Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики
Факультет информационных технологий и программирования
Кафедра «Компьютерные Технологии»

Бережковская Е.Ю.

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

Вариант №2

Санкт-Петербург
2011 год

Оглавление

Введение

В лабораторной работе требовалось исследовать эффективность применения так называемой «большой мутации». Для этого реализовывался плагин для виртуальной лаборатории «3genetic» на языке Java, который реализует конкретный принцип отбора особей и конкретный принцип мутации особи.

1. Постановка задачи

Задача данной лабораторной работы — исследовать зависимость эффективности работы генетического алгоритма, строящего автомат Мура для решения задачи о роботе, от шага *M* «большой мутации» («большая мутация» — это мутация или замена случайными 90% особей в поколении каждые *M* поколений). Критерий оценки эффективности автомата заключается в том, что робот должен доходить до цели за возможно меньшее количество шагов.

1.2. Задача о роботе, обходящем препятствия

Дано фиксированное плоское поле с препятствиями, размера 32×32, и робот, который видит клетку прямо перед собой. Робот может совершить следующие действия: пойти вперед (если там препятствие, то он останется на месте), повернуть налево, повернуть направо, ничего не делать. На поле заданы стартовая и целевая клетки для робота. Необходимо построить автомат Мура, который приведет робота к цели за наименьшее число действий (желательно, меньше чем 200).

2. Реализация

Виртуальная лаборатория состоит из ядра и подключаемых модулей. Для решения поставленной задачи требуется реализовать новый модуль генерации нового поколения, так как кроссовер и мутация уже реализованы в лаборатории.

2.1. Метод генерации очередного поколения

Начальное поколение состоит из фиксированного числа (150) случайно сгенерированных автоматов. Все автоматы в поколении имеют одинаковое наперед заданное число состояний (а именно восемь). Для генерации очередного поколения используется традиционный генетический алгоритм, если не пришло время большой мутации.

Большая мутация реализована следующим образом: из текущего поколения выбираются 90% произвольных особей и заменяются случайными.

Фитнес-функция реализована так: для текущей точки считается, сколько шагов осталось. Точнее, роботу дается возможность сделать не более чем 250 шагов, если он не достигнет цели ранее. Затем считается манхэттенское расстояние до цели и в качестве результата берется минус это расстояние минус количество сделанных шагов, деленное на 250.

$f(x,y) = -(|x-X|+|y-Y|) - \frac{steps}{250}$, где ***steps*** — число сделанных шагов, (*X*, *Y*) — цель.

3. Результаты

В ходе эксперимента для каждого значения *M* генерировались 50000 поколений. Делалось по 20 запусков, затем значения фитнес-функции по всем запускам усреднялись. Эксперимент показал, что с ростом шага «большой мутации» эффективность алгоритма падает, несмотря на то, что значение фитнес-функции у лучшей особи не убывало (см. рис. 1). Однако при совсем небольших *M* (меньших семи) поведение алгоритма качественно изменилось (см. рис. 2). Подобный эффект объясняется тем, что за такое небольшое число итераций достаточно хорошая особь просто не успевает появится. Лучшие результаты наблюдались при $M = 7$, среднее значение фитнес-функции достигало $-0{,}524$, т.е. робот достигал цели в среднем за 133 шага. Затем с ростом *M* число шагов возрастало. Наблюдалось некое улучшение при *M* = 500 и *M* = 600, однако оно было незначительным. Возможно, это следствие некоторой погрешности, неизбежной при ограниченном числе экспериментов.

В результате получилась следующая зависимость среднего значения фитнес-функции от шага «большой мутации», изображенная на рисунке 3.

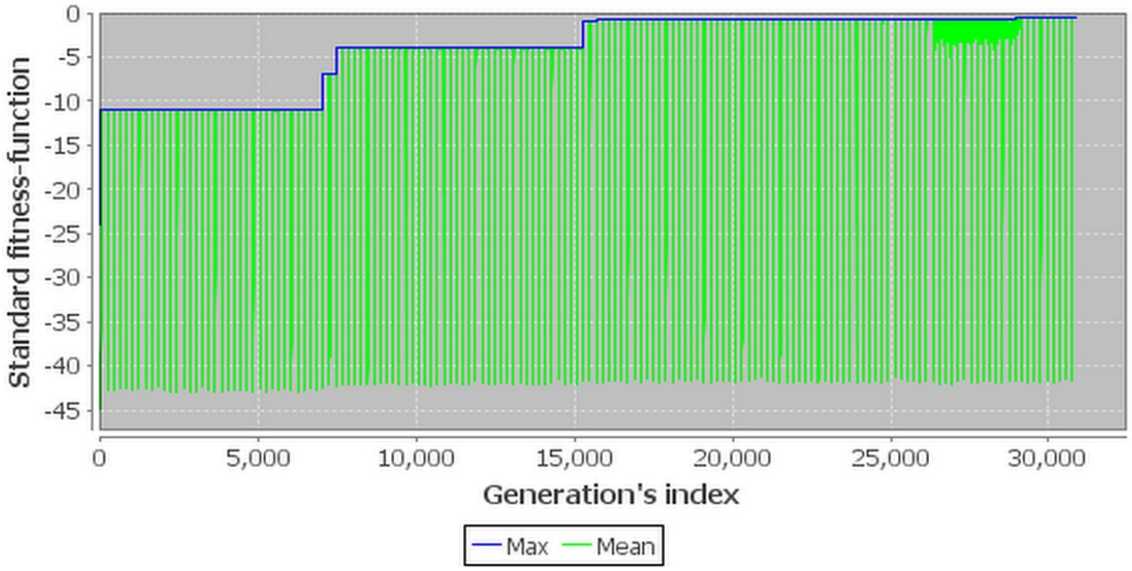

Рис. 1 — M = 200

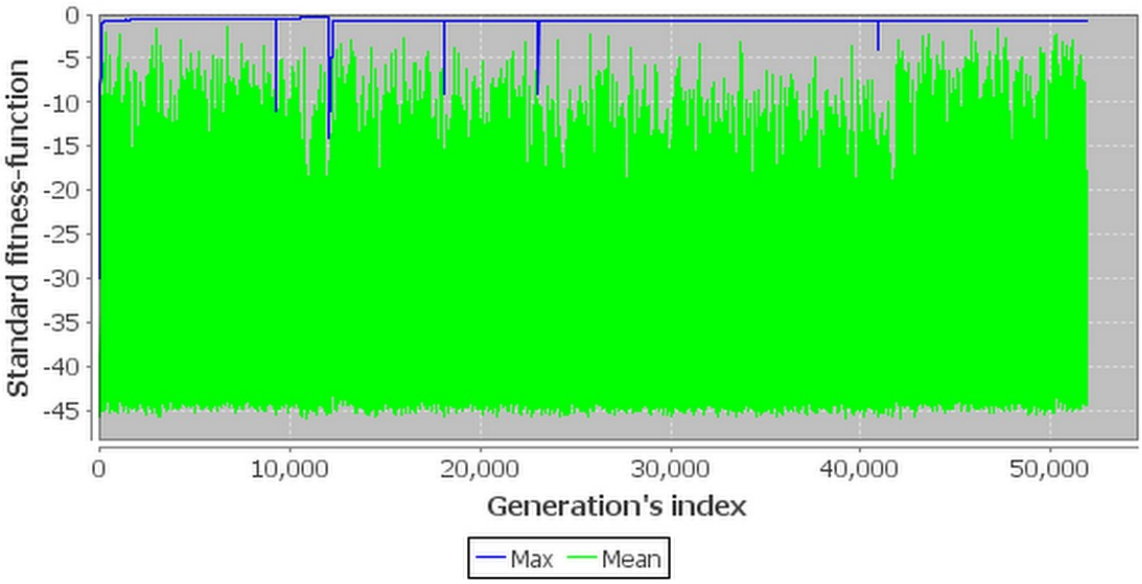

Рис. 2 — M = 6

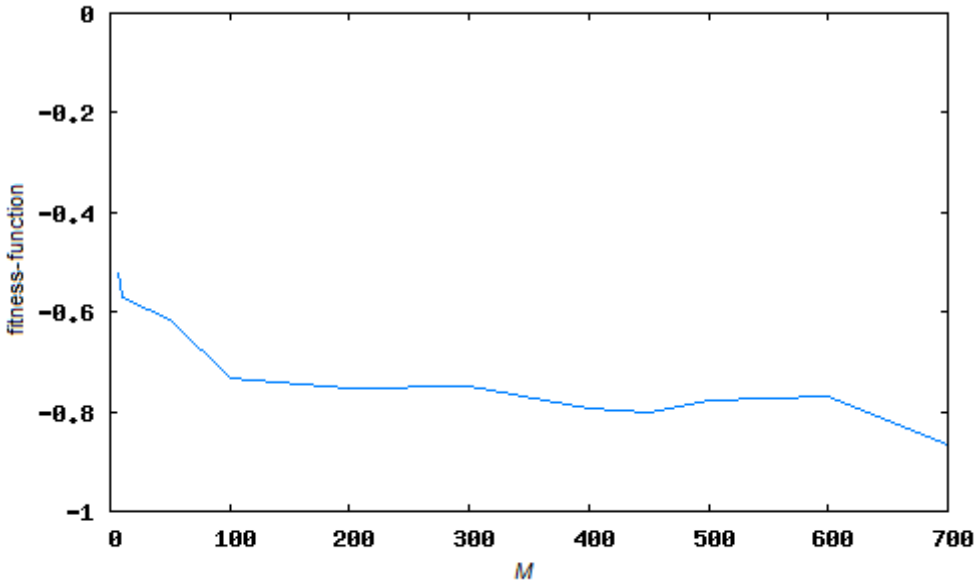

Рис. 3 — Значение фитнес-функции

Лучшие автоматы доходили до цели за 110 шагов. Один из таких автоматов изображен на рис. 4 (чтобы рисунок не был перегружен, над переходами не подписаны символы; черные стрелки означают переход по символу «0», красные — по символу «1», т.е. соответственно «нет стены» и «есть стена»).

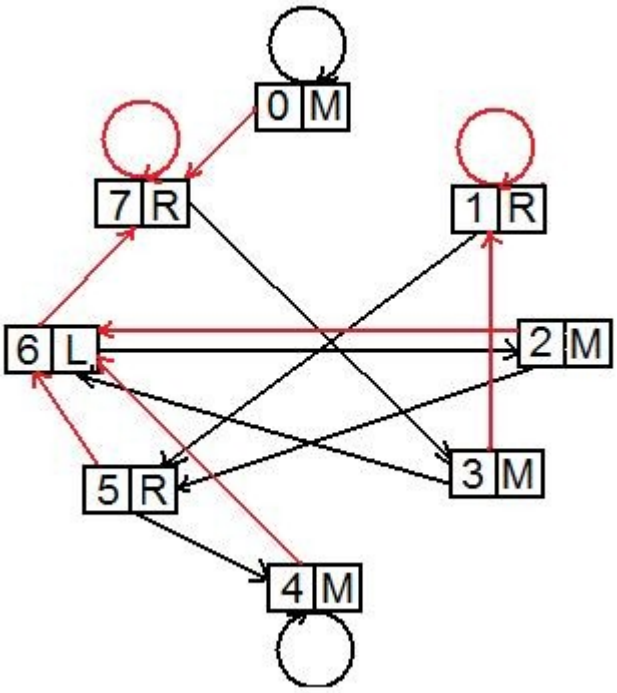

Рис. 4 — Автомат, доходящий до цели за 110 шагов

Маршрут робота на поле приведен на рис. 5 (синяя клетка — старт, зеленая — финиш, черные клетки — стены, точками изображен маршрут).

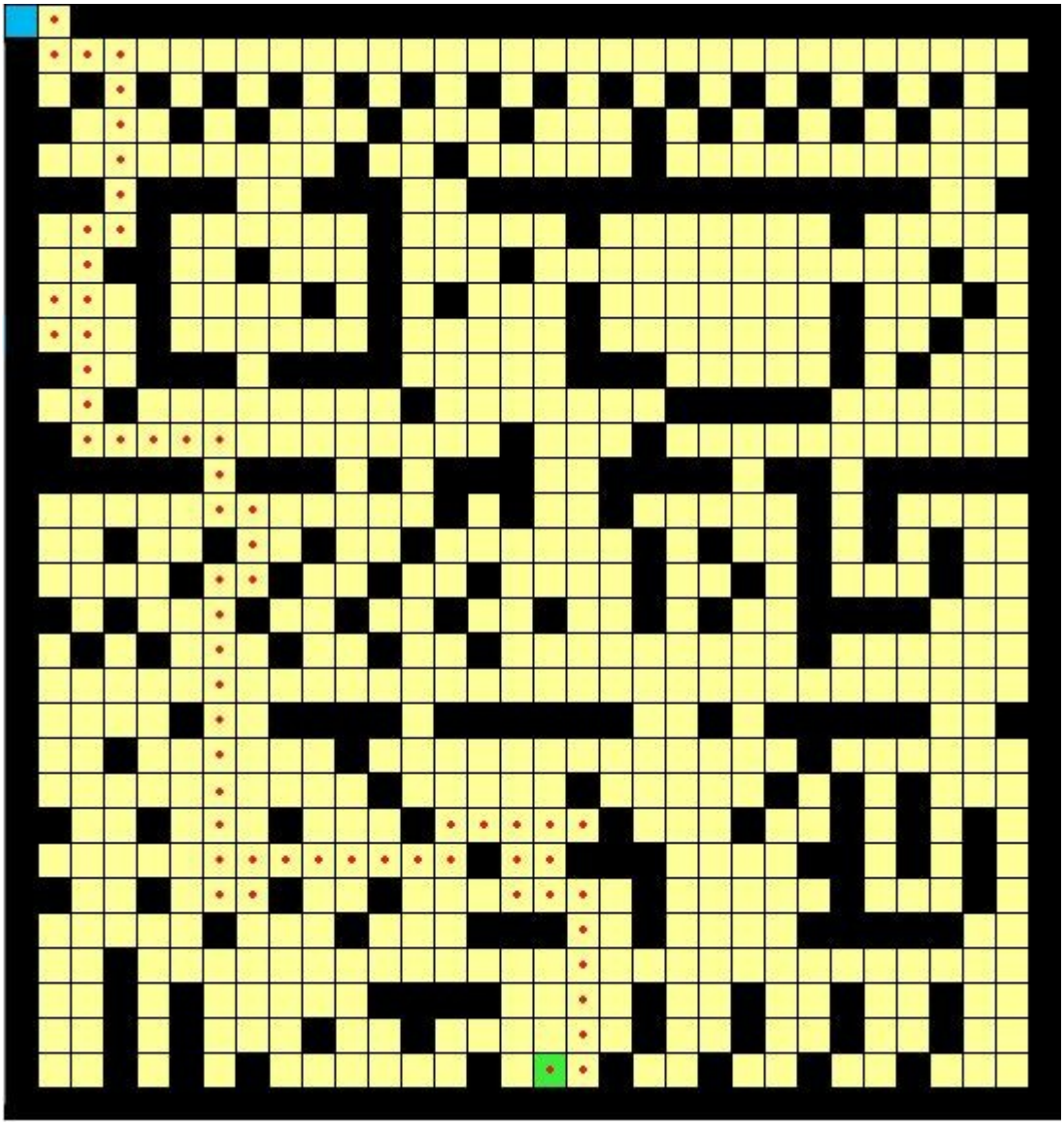

Рис. 5 — Маршрут на поле

Приложение. Исходный код

Файл CustomGA.java

```
package laboratory.plugin.algorithm.custom;

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 CustomGA<I extends Individual> implements Algorithm<I>{

    private List<FitIndividual<I>> generation;

    private final double probabilityMutation;
    private final int bigMutationStepSize;
    private final double elitismSize;

    private final IndividualFactory<I> factory;

    private final Random r;

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

    private final int fixedPart;
```

```
    private int counter;

    public CustomGA(final List<I> generation, final IndividualFactory<I>
factory, final Mutation<I> mut, final Crossover<I> cross, final
Selection<I> sel, final Fitness<I> fitness){


        this.probabilityMutation =
Config.getInstance().getMutationProbability();
        this.elitismSize = Config.getInstance().getElitismSize();
        this.bigMutationStepSize =
Config.getInstance().getBigMutationSteps();


        this.factory = factory;


        this.mut = mut;
        this.cross = cross;
        this.sel = sel;
        this.fitness = fitness;
        this.bestIndividual = generation.get(0);
        this.counter = 0;


        this.generation = Util.map(generation, new Functor1<I,
FitIndividual<I>>(){
            public FitIndividual<I> apply(I i){
                return cons(i);
            }
        });
        Collections.sort(this.generation);
        fixedPart = generation.size() / 20;


        r = new Random();
    }


    public CustomGA(final int sizeGeneration, final IndividualFactory<I>
factory, final Mutation<I> mut, final Crossover<I> cross, final
Selection<I> sel, final Fitness<I> fitness){
        this(Util.listFromFunctor(new Functor0<I>(){
                    public I apply(){
                        return factory.getIndividual();
                    }
                }, sizeGeneration),
             factory, mut, cross, sel, fitness);
    }
```

```
    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(){
      if (counter == bigMutationStepSize) {
            counter = 0;
            bigMutation();
      } else {
              int size = generation.size();
              int elite = (int) Math.round(elitismSize * size);

              List<FitIndividual<I>> newGeneration = new
ArrayList<FitIndividual<I>>(size);
              for (int i = 0; i < elite; i++) {
                 newGeneration.add(generation.get(i));
              }
              newGeneration.addAll(sel.apply(generation, fixedPart));
              while(newGeneration.size() < size){
                  List<I> s = cross.apply(Arrays.asList(winner(randomI(),
randomI()), winner(randomI(), randomI())));
                  for(I ind : s){
                      if (newGeneration.size() == size)
                            break;
                      newGeneration.add(cons(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);
            counter++;

    }

    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(){
      List<Integer> individuals = new ArrayList<Integer>();
      for (int i = 0; i < generation.size(); i++){
            individuals.add(i);
      }
      Collections.shuffle(individuals);
      double nSize = 0.9*generation.size();
      int newSize = (int) Math.round(nSize);
      List<FitIndividual<I>> newGeneration = new
ArrayList<FitIndividual<I>>();

      for (int i = 0; i < newSize; i++) {
            newGeneration.add(cons(factory.getIndividual()));
      }
      for (int i = newSize; i < generation.size(); i++){
            newGeneration.add(generation.get(individuals.get(i)));
      }
            generation = newGeneration;
         Collections.sort(generation);
    }
}
```

Файл StandartFitness.java

```
package laboratory.plugin.task.robot;

public class StandardFitness {

    public double calc(Mover solution) {
        Robot r = new SimpleRobot();
        solution.restart(r);
        final int maxSteps = (int) (1.25 * Robot.NUMBER_STEPS);
        int step = 0;
        for (; step < maxSteps; step++) {
            solution.move();
            if (r.getCurrent().equals(r.getTarget())) {
                break;
            }
        }

        int distance = Math.abs(r.getCurrent().x -
r.getTarget().x) +
                Math.abs(r.getCurrent().y - r.getTarget().y);
        return -distance - (double) step / maxSteps;
    }

    private static StandardFitness ourInstance = new
StandardFitness();

    public static StandardFitness getInstance() {
        return ourInstance;
    }

    private StandardFitness() {
    }
}
```