Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Факультет информационных технологий и программирования
Кафедра компьютерных технологий

Кулев Владимир Анатольевич

Метод генетического программирования на основе моделирования противоборствующих популяций автоматных программ

Научный руководитель: А. А. Шалыто, д.т.н., профессор

Санкт-Петербург
2010

ОГЛАВЛЕНИЕ

СПИСОК ТЕРМИНОВ

- *Генетический алгоритм* — эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путем последовательного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию.

- *Популяция* — совокупность индивидуумов, способная к устойчивому самовоспроизводству, относительно обособленная от других групп, с представителями которых потенциально возможен генетический обмен.

- *Индивид (особь)* — единичный представитель популяции.

- *Генотип* — наследственная информация, закодированная в хромосомах, которая вместе с факторами внешней среды определяет фенотип организма. Генотип не всегда соответствует одному и тому же фенотипу. Некоторые гены проявляются в фенотипе только в определенных условиях.

- *Фенотип* — совокупность характеристик, присущих индивиду на определённой стадии развития. Фенотип формируется на основе генотипа, опосредованного рядом факторов внешней среды.

ВВЕДЕНИЕ

Генетические алгоритмы являются одним из современных направлений в развитии интеллектуальных систем. Они успешно применяются во многих областях науки для решения разнообразных задач поиска и оптимизации. Примерами таких задач являются разнообразные задачи на графах, составление расписаний, построение нейронных сетей и прочих искомых объектов.

Одной из разновидностей генетических алгоритмов является генетическое программирование — автоматическое создание программ, закодированных подходящим для генетического алгоритма способом. Одним из таких представлений являются автоматные программы. Объединение конечных автоматов и генетического программирования позволяет получать эффективные решения нетривиальных задач.

Важной составляющей генетического алгоритма является функция приспособленности. В работах [1] [2] [3] [4], посвященных автоматической генерации конечных автоматов, эта функция задавалась фиксированным алгоритмом тестирования особи. Однако способ тестирования не всегда является очевидным. Для задач, в которых тестирование особи производится путем симуляции ее взаимодействия с внешней средой, неочевидными могут быть параметры внешней среды. В данной работе рассматривается класс задач, в которых поведение внешней среды, как и выводимой особи, может быть представлено с помощью конечного автомата. Целью данной работы является разработка нового метода генетического программирования, который для данного класса задач позволяет:

- в определенных пределах автоматически изменять алгоритм вычисления функции приспособленности;
- обеспечить более стабильную и эффективную работу генетического

алгоритма;

В работе решаются следующие задачи:

- описание метода автоматического изменения функции приспособленности за счет изменения конечного автомата, описывающего поведение внешней среды;
- построение модифицированного генетического алгоритма на основе описанного метода.

В главе 1 приводятся основные понятия и характеристики генетических алгоритмов, выполняется обзор существующих методов генерации конечных автоматов, основное внимание уделяется способу построения функции приспособленности. На основе анализа недостатков существующих решений осуществляется постановка задач данной работы.

В главе 2 предлагается новый метод, позволяющий автоматически изменять алгоритм вычисления функции приспособленности за счет изменения конечного автомата, управляющего поведением внешней среды. Строится модифицированный генетический алгоритм, работающий на основе описанного метода.

В главе 3 проводится экспериментальное исследование полученного метода на примере задачи об «Итерированной дилемме узника». Производится сравнительный анализ экспериментальных данных для предложенного метода и существующих методов. На основе сравнения делается вывод о пригодности метода для эффективного решения рассматриваемого класса задач.

В заключение работы перечисляются достигнутые результаты и приводится список открытых вопросов.

Работа содержит 52 страниц и состоит из списка терминов, введения, трех глав и заключения. Список литературы содержит 37 наименований. Работа иллюстрирована 18 рисунками и содержит 2 таблиц.

ГЛАВА 1. ОБЗОР

В данной главе приводится краткое описание генетических алгоритмов и способ их применения для генерации конечных автоматов. Рассматриваются различные подходы к построению функции приспособленности, а также методы улучшения результатов работы генетического алгоритма путем ее модификации. Производится анализ рассмотренных решений и их общих недостатков. На основе анализа формулируются задачи работы.

1.1. Генетические алгоритмы

В этом разделе кратко излагаются принципы работы генетических алгоритмов.

1.1.1. Эволюционные вычисления

Эволюционные вычисления — это методы оптимизации, базирующиеся на эволюции популяции особей, в процессе которой достигается максимальное значение целевой функции. Этот подход применим для задач оптимизации плохо определенных функций. Основной идеей эволюционных вычислений является использование принципа естественного отбора, заключающегося в том, что наиболее приспособленные особи дают потомство, формирующее следующее поколение. В среднем, следующее поколение является более приспособленным к внешней среде, чем предыдущее.

Сегодня эволюционные вычисления применяются в различных сферах науки: начиная с дизайна антенн для спутников и заканчивая расшифровкой генома человека.

1.1.2. Общая схема эволюционных алгоритмов

В эволюционных вычислениях [5] принято выделять подкласс эволюционных алгоритмов. При использовании эволюционных алгоритмов поиск оптимума ведется в пространстве гипотез X. Целевая функция $f: X \to R$ называется *функцией приспособленности* или *фитнесс-функцией*. Задача эволюционного алгоритма — максимизация функции приспособленности.

Особью эволюционного алгоритма принято называть некоторую гипотезу $i \in X$. Поколением или популяцией особей называют множество гипотез P_t, где t — возраст популяции, а мощность множества гипотез $|P_t|$ — размер популяции. Возраст особи $i \in P_t$ — число популяций $P_j : i \in P_t, j \le t$. Значение целевой функции для особи $i \in P_t$ будем обозначать как $f_t(i)$. Закодированный генотип особи принято называть хромосомой.

Обобщенная схема эволюционного алгоритма может быть представлена на рис. 1.

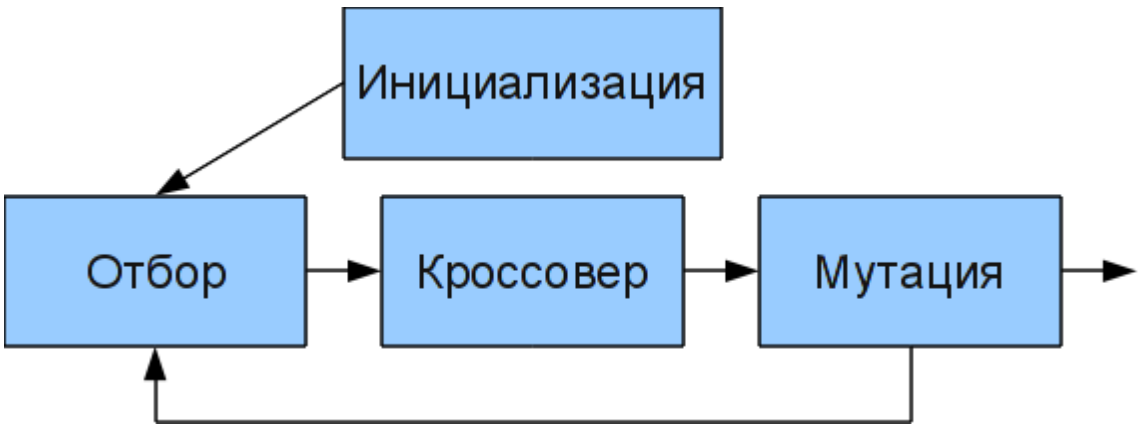

Рис. 1: Общая схема эволюционного алгоритма

Эта схема работает следующим образом:

- генерируется случайное множество хромосом;
- пока не выполняется критерий остановки:

- отбираются хромосомы, в наибольшей степени удовлетворяющие условиям задачи — хромосомы, имеющие большее значение функции f ;

- отобранные хромосомы скрещиваются, порождая следующее поколение;

- к некоторым особям потомства применяется мутация.

Математическое обоснование того, почему эволюционный алгоритм находит оптимальное решение, не приведено. Для наиболее простых моделей эволюционных алгоритмов сходимость может быть доказана с помощью теоремы о схемах [5]. В работе [6] показано, что в рамках некоторых условий использование скрещивания позволяет найти оптимум за конечное время, что неверно без его использования.

1.1.3. Классический генетический алгоритм

Генетический алгоритм был предложен Джоном Холландом в 1975 году в работе [5] и является первым из эволюционных алгоритмов. Особи в генетическом алгоритме представляются в виде строк фиксированной длины (обычно используются битовые строки). Рассмотрим работу генетического алгоритма более детально.

- **Инициализация.** Создается случайная начальная популяция. Способ создания случайной строки может быть специфичным для задачи, однако обычно применяется случайная генерация строк с равномерным распределением над каждым символом строки.

- **Отбор.** Существует несколько стратегий при выборе особей для проведения генетических операторов и при формировании нового

поколения. При *отбор по рулетке* вероятность выбора особи $i \in P_t$ прямо пропорциональна значению $f_t(i)$. При *пропорциональном отборе* находят значение $f' = \sum_{i \in P_t} f_t(i) / |P_t|$. Затем для каждой особи вычисляется отношение $\lfloor f_t(i) / f' \rfloor$, которое показывает сколько раз должна быть отобрана данная особь. Также в число часто используемых методов отбора входят *турнирный отбор*, *отбор усечением* и т. д.

- **Кроссовер.** На строках фиксированной длины скрещивание реализуют следующим образом: часть символов копируется из одной строки, остальные символы копируются из другой строки. Будем говорить, что результатом кроссовера двух строк $\{a_1 \dots a_n\}$ и $\{b_1 \dots b_n\}$ с маской $\{m_1 \dots m_n\}$, где $m_i \in \{0,1\}$, является строка $\{c_1 \dots c_n\}$, для которой

$$c_i = \begin{cases} a_i, & m_i = 0; \\ b_i, & m_i = 1. \end{cases}$$

  Таким образом, методы кроссовера отличаются методом создания маски. Наиболее распространены методы *одноточечного кроссовера*, *двухточечного кроссовера* и *однородного кроссовера*. Более сложные операторы кроссовера можно рассмотреть на примере скрещивания строк вещественных чисел — это «Flat crossover» [7], ``Simple crossover'' [8],[9], ``Arithmetical crossover'' [9], ``BLX- $\alpha$ crossover'' [10], ``Linear crossover'' [8], ``Discrete crossover'' [11], ``Extended line crossover'' [11] и пр.

- **Мутация.** Наиболее распространен следующий метод мутации — выбирается случайная позиция в строке и символ в этой позиции заменяется на случайный.

1.2. Генетическое программирование на автоматах

Генетическое программирование [12] является эволюционным алгоритмом, особями которого являются компьютерные программы. За последнее десятилетие в связи с резким увеличением мощности компьютеров возрос интерес к генетическому программированию как к средству автоматизации разработки ПО. В настоящее время с помощью генетического программирования получен ряд результатов, превосходящих аналогичные результаты людей, например создание сортирующей сети, квантовый алгоритм для задачи Гровера и т. д.

Ключевой идеей генетического программирования является представление хромосомы на достаточно высоком уровне абстракции, позволяющем учитывать специфику и структуру компьютерных программ [13], [14], [15], [1], [16], [17], [18].

1.2.1. Модель конечного автомата

Конечным детерминированным автоматом A типа Мили называется совокупность шести объектов $\{S, S_0, X, Z, \delta, \gamma\}$, которые имеют следующий смысл:

- S — конечное множество состояний автомата;
- $S_0 \in S$ — начальное состояние;
- X — конечное множество входных воздействий;
- Z — конечное множество выходных воздействий;
- $\delta : S \times X \to S$ — функция переходов автомата;

- $\lambda : S \times X \to Z$ — функция воздействий автомата,

где элементы множеств S, X и Z связаны в абстрактном времени $T = \{0,1,2,...\}$ следующими отношениями:

$$\begin{cases} s(t+1) = \delta(s(t), x(t)); \\ z(t) = \lambda(s(t), x(t)). \end{cases}$$

В рамках парадигмы автоматного программирования [19] конечный автомат Мили может быть обобщен за счет понятия предикатов — отображений вычислительных состояний в логические переменные. Заметим, что они не могут быть вынесены в состояния автомата, так как зависят от текущей ситуации. Переход будет осуществляться в зависимости не только от входных воздействий и текущего состояния, но и от значений предикатов. Предикаты также называются входными переменными. Также имеет смысл использовать несколько выходных воздействий при переходе, причем в таком случае необходимо задание порядка выполнения выходных воздействий. Таким образом, если множество предикатов обозначить за P, то указанные функции примут следующий вид:

$$\begin{cases} \delta : S \times X \times 2^P \to S; \\ \lambda : S \times X \times 2^P \to Z', \text{ где } Z' \in Z^*. \end{cases}$$

1.2.2. Кодирование конечных автоматов

В работах [20], [21] используется следующий подход к кодированию автоматов.

- Фиксируется число состояний автомата.

- Логика работы автомата представляется в виде таблицы, в которой для каждого состояния и каждого входного воздействия записывается пара

(новое состояние, воздействие).

- Полученная таблица записывается в виде строки. Поясним на примере, как таблица преобразуется в строку. На рис. 2 изображен граф переходов автомата, который может быть представлен в виде табл. 1.

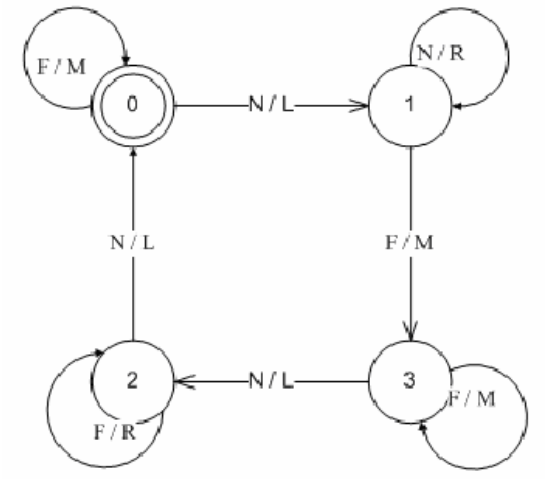

Рис. 2: Пример управляющего автомата

Таблица 1: Таблица переходов управляющего автомата

Состояние	Вход	Новое состояние	Действие
0	N	1	L
0	F	0	M
1	N	1	R
1	F	3	M
2	N	0	L
2	F	2	R
3	N	2	L
3	F	3	M

Теперь можно записать элементы этой таблицы в строку, выписав двойки (новое состояние, воздействие), соответствующие переходам. Для этого переходы упорядочиваются в порядке возрастания начального состояния, а в случае равенства — по номеру входного воздействия. Далее возможно применение одного из двух подходов: битовые строки и строки над числовым

алфавитом.

Кроме того, возможно хранение начального состояния в строке. Однако, анализ, проведенный в работе [20], показывает, что такой вариант не дает выигрыша по сравнению с зафиксированным начальным состоянием.

Для задачи построения автомата, распознающего регулярный язык, кроме того, необходимо ввести для всех состояний бит, указывающий на то, является ли это состояние допускающим.

Все таблицы, соответствующие состояниям одного автомата, имеют одинаковую размерность, так как число входных воздействий и воздействий объекта управления задано по условию задачи. Что касается управляющих состояний, то их число также должно быть известно.

Известен подход, при котором число управляющих состояний задается перед началом оптимизации и далее не изменяется. При таком подходе число состояний можно задать исходя из априорных представлений о сложности задачи, причем задать с некоторым “запасом”: в процессе оптимизации лишние состояния станут недостижимыми, и их можно будет автоматически исключить. Однако неоправданно большое число состояний негативно влияет на скорость эволюции.

В этом смысле более эффективным является постепенное наращивание числа управляющих состояний в процессе оптимизации. Этот вариант также несложно реализуется в рамках используемого подхода к представлению конечных автоматов в виде особей.

1.2.3. Эффективность генерации конечных автоматов

Вывод автоматов с помощью генетических алгоритмов может иметь

низкую эффективность в силу следующих причин.

- Автомат, соответствующий сгенерированной строке, может иметь недостижимые состояния. Информация, хранящаяся для этих состояний, является генетическим мусором.

- Представление не является естественным — двум разным строкам может соответствовать один и тот же автомат (одному фенотипу может соответствовать два и более генотипов). Это приводит к тому, что изоморфные автоматы соревнуются друг с другом, замедляя работу генетического алгоритма.

Одним из возможных подходов к решению данных проблем может являться применение эвристического оператора переупорядочивания. В качестве такого оператора на полных таблицах переходов может выступать метод *Move To Front (MTF)* [22]. Он заключается в перенумеровывании вершин автомата для приведения всех изоморфных автоматов к каноническому виду. Для этого из начальной вершины запускается обход в ширину. После этого вершины нумеруются в порядке обхода. Заметим, что недостижимые вершины автоматически помещаются в конец таблицы, соответствующей автомату. Пример такого преобразования приведен на рис. 3.

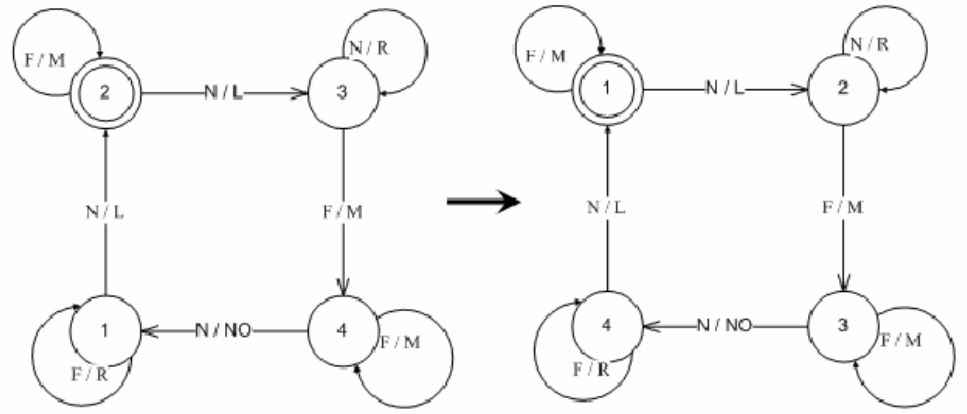

Рис. 3: Пример применения эвристики Move To Front

1.3. Способы построения функции приспособленности

Функция приспособленности является основополагающим компонентом генетического алгоритма. От ее свойств зависит то, насколько достоверно она позволяет определять степень пригодности определенного конечного автомата в качестве решения задачи, и насколько точно она позволяет сравнивать пригодность различных особей из популяции. В данном разделе рассматриваются различные способы построения функции приспособленности, применимые для конечных автоматов.

1.3.1. На основе симуляции внешней среды

Вычисление функции приспособленности на основе симуляции внешней среды получило наибольшее распространение благодаря своей очевидности и простоте. В некоторых ситуациях этот подход также представляется единственным возможным.

Пример использования этого подхода можно найти в статье [1] — в этой работе предлагается метод построения системы управления танком для популярной компьютерной игры Robocode. Описываемый метод позволяет на основе использования генетических алгоритмов строить достаточно простые системы управления танком. Функция приспособленности отражает результат

поединков против выбранного соперника. При этом, так как результат поединка существенно зависит от начального положения танков, то проводится не один поединок, а несколько. Если обозначить число очков, набранных построенным танком в бою, состоящим из 20 раундов, как s, а число очков, набранных его соперником, как s_e, то функцию приспособленности можно будет записать как $f = 100 * s/(s + s_e)$.

Другим примером является статья [2], в которой описывается совместное применение генетического программирования, конечных автоматов и искусственных нейронных сетей для построения системы управления беспилотным летательным аппаратом. Система управления строится как совокупность искусственной нейронной сети и конечного автомата. Нейронная сеть преобразует входные вещественные переменные в логические, которые подаются на вход конечному автомату. Он, в свою очередь, вырабатывает выходные воздействия. Для оптимизации этой модели используется генетическое программирование. Цель соревнований между двумя командами беспилотных летательных аппаратов состоит в том, чтобы один из летательных аппаратов команды переместился на максимальное расстояние от линии старта. Функция приспособленности особи вычисляется в ходе соревнований команды, аппараты которой управляются описываемыми особью системами управления беспилотным летательным аппаратом, с некоторыми командами, управляемыми системами управления, реализующими выбранную стратегию поведения. В качестве таких систем в настоящей работе используются системы, реализующие «агрессивную» и «простую» стратегию. Результатом вычисления функции приспособленности является сумма результатов команды, аппараты которой управляются описываемыми особью системами управления беспилотным летательным аппаратом, во всех соревнованиях, к которой прибавлено число побед, деленное на число соревнований, увеличенное на единицу.

1.3.2. На основе тестов

Метод построения конечных автоматов на основе тестов был описан в работе [3]. На начальном этапе проектирования программы выделяются входные переменные $\{x1, x2, ...\}$ и выходные воздействия $\{z1, z2, ...\}$. В качестве тестов для управляющего конечного автоматов рассматриваются пары последовательностей — $Input[i]$ и $Output[i]$. Первая описывает входные переменные, поступающие на вход автомату, а вторая — выходные воздействия, которые должен вырабатывать автомат при обработке этих событий. Для таких тестов справедливо свойство, которое можно сформулировать следующим образом — «префиксы тестов являются тестами» — если из входной последовательности событий удалить часть событий, находящихся в ее конце, то результат обработки автоматом этой последовательности будет префиксом исходной выходной последовательности. Поэтому в набор тестов также предлагается включать все префиксы теста.

Функция приспособленности основана на редакционном расстоянии. Для ее вычисления выполняются следующие действия: на вход автомату подается каждая из последовательностей $Input[i]$. Обозначим последовательность выходных воздействий, которую сгенерировал автомат на входе $Input[i]$ как $Output[i]$. После этого вычисляется функция:

$$FF_1 = \frac{\sum_{i=1}^{n} \left(1 - \frac{ED(Output[i], Answer[i])}{max(|Output[i]|, |Answer[i]|)}\right)}{n}.$$

Так как функция приспособленности должна зависеть не только от того, насколько «хорошо» автомат работает на тестах, но и от числа переходов, которые он содержит, ее предлагается вычислять по формуле:

$$FF_2 = \begin{cases} 10 * FF_1 + 0.01 * (100 - cnt), & FF_1 < 1 \\ 20 + 0.01 * (100 - cnt), & FF_1 = 1 \end{cases},$$

где *cnt* — число переходов в автомате. Эта функция приспособленности устроена таким образом, что при одинаковом значении функции FF1, отражающей «прохождение» тестов автоматом, преимущество имеет автомат, содержащий меньше переходов. Кроме этого, автомат, который «идеально» проходит все тесты, оценивается выше, чем автомат, проходящий тесты не идеально.

1.3.3. На основе верификации

В работе [4] описывается метод совместного применения генетического программирования и верификации моделей для построения автоматов управления системами со сложным поведением. Для описания требований к автоматным программам применяется язык *LTL*, в котором время линейно и дискретно. Синтаксис *LTL* включает в себя пропозициональные переменные *Prop*, булевы связки (¬, *Λ*, *V*) и темпоральные операторы. Последние применяются для составления утверждений о событиях в будущем. Используются следующие темпоральные операторы:

- **X** (ne**X**t) — *«Xp»* — в следующий момент выполнено *p*;
- **F** (in the **F**uture) — *«Fp»* — в некоторый момент в будущем будет выполнено *p*;
- **G** (**G**lobally in the future) — *«Gp»* — всегда в будущем выполняется *p*;
- **U** (**U**ntil) — *«pUq»* — существует состояние, в котором выполнено *q* и до него во всех предыдущих выполняется *p*;
- **R** (**R**elease) — *«pRq»* — либо во всех состояниях выполняется *q*, либо существует состояние, в котором выполняется p, а во всех предыдущих

состояниях выполнено q.

Множество *LTL*-формул таково:

- пропозициональные переменные *Prop*;
- *True*, *False*;
- если φ и ψ — формулы, то:
  - $\neg\varphi$, $\varphi \Lambda \psi$, $\varphi V \psi$ — формулы;
  - $X\varphi$, $F\varphi$, $G\varphi$, $\varphi U \psi$, $\varphi R \psi$ — формулы.

Так как описываемый метод предлагается использовать совместно с методом построения конечных автоматов на основе тестов, за основу функции приспособленности берется функция FF_2, описанная в разделе 1.3.2. Для каждой особи из текущей популяции при вычислении функции приспособленности верифицируется модель (конечный автомат) особи и результат учитывается для вычисления приспособленности. Приведем выражение для конечной функции приспособленности:

$$FF_3 = FF_2 + F * \frac{n_1}{n_2} \quad .$$

В указанной формуле как F обозначена «стоимость» выполнения всех формул (при проведении вычислительных экспериментов было выбрано значение $F = 10$), как n_1 — число *LTL*-формул, которые выполняются для автомата, а как n_2 — общее число *LTL*-формул, специфицирующих поведение системы со сложным поведением.

1.4. Влияние функции приспособленности на эффективность работы генетического алгоритма

В работе [23] проводится анализ влияния функции приспособленности на эффективность работы генетического алгоритма, и предлагается новый метод улучшения поиска оптимального результата, основанный на принципе модификации оценочной функции алгоритма.

В генетических алгоритмах выборка особей для скрещивания и мутации производится непосредственно на основе функции приспособленности для каждой особи. Поскольку генетические алгоритмы являются фактически способом решения оптимизационных задач, то традиционно в функции приспособленности участвуют только оптимизируемые параметры, значения которых вычисляются в ходе тестирования особей очередного поколения.

Однако зачастую особи обладают большим числом параметров, по которым их можно охарактеризовать, нежели те несколько, по которым их оценивает функция приспособленности. При этом особи, имеющие одинаковое значение приспособленности, могут оказаться сильно различающимися в остальных характеристиках, которыми можно их описывать. Одно значение функции приспособленности не дает представления о поведении особи в среде, тогда как совокупность характеристик особи может позволить понять принципы взаимодействия ее со средой. При этом очевидно, что поведение особи и определяет, насколько приспособлена или нет особь к внешней среде.

Предложенный метод заключается в выделении дополнительных характеристик особи и выявлении связи между их значениями и значениями оценочной функции. Выделение дополнительных характеристик особи производится для каждой задачи отдельно, поскольку универсальных характеристик, присутствующих у особей во всех генетических алгоритмах, не существует. Однако в случае, если решения генетического алгоритма ищутся в

виде конечного автомата, можно гарантированно выделить набор дополнительных характеристик.

Пусть $X=\{x_1, x_2, \ldots, x_m\}$ — множество входных воздействий, $Z=\{z_1, z_2, \ldots, z_n\}$ — множество выходных воздействий, а $C=\{c_1, c_2, \ldots, c_k\}$ — свойства взаимодействия особи со средой, не являющиеся входными или выходными воздействиями. Можно вычислить, сколько раз выполняется свойство или совершается любое из воздействий в процессе тестирования особи в среде. Более того, можно вычислить, сколько раз выполняются более сложные условия. Функция приспособленности, в свою очередь, является функцией от некоторых таких вычисленных значений.

Пусть S — функция, считающая число совершенных воздействий или число сколько раз выполнилось свойство, например: $S(x_1)$, $S(x_k/z_l)$, $S(c)$ и т. п. Также вводится функция $P_x(y, z, \ldots)=S(y)+S(z)+\ldots$, которая вычисляется при очередном воздействии x или выполнении свойства c. Далее строится многообразие введенных функций S, варьируя подставляемые переменные. Оно включает как функции от простых параметров, так и от непротиворечивых сочетаний входных и выходных воздействий. После этапа построения имеется два набора функций $S=\{S_{x_1}(p), S_{x_2}(p), \ldots\}$ и $P=\{P_{x_1}(p), P_{x_1, x_2}(p), \ldots\}$, которые объединяются в набор характеристик, обозначаемый как F.

Связь между оценочными характеристиками F и основной функцией приспособленности F_0 устанавливается в ходе работы итеративного алгоритма, заключающегося в тестировании генетического алгоритма с различными модификациями функции приспособленности на небольшом числе итераций. Итоговая функция приспособленности может быть записана как $F_m=F_0+F_{x_1}+\ldots+F_{x_k}$, где F_x — подстановка характеристики $x\in F$ или $1/x$.

Полученная оптимизированная функция приспособленности имеет преимущество по сравнению с оригинальной. При ее использовании генетический алгоритм обладает ускоренной сходимостью и показывает лучший конечный результат.

1.5. Постановка задачи

В разделе 1.3 были рассмотрены разнообразные способы построения функции приспособленности. Все они подразумевают создание фиксированной функции приспособленности до начала исполнения генетического алгоритма. При использовании симуляции внешней среды, приспособленность особи вычисляется по результатам симуляции работы автоматизированного объекта при одном или нескольких наборах параметров внешней среды. Такой подход не всегда эффективен, так как пространство возможных параметров внешней среды несоизмеримо больше его подмножества, на котором производится тестирование особей.

Однако, как показано в разделе 1.4, эффективность работы генетического алгоритма напрямую зависит от того, насколько разностороннюю оценку особи дает функция приспособленности. Целью данной работы является создание нового метода, позволяющего охватить большую часть пространства возможных параметров внешней среды, при этом сохраняя приемлемую скорость работы генетического алгоритма. В рамках этого метода производится автоматическое изменение функции приспособленности путем модификации набора параметров внешней среды, на котором производится тестирование особей.

ГЛАВА 2. ТЕОРЕТИЧЕСКИЕ ИССЛЕДОВАНИЯ

В данной главе описывается предлагаемый метод, позволяющий автоматически изменять алгоритм вычисления функции приспособленности в задачах, в которых возможно представление логики работы внешней среды в виде конечного автомата. Также производится построение модифицированного генетического алгоритма, основанного на данном методе.

2.1. Представление внешней среды в виде конечного автомата

При применении автоматов в программировании, как правило, используются не абстрактные модели автоматов, а модель автоматизированного объекта [24], объединяющая управляющий автомат и объект управления.

Важная черта устройства управления всех абстрактных машин – их конечность. Управляющий автомат не только имеет конечное число состояний, но, кроме того, реализуемые им функции переходов и выходов оперируют исключительно конечными множествами. Именно это свойство позволяет описывать логику поведения машины явно: в виде таблицы или графа переходов. Поэтому свойство конечности устройства управления необходимо сохранить при построении модели автоматизированного объекта. Более того, это свойство целесообразно усилить следующим неформальным требованием: число управляющих состояний, входных и выходных воздействий должно быть небольшим (обозримым). Управляющий автомат с тысячей состояний, безусловно, является конечным, однако, изобразить его граф переходов практически невозможно, что сводит на нет преимущества явного выделения управляющих состояний.

Напротив, число состояний объекта управления может быть сколь угодно большим (при переходе от модели к программной реализации оно с

необходимостью станет конечным, однако, может остаться необозримым). В процессе работы управляющему автомату требуется получать информацию о состоянии объекта управления и изменять его. Однако в силу свойства конечности автомат не может напрямую считывать и записывать это состояние. Для этого и требуются операции объекта управления. Небольшое число запросов, возвращающих конечные значения, позволяет автомату получать информацию о состояниях объекта управления, которую он способен обработать. Небольшое число команд используется для «косвенного» изменения состояний объекта управления.

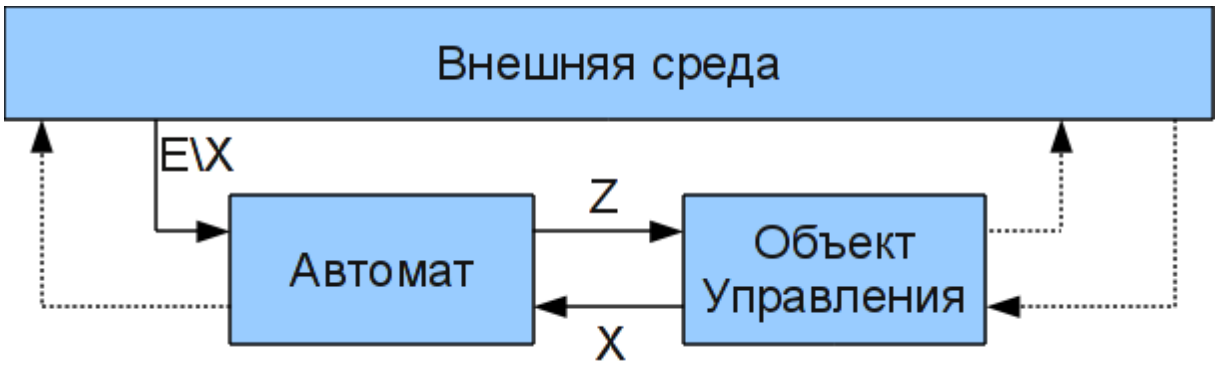

Рис. 4: Модель автоматизированного объекта

На рисунке 4 сплошными стрелками обозначены типичные для программных реализаций виды взаимодействия между автоматом, объектом управления и внешней средой. объектом управления и внешней средой. Автомат получает входные воздействия как со стороны среды, так и от объекта управления. В событийных системах часть или все компоненты входного воздействия со стороны среды могут быть событиями (множество событий обозначено на рисунке буквой E). Входное воздействие со стороны объекта управления формирует в модели обратную связь (от управляемого объекта к управляющему). Это воздействие может отсутствовать, тогда модель является разомкнутой – так в теории управления называются системы управления без обратной связи [25]. В противном случае модель называется замкнутой. Автомат, в свою очередь, воздействует на объект управления.

Пунктирными стрелками обозначены менее распространенные, хотя и возможные, варианты взаимодействия. Так, автомат может оказывать выходное воздействие и на внешнюю среду. Однако таких связей обычно можно избежать, включив все управляемые автоматом сущности в состав его объекта управления. Отметим, что в программировании, в общем случае, различие между объектом управления и внешней средой носит скорее концептуальный, а не формальный характер. Создавая модель системы со сложным поведением, разработчик производит ее декомпозицию на автоматизированные объекты, определяя тем самым объект управления каждого автомата. В целях минимизации связей между модулями программной системы целесообразно проводить декомпозицию таким образом, чтобы автомат оказывал выходные воздействия только на собственный объект управления. Кроме того, объект управления может взаимодействовать с внешней средой напрямую.

В рамках данной работы рассматривается ситуация, в которой поведение внешней среды автоматизированного объекта может быть представлено в виде конечного автомата. В этом случае внешняя среда сама может рассматриваться как автоматизированный объект, что порождает схему, изображенную на рис. 5.

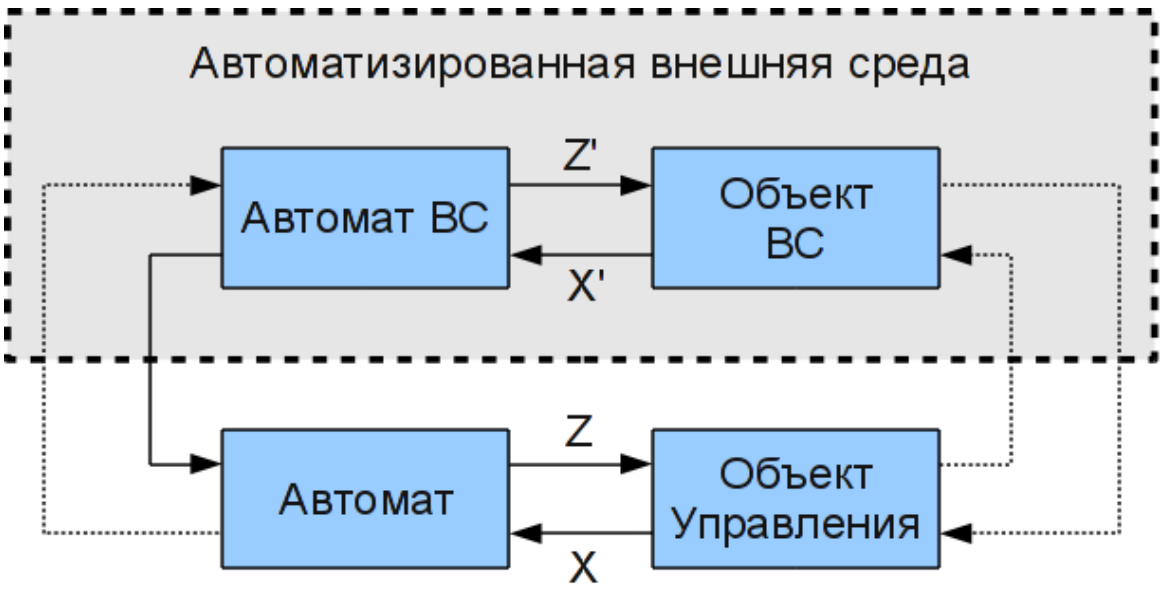

Рис. 5: Модель с автоматизированной внешней средой

Отметим, что полученная модель очень удобна в виду естественности взаимодействия ее компонент. Обмен информацией между автоматизированными объектом и внешней средой ведется через входные и выходные воздействия управляющих автоматов, а также через взаимодействие объектов управления.

Также можно выделить класс задач, в которых подразумевается наличие нескольких автоматных программ с одинаковыми наборами входных и выходных воздействий, играющих в некую игру. В таком случае, для каждого из игроков внешней средой будут являться его оппоненты, а входными воздействиями управляющего его автомата — выходные воздействия автоматов оппонентов, или их функция $f_z: Z^{N-1} \to X$, где N — число игроков.

2.2. Предлагаемый метод

Так как алгоритм работы внешней среды задается ее управляющим автоматом и программной реализацией объекта управления, представляется возможным изменение поведения окружающей среды путем модификации соответствующего конечного автомата.

Суть предлагаемого метода заключается в автоматическом подборе управляющего автомата внешней среды, обеспечивающее наилучшее тестирование целевого автоматизированного объекта. Так как целью тестирования является проверка на предмет соответствия заданным спецификациям, то наилучшим тестом можно считать тот, который выявляет наибольшее отклонение от данных спецификаций.

Пусть задана функция приспособленности $f_t(i, e)$ для особи $i \in P_t$ и управляющего автомата внешней среды $e \in P_{env}$. Тогда если известно максимальное возможное значение функции f_t для всех i и e

$f_{max}=max_{(i,e)}[f_t(i,e)]$, то уровень отклонения особи i от оптимального поведения можно оценить как $f_{inv}(i,e)=f_{max}-f_t(i,e)$. Тогда качество теста с автоматом e для всей популяции P_t вычисляется как среднее значение

$$f_{inv}(e)=\frac{\sum_{i\in P_t} f_{inv}(i,e)}{|P_t|} \quad .$$

Однако подбор одиночного управляющего автомата внешней среды не является эффективным способом всестороннего тестирования целевых особей. Поэтому предлагается хранить множество таких автоматов, которые будут образовывать популяцию особей внешней среды P_{env} . К этой популяции можно применить метод генетического программирования, используя функцию $f_{inv}(e)$ в качестве функции приспособленности. Также необходимо определить модифицированную функцию приспособленности для популяции P_t , вычисляемую как среднее значение $f_t(i)=\frac{\sum_{e\in P_{env}} f_t(i,e)}{|P_{env}|}$.

Если использовать генетический алгоритм для популяции P_{env} как составную часть основного алгоритма, то получится так называемый «мета-генетический» алгоритм [26]. В данном случае оптимальнее будет использовать модификацию классического алгоритма, в котором эволюция обеих популяций производится параллельно.

2.3. Модифицированный генетический алгоритм

Сформулируем этапы модифицированного генетического алгоритма, автоматически подбирающего управляющие автоматы внешней среды:

- **Инициализация.** Создаются две случайные начальные популяции целевых автоматов и управляющих автоматов внешней среды, с

соответствующими входными и выходными воздействиями. Размер популяций и параметры генерации случайных особей могут различаться.

- **Тестирование.** На этапе тестирования происходит вычисление функций приспособленности $f_t(i)$ и $f_{inv}(e)$ для всех особей $i \in P_t$ и $e \in P_{env}$. Для этого производится симуляция поведения каждого целевого автоматизированного объекта i в каждой автоматизированной внешней среде e, результатом которой является функция $f_t(i,e)$. Далее искомые значения функций приспособленности вычисляются по формулам, приведенным в разделе 2.2.

- **Отбор.** Выбор особей для проведения генетических операций и формирования нового поколения производится независимо для каждой из популяций, поэтому алгоритм отбора модификации не требует.

- **Кроссовер и мутация.** Способ проведения кроссоверов и мутаций также не отличается от классического алгоритма, однако можно сформулировать рекомендации по настройке его параметров. Как правило при применении оператора мутации задается ее «сила», определяющее насколько большая доля хромосомы будет подвергнута случайной модификации. Данный параметр, в совокупности с общим количеством производимых мутаций, определяет насколько быстро изменяется генотип популяции в ходе эволюционного процесса. Для обеспечения большей стабильности работы модифицированного генетического алгоритма предлагается устанавливать этот параметр для популяции управляющих автоматов внешней среды ниже, чем для популяции целевых автоматов.

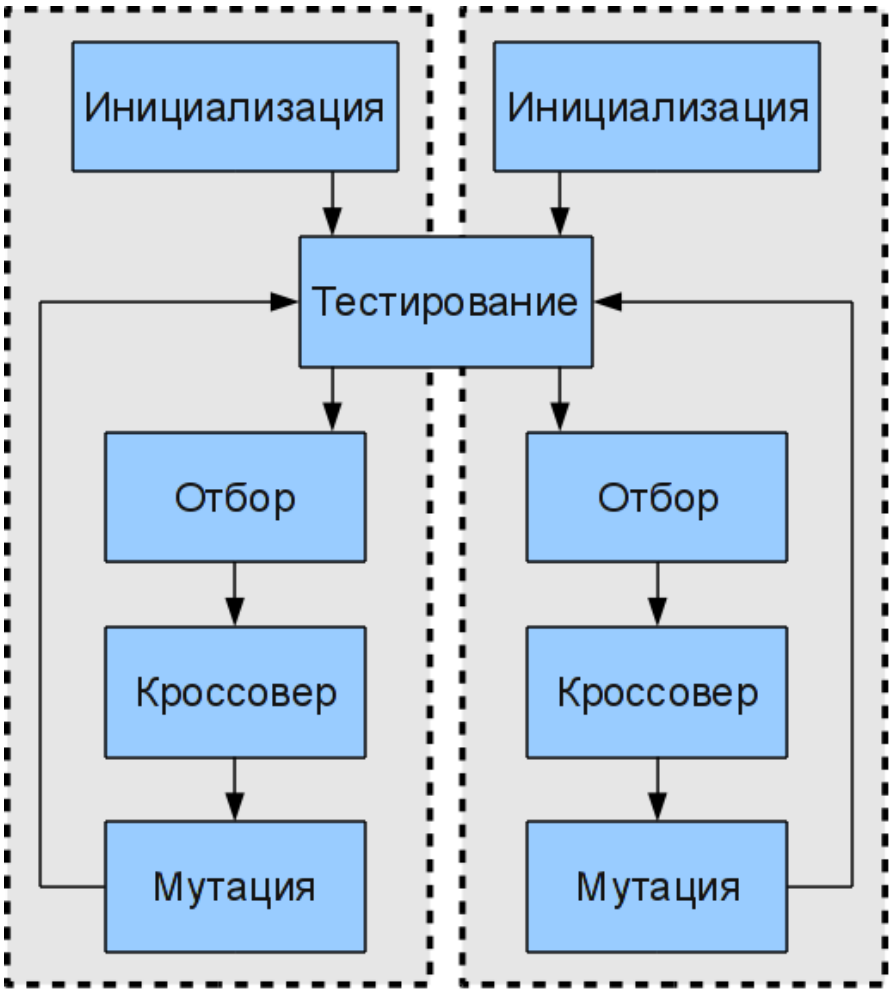

Рис. 6: Схема модифицированного генетического алгоритма

Принципиальная схема модифицированного генетического алгоритма изображена на рис. 6. На ней явно обозначены части алгоритма, выполняющиеся независимо для обеих популяций, и точка, в которой происходит их взаимодействие.

2.4. Оценка трудоемкости алгоритма

Проведем сравнительную оценку трудоемкости построенного и классического генетических алгоритмов. В подобных алгоритмах наибольшую вычислительную сложность как правило представляет задача симуляции окружающей среды. Предлагаемый метод подразумевает проведение на каждой итерации генетического алгоритма в $|P_{env}|$ большего числа симуляций, что

приведет к увеличению времени работы алгоритма примерно в то же число раз. Данное соотношение накладывает практическое ограничение на размер популяции управляющих автоматов окружающей среды.

Одним из способов уменьшения времени работы генетического алгоритма является использование распределенных вычислений. Как было показано в работе [27], они являются эффективным методом для быстрого получения конечных результатов. Так как основной эффект достигается за счет распределенного тестирования особей, применение распределенных вычислений к модифицированному генетическому алгоритму должно быть легко осуществимо на базе существующих разработок.

Также существуют возможные пути ускорения работы генетического алгоритма за счет тестирования неполного набора пар i и e, однако подробное изучение таких возможностей выходит за рамки данной работы и является отдельной темой для исследования.

ГЛАВА 3. ПРАКТИЧЕСКИЕ ИССЛЕДОВАНИЯ

В данной главе описывается практическая реализация модифицированного генетического алгоритма, описанного в разделе 2.3, а также проводится экспериментальное исследование метода на примере задачи об «Итерированной дилемме узника».

3.1. Инструментальное средство GAAP

Предложенный подход был реализован на основе инструментального средства GAAP (Genetic Algorithms for Automata Programming) [28] [29].

Программное средство GAAP предназначено для написания генетических алгоритмов и исследования их применительно к различным задачам (не обязательно к задачам генерации конечных автоматов). Данное программное средство было разработано автором совместно с Е. А. Мандриковым и применено в работах [23] [30] [31].

Программное средство разработано на языке *Java* с использованием автоматизированного средства сборки *Maven*[1] и техники *TDD*[2]. Программное средство обладает следующими достоинствами:

- программные интерфейсы (*API*[3]) просты для использования;
- программное средство может быть легко использовано в составе других приложений, написанных на языке *Java*;
- программное средство содержит реализацию различных эволюционных алгоритмов (в том числе реализацию островной модели) для решения

1 **Apache Maven** – система автоматической сборки проектов (http://maven.apache.org/)
2 **TDD (Test-Driven Development)** – разработка через тестирование
3 **API (Application Programming Interface)** – набор готовых классов, функций, структур и констант, предоставляемых приложением для использования во внешних программных продуктах.

исследуемых задач;

- содержит реализации различных представлений автоматов в виде хромосом;
- содержит реализации генетических операторов;
- содержит различные методы отбора особей;
- содержит набор компонентов для реализации распределённых вычислений функции приспособленности;
- содержит компоненты позволяющие наглядно продемонстрировать работу генетического алгоритма и работу автоматизированного объекта.

При выполнении данной работы в программное средство была добавлена возможность реализации генетических алгоритмов с несколькими взаимодействующими популяциями. Архитектура базовых модулей была переработана, чтобы добиться необходимой гибкости настройки. Основой для построения генетического алгоритма стал интерфейс *PopulationProcessor* – абстрактный обработчик популяции, который может изменять метаданные популяции или отдельных особей, а также замещать существующую популяцию новой. Диаграмма классов и интерфейсов, связанных с *PopulationProcessor* представлена на рис. 7.

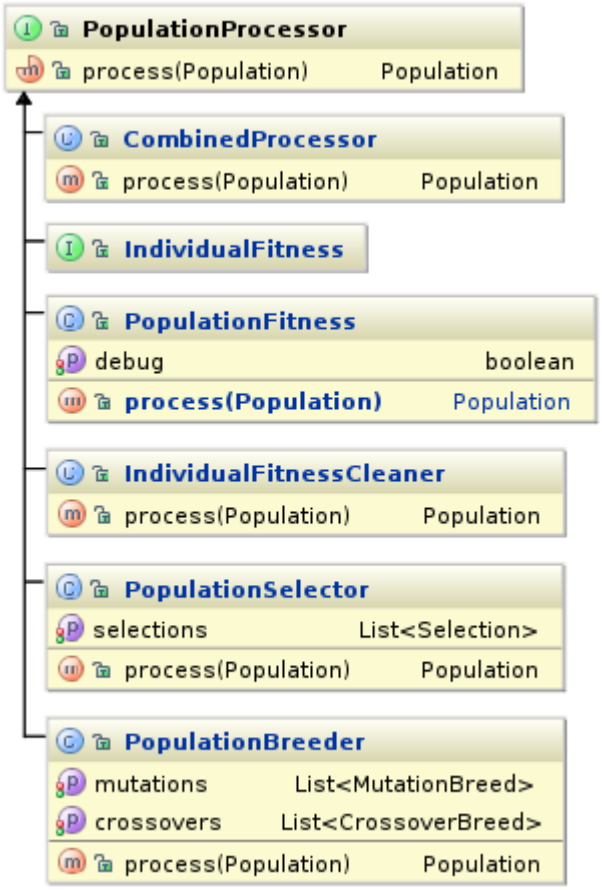

Рис. 7: Диаграмма классов и интерфейсов обработчика популяции

Здесь:

- *CombinedProcessor* – обработчик популяции, позволяющий представить несколько последовательно исполняющихся обработчиков в виде одного. Используется для улучшения декомпозиции составляющих генетического алгоритма;

- *IndividualFitness* – интерфейс обработчик а популяции, который производит подсчет функции приспособленности для всех особей, еще не имеющих ассоциированного значения приспособленности;

- *PopulationFitness* – обработчик, вычисляющий статистические значения функции приспособленности для всей популяции, такие как максимальное, минимальное и среднее значения;

- *PopulationFitnessCleaner* – обработчик, стирающий ассоциированное значение функции приспособленности для всех особей в популяции. Используется в ситуациях, когда необходим полный пересчет функции приспособленности;

- *PopulationSelector* – обработчик, отбирающий из расширенной популяции особи для дальнейшей работы генетического алгоритма;

- *PopulationBreeder* – .обработчик, вызывающий операции кроссовера и мутации. Результатом работы является новая расширенная популяция.

Таким образом, программная реализация генетического алгоритма заключается в задании набора обработчиков популяции и повторного применения их к начальной популяции, вплоть до достижения условия останова.

Генетический алгоритм с двумя популяциями был реализован за счет специального обработчика, позволяющего добиться синхронизации между двумя независимыми потоками исполнения. Две сущности данного обработчика добавляются в наборы обработчиков, непрерывно исполняющихся в параллельных потоках. При вызове одного из обработчиков текущий поток блокируется до тех пор, пока не произойдет вызов второго обработчика. После достижения этого условия, выполняется вложенный обработчик, ответственный за подсчет функции приспособленности.

3.2. Итерированная дилемма узника

В теории игр известна следующая некооперативная матричная игра с ненулевой суммой, обычно называемая «Дилемма узника» («Prisoner’s dilemma») [32]. Двое преступников пойманы и допрашиваются в отдельных

камерах. Срок тюремного заключения, который получит каждый из них, зависит как от его показаний, так и от показаний соучастника. В табл. 2 приведены сроки заключения в зависимости от решений, принятых игроками — их стратегий.

Таблица 2: Матрицы игры о "Дилемме узника"

	Сознаться	Отрицать
Сознаться	3; 3	0; 5
Отрицать	5; 0	1; 1

В данной игре предательство (стратегия «сознаться») строго доминирует над сотрудничеством (стратегией «отрицать»). Единственное равновесие в игре (по Нэшу) — признание обоих преступников, что приведет к ситуации (3; 3). При любой зафиксированной стратегии соучастника преступнику выгодно предать. Таким образом, действуя по отдельности рационально, игроки приходят к нерациональному решению (3; 3), хотя могли бы получить «всего» по году заключения (1; 1), выбрав стратегию «отрицать».

В книге «Эволюция кооперации» [33] Роберт Аксельрод исследовал расширение сценария «дилеммы узника», которое он назвал «итерированная дилемма узника». В ней участники делают выбор снова раз за разом и помнят предыдущие результаты. Аксельрод пригласил коллег со всего мира, чтобы разработать компьютерные стратегии, чтобы соревноваться в чемпионате. Программы, вошедшие в него, различались по алгоритмической сложности, начальной враждебности, способности к прощению и так далее.

Аксельрод открыл, что если игра повторялась долго среди множества игроков, каждый с разными стратегиями, «жадные» стратегии давали плохие результаты в долгосрочном периоде, тогда как более «альтруистические» стратегии работали лучше, с точки зрения собственного интереса. Он использовал это, чтобы показать возможный механизм эволюции

альтруистического поведения из механизмов, которые изначально чисто эгоистические, через естественный отбор.

Лучшей детерминистской стратегией оказалась «*Око за око*» («*Tit for Tat*»), которую разработал и выставил на чемпионат Анатолий Рапопорт. Она была простейшей из всех участвовавших программ, состояла всего из 4 строк кода на языке Бейсик. Стратегия проста: сотрудничать на первой итерации игры, после этого игрок делает то же самое, что делал оппонент на предыдущем шаге.

3.2.1. Применение генетического алгоритма

В статье [34] описывается эксперимент с применением генетического алгоритма для генерации стратегий управления узником. В качестве хромосомы использовалась битовая строка, кодирующая управляющую функцию $\{S, P, R, T\}^3 \to \{Cooperate, Defect\}$. На вход функции предоставляются исходы трех предыдущих раундов игры, которые могут быть следующими:

- $S(Sucker)$ - игрок отрицал, когда другой сознался;
- $P(Punishment)$ - оба игрока сознались;
- $R(Reward)$ - оба игрока отрицали;
- $T(Temptation)$ - игрок сознался, когда другой отрицал.

Функция приспособленности особей вычислялась путем турнира между особями популяции. Полученные в результате стратегии имели поведение, близкое к стратегии «*Око за око*» (около 95% ходов), достигая практически полного взаимодействия. В турнире многие особи набирали даже большее количество баллов, чем «*Око за око*», за счет эксплуатации особенностей своих

собратьев.

В статье [35] исследуется характер эволюционного процесса для разного числа запоминаемых предыдущих шагов. На рисунке 8 показан график роста функции приспособленности лучшей особи в популяции для разных длин истории.

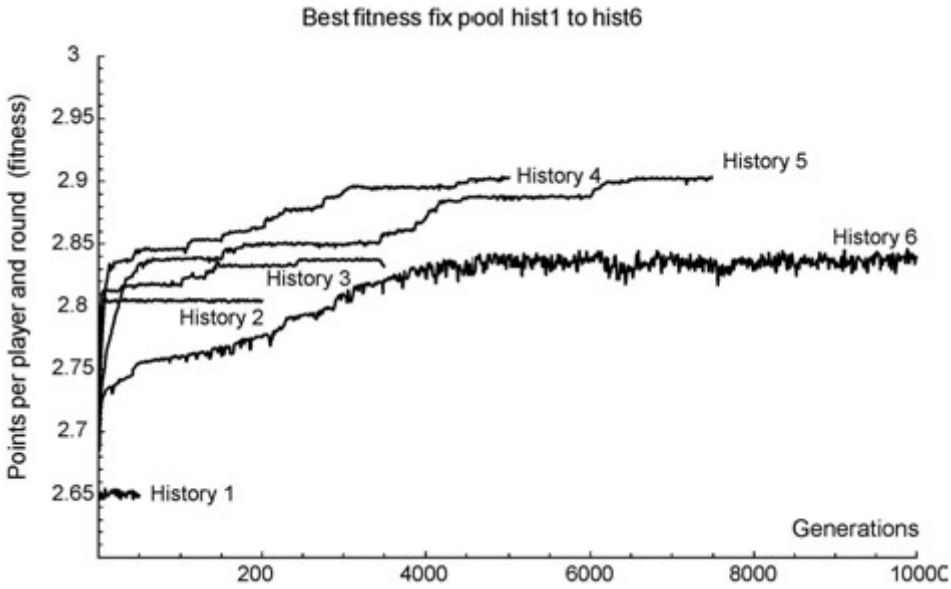

Рис. 8: График функции приспособленности лучшей особи в популяции

Проведя статистическое исследование получаемых результатов, авторами были подсчитаны средние значения функции приспособленности лучшей особи, а также среднеквадратичное отклонение. Как видно на рисунке 9, при наилучших средних результатах наблюдаются значительные колебания между отдельными запусками генетического алгоритма.

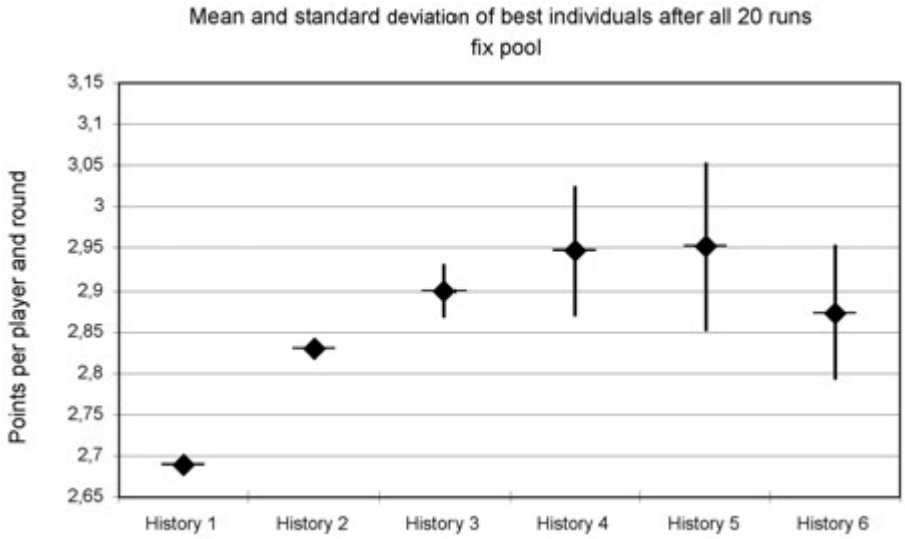

Рис. 9: Распределение лучших значений функции приспособленности

В работах [36], [37] приводятся исследования работы аналогичного генетического алгоритма для «итерированной дилеммы узника», использующего конечный автомат для описания стратегии поведения узника. Во всех описанных экспериментах эволюционный процесс показывает схожие свойства.

3.2.2. Эксперименты с двумя популяциями

Экспериментальное исследование генетического алгоритма для двух популяций конечных автоматов было проведено в двух вариациях. Общими настройками алгоритма были:

- размер популяции — 300;
- число состояний автомата — 5;
- коэффициент скрещиваний — 0.5;
- коэффициент мутаций — 0.5;

- коэффициент элитизма — 0.125.

Для оценки получаемого результата также подсчитывался выигрыш лучшей особи из популяции против стратегии «Око за око».

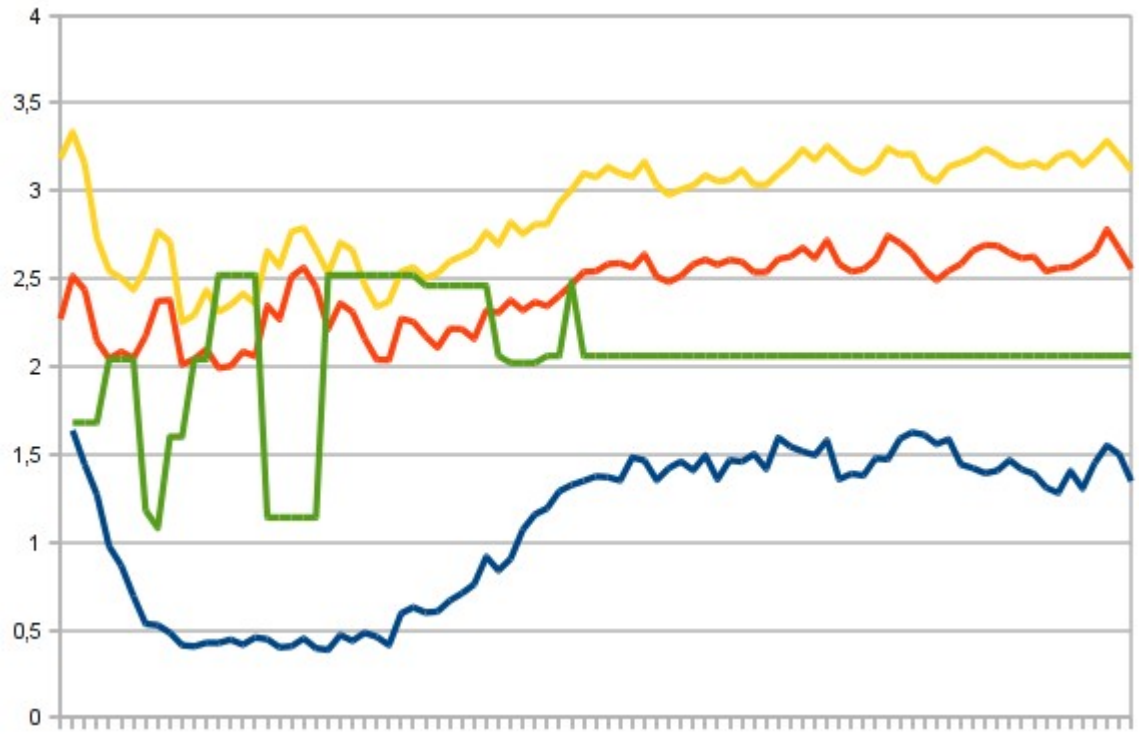

Рис. 10: Показания первой популяции с симметричной функцией приспособленности

В первом эксперименте функция приспособленности для обеих популяций вычислялась по одинаковому алгоритму, который далее будем называть «прямым». В этом случае приспособленность особи определяется как ее средний выигрыш при игре со всеми особями из противоположной популяции. Графики, отражающие ход эксперимента, приведены на рисунках 10 и 11. Три коррелирующие линии (синяя, желтая и красная) обозначают соответственно минимальное, максимальное и среднее значение функции приспособленности в популяции. Четвертая линия (зеленая) обозначает приспособленность лучшей особи из популяции при игра против стратегии «Око за око».

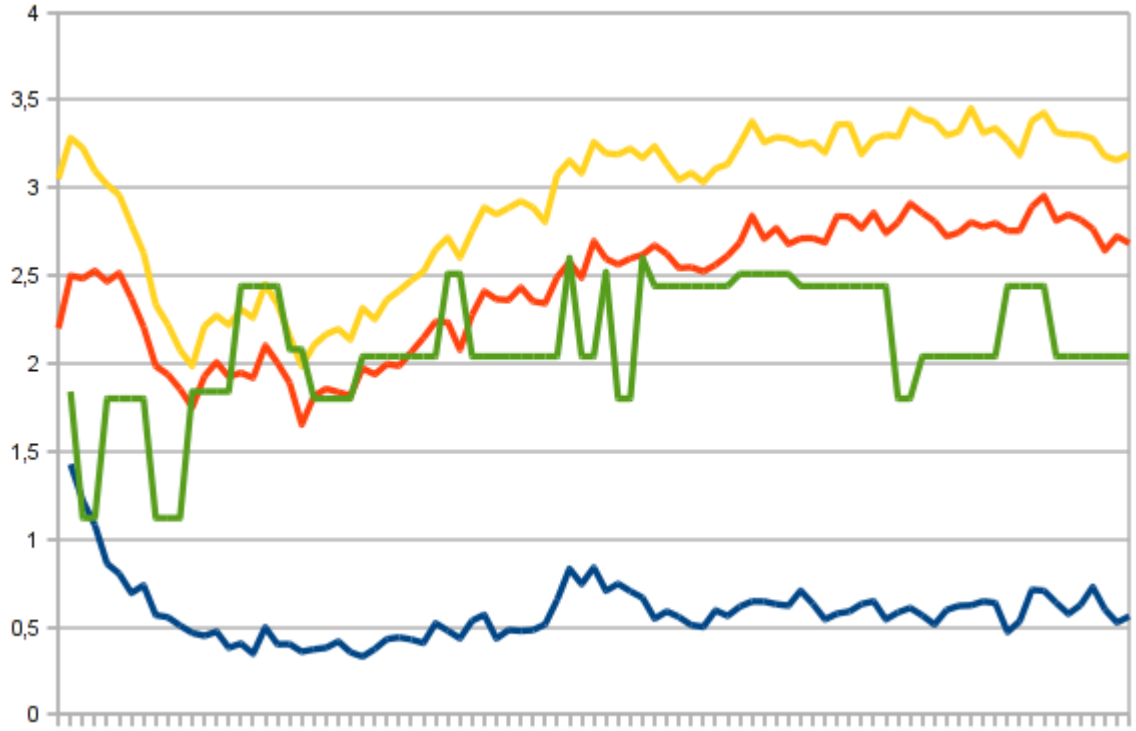

Рис. 11: Показания второй популяции с симметричной функцией приспособленности

Во втором эксперименте функция приспособленности одной из популяций была заменена на «обратную», которая вычисляется не как средний выигрыш игрока, а как средний проигрыш оппонента. Таким образом, одна из популяций берет на себя четко обозначенную роль контроля качества особей из противоположной популяции. По графикам 12 и 13, полученным в ходе эксперимента, можно судить о пользе такого подхода, так как получаемые решения лучше ведут себя в игре против неизвестной им стратегии «Око за око».

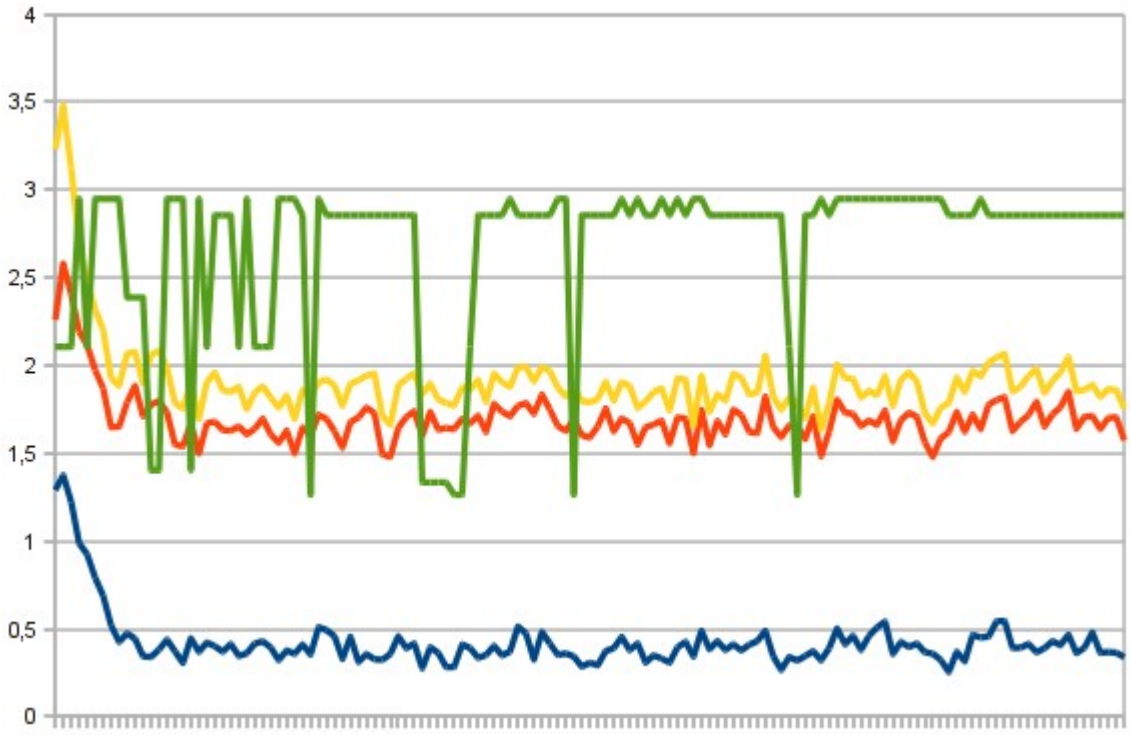

Рис. 12: Показания популяции с "прямой" функцией приспособленности

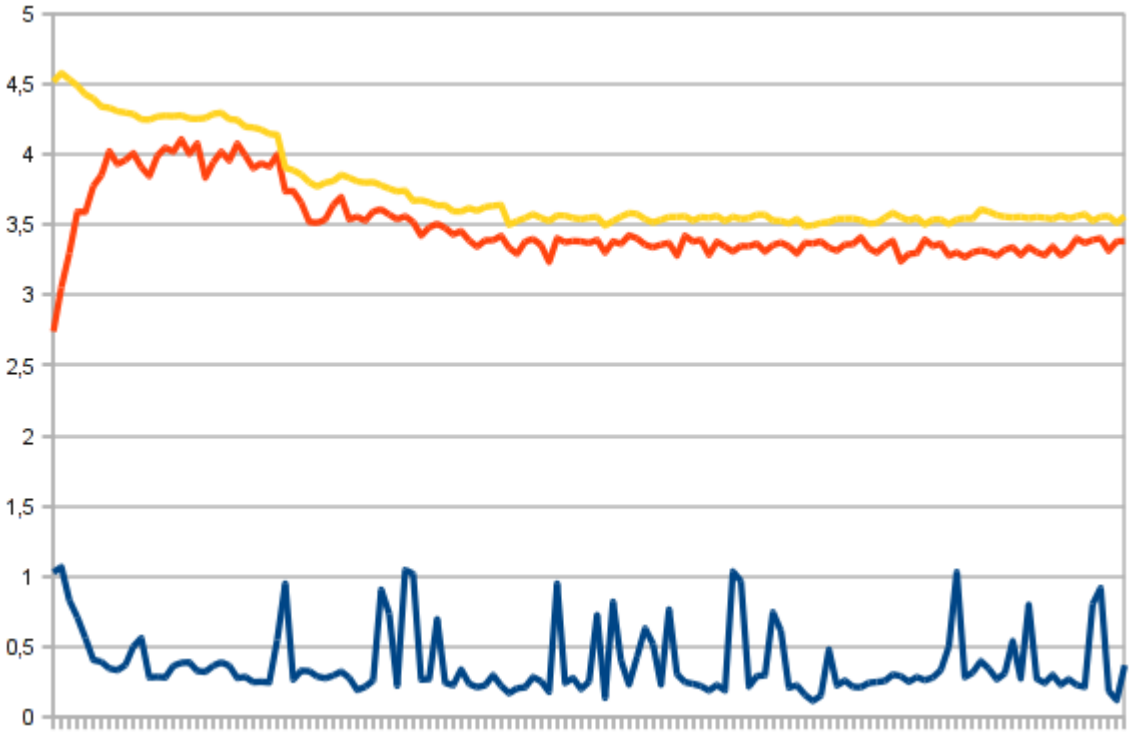

Рис. 13: Показания популяции с "обратной" функцией приспособленности

3.3. Задача о змейке

Рассмотрим простую задачу о змейке, состоящей из набора последовательных звеньев. Змейка передвигается по клетчатому полю в поисках яблок, выбирая одно из трех возможных направлений движения, как показано на рис. 14.

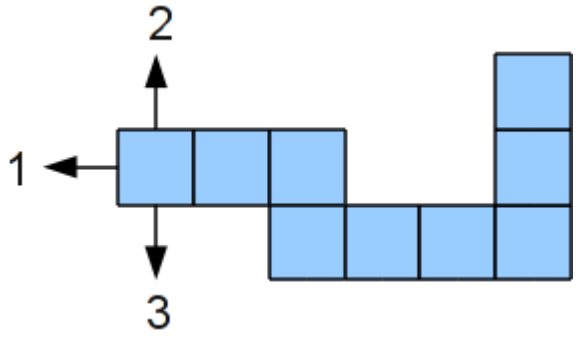

Рис. 14: Змейка и направления ее движения

Область видимости змейки (рис. 15) состоит из шести клеток, располагающихся относительно ее головы. Как выбор направления движения, так и область видимости задается относительно текущей ориентации головы змейки в целях упрощения управляющего автомата.

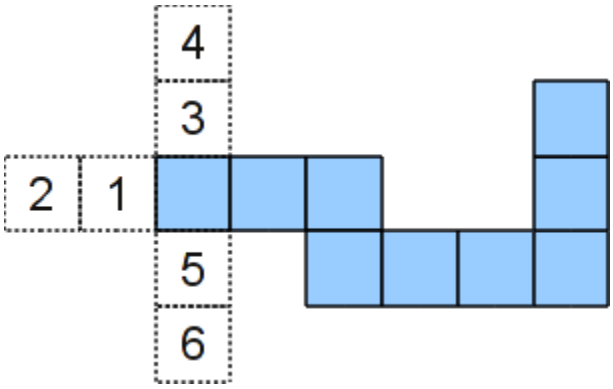

Рис. 15: Область видимости змейки

В каждый момент времени на поле располагается ровно одно яблоко. Его позиция определяется случайным образом так, чтобы оно оказалось в радиусе двух клеток от головы змейки по одному из возможных направлений, как показано на рис. 16.

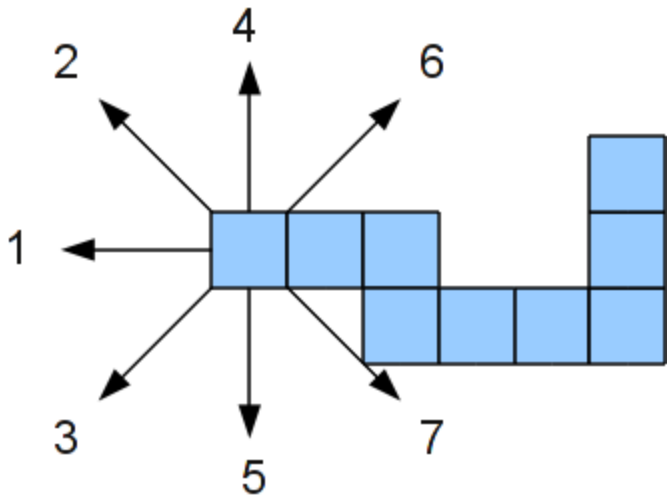

Рис. 16: Возможные направления для расстановки яблок

3.3.1. Случайное поведение внешней среды

В рамках данного эксперимента проводилась генерация управляющих автоматов для змейки. Основными настройками генетического алгоритма были:

- размер популяции — 1000;
- число состояний автомата — 10;
- коэффициент скрещиваний — 0.5;
- коэффициент мутаций — 0.5;
- коэффициент элитизма — 0.2.

Функция приспособленности вычислялась в результате симуляции внешней среды, в которой выбор расположения очередного яблока осуществлялся с помощью генератора случайных чисел.

Значение функции определялось как $f = N * 100 + T / 10 + S_{used}$, где N — количество съеденных яблок за 1000 шагов автомата, T — число сделанных шагов до столкновения или останова, а S_{used} — число состояний, использованных в процессе работы. График работы генетического алгоритма, отражающий максимальное, минимальное и среднее значение функции приспособленности, приведен на рис. 17.

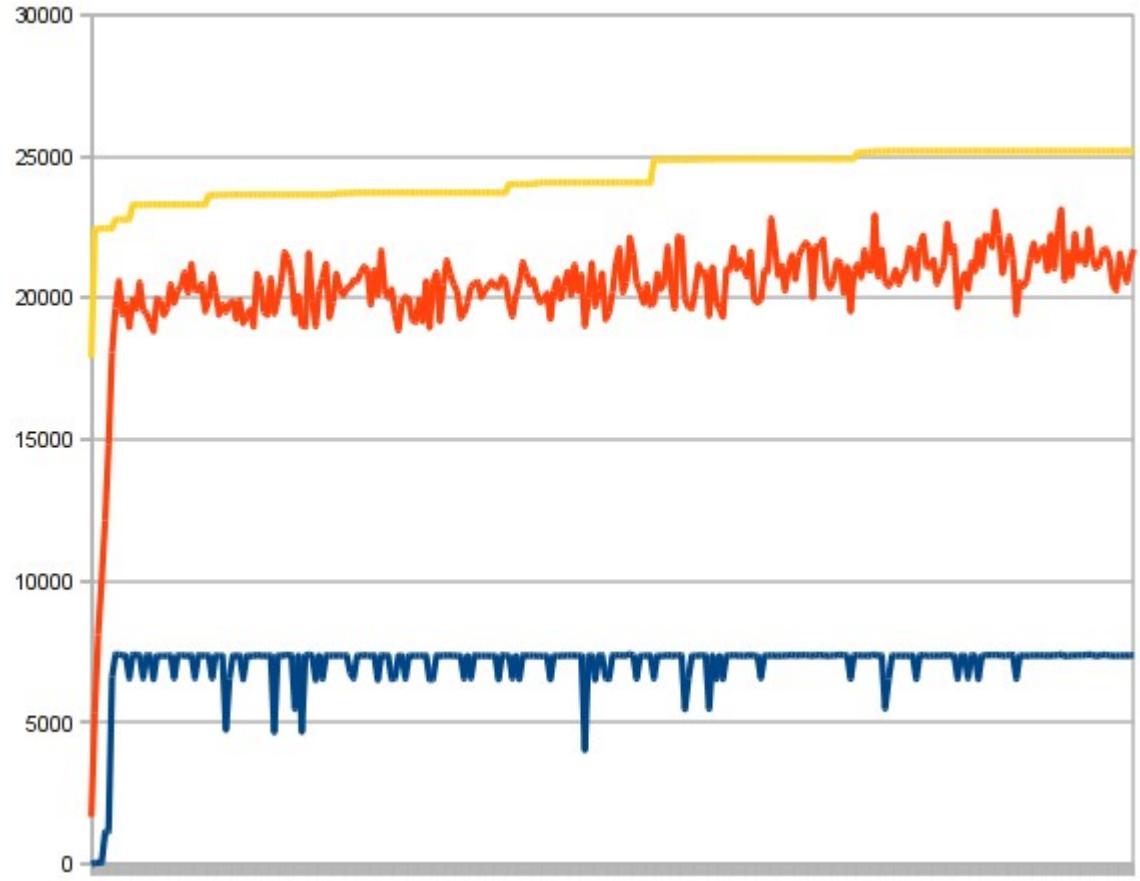

Рис. 17: График работы генетического алгоритма со случайным поведением среды

3.3.2. Управляющий автомат для внешней среды

Рассмотрим возможность задания конечного автомата, управляющего выбором расположения очередного яблока. Пусть входным воздействием автомата будет информация о том, с какого направления было съедено

предыдущее яблоко, а выходным воздействием — направление, в котором требуется создать новое. Расстояние до яблока — одна или две клетки — определяется случайно.

Генетический алгоритм из раздела 3.3.1 был дополнен популяцией конечных автоматов, управляющих внешней средой змейки. Функция приспособленности змейки была модифицирована как среднее значение для всех автоматов внешней среды. Функция приспособленности для автомата управления внешней средой вычислялась как среднее значение $f_{MAX} - f$ для всех змеек, где f_{MAX} - максимальное возможное значение приспособленности змейки. Размер дополнительной популяции был ограничен 100 особями в целях минимизации вычислительных затрат при тестировании.

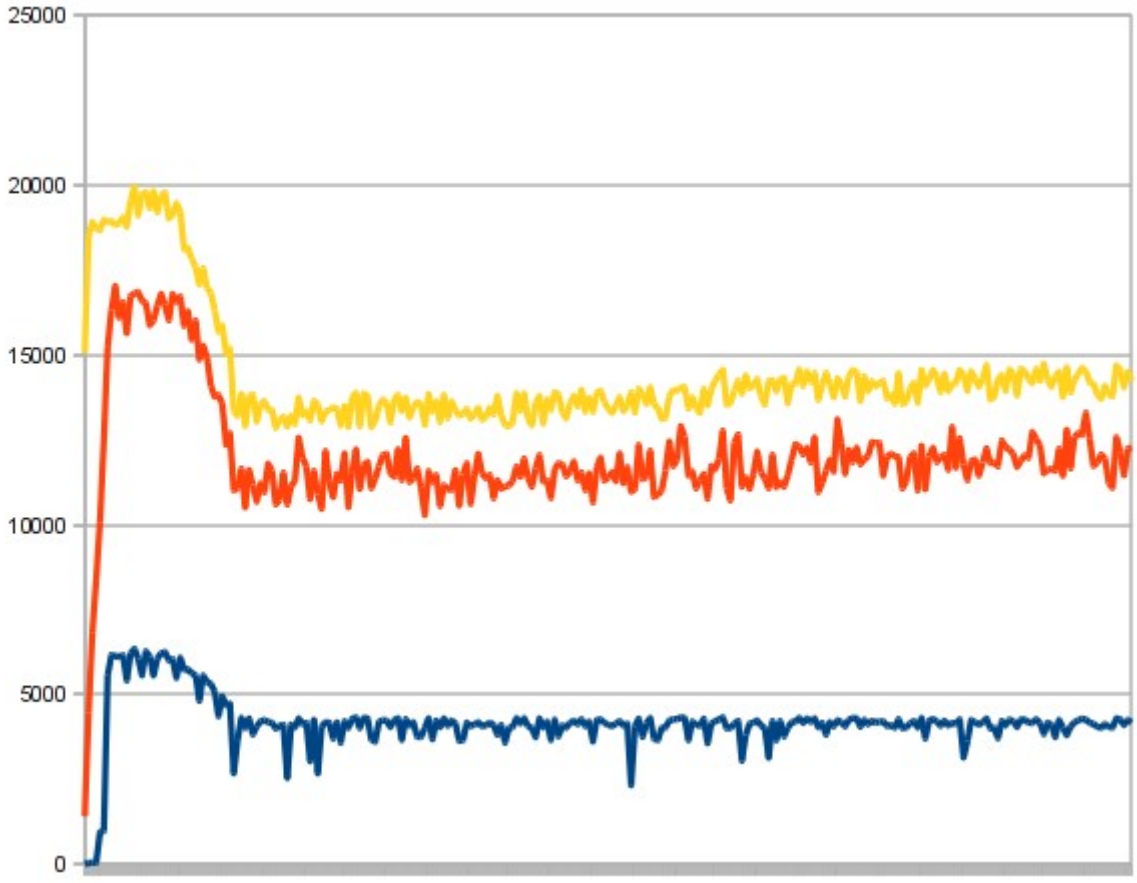

Рис. 18: График работы генетического алгоритма с двумя популяциями

Как видно из графика, приведенного на рис. 18, в данном эксперименте популяция змеек эволюционирует значительно медленнее, так и не достигая высоких значений функции приспособленности (не считая пика в начале работы). Изучение популяции автоматов, управляющих внешней средой, показало наличие специфической раскладки яблок, поедание которой представляется для змейки затруднительным. Обнаруженный автомат, раскладывающий яблоки, всегда выбирал направления 6 или 7, проиллюстрированные на рис. 16. Таким образом, змейке приходилось совершать несколько крутых разворотов и она рано или поздно врезалась сама в себя.

3.4. Анализ экспериментальных данных

Экспериментальные исследования, проведенные в данной главе, позволяют подтвердить эффективность предложенного метода.

В первом случае применение метода к задаче об «итерированной дилемме узника» позволило получить особей, способных эксплуатировать не только особенности своих собратьев, но и неизвестных им противников, что было показано на примере стратегии «Око за око».

Второй пример применения предложенного метода — представление внешней среды в виде конечного автомата для задачи о змейке — позволил установить ложно-положительный результат «наивного» решения данной задачи. В результате нахождения контрпримера к некорректному решению, генетический алгоритм получил возможность к дальнейшему росту приспособленности особей.

ЗАКЛЮЧЕНИЕ

Основные результаты работы состоят в следующем:

1. Разработан метод генетического программирования, позволяющий улучшить способ построения функции приспособленности для генетических алгоритмов, проводящих тестирование особей путем симуляции окружающей среды.

2. Модифицированный генетический алгоритм был интегрирован в инструментальное средство для генерации конечных автоматов при помощи генетических алгоритмов GAAP.

3. Построенный алгоритм и его реализация апробированы на двух экспериментальных задачах.

4. На основе анализа полученных экспериментальных данных сделан вывод об эффективности предложенного метода.

ПУБЛИКАЦИИ

1. Государственный контракт в рамках Федеральной целевой программы «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2012 годы»: Технология генетического программирования для генерации автоматов управления системами со сложным поведением.

2. *Е. А. Мандриков, В. А. Кулев, А. А. Шалыто* Применение генетических алгоритмов для создания управляющих автоматов в задаче о “флибах” // Информационные технологии. 2008, №1, стр. 42-45,89.

3. *Е. А. Мандриков, В. А. Кулев, А. А. Шалыто* Построение автоматов с помощью генетических алгоритмов для решения задачи о “флибах” / Сборник докладов X Международной конференции по мягким вычислениям и измерениям (SCM’2007). СПб.: СПбГЭТУ “ЛЭТИ”. 2007, Том 1, стр. 293-296.

4. *Е.А. Мандриков, В.А. Кулев, Ю.Д. Бедный, В.Р. Данилов* Разработка методов построения автоматов с помощью генетических алгоритмов / Сборник докладов IV Всероссийской межвузовской конференции молодых ученых (КМУ’2007). СПб.: СПбГУ ИТМО. 2007.

5. *Е.А. Мандриков, В.А. Кулев* Разработка инструментального средства для генерации конечных автоматов с использованием генетических алгоритмов // Научно-технический вестник СПбГУ ИТМО. Выпуск 53. Автоматное программирование, с. 100-103

6. *Е.А. Мандриков, В.А. Кулев* Программный комплекс для разработки и анализа различных генетических алгоритмов / Сборник докладов V Всероссийской межвузовской конференции молодых ученых (КМУ’2008). СПб.: СПбГУ ИТМО. 2008.

7. *Е.А. Мандриков, В.А. Кулев* Разработка инструментального средства для генерации конечных автоматов с использованием генетических алгоритмов / Сборник докладов Всероссийской научной конференции студентов и аспирантов “Молодые исследователи регионам”. ГОУ ВПО “ВГТУ”. 2008.

8. *Е.А. Мандриков, В.А. Кулев* Применение распределённых вычислений для автоматической генерации конечных автоматов с использованием генетических алгоритмов / Сборник докладов XI Международной конференции по мягким вычислениям и измерениям (SCM’2008). СПб.: СПбГЭТУ “ЛЭТИ”. 2008, с. 255-260.

9. *E.A. Mandrikov, V.A. Kulev* Development of Software System for State Machine Generation Using Genetic Algorithm / Proceedings of the Second Spring Young Researchers Colloquium on Software Engineering. SPb.: SPbSU. 2008. V. 1, pp. 59-60. (PDF)

10. *E.A. Mandrikov, V.A. Kulev* Applying Automata-Based Programming to Creating Business Processes Management Systems / Proceedings of the Third Spring Young Researchers Colloquium on Software Engineering. SPb.: SPbSU. 2009. pp. 114-115. (PDF)

СПИСОК ЛИТЕРАТУРЫ

1. *Бедный Ю. Д., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов в задаче "Умный муравей", 2007.
2. *Царев Ф. Н.* Совместное применение генетического программирования, конечных автоматов и искусственных нейронных сетей для построения системы управления беспилотным летательным аппаратом., 2008.
3. *Царев Ф. Н.* Разработка методов совместного применения генетического и автоматного программирования, 2009.
4. *Егоров К. В., Царев Ф. Н.* Совместное применение генетического программирования и верификации моделей для построения автоматов управления системами со сложным поведением, 2009.
5. *Holland J. H.* Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, 1975.
6. *MacKay D. J. C.* Information Theory, Inference, and Learning Algorithms., 2003.
7. *Radcliffe N. J.* Equivalence class analysis of genetic algorithms, 1991.
8. *Wright A. H.* Genetic algorithms for real parameter optimization, 1991.
9. *Michalewicz Z.* Genetic Algorithms + Data Structures = Evolution Programs, 1992.
10. *Eshelman L. J., Schaffer J. D.* Real-coded genetic algorithms and interval schemata, 1993.
11. *Muhlenbein H., Schlierkamp-Voosen D.* Predictive Models for the Breeder Genetic Algorithm I. Continuous Parameter Optimization, 1993.
12. *Koza J. R.* Genetic Programming: On the Programming of Computers by Means of Natural Selection, 1992.
13. *Данилов В. Р., Шалыто А. А.* Метод генетического программирования для генерации автоматов, представленных деревьями решений, 2008.
14. *Ferreira C.* Genetic representation and genetic neutrality in gene expression programming, 2002.
15. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Разработка библиотеки для генерации управляющих автоматов методом генетического программирования, 2007.
16. *Лобанов П. Г., Шалыто А. А.* Использование генетических алгоритмов для автоматического построения конечных автоматов в задаче о "Флибах", 2007.
17. *Царев Ф. Н., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов с минимальным числом состояний для задачи об "Умном муравье", 2007.
18. *Лобанов П. Г.* Использование генетических алгоритмов для решения задачи об "Умном муравье", 2007.
19. *Шалыто А. А.* Алгоритмизация и программирование для систем логического управления и "реактивных" систем, 2001.
20. *Angeline P. J., Pollack J.* Evolutionary module acquisition, 1993.

21. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A.* The Genesys System: Evolution as a Theme in Artificial Life, 1992.
22. *Chambers D. L.* Handbook of genetic algorithms. Complex coding systems., 1999.
23. *Каширин В. В.* Метод модификации оценочной функции для оптимизации работы генетического алгоритма, генерирующего конечные автоматы., 2008.
24. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. Учебно-методическое пособие, 2007.
25. *Заде Л., Дезоер Ч.* Теория линейных систем. Метод пространства состояний, 1970.
26. *Bäck Thomas* Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms, 1996.
27. *Мандриков Е. А.* Композиция методов генерации конечных автоматов на основе генетических алгоритмов, 2008.
28. *Мандриков Е. А., Кулев В. А.* Разработка инструментального средства для автоматической генерации конечных автоматов с ипользованием генетических алгоритмов, 2008.
29. *Mandrikov E. A., Kulev V. A.* Development of Software System for State Machine Generation Using Genetic Algorithms, 2008.
30. *Чеботарева Ю.К.* Применение генетических алгоритмов для генерации числовых последовательностей, описывающих движение, на примере шага вперёд человекоподобного робота.
31. *Кулев В.А.* Применение генетических алгоритмов для построения конечных автоматов при решении задач ориентирования на местности, 2008.
32. *Wikipedia* Prisoner's dilemma. http://en.wikipedia.org/wiki/Prisoner's_dilemma
33. *Axelrod R., Hamilton W.D.* The evolution of cooperation, 1981.
34. *Axelrod R.* The evolution of strategies in the iterated prisoner’s dilemma, 1987.
35. *Brunauer R., Locker A., Mayer H.A., Mitterlechner G., Payer H.* Evolution of iterated prisoner's dilemma strategies with different history lengths in static and cultural environments, 2007.
36. *Harrald P.G., Fogel D.B.* Evolving continuous behaviors in the iterated prisoner's dilemma, 1996.
37. *Fogel D.B., Inc N.S., La Jolla C.A.* Applying Fogel and Burgin's 'Competitive goal-seeking through evolutionary programming' to coordination, trust, and bargaining games, 2000.