Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра компьютерных технологий

Д. О. Соколов

Применение двухэтапного генетического программирования для построения автомата, управляющего моделью танка в игре Robocode

Дипломная работа

Научный руководитель: А. А. Шалыто

Санкт-Петербург
2009

Содержание

Введение

Искусственный интеллект является одним из наиболее перспективных разделов информатики. Сегодня многие теоретические разработки в этой области успешно применяются на практике. В ряде задач решения, созданные с помощью различных технологий искусственного интеллекта, не уступают разработанным людьми, а иногда и превосходят их.

Проблема построения управляющей системы для беспилотной техники стоит перед человечеством уже длительное время. Чаще всего данная проблема решается при помощи ручного труда. Однако данный метод не эффективен ввиду больших затрат ресурсов, а также качества построенных систем. В некоторых случаях построение управляющей системы при помощи ручного труда невозможно, ввиду сложности системы. Естественная идея — заставить компьютер строить управляющие системы.

Эволюционные вычисления являются одним из наиболее используемых направлений искусственного интеллекта и успешно применяются для автоматического создания программ [1]. Эффективность методов напрямую зависит от способа представления программы [2].

Для ряда задач поведение системы удобно представлять в виде конечных автоматов [3], [4].

Конечные автоматы, созданные с помощью генетических алгоритмов, успешно применяются в задачах распознавания регулярных языков. Другим применением эволюционных вычислений к построению автоматов является проектирование клеточных автоматов. В данной работе в качестве задачи, используется задача построения системы управления для модели танка в игре Robocode.

В настоящей работе предложен новый метод, основанный на совместном применении автоматов и нейронных сетей, а также модифицирован метод представления при помощи karva-деревьев ([5]). При уменьшении временных затрат на построение, в целях повышения результативности, в данной работе предложен подход разделения задачи на два этапа — *двухэтапное генетическое программирование*.

В заключение работы демонстрируются результаты апробации предложенных методов.

Глава 1. Генетические алгоритмы в применении к задачи управления системой со сложным поведением

1.1. Генетические алгоритмы

Многих исследователей давно интересовал вопрос — возможно ли эффективное построение важных и полезных для человека систем на основе принципов биологической эволюции?

В 1975 г. вышла книга Дж. Холланда «Адаптация в естественных и искусственных системах», в которой был предложен генетический алгоритм, исследованный в дальнейшем учениками и коллегами Дж. Холланда в Мичиганском университете. Эти работы заложили основы эволюционного программирования.

По сравнению с обычными оптимизационными методами эволюционные алгоритмы имеют особенности: параллельный поиск, случайные мутации и рекомбинации уже найденных хороших решений. Они хорошо подходят для оптимизации плохо определённых нелинейных функций [6].

Существуют различные модификации эволюционных алгоритмов, в том числе и генетические алгоритмы.

1.1.1. Общая схема генетических алгоритмов

На каждом этапе генетического алгоритма имеется множество *особей* x_k — *популяция*. Каждая особь характеризуется значением *функции приспособленности (фитнесс-функции)* — $f(x_k)$.

Задача генетического алгоритма — нахождение особи максимизирующее значение фитнесс-функции.

Общая схема выглядит следующим образом [6]:

1. Генерация стартового (начального) поколения.
2. Применение *генетических операторов*. Для получения новых особей.
3. Отбор особей для следующего поколения.
4. Если функция приспособленности лучшей особи в поколении удовлетворяет требованиям, то прекратить работу, иначе вернуться к шагу два.

Данная схема изображена на рисунке 1.1.

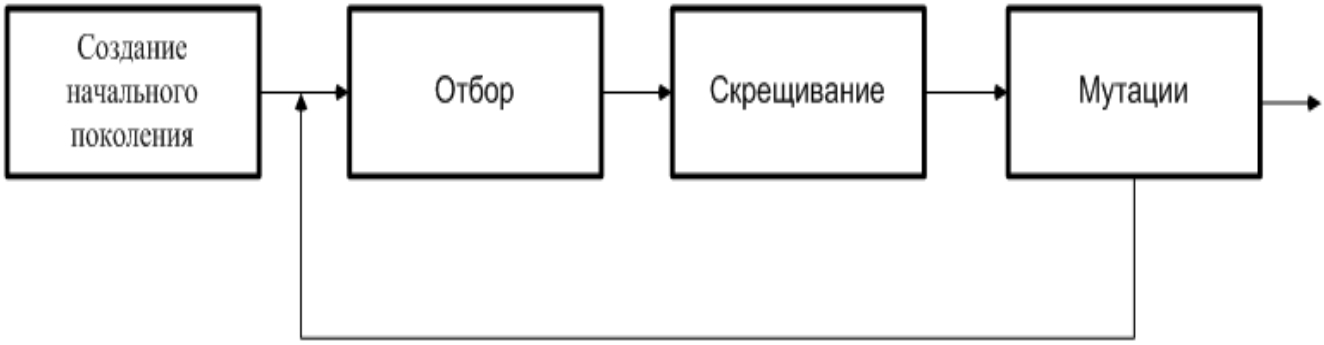

Рис. 1.1. Общая схема генетического алгоритма

1.1.2. Классический генетический алгоритм

Классический генетический алгоритм — простейшая реализация общей схемы генетического алгоритма (1.1.1). Особи представляются в виде битовых строк, для который определены генетические операторы *скрещивания (кроссовера)* и *мутации* (Более подробно операторы описаны в статье [6]). Операторы применяются последовательно.

1.2. Применение конечных автоматов при решении задачи управления

1.2.1. Постановка задачи

Для того, чтобы определить цель задачи управления формализуем понятие объекта управления. Данный объект характеризуется тройкой (X, Z, F), где:

- X — множество входных воздействий;
- Z — множество выходных воздействий;
- F — функция $X^n \to Z^m$, которая по заданным входным воздействиям выдаёт *реакцию* (т.е. множество выходных воздействий) управляемого объекта.

Цель задачи управления — построение, по заданным множествам X и Z, функции F удовлетворяющей требованиям конкретной задачи.

1.2.2. Конечные автоматы и особенности применения к данной задаче

Напомним, что представляет из себя конечный автомат: конечный автомат — это четвёрка (A, X, S, F), где:

- A — множество состояний;
- X — множество входных воздействий;
- $S \in A$ — начальное (стартовое) состояние;
- F — функция $F : A \times X^n \to A$ перехода.

В применении к данной задаче несколько модифицируем функцию переходов [7], а именно $F : A \times X^m \to A \times Z^n$, где Z — множество, описанное в разделе 1.2.1.

Основные достоинства применения конечных автоматов:

- полное отделение логики работы управляемого объекта от других частей реализации;

- возможность автоматического построения;
- учитывание зависимости работы управляемого объекта от предыстории.

Основные недостатки:

- сложность построения функции перехода, в том числе и в вручную;
- система управления не может изменяться во время работы;
- невозможность напрямую обрабатывать непрерывное множество входных воздействий.

Выводы по главе 1

В данной главе описаны общие сведения о генетических алгоритмах. Описана схема генетического алгоритма, при помощи модификации которой, в дальнейшем решается задача построения управляющей системы модели танка в игре Robocode.

Также в данной главе сформулирована общая задача управления системами со сложным поведением. К поставленной задаче адаптирован автоматный подход.

Глава 2. Задача управления моделью танка в игре Robocode

В данной главе ставится задача управления при создании системы управления танком для популярной компьютерной игры Robocode. А также проводится анализ предыдущих решений задачи.

Отметим, что не ставится цель создать танк, способный побеждать танки, построенные вручную, которые являются наилучшими из известных на сегодняшний день (например, танк voidious.Dookious). Алгоритмы, используемые в таких танках, являются достаточно сложными и их реализация занимает сотни килобайт, а работа по их созданию крайне трудоёмка.

Танки, создаваемые предлагаемым методом, должны побеждать в индивидуальном сражении каждого заранее выбранного соперника из тестового набора, который включает в себя танки, поставляемые вместе с игрой. Для каждого танка из тестового набора с помощью предлагаемого метода генерируется танк, который его побеждает. Модификация предлагаемого метода позволяет генерировать танки, системы управления которых, построены и без применения автоматного подхода.

На рисунке 2.1 в качестве примера приведена внешний вид среды игры Robocode во время одного из боев.

2.1. Обзор существующих решений

Известно большое число танков для рассматриваемой игры [8]. Наиболее простые из них реализованы всего несколькими строками кода, а программы, реализующие наиболее сложные танки, как отмечалось выше, занимают сотни килобайт. Большинство танков создано вручную. Существуют также работы [9] и [10], в которых танки строятся не вручную, а автоматически – с применением методов искусственного интеллекта.

В работе [9] использовано несколько таких методов (обучение с под-

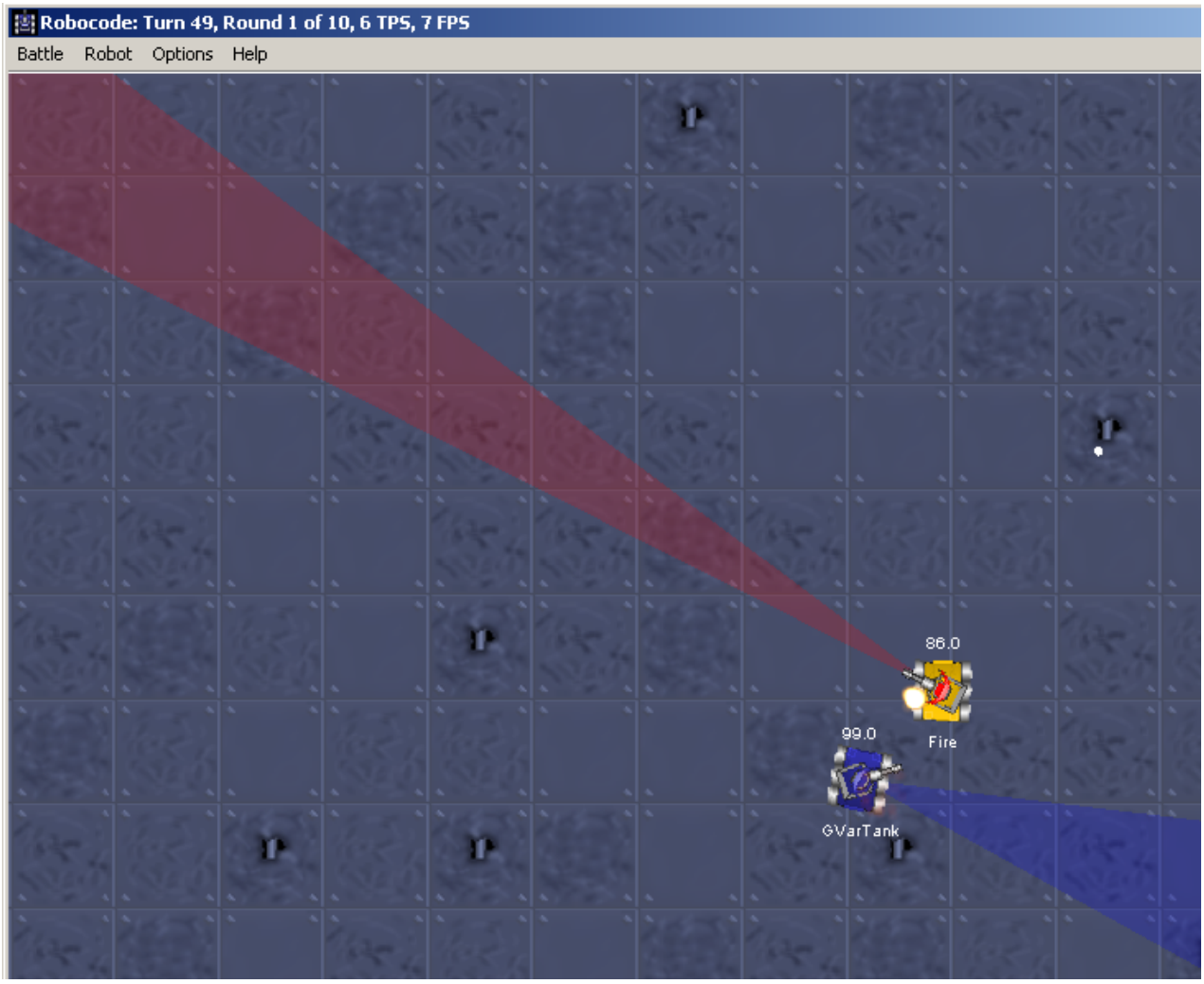

Рис. 2.1. Пример соревнования в игре *Robocode*

креплением, нейронные сети и генетические алгоритмы) для управления различными системами танка (радаром, системой движения и пушкой). Генетические алгоритмы в работе [9] применялись для создания систем управления радаром и движением танка. Однако их использование не привело к желаемому результату.

В работе [10] используется сравнительно сложный способ представления системы управления в виде особи генетического алгоритма – программы на языке *TableRex*. При этом длина программы составила 7776 бит.

Успешная попытка построения управляющей системы танка была

проделала в работе [5], данная работа является ее развитием.

2.2. Проблемы при решении задачи управления и предлагаемый подход их решения

При решении задачи управления остро встаёт проблема обработки вещественных переменных. В настоящей работе разработан метод обработки вещественных переменных при помощи нейронных сетей. Так же модифицирован метод обработки при помощи karva-деревьев (данный метод был разработан в работе [5]).

Как отмечает автор работы [5] основной проблемой при построении автоматов является большой размер пространства решений. В настоящей работе в разделе 3.4 произведена оценка размера пространства решений для двух подходов (оценка для деревьев разбора верна и для работы [5]). Также в главе 4 двухэтапный генетический алгоритм, который позволяет уменьшить пространство решений, а следовательно повысить качество получившихся решений. Также данный метод позволят сократить время работы алгоритма.

2.3. Формальная постановка задачи

Устройство танка схематично показано на рис. 2.2. Необходимо управлять следующими устройствами: радаром (Radar), пушкой (Gun) и системой движения (Body).

Управление танком осуществляется следующим образом (аналогично работе [5]). В каждый момент времени анализируется текущая ситуация (положение танка, его скорость и т.д.). На основе этой информации формируются четыре действия: передвижение (вперёд/назад), поворот, поворот пушки, стрельба. Указанные действия формируются в результате интерпретации функции управления, которая генерируется на основе генетиче-

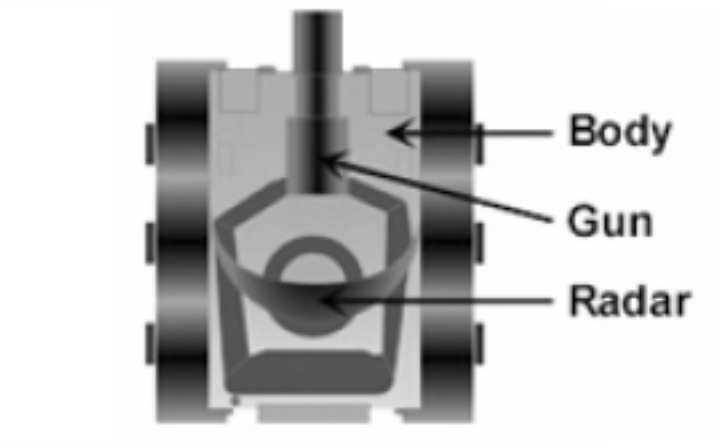

Рис. 2.2. Изображение модели танка

ских алгоритмов.

Переформулируем и конкретизируем постановку задачи управления данную в разделе 1.2.1. Обозначим текущее время в игре как $t \in N$. Обозначим множество позиций как S, а множество действий танка как A.

Позиция – элемент некоторого множества всех возможных ситуаций в игре в выбранный момент времени. Позиция включает в себя положение каждого танка, его скорость, направление, угол поворота пушки и радара, энергию, информацию о танке соперника и т. д. Задача управления для рассматриваемой игры в общем случае состоит в задании для каждого момента времени t функции управления $f_t : S^t \Rightarrow A$, которая на основании информации о позициях во все предыдущие моменты времени (получаемой из среды *Robocode*) определяет действие на объект управления в текущий момент времени. Таким образом, решение задачи управления танком – вектор $[f_1, f_2, ...]$.

Упростим задачу управления. Пусть:

1. Действие танка в текущий момент времени зависит только от позиции в этот момент времени.
2. Зафиксирован алгоритм управления радаром – вращение по кругу.
3. При выборе действия анализируется лишь упрошенная позиция –

элемент множества $S' = (x, y, dr, tr, w, dh, GH, h, d, e, E)$. Здесь:

- x, y – координаты танка соперника относительно нашего танка;
- dr – расстояние, которое осталось "доехать" нашему танку (после вызова метода AdvancedRobot.setAhead);
- tr – угол, на который осталось повернуться нашему танку;
- w – расстояние от нашего танка до края поля;
- dh – угол между направлением на танк соперника и пушкой нашего танка;
- GH – угол поворота пушки нашего танка;
- h – направление движения танка соперника;
- d – расстояние между нашим танком и танком соперника;
- e – энергия танка соперника;
- E – энергия нашего танка;

4. Множество действий $A' = g, p, d, h$, где:
   - g – угол поворота пушки;
   - p – энергия снаряда (не положительные значения означают, что выстрел не производится);
   - d – расстояние, на которое перемещается танк;
   - h – угол поворота танка.

Учитывая изложенное, задача построения системы управления танком сведена к заданию функции $f : S' \to A'$. Данную функцию будем искать при помощи генетического алгоритма.

Выводы по главе 2

В настоящей главе конкретизирована и адаптирована задача построения системы управления объектами со сложным поведением для модели

танка в игре Robocode. Описано взаимодействие модели танка с внешней средой.

Кроме того, проведён обзор предыдущих решений решений поставленной задачи. А также отмечены основные проблемы, возникающие при решении, которые решаются в данной работе.

Глава 3. Способы представления управляющей системы

Для управления моделью танка необходим объект, который по вещественным данным (т.е. по значениям элементов из множества входных воздействий) выдаёт также вещественные значения (т.е. значения для выходных воздействий). Для решения данной задачи автомат не подходит, так как автомат умеет анализировать конечное число вариантов входных воздействий, и выдавать также лишь конечное число значений. Проблема может быть решена несколькими способами, о которых пойдёт речь в данной главе.

В данной главе число входных переменных будем обозначать N_{in}, выходных N_{out}, а за N — число состояний автомата.

3.1. Нейронные сети

Для того, чтобы обрабатывать вещественные входные переменные в каждое состояние автомата поместим обработчик — нейронную сеть. в каждом состоянии используется своя нейронная сеть, но эти сети имеют одинаковую структуру (более подробно о структуре написано в разделе 3.1.1). Данные сети имеют N_{in} входов и $N_{out}+1$ выходов. пример изображён на рисунке 3.1.

Первые N_{out} выходов нейронной сети — это значения, которые непосредственно передаются модели танка, а последний выход отвечает за новое состояние, в которое переходит управляющий автомат. Чтобы по последнему выходу получить номер состояния, возьмём целую часть значения, которое получено на выходе (обозначим ее за W), существует три случая:

- $W \geq N$, тогда переходим в состояние с номером N;
- $W \leq 1$, тогда переходим в состояние с номером 1;

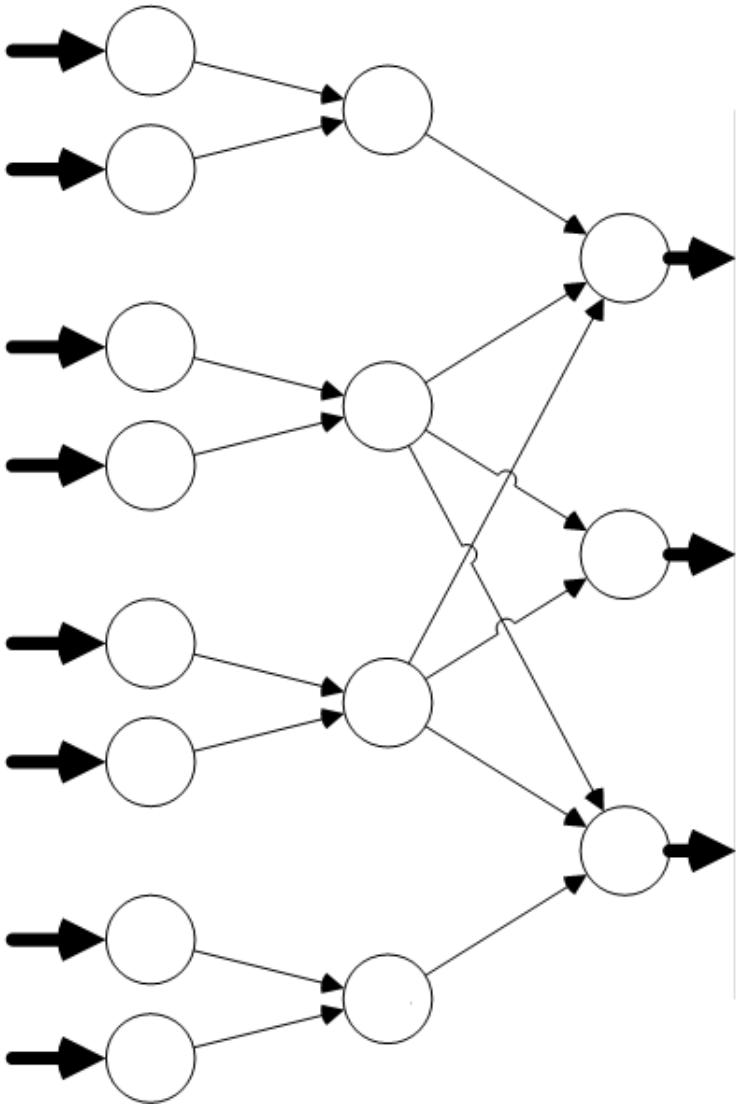

Рис. 3.1. Пример нейронной сети

- во всех других случаях переходим в состояние в номером W.

3.1.1. Структура нейронной сети

Структура нейронной сети задаётся вручную. Структура должна представлять собой ациклический граф и иметь по крайней мере $\max(N_{in}, N_{out})$ вершин. Входные значения подаются на нейроны с номерами от 1 до N_{in}, а выходные берутся с нейроном имеющих номера с $N_{net} - N_{out} - 1$ до N_{net}, где N_{net} — число нейронов.

Также вручную задаётся список возможных типов нейронов (т.е.

функций, которые вычисляют нейроны). В данной работе были использованы нейроны вычисляющие следующие функции:

- $\frac{a}{x} + b$;
- $\begin{cases} x, x \geq a \\ 0, x < a \end{cases}$
- $a * x + b$;
- $\frac{1}{1+\exp(-x)}$;
- x.

В данных обозначениях, a, b — параметры нейрона, x — сумма значений, поданных на нейрон.

Структура нейронной сети, используемая в настоящей работе, изображена на рис. 3.2.

3.1.2. Оператор мутации нейронной сети

Оператор мутации для нейронной сети выбирает случайным образом один из нейронов и мутирует его. Оператор мутации для нейрона выполняет следующие действия:

- с вероятностью 0.5 изменяет тип нейрона;
- изменяет вес одного из исходящих рёбер не более чем на 0.5 от текущего веса;
- если тип нейрона не менялся то изменяются параметры нейрона (3.1.1) не более чем на 200% от текущего значения.

Пример мутации изображён на рис. 3.3.

3.1.3. Оператор кроссовера (скрещивания) нейронных сетей

В данной работе было реализовано два оператора кроссовера для нейронных сетей по аналогии с *одноточечным* и *однородным* кроссовером

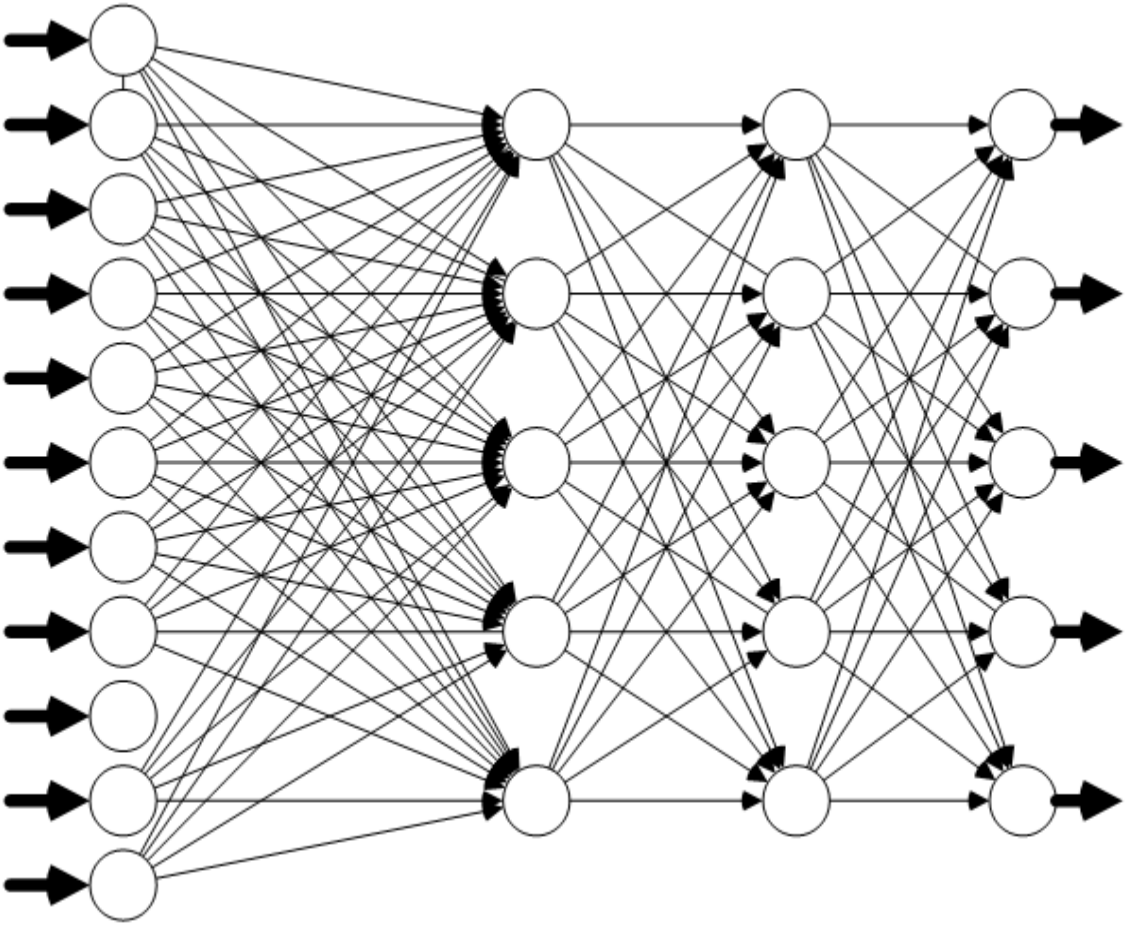

Рис. 3.2. Структура нейронной сети

для битовых строк [6]. Оба этих оператора принимают на вход две особи P_1, P_2 (*родители*) и возвращают также две особи S_1, S_2 (*дети*).

в данном разделе будем использовать следующие обозначения:

- $P_j.n_i$ — i-ый нейрон j-ой родительской особи;
- $S_j.n_i$ — i-ый нейрон j-ого ребёнка;
- w_j — вес j-ого ребра, выходящего из нейрона.

3.1.3.1. Оператор однородного кроссовера

Все нейроны попарно скрещиваются. После применения оператора кроссовера, для любых i, j равновероятны следующие случаи:

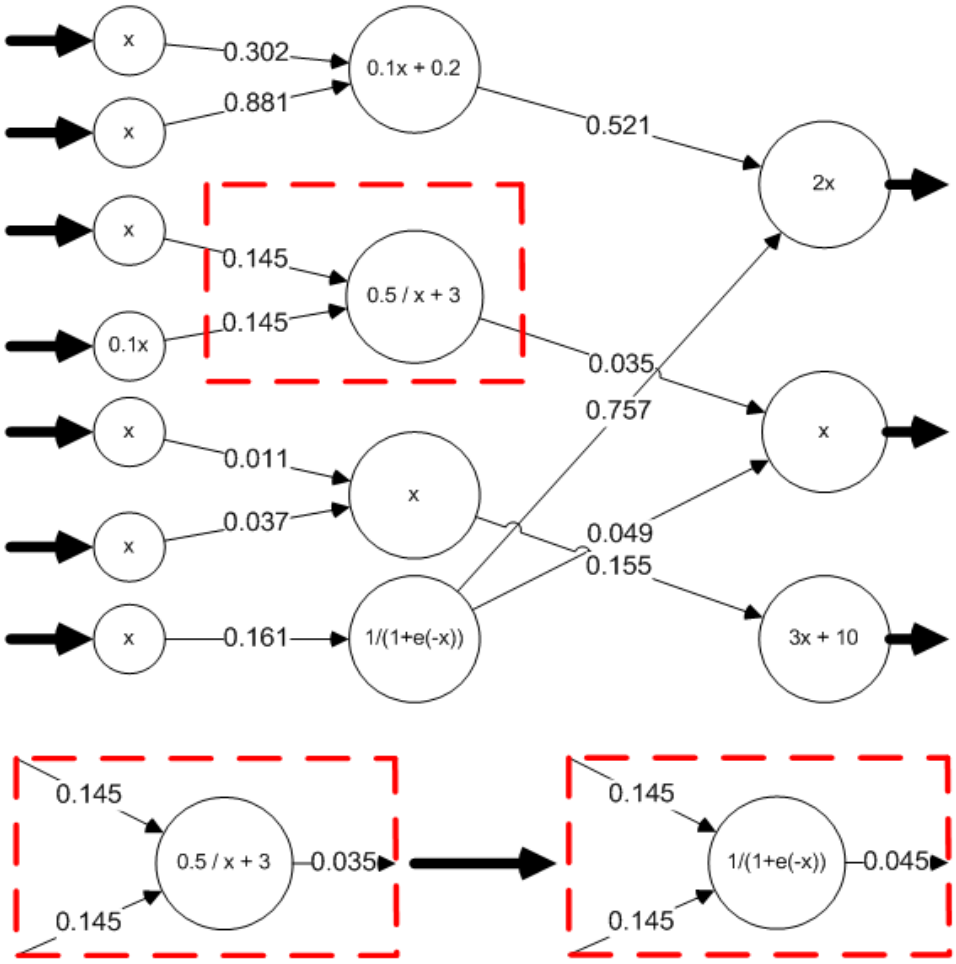

Рис. 3.3. Пример мутации нейронной сети

- $$\begin{cases} S_1.n_i = P_1.n_i \\ S_1.n_i.w_j = P_1.n_i.w_j \\ S_2.n_i = P_2.n_i \\ S_2.n_i.w_j = P_2.n_i.w_j \end{cases}$$
- $$\begin{cases} S_1.n_i = P_1.n_i \\ S_1.n_i.w_j = P_2.n_i.w_j \\ S_2.n_i = P_2.n_i \\ S_2.n_i.w_j = P_1.n_i.w_j \end{cases}$$

- $$\begin{cases} S_1.n_i = P_2.n_i \\ S_1.n_i.w_j = P_1.n_i.w_j \\ S_2.n_i = P_1.n_i \\ S_2.n_i.w_j = P_2.n_i.w_j \end{cases}$$
- $$\begin{cases} S_1.n_i = P_2.n_i \\ S_1.n_i.w_j = P_2.n_i.w_j \\ S_2.n_i = P_1.n_i \\ S_2.n_i.w_j = P_1.n_i.w_j \end{cases}$$

Работа данного оператора проиллюстрирована на рисунке 3.4 (за исключением весов рёбер).

3.1.3.2. Оператор *одноточечного* кроссовера

Разделим все нейроны на две части на два множества (обозначим их как V_1 и V_2), по удалённости от ближайшего нейрона, на который подаются входные значения, обозначим это расстояние для i-ого нейрона за d_i. Для начала заметим что наш граф по построению представляет собой *DAG* (directed acyclic graph) (см. раздел 3.1.1), поэтому для него применим алгоритм *DAG shortest path* [11].

Пусть $d_{max} = \max\limits_i(d_i)$, тогда разделим нейроны следующим образом, для любого $j \in \{1, 2\}$:

- если $d_i < \frac{d_{max}}{2}$, то $P_j.n_i \in V_1$
- если $d_i > \frac{d_{max}}{2}$, то $P_j.n_i \in V_2$
- если $d_i = \frac{d_{max}}{2}$, то $P_j.n_i$ равновероятно занесём в одно из множеств.

Разбиение проиллюстрировано на рисунке 3.5 (серым цветов отмечены вершины, принадлежащие множеству V_1).

После разбиения нейронов на два множества конструируем дочерних особей по следующим правилам:

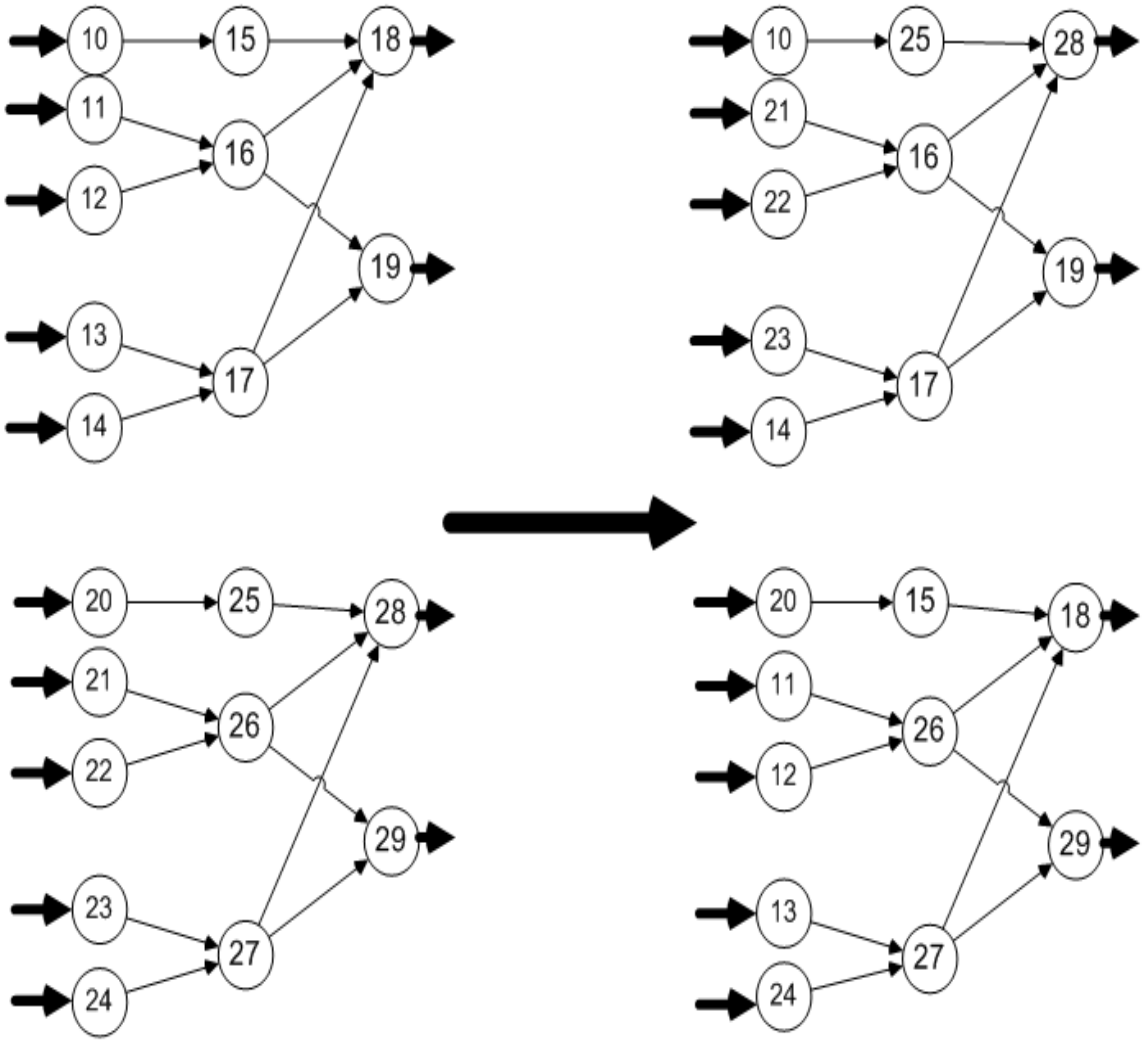

Рис. 3.4. Пример однородного скрещивания нейронных сетей

- если $P_1.n_i \in V_1$, то для любого j $\begin{cases} S_1.n_i = P_1.n_i \\ S_1.n_i.w_j = P_1.n_i.w_j \\ S_2.n_i = P_2.n_i \\ S_2.n_i.w_j = P_2.n_i.w_j \end{cases}$
- если $P_1.n_i \in V_2$, то для любого j $\begin{cases} S_1.n_i = P_2.n_i \\ S_1.n_i.w_j = P_2.n_i.w_j \\ S_2.n_i = P_1.n_i \\ S_2.n_i.w_j = P_1.n_i.w_j \end{cases}$

Пример одноточечного кроссовера изображён на рисунке 3.5.

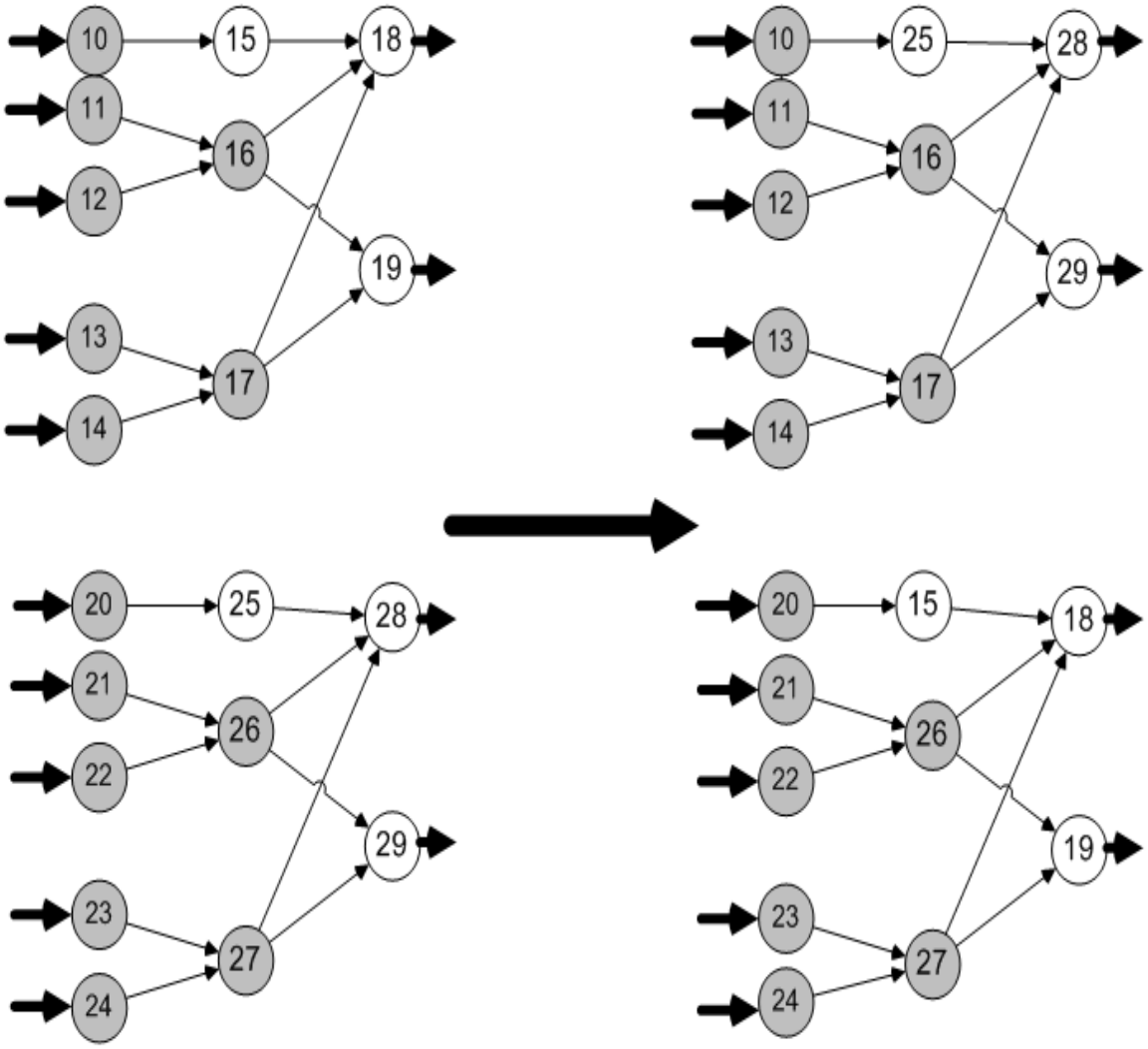

Рис. 3.5. Пример одноточечного скрещивания нейронных сетей

3.2. Деревья разбора

Данный способ собой комбинацию способов, описанных в работах [5] и [12]. В каждое состояние конечного автомата теперь поместим четыре, так называемых дерева разбора. А так же будем получать

Дерево разбора представляет собой дерево, в котором во внутренних вершинах находятся функции. У каждой внутренней вершины ровно столько потомков, какова арность функции в данной вершине. В листья подаются входные переменные или заранее зафиксированные константы (более подробно, а также формальное определение можно прочитать в ра-

боте [5]).

В данной работе в качестве внутренних функций и констант взяты аналогичные работе [5], а именно:

- $if(x < y)\ return\ w\ else\ return\ v$
- $\min(x, y)$
- $x * y$
- $-x$
- $\frac{1}{1+\exp(-x)}$
- $x + y + w$
- $x * y * w$
- константы: 0.1 0.5 1 2 5 10

Пример дерева разбора изображён на рисунке 3.6.

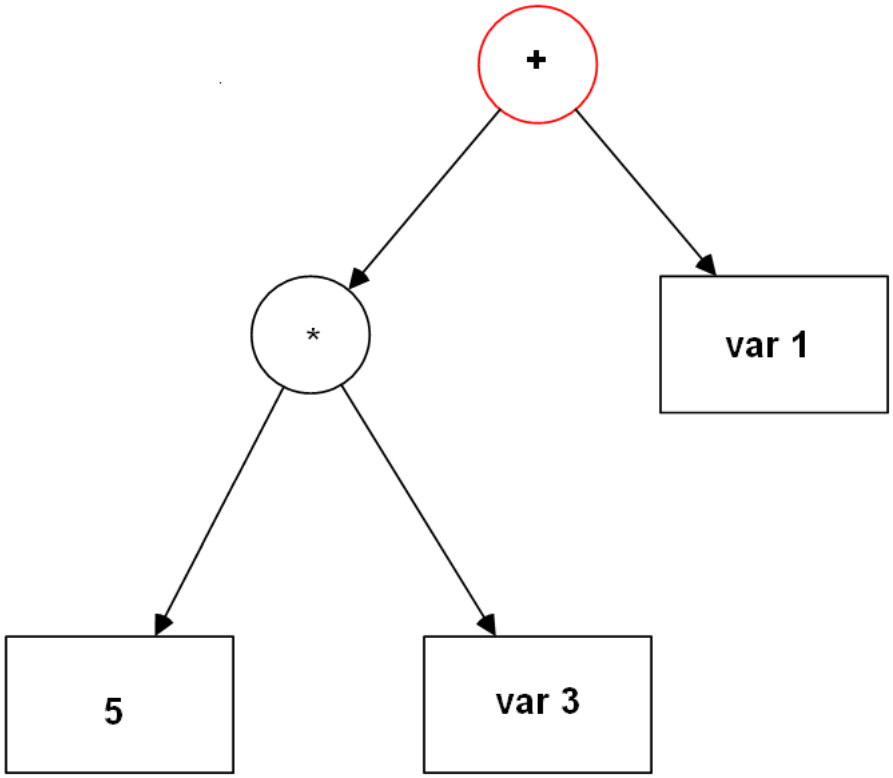

Рис. 3.6. Пример дерева разбора

Условное обозначение *var i* означает, что в данных лист подаётся значение i-ой переменной.

3.2.1. Генерация случайного дерева

Генерация случайного дерева выполняется рекурсивным образом. Алгоритм выглядит состоит из 4-ех шагов.

1. С вероятностью p генерируется внутренняя вершина, иначе переходим в шагу 4.
2. Равновероятно выбираем функцию.
3. Рекурсивно вызываем процедуру генерации нужного количества детей.
4. Равновероятно из множества переменных и констант выбираем одну.

Чтобы деревья получались не вырождались и не слишком *разрастались*, в настоящей работе вероятность $p = c_1 c_2^h$, где $c_1 \leq 1, c_2 \leq 1$ — константы, а h — текущая высота генерируемой вершины. Оценка числа вершин будет произведена в разделе 3.4.

3.2.2. Оператор мутации

Оператор мутации, также как и генератор случайного дерева, представляет собой рекурсивную процедуру. С вероятностью p запускаем процедуру мутации для случайного поддерева, иначе заменяем текущее поддерево на случайно сгенерированное.

Работа данной процедуры проиллюстрирована на рисунке 3.7.

3.2.3. Оператор скрещивания

Данный оператор также представляет собой рекурсивную процедуру, и, как и в случае с нейронными сетями (см. раздел 3.1.3) получает на вход две особи (P_1, P_2), и выдаёт две особи (S_1, S_2).

У особей P_1 и P_2 выбирается по случайному поддереву, при помощи алгоритма аналогичному, описанному в разделе 3.2.2. Данный алгоритм проиллюстрирован на рисунке 3.8.

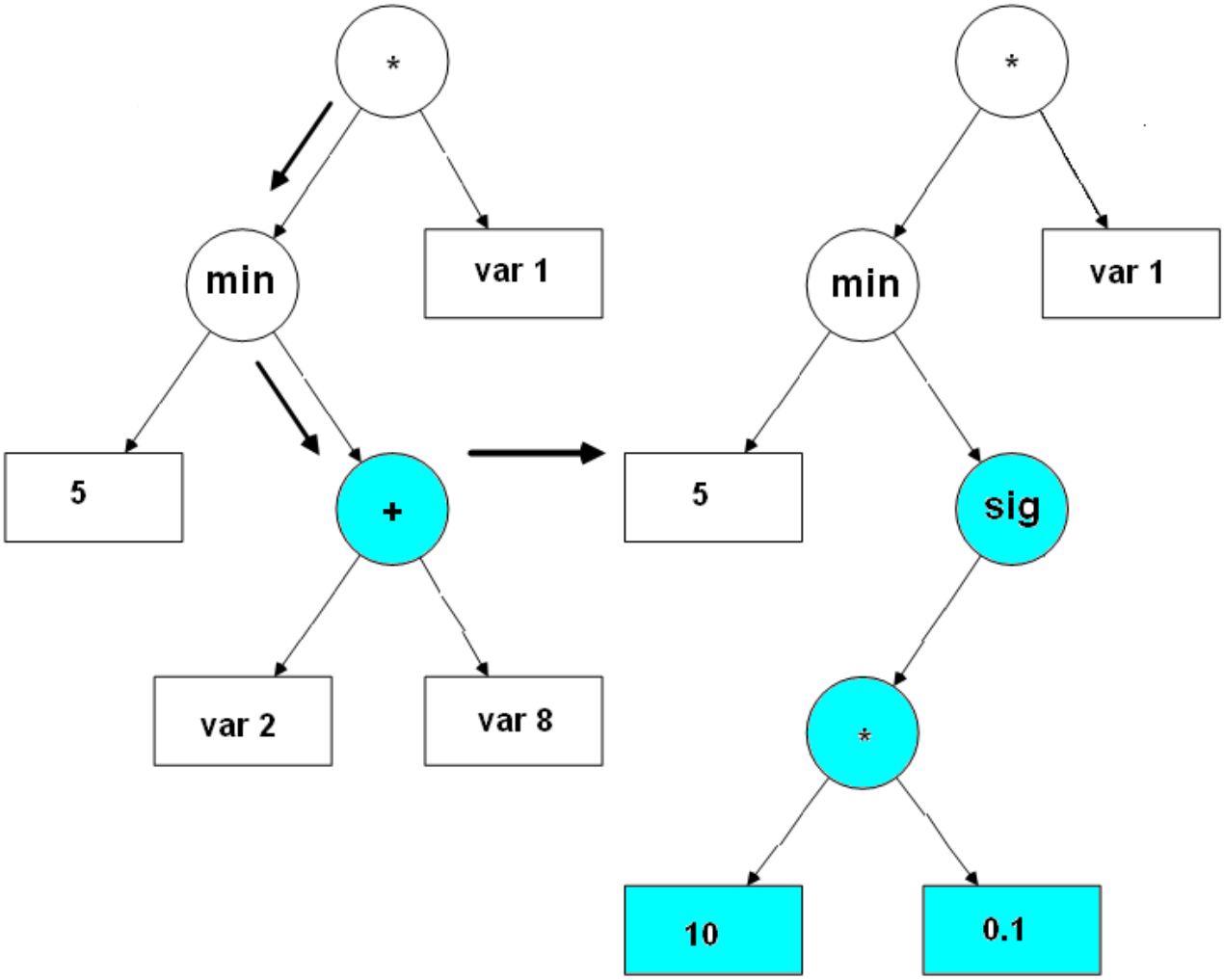

Рис. 3.7. Пример мутации дерева разбора

3.2.4. Совместное применение конечных автоматов и деревьев разбора

В отличие от нейронных сетей, дерево разбора выдаёт только одну вещественную переменную, поэтому в каждое состояние автомата поместим N_{out} деревьев разбора. Для того чтобы определить новое состояние в которое необходимо перейти автомату, будем также вместе с автоматом выращивать столько деревьев разбора, сколько необходимо логических переменных. При работе автомата будем считать, что если дерево разбора выдало значение больше нуля, то значение соответствующей логической переменной равно 1, иначе 0.

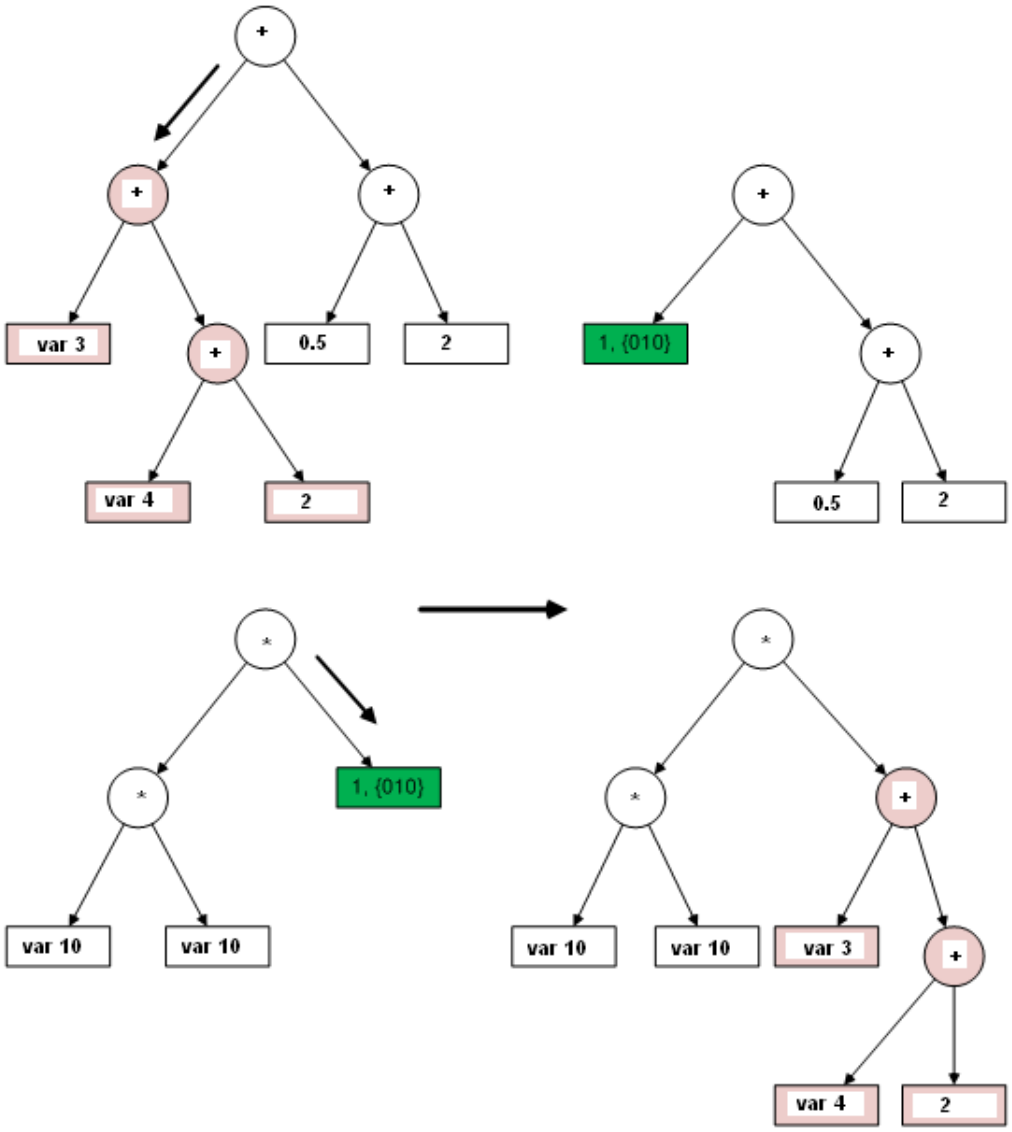

Рис. 3.8. Пример скрещивания деревьев разбора

3.3. Конечные автоматы

Для конечных автоматов с обработчиком вещественных переменных генетические операторы определены следующим образом.

Оператор мутации выполняет следующие действия:

- с вероятностью 0.5 изменяется стартовое состояние;
- равновероятно выбирается одно из состояний и мутирует;
- в случае с Karva-деревьями также мутирую деревья отвечающие за логические переменные.

Оператор скрещивания, также как и во всех предыдущих случаях, получает на вход две особи (P_1, P_2), и выдаёт две особи (S_1, S_2). За $P_i.n_j$ обозначим j-ое состояние автомата P_i. Тогда оператор скрещивания выполняет следующие действия:

1. Стартовое состояние S_i приравнивается стартовому состоянию P_i.
2. Для каждого $i \in 1..N$ проводим скрещивание для $P_1.n_i$ и $P_1.n_i$, результат обозначим за V_1, V_2. Возможны два случая:
   - $\begin{cases} S_1.n_i = V_1 \\ S_2.n_i = V_2 \end{cases}$
   - $\begin{cases} S_1.n_i = V_2 \\ S_2.n_i = V_1 \end{cases}$
3. в случае с Karva-деревьями попарно скрещиваются деревья отвечающие за логические переменные, а переходы скрещиваются аналогично работе [7].

3.4. Особенности применения конечных автоматов с нейронными сетями и автоматов с деревьями разбора

При использовании обоих подходов наблюдаются все стандартные проблемы при использовании генетических алгоритмов [6].

Основные особенности использования в качестве обработчика вещественных переменных нейронных сетей:

- в каждом состоянии необходима только одна нейронная сеть;
- зависимость между переменными (каждый нейрон влияет на каждую переменную).

Попробуем оценить размер пространства решений в случае с нейронными сетями.

Пусть V — число нейронов, E — число рёбер, а T — число типов нейронов. У каждого ребра есть вес, который представляет собой вещественное число, таким образом пространство решений уже представляет собой R^E. Но в компьютере точно невозможно представить вещественное число, предположим, что для представления взят тип *double* в языке *Java* — 64 бита, тогда $R^E \approx 2^{64E}$.

Каждый узел нейронной сети может быть одним из T вариантов, получаем T^V. Итого, одна нейронная сеть выбирается из $T^V 2^{64E}$ вариантов. Таким образом пространство решений автомата имеет размер $|\Omega_{net}| = (T^V 2^{64E})^N$.

Основные особенности использования в качестве обработчика вещественных переменных деревьев разбора:

- в каждом состоянии необходимо несколько деревьев разбора;
- необходимо получение дополнительных деревьев;
- отсутствие зависимости между переменными.

Попробуем оценить размер пространства решений в данном случае.

Напомним, что вероятность генерации внутренней вершины на высоте h выражается формулой $p = c_1 c_2^h$ (см. раздел 3.2.1). Математическое ожидание числа вершин в дереве разбора (E) — это константа зависящая от выбора параметров c_1, c_2. Таким образом число видов деревьев можно представить в виде Z^E, где Z — число различных функций. А для автомата в целом получим соотношение: $|\Omega_{tree}| = (Z^E)^{V_c}(Z^E)^{4N} N^{2^{V_c}}$, где V_c — число логических переменных.

Несложно заметить, что с учётом малости V_c, выполнено соотношение $|\Omega_{tree}| \ll |\Omega_{net}|$, таким образом при прочих равных условиях деревья разбора должны гораздо быстрее находить решение задачи, что полностью подтверждается практикой.

Выводы по главе 3

В настоящей главе описаны две методики обработки вещественных переменных при использовании конечных автоматов, а именно:

- совместное применение конечных автоматов и нейронных сетей;
- совместное применение конечных автоматов и деревьев разбора.

Таким образов решается одна из проблем, описанных в главе 2. Данные методы адаптированы к генетическим алгоритмам, то есть описаны генетические операторы для каждого из предложенных методов.

Также в данной главе произведена оценка пространства решений при построении соответствующих автоматов, размер данного пространства является существенной проблемой для автоматического построения управляющей системы.

Глава 4. Применение двухэтапного генетического алгоритма для построения конечных автоматов с деревьями разбора

В разделе 3.4 был оценён размер пространства решений для поиска управляющего автомата. Для его дальнейшего уменьшения можно предложить два способа:

- принципиально новый метод обработки вещественных переменных, или замена конечных автоматов на другую управляющую структуру;
- разбиение задачи построения управляющего автомата на части и решение данных частей по отдельности.

В данной главе рассматривается второй способ решения проблемы уменьшения пространства решений.

4.1. Схема *островного* двухэтапного генетического алгоритма

Островной генетический алгоритм, является расширением классического (см. раздел 1.1.2). Также для избежания классического проблемы генетических алгоритмов — попадания в локальные максимумы, в данный алгоритм были добавлены ещё один генетический оператор — оператор большой мутации. Общая схема данного алгоритма выглядит следующим образом (рис. 4.1).

Основные отличия островного от классического генетического алгоритма:

- разделение популяции на несколько, развивающихся независимо — разделение на *острова*;

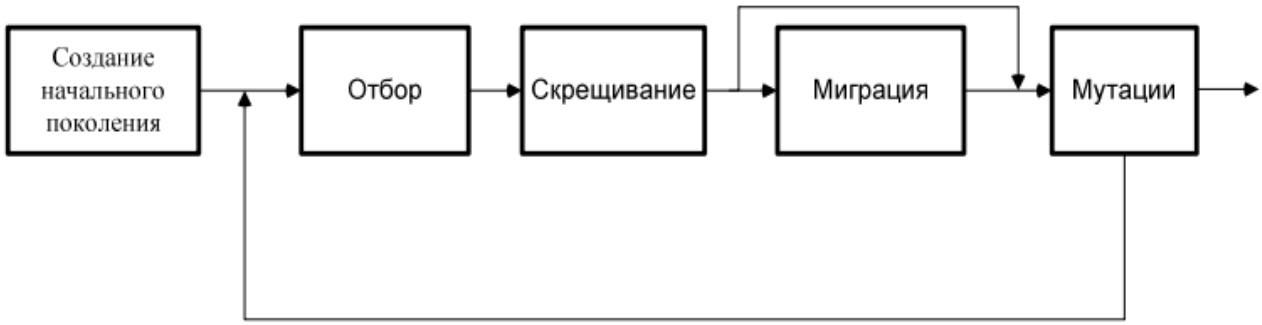

Рис. 4.1. Схема островного генетического алгоритма

- добавление оператора миграции (более подробно написано в разделе 4.2).

Каждый из двух этапов алгоритма проходит по этой схеме. На первом этапе строятся четвёрки деревьев разбора (т.е. фактически автомат, состоящий из одного состояния). Эти четвёрки сохраняются в *репозитории*. Причём для каждого из деревьев известно, за какое из действий модели танка он отвечает. На втором этапе генерируется сам управляющий автомат. Для генерации состоянии автомата используется репозиторий с сохранёнными деревьями. Более подробно написано в разделах 4.3 и 4.4.

4.2. Общие для двух этапов фазы генетического алгоритма

В качестве основной стратегии формирования следующего поколения используется элитизм. При рассмотрении текущего поколения отбрасываются все особи, кроме некоторой доли наиболее приспособленных – «элиты». Эти особи переходят в следующее поколение. После этого оно дополняется в определённой пропорции случайными особями, особями из текущего поколения, которые мутировали, и результатами скрещивания

особей из текущего поколения (отдельно отметим, что скрещиваться могут не только элитные особи, а все). Особи, «имеющие право» давать потомство, определяются «в поединке»: выбираются две случайные пары особей, и более приспособленная особь в каждой из них становится одним из родителей.

Через фиксированное число поколений каждый остров меняется с другим случайным числом случайно выбранных элитных особей.

Через заранее заданное число поколений происходит *большая мутация* — фиксированная доля островов заменяется островами со случайными особями. Проведение мутации в момент, когда функция приспособленности (имеются в виду только элитные особи) изменяется незначительно, невозможно, так как за счёт миграции особей между островами среднее значение функции приспособленности постоянно изменяет свое значение.

4.3. Особенности первого этапа генерации особей

Как уже было написано в разделе 4.1, на первом этапе генетического алгоритма выращиваются четвёрки деревьев разбора, или что тоже самое конечные автоматы. Начиная с заранее заданного поколения, лучшая особь добавляется в репозиторий.

4.3.1. Генерация начального поколения и случайных особей

Случайные особи генерируются по алгоритму, описанному в разделе 3.2.1. При генерации стартового поколения, все острова заполняются случайно сгенерированными особями.

4.3.2. Подсчёт фитнесс-функции

При подсчете фитнесс-функции используется моделирование соревнования между моделью танка, управляемая тестируемой особью и зара-

нее выбранным противником. Заранее выбрать фитнесс-функцию не представляется возможным, так как мы генерируем не всю особь целиком, а лишь одно состояние. В данной работе использовались следующие фитнесс-функции:

- $t.damage + t.bulletDamage$
- $\frac{(t.survival - n)n}{t.bulletDamage}$
- $\frac{t.score + t.survival * t.bulletDamage}{t.score + t.survival * t.bulletDamage + e.score + e.survival * e.bulletDamage}$
- $\frac{t.score}{t.score + e.score}$
- $t.score$

Поясним обозначения, используемые в фитнесс-функциях:

- t — модель танка, управляемая тестируемой особью;
- e — модель танка противника;
- $damage$ — нанесённый противнику ущерб (по средством попаданий и столкновений);
- $bulletDamage$ — нанесённый противнику только по средствам попаданий ущерб;
- $survival$ — число раундов в которых танк выжил;
- $score$ — число очков набранных танком (вычисляется системой *Robocode*);
- n — число раундов в соревновании.

Для избежания излишней *пассивности* (т.е. отсутствия стрельбы) танков, часть запусков алгоритма проводилась в запретом на движение у тестируемой модели танка.

4.4. Особенности второго этапа генерации особей

После первого этапа доступен репозиторий с деревьями разбора, где деревья сгруппированы по действиям, за которые они отвечают.

4.4.1. Генерация начального поколения и случайных особей

Состояния конечных автоматов генерируется следующим образом: из репозитория равновероятно выбирается по одной особи, отвечающей за каждое действие. Конечные состояния переходов генерируются случайным образом. Деревья разбора, отвечающие за логические переменные в конечном автомате, генерируются по алгоритму, описанному в разделе 3.2.1. При генерации стартового поколения, все острова заполняются случайно сгенерированными особями.

Также отметим, что операторы скрещивания и мутации к деревьям разбора, находящимся *внутри* состояний не применяются.

4.4.2. Подсчёт фитнесс-функции

В данной работе использовалась следующая фитнесс-функция: $\frac{t.score}{t.score+e.score}$ (используемые обозначение раскрыты в разделе 4.3.2)

4.5. Основные особенности применения двухэтапного алгоритма

Для начала оценим размер пространства решений в случае применения двухэтапного генетического алгоритма. (В данном разделе, ниже использованы обозначения из раздела 3.4).

Для деревьев, отвечающих за логические переменные соотношение не изменилось: $(Z^E)^{V_c}$. Также не изменилось соотношение переходов: $N^{2^{V_c}}$. Однако для генерации состояние теперь достаточно лишь выбрать 4 дерева

из репозитория, в таком случае очевидно соотношение: S^4, где S — размер репозитория. Таким образом: $|\Omega_{treenew}| = (Z^E)^{V_c} S^4 N^{2^{V_c}}$

Очевидно что, $|\Omega_{treenew}| \ll |\Omega_{tree}|$.

Выводы по главе 4

В настоящей главе описан двухэтапный генетический алгоритм, посредством которого решается проблема уменьшения пространства решений, описанная в главах 2 и 3.

Суть данного алгоритма заключается в разбиении исходной задачи, построения управляющей системы для модели танка в игре Robocode, на две части:

- построение обработчиков вещественных переменных в отдельно взятых состояниях будущего автомата (первый этап)
- конструирование автомата из готовых состояний (второй этап).

Также по средством данного метода уменьшается время работы алгоритма, так как первый этап необходимо выполнить лишь один раз, а необходимое число поколений на втором этапе меньше чем в работе [5]. И, что более важно, уменьшается размер пространства решений.

Кроме того, в настоящей главе описана модификация островного генетического алгоритма, используемая для решения поставленной задачи.

Глава 5. Апробация предложенных подходов и сравнение результатов

В настоящем разделе описаны результаты соревнований моделей танков под управлением систем построенных с помощью методов, описанных в предыдущих разделах, с моделями танков входящих с стандартную поставку системы *Robocode*.

5.1. Настройки особей и генетического алгоритма

Перечислим настройки генетического алгоритма, используемые при генерации особей:

- число островов — 4;
- число особей на острове — 52;
- доля элитных особей — 0.15;
- вероятность мутации — 0.02;
- период миграции — 11;
- период большой мутации — 140;
- доля островов, уничтожаемых при большой мутации — 0.8;

Структура нейронной сети была приведена в разделе 3.1.1. Для деревьев разбора установлены следующие параметры:

- c_1 — 1;
- c_2 — 0.85.

Здесь использованы обозначения из раздела 3.2.1.

Для второго этапа генетического алгоритма использовались те же параметры самого алгоритма, а также следующие настройки для автомата:

- число состояний — 4;
- число логических переменных — 2.

Подсчёт фитнесс-функции проводился на 40 раундах, и размер поля составлял 800×600.

5.2. Модель танка, управляемая системой, построенной с помощью метода нейронных сетей

Результаты, демонстрируемые моделью танка, управляемой системой, построенной с помощью метода нейронных сетей, на первом этапе алгоритма, показали нецелесообразность проведения второго этапа.

5.2.1. Ход эволюции

Приведём график эволюционного процесса для первого этапа генетического алгоритма (см. рис. 5.1). В качестве противника используется модель танка — sample.Tracker, а в качестве фитнесс-функции используется: $\frac{t.score}{t.score+e.score}$ (использованы условные обозначения из раздела 4.3.2).

Как видно из данного графика, для данной задачи, метод нейронных сетей существенно проигрывает методу, описанному в работе [5].

5.2.2. Результаты

Приводить нейронную сеть, сконструированную в результате данного алгоритма не имеет смысла, по причине ее больших размеров (Напомним, что на первом этапе генерируется автомат, состоящий из одного состояния — одна нейронная сеть).

Результаты соревнования данной особи проиллюстрированы на рис. 5.2.

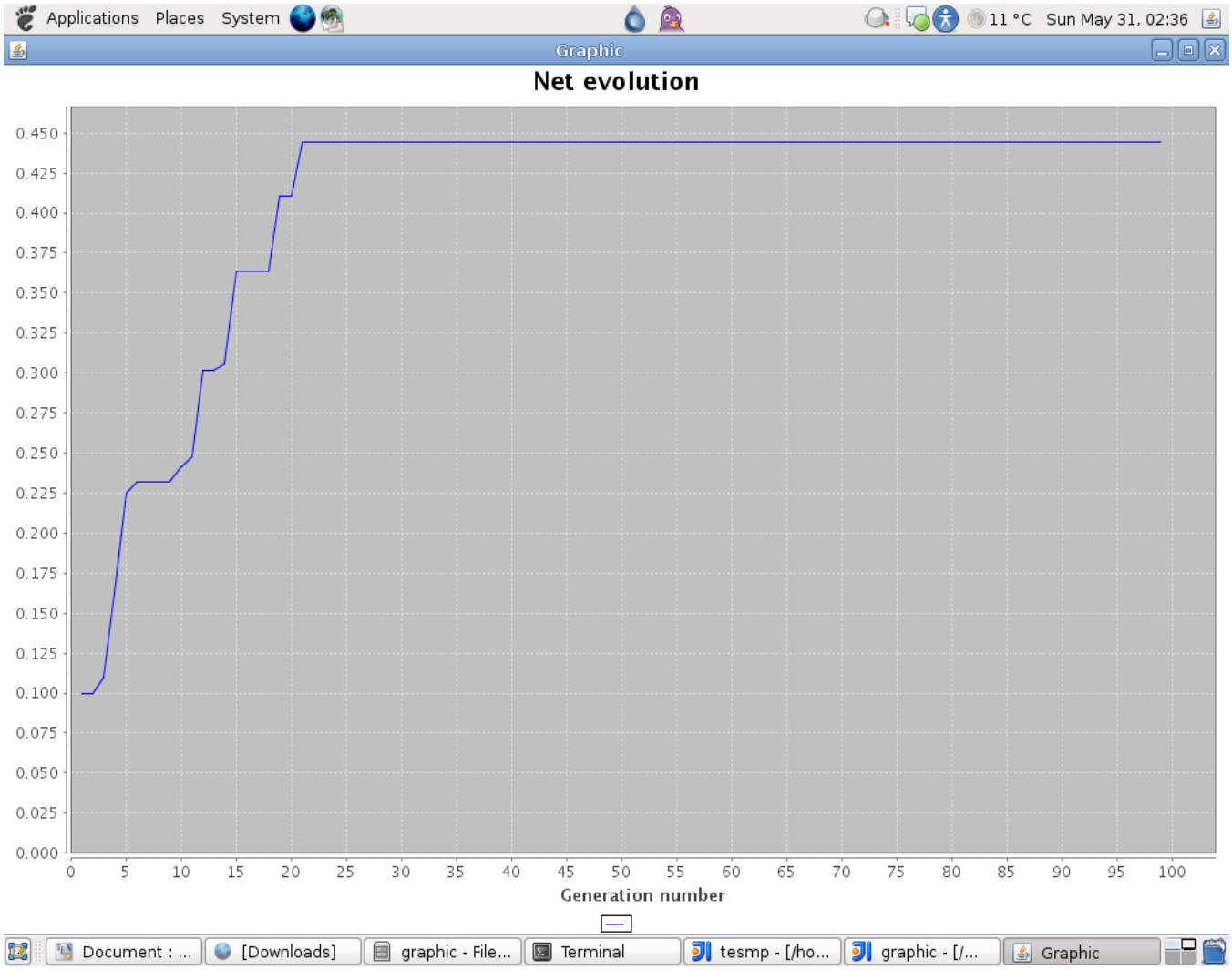

Рис. 5.1. График зависимости максимальной фитнесс-функции от поколения, для нейронных сетей

5.3. Модель танка, управляемая системой, построенной с помощью метода деревьев разбора

В данном разделе приведены результаты, демонстрируемые моделью танка, управляемой системой, построенной с помощью метода деревьев разбора.

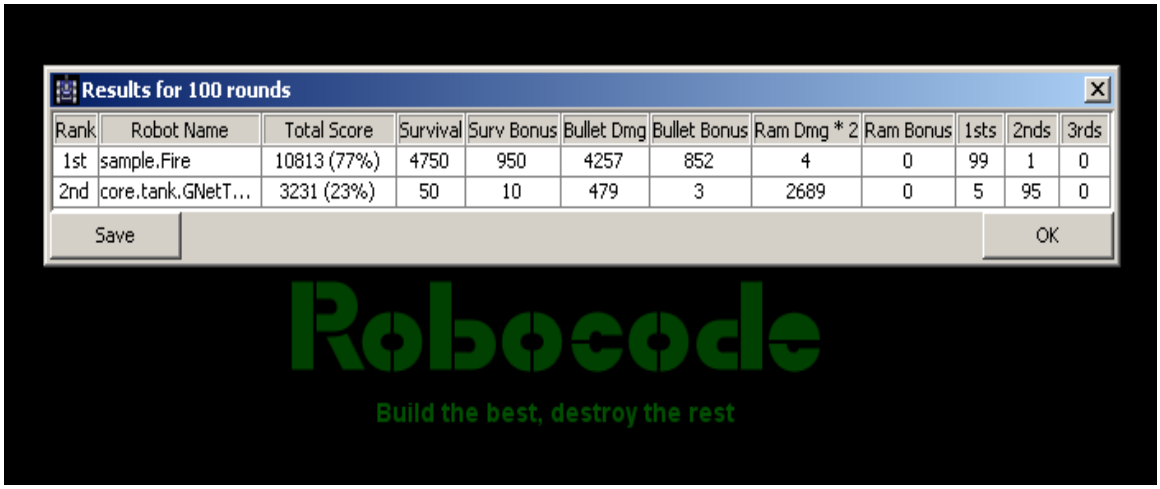

Рис. 5.2. Схема островного генетического алгоритма

5.3.1. Ход эволюции

Приведем пример графика эволюционного процесса для первого этапа генетического алгоритма (см. рис. 5.3). В качестве противника используется модель танка — sample.Fire, при этом у тестируемой модели танка отключена функция движения, в качестве фитнесс-функции используется: $\frac{(t.survival-n)n}{t.bulletDamage}$ (использованы условные обозначения из раздела 4.3.2).

На первом этапе проводились запуски против всех соперников, описанных в разделе на всех 5, для всех вариантов фитнесс-функции, описанных в разделе 4.3.2. Таким образом размер репозитория составил 304 дерева разбора для каждого из 4-ех типов.

Приведем пример графика эволюционного процесса для второго этапа генетического алгоритма (см. рис. 5.4). В качестве противника используется модель танка — sample.Tracker. $\frac{t.score}{t.score+e.score}$.

5.3.2. Результаты

Изобразим диаграмму переходов полученного автомата на рис. 5.5. Поясним условные обозначения. На ребрах диаграммы переходов находятся отметки в виде: $(+-)$. Данные отметки указывают условие перехода по данному ребру, а именно то, какое значение должно выбирать соответству-

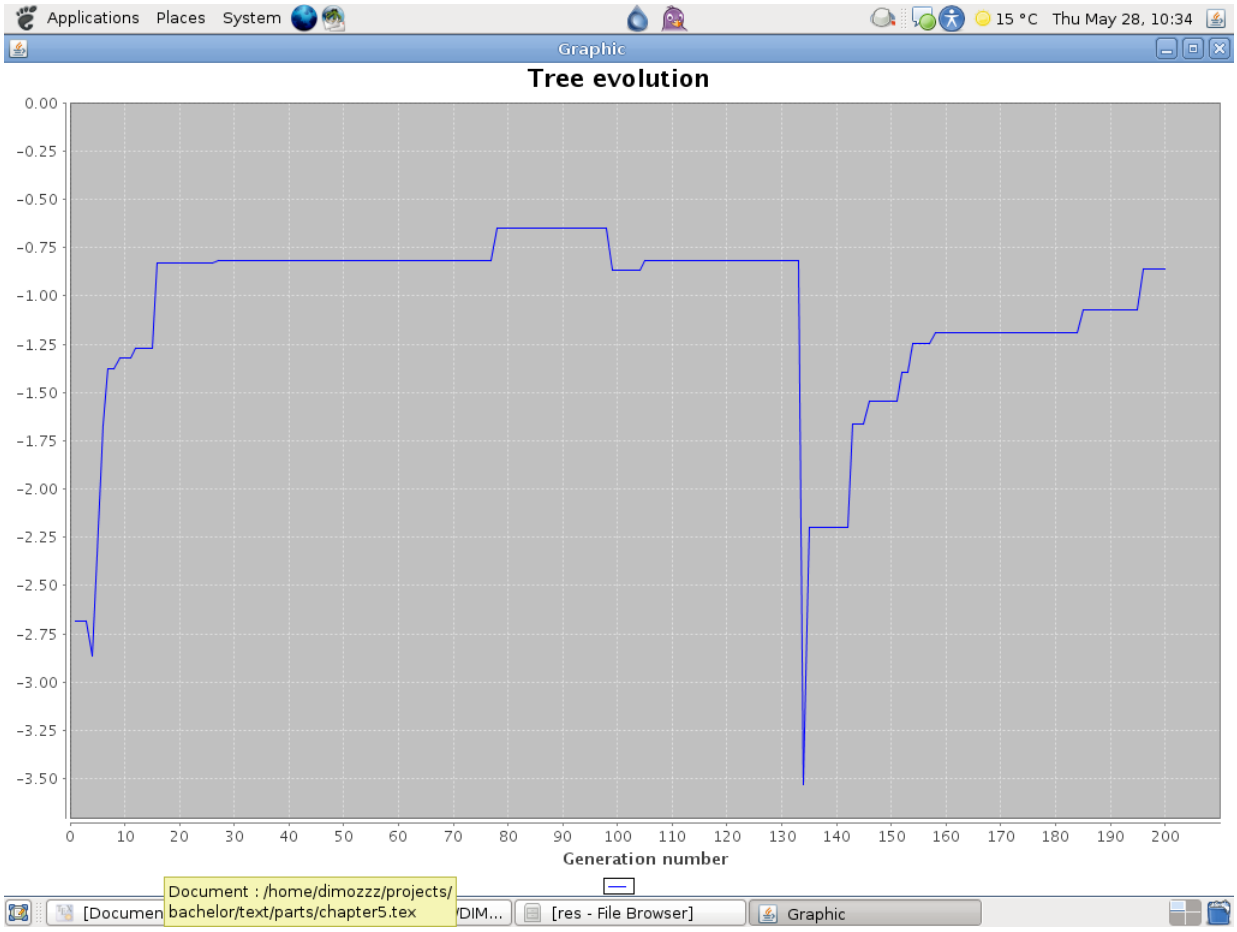

Рис. 5.3. График зависимости максимальной фитнесс-функции от номера поколения, для первого этапа построения конечного автомата при помощи деревьев разбора

ющее дерево разбора, отвечающее за булевы переменные, положительное или отрицательное.

Структуру состояний изобразить не представляется возможным из-за размеров деревьев разбора.

Результаты соревнования данной особи проиллюстрированы на рис. 5.6.

5.4. Сравнение результатов и выводы

Как видно из приведенных в данной главе графиков, а также из результатов, описанных в работе [5], методы по возрастанию результатов, получаемых с их помощью, можно упорядочить следующим образом.

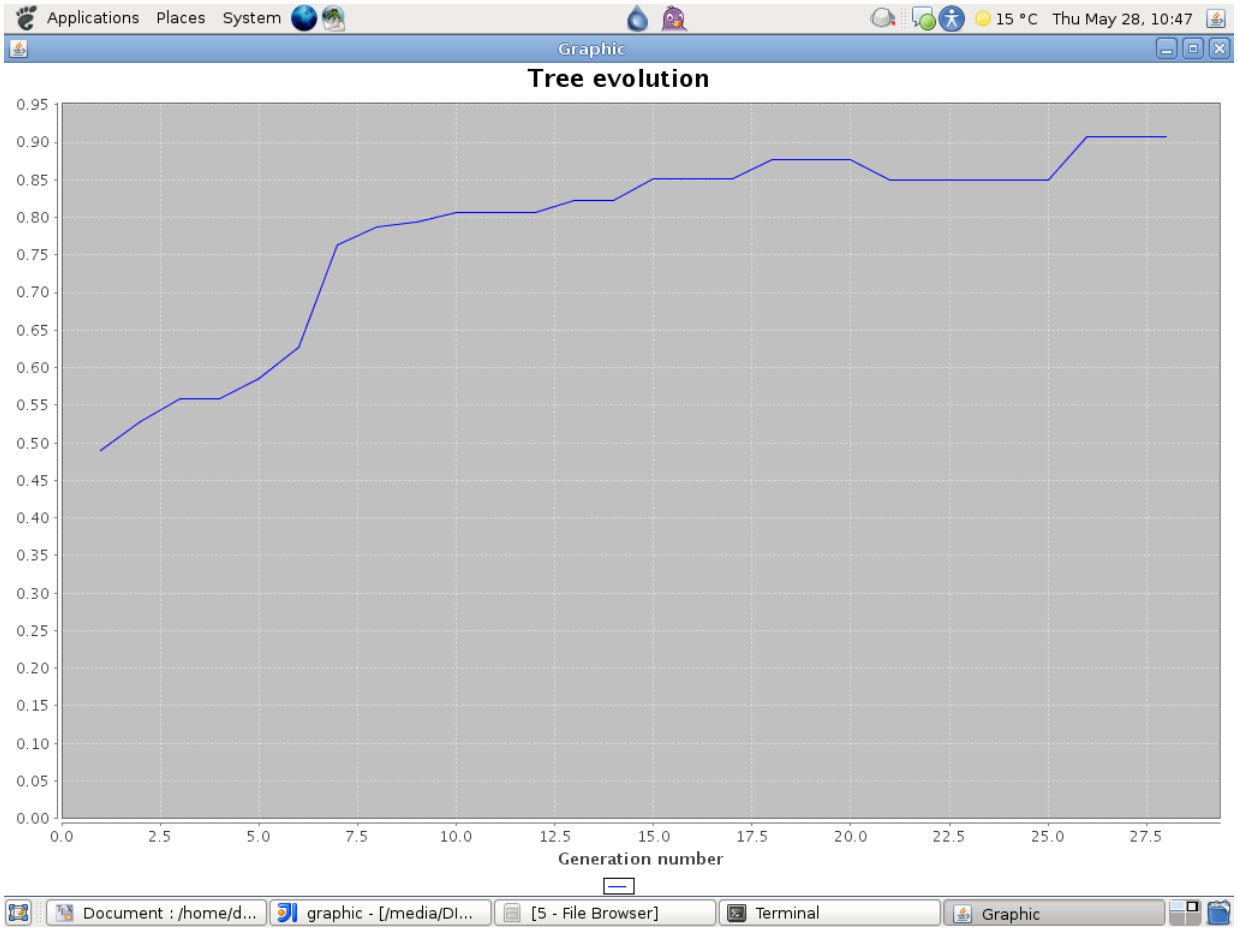

Рис. 5.4. График зависимости максимальной фитнесс-функции от номера поколения, для второго этапа построения конечного автомата при помощи деревьев разбора

1. Метод конечных автоматов с нейронными сетями в качестве обработчиков вещественных переменных.
2. Методы karva-деревьев (без автоматов), из работы [5], а также из настоящей работы.
3. Метод конечных автоматов с karva-деревьями в качестве обработчиков вещественных переменных из работы [5].
4. Метод конечных автоматов с деревьями разбора в качестве обработчиков вещественных переменных из настоящей работы.

В прочем разница в результатах, демонстрируемая методами с применением конечных автоматов и karva-деревьев, лежит в пределах погрешности измерений фитнесс-функции. Однако при использовании метода из

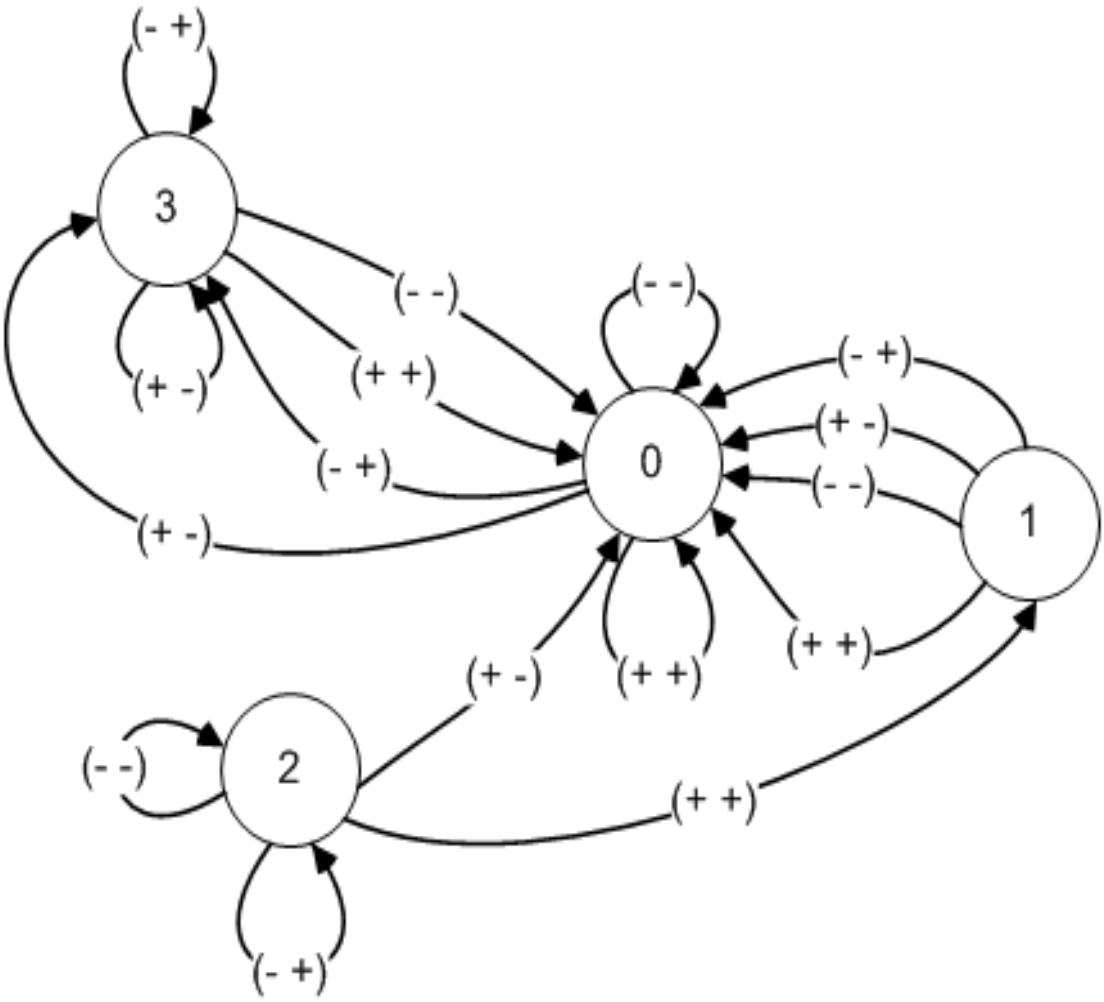

Рис. 5.5. Граф переходов автомата, полученного в результате работы второго этапа генетического алгоритма

настоящей работы (несмотря на вдвое увеличенное число раундов соревнований при подсчете фитнесс-функции), наблюдается ускорение процесса получения особи (на втором этапе) с заданной фитнесс-функцией в среднем в полтора раза (т.е. примерно 1.5—2 часа), по сравнению с работой [5].

Также данный метод позволяет достичь большей универсальности, так как на первом этапе генерируются различные *хорошие* стратегии поведения, то есть каждое состояние представляет собой уже законченную стратегию поведения, против конкретного поведения противника.

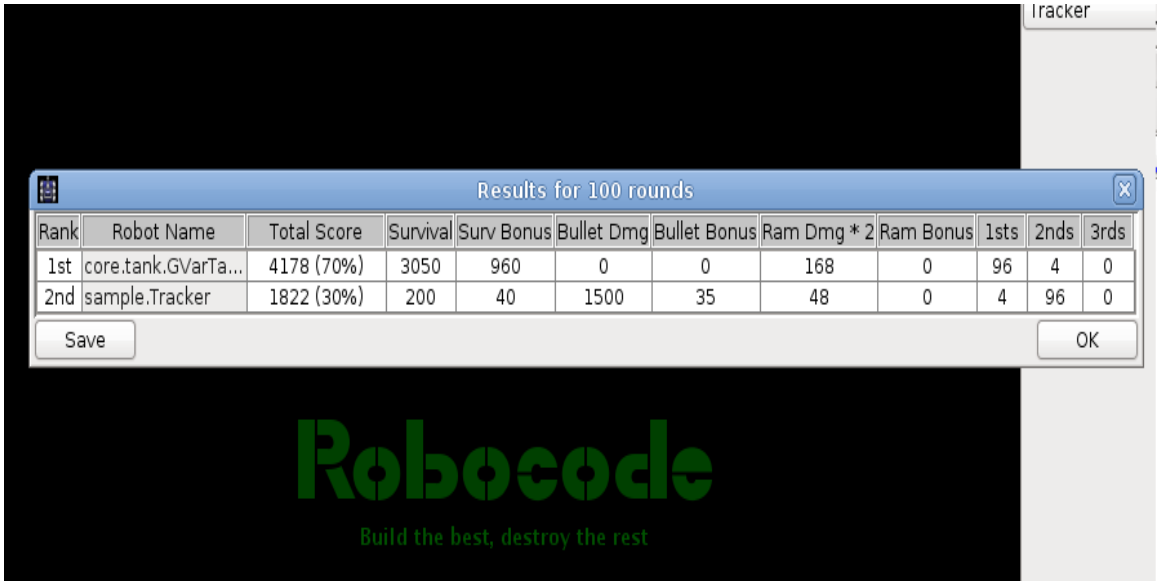

Рис. 5.6. Результат соревнования

Из недостатков предложенного метода стоит отметить длительное время проведения первого этапа алгоритма. На создание репозитория из 300 четверок деревьев потребовалось 8 суток.

Метод нейронных сетей в данной задаче демонстрирует не лучшие результаты, так как для обработки переменных требуется сеть большой глубины, что сильно увеличивает число нейронов и число ребер данной сети, а это в свою очередь сказывается на размере пространства решений, как указано в разделе 3.4. Данный метод применим в задачах, где выполнены следующие условия:

- зависимость между выходными значениями;
- требуется небольшая глубина нейронной сети (то есть от нейронная сеть достаточно небольшой вычислительной мощности).

Выводы по главе 5

В настоящей главе приведены результаты апробации методов из предыдущих глав. Приведены настройки островного генетического алгоритма.

Продемонстрированные результаты, позволяют говорить, что двухэтапный генетический алгоритм более эффективен, чем предыдущие методы решения задачи. Также демонстрируется эффективность совместного применения конечных автоматов и деревьев разбора.

В данной главе проведён анализ результатов. В том числе праведен анализ неудачи с методом совместного применения конечных автоматов и нейронных сетей.

Заключение

В настоящей работе предложен новый метод анализа вещественных переменных при использовании конечных автоматов — совместное применение конечных автоматов и нейронных сетей (отдельно отметим, что данный метод преследует другие цели, нежели метод, описанный в работе [13]). Разработаны генетические операторы для построения автоматов с нейронными сетями при помощи генетических алгоритмов.

Также в настоящей работе модифицирован метод с применением karva-деревьев из работы [5]. Для данного метода адаптированы генетические операторы, для *деревьев решений* из работы [12].

В данной работе задача построения управляющего автомата, для систем со сложным поведением, разбита на части:

- построение обработчиков вещественных переменных в одном отдельно взятом состоянии. При этом возможна более точная настройка поведения желаемого поведения от данного состояния;
- построение графа переходов, и вспомогательных конструкций (к примеру, дополнительные деревья разбора для генерации булевых переменных).

Также разбиение на части позволило улучшить качество решений, и сократить требуемые вычислительные ресурсы (для второго этапа), в случае когда предподсчет первого этапа выполнен заранее. К недостаткам метода можно отнести длительное время работы первой части алгоритма.

Для каждой части был реализованы модификация островного генетического алгоритма (двухэтапный генетический алгоритм).

Возможные направления дальнейшей работы:

- дальнейшее дробление задачи на части и разработка соответствующих методов генерации данный частей. Это целесообразно, так как

размер пространства решений для данной задачи, все ещё очень велико;

- сокращение пространства решений за счёт метода анализа главных компонент (к примеру использование *фильтра Хебба*) [14].

Благодарности:

Спасибо Юрию Бедному за идею исследования игры *Robocode*, а также Игорю Шендеровичу за моральную и интеллектуальную поддержку.

Источники

[1] J. Holland. *Sdaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. The University of Michigan Press, 1975.

[2] J. Koza. *Genetic programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press, MA: Cambridge, 1992.

[3] A. Aho, R. Sethi, and J. Ullman. *Compiler Design: Principles, Tools, and Techniques*. Наука, MA: Addison Wesley, 1986.

[4] А. Шалыто. *SWITCH-технология. Алгоритмизация и программирование задач логического управления*. Наука, СПб., 1998.

[5] Ю. Бедный. Применение генетических алгоритмов для генерации автоматов при построении модели максимального правдоподобия и в задачах управления.

[6] Б. Яминов. Генетические алгоритмы. 2005.

[7] Ф. Царев and А. Шалыто. Применение генетического программирования для генерации автомата в задаче об «умном муравье». *Сборник трудов IV-ой Международной научно-практической конференции*, 2:590–597, 2007.

[8] История игры robocode.

[9] M. Gade, M. Knudsen, and et al. Applying machine learning to robocode. 2003.

[10] J. Eisenstein. Evolving robot tank fighters. *Technical Report AIM-2003-023, AI Lab, MIT.*, 2003.

[11] Т. Кормен, Ч. Лейзерсон, and Р. Ривест. *Алгоритмы: построение и анализ*. МЦНМО: БИНОМ, М., 2004.

[12] В. Данилов. Технология генетического программирования для генерации автоматов управления системами со сложным поведением.

[13] Ф. Царев. Разработка метода совместного применения генетического программирования и конечных автоматов.

[14] С. Хайкин. *Нейронные сети. Полный курс*. Вильямс, М., 2006.