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

Н. И. Макаров

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

Вариант № 26

Санкт-Петербург
2009

Оглавление

Введение

В данной лабораторной работе исследуется применение генетических алгоритмов для автоматического построения конечных управляющих автоматов. Рассматривается задача об «Умном муравье - 3». С помощью метода рулетки генерируется конечный автомат Мура, который описывает поведение муравья.

Для эмуляции действий муравья студентами кафедры «Компьютерные технологии» СПбГУ ИТМО была создана программа «Виртуальная лаборатория» [1]. Решением поставленной задачи является плагин для данной лаборатории, который описывает представление муравья в виде автомата, а также реализует генетический алгоритм.

1. Постановка задачи

Цель лабораторной работы – построить такой автомат Мура, что муравей, поведение который им описывается, съедает как можно больше еды на поле.

1.1. Задача об «Умном муравье - 3»

Приведем описание задачи об «Умном муравье - 3» [2]. Игра происходит на поверхности тора размером 32 на 32 клетки (рис.1). В некоторых клетках (обозначены на рис.1 черным цветом) находятся яблоки (не больше одного в клетке). Клетки поля, в которых нет еды, обозначены серым цветом. Для каждой клетки вероятность появления еды в ней равна заранее определенной величине μ.

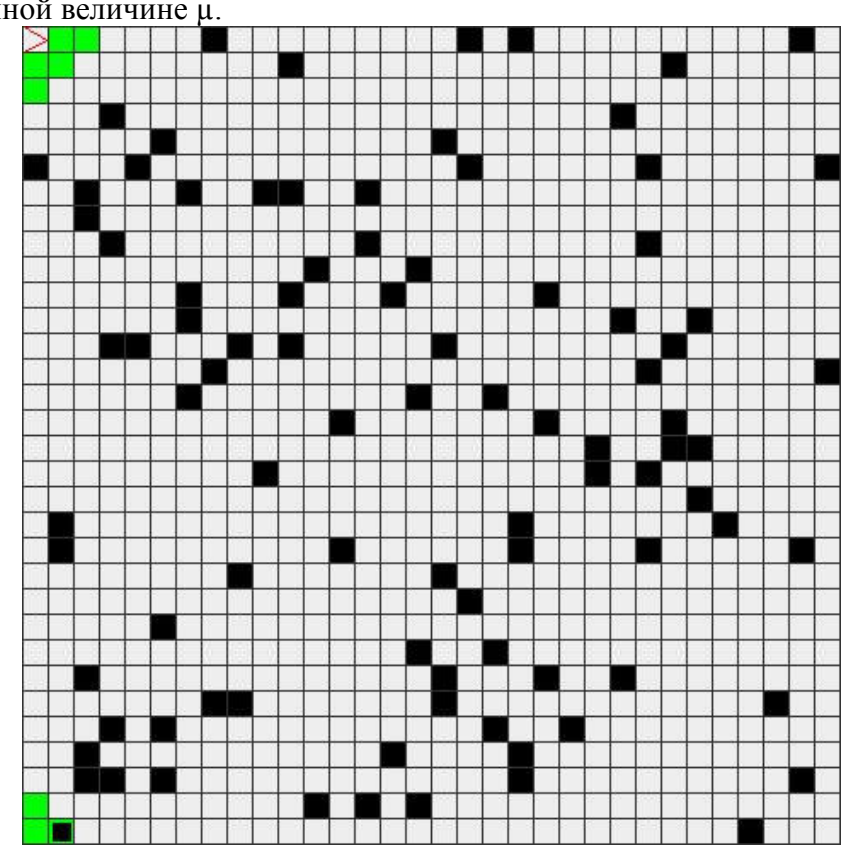

Рис. 1. Игровое поле

В левой верхней клетке в начале игры находится муравей (на рис.1 он обозначен красным треугольником, который показывает направление его взгляда). Он занимает одну клетку и смотрит в одном из четырех направлений (север, юг, запад, восток). В начале игры муравей смотрит на восток. Муравей умеет определять, в каких клетках из восьми находящихся рядом с ним и помеченных зеленым цветом находится еда. Таким образом, в каждой клетке для муравья существует $2^8 = 256$ входных воздействий. За один игровой ход муравей может совершить одно из трех действий:

- сделать шаг вперед, съедая при этом еду, если она там находится (M – move);
- повернуть налево (L – left);
- повернуть направо (R – right).

Съеденная муравьем еда не восполняется, муравей жив на протяжении всей игры, еда не является необходимым ресурсом для его жизни. Муравей может ходить по любым клеткам поля. Игра длится 200 ходов, на каждом из которых муравей совершает одно из трех действий.

По окончании игры подсчитывается число яблок, съеденное муравьем. Это значение и есть результат игры. Цель игры – создать муравья, который съест как можно больше еды. При этом отметим, что муравьи, съедающие всю еду, заканчивают игру с одинаковым результатом, который не зависит от того, сколько ходов им на это потребовалось.

1.2. Автомат Мура

Автомат Мура – конечный автомат, выходное действие которого зависит только от текущего состояния. Переход в новое состояние осуществляется при учете текущего состояния и входного воздействия.

Формально автомат Мура – совокупность шести объектов:

$$A = (S, S_0, X, Y, \delta, \mu),$$

где:

- S – множество состояний;
- S_0 – стартовое состояние;
- X – множество входных воздействий;
- Y – множество выходных действий;
- $\delta : S \times X \to S$ (таблица переходов по входным воздействиям для каждого состояния);
- $\mu : S \to Y$ (выходные действия муравья для каждого состояния).

2. Генетический алгоритм

Разработанный алгоритм генетического программирования [3] состоит из пяти частей:

- создание начального поколения;
- мутация;
- скрещивание (кроссовер);
- отбор особей для формирования следующего поколения;
- вычисление функции приспособленности (фитнес-функции).

2.1. Представление особи

Каждая особь представляет собой некоторый конечный автомат Мура с заданным числом состояний, описывающий поведение муравья.

В программе автомат представлен массивом состояний `State`, каждый из которых хранит выходное воздействие, а также 256 переходов в другие состояния по различным входным воздействиям. Также хранится указатель на стартовое состояние.

2.2. Создание начального поколения

Начальное поколение состоит из фиксированного числа (по умолчанию – 200) случайно сгенерированных автоматов Мура с заранее определенным числом состояний.

2.3. Мутация

При мутации стартовым состоянием становится случайное из автомата, затем случайно выбирается один из двух равновероятных вариантов:

- изменение действия, совершаемого муравьем, который находится в новом стартовом состоянии – случайно и равновероятно выбирается выходное воздействие из множества Y.
- изменение состояния, в которое ведет переход – случайно и равновероятно выбирается переход. После этого состояние, в которое ведет переход, заменяется на случайно выбранное состояние.

Также в генетическом алгоритме предусмотрена операция «большой мутации», которая осуществляет генерацию полностью новых случайных автоматов. Она может быть использована, когда в популяции поведение муравьев почти идентично.

2.4. Скрещивание

Оператор скрещивания получает на вход две особи и выдает также две особи. Процесс скрещивания происходит следующим образом. Создаются два потомка, эквивалентных родителям. Затем для каждого состояния равновероятно либо производится обмен информацией для этого состояния, либо не производится. Таким образом, состояния родителей перемешиваются в следующем поколении.

2.5. Отбор особей для формирования следующего поколения

Генерация очередного поколения осуществляется с помощью метода «рулетки»: всем автоматам ставятся в соответствие отрезки длины, равной значению фитнес-функции, затем

выбирается случайная точка, которая будет принадлежать одному из отрезков. После этого соответствующий автомат добавляется в следующее поколение. Выбор осуществляется столько раз, сколько особей должно быть в поколении. Затем производится скрещивание пар автоматов, и, наконец, мутация каждого из получившихся детей.

2.6. Вычисление функции приспособленности (фитнес-функции)

Функция приспособленности вычисляется следующим образом: проводится эксперимент с участием муравья с поведением, заданным текущим автоматом Мура. С помощью лаборатории эмулируются его перемещения в зависимости от расположения еды: он проходит по полю несколько раз, каждый раз подсчитывается число съеденных яблок. Полученные результаты усредняются (сумма делится на число попыток) и выдаются в качестве ответа.

2.7. Настраиваемые параметры алгоритма

Следующие параметры алгоритма генетического программирования могут быть изменены:

- размер поколения;
- число состояний;
- число экспериментов над каждым муравьем;
- вероятность мутации;
- время до «большой» мутации поколения;
- вероятность появления еды в каждой клетке.

Как показывают вычислительные эксперименты, изменение некоторых из этих параметров может существенно влиять на время поиска автомата, позволяющего муравью съесть всю еду. Отметим, что от распределения еды зависит число состояний автомата, который «съедает» наибольшее число яблок. Чем меньше еды – тем больше состояний потребуется, если все поле заполнено едой, то хватит двух состояний (с помощью одного муравей идет вперед, другое позволяет повернуть с прямой траектории).

3. Результаты

Вычислительные эксперименты проводились с различными значениями параметров алгоритма. Далее приводится описание графа переходов для автомата Мура с пятью состояниями (четветое состояние является начальным), значение фитнесс-функции для соответствующего муравья равно 42. Для каждого состояния показано выходное воздействие, затем таблица переходов по всем 256 входным воздействиям.
Вероятность появления еды для каждой клетки равна 0.1. Поэтому математическое ожидание числа яблок на поле равно 102.4, а вероятность мутации особи после скрещивания равна 0.02.

Action for this state=M, transitions=

1, 2, 4, 4, 5, 4, 1, 2, 2, 5, 2, 1, 2, 1, 2, 3, 1, 5, 5, 4, 3, 5, 1, 2, 4, 5, 1, 2, 1, 1, 4, 3, 3, 2, 2, 3, 2, 1, 5, 2, 5, 2, 1, 5, 1, 5, 2, 2, 5, 2, 4, 4, 2, 2, 1, 5, 4, 4, 4, 3, 5, 4, 1, 4,
2, 5, 1, 3, 2, 4, 1, 5, 3, 3, 1, 5, 1, 3, 1, 5, 3, 4, 2, 4, 2, 5, 3, 3, 5, 5, 2, 4, 2, 5, 5, 2, 2, 2, 1, 1, 1, 3, 4, 1, 3, 4, 3, 5, 4, 5, 2, 5, 1, 3, 3, 1, 3, 4, 1, 2, 5, 2, 1, 4, 1, 1, 1, 5,
4, 5, 5, 5, 1, 5, 2, 2, 4, 1, 1, 4, 2, 2, 4, 5, 4, 3, 4, 1, 2, 2, 5, 3, 4, 1, 1, 4, 2, 3, 2, 4, 2, 5, 5, 1, 1, 1, 1, 2, 3, 4, 1, 1, 2, 5, 4, 5, 1, 4, 2, 2, 4, 5, 1, 3, 2, 1, 2, 3, 5, 4, 3, 2,
1, 5, 5, 3, 1, 5, 4, 4, 2, 1, 1, 4, 1, 1, 4, 4, 2, 5, 3, 3, 2, 1, 3, 5, 2, 2, 3, 2, 3, 1, 2, 5, 4, 2, 3, 4, 2, 1, 3, 3, 4, 1, 1, 1, 4, 2, 1, 4, 3, 5, 2, 5, 2, 1, 4, 2, 3, 5, 4, 4, 2, 5, 2, 4

Action for this state=R, transitions=

3, 5, 4, 2, 5, 1, 2, 4, 4, 3, 4, 1, 1, 4, 4, 3, 3, 2, 5, 5, 5, 1, 4, 4, 1, 1, 5, 4, 3, 4, 2, 5, 1, 2, 2, 1, 1, 5, 2, 5, 4, 2, 4, 4, 5, 3, 4, 1, 5, 2, 3, 5, 5, 4, 2, 4, 1, 5, 1, 4, 3, 3, 2, 4,
5, 2, 4, 2, 1, 1, 2, 2, 1, 2, 2, 5, 4, 3, 4, 2, 2, 3, 5, 4, 3, 1, 1, 4, 1, 4, 3, 2, 3, 5, 4, 4, 2, 1, 5, 3, 2, 3, 1, 2, 1, 4, 3, 5, 5, 2, 3, 4, 5, 5, 2, 2, 3, 5, 5, 3, 5, 2, 5, 2, 4, 1, 1, 5,
2, 5, 5, 4, 4, 1, 2, 5, 5, 4, 4, 1, 1, 2, 1, 5, 1, 4, 3, 3, 2, 3, 4, 2, 2, 3, 4, 1, 3, 3, 3, 3, 4, 1, 2, 5, 3, 3, 2, 3, 1, 2, 5, 1, 1, 4, 3, 1, 5, 4, 3, 4, 3, 5, 4, 5, 5, 5, 4, 3, 5, 2, 4, 4,
5, 1, 3, 4, 4, 3, 4, 2, 1, 3, 1, 1, 4, 3, 3, 1, 4, 3, 3, 5, 2, 2, 5, 3, 3, 4, 2, 2, 5, 1, 5, 4, 2, 5, 5, 1, 4, 5, 3, 5, 3, 4, 5, 2, 5, 2, 3, 1, 1, 2, 5, 5, 4, 3, 4, 2, 3, 3, 4, 4, 2, 5, 4, 4

Action for this state=M, transitions=

4, 3, 4, 5, 3, 4, 2, 3, 4, 5, 3, 5, 2, 3, 4, 3, 2, 4, 3, 1, 5, 5, 4, 1, 5, 5, 4, 3, 4, 3, 2, 5, 2, 5, 4, 4, 4, 2, 5, 2, 2, 2, 1, 4, 5, 3, 5, 2, 2, 4, 5, 3, 3, 2, 5, 2, 1, 2, 5, 4, 3, 4, 5, 3,
4, 5, 2, 2, 1, 5, 2, 1, 4, 2, 3, 3, 4, 5, 3, 4, 5, 3, 5, 5, 2, 1, 4, 5, 4, 5, 1, 1, 2, 3, 2, 1, 2, 5, 1, 4, 2, 5, 3, 2, 1, 3, 3, 4, 4, 2, 4, 5, 2, 1, 1, 2, 3, 2, 5, 5, 4, 4, 2, 5, 1, 2, 1, 3,
3, 2, 2, 4, 4, 2, 2, 3, 5, 5, 3, 1, 3, 2, 4, 3, 4, 4, 1, 2, 1, 2, 5, 1, 1, 5, 2, 3, 3, 3, 2, 5, 5, 3, 4, 1, 4, 3, 4, 1, 1, 1, 4, 3, 1, 4, 2, 4, 3, 2, 5, 1, 2, 1, 4, 1, 4, 4, 4, 3, 2, 2, 3,
1, 4, 2, 1, 5, 1, 2, 5, 3, 2, 4, 4, 3, 4, 4, 2, 4, 1, 2, 1, 2, 2, 2, 4, 1, 1, 5, 3, 2, 3, 3, 2, 5, 3, 1, 3, 3, 2, 2, 3, 5, 1, 4, 5, 5, 5, 3, 5, 1, 5, 3, 1, 3, 4, 5, 4, 1, 4, 2, 5, 1, 2, 2, 3

Action for this state=M, transitions=

4, 1, 3, 3, 1, 2, 1, 3, 2, 1, 1, 4, 3, 1, 1, 3, 3, 3, 4, 5, 4, 4, 4, 3, 4, 5, 4, 5, 4, 2, 1, 5, 1, 1, 1, 3, 4, 2, 3, 5, 1, 4, 1, 5, 4, 4, 4, 5, 5, 1, 5, 2, 1, 5, 5, 3, 2, 3, 3, 3, 3, 2, 5, 1,
2, 5, 2, 1, 4, 2, 5, 3, 5, 1, 4, 5, 3, 4, 2, 4, 1, 2, 1, 2, 5, 2, 1, 3, 4, 5, 2, 4, 3, 1, 1, 2, 5, 1, 2, 1, 2, 4, 3, 4, 4, 4, 4, 2, 5, 4, 2, 4, 4, 1, 2, 5, 5, 1, 2, 4, 1, 3, 5, 1, 4, 2, 3, 1,
5, 3, 4, 2, 2, 3, 2, 5, 5, 4, 4, 5, 5, 1, 2, 4, 3, 5, 4, 1, 2, 3, 2, 4, 3, 4, 4, 5, 5, 4, 5, 1, 1, 3, 2, 4, 3, 3, 5, 5, 2, 2, 2, 3, 1, 2, 3, 5, 3, 5, 1, 4, 2, 1, 4, 3, 3, 1, 3, 5, 5, 1, 3, 1,
3, 1, 1, 4, 5, 1, 2, 5, 4, 1, 4, 1, 2, 4, 5, 4, 3, 5, 5, 5, 4, 3, 2, 4, 2, 5, 2, 3, 1, 5, 2, 4, 5, 1, 3, 3, 4, 5, 3, 4, 2, 2, 2, 5, 3, 1, 3, 4, 5, 5, 4, 1, 5, 1, 3, 3, 3, 3, 1, 5, 1, 5, 3, 4

Action for this state=M, transitions=

5, 4, 5, 1, 1, 4, 4, 2, 3, 5, 2, 3, 4, 3, 5, 1, 5, 3, 2, 5, 4, 4, 4, 4, 3, 5, 4, 5, 5, 4, 1, 3, 5, 5, 3, 3, 1, 1, 2, 2, 2, 4, 2, 3, 4, 4, 4, 4, 5, 1, 3, 3, 5, 1, 5, 1, 3, 4, 1, 5, 1, 2, 2, 1,
2, 2, 5, 4, 4, 2, 4, 1, 5, 4, 2, 5, 1, 5, 4, 3, 4, 1, 1, 2, 4, 1, 2, 4, 4, 3, 5, 5, 4, 4, 3, 2, 1, 1, 2, 4, 4, 5, 4, 3, 1, 3, 5, 2, 2, 4, 5, 5, 4, 1, 2, 5, 2, 1, 4, 1, 5, 1, 5, 2, 2, 5, 2, 1,
5, 4, 5, 3, 2, 3, 2, 5, 2, 3, 4, 5, 2, 4, 1, 1, 4, 2, 2, 3, 5, 1, 5, 2, 1, 1, 1, 5, 1, 5, 1, 3, 5, 3, 4, 3, 2, 2, 1, 4, 5, 4, 4, 2, 5, 2, 2, 5, 1, 2, 3, 4, 2, 2, 3, 2, 1, 3, 1, 3, 2, 1, 2, 1,
1, 4, 2, 2, 1, 3, 4, 2, 1, 3, 5, 5, 4, 1, 4, 1, 2, 1, 2, 3, 2, 2, 1, 3, 4, 3, 3, 2, 5, 5, 4, 1, 2, 4, 5, 1, 1, 1, 1, 4, 1, 5, 2, 2, 1, 5, 2, 3, 2, 5, 3, 5, 5, 5, 1, 5, 3, 1, 4, 3, 1, 1, 1, 3, 2

3.1. График максимального значения фитнес-функции

На рис.2 представлен график зависимости максимального значения фитнес-функции от номера поколения.

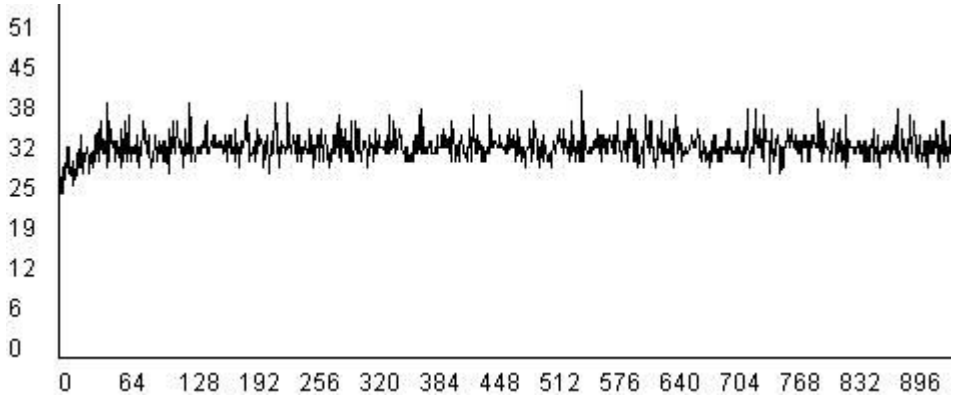

Рис. 2

Из рисунка следует, что существует много локальных максимумов. Это свидетельствует о том, что метод рулетки выбирает средние особи, самые лучшие могут не попасть в следующее поколение.

3.2. График среднего значения фитнес-функции

На рис.3 представлен график зависимости усредненного значения фитнес-функции от номера поколения.

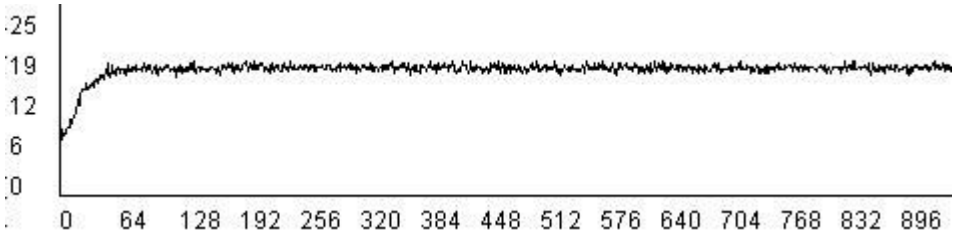

Рис. 3

Среднее значение достаточно быстро устанавливается на определенной величине.

Заключение

Использование метода рулетки плохо подходит для выделения наилучших особей при скрещивании. Такое поведение можно объяснить тем, что значения фитнес-функции для особей не отличаются разительно. Поэтому даже у самой приспособленной особи есть шанс не попасть в следующее поколение. Островной алгоритм позволяет получить б*о*льшие значения функции приспособленности для такого же представления особей.

Использование автоматов Мура для представления муравья в задаче об «Умном муравье - 3» оправдано, так как позволяет найти муравья, съедающего половину еды на поле. При этом еда расположена случайным образом.

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» и ее модификациях.
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. Исходный код плагина для «Виртуальной лаборатории» на языке *Java*

Представление особи

Файл `SmartMover.java`

```
package individual.smartant;

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

/**
 * @author Nikolay Makarov
 */
public class SmartMover implements ExtendedAntTask.Mover {

    /**
     * pointer to SmartAnt which is used in move procedure
     */
    public final SmartAnt automaton;

    private final double mu;
    private SmartAnt.State cs;
    private ExtendedAnt ant;

    /**
     * @see task.ant.extended.ExtendedAntTask.Mover#move()
     */
    public boolean move() {
        int cf = this.automaton.getCf();
        this.cs = this.automaton.move(this.cs, this.ant);
        return cf != this.automaton.getCf();
    }

    /**
     * @see task.ant.extended.ExtendedAntTask.Mover#restart
     * (task.ant.extended.Ant)
     */
    public void restart(Ant a) {
        this.ant = (ExtendedAnt) a;
    }

    /**
     * @param automaton pointer to SmartAnt which is used in move procedure
     * @param mu probability of food in the current cell
     */
    public SmartMover(SmartAnt automaton, double mu) {
        this.mu = mu;
        this.automaton = automaton;
        this.cs = automaton.state[automaton.initialState];
    }
```

```
    /**
     * @return probability of food in the current cell
     * @see task.ant.extended.ExtendedAntTask.Mover#getMu()
     */
    public double getMu() {
        return this.mu;
    }
}
```

Файл `SmartAntFactory.java`

```
package individual.smartant;

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

import java.util.Random;

/**
 * @author Nikolay Makarov
 */
public class SmartAntFactory implements IndividualFactory {

    private final Random r = new Random();

    public static int attempts;
    private final double mu;

    private final int countStates;

    /**
     * @see laboratory.common.ga.IndividualFactory#randomIndividual()
     */
    public SmartAnt randomIndividual() {
        SmartAnt ant = new SmartAnt(this.countStates,
                    this.r.nextInt(this.countStates), this.mu);
        for (int i = 0; i < this.countStates; i++) {
            Ant.Action action;
            byte[] endState = new byte[256];
            switch (this.r.nextInt(3)) {
              case 0: action = Ant.Action.L; break;
              case 1: action = Ant.Action.R; break;
              default: action = Ant.Action.M; break;
           }
            for (int j = 0; j < 256; j++) {
                endState[j] = (byte) this.r.nextInt(this.countStates);
            }
            ant.state[i] = new SmartAnt.State(action, endState);
        }
        return ant;
    }

    /**
     * @param countStates amount of states,
     * which is written in properties file
```

```
     * @param mu Probability of food in the current cell
     */
    public SmartAntFactory(int countStates, double mu) {
        this.countStates = countStates;
        this.mu = mu;
    }
}
```

Файл `SmartAnt.java`

```
package individual.smartant;

import laboratory.common.ga.Individual;
import laboratory.common.Visualizable;

import java.util.Random;

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

/**
 * @author Nikolay Makarov
 */
public class SmartAnt implements Visualizable {

    /**
     * Probability of changing initial state during mutation
     */
    public static double pmis = 0.66;

    /**
     * Probability of food in the current cell
     */
    public final double mu;

    private double fitness = Double.NEGATIVE_INFINITY;

    /**
     * Initial state of Moore automaton
     */
    public final int initialState;

    /**
     * Array with states
     */
    public final State[] state;

    /**
     * Number of different incoming signals
     */
    public final int ct = 256;

    private int cf = 0;

    /**
     * Class which represents state in Moore automaton.
     * Every state produces outgoing signal (L, R or M) and
```

```
 * give next state for every incoming signal
 */
public static class State {

    /**
     * Array of states which can be reached from current state
     */
    public final byte[] endState;

    /**
     * Outgoing signal (L, R or M)
     */
    public final Ant.Action action;

    /**
     * @param action Outgoing signal (L, R or M)
     * @param endState Array of states which can be reached
     * from current state
     */
    public State(Ant.Action action, byte[] endState) {
        this.endState = endState;
        this.action = action;
    }

}

/**
 * @param cs amount of states in Moore automaton
 * @param is initial state
 * @param mu Probability of food in the current cell
 */
public SmartAnt(int cs, int is, double mu) {
    this.state = new State[cs];
    this.initialState = is;
    this.mu = mu;
}

/**
 * @param s current state
 * @param ant Ant which can see 8 cells around itself,
 * it gave us information      * about them
 * @return next state, which will be obtained after proceeding information
 */
public State move(State s, ExtendedAnt ant) {
    boolean[] variables = ant.F();
    int power = 0;
    int index = 0;
    /**
     * We count id of incoming signal
     */
    for (boolean v: variables) {
        if (v) {
            /**
             * According to availability of the food in current cell
             * we can add degree of two to the index
             */
            index += (1 << power);
```

```
            }
            power++;
        }

        Ant.Action a = s.action;
        if ((a == Ant.Action.M) && variables[0]) {
            this.cf++;
        }
        if (a == Ant.Action.L) ant.L();
        else if (a == Ant.Action.R) ant.R();
        else if (a == Ant.Action.M) ant.M();
        return this.state[s.endState[index]];
    }

    /**
     * @return amount of food which was eaten
     */
    public int getCf() {
        return this.cf;
    }

    /**
     * Fitness function value
     * @see laboratory.common.ga.Individual#fitness()
     */
    @Override
    public double fitness() {
        if (this.fitness == Double.NEGATIVE_INFINITY) {
            this.fitness = 0;
            for (int j = 0; j < SmartAntFactory.attempts; j++) {
                State s = this.state[this.initialState];
                ExtendedAnt ant = new ExtendedAnt(this.mu);
                for (int i = 0; i < Ant.NUMBER_STEPS; i++) {
                    s = move(s, ant);
                }
                this.fitness += this.cf;
                this.cf = 0;
            }
            fitness /= SmartAntFactory.attempts;
        }
        return this.fitness;
    }

    /**
     * @see laboratory.common.ga.Individual#mutate(java.util.Random)
     */
    @Override
    public SmartAnt mutate(Random r) {
        int cs = this.state.length;
        SmartAnt ant = new SmartAnt(
        cs,
        (r.nextDouble() < pmis) ? r.nextInt(cs) : this.initialState,
        this.mu
        );
        System.arraycopy(this.state, 0, ant.state, 0, cs);
        int is = r.nextInt(cs);
        State s = this.state[is];
```

```
    if (r.nextBoolean()) {
        Ant.Action action;
        switch (r.nextInt(3)) {
            case 0:
                action = Ant.Action.L;
                break;
            case 1:
                action = Ant.Action.R;
                break;
            default:
                action = Ant.Action.M;
                break;
        }
        ant.state[is] = new State(action, this.state[is].endState);
    } else {
        byte[] endState = new byte[this.ct];
        System.arraycopy(s.endState, 0, endState, 0, this.ct);
        endState[r.nextInt(this.ct)] = (byte) r.nextInt(cs);
        ant.state[is] = new State(this.state[is].action, endState);
    }
    return ant;
}

/**
 * @see laboratory.common.ga.Individual#crossover
       * (laboratory.common.ga.Individual, java.util.Random)
 */
@Override
public SmartAnt[] crossover(Individual p, Random r) {
    SmartAnt[] parent = new SmartAnt[2];
    parent[0] = this;
    parent[1] = (SmartAnt) p;
    SmartAnt[] child = new SmartAnt[2];
    int ri = r.nextInt(2);
    int cs = this.state.length;
    for (int i = 0; i < 2; i++) {
        child[Math.abs(ri - i)] =
          new SmartAnt(cs, parent[i].initialState, this.mu);
    }
    for (int j = 0; j < cs; j++) {
        ri = r.nextInt(2);
        for (int i = 0; i < 2; i++) {
            child[Math.abs(ri - i)].state[j] = parent[i].state[j];
        }
    }
    return child;
}

/**
 * @see java.lang.Comparable#compareTo(java.lang.Object)
 */
@Override
public int compareTo(Individual individual) {
    return Double.compare(individual.fitness(), fitness());
}

/**
```

```
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString() {
        String res = this.state.length + " states (initial=" +
        (this.initialState + 1) + "), fitness=" + fitness() + "\n";
        for (State s : this.state) {
        res += "Action for this state=" + s.action + ", transitions=\n";
            for (int i = 0; i < this.ct; i++) {
                res += s.endState[i] + 1;
                if (i < this.ct - 1) {
                    res += ", ";
                }
                if ((i + 1) % 64 == 0) {
                    res += "\n";
                }
            }
            res += "\n";
        }
        return res;
    }

    /**
     * @see laboratory.common.Visualizable#getAttributes()
     */
    @Override
    public Object[] getAttributes() {
        return new Object[] {new SmartMover(this, this.mu)};
    }
}
```

Файл `FactoryLoader.java`

```
package individual.smartant;

import laboratory.util.AbstractLoader;

import java.util.jar.JarFile;

/**
 * @author Nikolay Makarov
 */
public class FactoryLoader extends AbstractLoader<SmartAntFactory> {

    /**
     * @see laboratory.common.Loader#load(java.lang.Object[])
     */
    public SmartAntFactory load(Object... args) {
        SmartAntFactory.attempts = properties.getInt("count.attempts");
        return new SmartAntFactory(this.properties.getInt("count.states"),
        this.properties.getDouble("mu"));
    }

    /**
     * @param file jar-file with this project
     * which contains necessary properties
```

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

Генетический алгоритм

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 Nikolay Makarov
 */
public class RouletteGALoader implements Loader<GA> {

    private final Parser properties;
    /**
     * @see laboratory.common.Loader#load(java.lang.Object[])
     */
    public GA load(Object... args) {
        return new RouletteGA(this.properties.getInt("sizeGeneration"),
                this.properties.getDouble("probabilityMutation"),
                (IndividualFactory) args[0]);
    }

    /**
     * Creates Parser object which will be used for reading properties
     * @param file jar-file with this project
     */
    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();
        }
        this.properties = new Parser(in);
    }

    /**
     * @see laboratory.common.Loader#getProperties()
     */
    public Properties getProperties() {
```

```
        return this.properties.getProperties();
    }
}
```

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 Nikolay Makarov
 */
public class RouletteGA implements GA {

    private List<Individual> generation;

    private final double probabilityMutation;

    private final IndividualFactory factory;

    private final Random r;

    /**
     * @see laboratory.common.ga.GA#nextGeneration()
     */
    public void nextGeneration() {
        int size = this.generation.size();
        // Sum of individuals' fitness values
        double totalFitness = 0.0;
        for (Individual individual : this.generation) {
        totalFitness += individual.fitness();
        }
        List<Individual> newGeneration = new ArrayList<Individual> (size);
        for (int i = 0; i < size; i++) {
        // random position on roulette
        double val = this.r.nextDouble() * totalFitness;
        // individual which corresponds to this position
        double sum = 0.0;
        for (int j = 0; j < size; j++) {
              double fval = this.generation.get(j).fitness();
              if (sum + fval > val) {
                    newGeneration.add(this.generation.get(j));
                    break;
              }
              sum += fval;
        }
        }
        /**
```

```
     * We can crossover neighbour individuals,
     * because we chose them at random positions earlier
     */
    for (int i = 0; i < size; i += 2) {
    Individual father = newGeneration.get(i);
    Individual mother = newGeneration.get(i + 1);
        Individual[] s = father.crossover(mother, this.r);
        newGeneration.set(i, s[0]);
        newGeneration.set(i + 1, s[1]);
    }
    if (size % 2 == 1) {
        // If amount of individuals is odd, we mutate last
        newGeneration.set(size - 1,
          newGeneration.get(size - 1).mutate(this.r));
    }
    // We mutate all of them
    for (int i = 0; i < size; i++) {
        if (this.r.nextDouble() < this.probabilityMutation) {
            newGeneration.set(i, newGeneration.get(i).mutate(this.r));
        }
    }
    this.generation = newGeneration;
    Collections.sort(this.generation);
}

/**
 * @see laboratory.common.ga.GA#getGeneration()
 */
public List<Individual> getGeneration() {
    return this.generation;
}

/**
 * @param sizeGeneration constant generation size
 * @param probabilityMutation probability of mutation after crossover
 * @param factory pointer to the individual factory
 */
public RouletteGA(int sizeGeneration, double probabilityMutation,
                IndividualFactory factory) {
    this.probabilityMutation = probabilityMutation;
    // Creating new generation
    this.generation = new ArrayList<Individual>(sizeGeneration);
    for (int i = 0; i < sizeGeneration; i++) {
        this.generation.add(factory.randomIndividual());
    }
    Collections.sort(this.generation);
    this.factory = factory;
    this.r = new Random();
}

/**
 * Creates new generation
 * @see laboratory.common.ga.GA#bigMutation()
 */
public void bigMutation() {
    for (int i = 0; i < this.generation.size(); i++) {
        this.generation.set(i, this.factory.randomIndividual());
```

```
        }
        Collections.sort(this.generation);
    }

    /**
     * @return best individual in current generation
     * @see laboratory.common.ga.GA#getBest()
     */
    public Individual getBest() {
        return this.generation.get(0);
    }
}
```

Скрипт для сборки плагина к «Виртуальной лаборатории»

Файл `build.properties`

```
deploy=deploy
debug.val=true
util=util

ga.dir=gas
individual.dir=individuals

ga.src=ga
individual.src=individual

project.name=Virtual Laboratory
main.class=laboratory.core.Main
```

Файл `build.xml` (для компиляции необходимы файлы `util.jar`, `common.jar` и `extended-ant.jar`, которые представляют «Виртуальную лабораторию»)

```
<?xml version="1.0"?>

<project name="visualizer" default="build">

    <property file="build.properties"/>

    <target name="build" depends="build.main, build.GA, build.individual" />

    <property name="common.build" value="${util}/common.jar"/>
    <property name="util.build" value="${util}/util.jar"/>
    <property name="exttask.build" value="${util}/extended-ant.jar"/>

    <target name="build.main">
        <mkdir dir="${deploy}"/>
    </target>

    <target name="build.GA">
        <mkdir dir="${deploy}/${ga.dir}"/>
        <antcall target="build.plugin">
            <param name="plugin" value="${ga.src}/roulette"/>
```

```
        <param name="plugin.classpath" value="${common.build};
            ${util.build}"/>
        <param name="plugin.dir" value="${ga.dir}"/>
    </antcall>
</target>

<target name="build.individual">
    <mkdir dir="${deploy}/${individual.dir}"/>
    <antcall target="build.plugin">
        <param name="plugin" value="${individual.src}/SmartAnt"/>
        <param name="plugin.classpath" value="${common.build};
          ${util.build}; ${exttask.build}"/>
        <param name="plugin.dir" value="${individual.dir}"/>
    </antcall>
</target>

<target name="build.plugin">
    <echo message="${plugin}"/>
    <echo message="${plugin.classpath}"/>
    <fail unless="plugin"/>
    <property file="${plugin}/plugin.properties" prefix="plugin"/>
    <fail unless="plugin.main.class"/>
    <fail unless="plugin.extension.name"/>
    <fail unless="plugin.comment"/>
    <condition property="plugin.src" value="src">
        <not>
            <isset property="plugin.src/"/>
        </not>
    </condition>
    <condition property="plugin.filter" value="">
        <not>
            <isset property="plugin.filter"/>
        </not>
    </condition>
    <condition property="plugin.graphic.settings" value="">
        <not>
            <isset property="plugin.graphic.settings"/>
        </not>
    </condition>
    <echo message="${plugin.filter}"/>
    <echo message="${plugin.graphic.settings}"/>
    <condition property="plugin.build" value="classes">
        <not>
            <isset property="plugin.build"/>
        </not>
    </condition>
    <condition property="plugin.jar" value="${plugin.extension.name}.jar">
        <not>
            <isset property="plugin.jar"/>
        </not>
    </condition>
    <mkdir dir="${plugin}/${plugin.build}"/>
    <javac
            srcdir="${plugin}/${plugin.src}"
            destdir="${plugin}/${plugin.build}"
            classpath="${plugin.classpath}"
            />
```

```
        <jar destfile="${deploy}/${plugin.dir}/${plugin.jar}">
            <fileset dir="${plugin}/${plugin.build}"/>
            <fileset dir="${plugin}/${plugin.resources}"/>
            <manifest>
                <attribute name="Main-Class" value="${plugin.main.class}"/>
                <attribute name="Extension-Name"
                    value="${plugin.extension.name}"/>
                <attribute name="Comment" value="${plugin.comment}"/>
                <attribute name="Filter" value="${plugin.filter}"/>
                <attribute name="GraphicSettings"
                       value="${plugin.graphic.settings}"/>
            </manifest>
        </jar>
    </target>

    <target name="clean" depends="clean.ga, clean.individual">
        <delete quiet="true" includeEmptyDirs="true">
            <fileset dir="${build}"/>
            <fileset dir="${deploy}"/>
        </delete>
    </target>

    <target name="clean.ga">
        <antcall target="clean.plugin">
            <param name="plugin" value="${ga.src}/roulette"/>
        </antcall>
    </target>

    <target name="clean.individual">
        <antcall target="clean.plugin">
            <param name="plugin" value="${individual.src}/SmartAnt"/>
        </antcall>
    </target>

    <target name="clean.plugin">
        <fail unless="plugin"/>
        <echo message="${plugin}"/>
        <property file="${plugin}/plugin.properties" prefix="plugin"/>
        <condition property="plugin.build" value="classes">
            <not>
                <isset property="plugin.build"/>
            </not>
        </condition>
        <delete quiet="true" includeEmptyDirs="true">
            <fileset dir="${plugin}/${plugin.build}"/>
        </delete>
    </target>
</project>
```