Санкт-Петербургский государственный университет
информационный технологий, механики и оптики

С. И. Попов

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

Вариант 13

Санкт-Петербург
2009

Оглавление

Ведение

В лабораторной работе изучается применение генетических алгоритмов для построения конечных автоматов Мили. В качестве примера применения автоматов рассматривается задача об «Умном муравье-3».

В рамках работы реализован плагин (программный компонент-прибавка к основной программе, позволяющий реализовать дополнительные функции) к «Виртуальной лаборатории», написанной студентами кафедры «Компьютерных технологий» СПбГУ ИТМО [1]. Для того чтобы визуализатор действий автомата, реализуемого в плагине, стал доступен пользователю, необходимо в конфигурационном файле эмулятора в «Виртуальной лаборатории» в параметр `filter` дописать строку `bitMealy`.

1. Постановка задачи

При выполнении лабораторной работы необходимо было построить с помощью генетических алгоритмов конечный автомат Мили, решающий задачу об «Умном муравье-3». При этом было необходимо применить представление автомата с помощью битовых строк, а способ скрещивания выбирался самостоятельно. При этом требовалось использовать традиционный генетический алгоритм и метод «рулетки» для генерации очередного поколения [2].

1.1. Задача об «Умном муравье-3»

В задаче об «Умном муравье-3» используется двумерный тор размером 32 на 32 (рис. 1). На некоторых клетках поля случайным образом расположены яблоки [3]. Процент заполнения поля задается пользователем, и предполагается, что муравей вначале находится в левом верхнем углу поля.

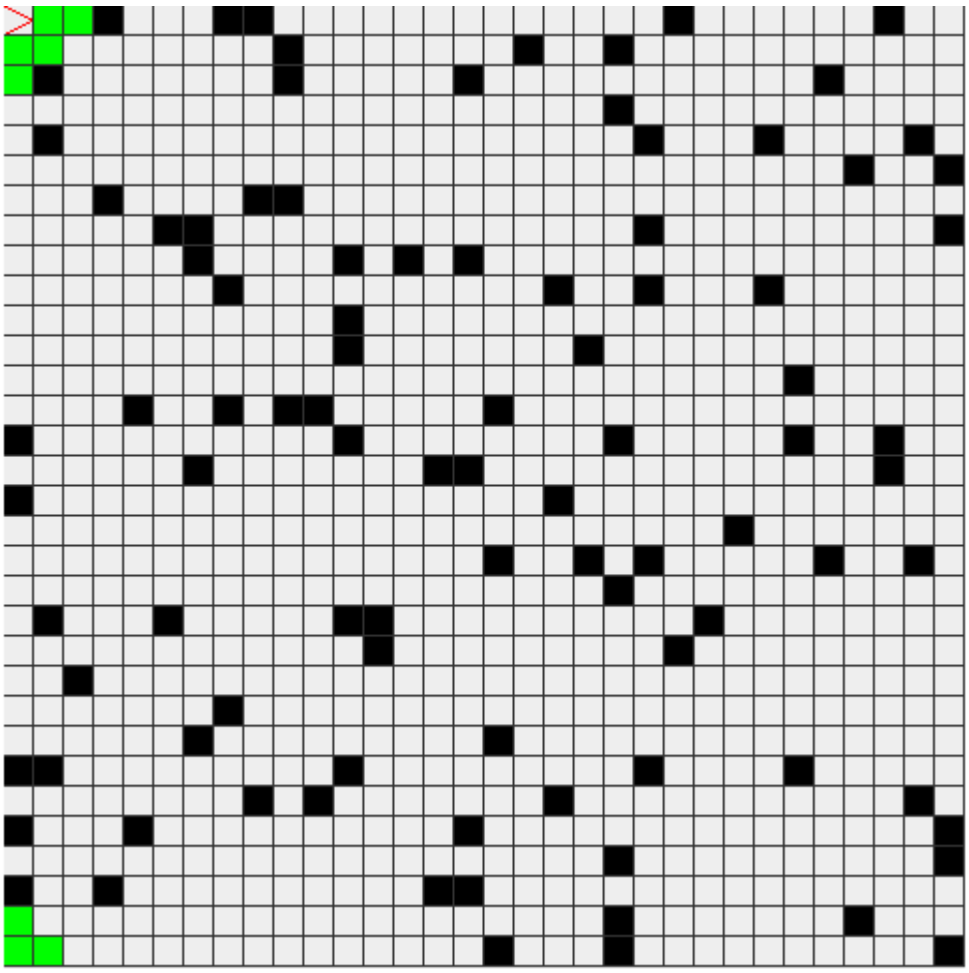

Рис. 1. Поле для муравья

Муравей «видит» восемь клеток (две слева, две справа, две спереди и по одной вперед по диагонали). За ограниченное (200) число ходов он должен съесть как можно больше яблок. Ход – поворот налево или поворот направо, или перемещение на одну клетку вперед.

2. Автомат Мили

Для муравья с помощью генетического алгоритма генерируется автомат Мили. Конечным детерминированным автоматом Мили называется совокупность пяти объектов:

A = (S, X, Y, δ, λ), где S, X и Y – конечные непустые множества, а δ и λ − отображения вида:

$\delta : S \times X \to S$ и

$\lambda : S \times X \to Y$

со связью элементов множеств S, X и Y в абстрактном времени T = {0, 1, 2, …} уравнениями:

s(t + 1) = δ(s(t), x(t))

y(t) = λ(s(t), x(t)), t из T.

Отображение δ называется функцией переходов автомата A, а отображение λ — функцией выходов автомата A.

2.1. Способ представления автомата

В работе автомат представлен в виде битовой строки. В ней под каждой состояние выделяется 256 * (k + 2) бит, где k – необходимое число битов для двоичного представления числа вершин в автомате [3]. Два бита отводятся под одно из трех возможных выходных действий автомата (10 – поворот муравья налево, 01 – поворот направо, 11 – перемещение на одну клетку вперед). 256 – число различных возможных входных воздействий (муравей видит восемь клеток, в каждой яблоко либо есть, либо его нет). Для хранения номера начального состояния автомата выделена переменная.

Приведем пример. Пусть автомат имеет 10 состояний, тогда один из его переходов будет выглядеть так:

0110 01 – перейти в состояние 6, при этом муравей должен повернуться направо. Под номер состояния отводятся необходимые четыре бита.

3. Генетический алгоритм

Генетический алгоритм – эвристический алгоритм нахождения максимума многопараметрической функции, по способу приближения к решению напоминающий биологическую эволюцию. В алгоритме используются три стадии отбора лучших полученных решений [4]:

1) сначала неким образом выбирается n лучших на данный момент решений;

2) затем из них специальным методом скрещивания получают n новых решений;

3) каждое из полученных решений затем с некоторой вероятностью мутирует.

3.1. Метод скрещивания

При вызове метода скрещивания, создаются две новые особи-автомата. Случайным взаимоисключающим образом каждому из потомков достается начальное состояние автомата-родителя.

Затем каждое из состояний предков случайным взаимоисключающим образом достается одному из потомков. В итоге получаются два автомата-потомка. В приложении приведен код на языке *Java*.

3.2. Оператор мутации

В операторе мутации сначала с заданной вероятностью (0,33) выбирается новое начальное состояние автомата Мили. Далее из всех состояний случайно выбирается одно. У него с вероятностью 50% выбирается, что будет мутировать: либо выходное воздействие, либо переход в следующее состояние.

Если выбрано изменение выходного действия, то случайным образом выбирается одно из 256 входных воздействий и для него случайным образом выбирается выходное воздействие из рассматриваемого состояния.

Если же выбрано изменение следующего состояния, то случайным образом выбирается одно из 256 входных воздействий и в качестве перехода автомата при нем случайно выбирается следующее состояние. В приложении приведен код на языке *Java*.

3.3. Метод вычисления функции приспособленности

Для вычисления функции приспособленности конкретный автомат запускается заданное пользователем число раз. При этом запоминается, сколько яблок успел съесть муравей за отведенные ему 200 шагов. Число съеденных яблок за все запуски суммируется и делится на число совершенных попыток. В итоге получается функция приспособленности данного автомата.

3.4. Метод генерации следующего поколения

Для генерации очередного поколения в работе используется метод рулетки [4], в котором вероятность каждой особи-автомата попасть на этап скрещивания пропорциональна ее функции приспособленности.

В программе каждой особи отводится соответствующий ее функции приспособленности промежуток на отрезке [0, 1]. Далее `n` (`n` – число особей в поколении) раз генерируется случайная величина на отрезке [0, 1]. На этап скрещивания попадает та особь, в чей промежуток попала случайная величина. Далее `n/2` раз из получившихся особей выбираются две пары, в каждой паре выбирается автомат с лучшей функцией приспособленности. Победители из пар скрещиваются методом, описанным выше.

После этапа скрещивания каждая из получившихся особей с заданной (0,02) вероятностью мутирует. В приложении приведен код на языке *Java*.

4. Полученные графики зависимости среднего и максимального значений функции приспособленности в зависимости от поколений

«Виртуальная лаборатория», к которой был написан плагин, получает значения функций приспособленности каждого автомата из каждого поколения и строит графики зависимости максимального и среднего значений функции от номера поколения.

Из графиков видно, что при использованных параметрах генетического алгоритма (200 особей в одном поколении, мутация автомата с вероятностью 0.02, один запуск автомата для получения функции приспособленности) максимальное значение этой функции 47 (рис. 2), что неплохо для рассматриваемой задачи «Умный муравей-3». Среднее же ее значение около 23-х (рис. 3).

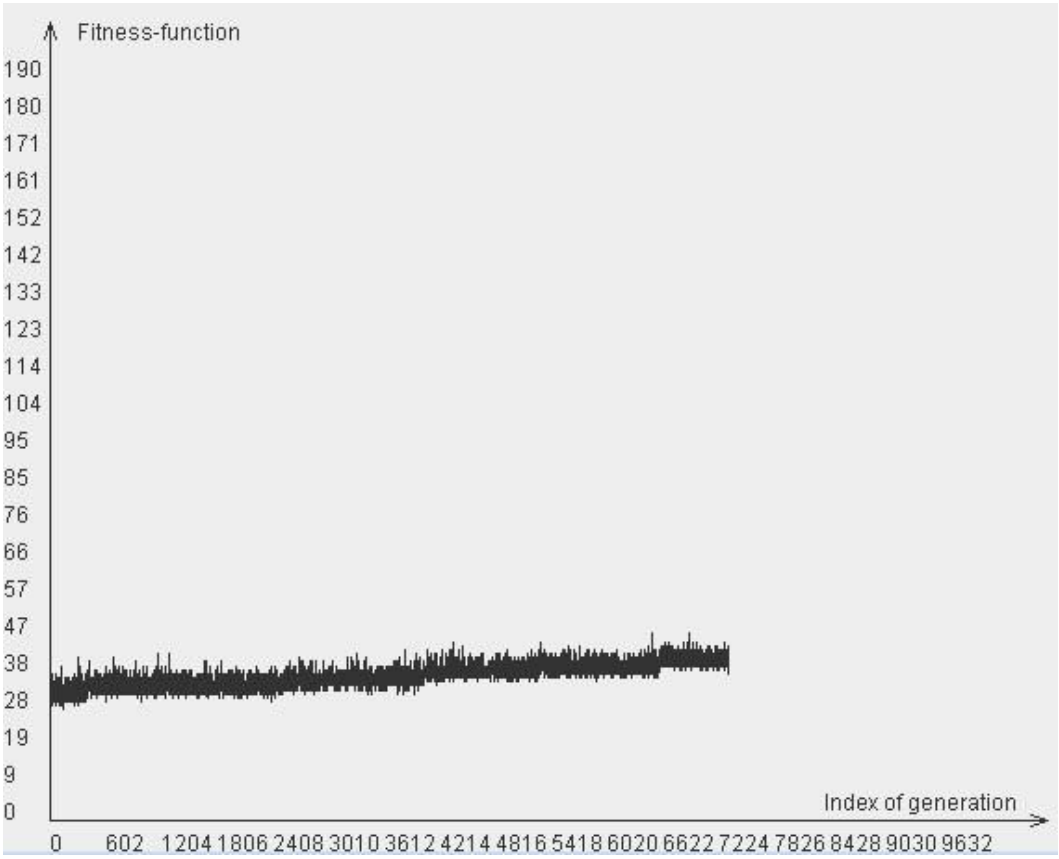

Рис. 2. График зависимости максимально значения функции приспособленности от номера поколения

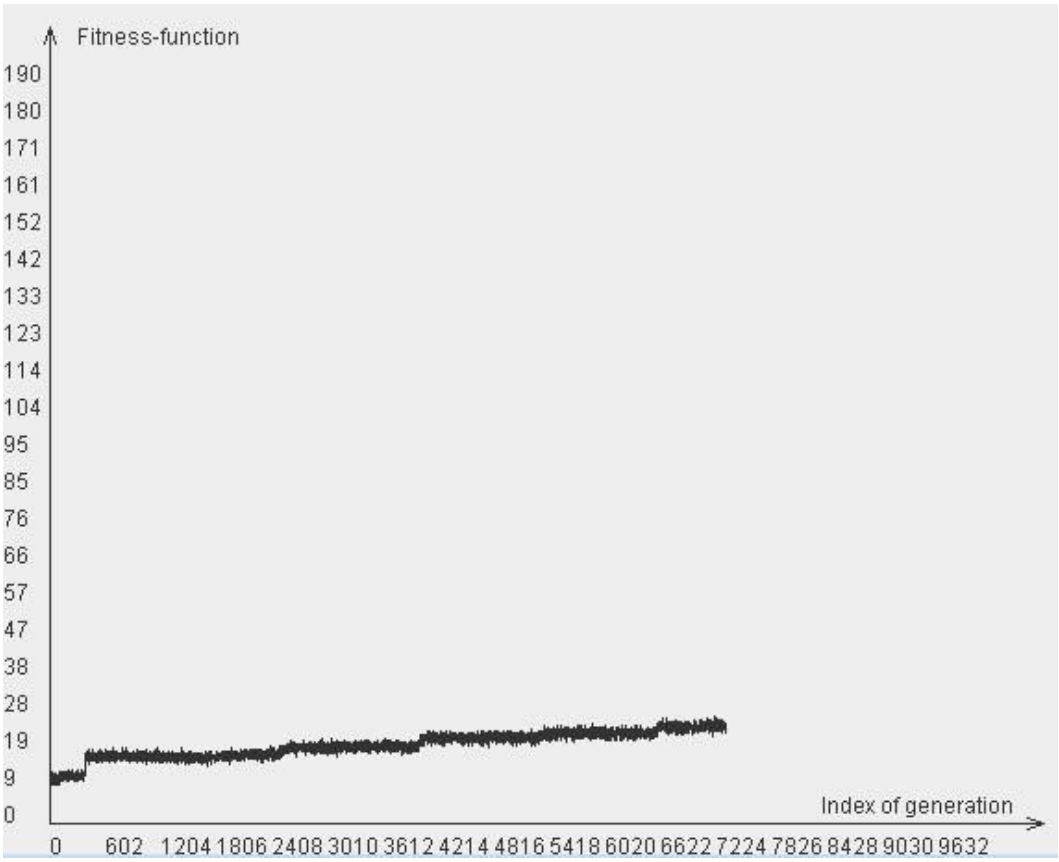

Рис. 3. График зависимости среднего значения функции приспособленности от номера поколения

Заключение

Полученные графики подтверждают правильность написанных алгоритмов в плагине. Значение функции приспособленности 47 можно считать неплохим результатом, учитывая то, что яблоки расположены на поле в случайном порядке. Работы по генерации автоматов Мили для задачи об «Умном муравье-3» автору неизвестны.

Источники

1. *Инструкция* по созданию плагинов к виртуальной лаборатории. http://svn2.assembla.com/svn/not_instrumental_tool/docs/pdf/interface_manual.pdf
2. *Условие* лабораторной работы. http://svn2.assembla.com/svn/not_instrumental_tool/docs/pdf/labs.pdf
3. *Бедный Ю. Д., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». http://is.ifmo.ru/works/_ant.pdf
4. *Яминов Б.* Генетические алгоритмы. http://rain.ifmo.ru/cat/view.php/theory/unsorted/genetic-2005

Приложение. Исходный код

Файл RouletteGA.java

```
package ga.roulette;

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;

/**
 * @author Sergey Popov.
 *
 * Генератор следующего поколения.
 */
public class RouletteGA implements GA {

    /**
     * Список с текущим поколением.
     */
    private List<Individual> generation;

    /**
     * Вероятность мутации особи.
     */
    private final double probabilityMutation;

    /**
     * Фабрика для автомата.
     */
    private final IndividualFactory factory;

    /**
     * Генератор случайных чисел.
     */
    private final Random r;

    /**
     * Генерация следующего поколения.
     */
    public void nextGeneration() {
        int size = generation.size();
        List<Individual> tmpGeneration = new
ArrayList<Individual>(size);
        double roulette[] = new double[size + 1];
        roulette[0] = 0;
        for (int i = 0; i < size; i++) {
```

```
            roulette[i + 1] = roulette[i] +
generation.get(i).fitness();
        }
        for (int i = 1; i <= size; i++) {
            roulette[i] /= roulette[size];
        }
        double arrow;
        int left, right, middle;
        for (int i = 0; i < size; i++) {
            arrow = r.nextDouble();
            left = 0;
            right = size;
            while (right - left > 1) {
                middle = (left + right) / 2;
                if (arrow < roulette[middle]) {
                    right = middle;
                } else {
                    left = middle;
                }
            }
            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);
    }

    /**
     * Возвращает текущее поколение.
     * @return текущее поколение.
```

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

    /**
     * Конструктор.
     * @param sizeGeneration размер поколения.
     * @param factory фабрика для автомата.
     */
    public RouletteGA(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();
    }

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

    /**
     * Возвращает лучшую особь.
     * @return лучшая особь.
     */
    public Individual getBest() {
        return generation.get(0);
    }
}
```

Файл RouletteGALoader.java

```
package ga.roulette;

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;
```

```
/**
 * @author Sergey Popov.
 *
 * Загрузчик для генератора следующего поколения.
 */
public class RouletteGALoader implements Loader<GA> {

    /**
     * Парсер.
     */
    private final Parser properties;

    /**
     * Загружает параметры для генератора следующего поколения.
     * @param args аргументы.
     * @return генератор следующего поколения.
     */
    public GA load(Object... args) {
        return new
RouletteGA(properties.getInt("sizeGeneration"),
                properties.getDouble("probabilityMutation"),
                (IndividualFactory) args[0]);
    }

    /**
     * Конструктор.
     * @param file JarFile.
     */
    public RouletteGALoader(JarFile file) {
        Properties in = new Properties();
        try {
            JarEntry ent = new JarEntry("rouletteGA.conf");
            in.load(file.getInputStream(ent));
        } catch (IOException e) {
            e.printStackTrace();
        }
        properties = new Parser(in);
    }

    /**
     * Возвращает Properties.
     * @return Properties.
     */
    public Properties getProperties() {
        return properties.getProperties();
    }
}
```

Файл BitMealyAnt.java

```
package individual.bitmealy;

import laboratory.common.ga.Individual;
```

```
import laboratory.common.Visualizable;

import java.util.Random;

import task.ant.extended.Ant;
import task.ant.extended.ExtendedAnt;

/**
 * @author Sergey Popov.
 *
 * Автомат Мили, представляемый битовой строкой.
 */

public class BitMealyAnt implements Visualizable {

    /**
     * Вероятность смены начального состояния автомата.
     */
    public static double pmis = 0.33;

    /**
     * Коэффициент заполненности поля яблоками.
     */
    public final double mu;

    /**
     * Функция приспособленности автомата.
     */
    private double fitness = Double.NEGATIVE_INFINITY;

    /**
     * Номер начального состояния автомата.
     */
    public final int is;

    /**
     * Число состояний в автомате.
     */
    public int cs;

    /**
     * Число съеденных яблок.
     */
    private int cf = 0;

    /**
     * Битовая строка для представления автомата.
     */
    public StringOfBits bitAutom;

    /**
     * Число входных воздействий.
     */
```

```
    public final int ct = 256;

    /**
     * Битовая строка для представления автомата.
     */
    public class StringOfBits {

        /**
         * Массив битов для представления автомата.
         */
        public boolean[] bitArray;

        /**
         * Число битов под действие муравья.
         */
        private final int bitsInAct = 2;

        /**
         * Число битов под хранение номера состояния.
         */
        private int bitsInNum = 31;

        /**
         * Число для перевода из двоичной системы счисления
         * в десятеричную и наоборот.
         */
        private int power = Integer.MAX_VALUE / 2 + 1;

        /**
         * Определение, сколько битов нужно для автомата.
         * @param cs число состояний автомата.
         */
        public StringOfBits(int cs) {
            if (cs < 1) throw new RuntimeException("Incorrect
arguments!");
            while ((cs - 1) / power != 1) {
                power >>= 1;
                bitsInNum--;
            }
            bitArray = new boolean[cs * (bitsInNum + bitsInAct)
* ct];
        }

        /**
         * Запись действия муравья в битовую строку.
         * @param ns номер состояния.
         * @param nt номер входного воздействия.
         * @param act записываемое действие муравья.
         */
        public void setAct(int ns, int nt, Ant.Action act) {
            if (ns > cs || nt >= ct || ns < 0 || nt < 0) {
                throw new RuntimeException("Incorrect
arguments!");
```

```
            }
            int begin = ns * (bitsInNum + bitsInAct) * ct
                    + (bitsInNum + bitsInAct) * nt + bitsInNum;
            if (act == Ant.Action.L) {
                bitArray[begin] = true;
                bitArray[begin + 1] = false;
            } else if (act == Ant.Action.R) {
                bitArray[begin] = false;
                bitArray[begin + 1] = true;
            } else if (act == Ant.Action.M) {
                bitArray[begin] = true;
                bitArray[begin + 1] = true;
            } else {
                throw new RuntimeException("Incorrect
arguments!");
            }
        }

        /**
         * Получает действие муравья.
         * @param ns номер состояния.
         * @param nt номер входного воздействия.
         * @return действие муравья.
         */
        public Ant.Action getAct(int ns, int nt) {
            if (ns > cs || nt >= ct || ns < 0 || nt < 0) {
                throw new RuntimeException("Incorrect
arguments!");
            }
            int begin = ns * (bitsInNum + bitsInAct) * ct
                    + (bitsInNum + bitsInAct) * nt + bitsInNum;
            if (begin < 0) throw new RuntimeException("Incorrect
arguments!");
            if (bitArray[begin]) {
                if (!bitArray[begin + 1]) {
                    return Ant.Action.L;
                } else {
                    return Ant.Action.M;
                }
            } else if ((!bitArray[begin])
                && (bitArray[begin + 1])) {
                    return Ant.Action.R;
            }
            return null;
        }

        /**
         * Запись номера состояния в битовую строку.
         * @param ns номер состояния.
         * @param nt номер входного воздействия.
         * @param state записываемый номер состояния.
         */
        public void setState(int ns, int nt, int state) {
```

```
            if (ns > cs || nt >= ct || state > cs || state < 0
|| ns < 0 || nt < 0) {
                throw new RuntimeException("Incorrect
arguments!");
            }
            int begin = ns * (bitsInNum + bitsInAct) * ct
                    + (bitsInNum + bitsInAct) * nt;
            long pow = power;
            for (int i = 0; i < bitsInNum; i++) {
                if (state / pow == 0) {
                    bitArray[begin + i] = false;
                } else {
                    bitArray[begin + i] = true;
                    state -= pow;
                }
                pow >>= 1;
            }
        }

        /**
         * Получает номер состояния.
         * @param ns номер рассматриваемого состояния.
         * @param nt номер входного воздействия.
         * @return номер состояния.
         */
        public int getState(int ns, int nt) {
            if (ns > cs || nt >= ct || ns < 0 || nt < 0) {
                throw new RuntimeException("Incorrect
arguments!");
            }
            int state = 0;
            long pow = power;
            int begin = ns * (bitsInNum + bitsInAct) * ct
                    + (bitsInNum + bitsInAct) * nt;
            for (int i = 0; i < bitsInNum; i++) {
                if (bitArray[begin + i]) {
                    state += pow;
                }
                pow >>= 1;
            }
            return state;
        }

    }

    /**
     * Конструктор автомата Мили, использующего битовою строку.
     * @param cs число состояний автомата.
     * @param is номер начального состояния автомата.
     * @param mu коэффициент заполненности поля яблоками.
     */
    public BitMealyAnt(int cs, int is, double mu) {
        bitAutom = new StringOfBits(cs);
```

```
        this.cs = cs;
        this.is = is;
        this.mu = mu;
    }

    /**
     * Шаг муравья.
     * @param ns номер состояния.
     * @param ant муравей.
     * @return номер нового состояния.
     */
    public int move(int ns, ExtendedAnt ant) {
        boolean[] variables = ant.F();
        int power = 1;
        int index = 0;
        for (int v = 0; v < variables.length; v++) {
            if (variables[v]) {
                index += power;
            }
            power <<= 1;
        }
        Ant.Action a = bitAutom.getAct(ns, index);
        if (a == Ant.Action.L) ant.L();
        else if (a == Ant.Action.R) ant.R();
        else if (a == Ant.Action.M) {
            ant.M();
            if (variables[0]) cf++;
        }
        return bitAutom.getState(ns, index);
    }

    /**
     * Число съеденных яблок.
     * @return кол-во съеденных яблок.
     */
    public int getCf() {
        return cf;
    }

    /**
     * Вычисляет функцию приспособленности.
     * @return результат функции приспособленности.
     */
    public double fitness() {
        if (fitness == Double.NEGATIVE_INFINITY) {
            fitness = 0;
            int s = is;
            for (int j = 0; j < BitMealyAntFactory.attempts;
j++) {
                ExtendedAnt ant = new ExtendedAnt(mu);
                for (int i = 0; i < Ant.NUMBER_STEPS; i++) {
                    s = move(s, ant);
                }
```

```
                fitness += cf;
                cf = 0;
            }
            fitness /= BitMealyAntFactory.attempts;
        }
        return fitness;
    }

    /**
     * Мутация автомата.
     * @param r генератор случайных чисел.
     * @return мутировавший автомат.
     */
    @Override
    public BitMealyAnt mutate(Random r) {
        BitMealyAnt ant = new BitMealyAnt(cs, (r.nextDouble() <
pmis) ? r.nextInt(cs) : is, mu);
        int state = r.nextInt(cs);
        System.arraycopy(bitAutom.bitArray, 0,
ant.bitAutom.bitArray, 0, bitAutom.bitArray.length);
        if (r.nextBoolean()) {
            switch (r.nextInt(3)) {
                case 0:
                    ant.bitAutom.setAct(state, r.nextInt(ct),
Ant.Action.L);
                    break;
                case 1:
                    ant.bitAutom.setAct(state, r.nextInt(ct),
Ant.Action.R);
                    break;
                case 2:
                    ant.bitAutom.setAct(state, r.nextInt(ct),
Ant.Action.M);
                    break;
            }
        } else {
            ant.bitAutom.setState(state, r.nextInt(ct),
r.nextInt(cs));
        }
        return ant;
    }

    /**
     * Скрещивание автоматов.
     * @param p второй родитель.
     * @param r генератор случайных чисел.
     * @return два потомка.
     */
    @Override
    public BitMealyAnt[] crossover(Individual p, Random r) {
        BitMealyAnt[] parent = new BitMealyAnt[2];
        parent[0] = this;
        parent[1] = (BitMealyAnt) p;
```

```
        BitMealyAnt[] child = new BitMealyAnt[2];
        int ri = r.nextInt(2);
        for (int i = 0; i < 2; i++) {
            child[Math.abs(ri - i)] = new BitMealyAnt(cs,
parent[i].is, mu);
        }
        for (int j = 0; j < cs; j++) {
            ri = r.nextInt(2);
            for (int i = 0; i < 2; i++) {
                for (int k = 0; k < 256; k++) {
                    child[Math.abs(ri - i)].bitAutom.setState(j,
k, parent[i].bitAutom.getState(j, k));
                    child[Math.abs(ri - i)].bitAutom.setAct(j,
k, parent[i].bitAutom.getAct(j, k));
                }
            }
        }
        return child;
    }

    /**
     * Сравнение автоматов.
     * @param individual второй автомат.
     * @return лучший автомат.
     */
    public int compareTo(Individual
 individual) {
        return Double.compare(individual.fitness(), fitness());
    }

    /**
     * Приведение к строке.
     * @return строка.
     */
    @Override
    public String toString() {
        String res = cs + " " + is + " " + " " + ct + fitness()
+ "\n";
        for (int i = 0; i < cs; i++) {
            for (int j = 0; j < ct; j++) {
                res += bitAutom.getState(i, j) + "  " +
bitAutom.getAct(i, j) + "\n";
            }
        }
        return res;
    }

    /**
     * Возвращает мувера.
     * @return массив объектов класса BitMealyAntMover.
     */
    @Override
    public Object[] getAttributes() {
```

```
        return new Object[] {new BitMealyAntMover(this, mu)};
    }
}
```

Файл BitMealyAntMover.java

```
package individual.bitmealy;

import task.ant.extended.ExtendedAntTask;
import task.ant.extended.ExtendedAnt;
import task.ant.extended.Ant;

/**
 * @author Sergey Popov
 *
 * Мувер для автомата Мили, представляемого битовой строкой.
 */
public class BitMealyAntMover implements ExtendedAntTask.Mover {

    /**
     * Коэффициент заполненности поля яблоками.
     */
    private final double mu;

    /**
     * Автомата Мили, представляемый битовой строкой.
     */
    public final BitMealyAnt automaton;

    /**
     * Число состояний в автомате.
     */
    private int cs;

    /**
     * Муравей.
     */
    private ExtendedAnt ant;

    /**
     * Сдвинуть муравья.
     * @return съел ли муравей яблоко.
     */
    public boolean move() {
        int cf = automaton.getCf();
        cs = automaton.move(cs, ant);
        return cf != automaton.getCf();
    }

    /**
     * Принимает нового муравья.
     * @param ant муравей.
     */
    public void restart(Ant ant) {
```

```
        this.ant = (ExtendedAnt) ant;
    }

    /**
     * Конструктор мувера.
     * @param automaton автомат.
     * @param mu коэффициент заполненности поля яблоками.
     */
    public BitMealyAntMover(BitMealyAnt automaton, double mu) {
        this.mu = mu;
        this.automaton = automaton;
        cs = automaton.is;
    }

    /**
     * Возвращает число съеденных яблок.
     * @return кол-во съеденных яблок.
     */
    public double getMu() {
        return mu;
    }
}
```

Файл BitMealyAntFactory.java

```
package individual.bitmealy;

import laboratory.common.ga.IndividualFactory;
import task.ant.extended.Ant;

import java.util.Random;

/**
 * @author Sergey Popov
 *
 * Фабрика для автомата Мили, представляемого битовой строкой.
 */
public class BitMealyAntFactory implements IndividualFactory {

    /**
     * Генератор случайных чисел.
     */
    private final Random r = new Random();

    /**
     * Число попыток запуска автомата.
     */
    public static int attempts;

    /**
     * Коэффициент заполненности поля яблоками.
     */
    private final double mu;
```

```
    /**
     * Число состояний в автомате.
     */
    private final int countStates;

    /**
     * Создание случайного автомата.
     * @return случайный автомат.
     */
    public BitMealyAnt randomIndividual() {
        BitMealyAnt ant = new BitMealyAnt(countStates,
r.nextInt(countStates), mu);
        for (int i = 0; i < countStates; i++) {
            for (int j = 0; j < 256; j++) {
                switch (r.nextInt(3)) {
                    case 0: ant.bitAutom.setAct(i, j,
Ant.Action.L); break;
                    case 1: ant.bitAutom.setAct(i, j,
Ant.Action.M); break;
                    case 2: ant.bitAutom.setAct(i, j,
Ant.Action.R); break;
                }
                ant.bitAutom.setState(i, j,
r.nextInt(countStates));
            }
        }
        return ant;
    }

    /**
     * Конструктор фабрики для автомата.
     * @param cs число состояний в автомате.
     * @param mu коэффициент заполненности поля яблоками.
     */
    public BitMealyAntFactory(int cs, double mu) {
        countStates = cs;
        this.mu = mu;
    }

}
```

Файл FactoryLoader.java

```
package individual.bitmealy;

import laboratory.util.AbstractLoader;

import java.util.jar.JarFile;

/**
 * @author Sergey Popov
 *
 * Загрузчик для фабрики автомата Мили, представляемого битовой
строкой.
```

```
 */
public class FactoryLoader extends
AbstractLoader<BitMealyAntFactory> {

    /**
     * Загружает параметры для фабрики.
     * @param args аргументы.
     * @return фабрика.
     */
    public BitMealyAntFactory load(Object... args) {
        BitMealyAntFactory.attempts =
properties.getInt("count.attempts");
        return new
BitMealyAntFactory(properties.getInt("count.states"),
properties.getDouble("mu"));
    }

    /**
     * Конструктор.
     * @param file JarFile.
     */
    public FactoryLoader(JarFile file) {
        super(file, "automaton.conf");
    }
}
```