) {
            if (r.nextDouble() < probabilityMutation) {
                newGeneration.set(i, newGeneration.get(i).mutate(r));
            }
        }
        generation = newGeneration;
        Collections.sort(generation);
    }

    public List<Individual> getGeneration() {
        return generation;
    }

    public SimpleFPSGA(int sizeGeneration, double probabilityMutation,
IndividualFactory factory) {
        this.probabilityMutation = probabilityMutation;
        generation = new ArrayList<Individual>(sizeGeneration);
        for (int i = 0; i < sizeGeneration; i++) {
            generation.add(factory.randomIndividual());
        }
        Collections.sort(generation);
        this.factory = factory;
        r = new Random();
    }

    public void bigMutation() {
        for (int i = 0; i < generation.size(); i++) {
            generation.set(i, factory.randomIndividual());
        }
        Collections.sort(generation);
    }

    public Individual getBest() {
        return generation.get(0);
    }
}
```

Файл SimpleFPSGALoader.java

```
ackage ga.simplefps;

import laboratory.common.Loader;
import laboratory.common.ga.GA;
import laboratory.common.ga.IndividualFactory;
import laboratory.util.Parser;

import java.io.IOException;
```

```
import java.util.Properties;
import java.util.jar.JarEntry;
import java.util.jar.JarFile;


public class SimpleFPSGALoader implements Loader<GA> {
    private final Parser properties;

    public GA load(Object... args) {
        return new SimpleFPSGA(properties.getInt("sizeGeneration"),
                properties.getDouble("probabilityMutation"),
                (IndividualFactory) args[0]);
    }

    public SimpleFPSGALoader(JarFile file) {
        Properties in = new Properties();
        try {
            JarEntry ent = new JarEntry("simpleFPSGA.conf");
            in.load(file.getInputStream(ent));
        } catch (IOException e) {
            e.printStackTrace();
        }
        properties = new Parser(in);
    }

    public Properties getProperties() {
        return properties.getProperties();
    }
}
```

Файл IslandFPSGA.java

```
package ga.islandfps;

import ga.simplefps.SimpleFPSGA;
import laboratory.common.ga.GA;
import laboratory.common.ga.Individual;
import laboratory.common.ga.IndividualFactory;

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

public class IslandFPSGA implements GA {

    private final GA[] islands;

    private int indexGeneration;

    private final int numberSwap;

    private final int periodSwap;

    private final int periodBigMutation;

    private final double percentKilledIslands;

    private Random r;

    public List<Individual> getGeneration() {
        List<Individual> gen = new ArrayList<Individual>();
        for (GA ga : islands) {
            gen.addAll(ga.getGeneration());
```

```
        }
        Collections.sort(gen);
        return gen;
    }

    public IslandFPSGA(int numberIslands, int sizeIsland, double
probabilityMutations,
                       IndividualFactory factory, double numberSwap, int
periodSwap, int periodBigMutation,
                       double percentKilledIslands) {
        indexGeneration = 0;
        islands = new GA[numberIslands];
        for (int i = 0; i < numberIslands; i++) {
            islands[i] = new SimpleFPSGA(sizeIsland, probabilityMutations,
factory);
        }
        this.numberSwap = (int) (numberSwap * sizeIsland);
        this.periodSwap = periodSwap;
        this.periodBigMutation = periodBigMutation;
        this.percentKilledIslands = percentKilledIslands;
        r = new Random();
    }

    public void nextGeneration() {
        indexGeneration++;
        for (GA ga : islands) {
            ga.nextGeneration();
        }
        if (indexGeneration % periodSwap == 0) {
            int ni = islands.length;
            List<Individual>[] 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 < numberSwap; j++) {
                    generation[i].set(size - j - 1,
generation[r.nextInt(ni)].get(r.nextInt(size)));
                }
            }
            for (int i = 0; i < ni; i++) {
                Collections.sort(generation[i]);
            }
        } else if (indexGeneration % periodBigMutation == 0) {
            bigMutation();
        }
    }

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

    public Individual getBest() {
        Individual best = islands[0].getBest();
        for (int i = 1; i < islands.length; i++) {
            Individual b = islands[i].getBest();
            if (b.fitness() > best.fitness()) {
                best = b;
            }
```

```
        }
        return best;
    }
}
```

Файл IslandFPSGALoader.java

```
package ga.islandfps;

import laboratory.common.Loader;
import laboratory.common.ga.GA;
import laboratory.common.ga.IndividualFactory;
import laboratory.util.Parser;
import java.io.IOException;
import java.util.Properties;
import java.util.jar.JarEntry;
import java.util.jar.JarFile;

public class IslandFPSGALoader implements Loader<GA> {
    private final Parser properties;

    public IslandFPSGALoader(JarFile file) {
        Properties in = new Properties();
        try {
            JarEntry ent = new JarEntry("islandFPSGA.conf");
            in.load(file.getInputStream(ent));
        } catch (IOException e) {
            e.printStackTrace();
        }
        properties = new Parser(in);
    }

    public GA load(Object... args) {
        return new IslandFPSGA(properties.getInt("numberIslands"),
properties.getInt("sizeIsland"),
                properties.getDouble("probabilityMutation"),
                (IndividualFactory) args[0],
properties.getDouble("numberSwap"), properties.getInt("periodSwap"),
                properties.getInt("periodBigMutation"),
properties.getDouble("percentKillIsland"));
    }

    public Properties getProperties() {
        return properties.getProperties();}}
```