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

О. С. Ларионов

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

Вариант №8

Санкт-Петербург
2011 г.

Оглавление

Введение 3
1. Постановка задачи 3
    1.1. Задача об «Умном муравье-3» 3
2. Реализация 3
    2.1. Операторы скрещивания и мутации 3
    2.2. Функция приспособленности 3
    2.3. Описание генетического алгоритма 4
    2.3. Разработанный метод отбора 4
    2.4. Турнирный метод отбора 4
3. Результаты работы 4
    3.1. Анализ стратегии 4
    3.2. Вариация p 6
Заключение 7
Источники 7
Приложение 1. Исходный код 8
    Файл MySelectionStrategy.java 8
Приложение 2. Выращенные автоматы 9
    Предикаты 9
    Автомат 1 9

Введение

В лабораторной работе требуется разработать метод отбора особей в следующее поколение, применить его к решению задачи об «Умном муравье-3» и сравнить с турнирным методом отбора. Для решения задачи используются библиотеки WatchMaker Framework и собственные модули, реализованные на языке программирования Java.

1. Постановка задачи

Задача лабораторной работы — разработать метод отбора особей в следующее поколение. В данной работе особями являются автоматы Мура, решающие задачу об «Умном муравье-3». Метод отбора должен поддерживать высокое разнообразие особей и не должен уступать по производительности «классическим» методам отбора.

1.1. Задача об «Умном муравье-3»

Дано поле размером 32 на 32 клетки, расположенное на поверхности тора. В некоторых клетках случайным образом расположена еда. Суммарно на поле 89 единиц еды. Муравей изначально находится в стартовой клетке и за один шаг может сделать одно из следующих действий:

- Повернуть налево
- Повернуть направо
- Сделать шаг вперед

Если муравей приходит в клетку с едой, то он съедает ее. Муравей видит впереди себя так, как показано на рис. 1. Всего делается 200 шагов. Требуется построить автомат с фиксированным числом состояний, который съедает как можно больше еды.

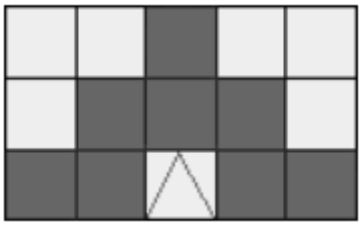

Рис. 1 — Схема «зрения» муравья

2. Реализация

Для решения поставленной задачи были реализованы представление автомата Мура в скрещенных таблицах, операторы скрещивания и мутации, а также стратегия отбора.

2.1. Операторы скрещивания и мутации

Операторы скрещивания и мутации автоматов Мура в сокращенных таблицах, примененные в алгоритме, достаточно подробно описаны в [2].

2.2. Функция приспособленности

Пусть a_1, a_2,…, a_n — количество еды, съеденной муравьем под управлением автомата A на каждом из n полей. Тогда функция приспособленности этого автомата

$$f(A) = \frac{\sum_{i=1}^{n} a_i}{n}.$$

2.3. Описание генетического алгоритма

Генетический алгоритм состоит из нескольких шагов. На первом шаге генерируются m случайных автоматов. Далее, пока не наступит условие завершения, выполняется следующая процедура:

1. Происходит скрещивание особей текущего поколения. Из двух автоматов получаются два новых автомата.
2. Происходит мутация получившихся автоматов.
3. Для автоматов вычисляется функция приспособленности.
4. Применяется стратегия отбора и формируется следующее поколение.

2.3. Разработанный метод отбора

Выборка автоматов в следующее поколение производилась так. Каждому автомату была назначена его вероятность прохода в следующее поколение, равная значению его фитнесс-функции, разделенному на сумму значений фитнесс-функции всех автоматов текущего поколения. Далее, в соответствии с вероятностями, случайным образом выбирался один автомат. Он проходил в следующее поколение, а некоторая часть p его вероятности распределялась равномерно между всеми оставшимися автоматами. Если p равно нулю, то стратегия эквивалентна методу отбора «Рулетка». Всего в следующее поколение с помощью отбора переходило $m - m * k$ автоматов, где k — процент элитизма. Остальные $m * k$ выбирались элитизмом.

2.4. Турнирный метод отбора

Турнирный метод отбора, использованный в лабораторной работе, описан в [4]. Его реализация была взята из библиотек WatchMaker Framework.

3. Результаты работы

3.1. Анализ стратегии

Генерировались автоматы из десяти состояний и с двумя значимыми предикатами. Вероятности мутации и кроссовера равнялись 0,1. Было произведено десять запусков для разработанной стратегии и для метода турнирного отбора. Для вычисления фитнесс-функции использовалось 50 случайных полей, на каждом запуске алгоритма набор полей был свой, но наборы полей на запусках алгоритма с одинаковыми номерами для обеих стратегий совпадали. Условие завершения генерации — достижение 15000 поколения. Переменная p на всех запусках равнялась 0,16, элитизм — 5%. В качестве примера результаты трех запусков отражены на рис. 2-4. На этих графиках показано, каким было максимальное значение функции приспособленности на каждом поколении на первом, седьмом и десятом запусках алгоритма для двух стратегий отбора. Из них видно, что успешность стратегий различалась от запуска к запуску. Однако если усреднить по десяти запускам максимальные значения фитнесс-функции на каждом поколении для каждой стратегии, то будет видно, что значение фитнесс-функции на каком-либо поколении при разработанной стратегии обычно выше, чем соответствующее значение при турнирном отборе. Это отражено на рис. 5. Высокое разнообразие особей поддерживается, так как в следующее поколение может пройти любая особь, включая самую слабую, чего нет в турнирном отборе.

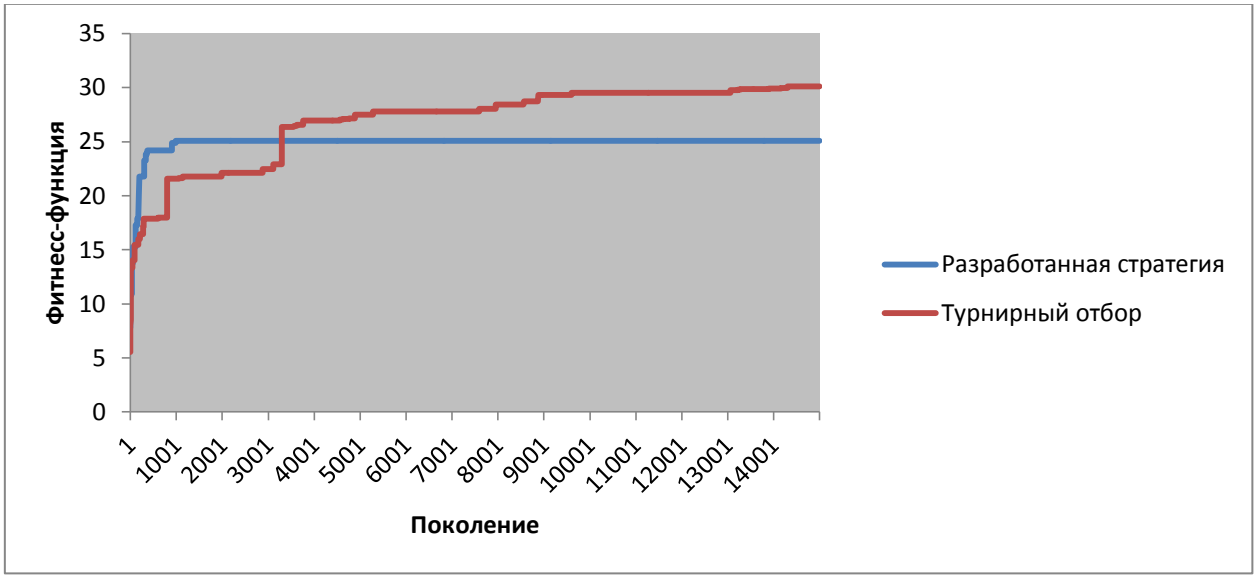

Рис. 2 — График зависимости максимального значения фитнесс-функции в поколении от номера поколения на первом запуске алгоритма.

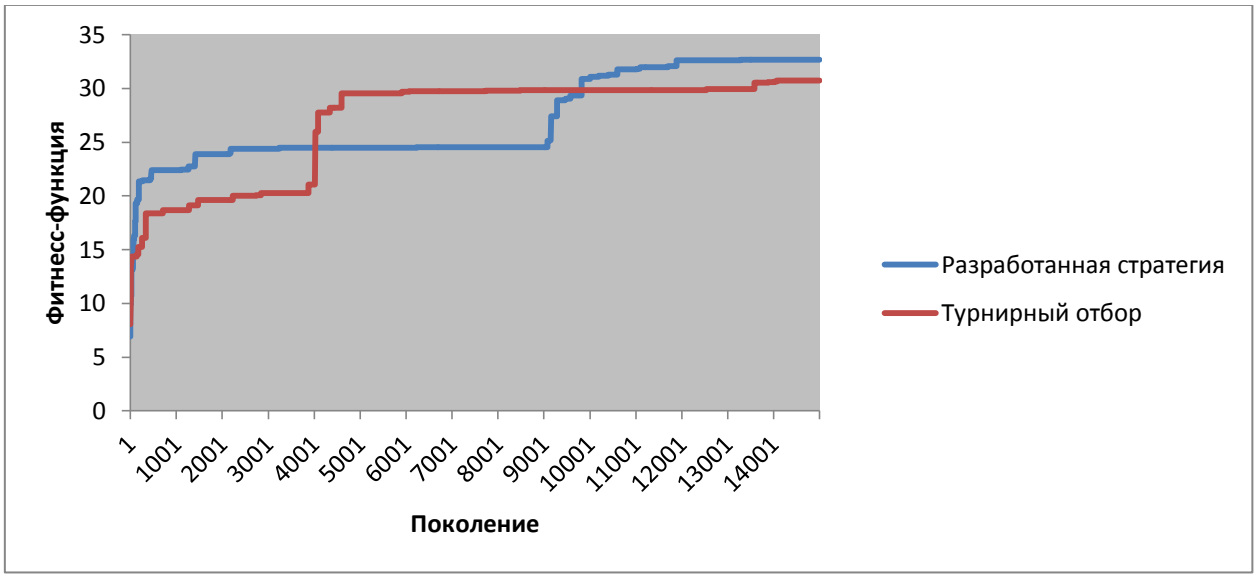

Рис. 3 — График зависимости максимального значения фитнесс-функции в поколении от номера поколения на седьмом запуске алгоритма.

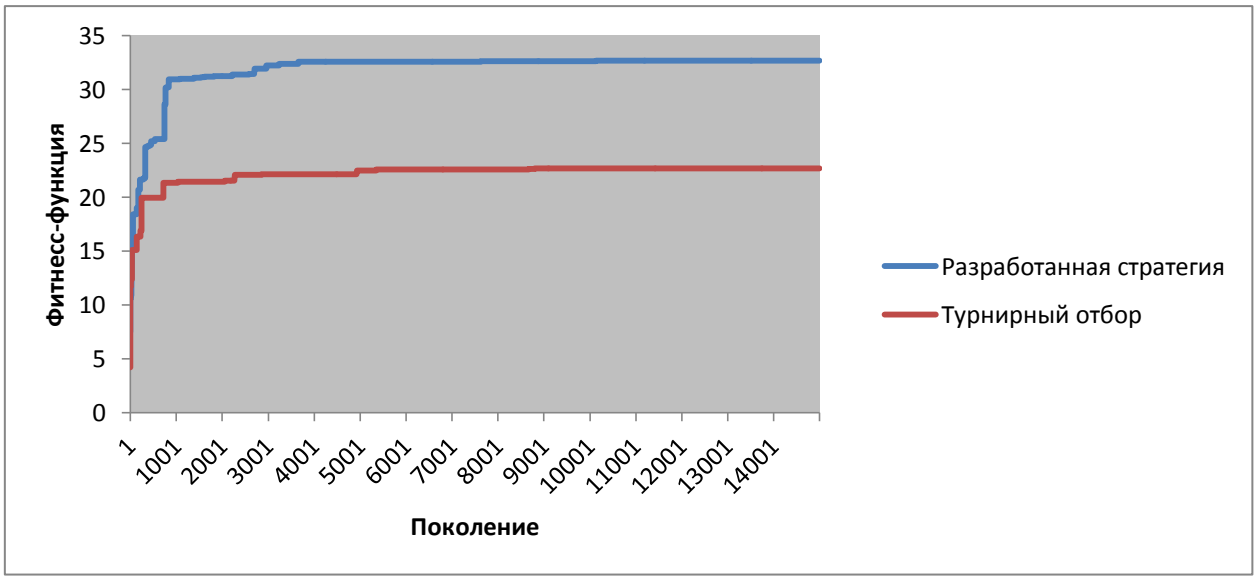

Рис. 4 — График зависимости максимального значения фитнесс-функции в поколении от номера поколения на десятом запуске алгоритма.

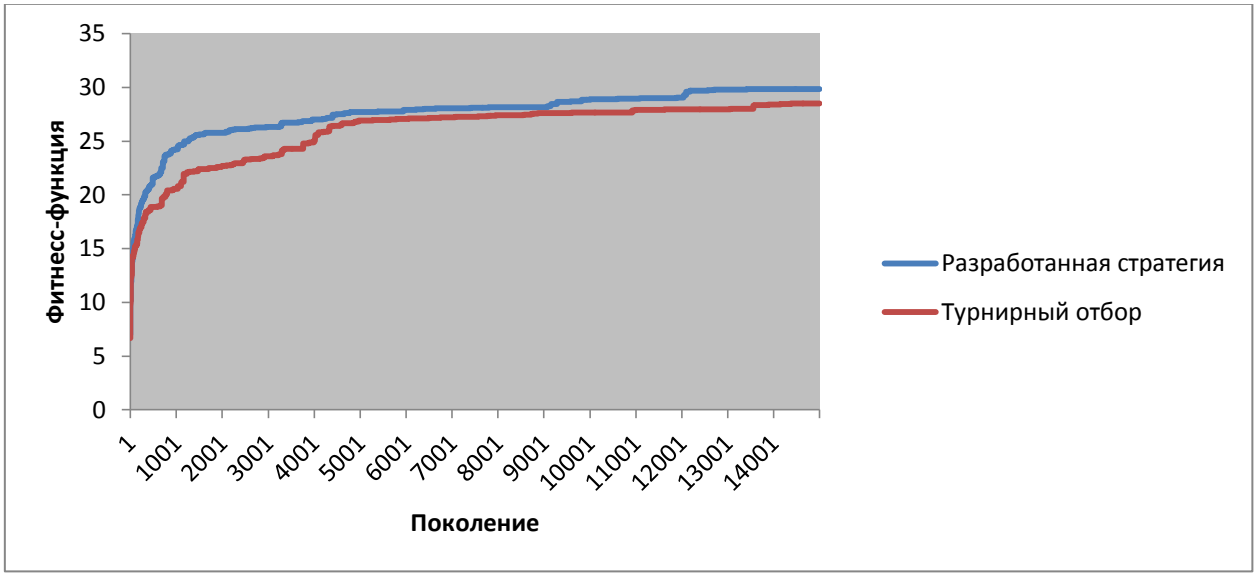

Рис. 5 — График зависимости среднего значения фитнесс-функции по десяти запускам от номера поколения.

3.2. Вариация *p*

В дополнение к анализу стратегии был проведен анализ влияния параметра *p* на эффективность метода отбора. Было проведено по пять запусков алгоритма для каждого значения *p* в диапазоне от 0 до 0,30 с шагом в 0,05. Получившиеся результаты можно увидеть на рис. 6. Из графика видно, что лучше всего алгоритм работает при *p* в диапазоне от 0,15 до 0,25.

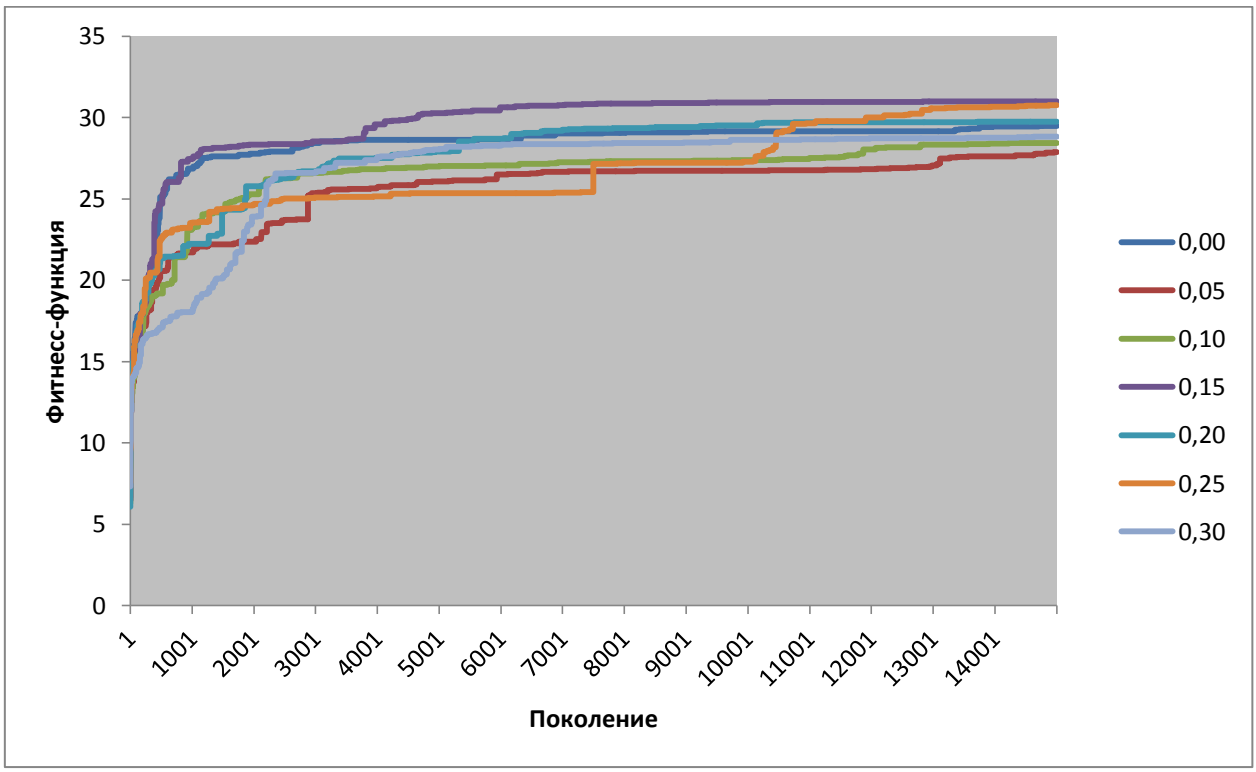

Рис. 6 — График роста среднего значения фитнесс-функции для разных значений *p*

Заключение

Как можно видеть из результатов, придуманная стратегия ничуть не уступает турнирному методу отбора, то есть автоматы достигают тех же значений функции приспособленности, что и в турнирной стратегии, а часто и превосходят их. Следовательно, разработанная стратегия применима к решению реальных задач и хорошо себя показывает при использовании в генетических алгоритмах.

Источники

1. *Поликарпова Н.И., Шалыто А.А.* Автоматное программирование. 2-к изд. СПб.: Питер, 2011.
2. *Поликарпова Н.И., Точилин В.Н., Шалыто А.А.* Метод сокращенных таблиц для генерации автоматов с большим числом входных переменных на основе генетического программирования. СПбГУ ИТМО, 2010.
3. *Документация WatchMaker Framework.* **http://watchmaker.uncommons.org/api/index.html**
4. *Mitchell M.* An Introduction for Genetic Algorithms. MIT Press, 1999.

Приложение 1. Исходный код

Файл MySelectionStrategy.java

```
package lab1;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import org.uncommons.watchmaker.framework.EvaluatedCandidate;
import org.uncommons.watchmaker.framework.SelectionStrategy;

public class MySelectionStrategy implements SelectionStrategy<Automata> {
	private double mult;
	public MySelectionStrategy(double mult) {
		this.mult = mult;
	}

	public <S extends Automata> List<S> select(
			List<EvaluatedCandidate<S>> cand, boolean isNatural,
			int numberToSelect, Random rnd) {

		double prob[] = new double[cand.size()];
		double sum = 0;
		if (!isNatural) {
			double max = cand.get(0).getFitness();
			for (int i = 0; i < cand.size(); ++i) {
				if (cand.get(i).getFitness() > max) {
					max = cand.get(i).getFitness();
				}
			}
			for (int i = 0; i < cand.size(); ++i) {
				prob[i] = max - cand.get(i).getFitness();
				sum += prob[i];
			}
		} else {
			for (int i = 0; i < cand.size(); ++i) {
				prob[i] = cand.get(i).getFitness();
				sum += prob[i];
			}
		}

		List<S> ans = new ArrayList<S>();

		for (int i = 0; i < numberToSelect; ++i) {
			double value = rnd.nextDouble() * sum;
			double cur = 0;
			for (int j = 0; j < cand.size(); ++j) {
				if (cur + prob[j] >= value) {
					ans.add(cand.get(j).getCandidate());
					double eq = mult * prob[j];
					prob[j] -= eq;
					eq /= cand.size() - 1;
					for (int k = 0; k < cand.size(); ++k) {
						if (k != j) {
							prob[k] += eq;
						}
					}
					break;
				} else {
					cur += prob[j];
				}
			}
		}

		return ans;
	}
}
```

Приложение 2. Выращенные автоматы

Предикаты

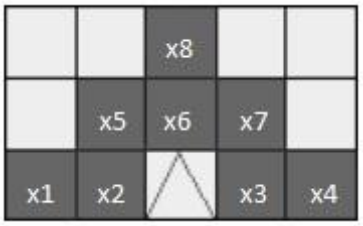

Рис. 6 — Предикаты и зона видимости муравья

Автомат 1

Выращен на десятом запуске. Автомат собирает в среднем 32,8 еды на 50 полях.

Таблица 1 — Значимые предикаты

$x1$	$x2$	$x3$	$x4$	$x5$	$x6$	$x7$	$x8$
1	1	0	0	0	0	0	0

Таблица 2 — Действия для состояний

1	2	3	4	5	6	7	8	9	10
Вперед	Вперед	Вперед	Направо	Налево	Налево	Направо	Направо	Налево	Направо

Таблица 3 — Переходы между состояниями

Исходное состояние	x1	x2	Переход
1	0	0	1
	0	1	1
	1	0	1
	1	1	6
2	0	0	7
	0	1	7
	1	0	7
	1	1	6
3	0	0	2
	0	1	8
	1	0	5
	1	1	5
4	0	0	0
	0	1	2
	1	0	2
	1	1	5
5	0	0	5
	0	1	1
	1	0	7
	1	1	4

Исходное состояние	x1	x2	Переход
6	0	0	0
	0	1	0
	1	0	1
	1	1	3
7	0	0	3
	0	1	8
	1	0	0
	1	1	2
8	0	0	2
	0	1	8
	1	0	5
	1	1	8
9	0	0	1
	0	1	3
	1	0	1
	1	1	1
10	0	0	8
	0	1	9
	1	0	4
	1	1	1