- объектно-ориентированная модель SCOOP (Simple Concurrent Object-Oriented Programming) [29].

Параллельная автоматная декомпозиция, которая была рассмотрена в параграфе 2.1, порождает еще одну модель параллельных вычислений: параллельно работающие автоматы, взаимодействие которых может осуществляться путем обмена номерами состояний, событиями или сообщениями [32, 99]. В чем же преимущество этой модели перед перечисленными выше?

В автоматном программировании управляющие автоматы и объекты управления всегда выделяются в явном виде. Если объекты управления двух автоматов не пересекаются (или область их пересечения невелика), то такие автоматы с большой вероятностью смогут эффективно работать параллельно. В результате при применении автоматного подхода параллелизм часто возникает естественно, без дополнительных преобразований.

В качестве примера рассмотрим задачу декодирования файлов формата GIF (сравнение автоматного и традиционного подходов к решению этой задачи подробно обсуждается в работе [100]). Задача состоит в том, чтобы преобразовать GIF-файл в матрицу цветов пикселей, составляющих изображение.

При декодировании таких файлов сначала декодируется заголовок изображения, а затем оставшаяся часть файла разбирается на блоки. Каждый блок, в свою очередь, состоит из заголовка и тела, представляющего собой LZW-код искомой последовательности цветов пикселей.

В работе [100] описано решение этой задачи с использованием программирования с явным выделением состояний. При этом автоматная декомпозиция приводит к выделению трех автоматов: первый отвечает за декодирование заголовка, второй разбивает основную часть файла на блоки, а третий расшифровывает LZW-коды. В результате такой декомпозиции оказывается, что второй и третий автоматы могут работать параллельно: каждый раз, когда второй автомат выделяет тело очередного блока, ему необязательно дожидаться, пока третий автомат преобразует это тело в последовательность цветов пикселей. Ему достаточно отправить выделенный LZW-код третьему автомату в качестве сообщения или передать его через очередь с параллельным доступом.

4.4. Автоматы и генетическое программирование

При использовании автоматного подхода для реализации сложного поведения построение управляющего автомата во многих случаях является шагом, наиболее сложным для программиста и порождающим наибольшее число ошибок. Кроме того, существуют задачи, для которых построить автомат вручную практически невозможно. В других случаях построенный автомат бывает неоптимальным. Возможное решение всех этих проблем — перепоручить построение управляющего автомата компьютеру.

Одним из наиболее эффективных методов автоматизированного конструирования программ является *генетическое программирование* [101]. Основная идея

генетического программирования состоит в построении программ путем применения *генетических алгоритмов* [102] к некоторой *модели вычисления*. При этом разработчику программы остается лишь задать *оценочную функцию*, определяющую для каждого результата вычисления в выбранной модели численное значение, называемое его *приспособленностью*.

Генетические алгоритмы — это метод оптимизации, основанный на концепциях естественной эволюции. Он оперирует *поколениями*, состоящими из *особей*. Первое поколение заполняется особями, сгенерированными случайным образом. Каждое последующее поколение формируется из особей предыдущего поколения в зависимости от их приспособленности путем перенесения их без изменений либо применения к ним с некоторыми вероятностями генетических операторов: *мутации* и *скрещивания*. Результатом оптимизации считается особь с максимальной оценкой из последнего поколения.

Для того чтобы применение генетического программирования для оптимизации автоматизированных объектов управления стало возможным, необходимо разработать представление автомата в виде особи, определить операции мутации и скрещивания автоматов и подобрать параметры генетического алгоритма, подходящие для автоматов. После этого для каждой задачи остается только реализовать объект управления и определить функцию приспособленности. Значение приспособленности автоматизированного объекта управления целесообразно задавать как функцию вычислительного состояния объекта управления после заданного числа тактов работы автомата.

Обычно в качестве моделей вычислений в генетическом программировании используют деревья, графы, команды процессора — «низкоуровневые» модели, имеющие ограниченный набор элементарных операций (таких, как, например, запись и чтение ячеек памяти, арифметические операции, вызовы подпрограмм и т. д.). Достоинство низкоуровневых моделей состоит в их универсальности: с их помощью можно построить любую программу целиком, единообразно, вне зависимости от специфики решаемой задачи. Однако такие модели обладают и серьезными недостатками. Во-первых, построенная программа из-за отсутствия высокоуровневой структуры редко бывает понятна человеку, что исключает возможность ее дальнейшей модификации вручную, обобщения полученного решения и т. п. Во-вторых, из-за того, что пространство допустимых программ в этом случае очень велико, генетическая оптимизация требует длительного времени и может использоваться только для небольших задач.

С другой стороны, если использовать в качестве модели вычислений автоматизированный объект, причем реализацию объекта управления производить вручную, пространство поиска значительно сокращается и генетический подход становится реалистичным. При этом используется разумное разделение труда: компьютер выполняет ту часть работы, которая лучше всего получается у него, а человек — то, что под силу только ему.

Первые работы в области генерации конечных автоматов с помощью генетических алгоритмов были выполнены около 40 лет назад [103]. Однако в этой работе автоматы рассматривались как одна из моделей человеческого интеллекта. Эта

модель оказалась непродуктивной (по крайней мере, при существовавшем в то время уровне развития вычислительной техники), и исследования в этом направлении почти прекратились. Они возобновились только через 20 лет [104–106]. При этом круг рассматриваемых задач расширился. В частности, стали изучаться задачи управления, однако в большинстве этих задач управляющие автоматы имели только одну входную переменную (см. пункт 4.4.1). С развитием автоматного программирования стала актуальной задача генерации управляющих автоматов с большим числом входных переменных (см. пункт 4.4.2).

4.4.1. Задача об «Умном муравье»

В качестве примера, в котором автоматическое построение логики сущности со сложным поведением имеет решающее значение, рассмотрим одну из классических задач из области совместного использования генетических алгоритмов и конечных автоматов — задачу об «Умном муравье» [106].

Муравей «живет» на поверхности тора размером 32 на 32 клетки (рис. 4.4). В нескольких клетках (обозначены на рис. 4.4 черным цветом) находится еда. Она расположена вдоль некоторой ломаной, но не во всех ее клетках. Клетки ломаной, в которых нет еды, обозначены серым цветом. Белые клетки не содержат еду и не принадлежат ломаной. Всего на поле 89 клеток с едой.

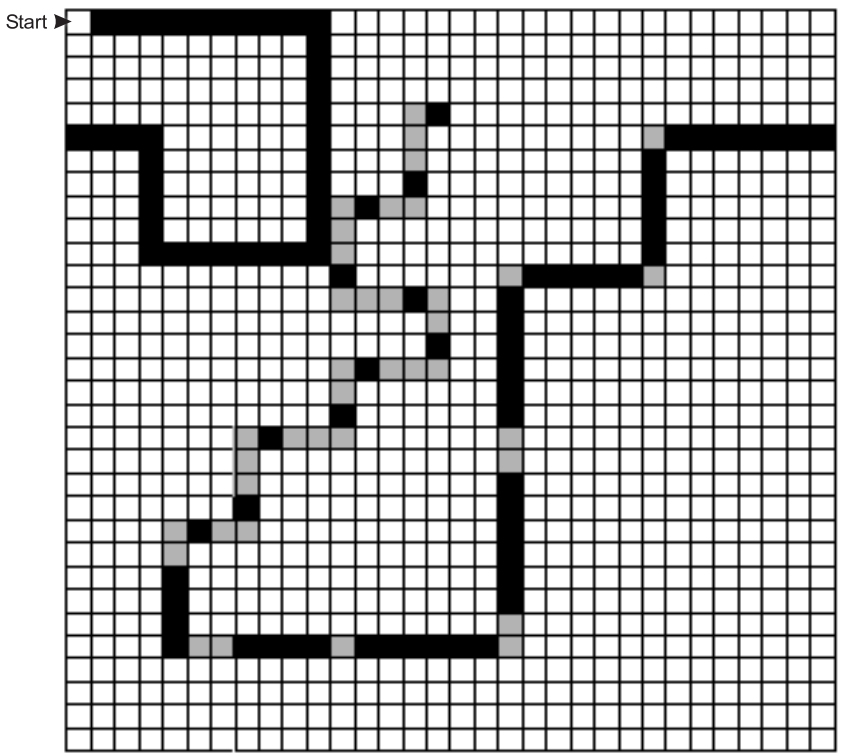

Рис. 4.4. Поле в задаче об «Умном муравье»

В клетке, помеченной меткой *Start*, в начале игры находится муравей. Он занимает одну клетку и смотрит в одном из четырех направлений (север, юг, запад, восток). В начале игры муравей смотрит на восток.

Муравей умеет определять, находится ли непосредственно перед ним еда. За один игровой ход муравей может совершить одно из четырех действий:

- сделать шаг вперед, съедая еду, если она там находится;
- повернуть налево;
- повернуть направо;
- ничего не делать.

Съеденная муравьем еда не восполняется, муравей жив на протяжении всей игры, еда не является необходимым ресурсом для его жизни. Ломаная не случайна, а строго фиксирована. Муравей может ходить по любым клеткам поля.

Игра длится 200 ходов, на каждом из которых муравей совершает одно из четырех действий. По истечении 200 ходов подсчитывается количество еды, съеденной муравьем. Это значение и есть результат игры.

Цель игры — создать муравья, который за 200 ходов съест как можно больше еды (желательно, все 89 единиц).

Один из способов описания поведения муравья — автомат Мили, у которого имеется одна входная переменная (находится ли еда перед муравьем), а множество выходных воздействий состоит из четырех упомянутых выше элементов. Схема связей этого автомата приведена на рис. 4.5.

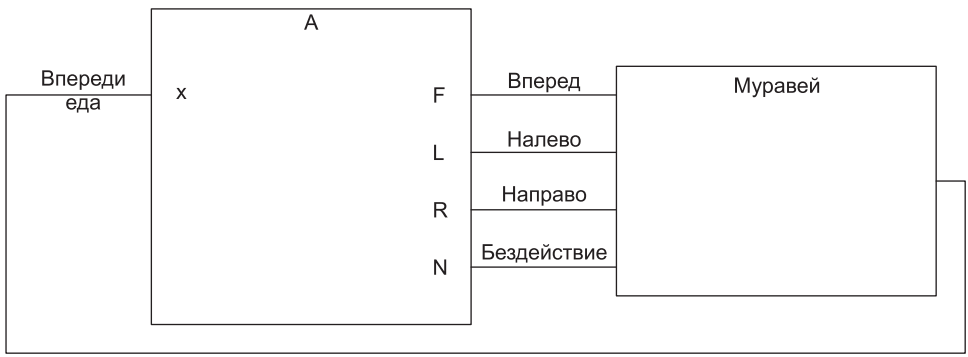

Рис. 4.5. Схема связей автомата «Умного муравья»

Автомат, решающий описанную задачу, крайне трудно построить вручную. Например, эвристически построенный автомат Мили с пятью состояниями из работы [104], граф переходов которого изображен на рис. 4.6, задачу не решает — он описывает поведение муравья, который съедает всего 81 единицу еды за 200 ходов, а всю еду — только за 314 ходов.

Эта задача была решена с применением генетических алгоритмов. При этом был построен автомат с восьмью состояниями [106]. Ниже описывается алгоритм, предложенный в работе [107], который позволил построить для решения рассматриваемой задачи автомат с семью состояниями.

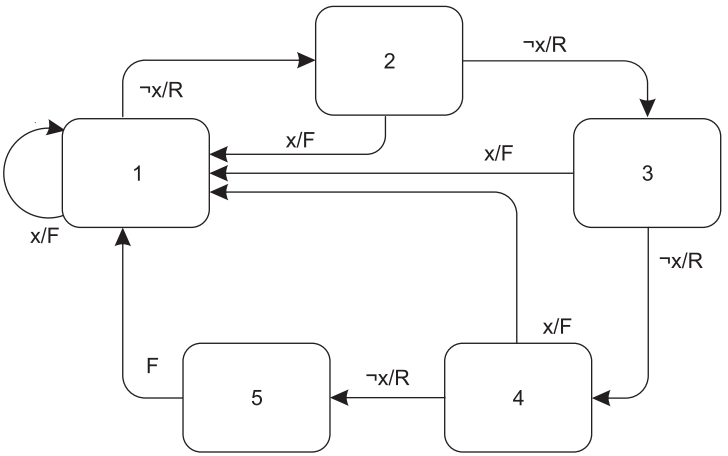

Рис. 4.6. Простой автомат «Умного муравья»

Пусть представление автомата в виде особи состоит из номера начального состояния и описания каждого состояния. Описание (в терминах генетического программирования — *хромосома*) состояния содержит описания двух переходов, соответствующих двум значениям входной переменной. Описание перехода состоит из номера состояния, в которое он ведет, и действия, выполняемого на этом переходе. Начальное поколение содержит фиксированное количество случайно сгенерированных автоматов. Все автоматы в поколении имеют одинаковое, наперед заданное число состояний.

Теперь опишем генетические операторы мутации и скрещивания в терминах предложенного представления автоматов.

При мутации случайно выбирается один из четырех равновероятных вариантов:

- **изменение начального состояния** — в этом случае новое начальное состояние выбирается случайно и равновероятно;
- **изменение действия на переходе** — случайно и равновероятно выбирается переход, и действие на нем изменяется на случайное;
- **изменение состояния, в которое ведет переход,** — случайно и равновероятно выбирается переход, после этого целевое состояние перехода изменяется на случайно выбранное;
- **изменение условия на переходе** — случайно и равновероятно выбирается состояние, после этого переходы из этого состояния, соответствующие различным значениям входной переменной, меняются местами.

Оператор скрещивания принимает на вход две родительские особи, а результатом также являются две особи. При этом первый ребенок может наследовать начальное состояние от первого родителя, а второй — от второго, или наоборот. Что касается переходов из заданного состояния, каждый ребенок может наследовать как переход по значению входной переменной «истина», так и переход по значению

«ложь» равновероятно от обоих родителей, но оба ребенка не могут наследовать один и тот же переход. Схема скрещивания изображена на рис. 4.7.

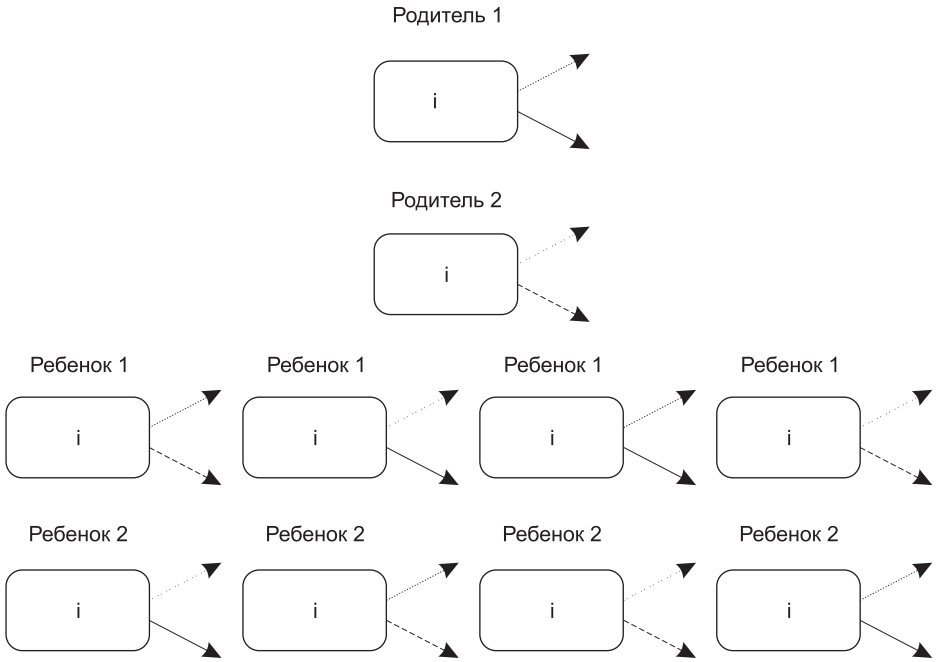

Рис. 4.7. Простая схема скрещивания состояний автомата

Перейдем к функции приспособленности. Определим ее следующим образом:

$$A + \frac{200 - T}{200},$$

где A — суммарное количество еды, съеденной муравьем за игру, T — номер хода, на котором муравей съедает последнюю единицу еды.

С использованием приведенных выше генетических операторов и функции приспособленности удалось построить автомат с семью состояниями, который позволяет муравью съесть все 89 единиц еды за 190 ходов. Граф переходов этого автомата приведен на рис. 4.8. Для того чтобы найти этот автомат, потребовалось перебрать 130 000 поколений и 160 миллионов особей, что немного по сравнению с общим числом автоматов с семью состояниями, одной входной и четырьмя выходными переменными.

Еще один автомат с семью состояниями был построен после перебора 250 миллионов особей, что также представляет незначительный процент от общего количества автоматов рассматриваемого вида.

В работе [108] приведено решение задачи об «Умном муравье» с помощью автомата Мура и системы из двух автоматов Мили.

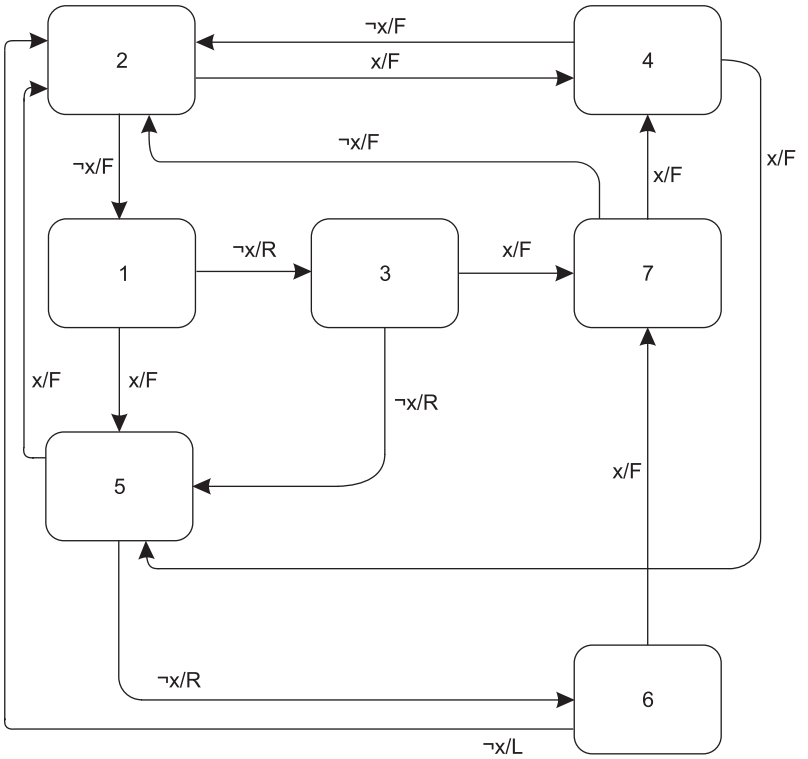

Рис. 4.8. Автомат «Умного муравья» построенный методом генетического программирования

Еще одна интересная задача, решаемая простым автоматом, который генерируется с помощью генетического алгоритма, описана в работе [109].

4.4.2. Методы генерации автоматов с большим числом входных переменных

В задаче об «Умном муравье» представление автомата в виде особи и реализация генетических операторов не составляли проблемы. Функции переходов и выходов для *каждого состояния* автомата были представлены в виде *полной таблицы* переходов и выходов: в этой таблице каждому возможному входному воздействию сопоставляется целевое состояние и выходное воздействие (рис. 4.9).

При использовании полных таблиц размер представления автомата растет экспоненциально с увеличением числа входных переменных. Это не позволяет использовать такие таблицы для автоматов с большим числом входных переменных. К счастью, реальные автоматы обладают свойством, отмеченным в параграфе 2.3: обычно только небольшая часть входных переменных является *значимой* в каждом состоянии. Исходя из этого свойства, можно предложить различные способы упрощения представления автоматов.

x_1	x_2	s	z_1	z_2	z_3
0	0	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	1	2	1	1	1

Рис. 4.9. Представление функций переходов и выходов для одного состояния автомата в виде полной таблицы

В настоящее время известно два способа компактного представления автоматов с большим количеством входных переменных: метод *сокращенных таблиц* [110] и метод *деревьев решений* [111].

Идея метода сокращенных таблиц состоит в том, что число значимых входных переменных в каждом состоянии ограничивается константой. При этом представление функции переходов и выходов для состояния автомата содержит описание множества значимых входных переменных и собственно сокращенную таблицу, в которой каждой комбинации значений выбранных переменных сопоставляется целевое состояние и выходное воздействие (рис. 4.10).

x_1	x_2	x_3	x_4	x_5	x_6
0	1	0	1	0	0

x_2	x_4	s	z_0	z_1	z_2
0	0	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	1	2	1	1	1

Рис. 4.10. Представление функций переходов и выходов для одного состояния с помощью сокращенной таблицы

Опыт показывает, что даже в системах со сложным поведением, которые содержат несколько десятков входных переменных, число значимых переменных в каждом состоянии обычно не превышает нескольких единиц. Поэтому для сложных задач размер сокращенной таблицы остается небольшим.

Операции скрещивания и мутации для сокращенных таблиц несколько сложнее, чем для полных. При мутации измениться может не только сама таблица, но и множество значимых переменных. При этом каждая из таких переменных с некоторой вероятностью заменяется другой, которая не принадлежит множеству.

При скрещивании сокращенных таблиц родительские хромосомы состояний могут иметь различные множества значимых переменных. Поэтому сначала необходимо выбрать, какие из этих предикатов будут значимы для хромосом детей. В настоящее время используется вариант, при котором переменные, значимые для обоих родителей, наследуются обоими детьми, а каждая из тех переменных, которые были значимы лишь для одной родительской особи, равновероятно достаются любому из двух детей. После этого скрещиваются сами сокращенные таблицы переходов. При этом на значения каждой строки таблицы ребенка влияют значения нескольких строк родительских таблиц. Конкретное значение, помещаемое в ячейку таблицы ребенка, определяется «голосованием» всех влияющих на нее ячеек таблиц родителей.

Для реализации метода сокращенных таблиц была разработана библиотека *AutoGen* [112]. Эта библиотека была успешно использована для генерации автомата верхнего уровня управления самолетом [113].

Перейдем ко второму методу компактного представления автоматов — методу деревьев решений. Дерево решений является удобным способом задания дискретной функции, зависящей от конечного количества дискретных переменных. Оно представляет собой помеченное дерево, метки в котором расставлены по следующему правилу:

- внутренние узлы помечены символами переменных;
- ребра — значениями переменных;
- листья — значениями искомой функции.

Для определения значения функции по значениям переменных необходимо спуститься от корня до листа и сформировать значение, которым помечен полученный лист. При этом из вершины, помеченной переменной x, переход производится по тому ребру, значение на котором совпадает со значением переменной x.

Пример дерева решений изображен на рис. 4.11. Это дерево реализует булеву функцию $\neg x_1 \neg x_2 \vee x_1 x_3$. Сокращение размера дерева по сравнению с полной таблицей истинности булевой функции достигается за счет использования ее специфики: если первая переменная ложна, то не играет роли значение третьей переменной, а если истинна — то второй.

Управляющий автомат можно представить в виде упорядоченного набора деревьев решений, каждое из которых описывает множество переходов из одного конкретного состояния и содержит в листьях не логические значения, а пары <целевое состояние, выходное воздействие>.

Определим генетические операции над деревьями решений. При мутации дерева выбирается случайный узел. После этого поддерево с корнем в этом узле изменяется на случайно сгенерированное. При скрещивании деревьев в каждой из родительских особей случайно выбирается по одному узлу. После этого поддерево с корнем в выбранном узле из первого родителя заменяется поддеревом с корнем

в выбранном узле из второго родителя. Результатом этой операции скрещивания, в отличие от предыдущих вариантов, является одна новая особь.

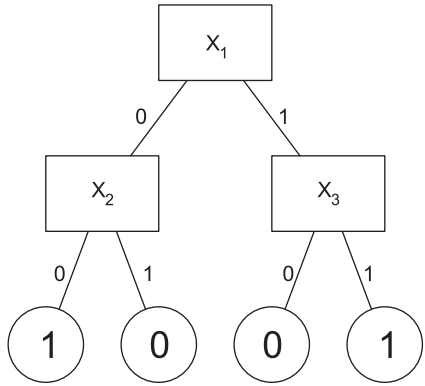

Рис. 4.11. Пример дерева решений

Этот метод был успешно использован при создании автоматов, управляющих беспилотными летательными объектами [114]. В работе [115] описано решение этой же задачи, в котором компоненты объекта управления не реализуются вручную, а представлены нейронными сетями и оптимизируются вместе с управляющим автоматом.

Кроме системы управления беспилотными летательными объектами с помощью генетических алгоритмов были созданы системы управления танком для игры Robocode [116] и моделью вертолета [117].

В работе [118] описано инструментальное средство для генерации автоматов с использованием генетических алгоритмов. В частности, это средство позволяет сократить время генерации автоматов за счет использования распределенных вычислений.

Отметим также работу В. Каширина (http://is.ifmo.ru/papers/kashirin/), посвященную оптимизации функций приспособленности.

Результаты работ по генерации автоматов для систем со сложным поведением на основе генетического программирования, полученные при участии авторов этой книги в ходе выполнения государственного контракта, приведены по адресу http://is.ifmo.ru/genalg/.

Для обучения генетическому программированию применительно к генерации управляющих автоматов под руководством одного из авторов книги создана виртуальная лаборатория, размещенная по адресу http://svn2.assembla.com/svn/not_instrumental_tool.

В заключение раздела отметим, что при использовании генетического программирования для генерации автоматов, применяемых в UniMod-программах, обеспечивается высокий уровень автоматизации построения этого класса программ, так как автоматически генерируются не только автоматы, но и программный код по ним. Эксперименты [75] показали, что для рассматриваемого класса программ только 20–30% программного кода пишется вручную.