Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Факультет информационных технологий и программирования
Кафедра компьютерных технологий

В. Р. Данилов

Методы представления функции переходов при генерации автоматов управления на основе генетического программирования

Научный руководитель: А. А. Шалыто, докт. техн. наук, профессор

Санкт-Петербург
2009

Оглавление

Введение

Генетическое программирование — метод автоматической генерации программ на основе эволюционных алгоритмов [1], использующий представление программ на высоком уровне абстракции. В настоящее время интерес к этой области искусственного интеллекта возрастает в связи с увеличивающимися вычислительными мощностями. В ряде задач с помощью генетического программирования были получены решения, превосходящие разработанные людьми. Отметим, что эффективность метода генетического программирования напрямую связана с используемым представлением программ.

Автоматное программирование [2] — парадигма программирования, использующая представление программ в виде системы автоматизированных объектов управления. При этом поведение системы описывается с помощью множества управляющих состояний, поведение в каждом из которых является относительно простым. Связь управляющих состояний с действиями описывается с помощью автомата управления. Целесообразно рассмотреть вопрос о применении автоматных моделей для представления программ в генетическом программировании. Из литературы известны способы генерации автоматов средствами генетического программирования, однако эти работы в основном описывают построение автоматов-распознавателей либо автоматов-преобразователей. Как правило, такие автоматы имеют сравнительно небольшое число входных переменных. В отличие от них, автоматы управления имеют значительно большее число входных переменных и характеризуются сложностью функции переходов из каждого состояния. Поэтому традиционные методы, основанные на представлении функции переходов автоматов полными таблицами, оказываются неприменимыми.

В настоящей работе предлагаются два новых метода эффективного представления функции переходов автоматов управления с булевыми входными переменными, основанных на деревьях решений и линейных бинарных графах.

Глава 1.
Применение генетического программирования для генерации автоматов

В этой главе определяются необходимые понятия и проводится краткий обзор существующих методов генетического программирования для генерации автоматов.

1.1. Генетическое программирование

Генетическое программирование [3] — метод автоматического конструирования программ. Данный метод является разновидностью эволюционных алгоритмов [1], особями которых являются компьютерные программы. Основной идеей метода 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 окружающей среде, чем предыдущее.

Общий алгоритм генетического программирования представлен на рис. 1.

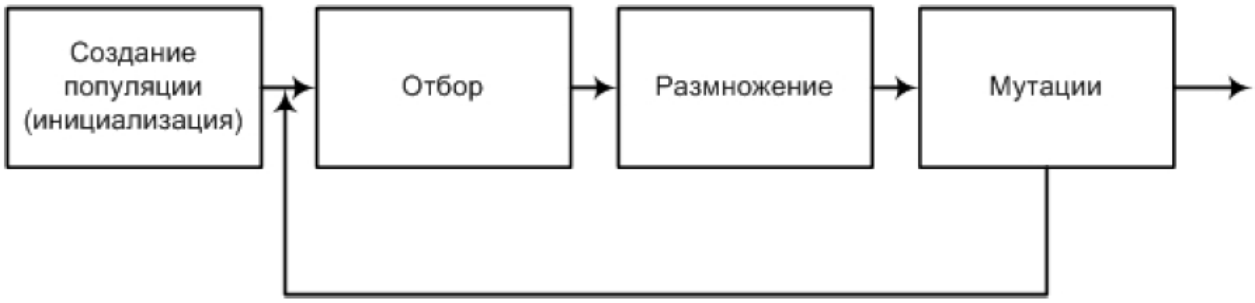

Рис. 1. Общая схема алгоритма генетического программирования

Алгоритм 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;

1.2. Конечные детерминированные автоматы

1.2.1. Определение конечного детерминированного автомата

Конечный детерминированный автомат — это математическая абстракция, позволяющая описывать пути изменения состояний объекта в зависимости от текущего состояния и входных данных. Ограничением является то, что множество возможных состояний автомата конечно. Конечный детерминированный автомат может быть описан пятеркой $\{Q, S, F, \Sigma, \delta\}$, где:

- Q — конечное множество состояний;
- $S \in Q$ — начальное состояние;
- $F \subset Q$ — множество допускающих состояний;
- Σ — входной алфавит;
- $\delta : Q \times \Sigma \to Q$ — функция переходов.

Работа конечного автомата может рассматриваться как последовательность тактов. За каждый такт происходит считывание очередного символа $a \in \Sigma$ из входной строки. После этого автомат переходит из состояния q в состояние $\delta(q, a)$. Первоначально автомат находится в состоянии S. Автомат допускает цепочку $w \in \Sigma^*$ тогда и только тогда, когда после считывания w он оказывается в одном из допускающих состояний.

Будем говорить, что автомат A допускает язык L над алфавитом Σ, если $\forall w \in \Sigma^*$, $w \in L$ тогда и только тогда, когда A допускает w.

1.2.2. Конечный автомат Мили

В ряде задач удобно описывать поведение объекта в виде конечного автомата Мили [2]. Такой конечный автомат может быть представлен шестеркой $\{Q, S, X, Y, \delta, \gamma\}$, где:

- Q — конечное множество состояний;
- $S \in Q$ — начальное состояние;
- X — конечное множество входных воздействий;
- Y — конечное множество выходных воздействий;
- $\delta : Q \times X \to Q$ — функция переходов;
- $\lambda : Q \times X \to Y$ — функция выхода.

При этом элементы множеств Q, X и Y связаны в абстрактном времени следующими отношениями:

$$\begin{cases} Q(t+1) = \delta(Q(t), X(t)); \\ \quad Y(t) = \lambda(Q(t), X(t)). \end{cases}$$

В рамках парадигмы автоматного программирования [2] конечный автомат Мили может быть обобщен за счет понятия входных переменных – 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 входных переменных. Таким образом, если множество входных переменных обозначить за P, то указанные функции примут следующий вид:

$$\begin{cases} \delta : Q \times X \times 2^P \to Q; \\ \lambda : Q \times X \times 2^P \to Q. \end{cases}$$

Заметим, что от входных переменных можно избавиться, увеличив набор входных воздействий. Однако такой способ усложняет логику системы для восприятия человеком. Кроме того, при этом множество входных воздействий растет экспоненциально от числа входных переменных.

1.3. Методы представления функции переходов автомата

Для генерации автоматов средствами генетического программирования необходимо определить представление автоматов в виде хромосомы и генетические операции над используемым представлением. Для того, чтобы задать

автомат необходимо определить функции переходов и выходов. Рассмотрим основные методы представления функции переходов. Отметим, что функции выходов могут быть заданы аналогичным образом.

1.3.1. Битовые строки

Проведем обзор применения представления функций переходов автомата с помощью битовых строк на основе работ [4, 5]. Опишем известный алгоритм кодирования автомата в строку:

1. Фиксируется число состояний автомата.
2. Логика работы представляется в виде таблицы, где для каждого состояния и каждого входного воздействия записывается пара *(Новое Состояние, Действие)*.
3. Таблица записывается в виде строки.

Продемонстрируем работу алгоритма на примере автомата, приведенного на рис. 2. Входными воздействиям автомата являются F и N, выходными воздействиями — L, R, M. Таблица переходов приведена в табл. 1.

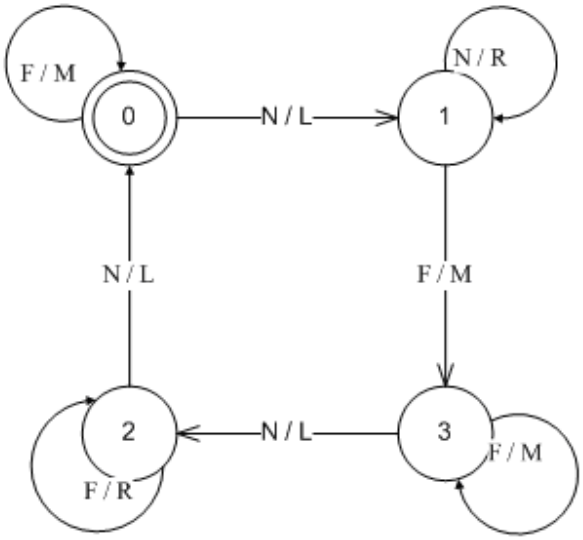

Рис. 2. Пример автомата

Теперь, занумеровав некоторым образом входные и выходные воздействия, можно записать элементы этой таблицы в строку, выписав двойки *(Новое Состояние, Действие)*, соответствующие переходам. Для этого переходы упорядочиваются в порядке возрастания начального состояния, а в случае равенства

Таблица 1. Таблица переходов 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

— по номеру входного воздействия. Такая техника налагает следующее ограничение: для того, чтобы строке всегда соответствовал корректный автомат необходимо, чтобы число его состояний и выходных воздействий было степенью двух. Как правило, это ведет к добавлению фиктивных выходных воздействий автомата.

Определим генетические операции над представлением функций переходов в виде битовых строк. Операция скрещивания выполняется с помощью *кроссовера* над соответствующими строками. Кроссовер 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.

В *методе одноточечного кроссовера* используется маска

$$m_i = \begin{cases} 0, i < k; \\ 1, i \geq k, \end{cases}$$

где k – случайное число от 1 до n.

В *методе двухточечного кроссовера* применяется маска

$$m_i = \begin{cases} 0, i < k \text{ или } i > l; \\ 1, k \leq i \leq l, \end{cases}$$

где k и l – случайные числа от 1 до n, $k \leqslant l$.

В *методе однородного кроссовера* маска является случайной с равномерным распределением.

Опишем используемый метод мутации — выбирается случайная позиция в строке, после этого символ в этой позиции заменяется на случайный.

1.3.2. Полные таблицы переходов

Рассматриваемый метод, описанный в работе [6], использует представление функции переходов напрямую в виде таблиц переходов, где для каждого состояния и каждого их входных воздействий хранится набор *(Новое Состояние, Флаги действий)*. При переходе вызываются все действия, для которых выставлен флаг в соответствующей строке. Отметим, что для каждого состояния используется отдельная таблица — функция переходов автомата определяется набором таблиц.

При этом в отличие от метода битовых строк такое представление позволяет использовать метод при генерации автоматов с любым числом состояний и выходных воздействий.

Генетические операции в этом случае производятся над таблицами. На рис. 3 иллюстрируется адаптация метода одноточечного кроссовера к рассматриваемому представлению управляющих автоматов. Руководствуясь аналогичным подходом, можно адаптировать к табличному представлению и другие способы скрещивания.

Алгоритм мутации состояния, представленного таблицей, проиллюстрирован на рис. 4 (на стрелках, обозначающих изменения значений в ячейках таб-

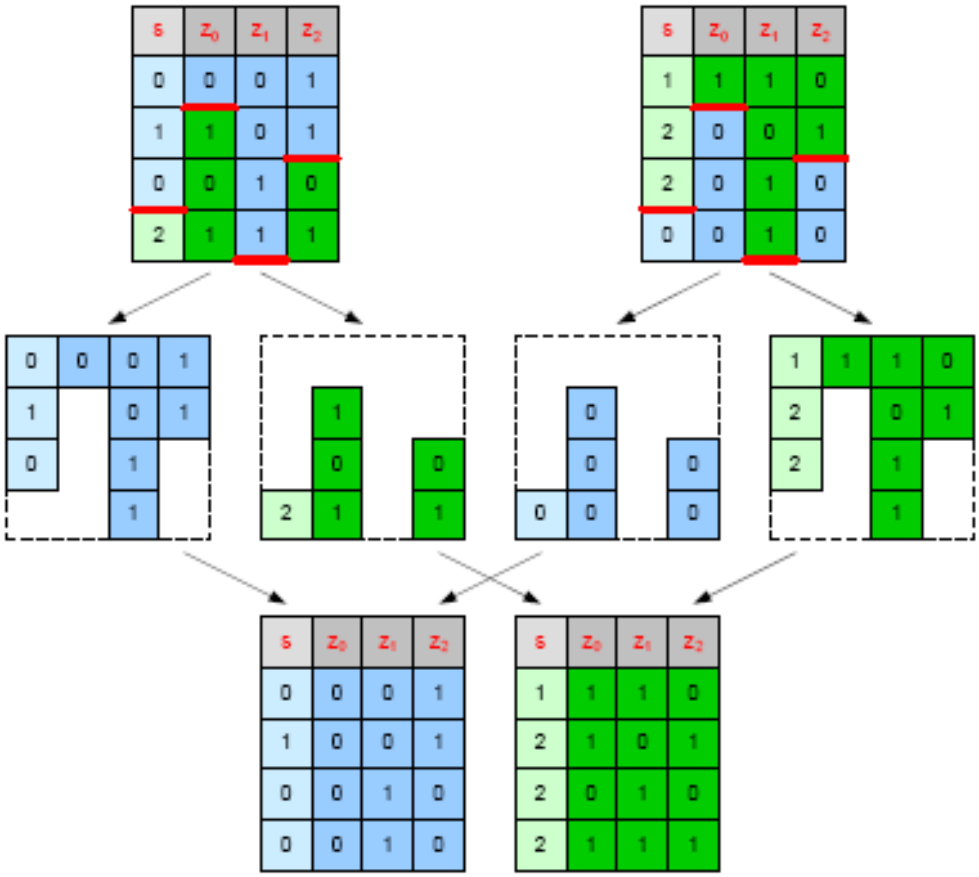

Рис. 3. Пример скрещивания таблиц переходов

лицы, написаны вероятности этих изменений). При мутации состояния с некоторой вероятностью может мутировать каждый элемент таблицы. При этом номер целевого состояния изменяется на любой из допустимых, а вместо исходного набора действий генерируется новый набор, вероятность появления единиц в котором равна доле единиц в исходном наборе.

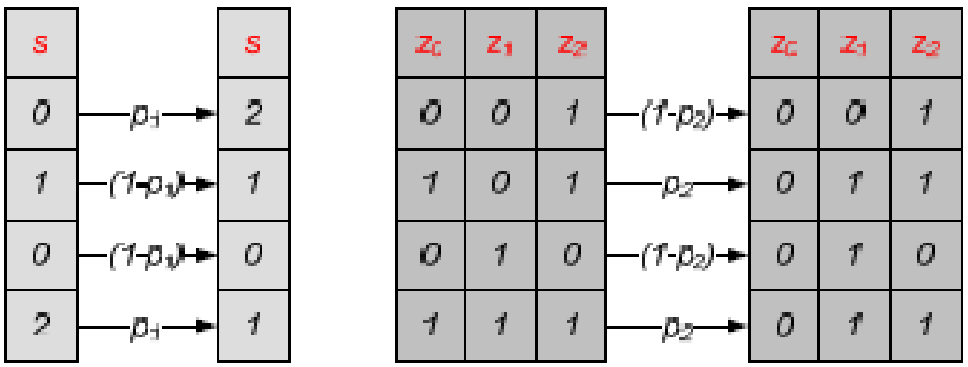

Рис. 4. Пример мутации таблицы переходов

1.3.3. Сокращенные таблицы переходов

Основным недостатком изложенных методов является экспоненциальный рост размерности хромосомы с увеличением числа входных переменных. Однако, в реальных задачах входные воздействия имеют локальную природу по отношению к управляющим состояниям автомата. В каждом состоянии значимым является лишь определенный, сравнительно небольшой набор входных воздействий.

Метод сокращенных таблиц [6] применим, когда число значимых в состоянии воздействий ограничивается некоторой константой r. К таблице, задающей сужение управляющей функции на данное состояние, в этом случае добавляется битовый вектор, описывающий множество значимых входных воздействий (рис. 5). В примере значимыми являются воздействия x_1 и x_3. Управляющая функция этого состояния описывается полной таблицей переходов для автомата, имеющего только r воздействий.

x_0	x_1	x_2	x_3	x_4	x_5
0	1	0	1	0	0

x_1	x_3	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. 5. Пример сокращенной таблицы переходов

Опишем операцию скрещивания над сокращенными таблицами. Поскольку родительские хромосомы, представленные сокращенными таблицами, могут иметь разные множества значимых воздействий, сначала необходимо выбрать, какие из этих воздействий будут значимы для хромосом детей. Опишем алгоритм выбора значимых входных воздействий детей. В первую очередь, отметим, что воздействия, использующиеся в обоих родителях, будут использованы и в обоих детях. Далее воздействия, присутствующие только у одного из родителей, случайным образом поровну перераспределяются между детьми. Работа этого алгоритма для родительских хромосом, проиллюстрирована на рис. 6.

После выбора значимых воздействий и точки деления родительских таблиц заполняются таблицы обоих детей. На значения каждой строки таблицы

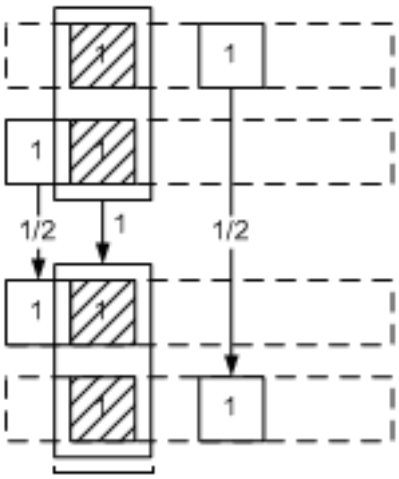

Рис. 6. Выбор значимых входных воздействий детей

ребенка влияют значения нескольких строк родительских таблиц. Выбираются те строки родительских таблиц, в которых воздействия, значимые для ребенка, имеют те же значения, что и в текущей строке одного из родителей. Причем, если воздействие значимо для обоих родителей, то его значение учитывается только для одного из родителей (в зависимости от положения строки i относительно точки деления). Затем данные всех выбранных строк усредняются — подсчитывается вероятность использования различных конечных состояний и выходных воздействий. Конечные значения для i-й строки ребенка вычисляются исходя из полученных вероятностей. Алгоритм заполнения таблиц переходов порождаемых автоматов проиллюстрирован на рис. 7.

Мутация сокращенных таблиц происходит в два этапа — мутирует множество значимых воздействий и сама таблица. Сначала каждое из значимых воздействий с некоторой вероятностью заменяется другим воздействием, не принадлежащим множеству. Далее с некоторой вероятностью может мутировать каждый элемент таблицы. При этом номер целевого состояния изменяется на любой из допустимых, а вместо исходного набора действий генерируется новый набор, вероятность появления единиц в котором равна доле единиц в исходном наборе.

1.4. Анализ достоинств и недостатков известных методов

Достоинством представления переходов строками фиксированной длины является простота реализации. Однако этот метод плохо подходит для задач со

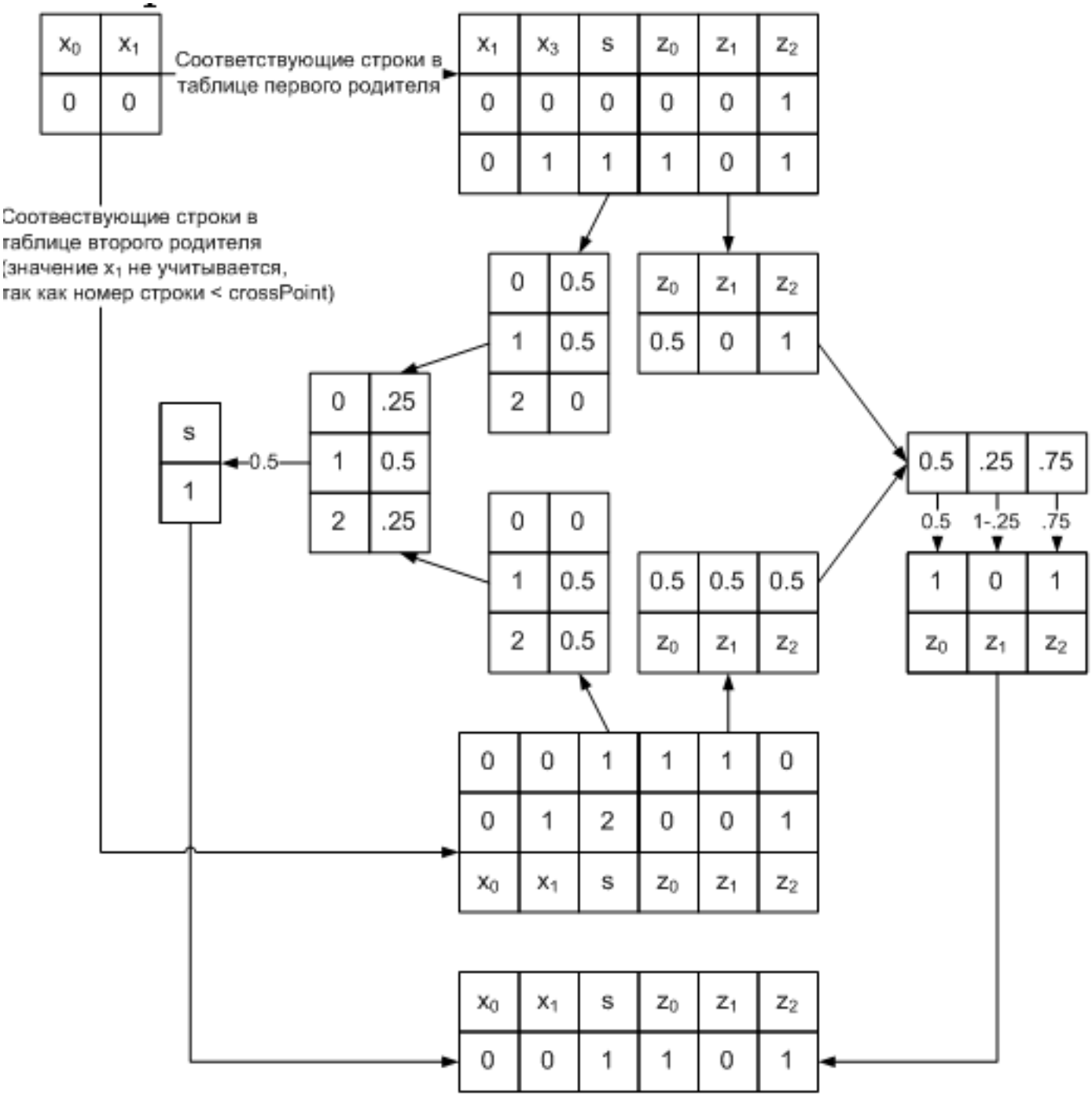

Рис. 7. Пример заполнения таблицы переходов ребенка

многими входными переменными. В силу того, что приходится задавать переход автомата для всех комбинаций значений переменных, длина строки, соответствующей автомату, растет экспоненциально.

Метод представления функции переходов полными таблицами переходов оказывается неприменимым по той же причине. Достоинством метода являет-

ся возможность задать несколько действий на переходе. Заметим, однако, что указать порядок их выполнения нельзя.

Метод представления функции переходов сокращенными таблицами представляется наиболее перспективным. Его недостатком является то, что параметр r, ограничивающий число важных в состоянии переменных необходимо определить еще до запуска генерации. Кроме того, указанный параметр является общим для всех состояний, что делает метод менее эффективным для генерации автоматов, функции переходов из различных состояний которого имеют различную сложность.

Выводы по главе 1

1. Выполнен обзор существующих реализаций эволюционных алгоритмов для генерации автоматов на примере приведенных задач.
2. Отмечены достоинства и недостатки известных подходов.

Глава 2.
Представление функции переходов автомата деревьями решений для применения в генетическом программировании

В настоящей главе предлагается новый метод представления функции переходов при генерации автоматов с помощью генетического программирования, основанный на деревьях решений [7]. Демонстрируется возможность задания ограничений на целевой автомат. Это придает методу дополнительную гибкость.

2.1. Деревья решений

Дерево решений является удобным способом задания дискретной функции, зависящей от конечного числа 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 выдать значение, которым помечен полученный лист. При этом из вершины, помеченной переменной x, переход производится по тому ребру, которое помечено тем же значением, что и значение x.

Пример дерева решений изображен на рис. 8. Здесь для простоты показано дерево, реализующее булеву функцию от переменных a, b и c.

Как правило, в большинстве практических задач дерево решений требует значительно меньше памяти по сравнению с заданием функции на всех наборах значений входных переменных (с помощью таблиц истинности).

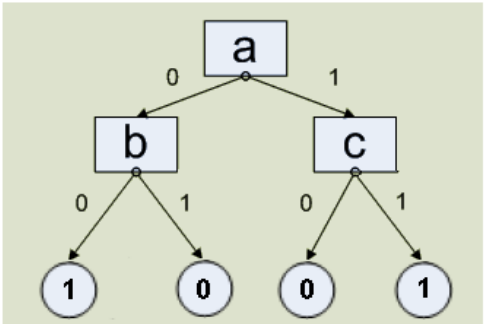

Рис. 8. Пример дерева решений

2.2. Представление автомата с помощью деревьев решений

Опишем предлагаемый метод представления автомата с помощью деревьев решений. Для задания автомата необходимо выразить его функции переходов и выходов с помощью деревьев решений. Возможно осуществить следующее преобразование автомата: вместо задания функций переходов и выходов для автомата в целом, представим эти функции для каждого состояния. Более формально: зададим для каждого состояния $q \in Q$ функцию $\sigma_q : X \to Q \times Y$, такую что $\sigma_q(x) = (\delta(q, x), \lambda(q, x)), \forall x \in X$.

Функции σ_q соответствуют функциям переходов и действий из состояния q. Каждая из этих функций может быть выражена с помощью дерева решений. В этих деревьях переменными являются входные переменные автомата, а множеством значений – все возможные пары *(Новое Состояние, Действие)*. Таким образом, автомат в целом задается упорядоченным набором деревьев решений.

На рис. 24 приведен пример автомата Мили. При этом a, b, c – входные переменные булевого типа, а L, R, F – выходные воздействия. Этот автомат, представленный с использованием деревьев решений, приведен на рис. 10.

При таком представлении функции переходов из каждого состояния можно ожидать значительного уменьшения длин хромосом по сравнению с хранением ее в виде таблицы. Как правило, в автоматном программировании [2] в каждом состоянии важен лишь сравнительно небольшой набор из всего множества входных переменных автомата. Таким образом, представление автомата с помощью набора деревьев решений должно значительно сократить длину хромосомы, а, следовательно, и ускорить генерацию.

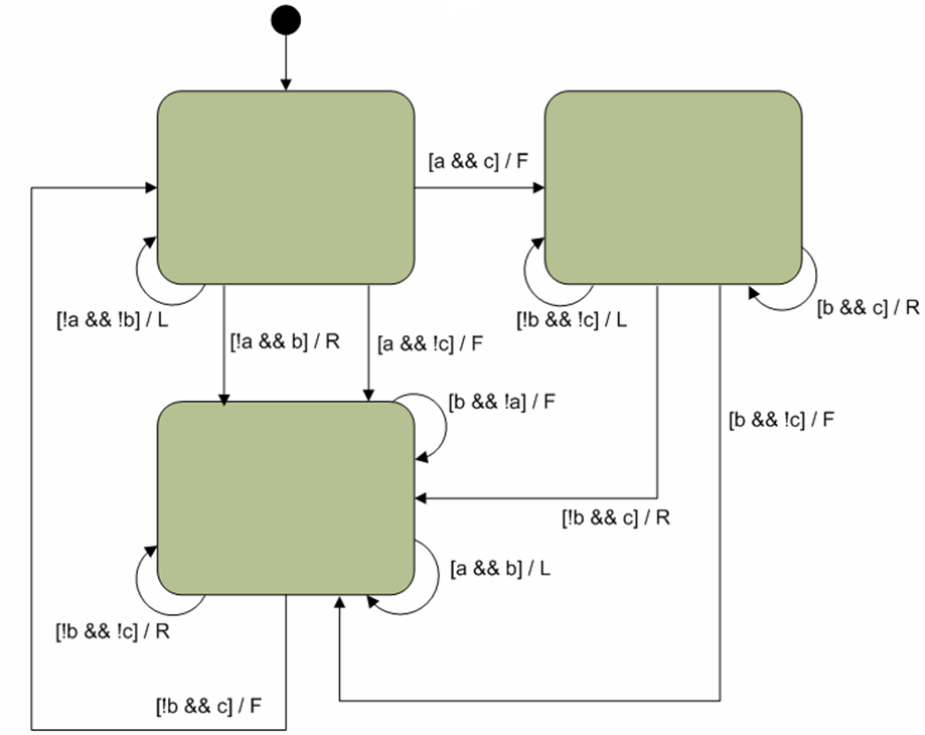

Рис. 9. Пример автомата Мили

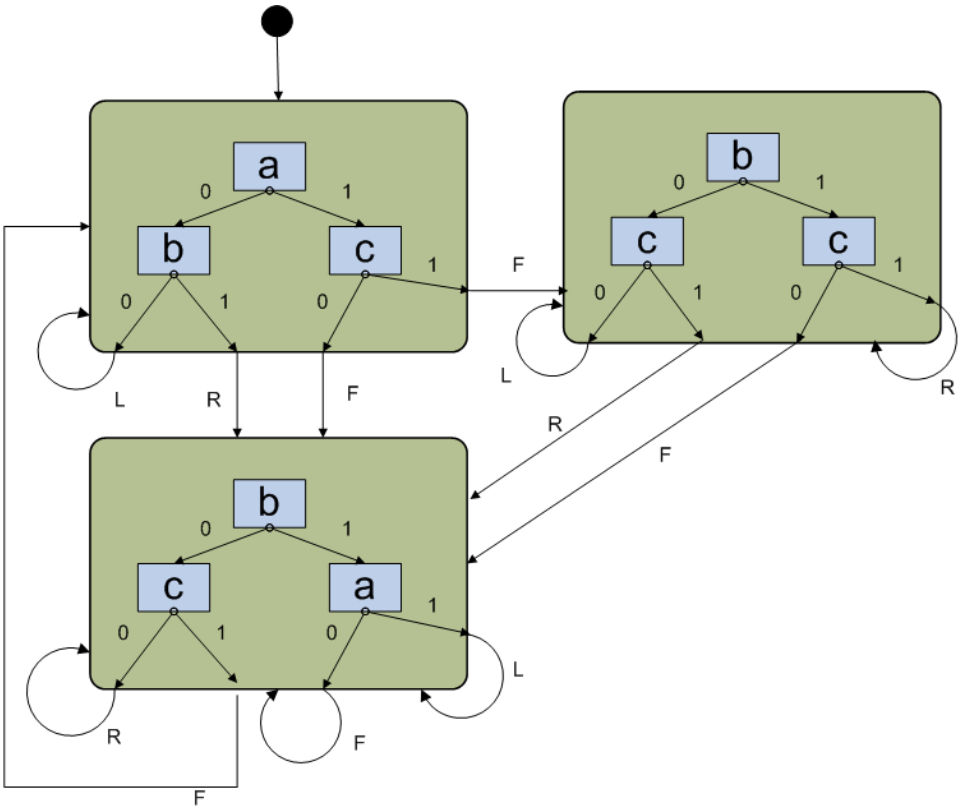

Рис. 10. Представление автомата Мили с помощью деревьев решений

2.3. Генетические операции

Для такого представления автоматов в эволюционных алгоритмах необходимо задать генетические операции. Они могут быть следующими:

- случайное порождение автомата — в каждом состоянии создается случайное дерево решений;
- скрещивание автоматов — скрещиваются деревья решений в соответствующих состояниях;
- мутация автомата — в случайном дереве решений выполняется мутация;
- отбор — допустимо применение любого метода, используемого в генетических алгоритмах.

Здесь считается, что число состояний в автомате фиксировано, и поэтому противоречий при выполнении определенных таким образом операций не возникнет.

После определения операций над набором деревьев решений, определим генетические операции над собственно деревьями решений. Это можно сделать с помощью операций, аналогичных операциям, используемым в генетическом программировании над деревьями разбора [3]. В деревьях решений переменные являются нетерминалами, а значения функции — терминалами. При этом арность нетерминала, соответствующего переменной x, равна мощности множества возможных значений переменной. Приведем описание генетических операций над деревьями решений:

- Случайное порождение дерева решений — случайным образом выбирается метка: либо одно из возможных значений функции переходов, либо одна из переменных. После этого создается вершина, помеченная выбранной меткой. При этом если была выбрана переменная, то рекурсивно генерируются дети вершины, иначе вершина становится листом дерева.
- Мутация — выбирается случайный узел в поддереве. После этого поддерево, соответствующее выбранному узлу, заменяется на случайно сгенерированное (рис. 11).
- Скрещивание — в скрещиваемых деревьях выбираются два случайных узла. После этого поддерево, соответствующее выбранному узлу в первом дереве, заменяется поддеревом, соответствующим узлу второго дерева (рис. 12).

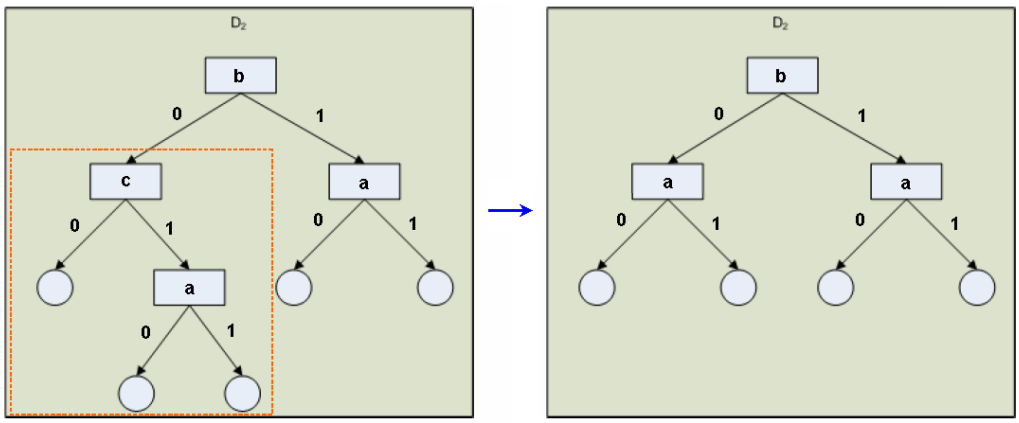

Рис. 11. Мутация деревьев решений

Отметим, что заданные таким образом операции могут порождать деревья, в которых некий атрибут встречается на пути от корня до листа дважды (например, дерево, полученное в результате скрещивания на рис. 12). Такие деревья являются корректными — определяют функцию на любом наборе значений атрибутов единственным образом, однако имеют недостижимые вершины. Действительно, если атрибут встречается на пути дважды, то его значение уже известно, следовательно, одна из ветвей недостижима ни при каких значениях предикатов. Появление недостижимых ветвей может влиять на эволюцию отрицательным образом по следующим причинам:

- Появление недостижимых ветвей ведет к увеличению высоты деревьев, экспоненциально увеличивая память, необходимую на хранение популяции. В условиях ограниченной памяти это ведет к уменьшению популяции.
- Недостижимые ветви дерева могут являться плохими приближениями искомой функции (так как не учитываются при подсчете функции приспособленности), но при этом, однако, они могут копироваться в потомков, замедляя генерацию.

Таким образом, необходимо ввести операцию обрезки — удаление недостижимых ветвей. Операция может быть выполнена следующим образом: узел, одна из дочерних вершин которого недостижима, заменяется на свою достижимую дочернюю вершину. На рис. 13 приведен пример обрезки недостижимых ветвей. Операция обрезки должна выполняться после генетических операций скрещивания и мутации.

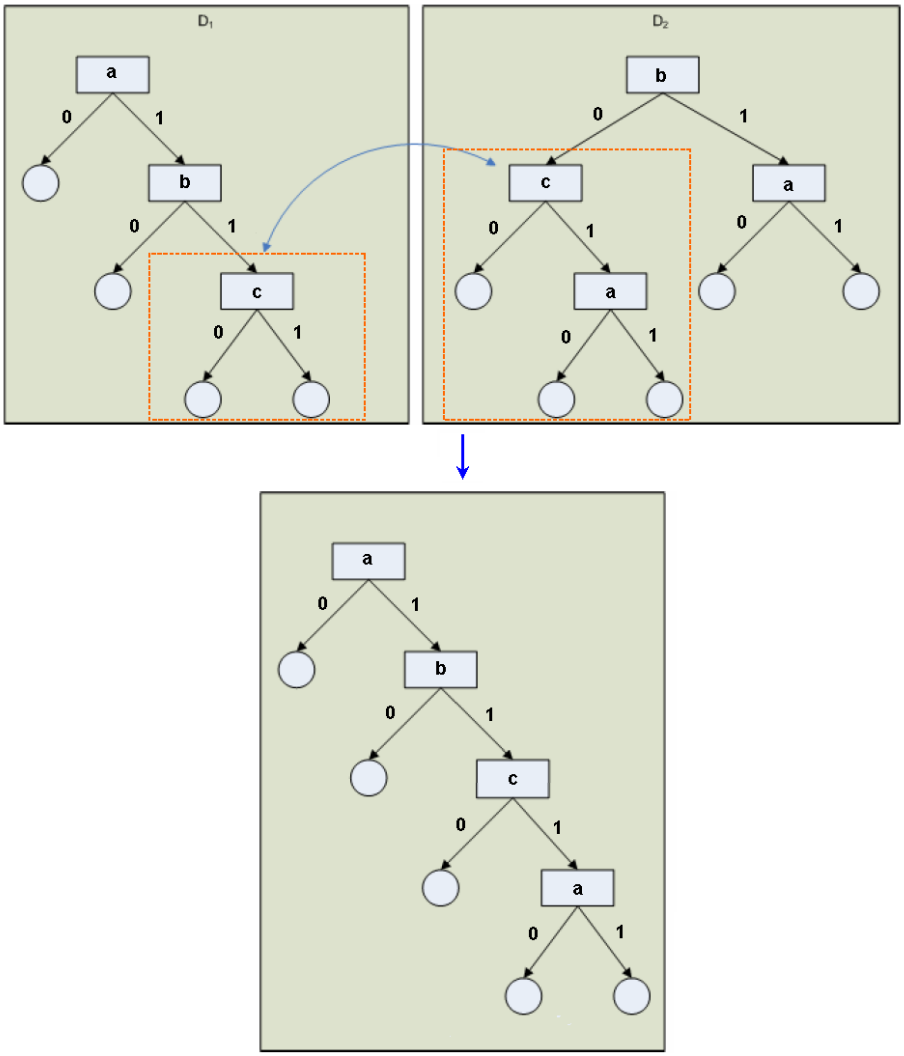

Рис. 12. Скрещивание деревьев решений

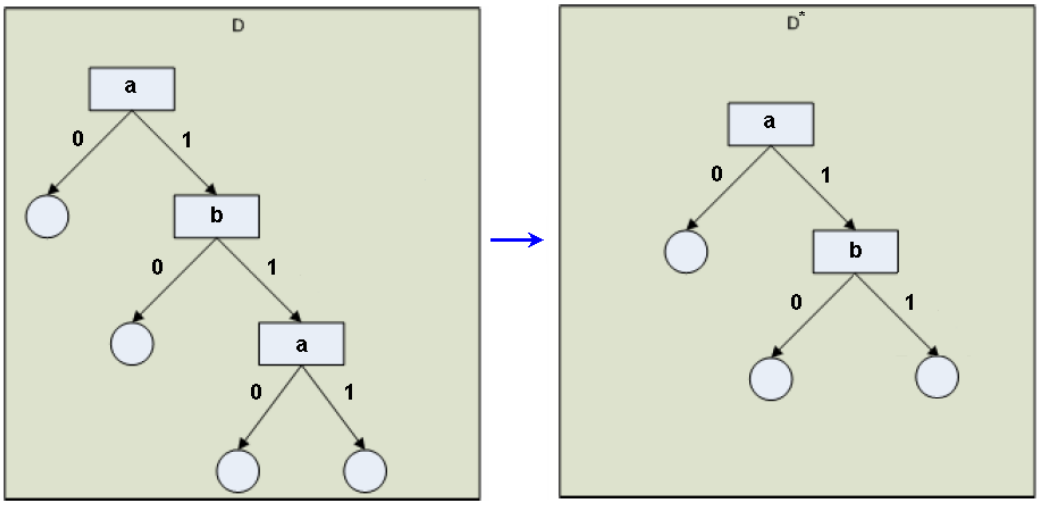

Рис. 13. Пример обрезки недостижимых ветвей

2.4. Пример применения разработанного метода

Примененим разработанный метод к решению задачи „Умный Муравей“. В данной задаче используется одна входная переменная булевого типа. Обозначим его X. Выходные воздействия поворота налево, направо и шага вперед обозначим L, R и F соответственно. В случае, если дерево решений, подвешенное в состоянии, не использует никаких входных переменных будем рисовать один переход.

Для простоты рассмотрим работу метода на примере популяции, размер которой равен четырем. Число состояний в каждом из автоматов положим равным двум.

Допустим, что на определенном шаге генерации получена некоторая популяция (рис. 14). Здесь под каждым из автоматов популяции приведено значение функции приспособленности — количество съеденной за отведенное время еды.

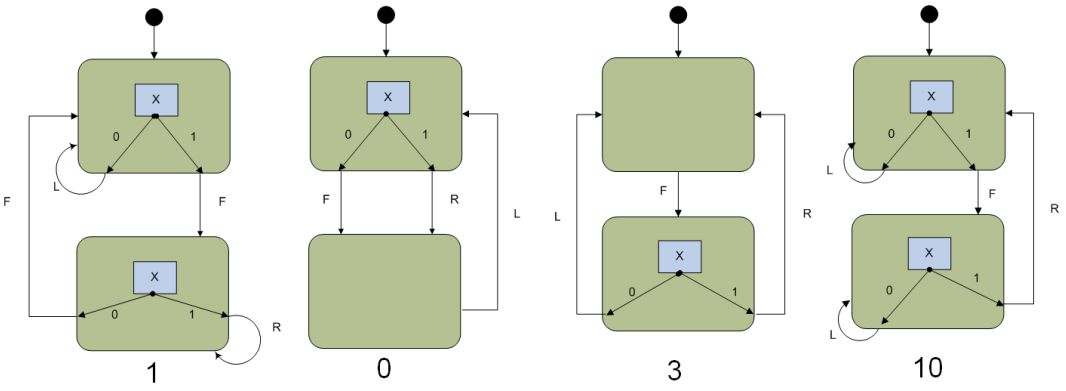

Рис. 14. Популяция автоматов

В случае, когда в качестве метода отбора применяется метод рулетки, для производства потомства отбирается несколько лучших представителей. Пусть в рассматриваемом примере отбираются два лучших автомата популяции. Тогда два оставшихся автомата покидают популяцию, а их место занимают потомки отобранных для размножения представителей. Порождение потомства путем скрещивания показано на рис. 15.

После этого порожденное скрещиванием потомство подвергается действию мутаций. Операция совершается над каждым из порожденных автоматов. С определенной долей вероятности происходит мутация в одном из состояний автомата — над соответствующим деревом решений. На рис. 16 приведен пример действия мутаций на порожденное потомство.

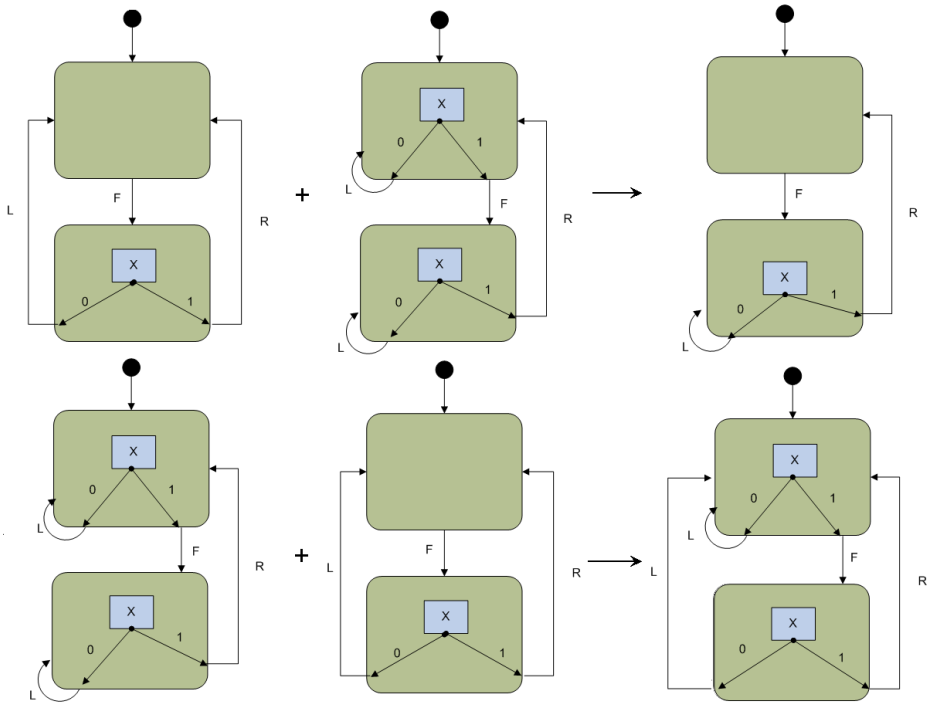

Рис. 15. Порождение потомства

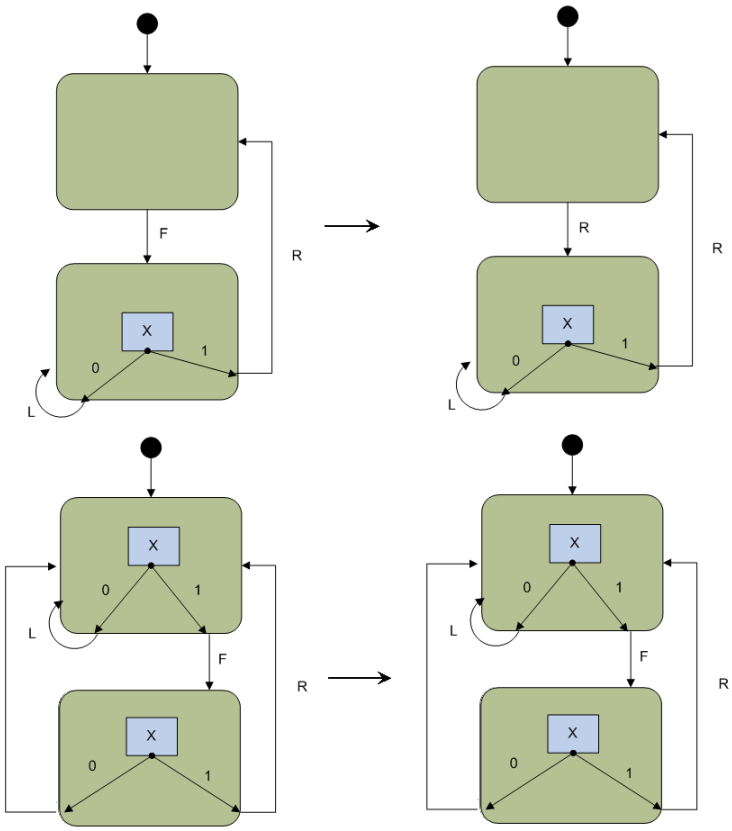

Рис. 16. Мутация потомства

Таким образом, популяция преобразовалась, как показано на рис. ??:

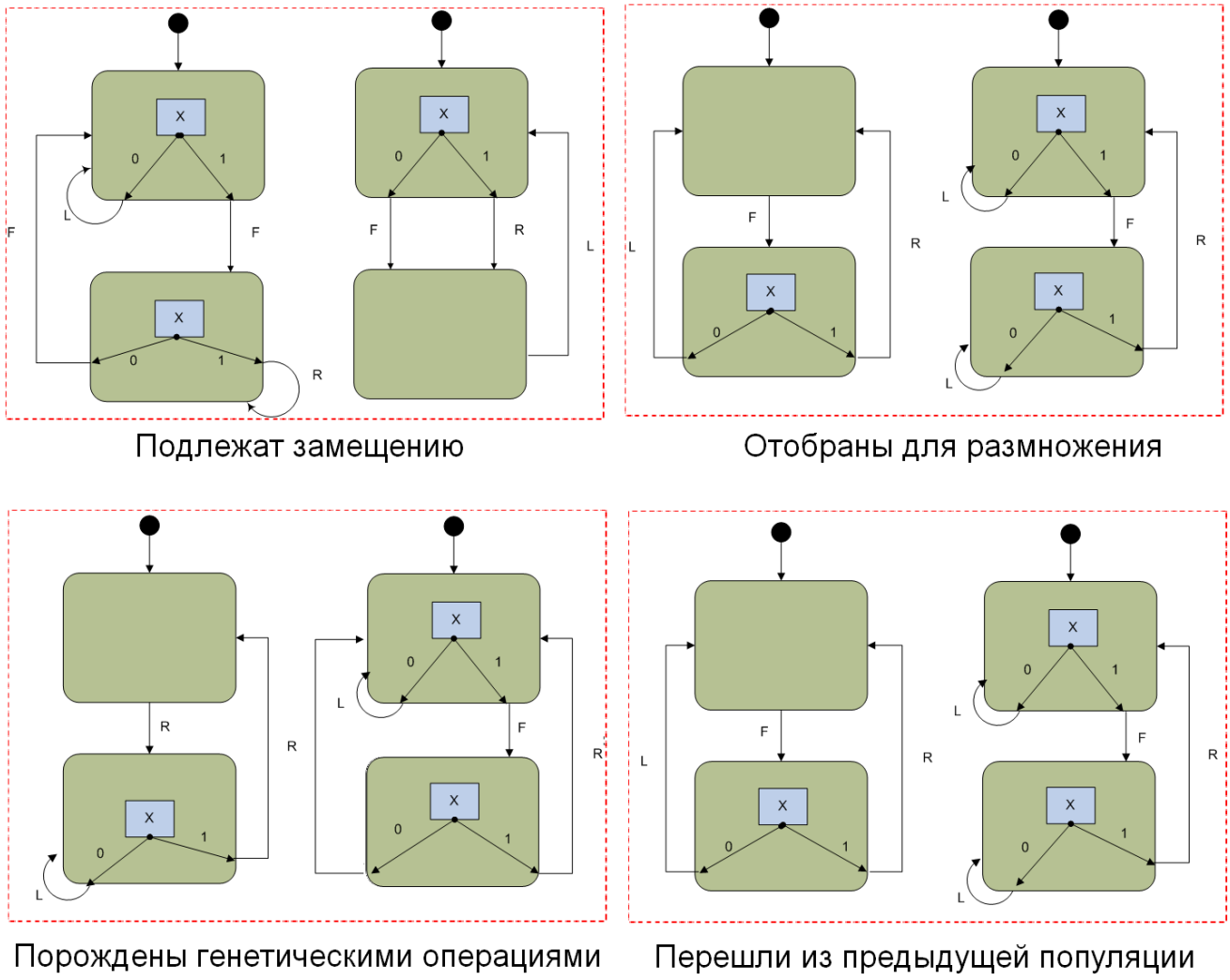

Рис. 17. Преобразование популяции

Многократное повторение подобных шагов генерирует исходный автомат.

2.5. Задание ограничений на целевой автомат

Иногда при применении эволюционных алгоритмов оказывается удобным задавать ограничения на целевую особь. Заметим, что для автоматов было бы полезно уметь ограничивать сложность логики и число переходов. При использовании деревьев решений в качестве представления автомата, сложность логики естественным образом соответствует глубине деревьев решений, а число переходов – числу листьев. Ограничения можно задать следующим способом: ввести функцию штрафа за нарушение ограничений и прибавлять ее к функции

приспособленности. Таким образом, неподходящие под ограничения хромосомы будут отсеяны самой эволюцией. Возможно применение и других способов, например повторение генетических операций скрещивания и мутации до получения особи, удовлетворяющей ограничениям.

2.6. Модификации метода

Разработанный метод обеспечивает возможность построения модификаций. Приведем некоторые из них:

- Возможно раздельное хранение функций переходов и выходов. В этом случае в каждом состоянии будет храниться не одно дерево, соответствующее функции σ_q , а два дерева, соответствующие функциям λ_q и δ_q. Генетические операции над автоматами, заданными таким образом, определяются аналогично.
- Предлагаемый метод выше был изложен для генерации автоматов Мили. Отметим, что он может быть адаптирован и для генерации автоматов Мура. В этом случае для каждого из переходов, который хранится в узле дерева решений, необходимо хранить только вершину, в которую ведет рассматриваемый переход. Кроме того, в каждом состоянии необходимо хранить ассоциированное выходное воздействие, совершаемое при входе в вершину. Генетические операции над автоматами дополняются скрещиванием наборов выходных воздействий в состояниях любым из способов, используемых в генетическом программировании, например методом одноточечного кроссовера.

2.7. Особенности метода

К достоинствам разработанного метода относятся:

1. Уменьшение длины хромосомы.
2. Естественность представления автомата для человека.
3. Возможность наложения ограничений на генерируемый автомат.

Уменьшение длины хромосомы осуществляется за счет большей выразительности деревьев решений по сравнению с таблицами переходов. Это отражается в уменьшении необходимого объема памяти для хранения хромосомы,

что позволяет увеличить популяцию, а также ускоряет вывод. Отметим, что для функций, плохо представимых с помощью деревьев решений, необходимый объем памяти наоборот возрастает. Уменьшение длины хромосомы ведет также и к ускорению генетических операций. Несмотря на то, что измерение функции приспособленности, как правило, является наиболее затратным шагом генетических алгоритмов, с увеличением числа предикатов время выполнения генетических операций может также стать существенным.

Естественность представления автомата для человека вытекает из упрощения представления функций с помощью деревьев решений по сравнению с полной таблицей переходов.

Возможность наложения ограничений на генерируемый автомат представляется довольно значительным преимуществом, так как позволяет сузить область поиска, тем самым ускоряя генерацию автоматов. Кроме того, часто требуется найти не оптимальное, а достаточно простое хорошее решение, что может быть осуществлено с помощью задания ограничений на целевой автомат.

Основными недостатками использования предложенного представления автоматов являются:

1. Возможность нескольких представлений одного и того же автомата.
2. Потеря части генов родителя при выполнении операции обрезки.

Отметим, что возможность нескольких представлений является общей особенностью представления хромосом на достаточно высоком уровне абстракции, например, такая возможность имеется также при представлении автоматов с помощью сокращенных таблиц. При этом неоднозначность представления обуславливается следующими причинами:

- неоднозначностью нумерации состояний;
- неоднозначностью представления функции с помощью деревьев решений.

Классическим способом устранения неоднозначности нумерации состояний является эвристика *MTF (Move To Front)* [8]. Она заключается в перенумерации состояний в порядке обхода графа переходов автомата в ширину. Особенностью применения этой эвристики к рассматриваемому методу является то, что необходимо определить порядок переходов из вершины. Это можно сделать, например, следующим образом: найти для каждого перехода лексикографически

минимальный набор значений входных переменных, соответствующий этому переходу. После этого рассматривать переходы в порядке лексикографического возрастания соответствующего набора. Рассмотрим путь от корня до листа дерева, соответствующего переходу. Значения всех переменных, встретившихся на этом пути, фиксированы для всех наборов, соответствующих переходу. Остальные переменные могут принимать любые значения. Лексикографически минимальным из подходящих наборов будет набор, в котором все переменные, значения которых не фиксированы, принимают минимальное значение.

Неоднозначность представления функции переходов может быть устранена любым алгоритмом построения дерева решений по набору значений функции. Важно отметить, что при этом частое перестроение деревьев может сыграть отрицательную роль, так как при этом теряется структура деревьев, которая влияет на генетические операции.

Потеря части генов родителя при операции обрезки ведет к тому, что хорошие поддеревья при копировании в потомков постепенно вырождаются. Однако, это представляется меньшей проблемой, чем появление генетического мусора — излишней информации. Одним из способов решения данной проблемы является применение модификации эвристик сжатия и расширения [5]. Наиболее естественным преобразованием сжатия является запрет на изменение одного из поддеревьев. После этого генетические операции не могут изменять сжатое поддерево. Преобразование расширения в таком случае логично сделать обратным операции сжатия. Модификация заключается в добавлении дополнительного требования: запрете обрезки сжатых поддеревьев.

Выводы по главе 2

1. Предложен новый метод представления функции переходов автоматов управления для использования в генетическом программировании.
2. Рассмотрена возможность задания ограничений на генерируемый автомат.
3. Выполнен теоретический анализ особенностей метода, выделены недостатки, предложены способы их устранения.

Глава 3.
Представление функции переходов автомата линейными бинарными графами для применения в генетическом программировании

Недостатком подхода, изложенного в главе 2, является повторное кодирование одинаковых поддеревьев. В настоящей главе предлагается метод представления функций переходов управляющих автоматов с помощью линейных бинарных графов.

3.1. Линейные бинарные графы

Более компактным представлением функций переходов по сравнению с деревьями решений являются разрешающие диаграммы [9]. Каждая из них представляет собой помеченный ацикличный ориентированный граф, в котором выделяют вершины двух типов:

- нетерминальные узлы;
- терминальные узлы.

При этом один из нетерминальных узлов является начальным. Все остальные узлы достижимы из него. Из каждого нетерминального узла выходит два ребра. Из терминальных узлов ребер не выходит. Метки в графе расставляются по следующим правилам:

- нетерминальные узлы помечаются названиями переменных;
- терминальные узлы — значениями функции;
- ребра — значениями переменных.

Для определения значения функции по значениям переменных необходимо пройти путь от начального узла до терминального, и сформировать значение, которым помечен этот терминальный узел. При этом из вершины, помеченной переменной x, переход производится по ребру с пометкой, равной значению

указанной переменной. На рис. 18 приведен пример разрешающей диаграммы, реализующей булеву функцию $f = a \oplus b \oplus c$.

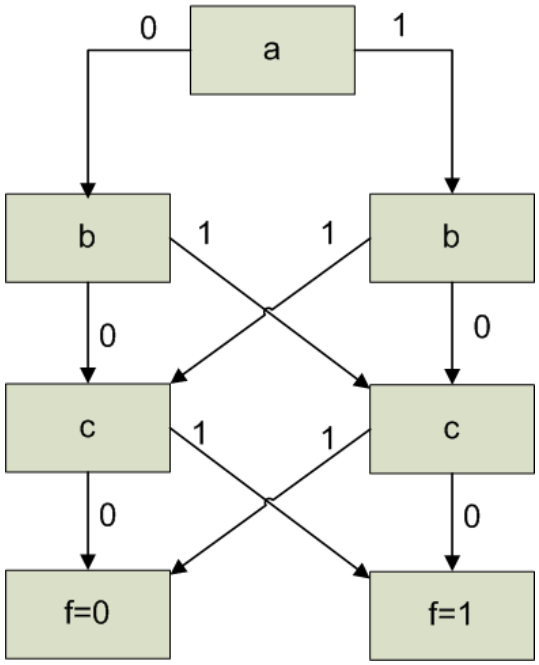

Рис. 18. Пример разрешающей диаграммы

Важным частным случаем разрешающих диаграмм являются линейные разрешающие диаграммы или линейные бинарные графы [10]. Узлы линейного бинарного графа могут быть пронумерованы таким образом, что из каждого нетерминального узла одно из ребер ведет в узел со следующим номером. При этом число элементов, необходимых для реализации булевой формулы бинарным линейным графом, линейно зависит от числа букв в формуле. Получающийся при этом линейный бинарный граф является планарным.

Пример линейного бинарного графа, реализующего булеву формулу $f = ab \vee c$, приведен на рис. 19.

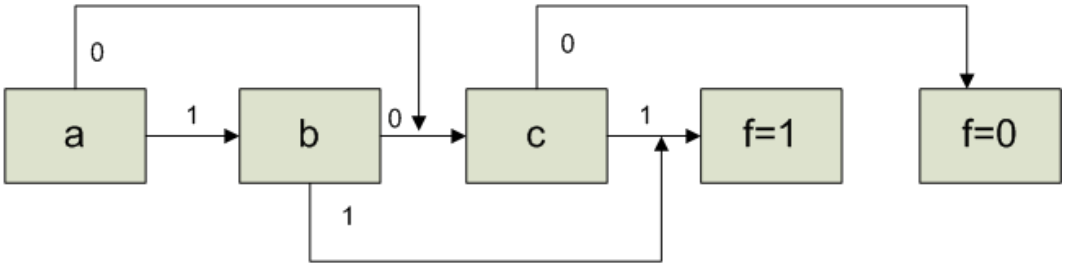

Рис. 19. Пример линейного бинарного графа

3.2. Представление автоматов линейными бинарными графами

Опишем предлагаемый метод представления автомата с помощью бинарных линейных графов. Введем функции переходов из каждого состояния, аналогично представлению функции переходов деревьями решений: $\forall q \in Q$ зададим $\sigma_q : X \to Q \times Y$, такую что $\sigma_q(x) = (\delta(q,x), \lambda(q,x)), \forall x \in X$. Определим эти функции σ_q с помощью линейных бинарных графов. Автомат в целом будем задавать упорядоченным набором линейных бинарных графов, соответствующих состояниям.

Определим способ задания функций переходов из состояния в автомате с помощью линейных бинарных графов. Пусть для примера функции переходов из состояния определяются следующими правилами (рис. 20):

- $ab \vee c$ — действие L;
- $\neg a \neg c$ — действие R;
- $a \neg b \neg c$ — действие F.

Эта система правил должна быть непротиворечива и полна.

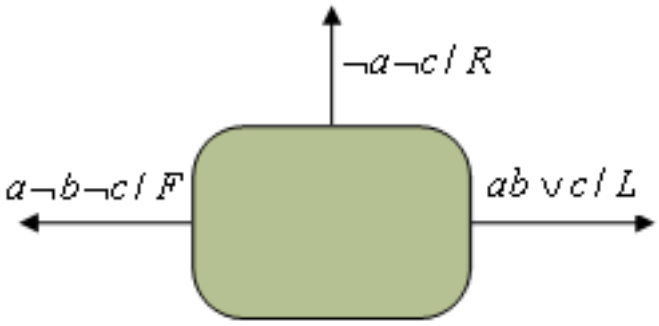

Рис. 20. Пример функции переходов из состояния

Построим для каждой из перечисленных формул по планарному линейному бинарному графу(рис. 21):

Заметим, что терминальные узлы, соответствующие значениям 0 и 1 всегда можно поменять местами, не нарушая свойства линейности бинарного графа. Перерисуем, сохраняя планарность, графы таким образом, чтобы терминальный узел, помеченный значением 0, будет стоять перед узлом, помеченным значением 1. Преобразованные линейные бинарные графы приведены на рис. 22.

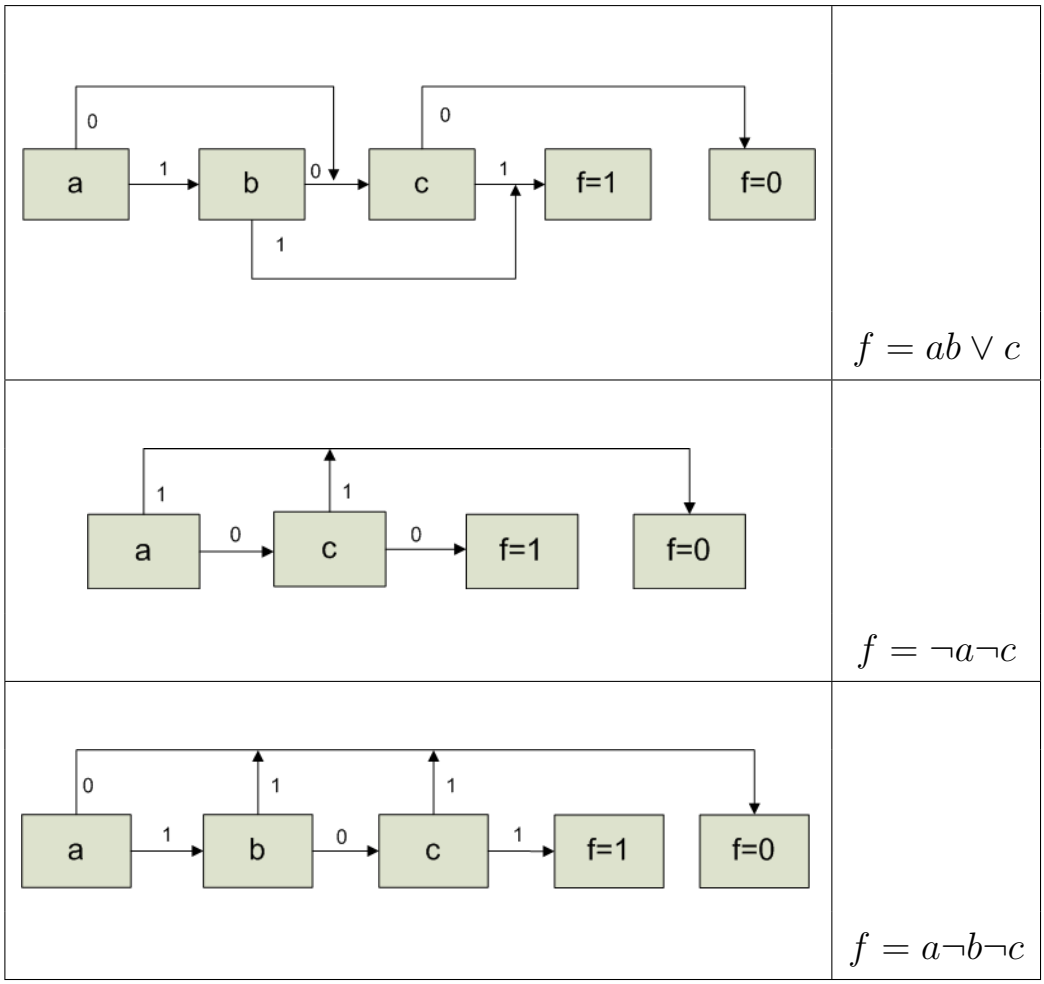

Рис. 21. Линейные бинарные графы для формул на переходах

После этого заменим в реализации каждой из перечисленных формул значение 1 на соответствующее значение функции выхода. Теперь будем последовательно заменять терминальные узлы, помеченный нулем, бинарными линейными графами, соответствующими реализациям следующих формул. Построенный бинарный граф будет также линейным.

В полученном линейном бинарном графе будет терминальный узел, помеченный значением 0, однако он оказывается недостижимым ни при каких значениях входных переменных, так как рассматриваемая система формул, записанных на переходах, является полной. Поэтому весь линейный бинарный граф, соответствующий последней подформуле $f = a\neg b\neg c$, можно заменить на терминальный узел F. В итоге получим следующий линейный бинарный граф (рис. 23).

Произвольная система функций переходов из состояния может быть выражена линейным бинарным графом аналогичным образом. При реализации

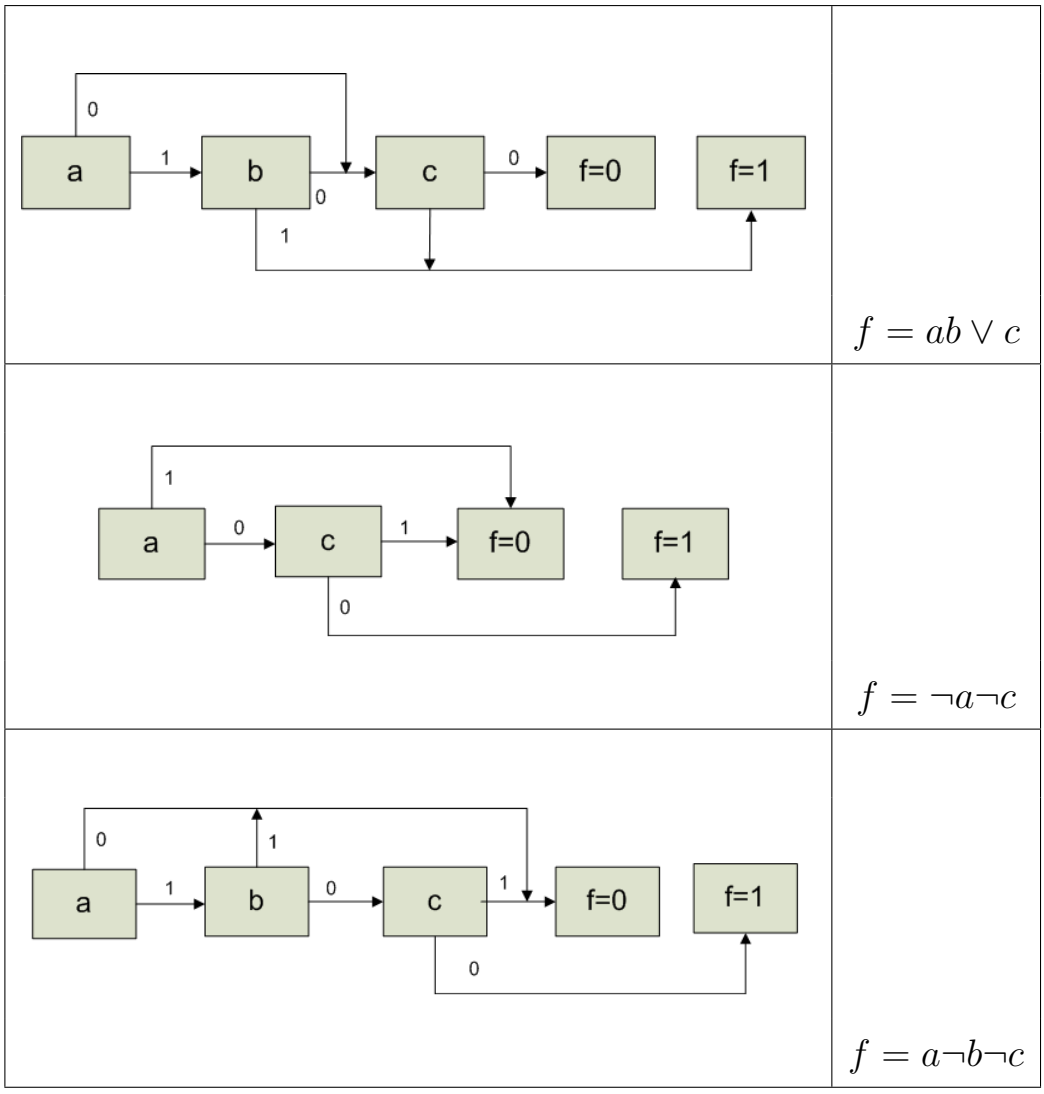

Рис. 22. Преобразованные линейные бинарные графы для формул на переходах

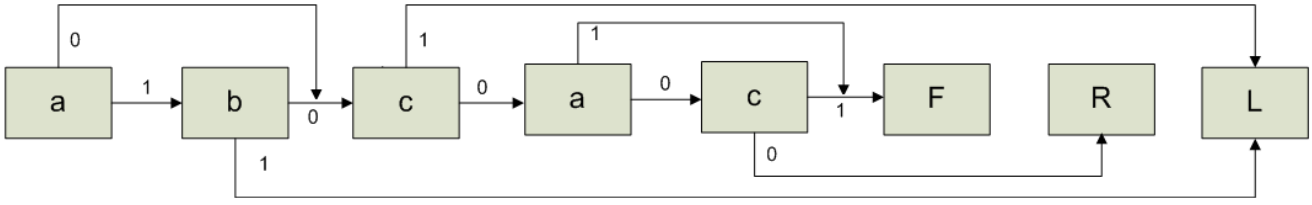

Рис. 23. Бинарный линейный граф, соответствующий функции переходов из состояния

этого графа предполагается, что значение входных переменных не меняется в процессе вычисления.

На рис. 24 приведен пример автомата Мили. При этом a, b, c – входные переменные булевого типа, а L, R, F – выходные воздействия. Этот же автомат, представленный линейными бинарными графами, приведен на рис. 25.

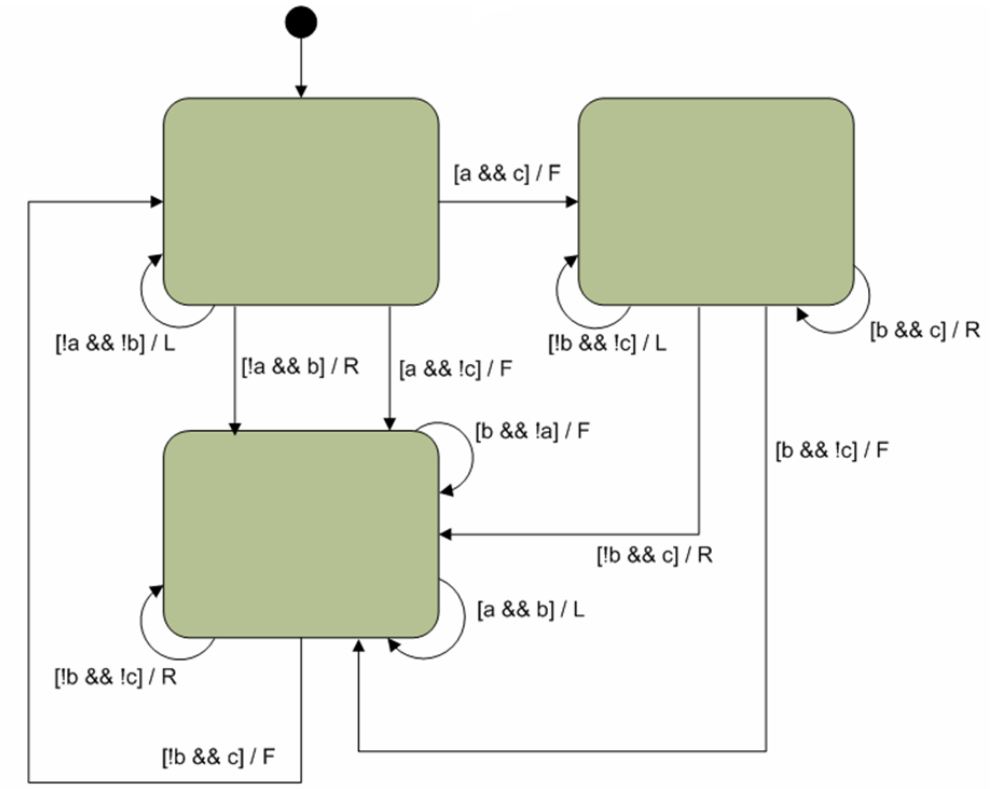

Рис. 24. Пример автомата Мили

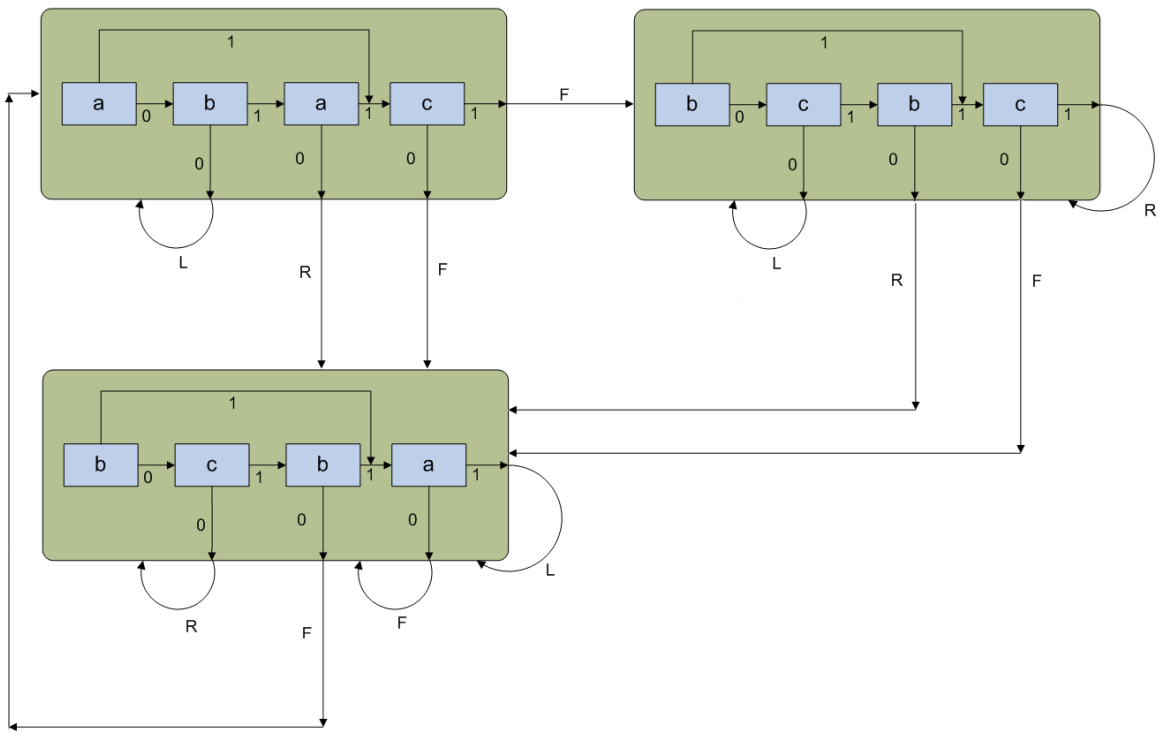

Рис. 25. Представление автомата линейными бинарными графами

3.3. Представление линейного бинарного графа в виде хромосомы

Определим представление линейных бинарных графов в виде хромосом:

1. Граф представляет собой упорядоченный список узлов. При этом считается, что узлы перечисляются в таком порядке, что из каждого узла одно из выходящих ребер ведет в следующий узел.
2. В каждом нетерминальном узле хранится переменная, которой помечен данный узел (переменная расщепления), значение этой переменной, соответствующее переходу в следующий узел, и номер узла, в который производится переход при инверсном значении переменной расщепления.
3. В каждом из терминальных узлов хранится значение соответствующей функции переходов.

Описанное представление хромосомы может быть записано в виде строки. При этом терминальные узлы являются последними узлами линейного бинарного графа. Поэтому возможно отдельно выделить части, кодирующие терминальные и нетерминальные узлы. Продемонстрируем это на примере. Закодируем в виде строки граф, приведенный на рис. 19. Заметим, что узлы этого графа пронумерованы в требуемом порядке слева направо. Тогда этому графу будет соответствовать следующая строка:

a13b04c15 10,

в которой указанные выше части бинарного графа разделены пробелом.

Поясним первые три символа этой строки. Из первой вершины, помеченной переменной a, переход в соседнюю вершину выполняется по единице, переход по нулю выполняется в третью вершину графа.

3.4. Генетические операции

Для использования представления автоматов в виде набора линейных бинарных графов в генетическом программировании необходимо определить генетические операции над автоматами. В качестве таких операций будем использовать:

- случайное порождение автомата — в каждом состоянии создается случайный линейный бинарный граф;
- скрещивание автоматов — линейные бинарные графы в соответствующих состояниях скрещиваются;
- мутация автомата — в линейном бинарном графе, который соответствует случайному состоянию, производится мутация.

При этом считается, что число состояний в автомате фиксировано. Поэтому противоречий при выполнении определенных таким образом операций не возникает.

Определим теперь генетические операции над линейными бинарными графами. Для скрещивания линейных бинарных графов можно применить любой из известных методов скрещивания строк, отдельно скрещивая части, описывающие нетерминальные и терминальные узлы. Например, это может быть одноточечный или двухточечный кроссовер [1]. При этом часть, соответствующая терминальным узлам, должна быть скопирована полностью из одной родительской хромосомы. В ином случае возможна ситуация, когда терминальная часть будет содержать только нули либо единицы.

Операция скрещивания для строк разной длины должна быть модифицирована, так как в ином случае возможна ситуация, когда получившийся линейный бинарный граф будет некорректен: ребра, скопированные из одного из родителей могут вести в несуществующие узлы. Количество узлов, скопированных из второй хромосомы должно быть достаточным для корректности линейного бинарного графа. При скрещивании линейных бинарных графов одинаковой длины модификация операции скрещивания не требуется.

Операция мутации может быть определена следующим образом: выбирается произвольный нетерминальный узел, после этого в него вносится одно из следующих изменений:

- замена переменной, которой помечен модифицируемый узел;
- замена значения переменной, которым помечен переход в следующий в порядке нумерации узел;
- замена номера узла, в который производится переход при несовпадении значения переменной.

3.5. Задание ограничений на целевой автомат

Рассмотрим возможность задания ограничений на автомат, представленный линейными бинарными графами. Заметим, что число терминальных узлов линейного бинарного графа соответствует числу исходящих ребер. Кроме того, число нетерминальных узлов линейного бинарного графа может служить оценкой сложности функции переходов из фиксированного состояния. Эти характеристики автомата могут быть внесены в функцию приспособленности автомата для задания ограничений.

3.6. Особенности метода

По сравнению с методом представления функции переходов деревьями решений недостатком метода является то, что упрощение функции переходов в процессе эволюции происходит более медленно. Этот эффект может быть частично нейтрализован внесением числа узлов в функцию приспособленности.

Отметим, что метод является компромиссом между компактным представлением функции переходов и представлением функции переходов в виде строк. Он сочетает простоту генетических операций (так как операции производятся фактически над строками), но представляет функции переходов из состояний в виде достаточно сложного объекта. Наиболее близким подходом является программирование с экспрессией генов [11]. Особями в этом алгоритме генетического программирования являются префиксные записи выражений, хранящиеся как строки фиксированной длины. Для того, чтобы по строке всегда можно было построить выражение, вводится требование корректности, заключающееся в том, чтобы с определенного места в строке встречались только терминалы. Заметим, что возможна ситуация, когда выражение строится по некоторому префиксу строки, оставшаяся часть которой ничему не соответствует.

Выводы по главе 3

1. Предложен новый метод представления функций переходов автоматов управления для использования в генетическом программировании, основанный на линейных бинарных графах.

2. Рассмотрена возможность задания ограничений на генерируемый автомат.
3. Проведен анализ особенностей метода.

Глава 4.
Реализация

В настоящей главе описывается программное средство *AutoAnt* для генерации автоматов управления с помощью генетического программирования, реализующее предложенные в работе методы.

4.1. Программное средство *AutoAnt*

Программное средство *AutoAnt* предназначено для написания генетических алгоритмов и исследования задачи об умном муравье и ее модификаций, рассмотренных в работе [12]. Данное программное средство было разработано автором совместно с Ю. Д. Бедным. Средство доступно по адресу `http://neerc.ifmo.ru/svn/automata/autoant`. Программное средство обладает следующими достоинствами:

- программные интерфейсы просты для использования;
- допускается использование программного средства из приложений, написанных на языке *Java*;
- программное средство содержит реализацию различных эволюционных алгоритмов для решения исследуемых задач;

На рис. 26 приведена диаграмма основных классов программного средства, которые находятся в пакете *ru.ifmo.ctddev.autoant.ga*.

Эти классы являются параметризуемыми(*generic*), что позволяет варьировать способ представления и реализовывать различные варианты эволюционных алгоритмов.

Опишем методы класса *Configuration*:

- *getChromosomeLength()* — возвращает размер хромосомы;
- *getCrossover()* — возвращает функтор, реализующий генетический оператор скрещивания;

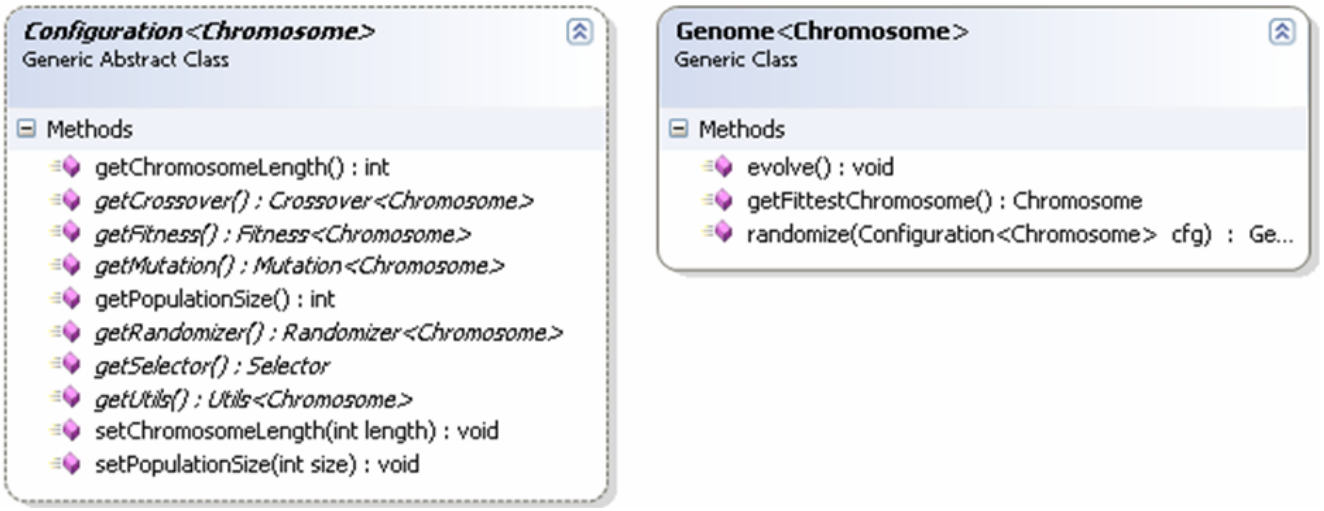

Рис. 26. Диаграмма основных классов программного средства *AutoAnt*

- *getFitness()* — возвращает функтор, реализующий подсчет значения функции приспособленности хромосомы;
- *getPopulationSize()* — возвращает размер популяции;
- *getRandomizer()* — возвращает функтор, реализующий генетический оператор создания случайной особи;
- *getSelector()* — возвращает функтор, реализующий генетический оператор выбора особей для кроссовера;
- *getUtils()* — возвращает вспомогательный объект, инкапсулирующий функциональность создания пустых хромосом, копирования хромосом и их сравнения.

Опишем методы класса *Genome*:

- *randomize(Configuration<Chromosome> configuration)* — создает популяцию случайных особей типа *Chromosome*, настроенную в соответствии с переданной конфигурацией configuration;
- *evolve()* — производит шаг эволюционного алгоритма, генерируя следующее поколение особей;
- *getFittestChromosome()* — возвращает хромосому особи популяции, обладающую наибольшим значением функции приспособленности.

Рассмотрим более подробно интерфейсы функторов генетических операторов. Диаграмма интерфейсов генетических операций приведена на рис. **??**.

Приведем краткую характеристику этих интерфейсов:

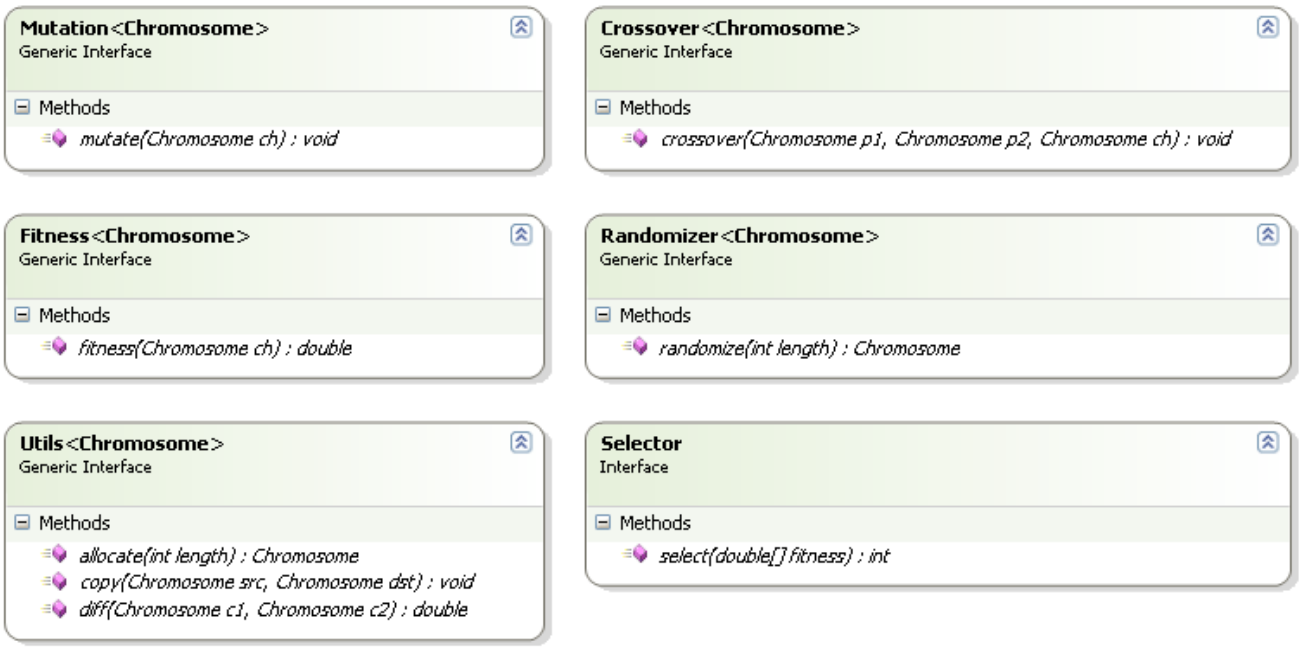

Рис. 27. Диаграмма интерфейсов генетических операций

- *Mutation* — интерфейс мутации;
- *Crossover* — интерфейс скрещивания, принимающего две особи и записывающего результат в третью;
- *Fitness* — интерфейс функции приспособленности;
- *Randomizer* — интерфейс создания случайной особи;
- *Utils* — интерфейс утилит, позволяющих копирование особей, их сравнение и создание пустых хромосом.
- *Selector* — интерфейс отбора, выбирающего особь для размножения по набору значений фитнесс-функции.

Для реализации собственного эволюционного алгоритма необходимо выполнить следующие шаги:

1. Реализовать класс–хромосому – представление сущности в виде особи эволюционного алгоритма.
2. Реализовать классы, наследующие интерфейсы генетических операций над классом-хромосомой.
3. Реализовать класс, наследующий класс *Configuration*.

После этого класс конфигурации может быть использован для создания генома особей, с которым могут выполняться генетические алгоритмы. Про-

граммное средство содержит готовые реализации следующих алгоритмов генерации автоматов:

- генетические алгоритмы над битовыми строками;
- генетические алгоритмы над таблицами переходов;
- метод генетического программирования, использующий представление автоматов деревьями решений;
- метод генетического программирования, использующий представление автоматов бинарными линейными графами.

Реализация всех перечисленных методов осуществляет шаг генетического алгоритма следующим образом:

- производится вычисление функции приспособленности;
- производится отбор особей для размножения. В качестве стратегии используется элитизм: выбирается определенная доля особей, имеющая максимальное значение функции приспособленности. Конкретное значение коэффициента отбора является настраиваемым параметром;
- все отобранные особи переходят в следующее поколение;
- пока в следующем поколении недостаточно особей, две случайных особи, отобранных для размножения, скрещиваются. После этого с определенной вероятностью к порожденной особи применяется операция мутации. Особь, полученная в результате, переходит в следующее поколение. Значение вероятности мутации является настраиваемым параметром.

4.2. Использование программных интерфейсов *AutoAnt*

Программное средство *AutoAnt* предоставляет программные интерфейсы (*API*) для генерации автоматов, что позволяет использовать его из других программ, реализованных на языке *Java*. Для разных методов предоставляются отдельные программные интерфейсы, так как они имеют различные настройки, передаваемые в качестве аргументов.

Рассмотрим сначала использование программного средства *для генерации автоматов, представленных деревьями решений*. Диаграмма классов, представляющих соответствующий программный интерфейс, приведена на рис. 28.

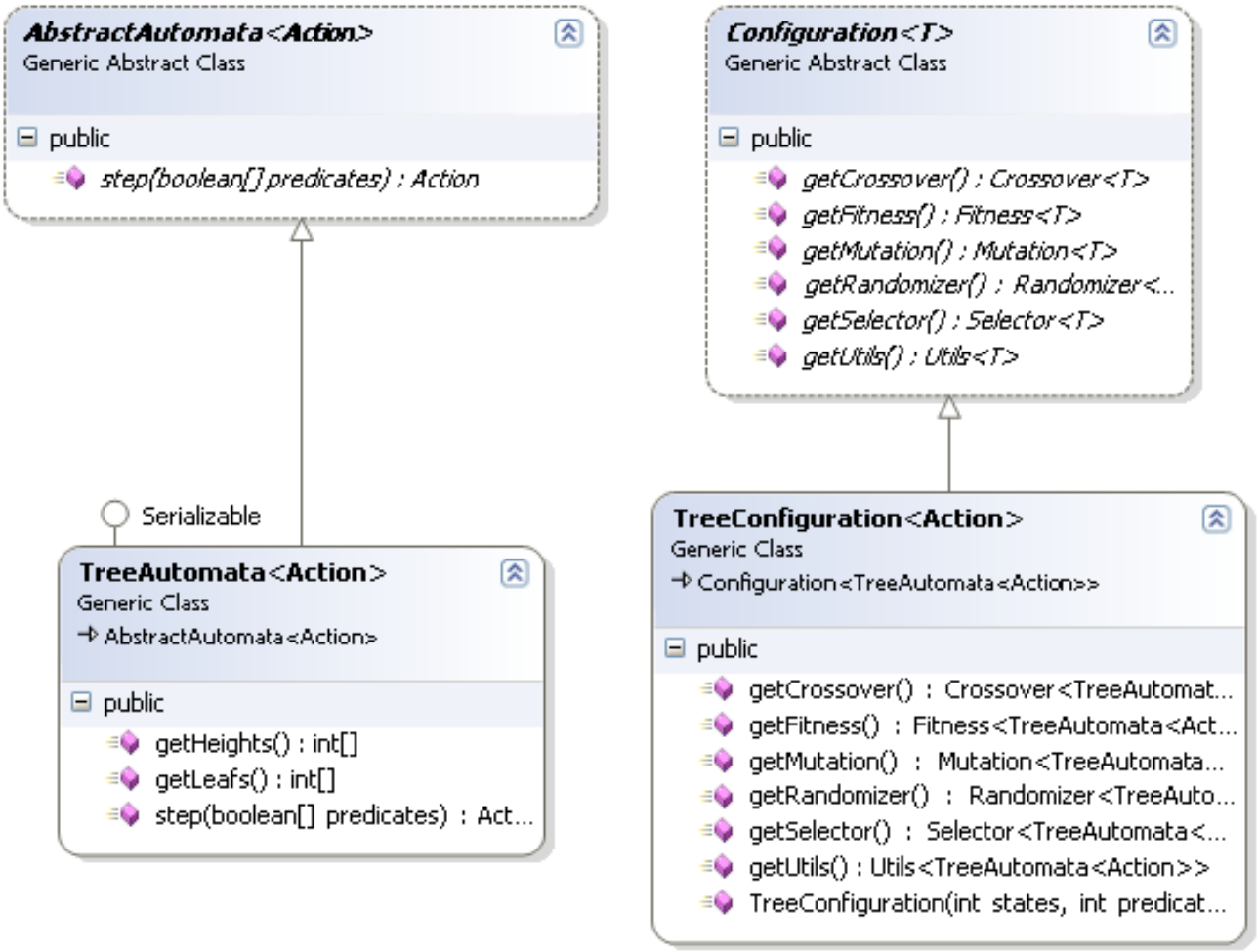

Рис. 28. *AutoAnt API* для генерации автоматов, представленных деревьями решений

Класс *TreeConfiguration* является готовой реализацией конфигурации алгоритма генетического программирования для построения автоматов, представленных деревьями решений. Этот класс является *generic*–классом, параметром которого является класс выходных воздействий автомата.

Класс *TreeAutomata* соответствует автомату, представленному деревьями решений. Класс является сериализуемым (реализует *java.lang.Serializable*), что позволяет сохранять и читать его из файла, а также передавать по сети. Метод *getHeights* возвращает массив, в котором для каждого состояния записана высота соответствующего дерева решений, метод *getLeafs* — массив, в котором для каждого состояния записано число листьев в соответствующем дереве решений.

Опишем порядок применения программных интерфейсов для генерации автоматов. Для этого программисту необходимо выполнить следующие шаги:

- реализовать класс *Action* выходных действий автомата, который требуется сгенерировать;
- реализовать функтор вычисления функции приспособленности, реализующий интерфейс *Fitness<TreeAutomata<Action»*.

Опишем сценарий использования *AutoAnt* из программного кода. Для генерации автоматов, соответствующих поставленной задаче, требуется выполнить следующие действия:

- создать конфигурацию алгоритма генетического программирования, используя конструктор *TreeConfiguration<Action>*, указав в параметрах число состояний, число входных переменных, размер популяции, список возможных выходных действий и реализованную на предыдущем этапе функцию приспособленности;
- создать популяцию автоматов, представленных деревьями решений, с помощью вызова метода *randomize* класса *Genome*, указав в параметре созданную конфигурацию;
- пока не достигнут критерий останова, выполнять шаг алгоритма генетического программирования с помощью вызова метода *evolve*;
- получить сгенерированный автомат вызовом *getFittestChromosome*.

Шаблон описанного сценария приведен ниже:

```
TreeConfiguration<Action> cfg = new TreeConfiguration<Action>(states,
    predicates, actions, size, fitness);

Genome<TreeAutomata<Action>> genome = Genome.randomize(cfg);
while (не выполняется критерий останова) {
    genome.evolve();
}

TreeAutomata<Action> result = genome.getFittestChromosome();
```

Теперь рассмотрим программные интерфейсы для генерации *автоматов, представленных линейными бинарными графами*. Диаграмма классов, предоставляющих соответствующий программный интерфейс, приведена на рис. 29.

Класс *LinGraphConfiguration* является готовой реализацией конфигурации алгоритма генетического программирования для построения автоматов, представленных бинарными линейными графами. Этот класс является *generic*–классом, параметром которого является класс выходных воздействий автомата. Класс *LinGraphAutomata* соответствует автомату, представленному линейными

Рис. 29. *AutoAnt API* для генерации автоматов, представленных линейными бинарными графами

бинарными графами. Метод *getLengths* возвращает массив, элементы которого соответствуют числу узлов линейного бинарного графа для каждого из состояний.

Сценарий использования программных интерфейсов для генерации автоматов, представленных линейными бинарными графами, аналогичен приведенному выше.

4.3. Установка и настройка программного средства *AutoAnt*

В данном разделе описывается установка и настройка программного средства *AutoAnt*.

4.3.1. Описание дистрибутива программного средства

Дистрибутив программного средства *AutoAnt* включает:

- *Java*–архив *autoant-ga.jar*, содержащий реализацию алгоритмов генетического программирования;
- *Java*–архив *autoant-framework.jar*, содержащий реализацию задачи об умном муравье [12];
- командный файл *autoant.cmd*;
- стандартный файл настроек *autoant.properties*.

Для запуска генерации автоматов в командном режиме необходимо запустить командный файл *autoant.cmd*.

Для установки программного средства *AutoAnt* необходимо распаковать архив дистрибутива в папку, в которую производится установка программного средства.

4.3.2. Настройка программного средства

Настройки, необходимые для работы программного средства, должны быть перечислены в файле *autoant.properties* в следующем формате:

```
property_name=property_value;
```

где `property_name` — свойство, а `property_value` — его значение.

Описание настроек приведено в табл. 2. Для всех свойств, не имеющих значения по умолчанию, значения должны быть явно указаны в файле настроек.

Таблица 2. Настройки программного средства *AutoAnt*

Свойство	Описание	Значение по умолчанию
`population.size`	Размер популяции	200
`target.fitness`	Целевое значение функции приспособленности	—
`states.count`	Число состояний генерируемого автомата	8
`predicates.count`	Число входных переменных	—
`populations.max`	Максимальное число популяций	100
`roulette.ratio`	Коэффициент отбора особей для размножения	0.25
`mutation.probability`	Вероятность мутации	0.02
`action.class`	Имя класса выходных состояний автомата. Должен являться перечислимым типом(*enum*) языка *Java*	—
`fitness.class`	Имя класса, осуществляющего вычисление функции приспособленности	—
`classpath`	CLASSPATH, используемый при запуске генерации. Должен содержать пути к классам, указанным в свойствах `action.class` и `fitness.class`	текущая директория
`representation`	Метод представления автоматов. Должен принимать одно из значений `trees` или `graphes` для представления деревьями решений или линейными бинарными графами соответственно	`trees`

Генерация будет выполняться, пока не будет построен автомат, имеющий значение функции приспособленности большее либо равное, чем указанное в параметре `target.fitness`, либо пока не будет сгенерировано число популяций, указанное в параметре `max.populations`.

Возможные сообщения об ошибке и методы их устранения приведены в табл. 3.

Таблица 3. Возможные сообщения об ошибках программного средства *AutoAnt*

Ошибка	Метод устранения	Возможная причина
`Mandatory property <Property> not defined`	Не указано значение обязательного свойства `<Property>`	Указать значение свойства `<Property>` в файле настроек
`System could not find the path specified: <Path>`	Путь, указанный в свойстве `classpath`, не существует	Исправить путь, указанный в свойстве `classpath`, на существующий
`Could not load fitness functor class`	Путь к классу, указанному в свойстве `fitness.class`, не указан в свойстве `classpath`	Добавить путь к классу, указанному в свойстве `fitness.class`, в свойство `classpath`
	Неверно указан класс, подсчитывающий функцию приспособленности	Проверить свойство `fitness.class`
`Could not load action class`	Путь к классу, указанному в свойстве `action.class`, не указан в свойстве `classpath`	Добавить путь к классу, указанному в свойстве `action.class`, в свойство `classpath`
	Неверно указан класс выходных действий автомата	Проверить свойство `action.class`
`Action class is not enumerable`	Класс выходных действий автомата не является перечислимым	Сделать класс выходных действий автомата перечислимым($enum$)

Выводы по главе 4

1. Описано программное средство *AutoAnt*.
2. Рассмотрено использование программных интерфейсов *AutoAnt*.
3. Описан дистрибутив программного средства, даны указания по его установке и настройке.

Глава 5.
Сравнение методов с известными

В этой главе выполняется сравнение разработанных методов с известными на задаче «Умный муравей–3» [12]. Выделяются ситуации, когда использование предложенных методов является более предпочтительным.

5.1. Задача «Умный муравей–3»

Приведем описание задачи. Муравей находится на случайном игровом поле. Поле представляет собой тор размером 32×32 клетки. При этом муравей видит перед собой некоторую область (рис. 30).

Рис. 30. Видимые муравью клетки

Еда (яблоки) в каждой клетке располагается с некоторой вероятностью μ . Значение μ является параметром задачи. Игра длится 200 ходов, за каждый из которых муравей может сделать одно из трех действий: поворот налево, поворо направо, шаг вперед. Если после хода муравей попадает на клетку, где есть яблоко, то оно съедается. Целью задачи является построение стратегии поведения муравья, при которой математическое ожидание числа съеденных яблок максимально. Автомат управления муравьем в этой задаче имеет восемь (число видимых клеток) булевых входных переменных — каждая из которых определяет, есть ли еда в клетке, соответствующей переменной.

5.2. Результаты экспериментов

Предложенный подход сравнивался с генетическими алгоритмами, оперирующими над битовыми строками и генетическими алгоритмами, оперирующими над полными таблицами переходов.

Эксперимент заключался в сравнении полученных значений функции приспособленности (числа съеденных яблок) за фиксированное число шагов с одинаковыми настройками. Запуск алгоритмов производился со следующими настройками:

- стратегия отбора — элитизм, для размножения отбираются 25% популяции, имеющих наибольшее значение фитнесс-функции;
- частота мутации — 2%;
- размер популяции — 200 особей;
- число популяций — 100;
- функция приспособленности — среднее число съеденных яблок на 200 случайных игровых полях, поля одной популяции совпадают, поля различных популяций различны;
- последнее измерение функции приспособленности осуществлялось на случайном наборе из 2000 игровых полей.

Значения функции приспособленности, полученные в результате экспериментов приведены в табл. 4.

Таблица 4. Результаты экспериментов

μ	0.04			
Количество состояний	2	4	8	16
Битовые строки	**19.11**	18.68	17.47	15.10
Таблица переходов	17.18	15.94	15.03	13.68
Деревья решений	18.28	**20.28**	18.60	**20.18**
Линейные бин. графы	18.82	16.44	**19.75**	15.46

Анализ результатов показывает, что предложенные методы являются более эффективными при $n \geqslant 4$ по сравнению с традиционными представлениями

функции переходов. При этом, в различных ситуациях один из методов оказывается более предпочтительным, чем другой. Определение целесообразности применения того либо иного метода является целью дальнейших исследований.

Отметим, что метод представления автоматов деревьями решений был кроме того независимо протестирован в работе [13] на примере задачи управления моделью беспилотного летательного аппарата. Исследования подтвердили эффективность предложенного метода.

Выводы по главе 5

1. Проведено сравнение разработанных методов с известными на примере задачи «Умный муравей–3».
2. Установлено, что предложенные методы бывают эффективнее известных.

Заключение

Основные результаты работы состоят в следующем:

1. Предложено два метода представления функций переходов автомата для использования в генетическом программировании.
2. Проведен анализ разработанных методов и рассмотрена возможность задания ограничений на генерируемый автомат.
3. Совместно с Ю. Д. Бедным реализован проект *AutoAnt*, включающий разработку среды для реализации генетических алгоритмов на примере задачи о муравье.
4. Выполнено сравнение методов с известными. Показано, что предложенные методы бывают эффективнее известных.

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. *Данилов В. Р.* Метод представления автоматов деревьями решений для использования в генетическом программировании //Научно-технический вестник СПбГУ ИТМО. Выпуск 53. Автоматное программирование, с. 103–108.

3. *Данилов В. Р., Шалыто А. А.* Метод генетического программирования для генерации автоматов, представленных деревьями решений /Труды научно-технической конференции "Научное программное обеспечение в образовании и научных исследованиях". СПб.: СПбГПУ. 2008, с. 174–181.

4. *Данилов В. Р.* Метод генетического программирования для генерации автоматов, представленных деревьями решений /Труды V Межвузовской конференции молодых ученых. СПб.: СПбГУ ИТМО. 2008.

5. *Данилов В. Р., Шалыто А. А.* Метод генетического программирования для генерации автоматов, представленных деревьями решений /Сборник докладов XI Международной конференции по мягким вычислениям и измерениям (SCM'2008). СПб.: СПбГЭТУ. 2008, с. 248–251.

6. *Данилов В. Р.* Программное средство AutoAnt для генерации автоматов, представленных деревьями решений /XXXVIII научная и учебно-методическая конференция профессорско–преподавательского и научного состава СПбГУ ИТМО. СПб.: СПбГУ ИТМО. 2009.

7. *Данилов В. Р., Шалыто А. А.* Метод представления функции переходов деревьями решений для генерации автоматов /V Международная конфе-

ренция «Интегрированные модели и мягкие вычисления в искусственном интеллекте». Коломна. 2009.

8. *Данилов В. Р., Шалыто А. А.* Представление функции переходов линейными бинарными графами при генерации управляющих автоматов с помощью генетического программирования /Научно-практическая конференция студентов, аспирантов, молодых ученых и специалистов «Интегрированные модели, мягкие вычисления, вероятностные системы и комплексы программ в искусственном интеллекте». Коломна. 2009.

9. *Данилов В. Р., Шалыто А. А.* Представление функции переходов линейными бинарными графами при генерации управляющих автоматов с помощью генетического программирования /III Всероссийская научная конференция «Методы и средства обработки информации». М.: МГУ. 2009.

Список литературы

1. *Гладков Л. А, Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит, 2006.

2. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб: Питер, 2009.

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

4. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A.* The Genesys System. UCLA, 1992.
`http://www.demo.cs.brandeis.edu/papers/ep93.pdf`

5. *Angeline P. J., Pollack J.* Evolutionary Module Acquisition / Proceedings of the Second Annual Conference on Evolutionary Programming. 1993.
`http://citeseer.ist.psu.edu/angeline93evolutionary.html`

6. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Применение генетического программирования для генерации автоматов с большим числом входных переменных // Научно-технический вестник СПбГУ ИТМО. Выпуск 53. Автоматное программирование. 2008, с. 24–42.

7. *Шеннон К.* Работы по теории информации и кибернетике. М.: Иностранная литература, 1963.

8. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volumes I. II, III. CRC Press, 1999.

9. *Bryant R. E.* Graph-based algorithms for boolean function manipulation // IEEE Transactions on Computers. 1986, pp. 677–691.

10. *Шалыто А. А.* Логическое управление. Методы аппаратной и программной реализации. СПб.: Наука, 2000, 780 с.

11. *Ferreira C.* Gene Expression Programming: A New Adaptive Algorithm for Solving Problems //Complex Systems. 2001. Vol. 13, issue 2, pp. 87–129. http://www.gene-expression-programming.com/webpapers/GEP.pdf

12. *Бедный Ю. Д., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». http://is.ifmo.ru/works/_ant.pdf

13. *Давыдов А. А., Соколов Д. О., Царев Ф. Н.* Применение генетического программирования и методов сокращенных таблиц переходов и деревьев решений для построения автоматов управления моделью беспилотного летательного аппарата // Научно-технический вестник СПбГУ ИТМО. Выпуск 53. Автоматное программирование. 2008, с. 60–79.