САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Ю. Д. Бедный

Применение генетических алгоритмов для генерации автоматов при построении модели максимального правдоподобия и в задачах управления

Магистерская диссертация

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

Санкт-Петербург
2008

Оглавление

ВВЕДЕНИЕ

Генетические алгоритмы являются одним из бурно развивающихся и перспективных направлений в искусственном интеллекте и программировании. С их помощью могут быть найдены решения многих сложных и интересных задач в различных областях. Например, автоматическое создание программ, задачи оптимального управления, регрессионный анализ и эмпирическое прогнозирование, теория игр, символьное интегрирование и дифференцирование, проектирование мультиагентных систем и клеточных автоматов, моделирование эволюционных процессов, предсказание структуры сложных биологических объектов и создание нейронных сетей. К сожалению, в нашей стране этой области исследований уделяется мало внимания. Одна из *задач* данной работы – хотя бы частично восполнить указанный пробел.

Часто автоматы обладают сложным поведением, как, например, в задачах управления, рассматриваемых в настоящей работе. В таком случае их проектирование является нетривиальной и трудоемкой задачей. Кроме того, статическое определение автоматов (**полное** их задание на этапе написания программы) не обладает достаточной гибкостью. Зачастую, предпочтительней создавать автоматы «на лету», не накладывая ограничений сверху на количество их состояний и сложность поведения. Возникает естественное желание – *автоматизировать процесс построения автоматов* и сделать его *динамическим*, поручив основную работу компьютеру.

Однако возможно ли автоматизированное построение автоматов? Каким способом оно может быть осуществлено? В данной работе *генетические алгоритмы* выбраны *как метод проектирования автоматов*. Вначале работы кратко излагаются ключевые концепции генетических алгоритмов и рассматривается ряд задач, в которых генетические алгоритмы успешно применялись для построения автоматов. На основе проведенного обзора для дальнейшего исследования выбираются две наиболее интересные (с точки

зрения автора) области применения генетических алгоритмов для построения автоматов – *построение моделей максимального правдоподобия* и *решение задач оптимального управления*. Эти области подробно исследуются в следующих двух главах. В действительности первая глава носит самостоятельный (обзорный) характер и не является необходимой для понимания результатов работы, изложенных в последующих главах.

В начале второй главы устанавливается неприменимость алгоритма Баума-Велша для построения некоторого класса задач скрытых марковских моделей. Далее рассматривается метод, в котором генетические алгоритмы используются для выбора начальных параметров алгоритма Баума-Велша. Такая модификация приводит к значительному улучшению результатов работы алгоритма Баума-Велша. Автором предлагается применять рассмотренный метод для решения практической задачи – нахождения ошибок в автоматах. Рассматривается один тип возможных ошибок – неучтенный переходы. Поставленная задача успешно решается. При этом производится сравнения и с другими возможными подходами, например, с алгоритмом случайного поиска.

Третья глава посвящена решению задач оптимального управления. Сначала выполняется описание этого типа задач на интуитивном уровне. Далее выполняется формальная постановка. Затем указываются проблемы, возникающие при решении поставленной задачи. В частности раскрываются недостатки автоматного подхода. После проведенного анализа предлагается новый метод решения задач оптимального управления, основанный на автоматном подходе. При этом генетические алгоритмы используются для автоматического построения автоматов. Для выяснения практической применимости предложенного метода производится его апробация при построении системы управления танком для популярной компьютерной игры *Robocode*. Результаты проведенных экспериментов указывают на достаточно высокую эффективность предлагаемого метода.

В заключение работы перечислены ее основные результаты. Также приводятся список открытых вопросов и список публикаций.

ГЛАВА 1. ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ ПОСТРОЕНИЯ АВТОМАТОВ

В начале данной главы приводится краткое описание генетических алгоритмов [1, 2]. Далее рассмотривается ряд задач, в которых применение генетических алгоритмов для автоматизированного построения автоматов [3] позволило улучшить существующие решения или получить решения, сравнимые по эффективности с существующими. Кроме того, следует принять во внимание, что трудозатраты при подходе, базирующемся на генетических алгоритмах, как правило, значительно меньше, чем при эвристическом построении автоматов. Важно отметить и то обстоятельство, что в некоторых случаях автоматизировано построенные автоматы, могут быть изучены, для выявления структуры «хороших» решений задачи.

Анализ, проводимый в данной главе необходим для более глубокого понимания области исследований, выявления интересных и перспективных направлений, в которых может быть сделано что-то новое и полезное. В действительности автором было изучено б*о*льшее число задач в данной области, чем рассматривается в настоящей главе. В настоящей главе рассматриваются наиболее интересные и малоизвестные задачи.

1.1. Генетические алгоритмы

Для природы характерна оптимальность и простота структуры различных биологических объектов, а также эффективность их работы. Многих исследователей давно привлекал вопрос – возможно ли эффективное построение важных и полезных систем на основе принципов биологической эволюции?

В 1966 г. Л. Фогель, А. Оуэнс и М. Уолш предложили схему эволюции автоматов, решающих задачи прогноза. В 1975 г. вышла книга Дж. Холланда “Адаптация в естественных и искусственных системах”, в которой был предложен генетический алгоритм. Эти работы заложили основы эволюционного программирования.

Генетические алгоритмы – это оптимизационный метод, базирующийся на эволюции популяции «особей». Каждая особь характеризуется приспособленностью – функцией ее генов. Задача оптимизации состоит в максимизации функции приспособленности. В процессе эволюции – в результате отбора, рекомбинаций и мутаций геномов особей происходит поиск особей с высокими приспособленностями.

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

Более подробное описание генетических алгоритмов и их разновидностей может быть найдено в работах [1 – 5].

Перейдем к рассмотрению задач.

1.2. Итерированная дилемма узника (теория игр)

В теории игр известна следующая бескоалиционная матричная игра с ненулевой суммой, обычно называемая «Дилемма узника» («*Prisoner’s dilemma*»). Двое преступников пойманы и допрашиваются в отдельных камерах. Срок тюремного заключения, который получит каждый из них, зависит как от его показаний, так и от показаний соучастника. В табл. 1 приведены сроки заключения в зависимости от решений, принятых игроками – их стратегий.

Таблица 1. Матрица игры «Дилемма узника»

	Сознаться	Отрицать
Сознаться	3, 3	0, 5
Отрицать	5, 0	1, 1

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

В 1980-ом году Роберт Аксельрод рассмотрел вариант данной игры, названный им «*Iterated Prisoner's Dilemma*», в котором между игроками проводится несколько партий и каждый игрок помнит историю трех предыдущих игр. Аксельрод организовал турниры, на которые программу мог прислать любой желающий. В первом турнире участвовало 14 программ, во втором 63. Результат программы в турнире равнялся количеству очков, набранных против остальных участников. Проведенные турниры показали [6], что «жадные» стратегии (предпочитающие сознаться, что являлось оптимальной стратегией в исходной игре «Дилемма узника») имеют низкую результативность. Победителем же обоих турниров стала простая программа, названная *Tit for Tat*, состоящая всего из 4 строк на языке BASIC, присланная Анатолием Рапопортом, которая первым ходом выбирала стратегию «отрицать», а дальше делала тоже, что оппонент на предыдущем ходу.

В 1987 году Аксельрод изучил возможность эволюционного порождения стратегий с помощью генетических алгоритмов. Стратегия программы кодировалась битовой строкой из 70 символов. В результате была получена

стратегия, дающая лучший чуть результат, чем *Tit for Tat* при фиксированных соперниках. В дальнейшем Аксельрод провел исследование, в котором среда (оппоненты) также эволюционировала. Было замечено, что в ходе первых двух десятков поколений преобладали «жадные» стратегии (сознаться), однако в ходе дальнейшей эволюции они уступали место «кооперирующимся» (подобным *Tit for Tat*). Наилучшие стратегии, полученные в ходе данного исследования, также являлись различными модификациями *Tit for Tat*.

В 1991 году Д. Фогель провел ряд экспериментов, в которых эволюционное программирование использовалось для вывода **автоматов**, кодирующих стратегии игроков [7]. На рис. 1 приведен пример автомата, кодирующего стратегию игрока. На данном рисунке действию стратегии «отрицать» соответствует символ *C*, а стратегии «сознаться» символ *D*. Стартовое состояние (состояние 1 на рисунке) выделено стандартным образом. Каждый переход помечен дугой вида (*P1*, *P2*)/*A*. Здесь *P1* – действие (стратегия) игрока на предыдущем ходу (C либо D), *P2* – действие оппонента на предыдущем ходу, *A* – действие игрока на текущем ходу. Стратегия игрока на первом ходу обозначена пунктирной линией.

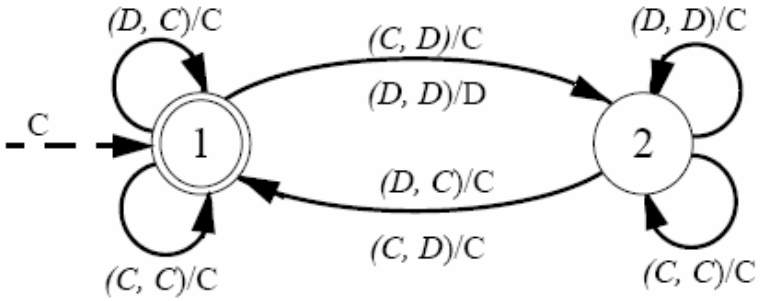


Рис. 1. Автомат, кодирующий стратегию игрока

Таким образом, особь эволюционного алгоритма являлась автоматом. Число состояний автомата находилось в диапазоне от одного до восьми. На каждом этапе эволюционного алгоритма (поколении) проводился турнир между особями популяции, на основе которого вычислялась функция

приспособленности. Затем каждый автомат из лучшей половины популяции путем одной из шести мутаций порождал одного потомка. Возможными мутациями являлись: добавление состояния, удаление состояния, изменение перехода, изменение действия на переходе, изменение начального состояния и изменение действия игрока на первом ходу.

Результаты экспериментов Фогеля в целом совпадают с результатами Аксельрода, с той лишь разницей, что процесс эволюции шел немного быстрее – «кооперирующиеся» стратегии занимали место «жадных» после первых 5 – 10 поколений.

1.3. Выбор праймера для полимеразной цепной реакции (биология)

В работе [8] авторы предлагают новой подход к решению известной в молекулярной биологии задачи выбора праймера для полимеразной цепной реакции (ПЦР) [9]. Естественным способом проверки пригодности праймера является попытка проведения ПЦР, однако в случае неудачи такая реакция приведет к существенным как материальным, так и временным затратам. Поэтому желательно иметь способ определения пригодности праймера без проведения ПЦР. Существует ряд критериев пригодности праймера для ПЦР, перечисленных в [9], благодаря которым могут быть построены различные эвристические методы проверки праймера.

В рассматриваемой же работе [8] предлагается принципиально другой подход, в котором эволюционный алгоритм используется для построения классификатора праймеров – **конечного автомата**, который позволяет в ряде случаев предсказывать, подходит ли данный праймер для проведения ПЦР. Конечный автомат строится на основе множества заранее известных тренировочных данных – праймеров, для которых известно подходят ли они для проведения ПЦР или не подходят.

Особью генетического алгоритма является конечный автомат с выделенным начальным состоянием. В экспериментах, описываемых авторами, автоматы имеют 64 состояниями. Множество входных воздействий – множество всех подмножеств множества нуклеотидов {A, T, G, C}. Состояния автомата бывают трех типов: хорошие, плохие и промежуточные. На рис. 2 приведен пример автомата описанного вида, с той лишь разницей, что для простоты изображено только пять состояний. Приведенные типы состояний обозначены символами «+», «–» и «?» для хороших, плохих и промежуточных типов состояний соответственно.

Для предсказания пригодности праймера необходимо подать его на вход автомату, считая суммарное количество хороших и плохих состояний, в которые переходит автомат по мере увеличения принимаемого префикса. Если количество плохих состояний превосходит количество хороших, то праймер считается не подходящим. Если количество плохих состояний меньше количества хороших, то считается что праймер подходит для ПЦР. В случае же равенства не дается ответа на вопрос классификации.

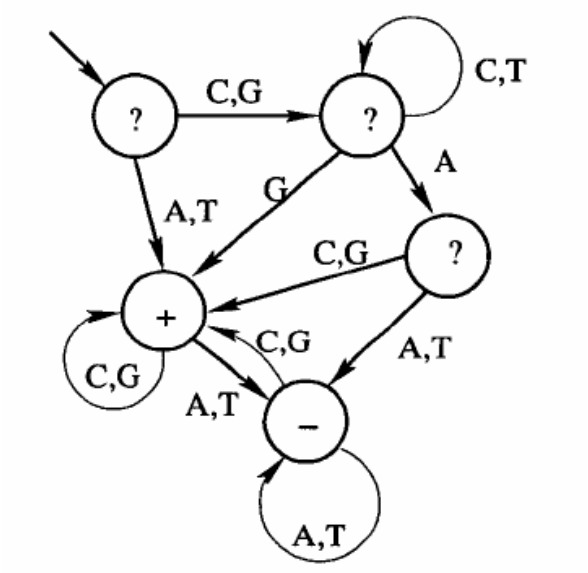


Рис. 2. Структура автомата, классифицирующего праймеры

Для задания функции приспособленности f автомата A используется тренировочное множество T из 695 праймеров, 440 из которых подходят для проведения ПЦР, а 255 не подходят. Пусть $p \in T$ – некоторый праймер, p_i – префикс данного праймера длины i : $len(p_i) = i$, $s(p_i)$ – состояние, в которое переходит автомат A, принимая на вход префикс p_i, $g(p_i) := g'(type(s(p_i)), good(t))$ – функция, возвращающая некоторое целое число в зависимости от $type(s(p_i))$ – типа состояния, в которое переходит автомат по префиксу p_i, и $good(t)$ – является ли праймер подходящим для ПЦР. Функция g', используемая в эксперименте, приведена в табл. 2.

Таблица 2. Функция g'

Праймер	Тип состояния		
	+	–	?
Подходящий	3	-4	0
Не подходящий	-5	7	0

Данная функция означает прирост приспособленности при увеличении длины префикса и соответственно переходе автомата в следующее состояние. При этом функция приспособленности автомата $f(A) = \sum_{p \in T} \sum_{i=1}^{len(p)} g(p_i)$.

В экспериментах, проведенных авторами, использовалась популяция из 600 автоматов. Каждый автомат как особь популяции кодировался битовой строкой. Эволюционный алгоритм содержал 100 поколений. Начальное поколение автоматов создавалось случайным образом. После вычисления функции приспособленности на элементах популяции, она разбивалась случайным образом на группы по четыре особи (автомата). Две наименее приспособленные особи в каждой четверке заменялись особями, полученными в результате рекомбинации двух наиболее приспособленных особей. Для рекомбинации применялась двухточечная рекомбинация строк. После

рекомбинации осуществлялась мутация, которая в каждом десятом случае изменяла начальное состояние, в 30% случаев изменяла переход по одному из входных воздействий и в 60% случаев изменяла тип состояния.

Авторы провели две серии экспериментов. В первой серии экспериментов эволюционный алгоритм был выполнен 100 раз. При этом перед каждым запуском начальная популяция инициализировалась случайными автоматами, а после запуска сохранялась наилучшая из полученных особей. Затем, из 100 наилучших особей была образована популяция P . Вторая серия состояла также из 100 экспериментов, однако, в каждом из экспериментов в качестве начальной популяции автоматов использовалась популяция P . Данный подход называется гибридизация.

Для тестирования наилучшего из автоматов, полученных в первой, и во второй серии экспериментов было выбрано тестовое множество, состоящее из 880 подходящих и 471 не подходящих для ПЦР праймеров. Наилучший из автоматов, полученных в результате первой серии экспериментов, показал результаты, приведенные в табл. 3.

Таблица 3. Результаты классификации для наилучшего автомата первой серии экспериментов

Праймер	Результат классификация		
	+	–	?
Подходящий	727	153	39
Не подходящий	208	263	

Авторы отмечают, что данное качество предсказаний считается весьма хорошим. При этом наилучший из автоматов, полученных во второй серии экспериментов, показал еще более хорошее качество классификации.

1.4. Искусственная этология (зоология)

Этология – это дисциплина зоологии, занимающаяся изучением поведения особи в естественной для неё среде. Искусственная этология – направление этологии, в котором особи и среда, в которой они обитают, являются объектом компьютерного моделирования. Преимущество данного подхода состоит в том, что в отличие от реального мира (среды обитания), обладающего огромным количеством факторов в разной степени влияющих на поведение особи, искусственный (виртуальный) мир, может быть создан достаточно простым для изучения. В то же время основные результаты, полученные в рамках упрощающих предположений, могут быть перенесены на реальный мир.

В работе [10] с помощью искусственной этологии изучаются процессы коммуникации между особями в ходе эволюции популяции данных особей. В рассматриваемой работе особи названы симоргами. Авторы работы определили требования к искусственной среде, необходимые для того, чтобы коммуникация имела смысл и была полезна симоргам. Эти требования таковы:

- каждый симорг должен обозревать часть среды обитания, которая недоступна обозрению других симоргов;
- среда обитания должна иметь возможность распространять сигналы посылаемые одним симоргом другому.

В среде обитания, построенной исследователями, каждый симорг находится в своей *локальной среде*, доступной для обозрения только ему. Состояние локальной среды задается элементов из подмножества натуральных чисел $S = [1..8]$ и определяется случайным процессом, то есть не может быть предсказано. Таким образом, единственным способом узнать состояние локальной среды другого симорга является коммуникация. Сообщение представляется элементом *S*. Для передачи сообщений между симоргами вводится *общая среда*, в которую симорг может передавать сообщения и из которой он может принимать сообщения других симоргов. Состояние общей

среды (также элемент *S*) определяется только одним, последним, сообщением, из помещаемых в нее симоргами. Топология среды обитания симоргов приведена на рис. 3.

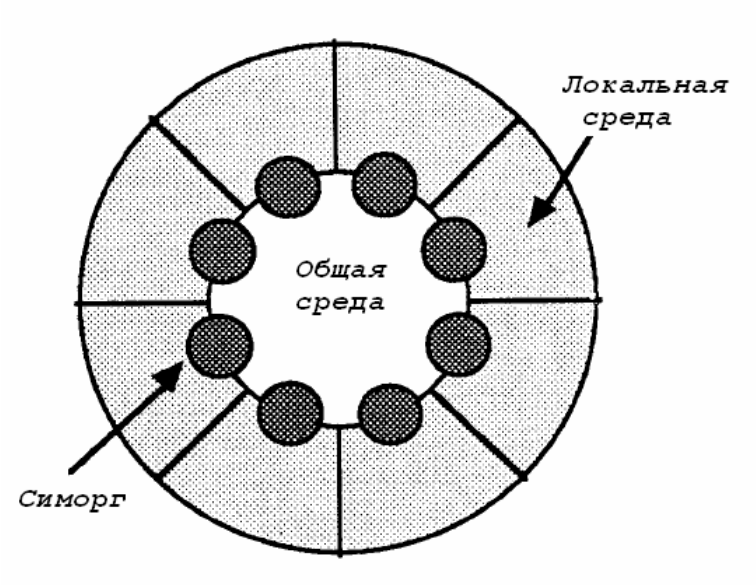


Рис. 3. Топология среды обитания симоргов

Симорг задается простейшим **конечным автоматом**, содержащим всего одно состояние. Следовательно, у симоргов нет памяти. Множеством входных воздействий автомата является декартово произведение множества состояний общей среды и множества состояний локальной среды (S^2). Определены два вида выходных воздействий автомата: *сообщение* (элемент *S*) и *действие* (также элемент *S*). Выходное воздействие первого вида изменяет состояние общей среды, а второго вида – обеспечивает «выживаемость» симорга. Оно должно быть как можно более «эффективно». Если действие симорга совпадает с состоянием общей среды (которое равно состоянию локальной среды последнего из симоргов отправивших сообщение), то «приз» (одно очко) получает как симорг последним отправивший сообщение, так и симорг, действие которого совпало с этим сообщением.

Функция приспособленности симорга вычисляется как сумма очков, набранных им в течение пяти раундов. В начале каждого раунда локальные среды симоргов инициализируется случайными значениями. Затем симорги по

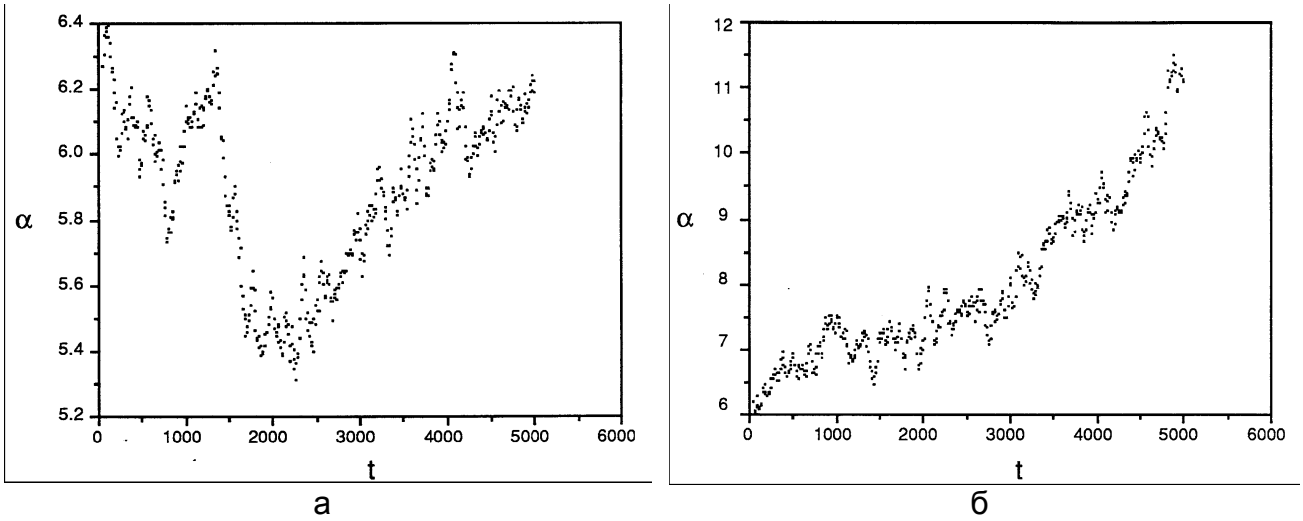
очереди (например, по часовой стрелке) осуществляют выходные воздействия. В зависимости от выходного воздействия симорга происходит изменение общей среды, добавление ему очков, а также тому симоргу, который последним отправил сообщение. При этом в течение раунда каждый симорг осуществляет десять входных воздействий.

На каждой итерации эволюционного алгоритма из популяции удаляется один симорг, затем выбираются два симорга для рекомбинации, в результате которой в популяцию добавляется новый симорг. Выбор симорга для удаления происходит обратно пропорционально его функции приспособленности, а выбор для рекомбинации – пропорционально данной функции.

Автомат, определяющий симорга, как особь генетического алгоритма, представляется в виде битовой строки, которая образуется следующим образом. Для каждого входного воздействия – пары (состояние общей среды, состояние локальной среды) в лексикографическом порядке в строку добавляется выходное воздействие. Выходное воздействие представляется в виде пары чисел, первое из которых равно нулю либо единице в зависимости от того является ли воздействие действием либо сообщением, а второе равно действию (сообщению). Таким образом, длина строки, кодирующей автомат, составляет $8 \times 8 \times 2 = 128$ символов, каждый из которых является число из диапазона [0, 8]. Используемый оператор рекомбинации – двухточечная рекомбинация на строках. После рекомбинации с низкой вероятностью осуществлялась мутация полученной особи.

Исследователи определили размер популяции равным 100. Эволюционный алгоритм содержал 5000 итераций (поколений). За приспособленность популяции было взято среднее значение функции приспособленности на элементах популяции. Для изучения влияния коммуникации на приспособленности популяции было проведено два эксперимента, в первом из которых возможность коммуникации отсутствовала – общая среда постоянно изменялась случайным образом.

На рис. 4 приведены зависимости функции приспособленности популяции (α) от количества итераций эволюционного алгоритма (t) для случая, когда возможность коммуникации отсутствовала (а) и когда коммуникация была разрешена (б).



а б

Рис. 4. Результаты экспериментов

Результаты экспериментов показали, что приспособленность популяции в случае возможности осуществления коммуникации между особями значительно превосходит приспособленность в отсутствии таковой.

1.5. Задача оптимизации

В работе [10] предлагается подход, позволяющий свести задачу глобальной оптимизации к задаче управления. Процесс нахождения глобального максимума сводится к перемещению по дискретизированному пространству поиска. Для управления перемещением вводится автомат Мили. Постановка задачи такова: дано некоторое множество $\Omega \subset R^2$, на нем определена функция $\Phi : \Omega \to R_+$. Перемещение объекта управления по пространству Ω описывается следующим образом: $x_{t+1} = x_t + f(\Phi(x_t), \Phi(x_{t-1}))$, где $x_t \subset \Omega$, функция $f : R_+ \times R_+ \to \{x \mid x_t + x \in \Omega\}$ – стратегия перемещения. Таким образом, перемещение на текущем шаге (в текущий момент времени) зависит

от значений функции Φ, исследованных на прошлом и позапрошлом шагах. Пусть $T \subset N, T > 1$ – время, отведенное на перемещение по пространству Ω. Задачей является максимизация функционала $\Psi(f,T) = \frac{\upsilon(\Phi(x_T)+1)}{\max\limits_{0\le t\le T}\Phi(x_t)+1} + \max\limits_{0\le t\le T}\Phi(x_t)$ по стратегии перемещения f. Таким образом, производится поиск стратегий, которые с одной стороны должны искать глобальный максимум (этому требованию соответствует второе слагаемое в сумме), а с другой стороны к окончанию поиска текущее положение x_T, должно быть «недалеко» от максимального из исследованных (этому требованию соответствует первое слагаемое). Константа υ в числителе первого слагаемого устанавливает приоритет между первым и вторым требованием.

Функция $\Psi(f,T)$ является функцией приспособленности для генетического программирования. Стратегия перемещения f представляется в виде изображенном на рис. 5.

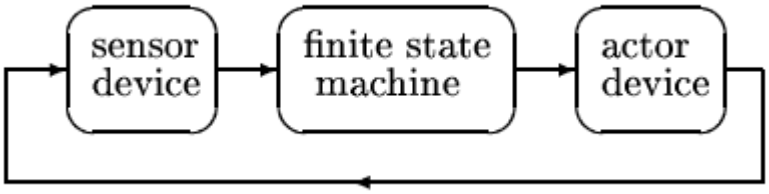


Рис. 5. Структура стратегии перемещения

Sensor device (преобразователь) переводит непрерывные значения $\Phi(x_t), \Phi(x_{t-1})$ из R_+ во множество X дискретных входных воздействий автомата Мили A, реализуя функцию $\sigma : R_+^2 \to X$. Данная функция наряду с автоматом A будет выводиться генетическим алгоритмом.

Actor device (устройство перемещения) переводит выходные воздействия автомата в команду по перемещению объекта управления по двумерной сетке, дискретизирующей Ω. Устройство перемещения хранит информацию о текущем и предыдущем положении устройства управления x_t и x_{t-1}, а также

вектор перемещения d_t, который определяет как направление перемещения, так и величину шага, которая может быть равна шагу сетки, либо удвоенному шагу сетки. В зависимости от выходных воздействий автомата устройство перемещения оставляет устройство управления на месте $x_{t+1} = x_t$, либо перемещает его в одно из четырех ортогональных направлений, при этом шаг перемещения может остаться неизменным, либо удвоиться, либо уменьшиться вдвое. Определяется множество возможных перемещений $Y = \{F, B, L, R, S, 2F, \frac{1}{2}F\}$, элементы которого соответствует перемещению вперед, назад, влево, вправо, вперед с удвоением шага, вперед с уменьшением шага вдвое.

Finite state machine (автомат) A задается множеством входных воздействий X, действий Y, состояний S, начальным состоянием $s_1 \in S$, функцией переходов автомата $\delta : X \times S \to S$ и функцией действий автомата $\gamma : X \times S \to Y$.

Традиционное генетическое программирование конструирует функции σ, δ, γ одновременно – дерево, являющееся особью ГП, определяет сразу три функции. Вершины дерева удовлетворяют следующей грамматике:

cond ::= ≥0(<arith>,{<rule>},{<rule>})

rule ::= γ(<state>,<output>) | δ (<state>,<state>) | <cond>

arith ::= <arith> + <arith> | <arith> - <arith> | <arith> <arith> | <arith> % <arith> | A | B | $c \in R$

state ::= $s \in S$

output ::= $y \in Y$

Дерево параметризуется константами *A, B*, которые при вычислении заменяются на $\Phi(x_t), \Phi(x_{t-1})$. Корнем дерева является *cond* выражение. Пример дерева выражений приведен на рис. 6. Для наглядности иллюстрации в данном дереве не отображена функция γ. Результатом вычисления выражения приведенного вида после подстановки значений $\Phi(x_t), \Phi(x_{t-1})$ являются конечное

множество переходов (<state>,<state>) и конечное множество выходных воздействий (<state>,<output>), которые определяют функции δ, γ. Если в множестве переходов (выходных воздействий) присутствуют два различных перехода (действия) из одного состояния (на рис. 6 при $A \geq B$ в множестве переходов есть два перехода из состояния 0: в состояние 1 и в состояние 2) выбирается самый левый из них (в приведенном примере будет выбран переход (0, 1)).

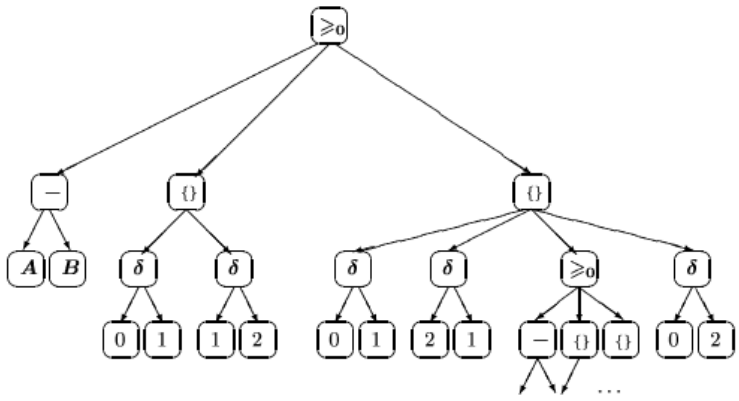


Рис. 6. Дерево выражений

Генетические операции (рекомбинация, мутация) определены таким образом, чтобы при их выполнении порождались корректные деревья. Используются три дополнительные генетические операции над вершинами, являющимися списком инструкции (данные вершины обозначены символом {} на рис. 6): удаление поддерева, добавление поддерева, изменение порядка на поддеревьях.

Эксперименты проводились на трех различных функциях Ф, приведенных ниже. Параметр n принимал значение равное двум, $v = 10$, $T = 50$, а константы μ_i были выбраны таким образом, чтобы $\Phi_i \in [0,1]$. Пространством поиска являлось множество $\Omega := [0,10]^2$. Автоматы содержали десять состояний. Популяция состояла из 10 автоматов, для каждого из которых случайным

образом выбиралось $x_0 \in \Omega$ – положение объекта управления в начальный момент времени.

$$\Phi_1(x) := -50\mu_1 \sum_{i=1}^{n} x_i \sin\sqrt{50x_i}$$

$$\Phi_2(x) := \mu_2 \sum_{i=1}^{n} \left(\tfrac{1}{4} x_i^2 - 10\cos(\pi x_i) + 10\right)$$

$$\Phi_3(x) := \mu_3 \sum_{i=1}^{n-1} 100\left(\tfrac{1}{5} x_{i+1} - \tfrac{1}{25} x_i^2\right)^2 + \left(\tfrac{1}{5} x_i - 1\right)^2$$

Графики функций **Ошибка! Объект не может быть создан из кодов полей редактирования.** и **Ошибка! Объект не может быть создан из кодов полей редактирования.** приведены на рис. 7.

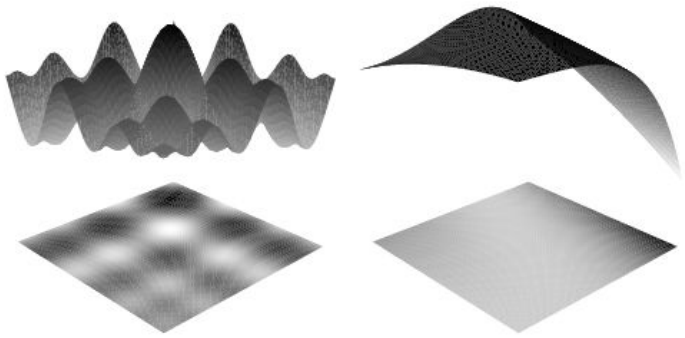


Рис. 7. Тестовые функции приспособленности Φ_1 и Φ_3

В результате экспериментов традиционное генетическое программирование сгенерировало стратегии перемещения f, для которых значение функции приспособленности близко к максимальному (единице). Кроме того, найденные стратегии являются устойчивыми как к изменению позиции x_0 объекта управления в начальный момент времени, так и к замене функции приспособленности Φ на другую, обладающую аналогичными свойствами.

1.6. Управление человекоподным роботом (роботехника)

Создание человекоподобных роботов одно из наиболее актуальных и перспективных направлений, как науки, так и технологии. Роботы такого типа могли бы выполнять различные задачи в нашем мире, «приспособленном под человека». Одной из задач роботехники является создание двуногих роботов, способных передвигаться подобно человеку.

В работе [12] предлагается использовать традиционное генетическое программирование для эволюционного построения автомата, осуществляющего управление движением робота *Elvina.* Данный робот изображен на рис. 8.

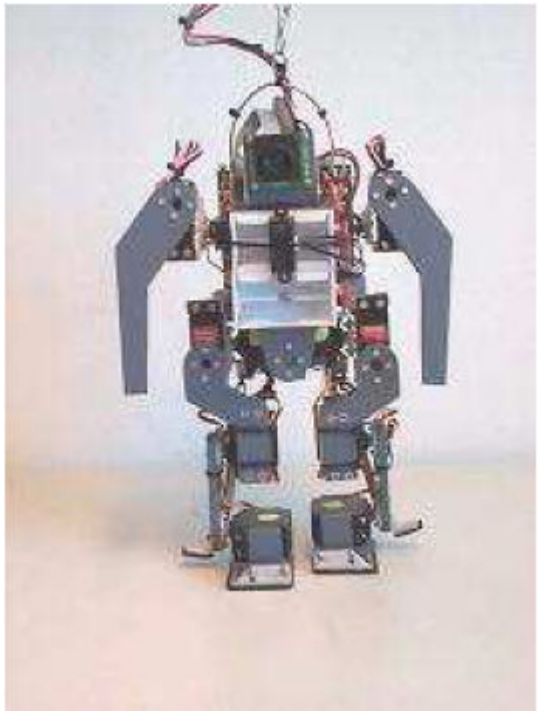


Рис. 8. Человекоподобный робот *Elvina*

Каждая из ног робота *Elvina* имеет 5 степеней свободы (одна из которых пассивная). Голова туловище и руки имеют одну степень свободы. Таким образом, в сумме у робота 14 степеней свободы. Двигатели, изменяющие положение частей тела (ног, рук, головы, туловища), управляются контроллером, выдающим одно из 256 значений 0..255 определяющих положение управляемой части тела. Для управления ходьбой необходимо

задать последовательность состояний робота, осуществляющую один цикл ходьбы.

Неподвижное состояние робота задается вектором $J_P = [j_1, j_2, ..., j_k]$, где k – число его степеней свободы. Тогда пространство W всех состояний робота (включая те, в которых он находится во время перемещения частей тела) есть линейная оболочка векторов $J_1, J_2, ...J_j$, где $j = 256^k$. Некоторые из этих состояний не являются статически устойчивыми – переход в такое состояние приводит к падению работа. Выделяют подпространство G векторов из W, определяющих статически устойчивые положения робота. Для осуществления одного цикла ходьбы достаточно задать m устойчивых состояний $P_1, P_2, ..., P_m$, в которых последовательно будет находиться робот в процессе движения. Поскольку пространство поиска достаточно велико, авторы строят функцию перехода $f(j_1, j_2, ..., j_k) = [j_1', j_2', ..., j_k']$, которая принимает текущее состояние робота и возвращает состояние, в которое необходимо перейти. **Данная функция реализует автомат без входных воздействий, действиями на переходах в котором являются команды двигателям частей тела об изменении положения.**

Для построения искомой функции используется традиционное генетическое программирование. Вводятся $2k$ регистров, половина из которых соответствует входным данным $j_1, j_2, ..., j_k$, а половина – выходным $j_1', j_2', ..., j_k'$. Как входные, так и выходные регистры изначально инициализируются значениями $j_1, j_2, ..., j_k$. Особь алгоритма представляет собой набор инструкций, записанных последовательно, осуществляющих операции над содержимым регистров. Каждая инструкция – четверка (*k1*, *k2*, *op*, *res*). Первые два параметра – номера регистров; третий параметр – одна из пяти функций, аргументами которой являются первые два параметра; последний параметр – выходной регистр, в который следует запись результат операции. После выполнения последней инструкции значения выходных регистров копируются

в $j'_1, j'_2, ..., j'_k$. Авторы определили пять функций двух аргументов: ADD (сложение), SUB (вычитание), MUL (умножение), DIV (безопасное деление), SINE (синус первого аргумента, второй аргумент отбрасывается).

Функция приспособленности вычислялась путем моделирования поведения робота в течение, примерно, 20 секунд. Записывались начальные координаты частей тела и конечные, затем значение функции принималось равным $fitness = W[1 - \frac{h_{start}}{h_{stop}}] - (d_{left} + d_{right})$. Здесь $\frac{h_{start}}{h_{stop}}$ – отношение начальной высоты положения робота к конечной, а $d_{left} + d_{right}$ – отклонение робота от прямолинейной траектории, W – некоторый постоянный коэффициент. Таким образом, предпочтение отдавалось роботам, способным поддерживать высоту положения (тем более не падать) и как можно меньше отклоняющимся от прямолинейной траектории движения.

В результате применения генетического алгоритма, авторам удалось вывести автомат, управляющий двуногим роботом, справляющегося с задачей прямолинейного человекоподобного движения.

Выводы по главе 1

В первой главе был рассмотрен ряд задач из различных областей, в которых генетические алгоритмы использовались для построения автоматов. Настоящая глава призвана показать, что генетические алгоритмы применимы для построения автоматов в самых разных областях, а также выявить те области, в которых такое применение наименее изучено.

На основании рассмотренных задач (лишь небольшая часть из которых приведена в первой главе) для подробного исследования были выбраны две области: построение моделей максимального правдоподобия и решение задач оптимального управления. Следующие главы содержат результаты исследований применимости генетических алгоритмов для построения автоматов в этих областях.

ГЛАВА 2. ГЕНЕРАЦИЯ АВТОМАТОВ ПРИ ПОСТРОЕНИИ МОДЕЛИ МАКСИМАЛЬНОГО ПРАВДОПОДОБИЯ

В настоящей главе проводятся исследования одного класса скрытых марковских моделей [13] – «сильно детерминированных». Выясняется, что для построения модели максимального правдоподобия для таких моделей традиционно используемый алгоритм Баума-Велша [14] не применим. В то же время, использование генетических алгоритмов для выбора начальных параметров позволяет *значительно* повысить эффективность алгоритма Баума-Велша. В действительности демонстрируется, что ключевая роль при построении модели максимального правдоподобия отводится генетическим алгоритмам.

Кроме того, выполняется постановка прикладной задачи, в которой «сильно детерминированные» скрытые марковские модели естественным образом применяются для поиска ошибок функционирования программно реализованных автоматов. Рассматривается ошибки определенного вида – неучтенные переходы между состояниями автомата. Для выявления ошибок анализируются последовательности выходных воздействий автомата на некоторых последовательностях входных воздействий. На основе этого анализа строится модель максимального правдоподобия автомата, которая сравнивается с заданным автоматом.

Изложение, однако, с целью сделать его более доступным и интересным, построено таким образом, что сначала формулируется указанная задача, а затем на ее примере исследуется применимости алгоритма Баума-Велша для построения моделей максимального правдоподобия для скрытых марковских моделей обладающих «сильной детерминированностью».

2.1. Идейное описание задачи

В последнее время всё шире применяется автоматное программирование [3]. Так как автоматные программы обычно используются во встроенных системах, то важно обеспечить корректное поведение автоматов.

Это может быть достигнуто за счет верификации [15] и тестирования. Данная работа посвящена разновидности тестирования, основанной на применении скрытых марковских моделей. Их использование позволяет на основе анализа выходов при фиксированных входах предсказать, какие ошибки содержит автомат, и как часто они проявляются. Предлагаемый подход имеет то преимущество, что он позволяет устанавливать (с некоторой вероятностью) где именно проявляются ошибки, а не только указывает на их наличие. При этом подход не предполагает какой-либо модификации исходного кода, реализующего автомат, например, добавления отладочной информации.

Пусть проводится наблюдение за работой заданного автомата. При этом описание его поведения известно. Однако из-за ошибок, допущенных при реализации, поведение автомата может не соответствовать заданному. В настоящей главе будем полагать, что исходное состояние и выходные воздействия в каждом состоянии определены достоверно. Кроме того, предполагается, что рассматриваемый автомат является автоматом Мура. Единственный компонент описания автомата, потенциально содержащий ошибки – *граф переходов* (матрица переходов). При этом предполагается, что ошибки могут быть только одного типа – реализованные условия переходов не соответствуют условиям в заданном графе – при заданном входе переход не должен выполняться, а он иногда выполняется. Ошибки такого типа назовём «неучтенными переходами».

Пусть задан граф, в котором переход между вершинами с номерами, например, ноль и три, помечен условием x_1, а при его реализации была допущена ошибка – это условие было реализовано как $x_1 \vee x_2$. При этом при $x_1=0$ переход выполняться не должен, а он иногда выполняется (при $x_2=1$). На

рис. 9 для входного воздействия $x_1=0$ пунктиром условно показан переход, который выполняться не должен, но после реализации иногда выполняется.

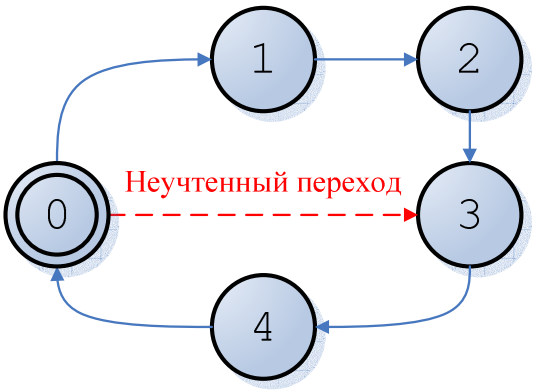


Рис. 9. Автомат с неучтенным переходом

В общем случае наблюдать за автоматом будем, подавая ему последовательность входных воздействий и анализируя последовательность выходных воздействий. Если заданный граф переходов реализован с ошибками указанного выше типа, то при некотором входе выход будет отличаться от ожидаемого. Задача состоит в том, чтобы, имея набор входных последовательностей и соответствующих им выходных последовательностей, обнаружить неучтенные переходы, а также выяснить, как часто по ним осуществляется переход.

2.1.1. Пример

Приведем пример, поясняющий сказанное (рис. 10).

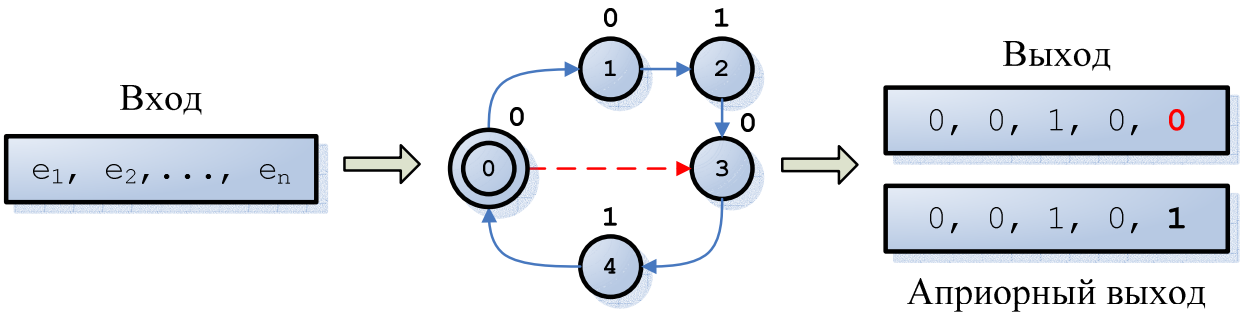
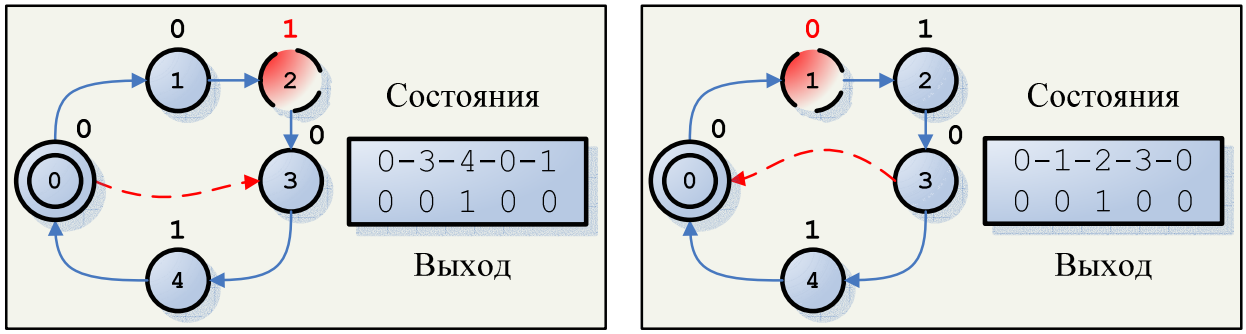


Рис. 10. Наблюдение за автоматом

На этом рисунке изображены пять шагов работы (переходов) автомата Мура, в котором выходные воздействия в состояниях указаны над соответствующими вершинами. Начальное состояние имеет номер ноль. Автомат принимает вход – последовательность входных воздействий $e_1, e_2 \ldots, e_n$. Непрерывными дугами изображены переходы, которые должны выполняться на данном входе, а пунктирной дугой – неучтенный переход (выполняемый на первом шаге). Последовательность состояний автомата при выполнении неучтенного перехода `0-3-4-0-1`, в то время как ожидаемая (априорная) последовательность состояний `0-1-2-3-4`. При этом выход автомата `0,0,1,0,0`, в то время как ожидаемый выход `0,0,1,0,1` (последние символы данных последовательностей отличаются).

Имея реальный и априорный выходы, весьма сложно установить, где была допущена ошибка. Для рассматриваемого примера, добавив переход `(0,3)`, можно согласовать теоретический выход автомата с наблюдаемым. Однако это же справедливо и для перехода `(3,0)` при последовательности состояний `0-1-2-3-0`. Действительно, последовательность выходных воздействий в обоих случаях имеет вид: `0,0,1,0,0` (рис. 11).



а) б)

Рис. 11. Восстановление матрицы переходов

Для того чтобы определить, какой из этих переходов более вероятен, рассмотрим ещё один шаг работы автомата, генерирующего за первые пять шагов последовательность выходных воздействий `0,0,1,0,0`. Автомат,

изображенный на рис. 11 слева, после пятого шага находится в состоянии один, и на шестом шаге переходит в состояние два (0-3-4-0-1-**2**). Автомат, изображенный справа – после пятого шага находится в состоянии ноль, а на шестом шаге переходит в состояние один (0-3-4-0-1-**1**). Таким образом, первый автомат на выходе формирует последовательность 0,0,1,0,0,**1**, в то время как второй – 0,0,1,0,0,**0**. Следовательно, по шестому элементу выходной последовательности можно сделать вывод о том, добавление какого перехода (3,0) или (0,3) **не** приведет к модели реального автомата. Если шестой элемент оказался равен нулю, то модель, изображенная на рис. 11, а, не соответствует наблюдаемому выходу, в другом случае – поведение автомата не может описываться моделью, приведенной на рис. 11, б.

В рассмотренном примере удалось установить, какая модель заведомо не соответствует наблюдениям («имеет нулевое правдоподобие»). Однако это не всегда возможно. Как правило, требуется из всех возможных моделей выбрать модель, имеющую наибольшее правдоподобие – максимальную вероятность наблюдения. Формализуем рассматриваемую задачу.

2.2. Постановка задачи

Будем рассматривать автоматы (с неучтенными переходами) следующего вида. Из каждого состояния существует один переход, выполняемый с больш*о*й вероятностью, и несколько переходов, выполняемые с малыми вероятностями. Переходы последнего типа соответствуют неучтенным переходам. Малые вероятности неучтенных переходов означают, что априорная матрица переходов в достаточной степени соответствует матрице переходов реализованного автомата. Другими словами, ошибки проявляются редко. В работе для простоты анализа накладывается ограничение – из каждого состояния существует не более двух неучтенных переходов (рис. 12).

Естественно также предположить, что граф переходов автомата связен. Обозначим число состояний автомата за N, а матрицу переходов размера N^2 за

$A = (a_{ij})$, где a_{ij} – вероятность перехода из состояния i в состояние j. При этом для $\forall i$ должно выполняться $\sum_{j=1}^{N} a_{ij} = 1$ – сумма вероятностей переходов из каждого состояния в другие равна единице.

Пусть, находясь в каждом из состояний, наблюдается некоторое выходное воздействие из конечного алфавита мощности M. Определим матрицу $B = (b_{ij})$ размера NM, где b_{ij} – вероятность наблюдать выходное воздействие j, перейдя в состояние i. При этом для $\forall i$ справедливо $\sum_{j=1}^{M} b_{ij} = 1$ – сумма вероятностей получения каждого из выходных воздействий равна единице. Для детерминированных автоматов каждому состоянию i соответствует единственное j, такое что $b_{ij} = 1$.

Пусть начальное состояние автомата имеет номер s. Зададим вектор $\pi = (\delta_{is})^T$, *s-ый* элемент которого равен единице, а остальные нулю. Определим модель, соответствующую наблюдаемому автомату, как $\lambda = (A, B, \pi)$.

На рис. 12 приведен пример модели некоторого автомата.

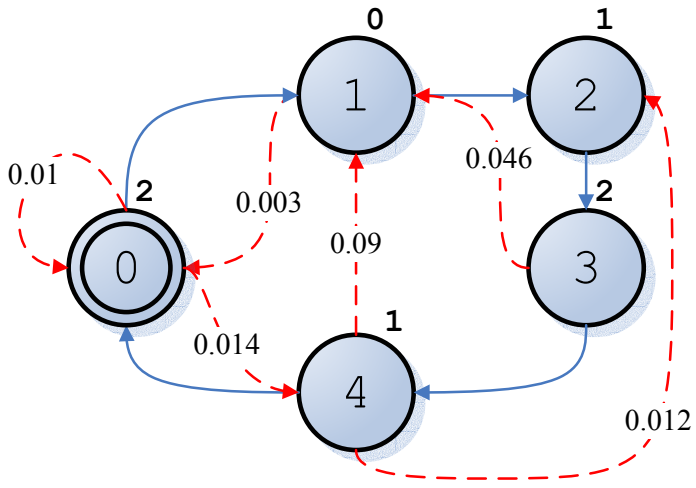


Рис. 12. Пример модели автомата

Эта модель $\lambda = (A, B, \pi)$ задаётся следующим образом:

$$A=\begin{pmatrix} 0.01 & 0.976 & 0 & 0 & 0.014 \\ 0.003 & 0 & 0.097 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0.046 & 0 & 0 & 0.954 \\ 0.908 & 0.09 & 0.012 & 0 & 0 \end{pmatrix},\ B=\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix},\ \pi=\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Выполним K раз T шагов работы автомата. При этом получим множество последовательностей выходных воздействий $O=\left\{O^{(1)}, O^{(2)},..., O^{(K)}\right\}$. Каждая из них имеет вид: $O^{(i)}=O_1^{(i)}O_2^{(i)}\cdots O_T^{(i)}$.

Пусть $P(O^{(i)}\mid\lambda)$ – вероятность того, что автомат, который имеет модель λ, за T шагов генерирует последовательность выходных воздействий $O^{(i)}$. Тогда $P(O\mid\lambda)=\prod_{i=1}^{K}P(O^{(i)}\mid\lambda)$ – вероятность наблюдения множества последовательностей O (при некотором фиксированном порядке его элементов).

Задача, рассматриваемая в настоящей главе, состоит в поиске матрицы переходов A, для которой $P(O\mid\lambda)$ принимает максимальное значение. Другими словами, необходимо найти $A'=\arg\max_{A}P(O\mid\lambda)$. Модель, полученная при решении поставленной задачи (модель максимального правдоподобия), выявляет неучтенные переходы, а также показывает, как часто они используются на данном наборе последовательностей входных воздействий.

2.3. Скрытые марковские модели

Таким образом, поставлена цель – разработать метод, который позволяет построить модель автомата, наиболее точно соответствующую набору последовательностей выходных воздействий при фиксированных последовательностях входных воздействий. Для решения этой задачи предлагается рассматривать искомую модель автомата λ, как *скрытую марковскую модель*. Данный подход представляется естественным, так как

скрытая марковская модель описывается таким же образом, как и модель автомата – в виде тройки $\lambda = (A, B, \pi)$ – и имеет тот же смысл.

Интерпретация модели автомата, как скрытой марковской модели, позволяет применять алгоритм Баума-Велша для построения модели максимального правдоподобия. Выбор этого алгоритма обусловлен тем, что он (являясь разновидностью *EM* алгоритма [16]), как правило, находит хорошее приближение к оптимальному решению при построении некоторой скрытой марковской модели максимального правдоподобия.

Далее, однако, показывается, что алгоритм Баума-Велша неэффективен при решении рассматриваемой задачи (что объясняется свойствами матрицы переходов). Затем описывается метод, основанный на совместном использовании этого и генетических алгоритмов.

2.3.1. Использование алгоритма Баума-Велша

В ходе исследований этот алгоритм применялся для решения поставленной задачи. Оказалось, что он «плохо работает» на «сильно детерминированных» моделях. Именно такого типа модели задают автоматы с редко осуществляемыми неучтенными переходами. Под термином «плохо работает» понимается то обстоятельство, что алгоритм Баума-Велша сходится к локальному максимуму, который значительно меньше глобального.

В свою очередь, под «сильно детерминированными» моделями понимаются те модели, компоненты которых (A, B, π) содержат большое число элементов, близких к единице и, как следствие, большое число нулевых элементов. Действительно, матрица A , для каждого состояния i имеет единственный переход, выполняемый с большой вероятностью и возможно небольшое число неучтенных переходов, которые редко проявляются – имеют маленькие вероятности. Можно более формально определить понятие «детерминированности», например, через дисперсию случайных величин

«номер состояния, в которое перейдет автомат из состояния *i*», однако ограничимся приведенным интуитивным пояснением.

В подтверждение того, что рассматриваемый алгоритм неэффективен для моделей указанного вида, рассмотрим одну из таких моделей. Данная модель проста – она соответствует бросанию «детерминированной» монетки, которая поочередно выпадает «орлом» и «решкой». Последовательности, генерируемые рассматриваемой моделью, имеют вид `1010101...` Данная модель может быть формализована следующим образом:

$$\lambda = (A,\ B, \pi),\ A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ B = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ \pi = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Заметим, что указанная модель легко строится человеком, однако при запуске алгоритма Баума-Велша с равномерными начальными параметрами

$$A = \begin{pmatrix} ½ & ½ \\ ½ & ½ \end{pmatrix},\ B = \begin{pmatrix} ½ & ½ \\ ½ & ½ \end{pmatrix},\ \pi = \begin{pmatrix} ½ \\ ½ \end{pmatrix}$$

на последовательности `1,0,1,0,1,0,1`, она не будет получена. Результатом работы этого алгоритма является модель

$$A = \begin{pmatrix} ½ & ½ \\ ½ & ½ \end{pmatrix},\ B = \begin{pmatrix} 0.43 & 0.57 \\ 0.43 & 0.57 \end{pmatrix},\ \pi = \begin{pmatrix} ½ \\ ½ \end{pmatrix},$$

которая «улавливает» лишь то свойство последовательности, что единиц в ней чуть больше чем нулей, а правдоподобие данной модели будет малым.

Известно, что для успешной работы алгоритма Баума-Велша важен выбор начальных параметров. Существуют различные эвристики для решения этой задачи. При их применении удается получить модель с правдоподобием единица для указанной последовательности. Однако для более сложных последовательностей, не удаётся найти универсальную и достаточно простую эвристику.

Для выбора начальных параметров для алгоритма Баума-Велша предлагается использовать генетические алгоритмы, что приводит к построению моделей с больш*и*м правдоподобием.

2.4. Генетические алгоритмы

Выбор начальных параметров с помощью генетических алгоритмов применялся при решении задачи предсказания вторичной структуры белков [20]. При этом в указанной статье определён вид матрицы переходов A.

В рассматриваемой задаче матрица B и вектор π являются фиксированными и не подлежат оптимизации. При этом оптимизируется только матрица A. Учитывая введенные ранее ограничения (из каждого состояния существует не более двух неучтенных переходов), вид матрицы переходов A полностью определяется заданием для каждого состояния i не более двух состояний j_1, j_2, для которых существует скрытый переход (i, j_k). Применим генетические алгоритмы для поиска этих переходов. Затем с помощью алгоритма Баума-Велша определим их вероятности.

Для проверки эффективности предлагаемого метода, случайным образом строится модель, задающая некоторый автомат с неучтенными переходами. Далее, по этой модели генерируется набор последовательностей выходных воздействий. Затем этот набор подаётся на вход алгоритму (использующему генетические алгоритмы и алгоритм Баума-Велша), который строит модель максимального правдоподобия. Полученная модель сравнивается с исходной моделью, оптимизированной алгоритмом Баума-Велша. На основе сравнения будет сделан вывод об эффективности предлагаемого метода. Сравнение проводится с моделью, полученной применением алгоритма Баума-Велша к исходной модели, а не с самой исходной моделью. Причина в том, что при ограниченном размере набора входных последовательностей модель, генерирующая эти последовательности, может иметь несколько меньшее правдоподобие, чем построенная по ним.

Зафиксируем число состояний автомата и число выходных воздействий, положив соответственно $N = 12$ и $M = 3$. Данные параметры выбраны произвольным образом, исходя лишь из тех соображений, чтобы задача построения модели максимального правдоподобия не была слишком простой.

2.4.1. Пространство поиска

Итак, предметом поиска являются матрицы размера 12 на 12, содержащих в каждой строке не более двух ненулевых элементов и единицу на диагонали соседней с главной (связанность графа) очень велико, а именно $\left(N^2/2\right)^N = 72^{12} \approx 1.94 * 10^{22}$. В задачах с таким большим пространством поиска, использование генетических алгоритмов, как правило, является более эффективным, нежели применение других методов. Например, в задаче классификации плотности для клеточных автоматов [17, 18, 19] пространство поиска содержало $2^{128} \approx 3.4 * 10^{38}$ элементов, а наилучший из известных алгоритмов решения данной задачи – генетический алгоритм.

2.4.2. Структура и параметры алгоритма

В экспериментах применялись стандартные генетические алгоритмы [1]. На рис. 13 приведена схема работы стандартных генетических алгоритмов, реализованная в работе.

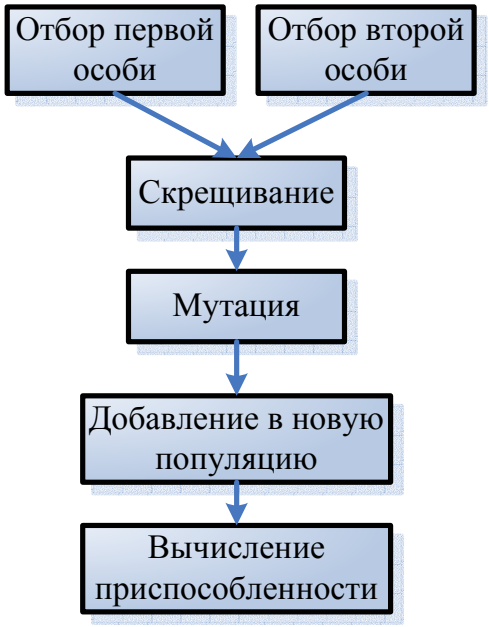


Рис. 13. Один шаг работы алгоритма

Популяция содержала 1000 особей. Процесс оптимизации выполнялся до тех пор, пока не находилось решения, удовлетворяющего поставленной задаче. Далее опишем способ представления особи популяции в виде хромосомы, а также генетические операции скрещивания и мутации, применявшиеся в работе.

2.4.3. Структура хромосомы

Хромосома задается массивом размера N на 2, каждый элемент которой – целое число от 0 до N – 1. На рис. 14 изображена структура некоторой хромосомы.

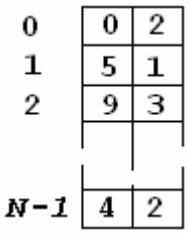


Рис. 14. Структура хромосомы

Как видно из приведенного рисунка, по хромосоме непосредственно восстанавливаются состояния, в которые выполняются неучтенные переходы. Например, для указанного рисунка из нулевого состояния существуют неучтенный переход только во второе состояние, из первого состояния – только в пятое, из второго – в девятое и третье, и так далее.

2.4.4. Генетические операции

В работе применяются стандартные генетические операции – отбор, мутация и скрещивание. Операция отбора организована таким образом, что особи сортируются по убыванию приспособленности, а затем выбираются пропорционально полученному после сортировки номеру. Такой вид отбора называется «*rank selection*» и подробно описан в работе [1]. Отбор пропорциональный значению приспособленности («*roulette wheel selection*») также был опробован, но дал несколько худшие результаты.

Оператор мутации выполняется стандартным образом – с небольшой вероятностью (в проведенных экспериментах она равнялась 0.1) каждое из чисел хромосомы изменяется на целое число от 0 до N – 1. Новое значение может совпадать с предыдущим.

Скрещивание также осуществляется стандартным образом, а именно, применяется одноточечное скрещивание («*single-point crossover*»), рассмотренное в книге [1]. Единственное отличие состоит в том, что с некоторой вероятностью (в проведенных экспериментах равной 0.3) особи не подвергаются скрещиванию, а случайным образом выбирается одна из них и возвращается как результат скрещивания.

2.4.5. Функция приспособленности

Приспособленность $f(x)$ особи x, задающей некоторую модель $\lambda = (A, B, \pi)$, определим как $f(x) = -\frac{1}{\ln P(O \mid \lambda)}$, где знаменатель – логарифм вероятности пронаблюдать данные O в модели λ. Определенная таким образом, функция приспособленности пропорциональна вероятности $P(O \mid \lambda)$. Можно попробовать определить функцию приспособленности просто как $P(O \mid \lambda)$. Однако в соответствии с введенным ранее определением, $P(O \mid \lambda) = \prod_{i=1}^{K} P(O^{(i)} \mid \lambda)$, следовательно, необходимо вычислить $P(O^{(i)} \mid \lambda)$ для каждого i, а затем перемножить полученные значения. Однако $P(O^{(i)} \mid \lambda)$ – очень малые по модулю величины и при их перемножении происходит потеря точности. Поэтому целесообразно перейти к логарифмам: $\ln P(O \mid \lambda) = \ln \prod_{i=1}^{K} P(O^{(i)} \mid \lambda) = \sum_{i=1}^{K} \ln P(O^{(i)} \mid \lambda)$.

Для определения вероятности наблюдать некоторый выход при заданной модели $P(O^{(i)} \mid \lambda)$, используется «*forward-backward procedure*» алгоритм, который описан в работе [14]. Данный алгоритм, по сути, является алгоритмом динамического программирования.

2.4.6. Генерация последовательностей входных воздействий

Построение модели $\lambda = (A, B, \pi)$, генерирующей набор последовательностей выходных воздействий, выполнялось следующим образом. Фиксировалось число состояний автомата *N*, оно равнялось 12. Также задавалось число возможных выходных воздействий *M,* оно равнялось трём. Для каждого состояния i ($i \in [0..N-1]$) случайным образом выбирались два целых числа j_1, j_2 (в диапазоне от 0 до N – 1) и создавались неучтенные переходы (i, j_1) и (i, j_2), выполняемые с небольшими вероятностями (в экспериментах эти вероятности равнялись 0.05). Матрица B и вектор π генерировались случайным образом, с соблюдением условия детерминированности.

Набор входных последовательностей состоял из десяти последовательностей, каждая из которых содержала 200 элементов. Генерации некоторой выходной последовательности выполнялась путем запуска автомата на 200 шагов работы.

2.4.7. Результаты эксперимента

Для каждой исходной модели λ^* , в результате работы генетического алгоритма получены модели, вероятность наблюдения (правдоподобие) которых примерно равна вероятности наблюдения исходной модели, оптимизированной алгоритмом Баума-Велша. В некоторых случаях правдоподобие найденных моделей даже превосходило правдоподобие оптимизированной исходной модели. В табл. 4 приведены значения логарифмов правдоподобия исходной модели $\ln P(O \mid \lambda^*)$ до, и после оптимизации алгоритмом Баума-Велша.

Таблица 4. Правдоподобие исходной модели

Логарифм правдоподобия модели	−690
Логарифм правдоподобия оптимизированной модели	**–678**

В табл. 5 приведены результаты работы генетического алгоритма для некоторой исходной модели.

Таблица 5. Работа генетического алгоритма (процесс эволюции)

Номер поколения	Максимальное значение	Среднее значение	Время, с.
1	−962	$-\infty$	26
2	−858	−890	62
3	−780	−791	79
4	−746	−750	81
5	−725	−732	82
6	−705	−711	76
7	−694	−698	75
8	−685	−691	66
9	−682	−684	65
10	−681	−682	59
11	−680	−681	59
12	−680	−680	62
13	−677	−679	62
14	−677	−677	56
15	−676	−676	52
16	**−675**	−675	53

Для наглядности результатов столбцы таблицы «Максимальное значение» и «Среднее значение» содержат логарифмы правдоподобия моделей $\ln P(O|\lambda)$, а не их приспособленности $f(x) = -\frac{1}{\ln P(O|\lambda)}$. Числовые значения указанных столбцов округлены до ближайшего целого. Среднее значение приспособленности особей популяции на первом шаге положено равным $-\infty$,

так как популяция содержала особь с нулевым правдоподобием (вследствие потери точности), а логарифм правдоподобия такой особи равен $-\infty$.

Из экспериментов (и в частности приведенной выше таблицы) можно сделать вывод о том, что генетические алгоритмы обладают высокой эффективностью при решении рассматриваемой задачи. Так в рассмотренном случае полученная модель имеет даже несколько большее правдоподобие ($\ln P(O \mid \lambda) = -675$), чем оптимизированная исходная (-678). Эффективность также подтверждается тем обстоятельством, что было выполнено всего $16 * 1000 = 16000$ вычислений функции приспособленности (1000 особей в популяции, 16 поколений). Таким образом, исследована лишь *очень* малая часть пространства поиска, содержащего порядка $4.34 * 10^{22}$ особей.

На рис. 15 приведен график, иллюстрирующий процесс эволюции популяции генетического алгоритма. Данный график построен по данным табл. 5.

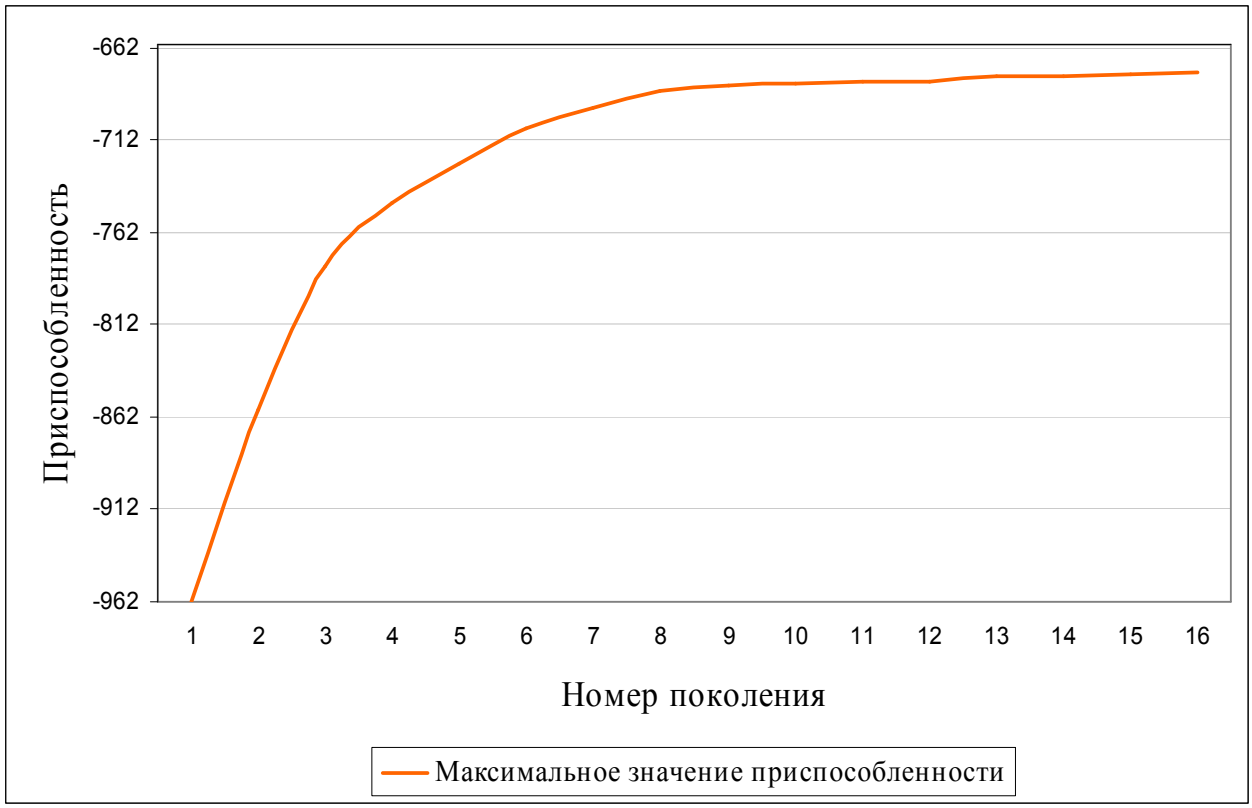


Рис. 15. Процесс эволюции популяции

2.5. Значение генетических алгоритмов

Интересно было бы выяснить – меняет ли оптимизация модели алгоритмом Баума-Велша вид матрицы переходов (добавляя или удаляя дуги), или только подбирает значения вероятностей для переходов, выполняемых с ненулевой вероятностью (найденных генетическими алгоритмами)? В ходе экспериментов алгоритм Баума-Велша *не* изменял вид матрицы переходов.

Таким образом, поиск неучтенных переходов выполнялся генетическими алгоритмами, а частота их использования (в ходе работы автомата) определялась алгоритмом Баума-Велша. Следовательно, основная заслуга в успешном построении моделей максимального правдоподобия для рассматриваемого случая принадлежит генетическим алгоритмам.

2.6. Сравнение с алгоритмом случайного поиска

Чтобы проверить, что рассматриваемая задача достаточно трудна и не может быть решена несложным способом, в работе также рассматривается более простой способ выбора начальных параметров для алгоритма Баума-Велша, а именно – случайный поиск. В этом случае, для каждой строки матрицы A случайным образом (равномерно) выбираются два целых числа (от 0 до N – 1). Для адекватного сравнения с генетическими алгоритмами, количество вычислений функции приспособленности (а значит и количество различных матриц рассмотренных при переборе) положено равным 100 000. В то же время, как указывалось ранее, пространство поиска содержит $4.34 * 10^{22}$ элементов. Поэтому, были основания полагать, что перебор будет малоэффективен при решении данной задачи. Проведенные эксперименты это подтвердили. В табл. 6. приведены результаты работы алгоритмы случайного поиска. Столбец таблицы «Максимальное значение» содержит максимальное значение логарифма правдоподобия рассмотренных моделей.

Таблица 6. Работа генетического алгоритма (процесс эволюции)

Количество просмотренных моделей	Максимальное значение
812	−868
10 939	−857
40 250	−856
59 432	−824
100 000	**−824**

Как видно из приведенной таблицы, перебрав сто тысяч моделей, алгоритм случайного поиска нашёл лишь модель с правдоподобием −824, что значительно меньше правдоподобия оптимизированной исходной модели (−678). В то же время генетические алгоритмы сгенерировали модель с правдоподобием −675.

Чтобы повысить эффективность перебора была рассмотрена следующая его модификация. Поиск осуществляется поэтапно, по мере оптимизации строк матрицы. А именно, для первой строки перебираются все возможные варианты расположения не более двух ненулевых элементов в этой строке (всего таких вариантов $N^2/2$). При этом остальные строки полагаются фиксированными. Из всех возможных вариантов выбирается тот, который обладает максимальным правдоподобием. Затем аналогичный перебор выполняется для второй строки матрицы и так далее вплоть до последней. При таком методе перебора поиск ведется не по всему пространству возможных решений, а только по очень небольшому его подмножеству, а именно просматривается только $N^2/2*N$ вариантов. При этом результаты работы данного метода оказались также значительно хуже результатов, полученных с использованием генетических алгоритмов.

2.7. Обобщение на вероятностные автоматы

В работе проводятся исследования применимости предлагаемого метода для выявления скрытых переходов в *вероятностных автоматах с детерминированной матрицей перехода*. Такого рода автоматы рассматриваются, например, в [21] (*Y-детерминированный* автомат). В автоматах рассматриваемого типа выходное воздействие на переходе определяется вероятностно, в то время как состояние, в которое осуществляется переход, детерминировано.

Для простоты рассматривались автоматы, содержащие всего два выходных воздействия ($M = 2$). Количество состояний автомата N полагалось равным 15. Исходная модель порождала тренировочное множество O, содержащее 50 последовательностей выходных воздействий. Каждая из таких последовательностей имела длину T, равную 200. Матрица переходов генерировалась тем же алгоритмом, что и в эксперименте с детерминированным автоматом, описанным ранее.

При генерации матрицы выходных воздействий применялся следующий алгоритм. Для каждого состояния i, вероятность наблюдения выходного воздействия j выбиралась случайным образом по следующему закону: $b_{ij}^* = |\xi|$, где $\xi -$ с.в. распределенная по стандартном нормальному закону, $\xi \sim N(0, 1)$. Затем для каждого i производилась нормировка по всем j: $b_{ij} = b_{ij}^* / \sum_{j=1}^{N} b_{ij}^*$. Таким образом, соблюдается условие $\sum_{j=1}^{N} b_{ij} = 1$.

Перед решением задачи с помощью генетических алгоритмов, выполнялась ее проверка «на сложность» алгоритмом случайного поиска с ограничением перебора в 100 000 моделей. Как и ожидалось (учитывая размер пространства поиска) алгоритм случайного поиска имеет низкую эффективность – логарифм правдоподобия моделей, найденных таким алгоритмом были меньше логарифма правдоподобия исходной модели.

Предлагаемый метод оказался эффективным и при построении моделей вероятностных автоматов – как правило, удавалось создать модель с правдоподобием не менее чем у исходной модели, просмотрев лишь незначительную часть пространства поиска. Таким образом, предлагаемый метод работает быстрее алгоритма случайного поиска и дает значительно лучшие результаты. В табл. 7 приведены данные сравнительной эффективности алгоритма случайного поиска и предлагаемого метода при построении модели *случайно сгенерированного* автомата с неучтенными переходами. Аналогичные сравнения выполнялись для большого количества автоматов с неучтенными переходами. Результаты этих сравнений также соответствуют приведенным в табл. 7.

Таблица 7. Эффективность метода для вероятностных автоматов с детерминированной матрицей переходов

Модель	Логарифм правдоподобия
Исходная	−6691
Исходная оптимизированная	−6676
Найденная алгоритмом случайного поиска	−6694
Построенная предлагаемым методом	−6675

Выводы по главе 2

В настоящей главе проведен анализ эффективности алгоритма Баума-Велша при построении моделей максимального правдоподобия для автоматов определенного вида – «сильно детерминированных». В результате анализа установлено, что в таких случаях алгоритм Баума-Велша неэффективен. Исследование причины неэффективности приводит к идее использования генетических алгоритмов для выбора начальных параметров алгоритма Баума-Велша. Метод построения модели максимального правдоподобия,

использующий указанную идею, показал высокую эффективность – он выводит модели с больш*и*м правдоподобием. Эффективность этого метода подтверждается также проведенным сравнением с алгоритмом случайного поиска.

Кроме того, предложен новый метод тестирования автоматов, основанный на построении скрытых марковских моделей. Этот метод применим для выявления одного типа ошибок – неучтенных переходов автомата. Достоинством метода является то, что поиск ошибок производится только по результатам работы автомата и не требует изменения его структуры или добавления отладочной информации. В заключение главы выполняется обобщение предложенного метода на вероятностные автоматы с детерминированной матрицей переходов.

ГЛАВА 3. РЕШЕНИЕ ЗАДАЧ ОПТИМАЛЬНОГО УПРАВЛЕНИЯ

В данной главе предлагается новый метод решения задач оптимального управления. Предлагаемый метод основан на автоматном подходе. В то же время генетические алгоритмы используются для автоматического построения автоматов. Метод успешно опробован на практической задаче – построение системы управления танком для популярной компьютерной игры *Robocode* [27]. Результаты, которые показал танк в боях с соперниками, подтверждает эффективность предлагаемого метода.

Глава состоит из трех разделов. В первом разделе производится общая постановка задачи оптимального управления и предлагается метод ее решения. Во втором – предложенный метод применяется для построения системы управления танком. В заключительном разделе подводятся итоги главы.

3.1. Постановка задачи оптимального управления в общем виде

3.1.1. Описание задачи управления

Опишем задачу управления таким образом, чтобы под это описание попал достаточно широкий класс задач. Например, такие задачи как: управления беспилотным летательным аппаратом, управление наземным средством передвижения, различными системами этих средств (двигателем, системой стабилизации т.д.), бытовыми устройствами (лифтом, телевизором, и т.д.), транспортными потоками, виртуальными объектами в играх и моделях (например, танком в игре *Robocode*, футболистами в виртуальном футболе). В то же время, потребуем достаточной формальности выполняемой постановки, чтобы в дальнейшем можно было предложить конкретный подход к ее решению.

Для начала, положим, что существует некоторая среда и объект, взаимодействующий со средой. Под взаимодействием понимается то обстоятельство, что объект влияет на среду, изменяя ее параметры. В свою очередь, среда влияет на объект, изменяя его параметры. Формальное определение термина «параметры» будет дано ниже. Интуитивно под ним следует понимать некоторый набор свойств объекта существенных для задачи управления. Другими словами, параметры объекта (среды) полностью описывают его (ее) состояние.

Рассмотрим, например, задачу управления беспилотным летательным аппаратом. Объект в данном случае – летательный аппарат (самолет, вертолет и т.д.). Среда – воздушное пространство, в котором осуществляется полет, взлетно-посадочная полоса, радиомаяк аэродрома, и т.д. Самолет имеет такие параметры как, например, масса, высота, скорость, ускорение, крен, тангаж, рыскание, положение элеронов, закрылок, руля высоты и т.д. К параметрам среды можно отнести плотность воздуха его скорость, погодные условия, различные вихревые потоки, перечень аэродромов, в которых может быть осуществлена посадка, состояние взлетно-посадочных полос в этих аэродромах, показания радиомаяков и т.д. Влияние среды на самолет очевидно – например, сильное изменение направления и скорости ветра приводит к отклонению самолета от прежнего курса. Влияние объекта на среду для рассматриваемой задачи не столь явно, но также присутствует. Например, вихревые потоки, создаваемые вертолетом, при посадке существенно влияют и на сам вертолет. Другой пример – принятие пилотом решения на посадку в конкретном аэродроме исключает его из списка доступных для посадки для других самолетов на некоторый период времени.

На рис. 16 изображены некоторые параметры самолета: крен, тангаж и рыскание.

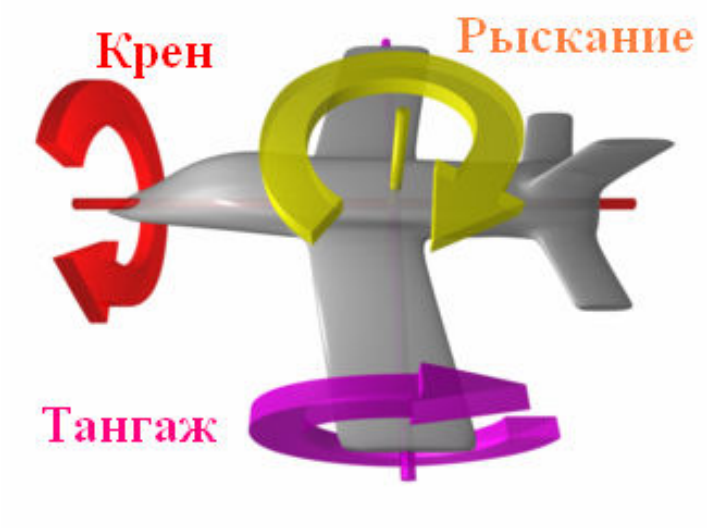


Рис. 16. Некоторые параметры самолета

Будем предполагать, что управление объектом выполняется на определенном временном интервале. Для задачи управления самолетом, это может быть временной интервал между получением разрешения на взлет и завершением посадки. Если же рассматривать задачу управления взлетом вертолета, то в качестве временного интервала естественно взять промежуток времени между разрешением на взлет и моментом, когда высота полета вертолета превысит некоторое пороговое значение. Для простоты формальной постановки задачи управления в дальнейшем, сделаем достаточно сильное предположение о дискретизируемости временного интервала, в котором решается задача управления. Будем считать, что время дискретно и временной интервал, на котором решается задача управления, может быть разбит на некоторое количество минимальных интервалов. Например, в качестве минимального временного интервала может быть выбрана одна секунда, миллисекунда или еще меньший отрезок времени. Величина минимального интервала определяется, как правило, скоростью изменения параметров задачи.

Под управлением предлагается понимать изменение некоторых параметров объекта. Например, приведенная выше задача управления самолетом состоит в воздействии на элероны, рули курса и высоты, изменении

подачи топлива и т.д. При этом в общем случае будут изменяться не только параметры, на которые осуществляется воздействие – *контролируемые*, но и зависящие от них (над которыми контроль непосредственно не осуществляется). Например, воздействие на элероны приводит к изменению крена самолета, управление рулями высоты изменяет тангаж, интенсивность подачи топлива – ускорение. При этом будем считать, что значения контролируемых параметров могут быть мгновенно изменены.

На рис. 17 приведена классификация параметров.

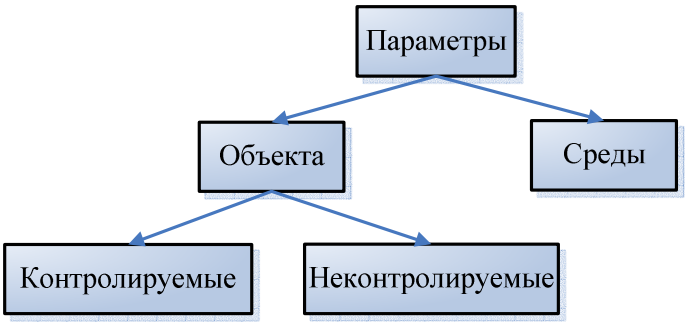


Рис. 17. Классификация параметров

Итак, зафиксируем некоторое конечное множество существенных для задачи управления параметров. В процессе управления объектом параметры (как объекта, так и среды) изменяются во времени. Таким образом, в результате задачи управления получим фазовую кривую (с учетом дискретезации времени – *ломаную*) в пространстве всех возможных параметров. Будем полагать, что существует некоторая функция оценки *качества* решения задачи управления. Эта функция по фазовой кривой, полученной в результате решения задачи управления, возвращает вещественное число, характеризующее то, насколько хорошо решена задача.

Например, для задачи управления самолетом точка фазовой кривой – значения всех параметров (высота самолета, его ускорение, координаты, скорость ветра, и т.д.) в некоторый момент времени между взлетом и посадкой. Функция оценки качества полета может, например, учитывать величину

отклонения самолета от заданного маршрута, экономность расхода топлива, величину перегрузок в процессе полета, «мягкость посадки», безопасность высот, на которых осуществлялся полет.

Положив, что функция оценки фазовой кривой возвращает вещественное число, тем самым было сделано предположение, что два произвольных решения задачи управления всегда сравнимы (по качеству решения задачи). Можно было бы ограничиться требованием частичной упорядоченности решений, однако требование полного порядка на практике является не слишком сильным. Действительно, могут возникнуть трудности при сравнении фазовых кривых, одна из которых соответствует полету в пределах курса, но на опасных высотах, а другая – слишком сильному отклонению от курса, но с соблюдением требования безопасности высоты полета. Однако будем считать, что всегда можно оценить качество полета в совокупности, например, присвоив требованиям (по высоте, курсу и т.д.) определенные веса и просуммировав с ними.

Задача управления состоит в том, чтобы, изменяя значения контролируемых параметров, получить фазовую кривую с максимальным значением качества решения. Как отмечалось ранее, изменение контролируемых параметров приводит и к изменению неконтролируемых параметров объекта. В свою очередь, неконтролируемые параметры объекта (наряду с контролируемыми) приводят к изменению параметров среды. Пример для задачи управления самолетом: рули высоты → тангаж → высота → плотность воздуха под крылом самолета.

Другими словами, задача управления состоит в том, чтобы задать *функцию управления*, которая в каждый момент времени анализирует информацию о значениях принимаемых параметрами во все предыдущие моменты времени, и на основе данной информации изменяет значение контролируемых параметров в текущий момент времени. При этом фазовая

кривая, получаемая в результате такого управления должна иметь максимальное *качество*.

3.1.2. Формальная постановка задачи управления

Задачи оптимального управления изучаются достаточно давно [22, 23]. Соответственно выполнена их формальная постановка и развита теория решения задач такого типа. В данном разделе задача управления будет сформулирована таким образом, чтобы легко было впоследствии описать предлагаемый метод решения. В целом приведенная формулировка согласуется с общепринятой формулировкой задачи оптимального управления [24].

Будем считать, что выделены $n \in N$ существенных для задачи управления вещественных параметров (как объекта, так и среды). Обозначим множество значений, принимаемых данными параметрами, как $Q = \mathrm{R}^n$. Задача управления осуществляется на некотором временном интервале, для простоты поставим ему в соответствие отрезок $[0, 1]$, разделенный на $T \in N$ минимальных временных интервалов.

Тогда некоторая кривая φ в пространстве фазовых параметров – элемент множества Q^T. Обозначим функцию, оценивающую **качество управления** как $g : Q^T \to \mathrm{R}$. Данная функция принимает в качестве аргумента кривую фазового пространства и возвращает вещественное число. Обозначим фазовую кривую для момента времени $t \in [1..T]$ как φ_t, при этом $\varphi_t \in Q^t$. тогда $\varphi = \varphi_T$.

Зависимости, существующие между различными параметрами, определяются средой (как правило, физическими законами среды). Для их формального задания определим функцию $h : Q \to Q$. Эта функция по значению параметров в текущий момент времени определяет их значения в следующий момент времени. Будем считать, что сама h во времени не изменяется.

Пусть среди всех параметров существует $m \in N,\ m \le n$ контролируемых параметров объекта. Под **функцией управления** будем понимать вектор

$f = [f_1, f_2, \dots, f_T]^{\mathrm{T}}$, где координата вектора – функция $f_t : Q^t \to \mathrm{R}^m$, которая, принимая фазовую кривую для некоторого момента времени t (элемент множества Q^t), возвращает элемент множества R^m – действие, которое предпринимается для управления объектом. В дальнейшем будет использовать термин «**правило управления**» применительно к f_t.

Введем для удобства вектор функций $F = [F_1, F_2, \dots, F_T]^{\mathrm{T}}$, $F_t : Q^t \to \mathrm{R}^n$. Координата вектора F_t, принимая на вход фазовую кривую для некоторого момента времени t, оставляет неконтролируемые параметры *без изменения*, а контролируемые изменяет по правилу, определяемому функцией f_t.

Определим значения параметров в начальный момент времени как $\varphi_0 \in \mathrm{R}^{\mathrm{n}}$. Тогда при фиксированной функции управления f фазовая кривая φ в пространстве параметров определяется однозначно следующим образом: $\varphi_t = h(F_t(\varphi_{t-1}))$, $t = 1..T$.

Итак, с учетом введенных определений задача управления формулируется следующим образом: $\max_f g(\varphi)$, при заданных h, φ_0.

Другими словами при заданных физических законах h и значению параметров в начальный момент времени φ_0 требуется найти функцию управления f, такую, что на определяемой ей фазовой кривой φ достигается максимальное значение функции g (оценки качества управления).

3.1.3. Проблемы, возникающие при решении задачи управления

При решении сформулированной в предыдущем разделе задачи управления возникает трудности различного характера. Перечислим их, а далее рассмотрим способы, позволяющие (частично) с ними справится.

Во-первых, зависимости между параметрами, зачастую, достаточно сложны. Например, изменение положения элеронов приводит к изменению крена, что в свою очередь приводит к изменению высоты, координат и других

параметров. Функция h задает эти зависимости, как правило, в неявном виде – через систему дифференциальных уравнений. Это в свою очередь означает, что поставленную вариационную задачу управления будет сложно (или невозможно) решить аналитически.

Кроме того, из формальной постановки задачи следует, что необходимо найти вектор $f = [f_1, f_2, \dots, f_T]^{\mathrm{T}}$, что в общем случае является сложной задачей. Это объясняется тем, что при большой дискретезации временного интервала, на котором решается задача управления, вектор f содержит большое число координат. Так, например, при времени полета самолета равном двум часам и минимальном интервале дискретезации равном одной секунде вектор будет содержать 7200 координат.

Наконец, поставленная задача управления осложняется тем, что каждая из координат вектора f – функция большого числа аргументов. Например, для задачи управления самолетом необходимо выделить как минимум десять параметров (для построения реалистичной модели значительно больше, возможно несколько сотен). Тогда область определения функции, принимающей решение об изменении контролируемых параметров на 85-ой минуте, есть $\mathrm{R}^{85 \cdot 60 \cdot 10} = \mathrm{R}^{51000}$.

Для упрощения поиска решения рассмотрим следующие подходы. Во-первых, не обязательно для каждого момента времени определять свой *способ управления* (координату функции управления во введенной терминологии). Например, способ управления самолетом вряд ли изменяется каждую секунду. В этом случае некоторая координата осуществляет управление объектом на протяжении интервала времени, который превосходит минимальный. В предельном случае будем искать решение задачи не в виде вектора $f = [f_1, f_2, \dots, f_T]^{\mathrm{T}}$, а в виде *единственной* функции $f^* : Q^T \to \mathrm{R}^m$, которая управляет объектом в каждый момент времени $t \in [1..T]$. Тогда решение задачи – функция управления $f = [f^*, f^*, \dots, f^*]^{\mathrm{T}}$.

Во-вторых, в каждый момент времени можно изменять контролируемые параметры, анализируя значения параметров не во все предыдущие моменты времени. В предельном случае при управлении анализируются значения параметров только в настоящий момент времени, а более ранние – игнорируются. Например, при управлении самолетом можно учитывать только его текущие координаты, но *не учитывать* всю траекторию движения. В этом случае, скорее всего, потребуется расширить множество параметров, чтобы сохранить *существенную* информацию о значениях параметров в предыдущие моменты времени. Например, если самолет должен посетить пункт назначения и вернуться обратно, то понадобится учитывать как минимум булевый параметр «в прямом или обратном направлении летит самолет», потому что только по текущим координатам самолета этого не определить.

В предельном случае приведенных упрощений – одно правило управления для всех моментов времени, принимающее в качестве аргумента только значения параметров в настоящий момент времени – решение оптимизационной задачи искать значительно проще. Однако наилучшее из решений такого вида может не удовлетворять заданным критериям оптимальности управления объектом, потому что пространство поиска сильно сужено. Поэтому важной представляется задача нахождения компромисса между ограниченностью пространства, в котором осуществляется поиск решения и сложностью решения задачи оптимизации. Далее будет предложено решение этой задачи, которое основано на автоматном подходе [3].

3.1.4. Автоматный подход и его недостатки

Как правило, при решении задачи управления можно выделить состояния, в которых может находиться объект управления. При управлении самолетом такими состояниями, например, являются: «получено разрешение на взлет», «разбег», «отрыв от земли», «заход на курс», «следование курсом», «переход на автоматическое удержание курса», «отказ двигателя», «приближение к

аэродрому назначения», «полет с экономным расходом топлива в случае неблагоприятных для посадки условий», «заход на посадку», «торможение» и т.д.

В задаче управления танком для игры *Robocode* можно выделить, например, следующие состояния: «танк обладает большой энергией и атакует», «танк уклоняется от атаки противника», «ведется обнаружение противника», «выполняется перемещение в заданный район поля», «идет сближение с противником» и т.д.

Поэтому при решении задач управления разумно использовать автоматный подход, который основан на явном выделении состояний управляемого объекта. При таком подходе эвристически определяется *конечные* множества входных и выходных воздействий, задается число состояний графа переходов автомата, его структура – условия на переходах и набор выходных воздействий в состояниях (автоматы Мура), или условия и набор выходных воздействий на переходах (автоматы Мили). Не уменьшая общности, положим, что используются автоматы Мура. Автоматный подход разрешает две поставленные выше задачи по выбору компромисса между общностью вида, в котором осуществляет поиск функции управления и простотой решения задачи следующим образом:

1. Количество *различных* координат функции управления полагается равным количеству состояний автомата. Таким образом, для каждого состояния автомата определяется свое правило управления объектом (координата искомой функции управления). Тогда одно и тоже правило может управлять объектом в различные моменты времени – функция управления декомпозируется по состояниям.
2. Координата функции управления, сопоставленная каждому состоянию, управляет объектом исходя только из значений параметров в настоящий момент времени. Область ее определения – множество Q.

Данный подход имеет и существенные недостатки. Во-первых, задача определения *конечного* множества входных и выходных воздействий достаточно трудна, потому что отсутствует разумный критерий качества ее решения. Так при определении множества входных воздействий некоторым образом производится дискретизация множества Q. Например, в работе [25] для задачи управления самолетом выделяются, следующие входные и выходные воздействия:

- «x1: самолет движется – принимает истинное значение при скорости воздушного потока более пяти узлов»;
- «x5: у поверхности земли – принимает истинное значение при высоте над землей менее пятнадцати футов»;
- «z13: половина газа – устанавливает дроссель в среднее положение»;
- «z16: взлетное положение закрылков – устанавливает требуемое положение закрылков на уровень 0.3, рекомендуемый для взлета».

Из данного описания следует, что дискретизация осуществляется по некоторым пороговым значениям («пяти», «пятнадцати», «половина», «0.3»). При этом если определение множества входных и *выходных* воздействий было осуществлено неудачно, то найденная при решении задачи оптимизации функция управления f будет порождать фазовую кривую φ с малым качеством управления $g(\varphi)$. Поэтому придется формировать множество входных и выходных воздействий заново либо эвристически, либо, включая пороговые значения в исходную задачу оптимизации, что еще больше ее усложнит.

Таким образом возникает и задача выбора компромисса между числом входных и выходных воздействий (подробность дискретизации) и размером пространства, на котором производится поиск решения задачи управления. Так в предельном случае, имея всего по одному входному (для каждого параметра) и выходному (для каждого контролируемого параметра) воздействию, наверняка, не удастся найти достаточно хорошее решение задачи управления.

Наоборот, разбив множество значений каждого параметра на очень большое число малых интервалов и ассоциировав с каждым их этих интервалов некоторое воздействие, скорее всего, удастся найти решение почти такого же качества, как и для исходной задачи управления. Однако нахождение такого решения будет связано с трудностями, обусловленными большим числом входных и выходных воздействий.

Другой проблемой при использовании автоматного подхода является сложность построения графа переходов автомата, в частности, определение условий на переходах. Как правило, построение логического выражения, при выполнении которого выполняется переход, также выполняется эвристически на основе знаний эксперта о решаемой задачи управления. В случае если качество решения задачи управления оказывается низким, сложно понять на каком переходе была допущена ошибка в построении условия и как ее исправить.

Таким образом, для применения автоматного подхода требуются существенные знания о решаемой задачи управления. Даже при наличии этих знаний построение автомата производится вручную, что трудоемко и потенциально подвержено ошибкам.

3.1.5. Предлагаемый метод

В этом разделе будет предложен метод решения задачи управления основанный на автоматном подходе, однако, лишенный перечисленных недостатков. Основной идеей предлагаемого метода является использование разновидности генетических алгоритмов – программирования с экспрессией генов [3]. Примеры применения данного метода для решения задач приведены в [25].

При этом будет решен ряд обозначенных выше проблем – сложности отыскания решения аналитическим способом, выбора конечного множества входных и выходных воздействий, построения условий на переходах. При

использовании предлагаемого подхода не требуется *эвристически* определять множество входных и выходных воздействий, выполнять построение условий (логических выражений) на переходах.

3.1.6. Общая идея

Для решения поставленной задачи управления с помощью генетических алгоритмов необходимо определить способ представления решения задачи управления в виде хромосомы (особи популяции генетического алгоритма), выбрать функцию приспособленности и определить генетические операции (мутация, скрещивание и отбор). Три перечисленные задачи далее будут рассмотрены подробнее. После решения этих задач генетические алгоритмы автоматически осуществляют поиск функции управления в пространстве возможных решений.

Схема работы генетических алгоритмов приведена на рис. 13 (вторая глава).

3.1.7. Представление решения задачи управления в виде хромосомы

Опишем способ, в котором решение задачи управления представляется в виде хромосомы. Зафиксируем N – число состояний автомата, построение которого будет выполняться в дальнейшем. Это число определяет «сложность» автомата. Его выбор осуществляется эвристически, однако не представляет существенной трудности – если сгенерированный в последствии автомат с заданным числом состояний не удовлетворяем выбранным критериям качества – можно увеличить число состояний, в противном случае – уменьшить и повторить процедуру поиска решения.

Для каждого состояния определим некоторое *правило управления* $f_i : Q \to \mathrm{R}^m,\ i = 1..N$. Для каждой пары различных состояний с номерами (i, j) зададим *функцию перехода* $f_{ij} : Q \to \mathrm{R}$. Таким образом, будет определен полный

граф, в вершинах которого находятся правила управления, а на ребрах – функции перехода. Как именно определяются правила управления и функции перехода будет рассмотрено далее.

При управлении объектом в каждый момент времени, находясь в состоянии *i*, последовательно выполняются следующие действия:

- вычисляются значения функций f_{ij} для всех j, $i \neq j$. Как только одно из вычисленных значений оказывается больше нуля ($f_{ij} > 0$), выполняется переход в состояние j. Если же все $f_{ij} \leq 0$, то состояние не изменяется;
- определяется значение *правила управления* f_i, и выполняется действие, соответствующее найденному значению.

На рис. 18 приведен пример автомата с тремя состояниями, решающего некоторую задачу управления.

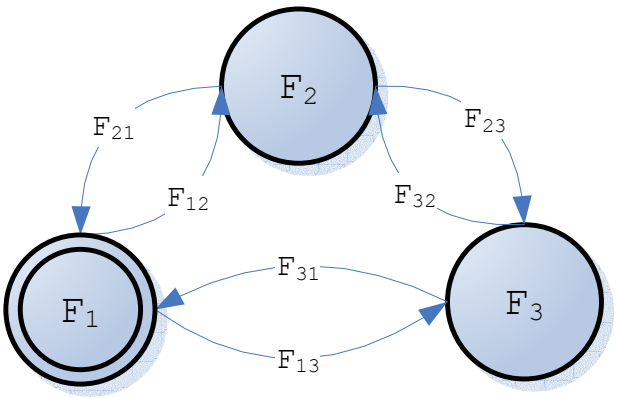


Рис. 18. Пример автомата

Функция управления может быть представлена как набор из $N \cdot (N-1)$ функций перехода и N правил управления. Напомним, что функция перехода – это функция, отображающая из множества значений параметров $Q = \mathrm{R}^n$ во множество вещественных чисел R. Правило перехода – функция, отображающая из Q в R^m. Предлагается правило управления рассматривать как набор из m функций отображающих из Q в R. Тогда хромосома, соответствующая решению задачи управления, может быть представлена как

набор из $N \cdot (N-1) + m \cdot N$ **функций, каждая из которых отображает из** R^n **в** R. Таким образом, задача представления решения в виде хромосомы свелась к представлению функции из R^n в R в виде к

омпоненты хромосомы.

Ограничим класс искомых отображений из R^n в R функциями, которые являются композицией некоторых базовых функций нескольких аргументов (от одного до *n*). Выбор базовых функций, как правило, определяется конкретной задачей. Требования, предъявляемые к набору базовых функций, таковы – набор должен быть достаточно полным, для того чтобы выразить необходимые зависимости, и не слишком избыточным, для того чтобы работа алгоритма поиска функции управления была эффективной.

Можно заранее взять достаточно широкий (для задач управления, встречающихся на практике) класс функций: арифметические, показательные, логарифмические, тригонометрические, условные, вероятностные и т.д. Например, в качестве базисных функций можно выбрать следующие: $+(x, y) = x + y$, $++(x, y, z) = x + y + z$, $n(x) = -x$, $*(x, y) = xy$, $**(x, y, z) = xyz$, *min*, *sin*, *cos*, *exp*, *log*, $|x|$, $s(x) = \frac{1}{1+e^{-x}}$, $if>(x, y, z, w) = x > y \ ? \ z : w$, $random_{[0..1]}$.

В качестве терминалов выступают значения параметров в текущий момент времени. Для задачи управления самолетом, терминалами, например, будут являться: v – скорость самолета, h – высота полета, e – запас топлива, t – время полета, φ, θ, ψ – углы Эйлера самолета и т.д. Кроме того, набор терминалов дополняется константами, например: 0, .5, 1, 100, π.

Определив набор терминалов и базовых функций, достаточно воспользоваться разновидностью генетических алгоритмов, обладающей высокой эффективностью – программированием с экспрессией генов (*Gene Expression Programming*) [3]. Программирование с экспрессией генов ищет функцию в виде, так называемых, *Karva*-деревьев. На рис. 19 приведен пример такого дерева.

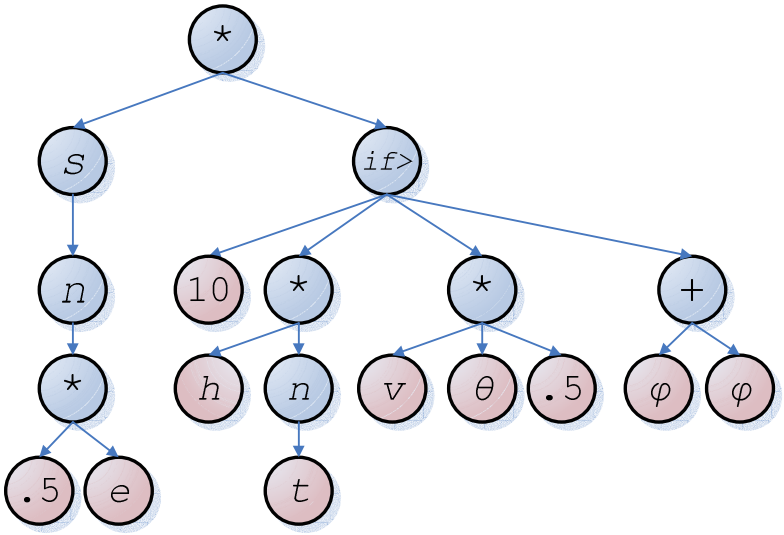


Рис. 19. Пример представления функции из R^n в R в виде Karva-дерева

Итак, хромосома – набор $N \cdot (N-1) + m \cdot N$ функций, каждая из которых представима в виде *Karva* – дерева.

3.1.8. Функция приспособленности, генетические операции

Для определения приспособленности хромосомы (в виде которой представлена некоторая функция управления) достаточно получить автоматное представление функции управления, что выполняется однозначно. Затем, промоделировав процесс управления объектом, управляемым автоматом, получим фазовую кривую, для оценки которой определена функция g.

Критерий остановки процесса эволюции определяется спецификой задачи и функцией качества решения – g. Например, если для задачи управления самолетом функция g возвращает число, обратное максимальной перегрузке за все время полета, а величина допустимой перегрузки – 4, то разумно остановить процесс эволюции при значении функции приспособленности равном 0.25.

Для управления игровыми объектами (например, футболистами в виртуальном футболе или танком в игре *Robocode*) естественно останавливать

процесс эволюции по достижении игровым объектом желаемого результата, например, победы над соперником с заданной вероятностью.

Итак, в данном разделе предложен метод автоматического построения автоматов, являющихся решением задачи управления. В следующем разделе производится апробация предложенного метода на задаче построения системы управления для танка в популярной компьютерной игре *Robocode*.

3.2. Создание системы управления танком в игре *Robocode*

В данном разделе производится апробация предложенного метода решения задач управления при создании системы управления танком для популярной компьютерной игры *Robocode*. Этот метод позволяет на основе использования генетических алгоритмов строить достаточно простые системы управления танком.

Отметим, что *не* ставится цель создать танк, способнй 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 трудоемка.

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, а также танк `newCynic.Cynical` [28]. Выбор последнего был связан с тем, что этот танк (как и прототип, на основе которого он построен [29]), в отличие от многих других танков, был предварительно спроектирован и для него существует достаточно подробная открытая проектная документация. 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 (можно рассматривать этот случай и как автоматный, для автомата с одним состоянием). При этом если автоматы не используются, то система управления описывается только одной функцией, формирующей выходные воздействия на объект управления. При автоматном подходе для каждого состояния автомата определена соответствующая функция управления, согласно описанию метода в предыдущем разделе.

На рис. 20 в качестве примера приведен внешний вид среды игры *Robocode* во время одного из боев. На этом рисунке разработанный танк назван *EvolvedTank*.

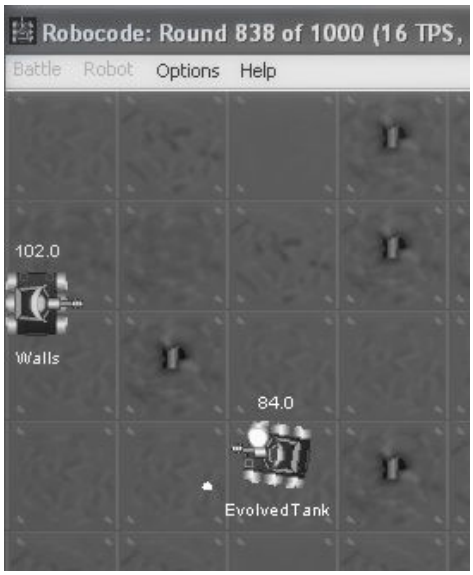


Рис. 20. Пример боя в игре *Robocode*

3.2.1. Обзор

Известно большое число танков для рассматриваемой игры [30]. 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 [31, 32], в которых танки строятся не вручную, а автоматически – с применением методов искусственного интеллекта.

В работе [31] использовано несколько таких методов (обучение с подкреплением, нейронные сети и генетические алгоритмы) для управления

различными системами танка (радаром, системой движения и пушкой). Генетические алгоритмы в работе [31] применялись для создания систем управления радаром и движением танка. Однако их использование не привело к желаемому результату – система управления радаром осуществляла его поворот на 360 градусов, а система передвижения задавала перемещение танка по кругу.

В работе [32] используется сравнительно сложный способ представления системы управления в виде особи генетического алгоритма – программы на языке *TableRex*. При этом длина программы составила 7776 бит.

3.2.2. Постановка задачи

Устройство танка схематично показано на рис. 21. Необходимо управлять следующими устройствами: радаром (`Radar`), пушкой (`Gun`) и системой движения (`Body`).

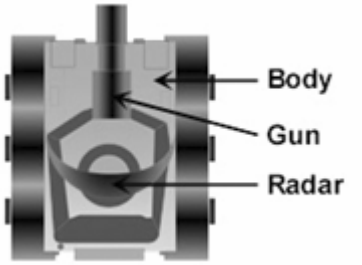


Рис. 21. Схематичное изображение танка

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

Перейдем к формальной постановке задачи. Обозначим текущее время в игре как $t \in N$, а максимальное время, которое возможно для данной игры – M. 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 $f_t : S^t \to A$, 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 $[f_1, f_{2,\ldots,} f_M]^T$.

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 упрощенная 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;

- множество действий $A' = \{g, p, d, h\}$, где g – угол поворота пушки, p – энергия снаряда (неположительные 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 $f : S' \to A'$. Эту функцию будем искать *автоматически* с использованием программирования с экспрессией генов. В отличие как от стандартных генетических алгоритмов, использующих строки фиксированной длины, так и от генетического программирования [5], использующего деревья, хромосомы в генетическом программировании с экспрессией генов представляются в виде строк фиксированной длины, которые преобразуются в, так называемые, *Karva*-деревья. Генетические операции с такими деревьями (в отличие от генетического программирования) *корректны по определению*. За счет этого, при использовании программирования с экспрессией генов не приходится тратить время на проверку корректности деревьев, получаемых в результате скрещивания и мутаций. Это обеспечивает используемому подходу указанные преимущества перед генетическим программированием.

Как уже было отмечено в предыдущем разделе, для решения поставленной задачи с помощью генетических алгоритмов (и, в частности, для программирования с экспрессией генов) необходимо определить способ представления функции в виде хромосомы (особи популяции генетического алгоритма), выбрать функцию приспособленности и определить генетические операции (мутация, скрещивание и отбор).

Как было отмечено ранее, предлагаемый метод позволяет строить системы управления, как с использованием автоматного подхода, так и без его применения. Ниже эти подходы рассматриваются отдельно.

3.2.3. Построение системы управления без применения автоматов

Сначала опишем подход, не предполагающий использования автоматов при создании системы управления.

Общая схема генетических алгоритмов. Схема одного шага работы генетических алгоритмов приведена на рис. 13.

3.2.4. Представление в виде хромосомы

Для представления в виде особи генетического алгоритма искомой функции управления $f : S' \to A'$ предлагается представлять ее в виде четырех функций f_i, каждая из которых возвращает вещественное число: f_1 – угол поворота пушки, f_2 – энергия снаряда, f_3 – расстояние, на которое перемещается танк, f_4 – угол поворота танка.

В свою очередь, функция f_i представляется в виде *Karva*-дерева – стандартного способа представления для программирования с экспрессией генов. Таким образом, хромосома состоит из четырех строк – линеаризованных *Karva*-деревьев, для представления которых применяются описываемые ниже функции и терминалы.

Требования, предъявляемые к набору базовых функций, используемых при задании хромосомы: набор должен быть достаточно полным, для того чтобы выразить необходимые зависимости действий от позиции, и не слишком избыточным, для того чтобы работа генетического алгоритма была эффективной.

Предлагается использовать следующий набор базовых функций: $+(x, y) = x + y$, $++(x, y, z) = x + y + z$, $n(x) = -x$, $*(x, y) = xy$, $**(x, y, z) = xyz$, min, $s(x) = \frac{1}{1+e^{-x}}$, $if>(x, y, z, w) = x > y \ ? \ z : w$.

Набор терминалов (координаты танка (`x`, `y`), его энергия (`e`) и т.д.) строится исходя из определения позиции (элемента множества S'). Кроме того,

этот набор необходимо дополнить некоторым множеством констант. В настоящей работе были выбраны следующие константы: 0, .5, 1, 2, 10.

В табл. 8 приведен пример сгенерированной хромосомы, а в табл. 9 – соответствующая этой хромосоме функция управления $f = (f_1, f_2, f_3, f_4)$, представленная через базовые функции.

Таблица 8. Представление функции в виде хромосомы

Линеаризованные Karva-деревья	
f_1	4 0 7 1 17 4 5 2 4 1 18 9 10 13 12 10 13 22 23 11 17 21 8 15 16 14 21 13 16 18 22 10 16 19 13 8 21 13 10 8 23
f_2	6 4 5 8 0 2 7 0 5 0 20 15 8 19 19 18 8 11 15 16 13 22 23 22 12 20 22 19 13 19 23 15 15 22 11 22 11 14 8 17 23
f_3	7 2 1 3 6 5 6 0 6 7 21 16 22 19 21 17 19 23 9 13 17 13 14 15 9 21 12 16 21 14 16 23 16 15 23 16 8 21 15 14 22
f_4	2 4 5 0 2 7 1 3 4 1 19 8 15 17 17 9 18 13 13 18 18 21 17 17 17 15 8 9 18 22 17 23 19 21 10 10 11 20 13 19 23

Таблица 9. Функции, соответствующие хромосоме

Функция $f = (f_1, f_2, f_3, f_4)$
f_1 = *(s[n(*(.5, e))], if>[10, *(n(E), dh), *(y, dr, .5), +(w, dr)])
f_2 = min(*[x, s(*(x, tr, 1))], *[+(s(2), h, if>(1, x, GH, GH), s(dh)])
f_3 = if>(+[*(GH, d, 10), min(GH, E)], n[s(y)], +[min(.5, 10), if>(.5, 0, 1, y), d], min[2, e])
f_4 = +(*[s(*(dh, dh)), +(n(d), GH)], *[if>(x, 1, 10, 10), n(y), +(dh, .5, .5)])

На рис. 22 приведен пример *Karva*-дерева для функции f_1 из табл. 9.

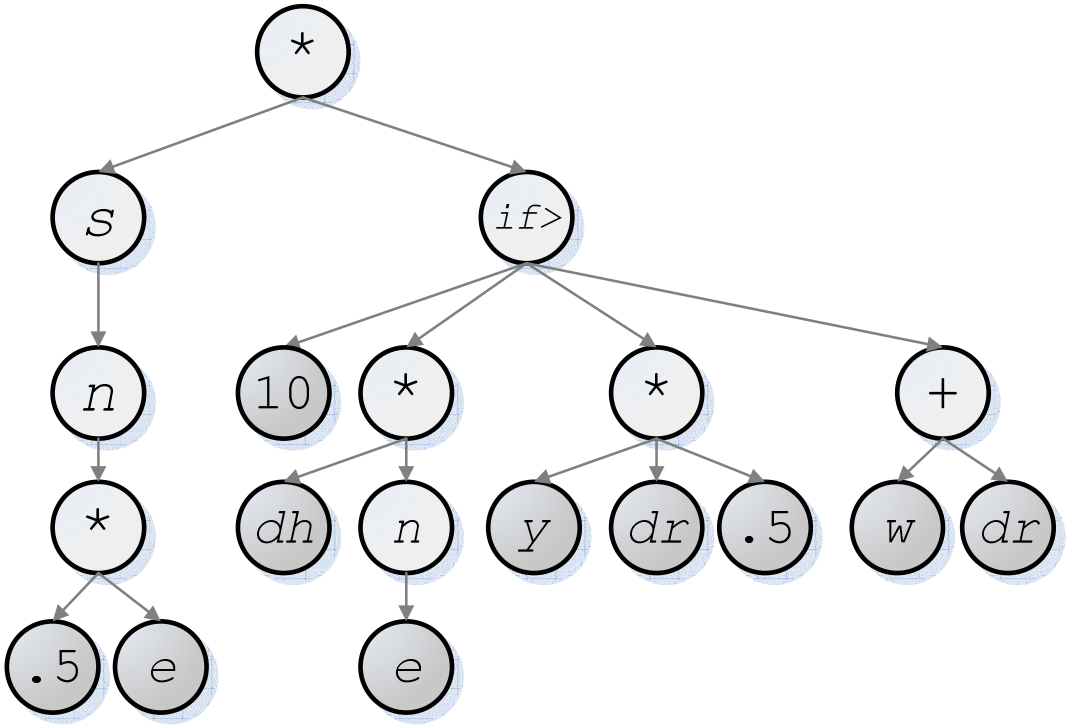


Рис. 22. Karva-дерево функции f_1 из табл. 9

3.2.5. Функция приспособленности

Опишем, как была выбрана функция приспособленности. Она отражает результат поединков против выбранного соперника. При этом отметим, что, так как результат поединка существенно зависит от начального положения танков, то проводится не один поединок, а несколько (двадцать). Увеличение числа сражений позволило бы получить более точный результат, однако это было бы связано с возрастанием времени вычисления функции приспособленности, которое и так велико.

Для того чтобы уравнять шансы соперников, выбор начальных положений танков производится следующим образом. Создается случайный набор пар координат $((x_1, y_1), (x_2, y_2))$. После этого для каждой из них проводится два поединка, в первом из которых танк соперника имеет начальные координаты (x_1, y_1), а во втором – (x_2, y_2). Обозначим число очков, набранных нашим танком в бою (состоящим из 20 раундов) как s, а число

очков, набранных его соперником, как s_e. При этом приспособленность танка будем вычислять следующим образом: $f = 100\frac{s}{s+s_e}$.

Процесс эволюции продолжается до тех пор, пока максимальное значение функции приспособленности для элементов популяции не превзойдет некоторого порогового значения. Например, пороговое значение, равное 70, означает, что танк в среднем «научился» побеждать своего соперника. При этом отметим, что значение функции приспособленности, равное 50, означает, что наилучший из танков *текущей популяции* в ходе двадцати раундов при случайном начальном положении танков, выигрывает половину раундов.

3.2.6. Генетические операции

Для реализации генетического алгоритма в настоящем разделе используются стандартные генетические операции: отбор, мутация и скрещивание. Отбор осуществляется по методу рулетки. Скрещивание хромосом (как наборов из четырех строк, например, приведенных в табл. 8) производится поэлементно. При этом для каждой пары строк с вероятностью p в качестве i-го элемента результатом скрещивания выбирается i-й элемент либо первой, либо второй хромосомы, а с вероятностью $(1 - p)$ – скрещивание производится стандартным образом с использованием метода битовой маски. Аналогично скрещиванию, мутация выполняется поэлементно. При этом каждый из символов строки с некоторой вероятностью заменяется некоторым другим. Более подробное описание указанных операций приведено в работе [1].

3.2.7. Эксперименты

При реализации генетического алгоритма число особей в популяции было выбрано равным 50. Один шаг эволюции (скрещивание, мутация, отбор, вычисление функции приспособленности) в популяции такого размера выполняется около пяти минут на компьютере *Intel Pentium M* 1.86 *GHz*. При этом практически все время тратится на вычисление функции

приспособленности. Данное обстоятельство не позволяет использовать популяции б*о*льших размеров. Так, например, даже при указанном числе особей время работы генетического алгоритма может занимать несколько часов.

На рис. 23 приведен пример графика, иллюстрирующего работу генетического алгоритма для генерации танка против соперника `sample.Walls`. По оси абсцисс отложен номер поколения, а по оси ординат – значение функции приспособленности (`Max` – максимальное значение среди особей популяции, а `Avg` – среднее значение).

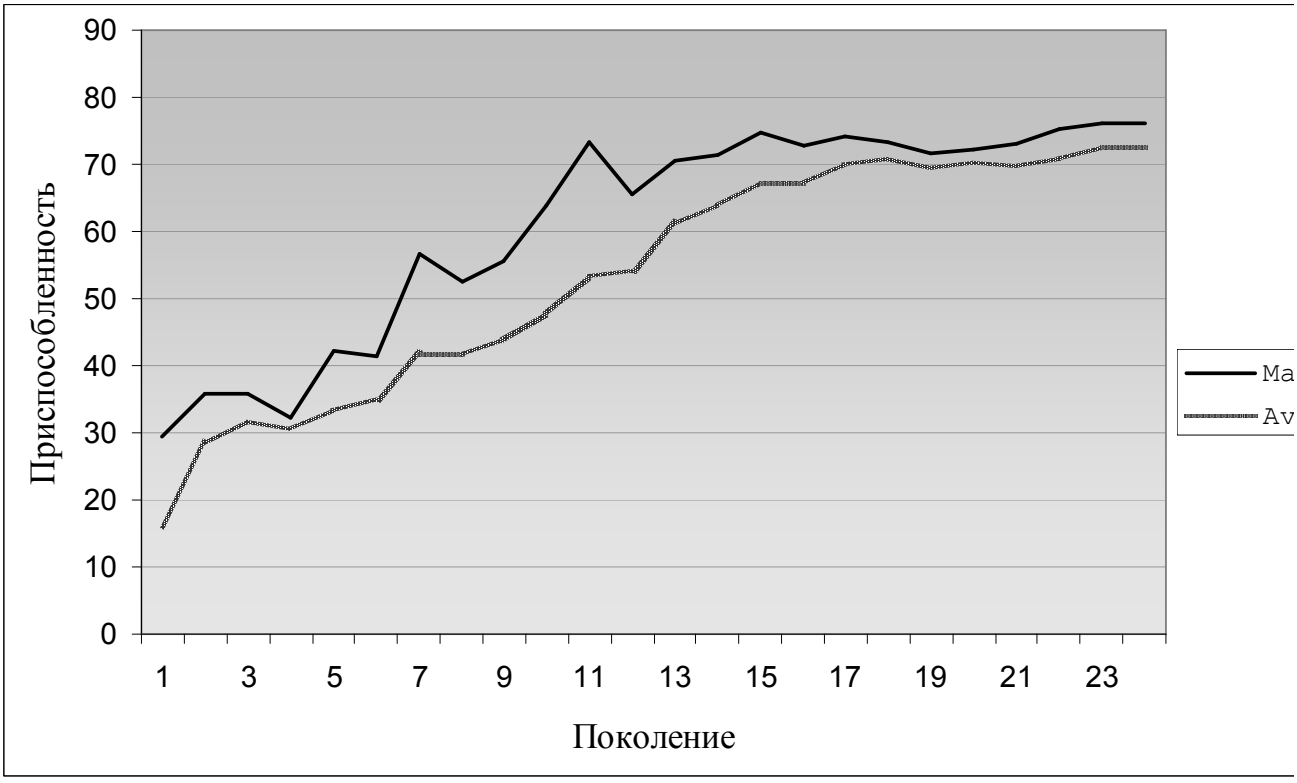


Рис. 23. Процесс эволюционного построения танка

Из рассмотрения графика максимального значения функции приспособленности (`Max`) следует, что уже на 11-ом поколении эволюцию можно было остановить, так как максимальное значение функции приспособленности превысило пороговое значение 70. Однако процесс был

остановлен после 23-го поколения, так как значение функции приспособленности перестало изменяться существенно.

Таким образом строится каждый танк в результате сражений с одним из танков тестового набора. В табл. 10 приведены описания систем управления, которые сгенерированы в ходе сражений с танками соперников `newCynic.Cynical` и `sample.Walls`. Эти танки использовались и в сражениях, результаты которых перечислены в табл. 4, так как указанные танки являются наиболее сильными в тестовом наборе.

Таблица 10. Представление функции управления сгенерированных танков

Соперник	Описание функции
`newCynic.Cynical`	0 2 1 6 3 6 0 6 13 19 14 13 23 10 28 18 13 17 18 13 19 14 17 23 15 13 18 9 29 30 15 14 13
	0 7 2 0 4 6 1 7 9 22 23 11 23 22 16 23 18 27 10 22 27 16 9 13 15 21 24 29 14 17 16 20 23
	3 4 8 1 6 0 0 5 14 17 21 24 21 17 19 11 19 11 24 18 28 23 23 15 10 24 16 22 14 30 26 14 15
	7 3 3 1 8 0 1 2 18 23 29 15 26 29 16 25 30 25 16 23 26 26 27 17 18 26 19 25 15 10 19 28 20
`sample.Walls`	7 0 6 8 2 3 8 6 0 5 23 13 18 9 12 20 23 14 20 27 29 12 14 26 14 10 13 26 10 17 22 26 21 18 14 30 14 21 20 23 9
	4 4 4 4 1 6 1 2 2 2 9 22 19 19 29 25 21 26 17 17 14 27 13 14 21 16 22 30 23 21 29 14 21 19 15 9 9 19 22 9 20
	24 0 3 2 1 5 7 1 2 1 16 11 21 15 25 18 25 30 18 17 21 9 25 19 26 10 28 28 13 12 11 28 28 18 14 14 14 26 11 30 28
	2 5 6 6 5 13 6 5 8 12 23 11 28 22 21 14 16 15 21 15 29 29 19 23 14 20 24 27 14 14 22 26 26 10 12 26 15 28 10 15 22

После генерации каждого танка, для оценки его эффективности были проведены бои с тем же соперником, против которого он выводился. Бои состояли из 100 раундов. В табл. 4 приведены результаты этих боев. В этой таблице в столбце “Счет” первое значение – число очков набранных нашим танком, а второе – танком соперника. В общем случае, как отмечалось выше, увеличение числа раундов приводит к более объективным результатам.

Таблица 11. Результаты поединков (100 раундов)	
Соперник	Счет
`newCynic.Cynical`	10933 : 8108
`sample.Walls`	9559 : 7240
`sample.SpinBot`	11414 : 8556
`sample.MyFirstRobot`	11964 : 5346
`sample.Corners`	18086 : 2971
`sample.Crazy`	10797 : 4278
`sample.Fire`	17610 : 5500
`sample.Tracker`	18642 : 4844
`sample.TrackFire`	16269 : 9851
`sample.Target`	17968 : 5
`sample.RamFire`	18572 : 3089

3.2.8. Автоматный подход

Эвристическое построение автоматов, управляющих танком, применялось в работе [28, 29]. В тоже время попыток автоматически построить автоматы для управления танком ранее не предпринималось.

В рассмотренном выше подходе, управление танком в любой момент времени осуществляется единственной функцией $f : S' \to A'$. При использовании автоматов эта функция декомпозируется с помощью состояний согласно подходу, предлагаемому в данной главе и описанному в предыдущем разделе. В каждом состоянии автомата определим функцию $f_i : S' \to A'$, где i – номер рассматриваемого состояния. Для каждой пары различных состояний с номерами (i, j) зададим *функцию перехода* $f_{ij} : S' \to R$. Построение функций переходов выполняется с помощью генетических алгоритмов, сами функции представляются в виде *Karva*-деревьев.

Табл. 12 иллюстрирует процесс эволюционного построения танка с автоматной структурой.

Таблица 12. Работа генетического алгоритма (процесс эволюции)

Номер поколения	Максимальное значение функции приспособленности	Среднее значение функции приспособленности	Время, с
0	31.15	15.70	122
1	36.71	26.42	138
2	34.93	28.91	152
3	40.30	32.96	168
…	…	…	…
9	52.25	43.74	261
10	55.53	41.15	358
…	…	…	…
21	64.49	59.14	599
22	69.44	60.18	606
23	91.38	62.55	631
24	89.45	72.68	621
…	…	…	…
36	93.73	91.48	778

Эволюция была остановлена на 36-ом поколении, так как рост максимального значения функции приспособленности практически остановился.

В табл. 13 приведен пример описания автомата из двух состояний в виде особи генетического алгоритма. Это описание содержит десять строк: восемь (f_1 – f_8) определяются состояниями и две (f_9, f_{10}) – переходами. Данный автомат выводился в результате боёв с танком `sample.Walls`.

Таблица 13. Представление функции управления сгенерированных танков

Описание автомата	
f_1	5 2 2 5 6 7 6 1 4 4 8 20 9 8 12 23 22 22 16 13 14 10 12 9 14 16 9 18 11 11 23 23 19 18 23 13 15 16 16 22 14
f_2	3 16 2 5 1 2 0 6 3 5 15 8 17 12 22 14 21 11 10 12 13 21 14 14 8 19 11 20 17 15 23 14 22 18 8 10 9 21 20 18 22
f_3	3 7 14 3 7 5 4 16 4 1 17 16 9 20 16 22 12 10 17 9 11 8 19 19 16 12 11 8 15 20 9 23 20 19 18 10 19 21 13 10 18
f_4	1 7 14 6 2 4 7 6 4 6 9 8 12 21 17 10 13 16 23 18 17 14 15 15 19 17 18 14 21 9 15 17 12 19 11 8 13 9 23 10 18
f_5	4 0 7 1 17 4 5 2 4 1 18 9 10 13 12 10 13 22 23 11 17 21 8 15 16 14 21 13 16 18 22 10 16 19 13 8 21 13 10 8 23
f_6	6 4 5 8 0 2 7 0 5 0 20 15 8 19 19 18 8 11 15 16 13 22 23 22 12 20 22 19 13 19 23 15 15 22 11 22 11 14 8 17 23
f_7	7 2 1 3 6 5 6 0 6 7 21 16 22 19 21 17 19 23 9 13 17 13 14 15 9 21 12 16 21 14 16 23 16 15 23 16 8 21 15 14 22
f_8	2 4 5 0 2 7 1 3 4 1 19 8 15 17 17 9 18 13 13 18 18 21 17 17 17 15 8 9 18 22 17 23 19 21 10 10 11 20 13 19 23
f_9	20 6 4 3 5 5 3 0 6 2 11 8 22 15 15 19 23 14 20 10 23 17 18 20 17 13 11 20 20 14 18 15 11 10 12 11 8 8 13 21 12
f_{10}	3 7 0 4 1 4 5 1 4 16 22 15 8 14 22 19 18 17 22 10 11 15 21 13 9 13 11 8 8 13 16 11 14 8 10 23 8 23 9 17 18

Сгенерированный танк в 1000 раундах побеждает танк соперника со счетом 180168 на 28278. Таким образом, в среднем наш танк побеждает противника с вероятностью около 86%. Это несколько меньше, чем полученное в табл. 12 максимальное значение функции приспособленности – 93.73. Данное обстоятельство объясняется тем, что результат боя из 20 раундов имеет большую погрешность из-за сравнительно небольшого числа экспериментов.

Применение автоматного подхода для описания поведения танка позволяет генерировать более сложные системы управления по сравнению с первым подходом, и, следовательно, создавать более сильные танки. Врочем на практике полученные танки обладают примерно такой же эффективностью, что

и танки, построенные без автоматов. Это, видимо, связано с тем, что пространство поиска (множество автоматов с фиксированным числом состояний) достаточно велико для того, чтобы была возможность генерировать “хорошие танки” за разумное время в условиях медленного вычисления функции приспособленности. Можно надеяться, что дальнейшая работа в этом направлении (например, ускорение вычисления функции приспособленности) позволит более эффективно использовать автоматный подход при построении танков с применением генетических алгоритмов.

3.2.9. Программная поддержка метода

Управление танком может осуществляться либо синхронно, либо асинхронно. В первом случае главный класс программы наследуется от библиотечного класса `Robot`, а во втором – от библиотечного класса `AdvancedRobot` [27]. В данной работе был выбран второй вариант. Таким образом, приведенный в приложении код, принадлежит классу, который наследуется от класса `AdvancedRobot`.

Метод поддержан двумя типами программ. Первая из них на основе использования генетических алгоритмов строит символическое описание системы управления, пример которого приведен в табл. 8, а вторая – интерпретирует ее и управляет танком (приложение).

Первая программа имеет две модификации, соответствующие традиционному и автоматному подходам. Вторая программа не зависит от используемого подхода и состоит из следующих основных компонентов:

- среда *Robocode*;
- система управления, которая по входным воздействиям, получаемым из среды, формирует выходные воздействия на объект управления;
- объект управления (танк), содержащий систему движения и пушку.

Радар реализован тривиально, и поэтому в отдельный класс не выделен.

Эта программа является главной и содержит обращение к некоторым классам, исходный код которых не приведен в приложении. Обе программы написаны на языке *Java*.

Выводы по главе 3

В результате исследований, описанный в данной главе, был предложен метод решения достаточно широкого класса задач управления. Преимуществом данного метода является возможность автоматического построения автоматов, управляющих объектами. При этом удалось ответить на ряд сложных вопросов, возникающих при решении задач оптимального управления – сложности поиска решения в аналитическом виде, дискретезации временного интервала и других.

Предлагаемый метод решения задачи управления был успешно опробован на игре *Robocode*, позволив автоматически генерировать системы управления для танков, способные побеждать достаточно сильных соперников. Структуры генерируемых систем управления достаточно просты, но в то же время танки с этими системами управления побеждают любой фиксированный танк из тестового набора. В заключение отметим, что предлагаемый метод не позволяет генерировать танки, побеждающие наиболее сильных из известных танков, так как в последних используются эвристически созданные сложные и эффективные алгоритмы управления.

ЗАКЛЮЧЕНИЕ

1. Установлена неприменимость алгоритма Баума-Велша при построении моделей максимального правдоподобия некоторого класса скрытых марковских моделей.
2. Для указанного класса показана эффективность метода построения модели максимального правдоподобия, основанного на использовании генетических алгоритмов для выбора начальных параметров.
3. Предложена и решена задача поиска ошибок в автоматах. При этом рассматривается один тип ошибок – неучтенные переходы между состояниями автомата.
4. Предложен универсальный метод решения задач оптимального управления, основанный на применении генетических алгоритмов для автоматического построения автоматов.
5. Предложенный метод опробован для построения системы управления танком в игре *Robocode*. Построенные танки показали эффективность предлагаемого метода.

НЕКОТОРЫЕ ОТКРЫТЫЕ ВОПРОСЫ И ЗАДАЧИ

1. Привести математически строгое определение понятия «сильной детерминированности», используемого во второй главе.
2. Исследовать зависимость успешности построения моделей максимального правдоподобия от «детерминированности» скрытых марковских моделей.
3. Обобщить метод тестирования автоматов на ошибки других типов кроме неучтенных переходов.
4. Исследовать эффективность метода автоматического построения автоматных систем управления на других задачах кроме *Robocode*.
5. Повысить эффективность метода построения танков, например, за счет выбора другого набора базовых функций или терминалов.

ПУБЛИКАЦИИ

1. Государственный контракт: «Технология генетического программирования для генерации автоматов управления системами со сложным поведением».
2. Труды V Межвузовской конференции молодых ученых, 2008.
3. Труды XII Всероссийской конференции «Фундаментальные исследования и инновации в технических Университетах», 2008.
4. XI Международной конференции по мягким вычислениям и измерениям, 2008.
5. IV Международная конференция по проблемам управления, 2009.
6. На рецензию в журнал «Известия РАН. Теория и системы управления».
7. Конкурс грантов 2008 года для студентов и аспирантов ВУЗов и академических институтов.
8. XV Всероссийская научно-методическая конференция «Телематика'2008».

ЛИТЕРАТУРА

1. *Mitchell M.* An Introduction to Genetic Algorithms. MA: The MIT Press, 1996.
2. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М. Физматлит. 2006.
3. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач. СПб.: Наука, 1998. http://is.ifmo.ru/books/switch/1
4. *Ferreira C.* Gene Expression Programming: A New Adaptive Algorithm for S olving Problems // Complex Systems, 2001. V. 13. № 2. P. 87–129. www.gene-expression-programming.com/webpapers/GEP.pdf
5. *Koza J.R.* Genetic programming. On the Programming of Computers by Means of Natural Selection. MA. The MIT Press. 1998.
6. *Axelrod R.* The Evolution of Cooperation. NY: Basic Books, 1984.
7. *Fogel. D.* The evolution of intelligent decision making in gaming // Cybernetics and Systems. 2001. V. 22, pp. 223-236.
8. *Ashlock D., Wittrock A., Tsui-Jung W.* Training finite state machines to improve PCR primer design / Proceedings of the Congress on Evolutionary Computation CEC'02. DC: IEEE Computer Society. 2002. V.1, pp. 13-18. http://ieeexplore.ieee.org/iel5/9601/30334/01393953.pdf
9. *Полимеразная цепная реакция.* http://en.wikipedia.org/wiki/Polymerase_chain_reaction
10. *MacLennan B.* Synthetic Ethology: An Approach to the Study of Communication / Proceedings of Artificial Life II: The Second Workshop on the Synthesis and Simulation of Living. CA: Addison Wesley. 1992. V. X, pp. 631-658. http://www.cs.utk.edu/~mclennan/papers/SEASC.pdf
11. *Frey C., Leugering G.* Evolving Strategies for Global Optimization – A Finite State Machine Approach / Proceedings of the Genetic and Evolutionary

Computation Conference (GECCO-2001). Morgan Kaufmann. 2001. pp. 27-33. http://citeseer.ist.psu.edu/456373.html

12. *Wolff K., Nordin P.* An Evolutionary Based Approach for Control Programming of Humanoids / Proceedings of the 3rd International Conference on Humanoid Robots (Humanoids'03). Karlsruhe: VDI/VDE-GMA. 2003. http://citeseer.ist.psu.edu/641298.html
13. *Николенко С. И.* Скрытые марковские модели. http://logic.pdmi.ras.ru/~sergey/teaching/mlbayes/07-hmm2.pdf
14. *Rabiner L. R.* A tutorial on hidden markov models and selected applications in speech recognition / Proceedings of the IEEE. 1989, vol. 77, no. 2, pp. 257 – 286. http://ieeexplore.ieee.org/iel5/5/698/00018626.pdf?arnumber=18626
15. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ. М.:МЦНМО, 2002.
16. *Dempster A. P., Laird N. M., Rubin D. B.* Maximum likelihood from incomplete data via the EM algorithm // Stat. Soc. 1977, vol. 39, no 1, pp. 1 – 38.
17. *Бедный Ю.Д.* Применение генетических алгоритмов для построения клеточных автоматов. Бакалаврская работа. 2006. http://is.ifmo.ru/papers/genalgcelaut/
18. *Гач П., Курдюмов Г.Л., Левин Л.А.* Одномерные однородные среды, размывающие конечные острова // Проблемы передачи информации. 1976. 14, 3.
19. *Ferreira C.* Discovery of the Boolean Functions to the Best Density Classification Rules Using Gene Expression Programming // Lecture Notes in Computer Science. 2002. Vol. 2278, pp. 51-60. www.gene-expression-programming.com/webpapers/ferreira-EuroGP02.pdf
20. *Won K., Hamelryck T., Prugel-Bennett A., Krogh A.* Evolving Hidden Markov Models for Protein Secondary Structure Prediction / Proceedings of the IEEE

(Evolutionary Computation). 2005, vol. 1, pp. 33 – 40. http://modem.ucsd.edu/won/77.pdf

21. *Поспелов Д. А.* Вероятностные автоматы. М.: Энергия, 1970.
22. *Понтрягин Л. С., Болтянский В. Г., Гамкрелидзе Р. В., Мищенко Е. Ф.* Математическая теория оптимальных процессов. М., 1969.
23. *Красовский Н. Н.*, Теория управления движением, М., 1968.
24. Задача оптимального управления. *Wikipedia*. http://en.wikipedia.org/wiki/Optimal_Control.
25. Технология генетического программирования для генерации автоматов управления системами со сложным поведением. Промежуточный отчет по этапу III. 2008.
26. Технология генетического программирования для генерации автоматов управления системами со сложным поведением. Промежуточный отчет по этапу I. 2007. http://is.ifmo.ru/genalg/_2007_01_report-genetic.pdf
27. Официальный сайт игры *Robocode*. http://robocode.sourceforge.net/
28. *Кузнецов Д. В., Шалыто А. А.* Система управления танком для игры "*Robocode*". Вариант 2. СПбГУ ИТМО. 2003. http://is.ifmo.ru/projects/robocode2/
29. *Туккель Н. И., Шалыто А. А.* Система управления танком для игры "*Robocode*". Вариант 1. Проектная документация. СПбГУ ИТМО. 2003. http://is.ifmo.ru/projects/tanks/
30. История игры *Robocode*. http://robowiki.net/cgi-bin/robowiki?History
31. *Gade M., Knudsen M., et al.* Applying Machine Learning to *Robocode*. 2003 www.csc.calpoly.edu/~team14fk/F04/dat3_report.pdf
32. *Eisenstein J.* Evolving robot tank fighters. Technical Report AIM-2003-023, AI Lab, MIT. 2003. http://www.ai.mit.edu/people/jacobe/research/robocode/genetic_tanks.pdf

ПРИЛОЖЕНИЕ

Листинг 1. Реализация автоматизированного танка

```
package my.robocode.all;
import my.robocode.TankControlSystem;
import my.robocode.Environment;
import robocode.AdvancedRobot;
import robocode.HitByBulletEvent;
import robocode.ScannedRobotEvent;
import robocode.util.Utils;

public class SimpleTank extends AdvancedRobot {

    private static final double RADAR_ANG_STEP = Math.PI / 4;

    private long t, tHit;
    private double x, y, h, v, e;

    // Интерпретатор символического описания системы управления
    private TankControlSystem control;

    // Составляющие объекта управления
    private Driver driver = new Driver();
    private Gun gun = new Gun();


    public void run() {
        setAdjustRadarForGunTurn(true);
        x = y = h = v = e = 0;
        t = tHit = -1;
        control.newRound();

        // Реализация шага работы среды Robocode
        while (true) {
            if (getRadarTurnRemainingRadians() == 0) {
                setTurnRadarRightRadians(RADAR_ANG_STEP);
            }
            if (t == -1) {      // enemy doesn't 'locked' yet
                doNothing();
                continue;
            }

           // Входные воздействия, получаемые из среды Robocode
            Environment env = new Environment(x - getX(), y - getY(), v, h,
getTime() - t, getX(), getY(), getVelocity(), getHeadingRadians(),
getGunHeadingRadians(), getEnergy(), e, getTime() - tHit, minWall(),
getDistanceRemaining(), getTurnRemainingRadians());

         // Система управления по входным воздействиям формирует четыре
         // выходных воздействия (act[]) на объекты управления (gun, driver)
            double[] act = control.execute(env.values());

           // Передача сформированных системой управления выходных воздействий
            // на объект управления
            gun.turn(this, act[0]);
            if (getGunHeat() == 0 && act[1] > 0) {
                gun.fire(this, act[1]);
            }
```

```
            driver.go(this, act[2]);
            driver.turn(this, act[3]);

            execute();
        }
    }

    // Обработка события – "Обнаружен танк соперника"
    public void onScannedRobot(ScannedRobotEvent event) {
        double d = event.getDistance();
        double a = Utils.normalAbsoluteAngle(getHeadingRadians() +
event.getBearingRadians());
        x = d * Math.sin(a) + getX();
        y = d * Math.cos(a) + getY();
        t = event.getTime();
        v = event.getVelocity();
        h = event.getHeadingRadians();
        e = event.getEnergy();
    }

    // Обработка события – "Попадание снаряда"
    public void onHitByBullet(HitByBulletEvent e) {
        tHit = e.getTime();
    }

    // Вспомогательная функция – вычисление расстояния до края поля
    private double minWall() {
        return Math.min(Math.min(getX(), getBattleFieldWidth() - getX()),
Math.min(getY(), getBattleFieldHeight() - getY()));
    }

}

// Объект управления "Танк"
// Система движения
class Driver {

    // Движение
    public void go(AdvancedRobot tank, double dist) {
      tank.setAhead(dist);
    }

    // Поворот
    public void turn(AdvancedRobot tank, double ang) {
      tank.setTurnRightRadians(ang);
    }
}

// Пушка
class Gun {

    // Стрельба
    public void fire(AdvancedRobot tank, double energy) {
      tank.setFire(energy);
    }

    // Поворот
    public void turn(AdvancedRobot tank, double ang) {
      tank.setTurnGunRightRadians(ang);
    }
}
```