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

Волков И. Р.

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

Вариант 3

Санкт-Петербург
2011

Оглавление

1. Введение

В данной работе проведено сравнение эффективности работы генетического алгоритма при использовании оператора селекции рангового отбора и оператора селекции турнирного отбора. Для решения этой задачи создана небольшая программа на C++ и Python.

1.1. Постановка задачи

Необходимо исследовать, при каком виде отбора быстрее найдется автомат Мили, содержащий семь состояний, решающий задачу о роботе. Для этого необходимо по очереди запускать генетические алгоритмы для генерации автомата с использованием разных функций отбора.

1.2. Задача о роботе, обходящем препятствия

Дано фиксированное плоское поле с препятствиями, имеющее размер 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 Мили, содержащий семь состояний, который приведет робота к цели за наименьшее число действий (желательно, меньше чем 200).

2. Реализация

2.1. Состав программы

Программа состоит из двух частей: небольшой библиотеки cppGeneticLibrary, и нескольких скриптов на языке Python [1]:

- cppGeneticLibrary;
  - MealyRobot — автомат Мили, функция приспособленности;
  - RobotField — поле для движения робота;
  - MealyRobotCrossover — функция скрещивания;
  - MealyRobotMutation — функция мутации;
  - GenerationStorage — структура данных для хранения поколения (оболочка над std::vector с дополнительными возможностями);
  - RankSelection — функция рангового отбора;
  - TournamentSelection — функция турнирного отбора;
  - ElitismSplit — функция элитизма;
  - Main — файл, содержащий код, экспортирующий классы и функции в Python (с помощью boost::python).
- GenerateRobot — скрипт, в котором описан генетический алгоритм;
- еще несколько дополнительных скриптов (визуализатор обхода поля роботом, построитель графиков и другие).

2.2. Описание алгоритма

Алгоритм состоит из следующих частей:

1. Генерация начального поколения.
2. Элитизм (разделение поколения: 10 % лучших особей переходят в следующее поколение напрямую, остальные 90 % переходят к шагу 3).
3. Отбор (ранговый или турнирный).

4. Копирование: особи, выбранные элитизмом и отбором, копируются до тех пор, пока их число не достигнет 90 % от размера поколения.
5. Скрещивание.
6. Мутация.
7. Слияние получившихся особей с 10 %, отобранными элитизмом на шаге 2.

Если число сгенерированных поколений меньше 5000, то осуществляется переход к шагу 2. Схема алгоритма представлена на рис. 1.

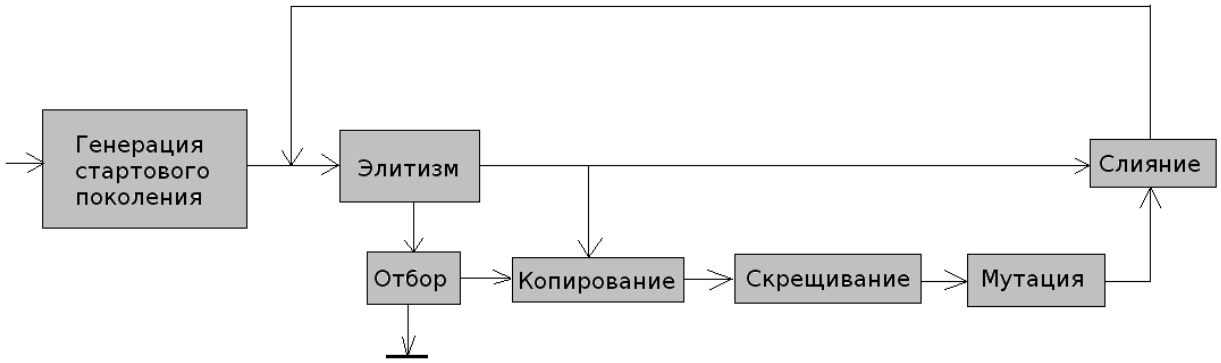

Рис. 1 — Схема алгоритма

2.3. Описание составных частей алгоритма

2.3.1. Скрещивание и мутация

Алгоритмы скрещивания и мутации реализованы также, как в виртуальной лаборатории [2].

2.3.2. Элитизм

Элитизм разбивает поколение на две части: 10 % с самой высокой функцией приспособленности и остальные.

2.3.3. Ранговый отбор

Отбор 12,5 % особей, лучших по рангу, то есть по функции приспособленности.

2.3.4. Турнирный отбор

Среди каждых восьми особей производится выбор одной по турнирной схеме. То есть сначала определяется победитель в каждой из четырех пар, затем выбранные четыре особи опять соревнуются в парах и, в заключение, соревнуются два оставшихся победителя. При соревновании в паре вероятность выигрыша особи пропорциональна ее функции приспособленности, то есть, если функция приспособленности первой особи равна f, функция приспособленности второй особи равна g, то вероятность выигрыша для первой особи равна $\frac{f}{f+g}$.

В результате турнирного отбора выбирается 12,5 % особей.

2.3.5. Функция приспособленности

Значения функций приспособленности особей рассчитываются после формирования

поколения по следующей формуле: $f=(1-\frac{d_1}{d_2})\cdot\frac{200}{steps}$, где d_1 — это расстояние между целевой клеткой и клеткой, в которой робот остановился (робот выполняет 10000 действий, если он не доходит до цели раньше), d_2 — это расстояние между стартовой и целевой клетками, *steps* — число действий (шагов и поворотов), выполненных роботом.

Если робот доходит до цели ровно за 200 действий, то значение функции приспособленности равно единице. Если робот доходит до цели меньше, чем за 200 действий, то значение функции приспособленности больше единицы. В остальных случаях значение функции приспособленности меньше единицы.

3. Результаты

Программа произвела 130 запусков генерации автомата с использованием рангового отбора и 130 запусков с использованием турнирного отбора. При каждом запуске сгенерировано 5000 поколений (не включая начальное). В каждом поколении содержатся ровно 100 особей.

После всех запусков генерации автоматов для каждого отбора было рассчитано среднее арифметическое из функций приспособленности в зависимости от номера поколения (с использованием скрипта AverageGraphicCounter). Затем для каждого отбора был построен график среднего арифметического (при помощи скрипта GraphicsViewer). Оба графика представлены на рис. 2.

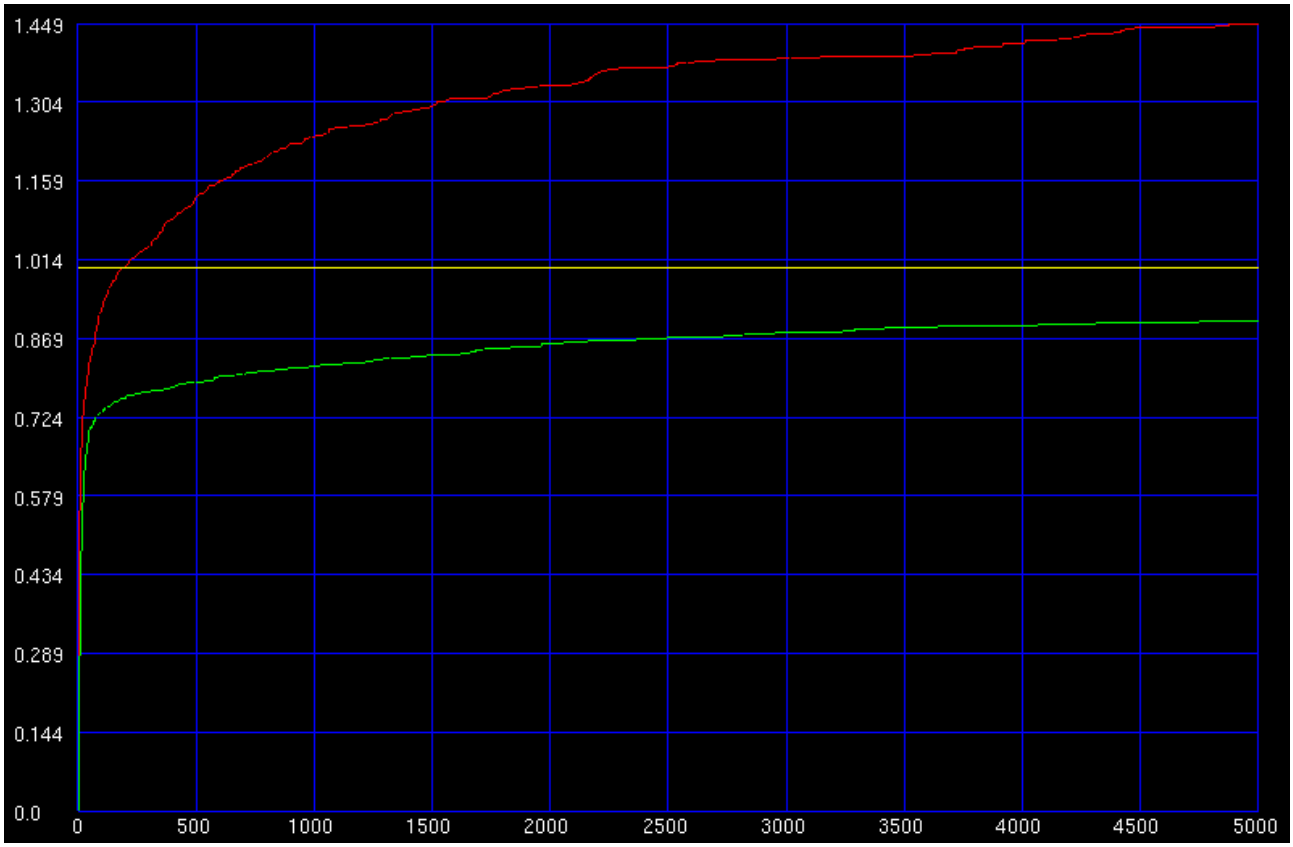

Рис. 2 — Среднее арифметическое функций приспособленности за 130 запусков:
красная линия — ранговый отбор, зеленая — турнирный,
желтая — значение единицы на вертикальной оси.
На горизонтальной оси отмечено число поколений.

4. Заключение

Результат работы программы показал, что генетический алгоритм, использующий ранговый отбор, среднестатистически создает особь с большей функцией приспособленности за меньшее число поколений, чем алгоритм, использующий турнирный отбор. Особь, имеющую наибольшую функцию приспособленности, сгенерировал алгоритм, использующий ранговый отбор. Маршрут робота, использующего эту особь, представлен на рис. 3.

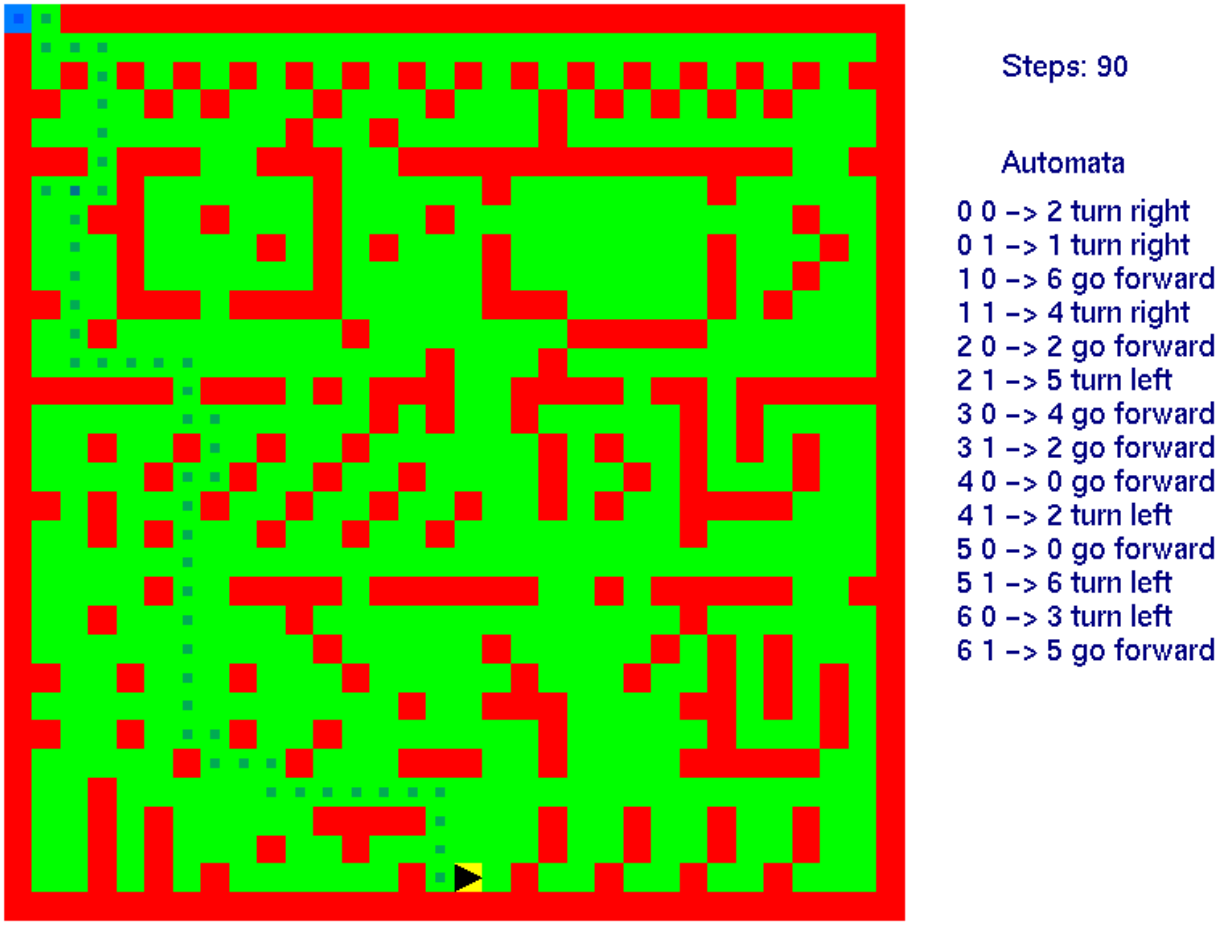

Рис. 3 — Маршрут робота, использующего особь с наибольшей функцией приспособленности (число действий робота равно 90).

5. Источники

1. Библиотека cppGeneticLibrary и скрипты
   http://www.kt15.ru/svn/volkovGeneticLab; логин: anonymous; пароль: 12345
2. Виртуальная лаборатория «3genetic»
   http://rain.ifmo.ru/~buzdalov/lab-2011/3genetic.zip

6. Приложение. Исходный код генетического алгоритма

GenerateRobot.py

```
#! /usr/bin/env python

import os
import sys
import cppGeneticLibrary

GENERATION_SIZE = 100
STATES_COUNT    = 7

FITNESS_GOAL          = 1.0
MAX_GENERATIONS_COUNT = 2000
ELITISM_SPLIT_QUOTA   = 0.1
RANK_SELECTION_QUOTA  = 0.125

def generateRobot(fieldFileName, selectionType, fileNameToSaveRobotTo,
maxGenerationsCount = MAX_GENERATIONS_COUNT):
    assert os.path.isfile(fieldFileName), "File [{0}] does not
exist".format(fieldFileName)

    cppGeneticLibrary.initRandomizer()
    robotField = cppGeneticLibrary.RobotField(fieldFileName)

    generation = cppGeneticLibrary.GenerationStorage()
    for i in xrange(GENERATION_SIZE):
        mealyRobot = cppGeneticLibrary.MealyRobot(STATES_COUNT, True)
        generation.push(mealyRobot)

    bestFitness = 0.0
    generationNumber = 0

    fitnessArray = [0.0]

    while generationNumber < maxGenerationsCount:
        generationNumber += 1

        generation.calcFitnessForEach(robotField)

        bestFitness = generation.getBestCalculatedFitness()

        print "Generation {0}; best fitness = {1}".format(generationNumber,
bestFitness)
        fitnessArray.append(bestFitness)

        if generationNumber >= maxGenerationsCount:
            break

        generationSplit = cppGeneticLibrary.elitismSplit(generation,
ELITISM_SPLIT_QUOTA)

        generationSplitBSize = generationSplit.b.size()
        if selectionType == "rank":
            selection = cppGeneticLibrary.rankSelection(generationSplit.b,
RANK_SELECTION_QUOTA)
        elif selectionType == "tournament":
            selection = cppGeneticLibrary.tournamentSelection(generationSplit.b)
```

```
        else:
            assert False, "Selection [{0}] unknown".format(selectionType)

        selection.append(generationSplit.a)
        selectionCopy = cppGeneticLibrary.GenerationStorage(selection)  # making
a copy (not a pointer)
        while selection.size() < generationSplitBSize:
            selection.append(selectionCopy)
        selection.trimToSize(generationSplitBSize)
        selection.shuffle()

        crossoverProduct = cppGeneticLibrary.mealyRobotCrossover(selection)

        mutationProduct = cppGeneticLibrary.mealyRobotMutation(crossoverProduct)
        mutationProduct.append(generationSplit.a)

        generation = mutationProduct

    #saving fitness values anyway
    with open(fileNameToSaveRobotTo + "_fitness.txt", "w") as f:
        for i in xrange(len(fitnessArray) - 1):
            f.write("{0} ".format(fitnessArray[i]))
        f.write("{0}".format(fitnessArray[len(fitnessArray) - 1]))

    if bestFitness >= FITNESS_GOAL:
        print "Best automata:"
        best = generation.getBest()
        best.debugPrint()
        best.printFieldWithWay(robotField)
        best.saveToFile(fileNameToSaveRobotTo)
        return generationNumber

    return -1

if __name__ == "__main__":
    if len(sys.argv) < 4:
        print "Usage: GenerateRobot.py <field_file_name> <selection_type (rank
or tournament)> <file_name_to_save_robot_to> [<max_generations_count>]"
        sys.exit(0)

    if not os.path.isfile(sys.argv[1]):
        print "Error: Field file does not exist"
        sys.exit(1)

    if sys.argv[2] != "rank" and sys.argv[2] != "tournament":
        print "Error: Selection type incorrect"
        sys.exit(1)

    maxGenerationsCount = MAX_GENERATIONS_COUNT
    if len(sys.argv) == 5:
        maxGenerationsCount = int(sys.argv[4])

    generations = generateRobot(sys.argv[1], sys.argv[2], sys.argv[3],
maxGenerationsCount)

    if generations == -1:
        print "Could not generate robot in {0}
generations".format(maxGenerationsCount)
    else:
        print "OK. Generations: {0}".format(generations)
```