Санкт-Петербургский государственный университет
информационных технологий, механики и оптики
Факультет информационных технологий и программирования
Кафедра компьютерных технологий

Д.А. Чащин

Отчет по лабораторной работе «Автоматное программирование и генетические алгоритмы на примере задачи об «Умном муравье»

Вариант № 1

Санкт-Петербург
2010

Оглавление

Введение

Цель лабораторной работы – построить с помощью генетических алгоритмов конечный автомат Мили, решающий задачу об «Умном муравье».

При выполнении работы использовался программный комплекс для изучения методов глобальной оптимизации `GlOpt`[1], разработанный студентами кафедры «Компьютерные технологии» СПбГУ ИТМО. Данный комплекс позволяет реализовывать генетический алгоритмы в виде исполняемых плагинов и изучать процесс их выполнения.

1. Постановка задачи

Задача данной лабораторной работы – построить «умного муравья», представленного в виде автомата Мили. Управляемый построенным автоматом муравей должен успевать съесть всю еду на поле за 200 шагов.

1.1. Задача об «Умном муравье»

Игра происходит на поверхности тора размером 32x32 клетки. В некоторых клетках находится еда. За ход муравей может выполнить следующие действия:

- повернуть налево;
- повернуть направо;
- сделать шаг вперед и, если в новой клетке есть еда, съесть ее;
- ничего не делать.

Игра длится 200 ходов. Цель задачи – найти муравья, который за минимальное число ходов ест как можно больше яблок. Всего яблок 89. Желательно чтобы в автомате, представляющем муравья, число состояний было минимально. Вид игрового поля показан на рис. 1.

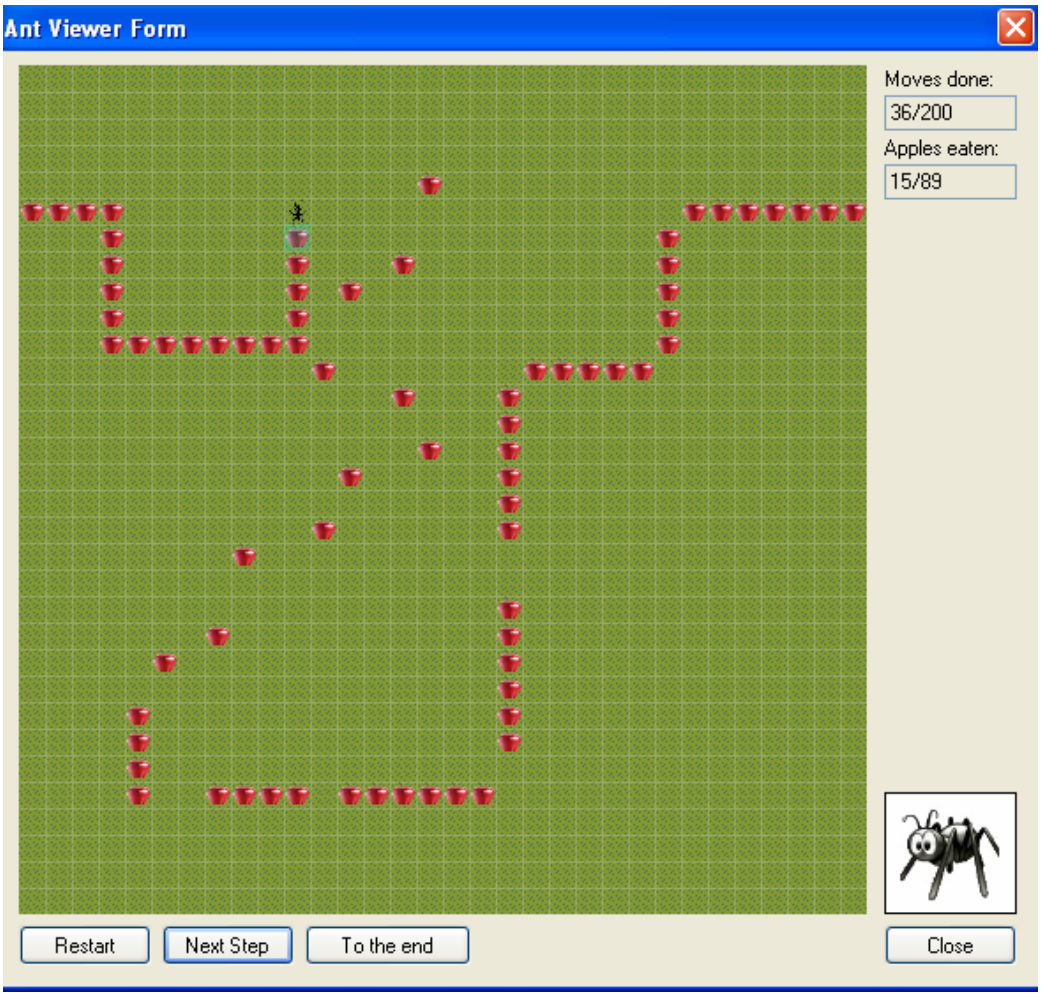

Рис. 1

2. Реализация

Для реализации алгоритма с помощью *GlOpt* необходимо реализовать два класса, наследованные соответственно от `SearchOperator` и `GeneticOptimizationAlgorithm`.

2.1. Представление автоматов

Автомат будем представлять в виде бинарной строки, хранящейся в массиве `bool value[]`, следующим образом: длина массива `равна(Length+2)*2*N`, где `N` – число состояний в автомате, `Length` – максимальное число знаков в двоичном представлении числа `N`. В первых `(Length+2)*N` битах содержатся переходы по входному воздействию «впереди нет яблока», в следующий `(Length+2)*N` битах содержатся переходы по входному воздействию «впереди есть яблоко». Каждый переход закодирован следующим образом:

0 1 0 0 1 . . . 1 0 01

Первые `Length` символов (выделено жирным шрифтом) кодируют номер следующего состояний. Последние два символа кодируют выходное воздействие

0 – переход в следующую клетку;
1 – поворот влево;
2 – поворот вправо.

Если случается что это значение равно трем, то производится случайное действие.

2.2. Метод скрещивания

Все особи текущего поколения размещаются на рулетке, причем каждому муравью отводится сектор, размером пропорциональный его фитнес-функции. Запуская рулетку `2*n` раз, разбиваем особи на `n` пар. Далее каждая пара скрещивается с вероятностью `CrosoverProbability`. Для скрещивания берем двух особей, делим двоичный код каждой из них на две части в произвольном, одинаковом для каждой из них, месте. И меняем получившиеся части местами.

Рассмотрим пример. Возьмем две особи:

0 1 1 0 1 1 1 1 10 0 1 0 1 1 0 1
1 0 1 1 1 **0 0 0 0 0 0 1 0 1 1 0 1**

Получим две новые:

0 1 1 0 1 0 0 0 0 0 0 1 0 1 1 0 1
1 0 1 1 1 1 1 1 10 0 1 0 1 1 0 1

Новые особи записываются в следующее поколение. Те особи, которые не скрещивались, сразу переходят в следующее поколение.

2.3. Метод мутации

Каждый бит в особи меняется на противоположный с вероятность `MutationProbability`.

2.4. Метод генерации очередного поколения

В начале генерируется `X` случайных особей. Число `X` фиксировано и остается неизменным на протяжении работы всего алгоритма. На каждом шаге алгоритма производится операция скрещивания, описанным ранее методом. Затем операция мутации.

В случае если значение фитнес-функция не изменилась на протяжении 500 поколений, производится большая мутация. Таким образом, все особи, за исключением самой приспособленной, заменяются на новые, построенные случайным образом. Наилучшая особь добавляется в новую популяцию два раза.

Если даже по прошествии 50 больших мутаций наибольшее значение фитнес-функции не изменилось, то производится очень большая мутация. Все особи, кроме наилучшей, заменяются на новые, а наилучшая мутирует с вероятностью 80%.

2.5. Способ вычисления функции приспособленности

Будет рассчитывать фитнес-функцию по формуле:

$$fitnes = \text{Число съеденных яблок} - \frac{\text{Ход, на котором, муравей последний раз съел яблоко}}{\text{Число ходов по условию задачи}}$$

3. Результаты работы модуля

В результате работы генетического алгоритма был получен автомат Мили из восьми состояний, который решает задачу об «Умном муравье». Значение функции приспособленности полученного «муравья» равно 88,01.

3.1. Графики максимального, среднего и минимального значений функции приспособленности

На рис. 2 изображены графики максимального (красным цветом), среднего (синим цветом) и минимального (зеленым цветом) значений фитнес-функции особей за время работы генетического алгоритма.

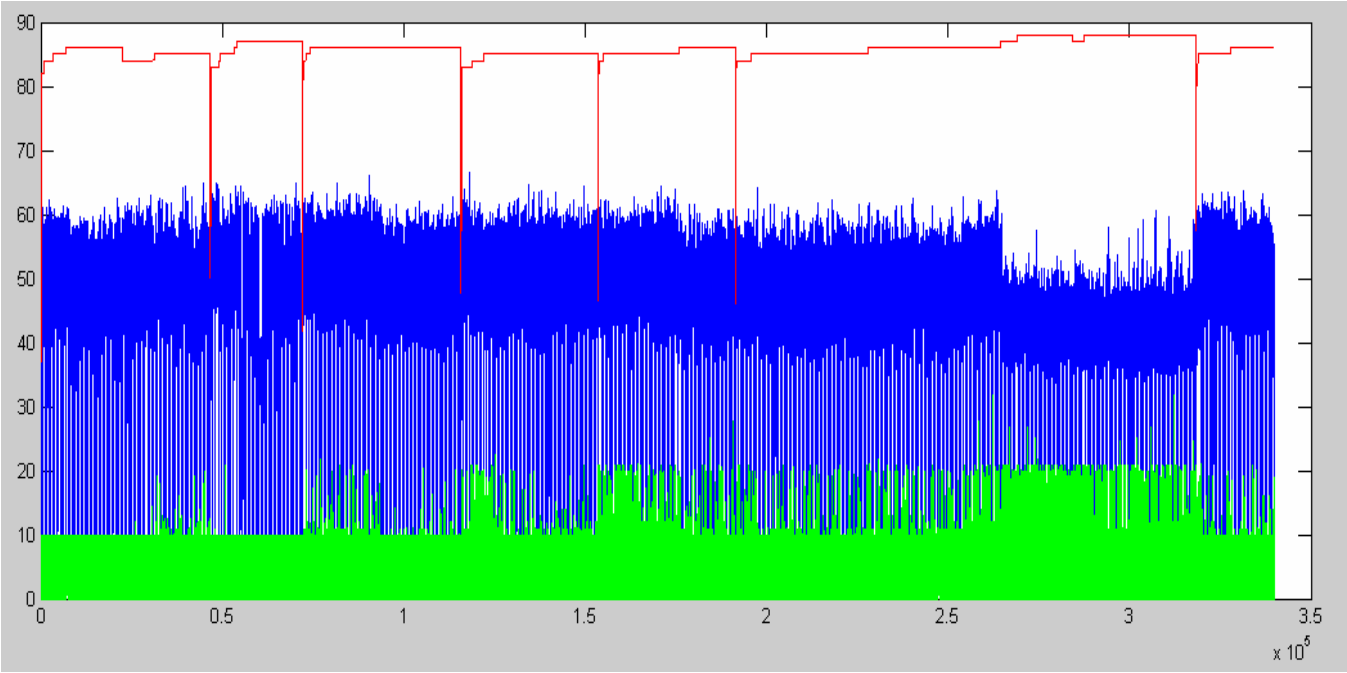

Рис. 2

3.2. Автомат в двоичном представлении

В двоичном представлении полученный автомат имеет следующий вид:

0110 0010 1100 1110 1001 0110 1011 0001 1110 0001 0000 0000 0000 1110 0011 0011 0000 1000 0100 0111

3.3. Граф переходов полученного автомата

Граф переходов автомата, построенного в результате работы генетического алгоритма, представлен на рис. 3. Символы 0 и 1, поступающие на вход автомата, соответствуют значениям входных воздействий «впереди нет яблока» и «впереди есть яблоко».

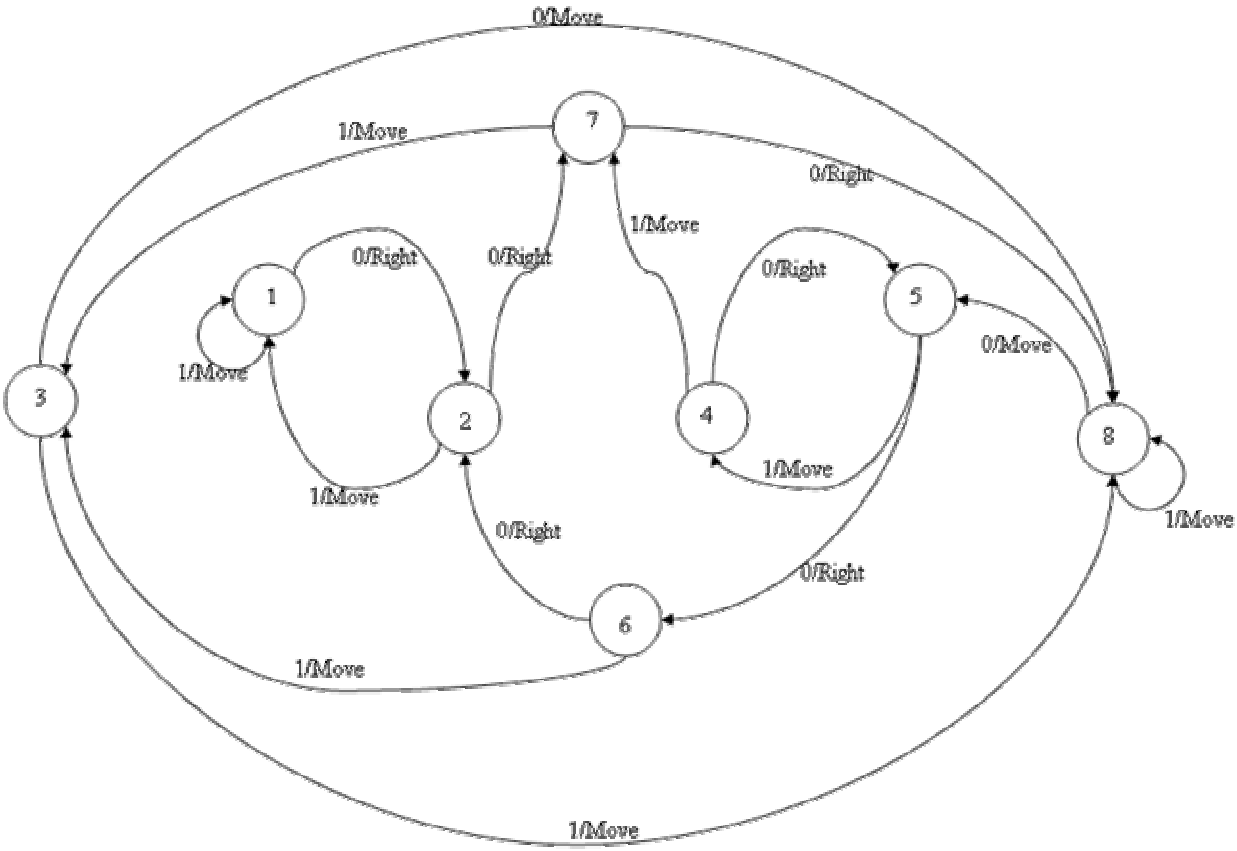

Рис. 3

Заключение

В результате работы построен автомат Мили из восьми состояний, кодирующий «муравья», который успевает съесть все яблоки менее чем за 200 шагов. Для работы алгоритма потребовалось около шести часов, что в несколько раз меньше ожидаемого времени поиска методом перебора.

Источники

Документация к комплексу для изучения методов глобальной оптимизации `GlOpt`.
http://is.ifmo.ru/courses/_giopt-src.rar