Федеральное агентство по образованию

УДК 004.4’242
ГРНТИ 28.23.29
Инв. № 390173-1

ПРИНЯТО:	УТВЕРЖДЕНО:
Приемочная комиссия Государственного заказчика:	**Государственный заказчик Федеральное агентство по образованию**
От имени Приемочной комиссии **_____/Мосичева И.А. /**	**От имени Государственного заказчика** **_____/Бутко Е.Я./**

НАУЧНО-ТЕХНИЧЕСКИЙ ОТЧЕТ

о выполнении 1 этапа Государственного контракта № П2236 от 11 ноября 2009 г.

Исполнитель: Государственное образовательное учреждение высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики»
Программа (мероприятие): Федеральная целевая программа «Научные и научно-педагогические кадры инновационной России» на 2009-2013 гг., в рамках реализации мероприятия № 1.2.1 Проведение научных исследований научными группами под руководством докторов наук.
Проект: Применение методов искусственного интеллекта в разработке управляющих программных систем
Руководитель организации: Васильев Владимир Николаевич
Руководитель проекта: Шалыто Анатолий Абрамович

Согласовано: **Управление научных исследований и инновационных программ** **От имени Заказчика** **_____/Кошкин В.И./**

Санкт-Петербург
2009 г.

СПИСОК ОСНОВНЫХ ИСПОЛНИТЕЛЕЙ

по Государственному контракту № П2236 от 11 ноября 2009 г.
на выполнение поисковых научно-исследовательских работ для государственных нужд

Организация-Исполнитель: Государственное образовательное учреждение высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики»

Руководитель темы:

доктор технических наук, профессор	_____ подпись, дата	Шалыто А. А.

Исполнители темы:

кандидат технических наук, без ученого звания	_____ подпись, дата	Шопырин Д. Г.
кандидат технических наук, без ученого звания	_____ подпись, дата	Гуров В. С.
кандидат технических наук, без ученого звания	_____ подпись, дата	Корнеев Г. А.
без ученой степени, без ученого звания	_____ подпись, дата	Царев Ф. Н.
без ученой степени, без ученого звания	_____ подпись, дата	Данилов В. Р.
без ученой степени, без ученого звания	_____ подпись, дата	Клебан В. О.

без ученой степени, без ученого звания	_____ подпись, дата	Яминов Б. Р.
без ученой степени, без ученого звания	_____ подпись, дата	Кочелаев Д. Ю.
без ученой степени, без ученого звания	_____ подпись, дата	Гниломедов И. И.
без ученой степени, без ученого звания	_____ подпись, дата	Соколов Д. О.
без ученой степени, без ученого звания	_____ подпись, дата	Давыдов А. А.
без ученой степени, без ученого звания	_____ подпись, дата	Чеботарева Ю. К.
без ученой степени, без ученого звания	_____ подпись, дата	Царев М. Н.
без ученой степени, без ученого звания	_____ подпись, дата	Малаховски Я. М.
без ученой степени, без ученого звания	_____ подпись, дата	Буздалов М. В.
без ученой степени, без ученого звания	_____ подпись, дата	Родиков Д. Е.

РЕФЕРАТ

Отчет 109 с., 4 гл., 45 рис., 26 табл., 118 источников.

Ключевые слова: искусственный интеллект; управляющие системы; программные системы; генетические алгоритмы; человекоподобные роботы.

В настоящем отчете излагаются результаты выполнения *поисковых научно-исследовательских работ по направлениям «Механика», «Информатика», «Математика» по проблеме «Применение методов искусственного интеллекта в разработке управляющих программных систем»*, выполняемых в рамках государственного контракта, заключенного между Федеральным агентством по образованию и государственным образовательным учреждением высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» в соответствии с решением *Единой комиссии* (протокол от 22 октября 2009 г. № 3/НК-408П) по конкурсу № НК-408П «Проведение поисковых научно-исследовательских работ по направлениям: «Механика», «Информатика», «Математика» в рамках мероприятия 1.2.1 «Проведение научных исследований научными группами под руководством докторов наук» по направлению 1 «Стимулирование закрепления молодежи в сфере науки, образования и высоких технологий» федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009 – 2013 годы, утвержденной постановлением Правительства Российской Федерации от 28 июля 2008 года № 568 «О федеральной целевой программе «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы».

Целями настоящего этапа являются:

1. Выполнение аналитического обзора.
2. Выбор и обоснование оптимального варианта направления исследований.
3. Подготовка плана проведения теоретических и экспериментальных исследований.
4. Разработка и программная реализация метода применения генетических алгоритмов для построения системы управления виртуальным человекоподобным роботом (на примере управления ходьбой).

При выполнении первого этапа работ использовался следующий инструментарий:

1. Статьи в ведущих зарубежных и российских журналах, монографии и патенты за период 1998–2008 гг.
2. Результаты автоматического поиска в сети Интернет по ключевым словам и шаблонам.
3. Работы членов коллектива, опубликованные в рамках НИР по схожей тематике.
4. Аналитический обзор, составленный в рамках НИР.
5. Постановление Правительства Российской Федерации от 4 мая 2005 г. № 284 «О государственном учете результатов научно-исследовательских, опытно-конструкторских и технологических работ гражданского назначения».
6. Язык программирования *Java*.
7. Интегрированная среда разработки *Eclipse*.
8. Система трехмерного моделирования *Webots*.
9. Персональный компьютер.
10. Часть 4 Гражданского кодекса Российской Федерации.
11. ГОСТ 7.32-2001 «Отчет о научно-исследовательской работе. Структура и правила оформления».

Излагаются результаты выполнения аналитического обзора по следующим направлениям: методы поиска при условиях ограничений в разработке программного обеспечения, обучение с

подкреплением, нечеткие множества и нечеткая логика, искусственные нейронные сети, построение систем управления мобильными роботами, управление человекоподобными роботами, мультиагентные системы.

Приводится обоснование выбора оптимального варианта направления исследований, а также план проведения теоретических и экспериментальных исследований.

Описывается метод применения генетических алгоритмов для построения системы управления виртуальным человекоподобным роботом (на примере управления ходьбой). Приводится его программная реализация и результаты экспериментального исследования.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящем отчете излагаются результаты выполнения *поисковых научно-исследовательских работ по направлениям «Механика», «Информатика», «Математика» по проблеме «Применение методов искусственного интеллекта в разработке управляющих программных систем»*, выполняемых в рамках государственного контракта, заключенного между Федеральным агентством по образованию и государственным образовательным учреждением высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» в соответствии с решением *Единой комиссии* (протокол от 22 октября 2009 г. № 3/НК-408П) по конкурсу № НК-408П «Проведение поисковых научно-исследовательских работ по направлениям: «Механика», «Информатика», «Математика» в рамках мероприятия 1.2.1 «Проведение научных исследований научными группами под руководством докторов наук» по направлению 1 «Стимулирование закрепления молодежи в сфере науки, образования и высоких технологий» федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009 – 2013 годы, утвержденной постановлением Правительства Российской Федерации от 28 июля 2008 года № 568 «О федеральной целевой программе «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы».

Целями настоящего этапа являются:

1. Выполнение аналитического обзора.
2. Выбор и обоснование оптимального варианта направления исследований.
3. Подготовка плана проведения теоретических и экспериментальных исследований.
4. Разработка и программная реализация метода применения генетических алгоритмов для построения системы управления виртуальным человекоподобным роботом (на примере управления ходьбой).

Отчет имеет следующую структуру. В первой главе приводятся результаты выполнения аналитического обзора по следующим направлениям:

- методы поиска при условиях ограничений в разработке программного обеспечения;
- обучение с подкреплением;
- нечеткие множества и нечеткая логика;
- искусственные нейронные сети;
- построение систем управления мобильными роботами;
- управление человекоподобными роботами;
- мультиагентные системы.

Во второй главе приводится обоснование выбора оптимального направления исследований.

В третьей главе приводится план проведения теоретических и экспериментальных исследований.

В четвертой главе приводится описание метода применения генетических алгоритмов для построения системы управления виртуальным человекоподобным роботом (на примере управления ходьбой), а также его программная реализация на языке программирования *Java*.

Каждая из глав снабжена выводами, кратко резюмирующими содержание главы. В заключении дается общая оценка работ по этапу.

Исследования, проводимые, например, в университетах Йорка, Брунеля (Великобритания), в исследовательском центре *British Telecom*, в университете Дрекслера (Филадельфия, США) показывают, что методы поиска в условиях ограничений могут успешно применяться для решения ряда задач, возникающих при разработке программных систем. В число таких задач входят, например, автоматическое построение тестов, разбиение программ на модули, прогнозирование времени отклика.

Методы поиска в условиях ограничений, в свою очередь, являются одним из разделов искусственного интеллекта. Среди других разделов искусственного интеллекта, которые могут найти применение при построении управляющих программных систем, можно назвать: мультиагентные системы, обучение с подкреплением, искусственные нейронные сети, робототехника, методы поиска при условиях ограничений, представление знаний, логический вывод, планирование действий, методы вероятностных рассуждений и рассуждений в условиях неопределенности, в том числе нечеткие множества и нечеткая логика, обработка естественного языка и машинное зрение.

Одним из важнейших направлений искусственного интеллекта является создание роботов. Исследования по этому направлению ведутся в большом числе научных центров и университетов во всем мире.

Во многих работах методы искусственного интеллекта применяются либо при программной реализации управляющей системы, либо для решения задач обеспечения процесса разработки. В настоящей работе делается попытка распространить область применения методов искусственного интеллекта на задачи автоматизированного построения управляющих программных систем. При этом в качестве предметной области выбрано построение систем управления малоразмерными мобильными роботами.

Изложенное позволяет утверждать, что результаты выполнения научно-исследовательской работы будут превышать мировой уровень разработок в рассматриваемой области.

1. АНАЛИТИЧЕСКИЙ ОБЗОР

В соответствии с книгой [19] в настоящее время в искусственный интеллект входят следующие разделы:

- мультиагентные системы;
- обучение с подкреплением;
- искусственные нейронные сети;
- робототехника;
- методы поиска при условиях ограничений;
- представление знаний;
- логический вывод;
- планирование действий;
- методы вероятностных рассуждений и рассуждений в условиях неопределенности, в том числе нечеткие множества и нечеткая логика;
- обработка естественного языка;
- машинное зрение.

Наиболее важными в разработке управляющих программных систем представляются рассматриваемые в настоящей главе направления:

- методы поиска при условиях ограничений в разработке программного обеспечения;
- обучение с подкреплением;
- нечеткие множества и нечеткая логика;
- искусственные нейронные сети;
- построение систем управления мобильными роботами;
- управление человекоподобными роботами;
- мультиагентные системы.

1.1. МЕТОДЫ ПОИСКА ПРИ УСЛОВИЯХ ОГРАНИЧЕНИЙ В РАЗРАБОТКЕ ПРОГРАММНЫХ СИСТЕМ

В настоящем разделе описывается применение метаэвристических алгоритмов, или алгоритмов направленного поиска при разработке программных систем.

1.1.1. Введение

Многие задачи, возникающие в процессе разработки программных систем, практически не поддаются аналитическим решениям. Это связано с тем, что процесс разработки программных систем описывается множеством противоречащих ограничений, многие из которых определены недостаточно четко, и множеством параметров, сильно зависящих друг от друга. Даже незначительное изменение одного параметра может привести к существенным изменениям в нескольких смежных областях, что серьезно затрудняет процесс поиска оптимального решения проблемы.

С другой стороны, решение задачи, поставленной в рамках разработки программных систем, не всегда требует точного или оптимального решения. Чаще требуется найти «приемлемое» или «почти оптимальное» решение. Кроме того, далеко не всегда ясно, как найти оптимальное решение, но почти всегда можно определить, какое из двух предложенных решений более приемлемо.

Метаэвристические алгоритмы – семейство алгоритмов, задачей которых является поиск оптимального или близкого к нему решения задачи в многомерном пространстве поиска. Как правило, задачи, при решении которых используются такие алгоритмы, практически невозможно решить точно, однако два решения-кандидата можно сравнить друг с другом и выяснить, какое из них лучше. Метаэвристические алгоритмы успешно применяются для решения значительного числа

инженерных задач – от задачи распределения нагрузки и разводки печатных плат до управления летательными аппаратами.

1.1.2. Метаэвристические алгоритмы

В данном подразделе описаны некоторые хорошо известные и широко используемые метаэвристические алгоритмы.

1.1.2.1. Метод спуска

Наиболее простым из метаэвристических алгоритмов является *метод спуска* (*hill climbing*). Процесс поиска начинается со случайно выбранной точки в пространстве решений задачи. На каждой итерации рассматриваются точки, соседние с текущей. Если некоторая соседняя точка представляет собой более оптимальное решение, то она становится текущей на следующей итерации. В противном случае поиск считается оконченным, а текущая точка – оптимальным решением.

Главной проблемой метода спуска является то, что найденное решение является локальным оптимумом, который может оказаться гораздо хуже, чем глобальный. Однако метод спуска является достаточно простым алгоритмом, и во многих случаях его применение дает достаточно хорошие результаты, например при разбиении проекта на модули [57, 84] и оценке стоимости проекта [70].

У метода спуска существует множество вариаций. Например, вопрос о том, нужно ли выбирать первую соседнюю точку, оказавшуюся оптимальнее текущей, или нужно рассмотреть все соседние точки и выбрать среди них наиболее оптимальный вариант, порождает различные версии алгоритма (*first ascent hill climbing*, *steepest ascent hill climbing*). В любом случае, главным свойством всех реализаций метода спуска остается то, что он находит только локальные оптимумы. Этот недостаток преодолевается различными способами с помощью других метаэвристических алгоритмов.

1.1.2.2. Метод имитации отжига

Метод имитации отжига [80] также является локальным метаэвристическим алгоритмом поиска. В этом методе каждому потенциальному решению x сопоставляется функция стоимости E, которую требуется минимизировать. Чем больше функция стоимости, тем менее оптимальным считается данное решение. В частности, высокие значения функции стоимости присваиваются тем решениям, которые нарушают какие-либо ограничения, либо находятся достаточно близко от границ этих ограничений в пространстве поиска.

Метод имитации отжига также работает итеративно. Пусть на некоторой итерации текущее решение есть x. Вначале генерируется решение x_1, находящееся близко к решению x в пространстве поиска. Если значение функции стоимости для x_1 меньше, чем для x, то x_1 выбирается как текущее решение для следующей итерации. Если же оно превосходит значение функции стоимости для x, то x_1 выбирается как текущее решение с вероятностью $p=e^{-dE/T}$, где $dE = E(x_1)-E(x)$, а T – изменяющийся во времени параметр метода имитации отжига. Возможность выбора на некоторых итерациях менее оптимального решения связана с необходимостью избежать преждевременной сходимости к локальному, а не к глобальному оптимуму.

В начале работы метода имитации отжига T относительно велико, что означает сравнительно высокую вероятность выбрать менее оптимальное решение. В процессе поиска T уменьшается по некоторому закону, и через несколько итераций менее оптимальное решение выбирается уже со значительно меньшей вероятностью. Функцию, по которой T зависит от числа итераций, иногда называют «функцией убывания температуры», или «функцией охлаждения», по аналогии с физическим процессом отжига. Совокупность функции T от числа итераций, критерия останова алгоритма, а также распределения вероятностей, используемых для генерации решений, близких к данному, называется схемой метода имитации отжига.

Название метода имитации отжига возникло в работе [80], где была предпринята попытка использовать процесс, напоминающий процесс отжига, применяемый в металлургии. Этот процесс подразумевает нагрев металла или сплава до высокой температуры и охлаждения его по некоторому закону до комнатной температуры. Вероятность изменения положения атома в кристаллической решетке связана с потенциальной энергией решетки и температурой по закону Максвелла-Больцмана. По этой причине, при охлаждении металла по определенному закону общая потенциальная энергия решетки значительно уменьшается, что улучшает прочностные и иные характеристики металла.

На практике метод имитации отжига показывает хорошие результаты, однако настройка его для решения конкретной задачи может затратить значительное время. Эффективность метода существенно зависит от выбранной схемы, в частности, от начальной температуры и функции охлаждения.

1.1.2.3. Поиск с запретами

Алгоритм *поиска с запретами* (*tabu search*) [53] внешне напоминает метод спуска, но содержит механизм, препятствующий остановке алгоритма в локальном оптимуме. На каждой итерации этот алгоритм рассматривает множество решений, соседних с текущим решением в пространстве поиска. Далее, на некоторые из этих решений накладывается *запрет (tabu)* – это значит, что данное решение нельзя выбирать на данном шаге. Запреты помогают процессу поиска не зациклиться и не остановиться в локальном оптимуме. Среди оставшихся решений лучшее выбирается в качестве текущего решения для следующей итерации.

Например, на некотором шаге из решения X было получено решение Y с помощью действия v, а действием, обратным к v, является u. Тогда можно наложить запрет (Y, u, X) – запрещается из Y действием u перейти в X. Такой запрет может оказаться неэффективным вследствие того, что может быть найден путь, обходящий этот запрет. Поэтому может потребоваться наложить запрет (u, X), который не разрешает с помощью действия u переходить в X, или даже запрет (u), не разрешающий применять действие u.

Стратегии накложения запрета могут различаться, но общий их смысл состоит в том, чтобы, однажды выйдя из локального оптимума, не возвращаться в него в течение обозримого времени. Как правило, запреты хранятся в кратковременной памяти – по истечении некоторого времени запрет снимается.

Иногда чрезмерно строгие запреты приводят к стагнации процесса поиска. Для того чтобы этого не происходило, время от времени необходимо нарушать запреты. В некоторых случаях можно показать несостоятельность запрета, например, если запрещенное решение оптимальнее, чем лучшее из ранее найденных. В общем случае, после составления множества запрещенных решений предлагается выделить из них множество *многообещающих (aspirant)* решений и разрешить переход в них.

1.1.2.4. Генетические алгоритмы

Генетические алгоритмы [30, 61], в отличие от изложенных ранее подходов, выполняют поиск оптимальных решений сразу в нескольких точках пространства. Вначале случайным образом генерируется некоторое число решений (особей, в терминологии генетических алгоритмов), образующих начальную *популяцию*. Далее, особи этой популяции *скрещиваются* и *мутируют,* формируя новое поколение. Скрещивание (*кроссовер*, *рекомбинация*) – фундаментальная операция в генетических алгоритмах, обеспечивающая обмен генетическим материалом между особями. Мутация – не менее важная составляющая. Она позволяет получать новый генетический материал, а также предотвращать алгоритм от «застревания» в области локального оптимума.

В классическом генетическом алгоритме особь кодируется строкой над небольшим алфавитом (как правило, это битовая строка), по аналогии с хромосомой, кодирующей наследственную

информацию в живых организмах. По этой причине особь генетического алгоритма нередко также называют хромосомой. Для битовых строк известно несколько стандартных операций скрещивания и мутации, что, однако, не ограничивает возможные варианты выбора этих операций.

Приспособленность конкретной особи – то, насколько решение, представленное данной особью, удовлетворяет задаче, описывается *функцией приспособленности*, или *фитнес-функцией*.

Новая популяция выбирается из особей старого и нового поколения, следуя критерию отбора. Этот критерий отдает предпочтение более приспособленным особям, в то же время, давая шанс и менее приспособленным. Таким образом, в популяции поддерживается необходимое разнообразие особей. При этом лучшие особи выживают гораздо чаще. Можно сказать, что идея генетического алгоритма берет свои истоки в учении Чарльза Дарвина. Однако, эволюция Дарвина в некотором смысле «слепа» – улучшения в генотипе популяции происходят случайно. В генетических же алгоритмах улучшение особей в популяции является основной целью.

Эффективность генетических алгоритмов во многом зависит от выбора функции приспособленности. Она должна не только «поощрять» более оптимальные решения, но и «направлять» генетический алгоритм к этим решениям.

В целом, генетические алгоритмы определяют:

- алгоритмы скрещивания, мутации и селекции;
- определение вероятностей скрещивания и мутации;
- вид функции приспособленности.

1.1.2.5. Генетическое программирование

Генетическое программирование [72] – разновидность генетического алгоритма, которая ставит целью получение программы, решающей конкретную задачу. Функция приспособленности в данном случае определяется тем, насколько полученная программа хорошо решает поставленную задачу. Как правило, программа представляется в виде *дерева разбора.* Поэтому и операции скрещивания и мутации определены на деревьях разбора. По этой причине, они более оригинальны, посравнению с операциями, которые обычно используются в генетических алгоритмах.

Генетическое программирование может применяться для поиска функций, описывающих или предсказывающих данные. Как и в общем случае метаэвристических алгоритмов, от этих функции требуется не абсолютно точное, а скорее достаточно хорошее следование требованиям.

1.1.3. Использование метаэвристических алгоримов для решения задач разработки программного обеспечения

Основное преимущество метаэвристических алгоритмов в целом – это то, что они способны искать оптимумы функций, являющихся чрезмерно сложными и даже имеющих разрывы.

Для того, чтобы использовать метаэвристический алгоритм для решения какой-либо задачи, ее сначала необходимо сформулировать как задачу поиска или оптимизации. Например, как показано в разд. 1.1.4, задача автоматического поиска тестов может быть сформулирована как поиск теста в некотором пространстве возможных тестов с целью оптимизации используемого критерия качества теста. После того, как необходимая формулировка выполнена, необходимо определить тип метаэвристического алгоритма, а также его параметры.

Для любого метаэвристического алгоритма потенциальное решение должно быть соответствующим образом закодировано. Также, должна быть определена функция, определяющая, насколько данное решение подходит. Она называется «функция приспособленности» или «функция стоимости». Эта функция определяет «ландшафт», на котором будет производиться поиск. Если на этом ландшафте будут слишком острые «пики» или, наоборот, неровности будут незначительными, то поиск по этому ландшафту будет затруднен. Следовательно, при использовании

метаэвристических алгоритмов при разработке программного обеспечения значительные усилия должны быть затрачены на определение и описание подходящих функций приспособленности.

В рассматриваемых алгоритмах применяется концепция «изменения решения». Оно может быть как «малым перемещением», сохраняющим основные свойства решения, так и «большой мутацией», которое может существенно отличаться от начального. Доступные операции изменения решения определяются методом представления решения в виде точки в пространстве поиска.

Определение представления, функции приспособленности и операции изменения решения – необходимые условия для реализации любого из описанных алгоритмов, поэтому при применении любого метаэвристического алгоритма нужно начинать с этих определений. После этого становится возможным применение техник локального поиска, таких как метод спуска и метод имитации отжига. При необходимости применить генетические алгоритмы необходимо также реализовать операцию скрещивания.

Во многих задачах решение определяется совокупностью вещественных или целочисленных параметров. В этом случае «малое изменение» параметра целесообразно реализовать как следующее или предшествующее значения для целочисленных параметров или как произвольное малое приращение для вещественных параметров. Изменение же всего решения в целом определяется как изменение одного или нескольких параметров этого решения.

Оператор скрещивания, как правило, принимает в качестве аргументов двух особей (родителей) и возвращает одну или двух особей (потомков). Оператор должен быть устроен таким образом, чтобы у потомков в каком-либо виде сохранялись черты родителей.

Общие рекомендации по применению метаэвристических алгоритмов для решения задач разработки программного обеспечения звучат следующим образом:

1. Определить, достаточно ли данная задача сложна, чтобы решать ее метаэвристическими алгоритмами (является ли пространство поиска чрезмерно большим, для того чтобы использовать на нем традиционные алгоритмы, применяемые в задачах такого рода).
2. Определить представление потенциальных решений, позволяющее использовать метаэвристические алгоритмы.
3. Определить функцию приспособленности.
4. Начать с применения метода спуска. Если он дает лучшие результаты, чем случайный поиск, то имеет смысл применить иные метаэвристические алгоритмы.

Содержание последнего пункта определяется тем, что во многих задачах даже метод спуска часто дает удовлетворительные результаты [57, 70, 83, 84]. Существуют, однако, и такие задачи [111], в которых генетические алгоритмы работают значительно лучше метода спуска. Однако даже для подобных задач применение метода спуска может быть начальной точкой для анализа задачи. Если применение метода спуска не дает хороших результатов, то это может означать, что задача проанализирована недостаточно тщательно или используемое представление является неэффективным. Возможно, это также означает, что данная задача не может быть удовлетворительно решена метаэвристическими алгоритмами в целом.

Задачи, которые возможно эффективно решать метаэвристическими алгоритмами, как правило, обладают следующими характеристиками:

- Большое пространство поиска решений. Если пространство поиска достаточно мало для того, чтобы все решения можно было перебрать, нет смысла использовать более продвинутые техники.
- Не существует известных эффективных решений данной задачи. Действительно, не имеет смысла применять новые, заведомо менее эффективные алгоритмы для уже решенной задачи. Однако, если существующие решения покрывают лишь часть возможных входных данных, применение метаэвристик может помочь для того, чтобы рассмотреть оставшиеся варианты входных данных.

- Существование подходящей функции приспособленности. Например, во многих отраслях разработки программного обеспечения существует множество метрик, подходящих для этой цели.
- Приемлемое время для генерации новых потенциальных решений, а также для вычисления функции приспособленности.

Ниже приведены свойства задач, которые делают метаэвристические алгоритмы неприемлемыми для поиска решений:

- связь задачи с выявлением требований к проектируемой системе;
- необходимость диалога программы и человека;
- нечеткие требования или частичное их отсутствие.

1.1.4. Применение метаэвристических алгоритмов при тестировании программного обеспечения

При генерации тестов для программного обеспечения, особями, которые должны быть оптимизированы, естественно, являются тесты. Они должны быть представлены в таком формате, который позволяет производить мутацию, скрещивание и малое изменение особи.

Как правило, при тестировании стремятся оптимизировать некоторые численные критерии, оценивающие набор тестовых данных. Множество методов автоматической генерации тестов направлены на улучшение качества покрытия кода или спецификации. Качество покрытия может быть измерено и включено тем или иным образом в функцию приспособленности.

1.1.4.1. Структурное тестирование

Структурное тестирование, или тестирование «белого ящика», ставит задачу определить адекватность набора тестов в том случае, когда структура кода известна. Как правило, качество набора тестов в структурном тестировании измеряется долей конструкций языка, которые исполняются в процессе тестирования. Эта доля называется *покрытием* (*coverage*) кода. Для данной программы разумно требовать полного покрытия – того, что все конструкции кода выполняются во время исполнения хотя бы одного теста.

Множество видов покрытия основаны на применении графа управляющей логики программы. Перечислим некоторые из них:

- *покрытие инструкций* (*statement coverage*) – доля инструкций в коде, исполняемых в процессе тестирования;
- *покрытие ветвлений* (*branch coverage*) – доля ветвлений (ребер в графе управляющей логики, которые исходят из вершины с исходящей степенью, большей единицы), покрытых в процессе тестирования;
- *покрытие путей* (*path coverage*) – доля возможных путей в графе управляющей логики, пройденных в процессе тестирования.

Рассмотрим какой-либо вид покрытия. Генерация случайных тестов может привести к высокой доле покрытия. Тем не менее, в коде могут встречаться инструкции, которые практически невозможно достичь с использованием случайных тестов. С учетом этого, разумно проводить тестирование в два этапа: использовать случайные тесты для того, чтобы покрыть большую часть кода, а затем использовать более сложные алгоритмы с целью создать тесты, покрывающие остальные участки кода. Во многих работах предлагается использование метаэвристик для достижения этой цели [67, 68, 93, 104].

Пусть некоторый участок кода еще не был покрыт в процессе тестирования. В этом случае тестировщик может выбрать некоторый путь до этого участка кода и попытаться «вырастить» тест, который соответствует этому пути. В качестве примера рассмотрим код на рис. 1. Если необходимо

```
if (a > b + c || b > a + c || c > a + b) {
        out = 'not a triangle';
} else if (a == b && b == c) {
        out = 'equilateral';
} else if (a == b || b == c || a == c) {
        out = 'isosceles';
} else {
        out = 'scalene';
}
```

Рис. 1. Пример фрагмента кода для тестирования

протестировать ветку `then` последнего оператора `if`, достаточно добиться того, чтобы в первых двух операторах `if` условие оператора стало равным `false`, а в последнем – `true`.

Возможно, что путь, выбранный тестировщиком, не может быть пройден ни одним тестом. В этом случае, если процесс поиска необходимого теста затягивается на неопределенное время, он должен быть остановлен и необходимо выбрать новый путь. Функция приспособленности может быть определена, по тому, насколько близко путь исполнения приближается к намеченному пути (при этом подсчитывается, например, число общих предикатов у этих путей [93]).

Вновь обратимся к рис. 1. Допустим, необходимо протестировать ветку `then` последнего оператора `if`, а тест проходит в ветку `else` первого оператора и в ветку `then` второго. В этом случае, его функция приспособленности равна единице.

Если критерий качества тестов задан, то функция приспособленности может быть улучшена. Например, в работах [67, 68], где рассматривается применение генетических алгоритмов к тестированию ветвлений, отдельно выделяют случай, когда путь исполнения теста доходит до вершины графа управляющей логики, в которой происходит тестируемое ветвление. В этом случае, для теста вычисляется дополнительная функция приспособленности. Она зависит от того, в какую из возможных веток оператора ветвления идет исполнение теста.

Если при исполнении теста в данной вершине выбирается неверное направление, то функция приспособленности зависит от того, насколько переменные, определяющие направление, близки к тому, чтобы в совокупности выбрать верное направление. Чем ближе переменные к верному набору, тем большим будет значение приспособленности. Например, если на рис. 1 необходимо добиться выбора ветки `then` во втором операторе `if`, а процесс исполнения теста проходит в ветку `else`, то функцию приспособленности разумно выбрать обратно пропорциональной тому, насколько `a` близко к `b` и `b` близко к `c`: $f = \frac{1}{1+|a-b|+|b-c|}$

Если в вершине выбирается верное направление, функция приспособленности тем выше, чем ближе значения условий выбора к пограничным. Это обуславливается тем, что, как правило, тесты, которые направлены на тестирование вокруг пограничных значений, выявляют больше ошибок.

Функции приспособленности, описанные в работах [67, 68, 93], возможно, могут быть скомбинированы. Например, для того, чтобы достичь тестируемого условного оператора, может быть использована функция из работы [93] – глобальная функция, в то же время, когда оператор достигнут, имеет смысл использовать локальную функцию из работ [67, 68].

1.1.4.2. Тестирование времени отклика

Системы управления реального времени обычно разрабатываются с тем условием, чтобы отклик на воздействие был выполнен в некоторый фиксированный промежуток времени. Если формирование отклика затрачивает больше времени, это может привести к критическим ошибкам в системе. Поэтому часто важно определить наихудшее время выполнения той или иной функции или метода. Эта задача, как правило, чрезвычайно сложна, поскольку во многом зависит от используемого оборудования или компилятора.

Генетические алгоритмы в работах [112, 113] использовались при поиске минимального и максимального времени выполнения функции. Для тестовых данных, описанных хромосомой, функция приспособленности зависит от времени выполнения функции на этих данных: если требуется найти максимальное время выполнения, то она пропорциональна этому времени, если же требуется найти минимальное время, то она обратно пропорциональная времени выполнения. Похожим образом был использован метод отжига [105]. Эти методы не только работают существенно лучше случайного поиска, но и способны находить новые результаты [113].

1.1.5. Методы построения функций приспособленности

На основании изложенного можно сделать вывод о том, что при применении методов поиска в условиях ограничений, решающее значение имеет функция приспособленности, а также метод представления решений. В настоящем разделе рассматриваются методы построения функций приспособленности для построения конечных автоматов для управляющих систем.

1.1.5.1. Анализ существующих методов построения функций приспособленности

Существует множество способов задания функций приспособленности (оценочных функций). Например, оценочная функция может быть заданной явно – возвращать вещественное значение, зависящее только от определенной особи, которое постоянно для этой особи. Примером такой функции может быть функция, получающая на вход битовую строку, задающую некоторый алгоритм, и возвращающая число, соответствующее эффективности этого алгоритма. В этом случае приспособленность особи определяется ее способностью решать поставленную задачу, например, отслеживать путь [66].

Оценочная функция также может основываться на ранге битовых строк в популяции [114]. К сожалению, в том случае, когда для кодирования особей используется способ, отличный от битовых строк, применение этой функции становится затруднительным.

Можно построить оценочную функцию, в основе которой будет лежать на один из методов отбора. Например, турнирный отбор [55].

В тех случае, когда структуры, использующиеся для кодирования особей, имеют переменную длину, может потребоваться найти такое решение поставленной задачи, длина которого будет минимальной или хотя бы ограниченной разумными пределами. Это несложно сделать, если включить в оценочную функцию размер структуры, использующейся для кодирования особи. Пусть f_i определяет на сколько хорошо i-я особь решает поставленную задачу, а $length_i$ – размер структуры, использующейся для кодирования i-ой особи. В качестве оценочной функции можно использовать такую функцию F_i, что

1. $F_i < F_j$ для любых i и j, при которых $f_i < f_j$;
2. $F_i < F_j$ для любых i и j, при которых $f_i = f_j$ и $length_i < length_j$;
3. $F_i = F_j$ для любых i и j, при которых $f_i = f_j$ и $length_i = length_j$.

Построенная таким образом оценочная функция, в первую очередь, отдает предпочтение тем особям, которые лучше решают поставленную задачу. При этом она также учитывает и размер

структуры, использующейся для кодирования особи. Примером задачи, в которой была бы полезна такая оценочная функция, может служить любая задача, решаемая с помощью представления в виде конечного автомата. При этом, как правило, желательно построить автомат с минимальным числом состояний. Число состояний автомата можно использоваться вместо длины структуры, задающей особь.

Оценочная функция может быть определена через взаимодействие особей с окружающей средой. В этом случае значение оценочной функции для конкретной особи изменяется в зависимости от окружающей среды. Один из путей построения такой оценочной функции состоит в проведении соревнований между особями популяции. В работе [61] Дж. Холланд комбинирует совместную эволюцию и генетические алгоритмы в системе. Цель его работы – исследование совместной эволюцию организмов, заданных строками фиксированной длины, в «миниатюрном мире». В рассматриваемой системе всего одна популяция искусственных организмов. Окружающая среда для каждого организма состоит из всех остальных организмов.

Еще один пример оценочной функции, зависящей от всех особей в текущем поколении, приведен в работе [97]. В этой работе битовые строки, задающие особей, интерпретируются, как стратегии для игры в пятнашки. В указанной игре принимает участие два или более игроков, один из которых назначается преследователем. Преследователь пытается догнать других игроков, которые, в свою очередь, хотят убежать от него. Такое поведение часто встречается в природе. Большинство взаимодействий типа «хищник – жертва» включают в себя преследование и бегство. Игра в пятнашки включает в себя смену ролей. Для успешной игры требуется, как умение догонять, так и умение убегать. Цель преследователя догнать другого игрока. Когда преследователь приближается на достаточно близкое расстояние к другому игроку, роли меняются и беглец становится преследователем. Цель работы [97] – автоматически построить контроллер для игры в пятнашки, используя эволюционный процесс, оценочная функция которого основана только на результатах соревнований между контроллерами.

В традиционной игре в пятнашки нет формального метода для подсчета очков. Интуитивно понятно, что цель игры – не стать преследователем. Выигравшей в соревновании между двумя котроллерами, считается тот, для которой остается беглецом больший процент времени (число шагов) от общего времени игры. При таком подходе используется нормализованная мера определения приспособленности для одного соревнования: ноль – для худшего контроллера, единица – для лучшего. Для определения эффективности контроллера он сравнивается с другими контроллерами.

Для сравнения контроллеров выполняются две серии из четырех игр между ними. В первой серии первый контроллер начинает игру преследователем, во второй серии – второй. Перед каждой игрой начальные позиции и направления игроков задаются случайным образом. Из-за этого игры между одними и теми же игроками могут приводить к разным результатам. Каждая игра состоит из 25 шагов. Очки, которые получает контроллер за игру – число шагов, в течение которых он не был преследователем, деленное на 25.

Для определения приспособленности контроллера он участвует в сериях игр против шести других игроков. Оппоненты выбираются с помощью равномерного случайного отбора. Очки, полученные во всех играх, усредняются для определения окончательного значения оценочной функции. Значение оценочной функции, равное 50%, означает, что способности контроллера сопоставимы со средними способностями особей в популяции. Значение оценочной функции, превышающее 50%, означает, что способности контроллера выше среднего уровня способностей особей в популяции. Так как противники выбираются случайным образом, а начальные параметры игр рандомизированные, то значение оценочной функции может существенно варьироваться и позволяет получить только грубую оценку фактической эффективности особи.

Несколько исследователей (например, в работах [29, 48, 82]) экспериментировали с итерированной дилеммой узника. В этой задаче оценочная функция основывается на таблице баллов

для следующих ситуаций: сотрудничество, противостояние, противостояние-сотрудничество. Особи с различными стратегиями из эволюционирующей популяции играют друг против друга. Необходимо отметить, что в работе [29] эволюционирующие стратегии играли против девяти лучших компьютерных программ, предложенных на международном компьютерном чемпионате. Лучшая стратегия для одного игрока (стратегия задавалась строкой из 70 бит с возможностью заглядывать на три хода назад) была получена в результате соревнований с девятью лучшими стратегиями, построенными вручную. Эти стратегии и задавали окружающую среду, к которой пыталась приспособиться популяция. В работах [48, 82] оценочная функция определялась через взаимодействие особей с изменяющимся окружением, которое представлено в виде эволюционирующей популяции. В работе [82] популяция состояла из автоматов с 16 состояниями, которые задавались с помощью битовых строк длиной в 148 символов. Каждая строка в популяции представляла собой полную стратегию, руководствуясь которой игрок принимал решения. Автомат определял выбор хода игроком для любой последовательности ходов противника.

Как отмечалось выше, другой путь для создания изменяющегося окружения – использование конкурирующих популяций. В работе [60] эволюционировали две популяции: популяция сортирующих сетей и популяция перестановок чисел. Каждая перестановка состояла из 16 элементов. Чем в большем числе перестановок из конкурирующей популяции может правильно упорядочить элементы сортирующая сеть, тем выше ее относительная эффективность (приспособленность к среде). И наоборот, относительная приспособленность перестановки чисел определяется числом сортирующих сетей, которые не могут упорядочить ее элементы. Каждая популяция создает окружающую среду, к которой пытается приспособиться конкурирующая популяция. Каждая особь в каждой популяции участвует в определенном числе состязаний с особями конкурирующей популяции. Значение оценочной функции для каждой сортирующей сети равно числу тестовых перестановок, элементы которых она правильно упорядочила в процессе соревнований. Значение оценочной функции для каждой перестановки равно числу сортирующих сетей, которые не смогли правильно упорядочить ее элементы в процессе соревнований. В работе [60] отмечается, что каждая популяция эволюционируя, училась эксплуатировать слабости и избегать сильных сторон противоборствующей популяции. Такой подход, напоминающий природную модель «хищник – жертва», позволяет избежать необходимости тестировать каждую сортирующую сеть на всех перестановках 16 элементов для того, чтобы определить абсолютную эффективность каждой особи.

Еще одна оценочная функция предлагается в работе [96]. В ней генетический алгоритм используется для создания самовоспроизводящихся программ. В данном подходе особи располагаются в *Tierra's Reaper* -очередях. Особи в начале очереди, как правило, наиболее старые или не способные к самовоспроизведению. Они удаляются, когда становится мало памяти. В такой системе оценочная функция определяется через *Reaper*. Автор данного подхода (тропический эколог) моделировал с помощью компьютера экологические системы – повторяющие системы, найденные в природе.

Одна из основных целей при разработке нового или улучшении существующего генетического алгоритма – увеличение скорости, с которой генетический алгоритм сходится к лучшему решению задачи. В связи с тем, что вычисление оценочной функции, как правило, является наиболее трудоемкой частью алгоритма, эффективность алгоритма часто измеряется в виде числа вычислений оценочной функции, которое было сделано до получения лучшего решения. Для любого генетического алгоритма производительность может сильно зависеть от структуры пространства поиска, оценочной функции и от других его параметров. Например, от размера популяции и вероятности применения оператора мутации.

В работе [63] демонстрируется эффективность инкрементальной эволюции, при которой избирательность оценочной функции увеличивается поэтапно.

В работе [64] рассматриваются оценочные функции и способы их адаптации для увеличения эффективности генетических алгоритмов. В работе [75] показано, что если оценочная функция может быть представлена в качестве линейной комбинации отдельных оценочных функций, то можно настроить коэффициенты в этой комбинации в процессе работы генетического алгоритма таким образом, что скорость, с которой сходится алгоритм, существенно увеличится.

Оценочная функция определят направление поиска для генетического алгоритма. Если операторы скрещивания и мутации являются движущей силой эволюции, то оценочная функция определят цель эволюции и пути достижения этой цели. Для того чтобы генетический алгоритм смог найти «идеальное» решение при построении оценочной функции необходимо учесть все характеристики решения, важные для исследователя.

В том случае, когда целью является изучение эволюционных процессов (получение идеального решения отходит на второй план) используются оценочные функции, основанные на конкуренции особей. Применение совместной эволюции позволяет изучать такие особенности, как возникновение кооперации между особями популяции и взаимное влияние популяций на эволюцию друг друга. Оценочные функции, основанные на изменяющемся окружении, также полезны, если использование явно заданной оценочной функции невозможно [60].

В результате исследования установлено влияние оценочных функций на результаты применения генетических алгоритмов. Для повышения качества решений предлагается использовать методы, учитывающие специфику задачи при построении таких функций. Таким образом, работы по разработке рекомендаций по выбору оценочных функций являются актуальными.

1.1.5.2. Пример простейшей оценочной функции

В классическом генетическом алгоритме оценочная функция определяется как $f_i \diagup \overline{f}$, где f_i – приспособленность особи i, а $\overline{f}$ – средняя приспособленность всех особей в популяции. Значение оценочной функции используется для формирования промежуточной популяции особей (промежуточная популяция применяется для выбора родительских особей для скрещивания). Для каждой особи i, где $f_i \diagup \overline{f}$ больше единицы, целая часть числа обозначает, сколько копий этой особи гарантированно добавляется в промежуточную популяцию. Для всех особей (включая те, у которых $f_i \diagup \overline{f}$ меньше 1.0) в промежуточную популяцию добавляются их копии с вероятностью равной дробной части $f_i \diagup \overline{f}$. Например, одна копия особи для которой $f_i \diagup \overline{f} = 1.36$, сразу добавляется в промежуточную популяцию и еще одна копия добавляется с вероятностью 0.36. Копия особи, значение оценочной функции для которой $f_i \diagup \overline{f} = 0.54$, будет добавлена в промежуточную популяцию с вероятностью 0.54.

1.1.5.3. Построение оценочной функции для совместной эволюции популяций

Эволюционные процессы в природе не редко описывают так, как если бы популяция особей пыталась адаптироваться к неизменной окружающей среде. Такой подход является только первым уровнем приближения к реальной ситуации. Окружающая среда фактически состоит как из физической среды (которая обычно относительно постоянна), так и из независимо действующих биологических популяций особей, которые одновременно пытаются адаптироваться к их

окружающей среде. Действия каждой из этих независимо существующих биологических популяций (видов) обычно оказывает эффект на другие виды. Иными словами, окружающая среда отдельно взятого вида включает в себя все остальные биологические виды, которые обитают в той же физической среде, что и выбранный вид, в течение промежутка времени, когда он пытается адаптироваться к среде. В биологии термин совместная эволюция часто используется для того, чтобы подчеркнуть тот факт, что все виды, одновременно существующие в некоторой окружающей среде, совместно эволюционируют.

Биологический пример совместной эволюции представлен в работе [61]. Определенные виды растений могут столкнуться с окружающей средой, содержащей насекомых, которые питаются этими растениями. Защищаясь от хищников (увеличивая тем самым вероятность своего выживания), растения через определенное время, могут приобрести крепкую кожуру, которая будет мешать насекомым их поедать. Спустя некоторый период времени, насекомые могут приобрести сильные челюсти, которые позволят популяции насекомых продолжить есть растения и увеличат вероятность их выживания в окружающей среде. Через некоторое время растения могут научиться вырабатывать яд, который будет защищать их от насекомых. Однако насекомые снова через некоторое время, могут научиться вырабатывать пищеварительный фермент, нейтрализующий эффект яда. Благодаря этому популяция насекомых сможет продолжить питаться растениями.

В иерархическом алгоритме совместной эволюции используется две (иногда более) популяции особей. Окружающей средой для первой из них является вторая популяция. И наоборот, окружающей средой для второй популяции является первая популяция.

Процесс совместной эволюции обычно начинается с того, что обе популяции состоят из плохо приспособленных особей (если производить измерение абсолютной приспособленности). Затем первая популяция пытается приспособиться к окружающей среде, в качестве которой для нее выступает вторая популяция. В то же время вторая популяции пытается адаптироваться к окружающей среде, созданной первой популяцией.

Этот процесс осуществляется с помощью тестирования каждой особи из первой популяции с каждой особью из второй популяции (или из подмножества особей второй популяции). В результате такого тестирования определятся относительная (конкурентная) приспособленность особи. Относительная приспособленность одной особи из первой популяции отражает ее эффективность по отношению к окружающей среде, роль которой играет вторая популяция. Затем каждая особь из второй популяции тестируется с каждой особью из первой популяции (или из подмножества особей первой популяции).

Необходимо отметить, что измерение относительной приспособленности особи в процессе совместной эволюции не является измерением абсолютной приспособленности особи по сравнению с идеальным противником. Это просто относительное измерение, определяющее на сколько хорошо особь приспособилась к текущей популяции особей противников. Допустим, существует популяция боксеров, которые умеют наносить удары только левой рукой, и существует особь возможности защиты которой позволяют ей хорошо защищаться, но только от ударов левой рукой. Относительная приспособленность такой особи будет очень высока. Однако абсолютная приспособленность будет низкой, когда тестирование будет проводиться с участием противника, который умеет наносить удары как левой, так и правой рукой (идеальные противник).

Даже на начальном этапе, когда особи обоих популяций плохо приспособлены (и абсолютно, и относительно), некоторые особи будут иметь хотя бы немного лучшую относительную приспособленность, чем другие. Это означает, что некоторые особи в каждой популяции более эффективны при взаимодействии с особями из популяции противников по сравнению с большинством особей из собственной популяции.

Операция «воспроизведение потомства пропорционально приспособленности» (основана на принципе Дарвина, суть которой в том, что выживают и оставляют потомство наиболее

приспособленные особи) применяется ко всем популяциям. В качестве значения оценочной функции используется текущая относительная приспособленность особи. После этого оператор скрещивания может применяться к такой паре родительских особей, в которой, по крайней мере, одна особь выбрана на основе ее относительной приспособленности.

В течение всего периода времени обе популяции стремятся совместно эволюционировать и достичь высшего уровня эффективности, определяющегося абсолютной приспособленностью. Обе популяции эволюционируют, не ставя перед собой цель, достичь абсолютной приспособленности к некой внешней окружающей среде. В некоторых случаях обе популяции особей могут эволюционировать до того уровня эффективности, который равен максимальной возможной абсолютной эффективности. Таким образом, алгоритм иерархической совместной эволюции представляет собой самоорганизующийся обоюдно-развивающий процесс, движущей силой которого является относительная, а не абсолютная эффективность особей.

1.1.5.4. Адаптивные инкрементальные оценочные функции

Предложенный в работе [64] подход является результатом объединения идей из работ [63, 75]. Использование адаптивной инкрементальной оценочной функции основано на том факте, что популяция организмов эволюционирует поэтапно. На каждом этапе эволюции к оценочной функции добавляется новый компонент. Перед началом каждого этапа эвристическим методом определяется эффективность применения для каждого неиспользуемого компонента оценочной функции. Наиболее эффективный компонент добавляется в текущую оценочную функцию и используется на данном этапе эволюции. В результате эволюция сначала идет с использованием наиболее эффективных компонентов оценочной функции, оставляя менее эффективные компоненты на более поздние этапы работы генетического алгоритма.

Для применения адаптивной инкрементальной оценочной функции необходимо переопределить оценочную функцию, по сравнению с тем определением, которое приведено в работе [75]:

- пусть оценочная функция F имеет следующий вид $F = \sum_k F_i$, где $\{F_i\}$ – множество оценочных функций, связанных соответственно с множеством свойств $\{P_i\}$, которые совместно определяют задачу;
- пусть каждая оценочная функция F_i нормализована так, что особь c будет решением, тогда и только тогда, если $F_i(c) = 1$ для всех i;
- введем коэффициенты $\{\alpha_i\}$, где $\alpha_i > 0$ для любого i и $\sum_k \alpha_i = 1$;
- определим $F' = \sum_k \alpha_i F_i$.

В дополнение для упрощения описания инкрементальной природы предложенного процесса введем следующие множества и переопределим F и F' через них:

- Φ – множество, состоящее из всех оценочных функций, пронумерованных с помощью индекса i;
- ϕ – множество индексов оценочных функций, использующихся на текущем этапе эволюционного процесса (оно в начале $\varnothing$);
- $F(c) = \sum_{i \in \Phi} F_i$;
- $F'(c) = \sum_{i \in \phi} \alpha_i F_i$.

Приведем описание генетического алгоритма, в котором применяется адаптивная инкрементальная оценочная функция:

1. Инициализируется популяция особей p.

2. Для всех особей $c \in p$ вычисляется $F(c)$.
3. Выбирается лучшая особь $hero$ таким образом, что $F(hero) = \max_{c \in p} F(c)$.
4. $\phi \Leftarrow \varnothing$.
5. До тех пор, пока $F(hero) < 1$, выполняется следующие действия:
   - Для всех $i \in \Phi$ вычисляется $\overline{F_i}(p)$ (здесь и далее под значением $\overline{F_i}(p)$ понимается среднее значение, которое принимает оценочная функция на текущей популяции особей p).
   - Если $\phi \neq \Phi$, то выбираем j такое, что $\overline{F_j}(p) = \min_{i \in \Phi - \phi} \overline{F_i}(p)$, и добавляем j в ϕ ($\phi \Leftarrow \phi \cup j$).
   - Для того чтобы настроить множество коэффициентов $\{\alpha_i\}$ согласно эффективности компонент оценочных функций, для каждого $i \in \phi$ определяем следующим образом:

$$\alpha_i \Leftarrow \frac{\frac{1}{\overline{F_i}(p)}}{\sum_{i \in \phi} \frac{1}{\overline{F_i}(p)}}.$$

   - Для всех особе $c \in p$ вычисляется $F'(c)$.
   - Выбирается лучшая особь $subhero$ таким образом, что $F'(subhero) = \max_{c \in p} F'(c)$.
   - До тех пор пока $F'(subhero) < 1$, популяцию p эволюционирует, используя в качестве оценочной функции F'.
   - Если $F(subhero) > F(hero)$, то $hero \Leftarrow subhero$.

В качестве эвристической оценки значимости компонента оценочной функции используется величина обратная среднему значению компонента оценочной функции на текущей популяции решений $\frac{1}{\overline{F_i}(p)}$. Другими словами, чем меньше среднее значение компонента оценочной функции, тем хуже особи в популяции ей удовлетворяют. На каждом этапе эволюции производится перенастройка множество коэффициентов $\{\alpha_i\}$, так чтобы увеличить вклад наиболее значимых компонентов оценочных функций.

К сожалению, подход, основанный на адаптивных инкрементальных оценочных функциях, может быть применен только к узкому классу задач.

1.1.6. Методы выбора функций приспособленности

1.1.6.1. Выбор функции приспособленности для задачи о флибах

В книге [50] рассматривается подход к проблеме автоматического построения конечных автоматов с помощью эволюционных методов. Первая задача, рассматриваемая в работе [50] – задача предсказания. На вход конечного автомата (флиба [10]) подается символ, соответствующий текущему состоянию окружающей среды. Задача флиба предсказать состояние среды в следующий момент времени. Для того, что бы определить, на сколько корректно автомат предсказывает изменения среды, ему на вход подается последовательность входных символов, соответствующих состояниям окружающей среды. Каждый выходной символ, генерируемый флибом, сравнивается со следующим входным символом, поступающим из среды. Процент правильных предсказаний есть мера способности автомата предсказывать поведение среды на основе предшествующих символов.

Рассмотрим, как работает оценочная функция, на примере автомата M_0, диаграмма переходов для которого изображена на рис. 2.

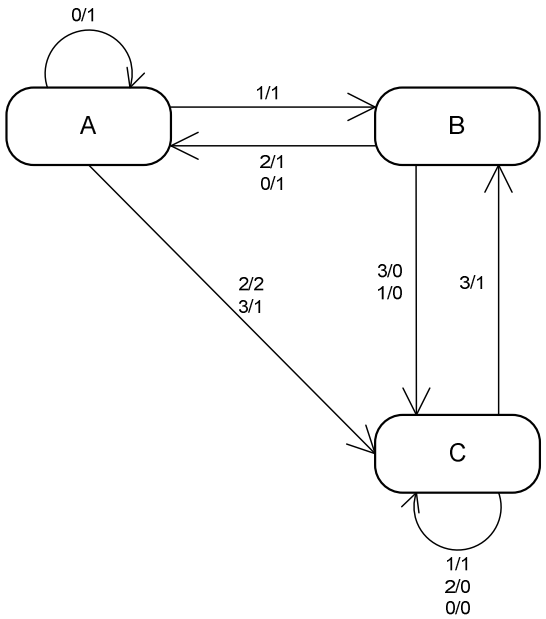


Рис. 2. Диаграмма переходов автомата

Пример обработки автоматом последовательности символов приведен в табл. 1.

Таблица 1. Результат работы автомата

Текущее состояние	B	A	C	C	C	C	B	C	C	B	A
Входной символ	2	2	1	0	1	3	3	0	3	0	1
Выходной символ		1	2	1	0	1	1	0	0	1	1
Стоимость ошибки		1	1	1	1	1	1	0	1	1	0

В первой строке этой таблицы показана последовательность текущих состояний флиба в процессе его реагирования на последовательность входных символов, представленную во второй строке таблицы. Состояние *B* было произвольно выбрано в качестве начального состояния. В третьей строке приведена последовательность символов, полученных в результате работы флиба. Необходимо отметить, что эти символы сдвинуты на одну позицию вправо для облегчения сравнения. Значения выходных символов в этой строке таблицы соответствуют предсказанным состояниям среды.

Последняя строка образована путем сопоставления фактического и предсказанного символов при учете «матрицы стоимостей» [50] (или «матрицы штрафов») $A = \{a_{ij}\}$. В данном примере $a_{ij} = 0$, если $i = j$, и $a_{ij} = 1$, если $i \neq j$. Стоимость каждого преобразования находится по матрице стоимостей, где индекс i относится к предсказанному символу (значению выходной переменной, генерируемой флибом), а j – к очередному символу входной последовательности. Оценочная функция F_1 (мера предсказательной способности машины) определяется как отношение суммы стоимостей всех ошибок к числу символов входной последовательности. Так средняя стоимость ошибок автомата M_0 находится по формуле $F_1 = \dfrac{Сумма\_стоимостей\_ошибок}{Длина\_входной\_последовательности} = \dfrac{8}{10} = 0.8$.

Как уже было отмечено выше, оценочная функция строится на основе матрицы стоимостей, приписанных каждому возможному правильному и ошибочному предсказанию следующего состояния среды. Например, если целью является минимизация величины ошибки предсказания каждого следующего символа, элементы матрицы стоимостей будут принимать значения $a_{ij} = |i - j|$. Если требуется минимизировать среднеквадратичную ошибку предсказания, элементы матрицы стоимостей будут иметь вид $a_{ij} = (i - j)^2$. Если требуется «точное попадание», то все диагональные элементы в матрице ошибок будут равны нулю, а не диагональные элементы будут одинаковы. Такой же подход используется и в рассматриваемом выше примере. Если правильное предсказание некоторого символа имеет большую значимость, то это может быть выражено в матрице стоимостей, определяющей цель предсказания. Так матрица стоимостей в табл. 2 указывает, что более важно правильное предсказание символа «0», менее важно правильное предсказание символа «1» и, наконец, еще менее важно правильное предсказание символа «2». В то же время из нее следует, что предпочтительнее ошибиться, приняв нуль за единицу, чем наоборот. Такой способ задания цели отражает относительный вес каждого возможного исхода. В рассмотренном примере не взымается дополнительный штраф за столь сложную цель. По мере сравнения символов на выход автомата с очередными входными символами, фактически появившимися в окружающей среде флиба, по матрице цели определяется стоимость соответствующей ошибки. Сумма соответствующих стоимостей характеризует ценность автомат с точки зрения его способности предсказывать только что предъявленную последовательность символов.

Таблица 2. Пример матрицы стоимостей

		Предсказанные символы		
		0	1	2
Фактические символы	0	0	3	4
	1	4	1	4
	2	4	4	2

В общем случае последовательность возникших в процессе эволюции автоматов будет зависеть от цели, заданной с помощью оценочной функции. Нет необходимости ограничивать эволюцию одной лишь неизменной целью. Если цель будет изменяться по мере протекания процесса в реальном времени, то новые поколения автоматов будут постепенно отражать эти изменения. Конечно, все сказанное выше об оценочных функциях, построенных на основе матрицы штрафов, в равной степени применимо и к матрице платежей.

Оценочная функция может быть еще более гибкой. Например, если требуется предсказывать каждый второй символ, то для этого достаточно сравнивать значения выходной переменной флиба с символами среды со сдвигом на два такта. Небольшое изменение оценочной функции позволяет использовать этот подход для предсказания многомерных сред. Символы, описывающие изменяемые параметры среды, могут быть скомбинированы в новый алфавит символов, описывающий среду однозначным образом. Результаты предсказаний получаются в том же алфавите, так что они могут быть интерпретированы в виде индивидуальных предсказаний по каждому символу в отдельности.

До сих пор предполагалось, что каждый автомат следует оценивать по всей его доступной предыстории. Очевидно, что такая стратегия не всегда является наилучшей. Например, если среда изменяет свои статистические свойства, может оказаться более целесообразным ограничиться использованием некоторых последовательностей, для того чтобы повысить вероятность нахождения наилучшей логики для настоящего времени. В случае полного отсутствия информации о характере

среды, выбор конкретной длины последовательности может оказаться весьма затруднительным. Можно использовать эвристическую процедуру, которая будет последовательно изменять длину последовательности, учитываемую оценочной функцией. В каждый момент времени две (или более) популяции автоматов эволюционируют на различной длине предыстории. Затем их показатели сравниваются. Это дает возможность определить, какие следующие длины предысторий подлежат рассмотрению. Полную длину последовательности следует запоминать как можно дольше, для того чтобы длина предыстории при желании могла быть расширена. Трудность этой задачи возрастает вместе с ограничением, налагаемым на объем памяти автомата вплоть до того момента, когда единственной альтернативой становится метод скользящего интервала – сохранение фиксированной длины предыстории.

В книги [50] описывается эксперимент, связанный с влиянием длины начального запоминания на качество предсказания. На протяжении начальной опытной последовательности поведение предсказывающей системы почти случайное. Ошибки, допущенные в этой последовательности, неизбежно уменьшают величину показателя качества. Этот эффект может быть устранен, если первое предсказание делается после испытания достаточно большого числа символов. На рис. 3 приведены сравнительные характеристики предсказания для автоматов с начальным запоминанием двух и 20-и символов.

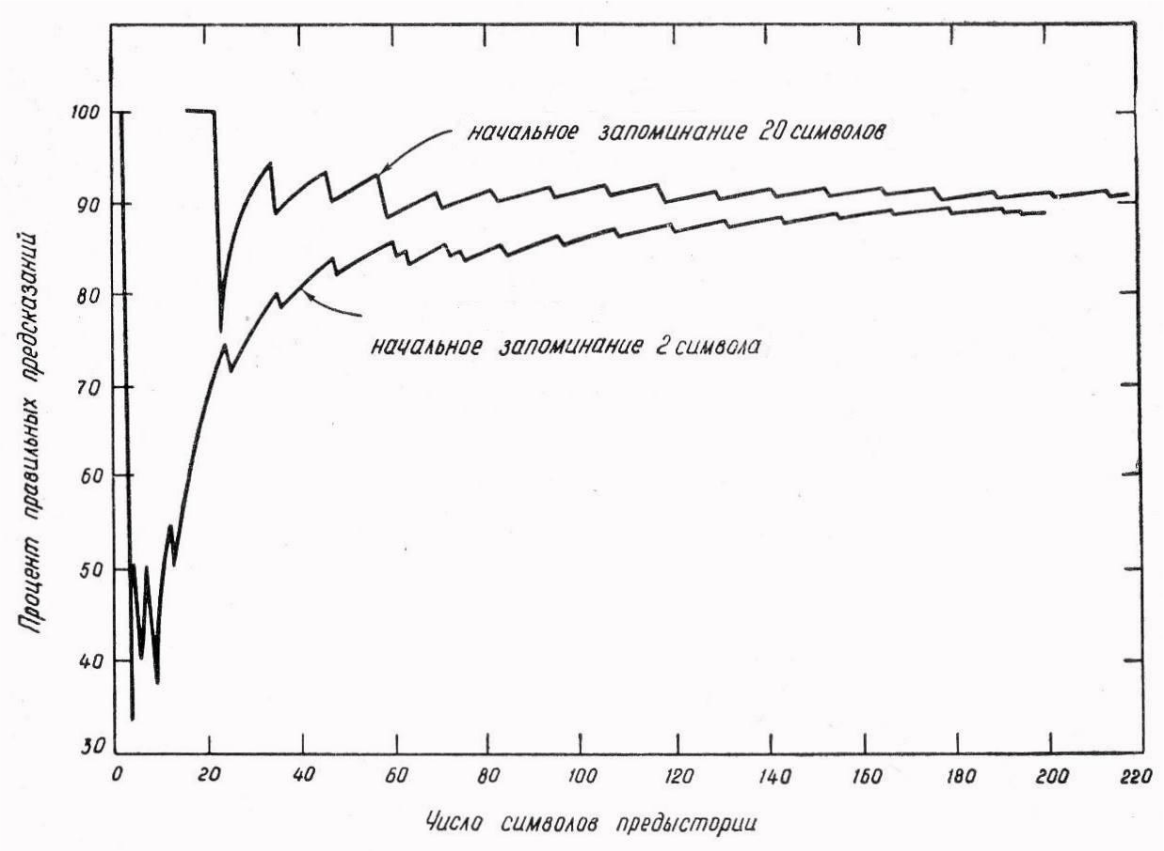


Рис. 3. Сравнение результатов предсказаний при различной длине предыстории

Эксперимент проводились при следующих условиях: циклическая среда вида 101110011101; три автомата на поколение; однократная мутация; случайный ряд единиц; пять поколений на предсказание.

В работе [50] показано как строить различные оценочные функция для задачи предсказания изменений характеристик окружающей среды. Оценочная функция изменятся с учетом целей,

которые ставит перед собой исследователь. Применение специальных модификаций оценочных функций позволяет ускорить построение искомого автомата.

1.1.6.2. Использование совместной эволюции для построения минимаксных стратегий для пары игроков

В работе [71] совместная эволюция применяется для построения минимаксных стратегий для пары игроков, играющих в игру на двоих, правила которой заданы с помощью дерева игры, изображенного на рис. 4.

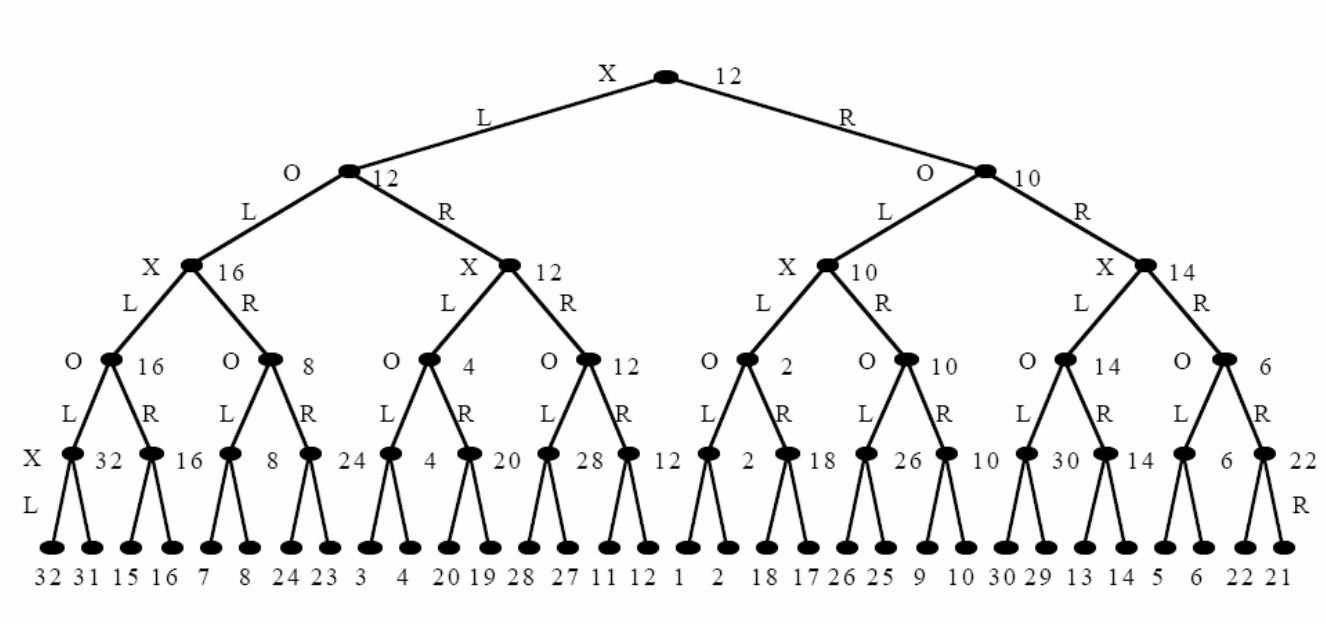


Рис. 4. Дерево игры с выплатами

В случае алгоритма совместной эволюции нет идеального противника, на котором можно было бы тренировать популяцию особей. Вместо него используются две совместно эволюционирующих популяции. Перед началом эволюции обе популяции заполняются особями, сгенерированными с помощью случайных композиций элементов из набора допустимых функций и терминалов.

Рассмотрим простую дискретную игру, дерево игры для которой изображено на рис. 4. Рядом с каждым узлом дерева нанесена метка, соответствующая игроку, который должен сделать ход. Каждая ветвь дерева помечена вариантом выбора (`L` или `R`), который может сделать игрок. Каждому листу дерева присвоена некоторая сумма вознаграждения (для игрока *X*).

Это игра соревнование для двоих, в которой игроки при каждом ходе принимают одно из двух возможных альтернативных решений. При каждом ходе игрок выбирает куда идти: `L` – налево или `R` – направо. После того как игрок *X* делает три хода и игрок *O* делает два хода, игрок *X* получает, а игрок *O* отдает, вознаграждение, соответствующее текущему листу дерева игры.

Каждый игрок имеет доступ к полной информации о предыдущих ходах его противника и о своих собственных ходах. История игры содержится в пяти переменных: `XM1` – первый ход игрока *X*, `OM1` – первый ход игрока *O*, `XM2` – второй ход игрока *X*, `OM2` – второй ход игрока *O* и `XM3` – третий ход игрока X. Каждая из этих пяти переменных может принимать одно из трех возможных значений: `L` – налево, `R` – направо или `U` – не определенно. Значения переменной перестает быть неопределенным, после того как один из игроков делает ход, связанный с ней. В начале значения всех пяти переменных не определены. Список переменных, значения которых не определены, задает состояние, в котором находится на текущий момент процесс игры. Например, если оба игрока

сделали по одному ходу, переменные XM1 и OM1 определены (принимают значение или L или R), но значения трех оставшихся переменных (XM2, OM2 и XM3) не определены (принимают значение U).

Стратегия для конкретного игрока определяет, какой выбор он сделает в каждой из возможных для него ситуаций. В частности, стратегия для игрока *X* должна определять его первый ход, если он находится в начале игры. Стратегия для игрока *X* также должна определять его второй ход, если игрок *O* сделал один ход. Она должна определять его третий ход, если игрок *O* уже сделал два хода. Так как игрок *X* ходит первым, его первый ход не зависит от предыдущих ходов. Однако второй ход игрока *X* будет зависеть от первого хода игрока *O* (от переменной OM1) и от выбора который игрок *X* сделал при своем первом ходе (от переменной XM1). Таким же образом третий ход игрока *X* будет зависеть от первых двух ходов игрока *O* и от его собственных двух первых ходов. Аналогично сказанному выше, стратегия для игрока *O* должна определять выбор для игрока *O* во всех возможных ситуациях, в которые он может попасть. Стратегии в работе [71] заданы в виде компьютерных программ (*S*-выражений на языке *LISP*), на вход которых подаются значения переменных, содержащих историю игры. Значение выходной переменной, генерируемое программой, определяющей стратегию игрока, соответствует выбору (L или R), который игрок делает при соответствующем ходе. Таким образом, множество терминалов состоит из следующих элементов T = {L, R}.

Четыре проверочные функции CXM1, COM1, CXM2 и CMO2 позволяют делать выбор в зависимости от того, какие значения принимают переменные, содержащие историю игры. Каждая из этих функций – определенный вариант *CASE*-функции языка *LISP*. Например, функция CMX1 принимает на вход три аргумента. Первый аргумент вычисляется, если переменная XM1 принимает значение U (не определено). Второй аргумент вычисляется, если переменная XM1 принимает значение L (налево) и третий аргумент, если переменная XM1 принимает значение R (направо). Функции COM1, CXM2 и CMO2 построены аналогично. Таким образом, множество функций состоит из следующих элементов F = {CXM1, COM1, CXM2, CMO2}. Каждая из этих функций принимает на вход три аргумента.

Задача состоит в построении стратегий для обоих игроков этой игры с помощью генетического алгоритма, основанного на совместной эволюции.

Оценочная функция, определяющая эффективность стратегии игрока, базируется на среднем вознаграждении, которое получает игрок в результате состязаний со всеми игроками из популяции соперников.

Абсолютная эффективность конкретной стратегии для конкретного игрока определяется, когда он играет против минимаксной стратегии противника. Заметим, что для вычисления абсолютной эффективности стратегии для игрока *X*, необходимо проверить ее для четырех возможных комбинаций ходов игрока *O* (игрок *O* делает выбор L или R на первом и втором ходах). Для вычисления абсолютной эффективности стратегии для игрока *O*, необходимо проверить ее для восьми возможных комбинаций ходов игрока *X* (этот игрок делает выбор L или R на первом, втором и третьем ходах). Необходимо отметить, что проверка четырех или восьми комбинаций ходов противников не делается при вычислении относительной эффективности стратегии. Для двух минимаксных стратегии противники играют друг против друга. При этом размер вознаграждения равен 12. Минимаксные стратегии имеют преимущество перед не минимаксными стратегиями других игроков.

Как упоминалось выше, в алгоритме совместной эволюции никак не применяется минимаксная стратегия противника. Алгоритм совместной эволюции использует только относительную эффективность стратегий.

Эксперимент проводился при 300 особях в популяции. Программа, соответствующая лучшей стратегии для игрока *X* в первом поколении, имела следующий вид:

```
(COM1 L
      (COM2 (CXM1 (CXM2   R
                          (CXM2 R R R)
                          (CXM2 R L R))
                  L
                  (CXM2 L R (COM2 R R R)))
      (COM1 R
            (COM2 (CXM2 L R L) (COM2 R L L) R)
            (COM2 (COM1 R R L)
                  (CXM1 R L R)
                  (CXM1 R L L)))
            (CXM1 (COM2 (CXM1 R L L) (CXM2 R R L) R)
                  R
                  (COM2 L R (CXM1 L L L))))
      R).
```

После упрощения эта программа принимает вид

```
(COM1 L (COM2 L L R) R).
```

Значение оценочной функции для этой особи равно 10.08.

Программа, соответствующая лучшей стратегии для игрока *O*, так же была сложной. После упрощения она приняла следующей вид

```
(CXM2 R (CXM1 # L R) (CXM1 # R L)).
```

Следует отметить, что при упрощении программы был использован символ # для обозначения тех ситуаций, которые никогда не могли возникнуть. Относительная эффективность этой стратегии 7.57.

Ни лучшая стратегия для игрока *X*, ни лучшая стратегия для игрока *O* из первого поколения не достигли максимальной эффективности.

Необходимо отметить, что значения оценочной функции для лучшей стратегии для игрока *X* и для лучшей стратегии для игрока *O* из первого поколения (10.08 и 7.57, соответственно) вычислялись, как усредненное вознаграждение, полученное в результате состязаний с трехстами особями из популяции противников.

Во втором поколении значение оценочной функции для лучшей стратегии для игрока *X* стало равным 11.28. Эта стратегия не являлась минимаксной стратегией. Максимальное значение абсолютной эффективности не было достигнуто.

Во втором поколении значение оценочной функции для лучшей стратегии для игрока *O* стало равным 7.18. Программа соответствующая этой стратегии представлена ниже:

```
(CXM2 (CXM1 R R L) (CXM2 L L (CXM2 R L R)) R).
```

Эта лучшая стратегия для игрока *O* во втором поколении и есть минимаксная стратегия для игрока *O*. Если бы эта стратегия состязалась с минимаксной стратегией для игрока X, ее вознаграждение было бы равным 12.

Лучшую стратегию для игрока *O* можно упростить до вида

```
(CXM2 (CXM1 # R L) L R).
```

В поколениях между вторым и пятнадцатым число особей в *O*-популяции, достигших абсолютной эффективности было 2, 7, 17, 28, 35, 40, 50, 64, 73, 83, 93, 98 и 107, соответственно. Таким образом, программы, реализующие минимаксную стратегию для игрока *O*, начали доминировать в популяции.

В пятнадцатом поколении значение оценочной функции для лучшей стратегии для игрока *X* стало равным 18.11. Программа, соответствующая этой стратегии, приведена ниже:

```
(COM2 (COM1 L L (CXM1 R R R))
      L
      (CXM1 (COM1 L L (CXM1 R R R))
            (CXM2 L R R)
            R)).
```

Эта лучшая стратегия для игрока *X* в тринадцатом поколении и есть минимаксная стратегия для игрока *X*. Если бы эта стратегия состязалась с минимаксной стратегией для игрока *O*, то ее вознаграждение было бы равным 12.

Лучшую стратегию для игрока *X* можно упростить до вида

```
(COM2 (COM1 L L R) L R).
```

В поколениях между 16 и 31, число особей в *X* популяции, достигших абсолютной эффективности, было 3, 4, 8, 11, 10, 9, 13, 21, 24, 29, 43, 32, 52, 48 и 50, соответственно. Таким образом, программы, реализующие минимаксную стратегию для игрока *X*, начали доминировать в популяции. Тем временем программы, реализующие минимаксную стратегию для игрока *O*, начали еще сильнее доминировать в *O*-популяции.

В 39-ом поколении число особей в *O*-популяции, достигших абсолютной эффективности, стало равным 188 (почти две трети популяции), а особей в *O*-популяции, достигших абсолютной эффективности, стало равным 74 (примерно четверть). В 39-ом поколении минимаксные стратегии в поколения *X* стратегий для обоих игроков стали доминирующими.

Интересно, что значение оценочной функции для 74 стратегий для игрока *X* было равным 19.11, а для 188 стратегий для игрока *O* – 10.47. Ни одно из этих значений не равно 12, так как популяции не полностью заполнены минимаксными стратегиями.

Необходимо отметить, что генетический алгоритм, основанный на совместной эволюции, смог построить минимаксные стратегии для обоих игроков, несмотря на то, что сначала у него не было информации о минимаксной стратегии ни для одного из противников.

1.1.6.3. Использование инкрементальной функции приспособленности для генерации конечных автоматов, распознающих регулярные выражения

Рассмотрим пример использования инкрементальных оценочных функция для построения конечных автоматов, распознающих регулярные выражения [63].

Конечный автомат читает входную строку из регулярного выражения и принимает или отвергает ее. Регулярные выражения создаются по следующим правилам:

1. Конечный алфавит.
2. Объединение, пересечение или конкатенации двух регулярных выражение – регулярное выражение.
3. Дополнение регулярного выражения также является регулярным выражением.
4. Повторяемость регулярных выражений, также называемая замыканием Клини.
5. Пустая строка – регулярное выражение.

Благодаря конечному алфавиту претендентов на решения легко проверить. Так как результатом пересечения, объединения или конкатенации двух регулярных выражений является регулярное выражение, то выведение новых регулярных выражений не вызывает сложностей.

Стандартный конечный автомат, распознающий регулярные выражения, имеет только два возможных значения выходной переменной: «принять» и «не принять». Поэтому возникают сложности при сравнении эффективности двух конечных автоматов. Можно только определить корректно работает конечный автомат или нет, но невозможно понять, на сколько близко его поведение к желаемому. Для решения этой проблемы в работе [63] предлагается два способа решения этой проблемы. Первый из них состоит в увеличении числа моментов времени, в которых происходит считывания значения выходной переменой, генерируемой эволюционирующим конечным автоматом. Выходная переменная проверяется после чтения автоматом каждого входного символа. Предполагается, что каждый входной символ может быть последним символом строки. Второй способ заключается в разбиения значения выходной переменной «не принять» на два отдельных значения: «отклонить» и «не известно». Выходная переменная может принимать значение «не известно», пока процесс разбора не завершится. Если входной символ приводит к ошибке в разборе, то выходная переменная принимает значение «отклонить». Устройство остается в состоянии отклонить, так как поступающие на вход символы не могут изменить того факта, что строка не удовлетворяет регулярному выражению.

Пример разбора строки помощью изложенного метода показан на рис. 5.

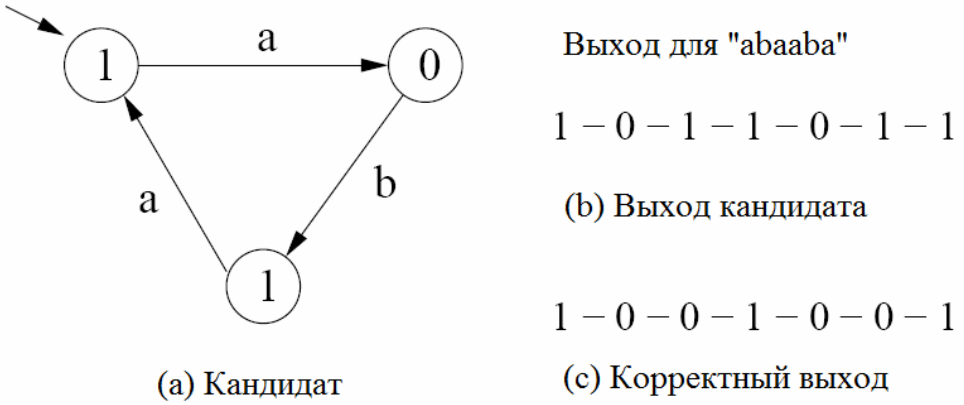


Рис. 5. Выход конечного автомата при разборе строки “abaaba”

На рис. 5 изображен один из кандидатов на распознавания строк, соответствующих регулярному выражению “`(aba)*`”. Все значения выходной переменной, равные «отклонить», удалены для упрощения примера. Заметим, что кандидат выдал корректное значение выходной переменной ‘1’ («принять») в первом состоянии (пустая строка удовлетворяет рассматриваемому в примере регулярному выражению). На вход кандидату подавалась допустимая строка “`abaaba`”. На рис. 5 приведен список выходных значений, выданных кандидатом, и корректный список значений, для автомата, умеющего корректно распознавать строки, соответствующие регулярному выражению.

Предложенный в работе [63] алгоритм использует пять независимых операторов мутации: добавить состояние, удалить стояние, изменить начальное состояние, изменить переход между состояниями, изменить значение выходной переменной. Оператор скрещивания, используемый в работе [63], создает новый конечный автомат, комбинируя части, вырезанные из двух родительских автоматов. Благодаря оператору скрещивания новый автомат наследует часть характеристик от каждого из своих родителей.

В работе [63] описывается ряд экспериментов, проводимых для сравнения эффективности инкрементального генетического алгоритма с обычным генетическим алгоритмом. Для каждого

набора начальных данных производится серия испытаний. В большинстве испытаний удается построить автомат, удовлетворяющий условию задачи. Случай “aab(ba)*bba(abjba)bb” слишком сложен для обычного генетического алгоритма, в то время как итеративный алгоритм справляется с поиском правильного решения. Размер популяции во все экспериментах равен 300 особям. При формировании нового поколения треть особей выбирается из текущего поколения без изменений, треть особей формируется с помощью оператора скрещивания, а еще треть формируется с помощью оператора мутации. Проверка эффективности генетического алгоритма требует большого числа тестов. В связи с этим число тестов выбрано равным 500. Максимальное число поколений ограничено десятью тысячами. После того как алгоритм находит корректное решение, эволюция продолжается в течение еще 50 поколений, для того чтобы уменьшить размер полученного конечного автомат.

Первая группа экспериментов связана с построением автоматов для регулярных выражений, заданных строками различной длины. В целях сравнения далее приводятся усредненные результаты из всех тестовых групп. В первом случае требовалось построить автомат для регулярного выражения заданного строкой “abba”. Эксперименты проводились с различной длиной приращения символов, добавляемых на каждом этапе работы генетического алгоритма. Использовались длина приращения символов – ноль (обычный генетический алгоритм), длина приращения символов – единица (“a”, “ab”, “abb”, “abba”) и длина приращения символов – два (“ab”, “abba”). Результаты этих экспериментов приведены на рис. 6. На графике по вертикальной оси отложено среднее число поколений, необходимых для построения искомого автомата в каждом случае. Каждый прямоугольник на графике соответствует группе экспериментов с фиксированной длиной приращения символов. Прямоугольник “0” соответствует стандартному генетическому алгоритму. Остальные прямоугольники соответствуют инкрементальным генетическим алгоритмам. Использование инкрементальной оценочной функции уменьшило число поколений, требующихся для построения автомата, на 16%.

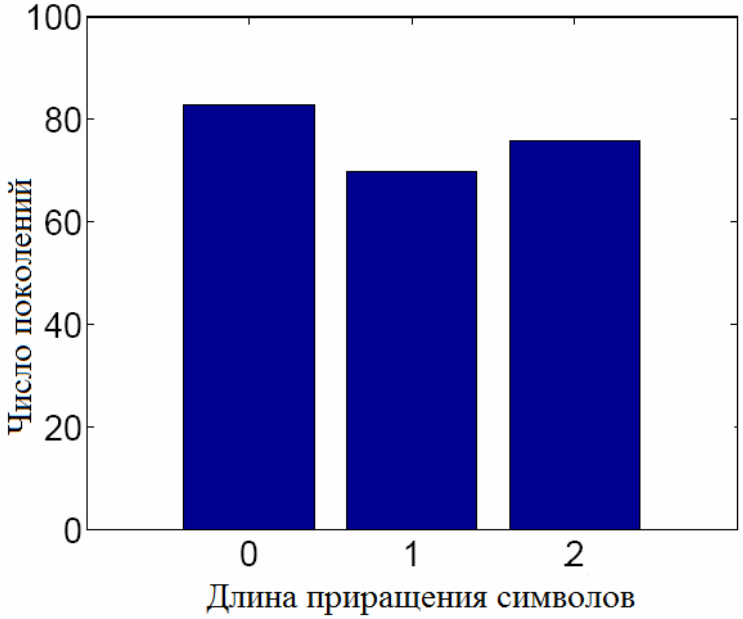


Рис. 6. Построение автомата для строки из четырех символов

Эффект от использования инкрементальных оценочных функций особенно хорошо заметен при построении автоматов для более длинных строк. На рис. 7, 8 приведены результаты экспериментов для строк длиной 6 и 12 символов соответственно. Использование инкрементальной

оценочной функции с длиной приращения символов, равной трем, в эксперименте по построению автомата, соответствующего регулярному выражению, заданному строкой из шести символов, позволило уменьшить число поколений по сравнению со стандартным генетическим алгоритмом на 80%.

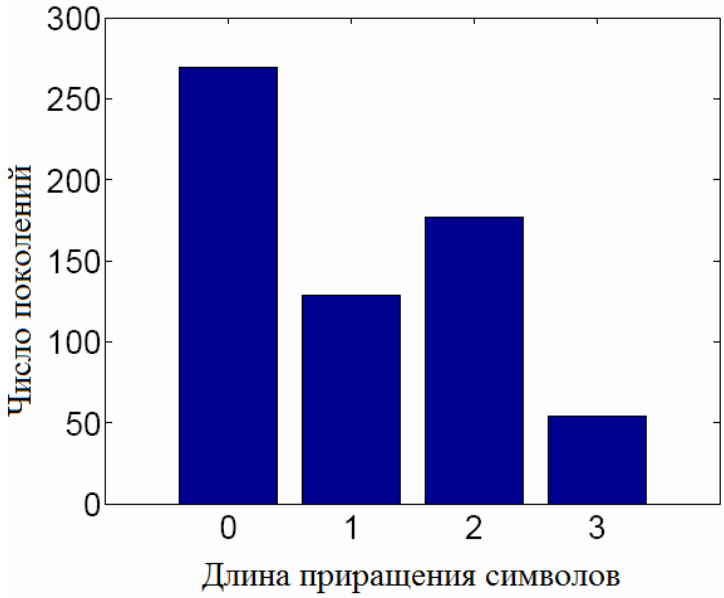


Рис. 7. Построение автомата для строки из шести символов

Использование инкрементального генетического алгоритма с длиной приращения символов, равной единице, в эксперименте по построению автомата, соответствующего регулярному выражению, заданному строкой из двенадцати символов, позволило сократить число поколений на 50%. Увеличение скорости сходимости при длине приращения символов, равной трем, можно объяснить при рассмотрении строк, с помощью которых задавались регулярные выражения. В этих экспериментах строились строки для регулярных выражений, заданных строками “abbabb” и “abbabbbababb”. Можно заметить, что символы “abb” повторяются в этих строках. Как только автомат для регулярного выражения “abb” построен, он может использоваться как основа для дальнейшей эволюции. При наличии в поколении автомата для регулярного выражения заданного строкой “abb”, генетический алгоритм быстро строит автомат для регулярного выражения, заданного строкой “abbabb”.

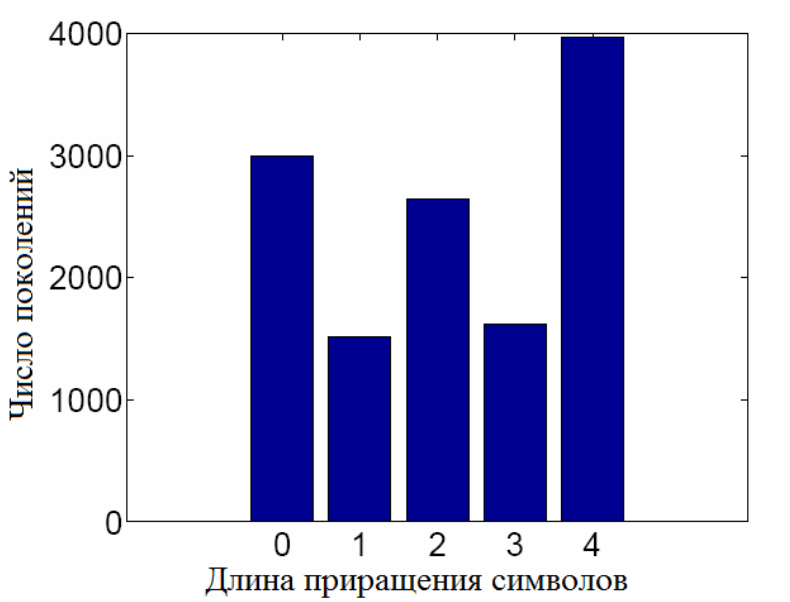


Рис. 8. Построение автомата для строки из двенадцати символов

В работе [63] также описываются две серии экспериментов для регулярных выражений, заданных с помощью объединения строк, состоящих из трех символов. Первая серия экспериментов проводилась для регулярного выражения, заданного через объединение шести строк “aaa”, “aab”,…, “bab”. Вторая серия экспериментов проводилась над объединением строк из первой серии экспериментов, к которому была добавлена строка “bba”. На рис. 9 показаны результаты первой серии экспериментов. Отметим, что инкрементальный генетический алгоритм смог построить автомат при числе добавленных строк, равном двум и трем. Инкрементальный эволюционный подход плохо работает при добавлении строк в объединение. Например, при построении автомата, принимающего только строки вида “aaa”, на него накладывается ограничение отклонять строки вида “aab”. После перехода к этапу, на котором требуются построить автомат для регулярного выражения “aaa|aab”, из уже найденного решения необходимо удалить часть, отвечающую за отклонение строки вида “aab” и добавить новую функциональность.

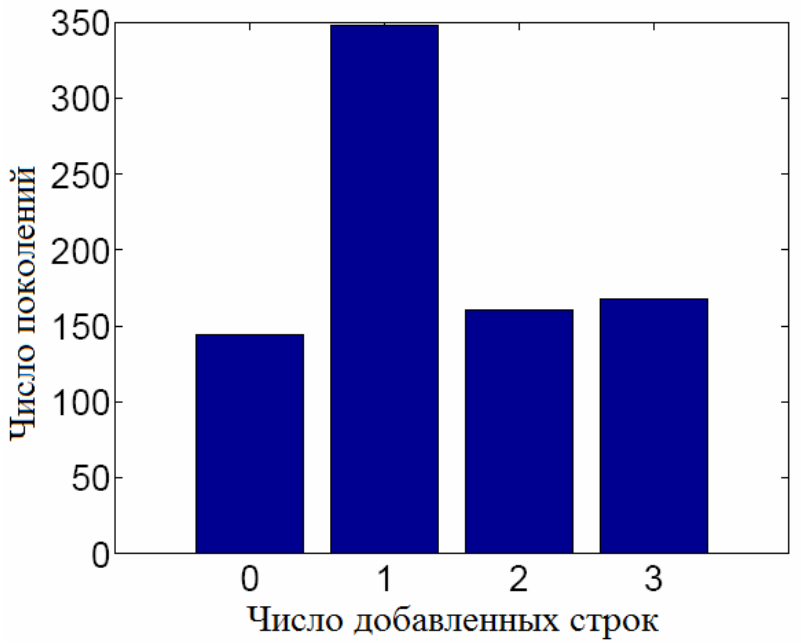


Рис. 9. Построение автомата для объединения из шести строк

Для получения автомата, который решает задачу во второй группе экспериментов, потребовалось значительно больше поколений. Отклонение только строк вида “bbb” требует значительно более сложного конечного автомата. Усложнение задачи существенно замедляет работу стандартного генетического алгоритма. На рис. 10 показаны усредненные результаты этой серии экспериментов. При числе добавленных строк на этап, равном двум, число поколений, необходимое для построения корректно работающего по сравнению со стандартным генетическим алгоритмом, уменьшается на 88%.

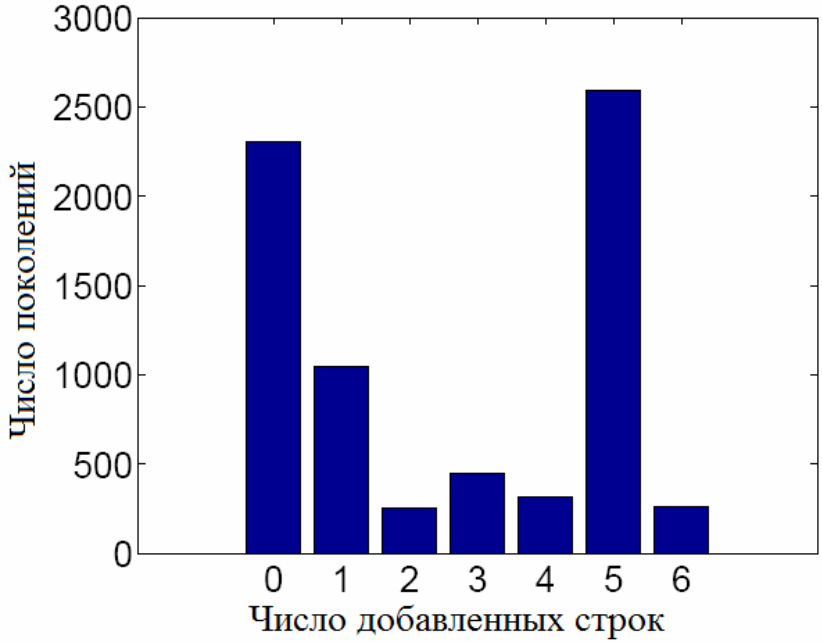


Рис. 10. Построение автомата для объединения из семи строк

Необходимо отметить, что в большинстве экспериментов генетические алгоритмы, в которых использовалась инкрементальные оценочные функции, показали существенно лучшие результаты по сравнению со стандартными алгоритмами.

1.1.7. Методы представления решений

В настоящем разделе рассматриваются методы представления решений. Так как для применения генетических алгоритмов необходимо разработать метод представления решений, поддерживающий как операцию мутации, так и операцию скрещивания, то анализ методов представления решений осуществляется применительно именно к генетическим алгоритмам.

1.1.7.1. Анализ методов

В работах [26, 66] используется следующий подход к кодированию автоматов.

1. Фиксируется число состояний автомата.
2. Логика работы автомата представляется в виде таблицы, в которой для каждого состояния и каждого входного воздействия записывается пара (новое состояние, действие).
3. Полученная таблица записывается в виде строки.

Поясним на примере, как таблица преобразуется в строку. На рис. 11 изображен граф переходов автомата, который может быть представлен в виде табл. 3.

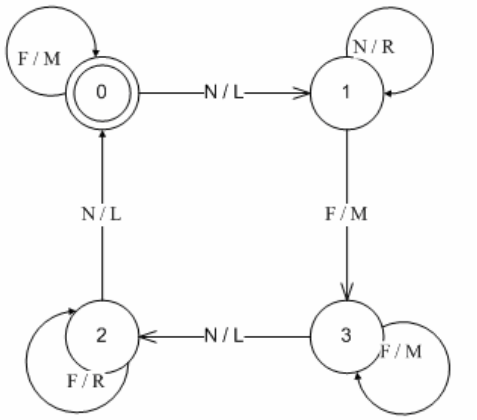


Рис. 11. Пример управляющего автомата

Таблица 3. Таблица переходов управляющего автомата

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

Теперь можно записать элементы этой таблицы в строку, выписав двойки (новое состояние, действие), соответствующие переходам. Для этого переходы упорядочиваются в порядке возрастания

начального состояния, а в случае равенства – по номеру входного воздействия. Далее возможно применение одного из двух подходов: битовые строки и строки над числовым алфавитом.

Кроме того, возможно хранение начального состояния в строке, как это показано в разд. 3.1.4.1. Однако, анализ, проведенный в работе [26], показывает, что такой вариант не дает выигрыша по сравнению с зафиксированным начальным состоянием.

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

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

Известен подход, при котором число управляющих состояний задается перед началом оптимизации и далее не изменяется. При таком подходе число состояний можно задать исходя из априорных представлений о сложности задачи, причем задать с некоторым «запасом»: в процессе оптимизации лишние состояния станут недостижимыми, и их можно будет автоматически исключить. Однако неоправданно большое число состояний негативно влияет на скорость эволюции. В этом смысле более эффективным является постепенное наращивание числа управляющих состояний в процессе оптимизации. Этот вариант также несложно реализуется в рамках предлагаемого подхода к представлению конечных автоматов в виде особей.

Теперь опишем генетические операторы над хромосомами состояний, записанными в виде полных таблиц.

На рис. 12 приведена иллюстрация процесса адаптации к предложенному представлению управляющих автоматов одного способа скрещивания, известного как одноточечное скрещивание. Руководствуясь схожим подходом, можно адаптировать к табличному представлению и другие способы скрещивания.

Алгоритм мутации состояния, представленного таблицей, проиллюстрирован на рис. 13 (на стрелках, обозначающих изменения значений в ячейках таблицы, написаны вероятности этих изменений). При мутации состояния с некоторой вероятностью может мутировать каждый элемент таблицы. При этом номер целевого состояния изменяется на любой из допустимых, а вместо исходного набора действий генерируется новый набор, вероятность появления единиц в котором равна доле единиц в исходном наборе.

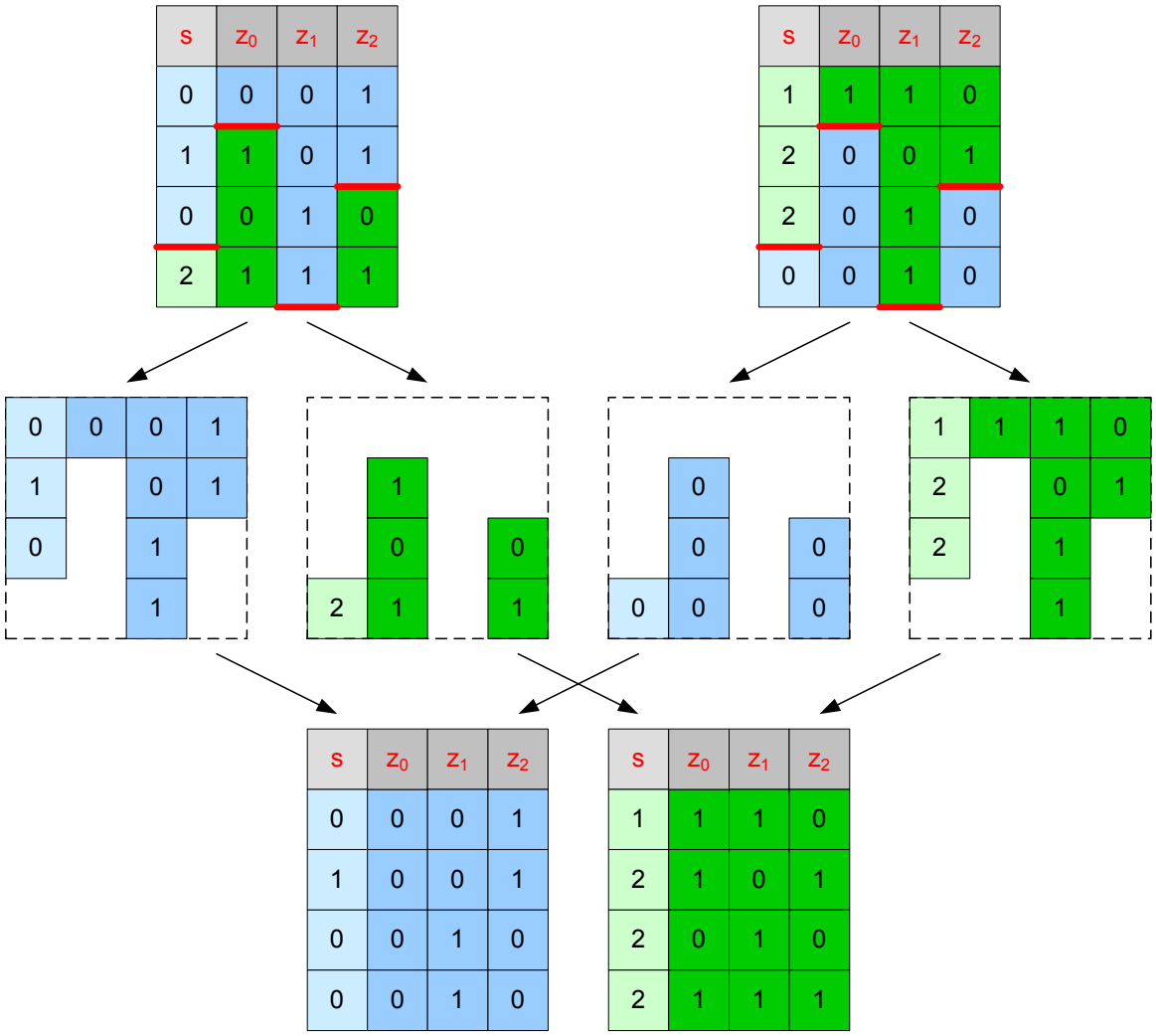


Рис. 12. Пример скрещивания полных таблиц переходов

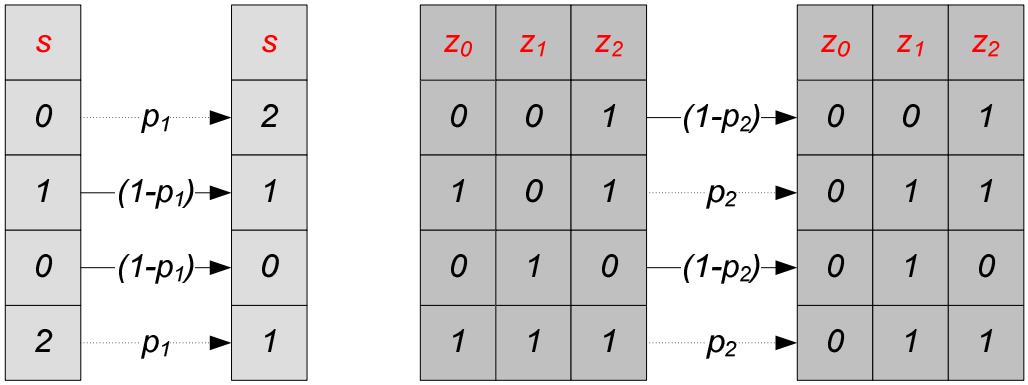


Рис. 13. Пример мутации полных таблиц переходов

Вывод автоматов с помощью генетических алгоритмов может иметь низкую эффективность в силу следующих причин.

- Автомат, соответствующий сгенерированной строке, может иметь недостижимые состояния. Информация, хранящаяся для этих состояний, является генетическим мусором.
- Представление не является естественным – двум разным строкам может соответствовать один и тот же автомат (одному фенотипу может соответствовать два и более генотипов). Это приводит к тому, что изоморфные автоматы соревнуются друг с другом, замедляя работу генетического алгоритма.

Одним из возможных подходов к решению данных проблем может являться применение эвристического оператора переупорядочивания. В качестве такого оператора на полных таблицах переходов может использоваться метод *MTF (Move To Front)* [40]. Он состоит в перенурации вершин автомата для приведения всех изоморфных автоматов к каноническому виду. Для этого из стартовой вершины запускается обход в ширину. После этого вершины нумеруются в порядке обхода. Заметим, что недостижимые вершины автоматически помещаются в конец таблицы, соответствующей автомату. Пример такого преобразования приведен на рис. 14.

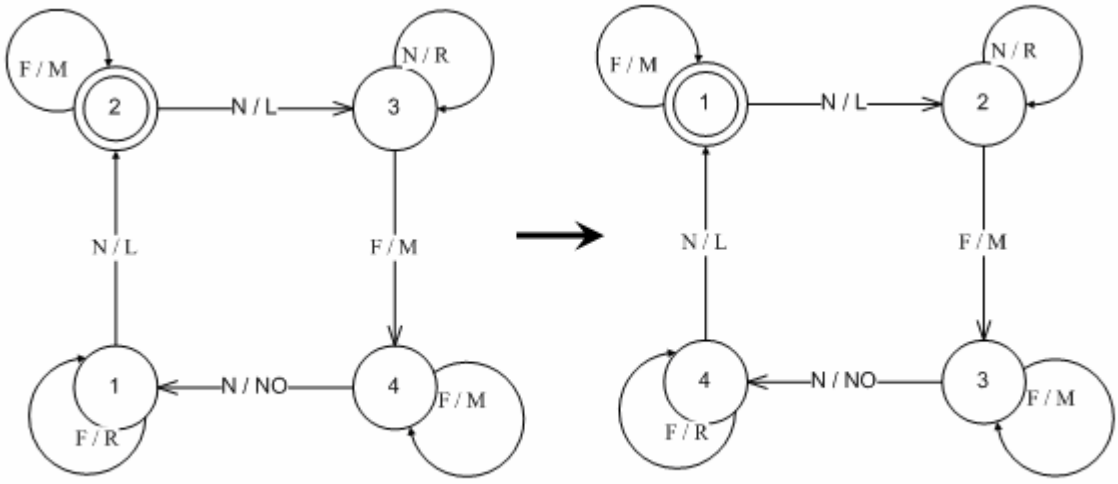


Рис. 14. Пример применения эвристики *MTF*

В данном примере обход графа переходов начинается из вершины, первоначально имеющей номер «2», и заканчивается вершиной с номером «1», пройдя по пути «2 – 3 – 4 – 1». После этого производится перенумерация вершин согласно этому пути.

Анализ, проведенный в работе [40], подтверждает предположение о том, что эвристика *MTF* ускоряет вывод автоматов.

1.1.7.2. Битовые строки

Использование битовых строк для представления полных таблиц переходов накладывает следующее ограничение – для того, чтобы строке всегда соответствовал корректный автомат необходимо, чтобы число его состояний и выходных воздействий было степенью двух. Как правило, это ведет к добавлению фиктивных выходных воздействий автомата.

1.1.7.3. Строки над числами

При использовании строк над числовым алфавитом проблем с корректностью не возникает. Заметим, что алфавит для состояний и выходных воздействий может различаться. Этот метод эквивалентен методу, использующему представление автомата в виде таблицы переходов [36].

1.1.7.4. Достоинства и недостатки

Достоинством применения генетических алгоритмов над строками фиксированной длины являются простота реализации и естественность представления для генетических алгоритмов, что позволяет применять различные виды мутаций и скрещиваний для достижения лучшего результата.

Однако этот метод плохо подходит для задач со многими входными переменными, так как приходится задавать переходы автомата для всех комбинаций значений переменных. Длина строки, соответствующей автомату, растет экспоненциально. Поэтому увеличиваются затраты памяти на хранение данного представления и растет время работы генетического алгоритма.

Опыт показывает, что в реальных задачах, управляющие автоматы, построенные вручную, имеют гораздо меньше переходов. Причина этого, видимо, в том, что в большинстве задач входные воздействия имеют «локальную природу» по отношению к управляющим состояниям. В каждом состоянии значимым является лишь определенный, небольшой набор входных воздействий, остальные же не влияют на выбор перехода. Поэтому возникает задача разработки представления, которое бы позволило ограничивать число значимых для состояния воздействий. Кроме того, навязывание этого свойства в процессе оптимизации позволяет получить результат, более похожий на автомат, построенный вручную, а, следовательно, более понятный человеку.

1.1.8. Сравнение методов представления решений

В настоящем разделе приводится сравнение методов представления решений на примере задачи о «флибах» и итерированной дилеммы узника.

1.1.8.1. Строки в задаче о «флибах»

В работе [2] для решения задачи о «флибах» применяется генетический алгоритм, использующий кодирование хромосом в виде строк. Проиллюстрируем на примере способ кодирования автомата в виде строки. Рассмотрим автомат, граф переходов которого показан на рис. 15, в котором в начале каждой дуги указывается значение входной переменной, а в конце – значение выходной переменной.

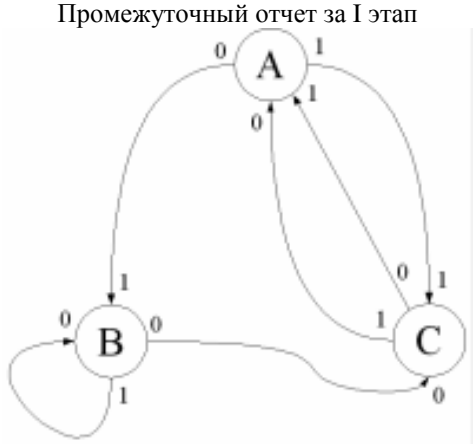


Рис. 15. Автомат из тремя состояниями для задачи о «флибах»

Табл. 4 – это таблица переходов и выходов этого автомата.

Таблица 4. Таблица переходов и выходов автомата для задачи о «флибах»

Старое состояние	0	Новое состояние	1	Новое состояние
A	1	B	1	C
B	0	C	0	B
C	1	A	0	A

В этой таблице для каждого состояния и значения входной переменной указаны два элемента: значения выходной переменной (второй и четвертый столбцы) и новое состояние (третий и пятый столбцы). Для представления таблицы переходов в виде строки столбцы со второго по пятый записываются в построчно. Например, для указанного автомата соответствующей строкой будет: `1B1C0C0B1A0A`.

Отметим, что число символов в такой строке в четыре раза превышает число состояний кодируемого ей автомата. Далее, если пронумеровать состояния автомата неотрицательными целыми числами, начиная с нуля, то если «флиб» находится в состоянии с номером `s`, а текущее входное воздействие равно `i`, то состояние, в которое перейдет «флиб» содержится в символе номер `4s+2i+1`, а значение выходного воздействия – в символе с номеров `4s+2i`.

Как показывают вычислительные эксперименты, проведенные в работе [10], описанная структура хромосомы позволяет строить «флибы», достаточно хорошо предсказывающие изменение среды. Однако точность предсказания все равно остается весьма далекой от 100%.

Например, при задающей среду битовой маске `11110100101111101001`, двадцати состояниях «флиба» и 400 поколениях наилучшая достигнутая точность предсказания равна 88%, а средняя – 78.3%. При битовой маске среды `10101111011000111101111100011001`, тридцати состояниях «флиба» и 1600 поколениях наилучшая достигнутая точность составила 87%, а средняя – 75.86%. Отметим, что существуют структуры хромосом, позволяющие при том же числе состояний и поколений строить «флибы», гораздо лучше предсказывающие среду.

1.1.8.2. Генетическое программирование в задаче о «флибах»

В работе [10] предлагается способ структурированного представления хромосом в задаче о «флибах», который не использует представление в виде битовых строк.

Этот способ основывается на кодировании «флиба» с помощью трех классов: `Flib`, `State` и `Branch`. Главным классом, представляющим «флиб», является класс `Flib`. Классы `State` и `Branch` реализуют его состояния и переходы, соответственно. Во всех этих классах имеется метод `Clone`, предназначенный для создания копии соответствующего объекта.

Опишем внутреннюю структуру указанных классов. Массив `_states` в классе `Flib` содержит состояния «флиба». Поле `_curStateIndex` класса `Flib` содержит номер текущего состояния «флиба». Число правильно предсказанных символов хранится в поле `_guessCount`. Метод `Step` переводит «флиб» в новое состояние и обновляет значение числа правильно предсказанных символов. Метод `Nulling` переводит «флиб» в начальное состояние.

В массиве `_branches` класса `State` хранятся дуги переходов из данного состояния. Номер элемента в массиве соответствует значению входной переменной.

Поля `_stateIndex` и `_output` класса `Branch` содержат номер состояния, в которое ведет переход, и значение выходной переменной, соответственно. Константа `TARGET_COUNT` определяет размер множества значений, которые может принимать выходная переменная, генерируемая «флибом».

Для описанного представления хромосомы в работе [10] были реализованы операции скрещивания и мутации.

Операция скрещивания. Операцию скрещивания для хромосом указанной структуры можно описать следующей последовательностью шагов:

1. Случайно выбирается одно из состояний нового «флиба».
2. В новый «флиб» добавляются состояния первого родителя с номерами, меньшими, чем у выбранного состояния, и состояния второго родителя с номерами, большими, чем у выбранного состояния.
3. Формируется и добавляется новое состояние, образованное из состояний обоих родителей с номерами, равными номеру выбранного состояния. Переходы из нового состояния формируются с помощью аналогичного алгоритма одноточечного скрещивания.

Операция мутации. Операцию мутации для указанной структуры хромомосмы можно описать следующей последовательностью операций:

1. Случайным образом выбирается состояние «флиба».
2. Случайным образом выбирается дуга перехода из выбранного состояния.
3. Случайным образом выбирается, что будет подвергнуто изменению – значение выходной переменной, соответствующее выбранному переходу, или номер состояния, в которое ведет выбранный переход.
    a. Если было определено, что изменяется значение выходной переменной, то ей присваивается выбранное случайным образом значение.
    b. Если было определено, то изменяется номер состояния, в которое ведет переход, то переход перенаправляется в случайно выбранное состояние.

Отметим, что генерации нового поколения вследствие применения операторов скрещивания и мутации дуги переходов во «флибах» изменяются случайным образом. Из-за этого в некоторых «флибах» могут появиться состояния, в которые невозможно попасть из начального состояния ни при какой последовательности значений входной переменной. Такие состояния назовем *недостижимыми*. Наоборот, состояния, в которые можно попасть из начального состояния при подаче на вход «флиба» некоторой последовательности состояний среды, назовем *достижимыми*.

Таким образом, после применения операций мутации и скрещивания в некоторых «флибах» могут появиться недостижимые состояния. Так как недостижимые состояния не несут никакой полезной информации, то в работе [10] предлагается *алгоритм восстановления связей*, применение

которого к «флибу» преобразует дуги его переходов так, чтобы в нем не было недостижимых состояний – все состояния были достижимыми.

Этот алгоритм можно записать в виде следующей последовательности шагов:

1. Формируется список доступных состояний – для этого можно применить любой метод обхода графов, например, обход в глубину или в ширину.
2. Выполняется цикл по всем состояниям. Если обрабатываемое на очередной итерации цикла состояние не входит в число достижимых, то для него выполняются следующие операции:
    a. Случайным образом выбирается состояние из списка достижимых состояний.
    b. Случайным образом выбирается одна их дуг выбранного состояния.
    c. В текущем состоянии одна из дуг заменяется дугой, которая ведет в то же состояние, что и дуга, выбранная в пункте b.
    d. Выбранная в пункте b, дуга заменяется дугой, которая ведет в текущее состояние.
    e. Обновляется список достижимых состояний – в него добавляется текущее состояние и все состояния, достижимые из него.

Отметим, что основным отличием алгоритма из работы [10] от алгоритма из работы [2] является то, что во всех «флибах» в каждом поколении все состояния являются достижимыми – никакой лишней неиспользуемой информации в хромосомах не содержится.

Приведем результаты вычислительных экспериментов, проведенных в работе [10]. Было проведено два вычислительных эксперимента, которые отличались битовой маской, задающей изменение среды, числом состояний «флиба» и числом поколений, в течение которых шла эволюция «флибов». Кроме этого, каждый раз алгоритм запускался как с применением восстановления связей между состояниями, так и без применения этого алгоритма.

При задающей среду битовой маске `11110100101111101001`, двадцати состояниях «флиба» и 400 поколениях наивысшая достигнутая точность предсказания равна 100% как при использовании восстановления связей между состояниями, так без применения этого алгоритма. Средняя точность предсказания составляет 92,26% без использования алгоритма восстановления связей, и 93,48% – при его использовании.

При битовой маске среды `10101111011000111101111100011001`, тридцати состояниях «флиба» и 1600 поколениях наивысшая достигнутая точность составила 97% как при использовании алгоритма восстановления связей между состояниями, так без его применении. Средняя точность предсказания при использовании восстановления связей между состояниями составила 92,72%, а без использования этого алгоритма – 90,44%.

Из изложенного следует, что предлагаемая в работе [10] структура хромосомы позволяет более эффективно строить «флибы», нежели чем при использовании хромосом, структура которых описана в работе [2]. Более того, применение операции восстановления связей между состояниями позволяет получать более точные предсказатели, чем при применении генетического алгоритма, который не использует такую операцию.

1.1.8.3. Генетический алгоритм в итерированной дилемме узника

В работе [52] для построения оптимальных стратегий, описываемых конечным автоматом, для итерированной дилеммы узника применяется генетический алгоритм. При этом строятся не обычные конечные автоматы, а вероятностные – их переходы снабжены вероятностями, с которыми выбирается этот переход. При этом сумма вероятностей по всем переходам из состояния равна единице. Основной целью работы [52] является не построение оптимального автомата, а описание алгебраической структуры, описывающей поведение в играх типа «итерированной дилеммы узника». Эта структура оказывается удобной для оптимизации с помощью генетических алгоритмов.

Опишем более подробно эту структуру и способ применения к ней генетических операторов мутации и скрещивания. Каждый автомат описывается несколькими матрицами. Их число совпадает с размером входного алфавита, а размер каждой матрицы составляет n×n, где n – число состояний в автомате. Обозначим матрицу, соответствующую входному символу c, как T(c). В матрице j-ый элемент ее i-ой строки равен вероятности перехода из состояния i в состояние j по входному символу c. Все элементы матриц являются вещественными числами из отрезка [0, 1]. Сумма элементов каждой строки каждой матрицы равна единице.

При этом отметим, что выходной символ автомата зависит только от состояния – рассматриваемые в работе [52] автоматы являются автоматами Мура. Более того, соответствие состояний выходным символам задано заранее и не может быть изменено. Таким образом, это соответствие не должно быть закодировано в хромосоме.

Исходя из изложенного, в хромосому автомата были включены только матрицы переходов. Для получения хромосомы матрицы записываются построчно. Например, если размер входного алфавита равен двум, число состояний автомата равно четырем, матрицы переходов приведены в табл. 5, 6, то хромосома будет иметь следующий вид: (0,1; 0,2; 0,5; 0,2); (0,2; 0.3; 0,5; 0,0); (0,3; 0,1; 0,5; 0,1); (0,4; 0,0; 0,5; 0,1;); (0,2; 0,1; 0,1; 0,6); (0,2; 0,3; 0,1; 0,4); (0,2; 0,3; 0,1; 0,4); (0,4; 0,1; 0,1; 0,4); (0,6; 0,1; 0,1; 0,2).

Таблица 5. Матрица переходов, соответствующая входному символу «0»

0,1	0,2	0,5	0,2
0,2	0,3	0,5	0,0
0,3	0,1	0,5	0,1
0,4	0,0	0,5	0,1

Таблица 6. Матрица переходов, соответствующая входному символу «1»

0,2	0,1	0,1	0,6
0,2	0,3	0,1	0,4
0,4	0,1	0,1	0,4
0,6	0,1	0,1	0,2

Отметим, что в используемом методе представления матриц внутри хромосомы матрицы по-прежнему отделяются друг от друга некоторыми границами. Также отметим, что вещественные числа не предлагается кодировать битовыми строками. Таким образом, все генетические операции оперируют с каждым числом, как с единым целым.

Приведем подробное описание генетических операций скрещивания и мутации для описанной структуры хромосомы.

Операция скрещивания. Рассмотрим две родительские хромосомы. Запишем их представления друг за другом в виде одной строки. После этого применим к полученной последовательности строк случайную перестановку. Первая половина полученной последовательности будет хромосомой одного из потомков, вторая половина – другого.

Операция мутации. Для выполнения операции мутации случайно выбирается одна из матриц, задающая автомат. У выбранной матрицы случайно выбранная строка заменяется на случайно сгенерированную.

Стратегии для итерированной дилеммы узника могут быть заданы автоматом Мили. Примеры таких автоматов приведены на рис. 16, 17.

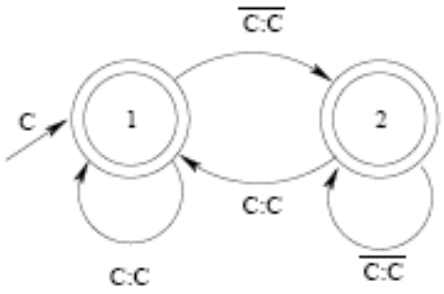


Рис. 16. Автомат, задающий стратегию «око за око»

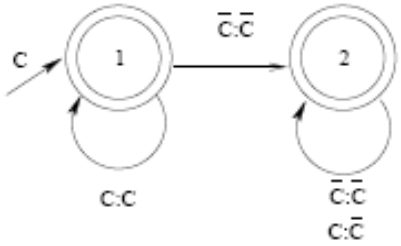


Рис. 17. Автомат, задающий «мстящую» стратегию

Поясним обозначения, используемые на этих рисунках. Метки на переходах имеют следующий формат: `входной символ : выходной символ`. Начальным состоянием в обоих автоматах является состояние 1. Символ *c* соответствует сотрудничеству (cooperate), а символ $\overline{c}$ – предательству (betray).

Первый из этих автоматов (рис. 16) задает стратегию «повторять действия противника на предыдущем ходе», а второй – начинает выбирать действие «предать» как только так поступил его противник. Недостаток этих автоматов состоит в том, что они статически задают стратегии поведения и плохо подходят для эволюционной оптимизации.

Для устранения этого недостатка в работе [52] предлагается использовать *вероятностный* автомат с двумя состояниями, который приведен на рис. 18.

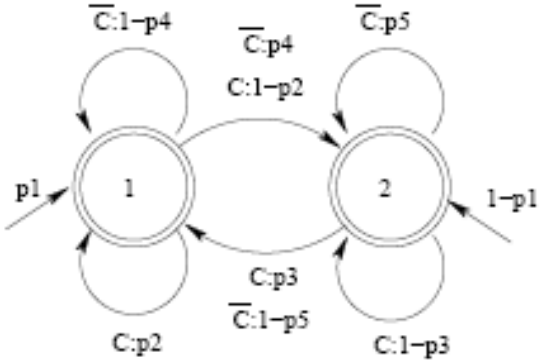


Рис. 18. Вероятностный автомат для итерированной дилеммы узника

Первое состояние этого автомата соответствует выбору действия «сотрудничать», второе – «предать». Матричное представление этого автомата приведено в табл. 7, 8.

Таблица 7. Матрица переходов, соответствующая входному символу «*C*»

p_2	$1 - p_2$
p_3	$1 - p_3$

Таблица 8. Матрица переходов, соответствующая входному символу «$\overline{C}$»

p_4	$1 - p_4$
p_5	$1 - p_5$

Кроме двух указанных автоматов необходимо еще задать вероятность того, что первое состояние является начальным – обозначим ее как p_1. Тогда вероятность того, что начальным будет второе состояние, равна $1 - p_1$.

Применение генетического алгоритма с описанными выше операциями мутации и скрещивания позволило построить вероятностный автомат с двумя состояниями, который показал результаты, сравнимые с результатом детерминированного автомата из восьми состояний, рассмотренного следующем разделе.

1.1.8.4. Генетическое программирование в итерированной дилемме узника

В работе [48] для поиска оптимальной стратегии поведения в итерированной дилемме узника используется генетическое программирование – хромосома представляет автомат в виде графа переходов, а в процессе оптимизации используются только операции мутации, а операции скрещивания не используются.

Число состояний в исследуемых автоматах было ограничено восемью для того, чтобы построенные автоматы могли быть в дальнейшем проанализированы вручную. При этом начальное поколение состояло из 100 автоматов, в каждом из которых было от одного до пяти состояний. В каждом состоянии по каждой паре входных символов {(C, C), (C, D), (D, C), (D, D)} на выход подавался случайный символ из множества {C, D}.

В дальнейшем эти автоматы подвергались мутациям, в число которых входили:

1. Изменение начального состояния.
2. Добавление состояния.
3. Удаление состояния.
4. Изменение выходного символа.
5. Изменение перехода.

Каждый раз выбирался случайно и равновероятно одна из разновидностей мутации. После этого случайно выбирался один из способов выполнения такой мутации. Например, если была выбрана мутация «изменение выходного символа», то случайно выбиралось состояние автомата. Далее случайно выбирался переход из этого состояния, и изменялся символ, соответствующий этому переходу.

Для определения приспособленности особей проводился турнир по круговой системе. При этом каждая игра состояла из 151 раунда. После этого выбирались 100 лучших особей, которые переходили в следующее поколение. Этот процесс продолжался на протяжении 50 поколений.

С помощью этого алгоритма при одном из запусков был построен автомат с восемью состояниями, граф переходов которого приведен на рис. 19.

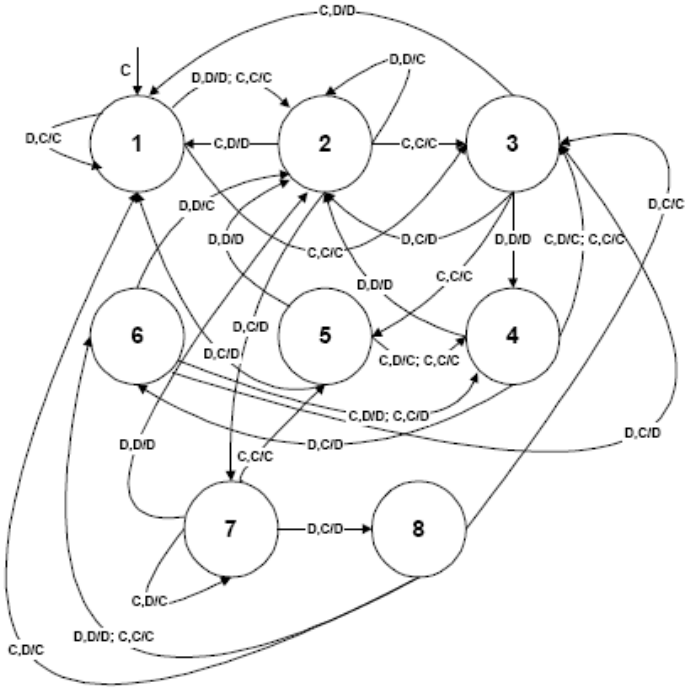


Рис. 19. Автомат с восемью состояниями для итерированной дилеммы узника

1.1.9. Рекомендации по выбору метода представления решений

В настоящем разделе приводятся рекомендации по выбору метода представления решений.

1.1.9.1. Задача о «флибах»

В разд. 1.1.8 были рассмотрены различные структуры хромосом, которые применяются для построения оптимальных управляющих автоматов в задаче об «флибах»: строки (разд. 1.1.8.1) и структурированное представление (разд. 1.1.8.2). Далее будет проведено их сравнение и даны некоторые рекомендации по выбору структуры хромосом, пригодных для решения сходных задач.

В табл. 9 приведена краткая сравнительная характеристика описанных ранее структур хромосом для задачи об «флибах». В этой таблице приведены основные параметры структур хромосом (например, «Метод построения автоматов», «Используются ли операторы мутации и скрещивания?», «Какие используются дополнительные операции?», «Является ли фиксированным число состояний в автомате?»). Кроме этого, в ней приведены параметры, показывающие сложность программной реализации метода, использующего такую структуру хромосом (например, «Требуется ли преобразование хромосомы в автомат при вычислении функции приспособленности?»).

Изучение этой таблицы позволяет ожидать, что второй способ (структурированное представление) позволяет добиться лучших результатов в этой задаче, так как при его использовании не требуется декодирования строки для получения графа переходов автомата. Еще одной причиной для подобных ожиданий является использование метода «Восстановление связей между состояниями», с помощью которого обеспечивается достижимость всех состояний. Отметим также, что оба метода обладают примерно одинаковой сложностью реализации, так как в одном требуется декодирование строки для получения графа переходов автомата, а во втором – реализация алгоритма восстановления связей между состояниями, который включает в себя обход графа переходов автомата с помощью поиска в глубину или в ширину.

Таблица 9. Сравнение структур хромосом в задаче о «флибах»

Параметр, по которому выполняется сравнение	Строки (разд. 1.1.8.1)	Структурированное представление (разд. 1.1.8.2)
Метод построения автоматов	Генетический алгоритм	Генетический алгоритм
Объект, который представляется в виде хромосомы	Конечный автомат Мили	Конечный автомат Мили
Требуется ли преобразование хромосомы в автомат при вычислении функции приспособленности?	Да, необходимо декодировать строку	Нет, автомат хранится в виде графа переходов
Используется ли оператор мутации?	Да	Да
Используется ли оператор скрещивания?	Да	Да
Какие используются дополнительные операции?	Никакие	Восстановление связей между состояниями
Является ли фиксированным число состояний в автомате?	Да, некоторые состояния могут быть недостижимы	Да, все состояния достижимы
Действия, которые необходимо произвести при кодировании автомата в хромосому	Преобразование автомата в строку по методу, описанному в разд. 1.1.8.1	Отсутствуют, хромосома хранит граф переходов автомата

Приведем результаты использования описанных структур для построения автоматов в задаче о «флибах» для сред, заданных масками `1111010010111101001` и `10101111011000111101111110011001` (табл. 10).

Таблица 10. Результаты применения структур хромосом в задаче о «флибах»

	Строки (разд. 1.1.8.1)	Структурированное представление (разд. 1.1.8.2)
Первая маска среды, 400 поколений, 20 состояний	Лучший результат: 88% Средний результат: 78,3%	Лучший результат: 100% Средний результат: 93,48%
Вторая маска среды, 1600 поколений, 30 состояний	Лучший результат: 87% Средний результат: 75,86%	Лучший результат: 97% Средний результат: 92,72%

Из табл.18 следует, что применение структурированного представления хромосом с использованием операции восстановления связей между состояниями позволяет добиться лучших результатов в задаче о «флибах». При этом сложность реализации этого метода примерно такая же, как и у метода, использующего представление хромосом в виде строк. Таким образом, для задачи о «флибах» и аналогичных задач лучшим оказывается структурированное представление хромосом с применением восстановления связей между состояниями (разд. 1.1.8.2).

1.1.9.2. Итерированная дилемма узника

В разд. 1.1.8 рассмотрены различные структуры хромосом, которые применяются для построения оптимальных управляющих автоматов в итерированной дилемме узника: графы переходов и представление вероятностных автоматов в виде матриц (разд. 1.1.8.3). Далее будет проведено их сравнение и даны некоторые рекомендации по выбору структуры хромосом, пригодных для решения сходных задач.

В табл. 11 приведена краткая сравнительная характеристика описанных ранее структур хромосом для итерированной дилеммы узника. В этой таблице приведены основные параметры структур хромосом (например, «Метод построения автоматов», «Используются ли операторы мутации и скрещивания?», «Какие используются дополнительные операции?», «Является ли фиксированным число состояний в автомате?»). Кроме этого, в ней приведены параметры, показывающие сложность программной реализации метода, использующего такую структуру хромосом (например, «Требуется ли преобразование хромосомы в автомат при вычислении функции приспособленности?»).

Информация, представленная в табл. 19, позволяет ожидать, что более эффективным окажется представление хромосом в виде вероятностных автоматов и их матриц переходов. Это связано с вероятностным характером задачи, включающей в себя несколько повторений. С другой стороны, детерминированный автомат позволяет добиться более стабильных результатов, к тому же его поведение может быть изучено вручную, в то время как поведение вероятностного автомата предсказать намного сложнее. Кроме того, для матричного представления используется оператор скрещивания, применение которого обычно позволяет добиться лучших результатов, нежели чем при применении только мутаций.

Таким образом, для итерированной дилеммы узника и сходных с ней задач более эффективным оказывается представление вероятностных автоматов в виде матриц переходов – эта структура хромосом описана в разд. 1.1.8.3.

Таблица 11. Сравнение структур хромосом в итерированной дилемме узника

Параметр, по которому выполняется сравнение	Матрицы (разд. 1.1.8.3)	Графы переходов (разд. 1.1.8.4)
Метод построения автоматов	Генетический алгоритм	Генетическое программирование
Объект, который представляется в виде хромосомы	Вероятностный автомат с действиями в состояниях	Конечный автомат Мили
Требуется ли преобразование хромосомы в автомат при вычислении функции приспособленности?	Да, необходимо декодировать матричное представление автомата	Нет, автомат хранится в виде графа переходов
Используется ли оператор мутации?	Да	Да
Используется ли оператор скрещивания?	Да	Нет
Какие используются дополнительные операции?	Никакие	Никакие
Является ли фиксированным число состояний в автомате?	Да, количество состояний равно двум	Нет, некоторые состояния могут быть недостижимы
Действия, которые необходимо произвести при кодировании автомата в хромосому	Преобразование автомата в матрицы переходов по методу, описанному в разд. 1.1.8.3	Отсутствуют, хромосома хранит граф переходов автомата

1.2. Обучение с подкреплением

Обучение с подкреплением (*Reinforcement Learning*) является наиболее распространенным видом машинного обучения. В настоящем разделе излагается метод построения систем управления с помощью обучения с подкреплением на основе работ [94, 100].

В ходе обучения с подкреплением автономный агент взаимодействует с окружающей средой, на каждом шаге совершая одно из фиксированных действий. Среда, в свою очередь, некоторым образом поощряет агента за каждое действие. Более формально, пусть S – множество возможных состояний агента, $A(s)$ – множество возможных действий из состояния $s \in S$. Агент, находясь в состоянии $s_t \in S$, совершает действие $a \in A(s_t)$, после чего в зависимости от совершенного действия переходит в новое состояние s_{t+1} и получает премию r_{t+1}. Будем говорить, что $\pi : S \to A$ определяет стратегию агента, если $a(s_t) = \pi(s_t)$.

Целью является построение такой стратегии, которая максимизирует математическое ожидание некоторой целевой функции F. В практических задачах наиболее часто используется модель бесконечного горизонта (*Infinite horizon model*):

$$F = \sum_{t=0} r_t \cdot \gamma^t, 0 \le \gamma < 1.$$

В случае, когда функции, определяющие по состоянию и совершенному действию новое состояние и премию, известны заранее, оптимальное решение может быть найдено с помощью

динамического программирования [34]. Однако в общем случае эти функции не известны агенту и вся информация о них получается непосредственно в ходе обучения.

Традиционным способом решения задач обучения с подкреплением является алгоритм *Q-learning* [110]. Введем функцию $Q: S \times A \to R$, определяющую для каждой пары состояние-действие (s,a) максимальную прибыль, которую получит агент, если, находясь в состоянии s, выберет действие a. Данная функция связана с оптимальной стратегией π^* следующим образом:

$$\pi^*(s) = \arg\max_a Q(s,a).$$

Таким образом, для нахождения оптимальной стратегии достаточно вычислить функцию Q.

Заметим, что эта функция удовлетворяет следующему реккурентному соотношению:

$$Q(s,a) = r_t + \gamma \cdot \max_{a \in A(s_t)} Q(s(a),a),$$

где $s(a)$ – состояние, в котором агент окажется после выполнения действия a. Тогда функция Q может быть вычислена итеративно в процессе взаимодействия с внешней средой – непосредственно во время решения задачи. Рассмотрим алгоритм подсчета более подробно.

На каждом шаге оценки значений функции Q хранятся в таблице, входами которой являются состояние и действие. При выполнении очередного действия оценка функции пересчитывается некоторым способом. Конкретный вариант пересчета определяет поведение агента. При этом возникает общая проблема, известная как «исследование или эксплуатация» (*Exploration vs. Exploitation*). Она заключается в том, что, с одной стороны, лучшие стратегии будут стабилизироваться, а с другой – полученные значения функции Q могут быть использованы для поиска более выгодных стратегий. Необходимо найти некий компромисс между использованием найденных стратегий и исследованием новых.

Классическим способом пересчета функции Q является метод одношаговой временной разности (*Temporal Difference*) [94]. При применении этого метода выражение для пересчета функции Q принимает следующий вид:

$$Q(s_t,a_t) \leftarrow (1-\alpha) \cdot Q(s_t,a_t) + \alpha \cdot (r_t + \gamma \cdot \max_{a \in A(s_t)} Q(s_{t+1},a))$$

Здесь $0 < \alpha \le 1$ – параметр, определяющий скорость изменения оценки функции.

Метод одношаговой временной разности является частным случаем метода временной разности (*TD(λ)*) [94]. Этот метод учитывает все последние шаги, а не только последний шаг агента. Можно определить временную разность d_i как разность предсказания будущего значения функции Q с реально полученным значением через i шагов. Ошибки предсказания на более давних шагах вносят меньший вклад по сравнению с ошибкой предсказания последнего на последнем шагу. Более формально, пересчет осуществляется по следующему правилу:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \cdot \sum_{i=1}^{t} \lambda^{i-1} \cdot d_i, \text{ где } 0 \le \lambda \le 1.$$

Методу одношаговой временной разности соответствует $\lambda = 0$.

Заметим, что на ранних стадиях обучения значения оценки функции Q могут сильно отличаться от ее реального значения. Если при этом всегда использовать максимум для выбора следующего действия – выполнять жадный выбор, то конечные значения оценки могут не сойтись к реальным значениям функции. Этот недостаток частично разрешен в алгоритме *SARSA (State-Action-Reward-State-Action)* [98]. При использовании этого подхода выбор очередного действия агента

осуществляется случайно. При этом учитываются текущая оценка значения функции Q – состояния с большим значением функции имеют большую вероятность быть выбранными. Правило для пересчета функции Q имеет следующий вид:

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot Q(s_t, a_t) + \alpha \cdot (r_t + \gamma \cdot Q(s_{t+1}, a_{t+1})),$$

где a_{t+1} – выбранное действие. Можно видеть, что при использовании жадной стратегии правило пересчета полностью соответствует правилу обновления метода одношаговой временной разности.

В практических задачах пространство состояний часто имеет большую размерность. При решении таких задач оказывается невозможным хранить оценки функции Q в виде таблицы. Вместо этого функция Q аппроксимируется. При этом аппроксимация обновляется в ходе обучения. Проведем краткий обзор используемых методов аппроксимации.

В работе [31] для аппроксимации функции Q применяется метод дискретизации. Пространство возможных состояний разбивается на области, для каждой из которых оценка значения хранится в таблице. Тем самым, некоторые состояния исходной задачи полагаются неразличимыми, после этого задача сводится к задаче меньшей размерности. Недостатком такого подхода является то, что его эффективность напрямую зависит от используемого разбиения.

Одной из первых работ, где для аппроксимации функции Q используются нейронные сети, является работа [101]. При этом автором предлагается метод пересчета коэффициентов нейронной сети непосредственно с учетом метода временной разницы. Пусть w – вектор коэффициентов нейронной сети. Пересчет осуществляется по следующей формуле:

$$w \leftarrow w + \alpha \cdot d_0 \cdot \nabla_w \left[\sum_{j=0}^{t-1} \lambda^j \cdot Q(s_{t-j}, a_{t-j}) \right].$$

Здесь ∇_w — градиент по w, $d_0 = r_t + \gamma \cdot \max_{a \in A(s_t)} Q(s_{t+1}, a) - Q(s_t, a_t)$ – одношаговая временная разность.

В работе [102] используются нейронные сети, которые состоят из нескольких слоев. Эти слои представляют собой разбиение пространства состояний на некоторые гиперпрямоугольники (рис. 20).

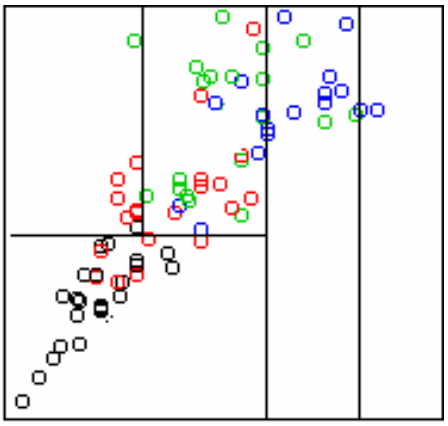


Рис. 20. Разбиение на гиперпрямоугольники

В каждом слое выбирается гиперпрямоугольник, который содержит состояние системы, поданное на вход нейронной сети. Таким образом, состоянию системы ставится в соответствие множество гиперпрямоугольников. Выходом нейронной сети является взвешенная сумма индексов

соответствующих гиперпрямоугольников. Вектор коэффициентов взвешенной суммы может обновляться в процессе обучения с использованием метода временной разницы.

В работе [33] функция аппроксимируется с помощью деревьев классификации (*Regression Tree*) – обобщения деревьев решений. В листьях дерева классификации хранится значение функции, в его внутренних узлах производится расщепление в зависимости от значений аргументов функции. Пример дерева классификации приведен на рис. 21.

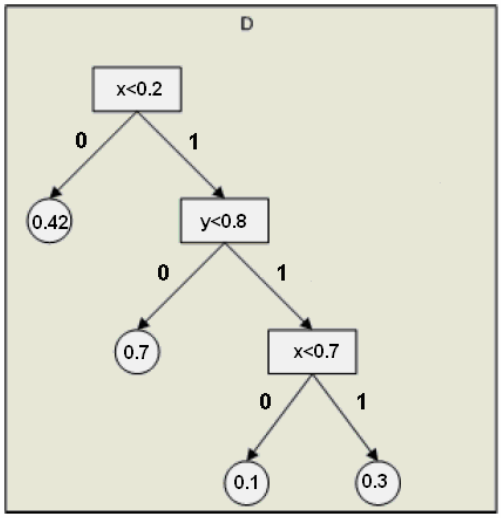


Рис. 21. Пример дерева классификации

В работе [92] используется аппроксимация функции с помощью *Kernel Regression* – в качестве приближения функции используется взвешенное среднее с произвольным ядром K:

$$F(x) = \frac{\sum_{i=1}^{n} K(x - x_i) \cdot y_i}{\sum_{i=1}^{n} K(x - x_i)}.$$

В табл. 172 приводится обзорное сравнение указанных работ.

Таблица 12. Сравнение работ, использующих обучение с подкреплением

Название работы	Решаемая задача	Метод аппроксимации
Neuron-like elements that can solve difficult learning control problems	Задача балансировки шеста на тележке	Дискретизация
Implementation details of *TD(λ)* procedure for case of vector predictions and backpropagation	Не указана	Нейронные сети, пересчет производится с учетом временной разницы
Reinforcement Learning for Multi-linked Manipulator Control	Задача управления роботом	Нейронные сети, состоящие из нескольких слоев
Tree-Based Batch Mode Reinforcement Learning	Задача балансировки при управлении велосипедом	Деревья классификации
Kernel-based Reinforcement Learning in Average-cost Problems	Выбор оптимального портфеля акций	Kernel Regression

1.3. НЕЧЕТКАЯ ЛОГИКА

В настоящем разделе излагаются основы теории нечетких множеств и нечеткой логики.

Математическая теория *нечетких множеств* и *нечеткая логика* являются обобщениями классической теории множеств и классической формальной логики. Данные понятия были впервые предложены Лотфи Заде [6]. Основной причиной появления новой теории стало наличие нечетких и приближенных рассуждений при описании человеком процессов, систем, объектов.

Приведем основные определения, используемые в нечеткой логике.

1.3.1. Нечеткая математика

Определение 1. Нечетким множеством C называется множество упорядоченных пар вида $C = \{\mathrm{MF_c}(x)/x\}, x \in X, \mathrm{MF_c}(x) : X \to [0,1]$. Множество X называется универсальным, функция $\mathrm{MF_c}(x)$ – функцией принадлежности.

Значение $\mathrm{MF_c}(x) = 0$ означает отсутствие принадлежности к множеству, значение «1» – полную принадлежность.

Для нечетких множеств, как и для обычных, определены основные логические операции. Основными, необходимыми для расчетов, являются пересечение и объединение. В общем виде для операций пересечения и объединения используются *операторы t-нормы* и *t-конормы* соотвественно [44, 116].

Определение 2. Оператором t-нормы называется функция $T : [0,1] \to [0,1] \times [0,1]$, обладающая следующими свойствами:

- $T(0,0) = 0$.
- $T(x,1) = T(1,y) = 1$.
- $T(x,y) = T(y,x)$.
- $T(x,T(y,z)) = T(T(x,y),z)$.
- $x \le z, y \le u \Rightarrow T(x,y) \le T(z,u)$.

Определение 3. Оператором t-конормы называется функция $S:[0,1]\to[0,1]\times[0,1]$, обладающая следующими свойствами:

- $S(0,0)=0$.
- $S(x,1)=S(1,x)=x$.
- $S(x,y)=S(y,x)$.
- $S(x,S(y,z))=S(S(x,y),z)$.
- $x\le z, y\le u \Rightarrow S(x,y)\le S(z,u)$.

Пересечение двух нечетких множеств (нечеткое «И»): $A\cap B: MF_{A\cap B}(x)=T(MF_A(x),MF_B(x))$.

Объединение двух нечетких множеств (нечеткое «ИЛИ»): $A\cup B: MF_{A\cup B}(x)=S(MF_A(x),MF_B(x))$.

Заметим, что свойства *t*-норм и *t*-конорм суть обобщение свойств операторов конъюнкции и дизъюнкции в классической двоичной логике. Они играют ключевую роль в нечеткой логике, и, соответственно, нечетком управлении, а потому рассмотрим их подробнее.

В табл. 13 приведены примеры *t*-норм.

Таблица 13. Примеры *t*-норм

Название	Формула
Минимум	$x\wedge y=\min(x,y)$
Произведение	$T(x,y)=x\cdot y$
Конъюнкция Лукасевича	$x\otimes y=\max(0,x+y-1)$
Сильное произведение	$T_W(x,y)=\begin{cases}\min(x,y), \text{если } \max(x,y)=1\\ 0, \text{в противном случае}\end{cases}$
Нильпотентный минимум	$T_F(x,y)=\begin{cases}\min(x,y), \text{если } x+y>1\\ 0, \text{в противном случае}\end{cases}$

Определение 4. Если $T_1(a,b)\le T_2(a,b)$ для любых $a,b\in[0,1]$, то говорят, что T_2 *сильнее* T_1, и пишут $T_1\le T_2$.

Утверждение 1. Для каждой *t*-нормы верно:

$$T_W\le T\le \wedge$$

Определение 5. t-конорма S называется *двойственной t*-норме T, если для всех $a,b\in[0,1]$ верно равенство $S(a,b)=1-T(1-a,1-b)$.

В табл. 14 приведены *t*-конормы, двойственные *t*-нормам, перечисленным в табл. 13.

Таблица 14. Примеры *t*-конорм

Название	Формула
Максимум	$x \vee y = \max(x, y)$
Вероятностная сумма	$S(x, y) = x + y - x \cdot y$
Дизъюнкция Лукасевича	$x \oplus y = \min(1, x + y)$
Сильная сумма	$S_W(x, y) = \begin{cases} \max(x, y), \text{если } \min(x, y) = 0 \\ 1, \text{в противном случае} \end{cases}$
Нильпотентный максимум	$S_F(x, y) = \begin{cases} \max(x, y), \text{если } x + y < 1 \\ 1, \text{в противном случае} \end{cases}$

Аналогично *t*-нормам для *t*-конорм вводится частичный порядок и устанавливается следующее утверждение:

$$\vee \le S \le S_W .$$

1.3.2. Нечеткая переменная

Определение 6. Нечеткой переменной называется набор (*N*, *X*, *A*), где *N* – это название переменной, *X* – универсальное множество (область рассуждений), *A* – нечеткое множество на *X*.

Значениями *лингвистической переменной* могут быть нечеткие переменные – лингвистическая переменная находится на более высоком уровне, чем нечеткая переменная. Каждая лингвистическая переменная состоит из:

- названия;
- множества своих значений, которое также называется базовым терм-множеством *T*. Элементы базового терм-множества представляют собой названия нечетких переменных;
- универсального множества *X*.

1.3.3. Нечеткий логический вывод

Определение 7. Нечетким n-арным отношением R, заданным на области определения $X = X_1 \times \ldots \times X_n$, называется упорядоченное множество

$$R = \{((x_1, \ldots, x_n), MF_R(x_1, \ldots, x_n)) \mid (x_1, \ldots, x_n) \in X\}, \text{где } MF_R : X \to [0,1]$$

Часто рассматривают отношения, полученные путем агрегации нечетких множеств, заданных на различных одномерных областях [17]. Например, высказывание $(x_1 = A)[\wedge,\vee](x_2 = B)$ задает отношение с функцией принадлежности $MF_R(x_1, x_2) = [T,S](MF_A(x_1), MF_B(x_2))$, где $[T,S]$ – некоторая $[t\text{-норма}, t\text{-конорма}]$.

Определение 8. Нечеткой импликацией называется нечеткое отношение, связанное с правилом R, простейшая форма которого следующая:

$$IF(x = A)\ THEN\ (y = B), \text{где } x, y \text{ - нечеткие переменные}, MF_R : D(x) \times D(y) \to [0,1].$$

Наиболее простым и популярным является **оператор импликации Мамдани** [78], основанный на предположении, что степень истинности заключения $MF_B(y)$ не может быть выше, чем степень выполнения условия $MF_A(x)$:

$$MF_{A \to B}(x, y) = \min(MF_A(x), MF_B(y))$$

Оператором, имеющим наилучшие характеристки по определенному набору критериев, является **импликация Лукасевича** [15]:

$$\min(1,1-\mathrm{MF}_R(x)+\mathrm{MF}_R(y))$$

Основой для проведения операции *нечеткого логического вывода* является база правил, содержащая нечеткие высказывания в форме «IF-THEN» и функции принадлежности для соответствующих лингвистических термов. При этом должны соблюдаться следующие условия:

1. Существует хотя бы одно правило для каждого лингвистического терма выходной переменной.
2. Для любого терма входной переменной имеется хотя бы одно правило, в котором этот терм используется в качестве предпосылки (левая часть правила).

Пусть в базе правил имеется m правил вида:

$$\begin{array}{lllll}
R_1: & \text{IF} & x_1 = A_{11} \wedge \ldots \wedge x_n = A_{1n} & \text{THEN} & y = B_1 \\
\ldots & & & & \\
R_i: & \text{IF} & x_1 = A_{i1} \wedge \ldots \wedge x_n = A_{in} & \text{THEN} & y = B_i \\
\ldots & & & & \\
R_m: & \text{IF} & x_1 = A_{m1} \wedge \ldots \wedge x_n = A_{mn} & \text{THEN} & y = B_m
\end{array}$$

где $x_k, k = 1\ldots n$ – входные переменные; y – выходная переменная; A_{ik} – заданные нечеткие множества с функциями принадлежности.

Результатом нечеткого вывода является четкое значение переменной y на основе заданных четких значений $x_k, k = 1\ldots n$.

Любое правило можно привести к такому виду, какой используется выше. Действительно, если в правиле встречается дизъюнкция, его можно представить в виде двух более простых правил.

$$\text{IF } (x_1 = A_1) \vee (x_2 = A_2) \text{ THEN } y = B \Leftrightarrow \begin{cases} \text{IF } x_1 = A_1 \text{ THEN } y = B \\ \vee \\ \text{IF } x_2 = A_2 \text{ THEN } y = B \end{cases}$$

$$\text{IF } x = A \text{ THEN } (y_1 = B_1) \vee (y_2 = B_2) \Leftrightarrow \begin{cases} \text{IF } x = A \text{ THEN } y_1 = B_1 \\ \vee \\ \text{IF } x = A \text{ THEN } y_2 = B_2 \end{cases}$$

В общем случае алгоритм нечеткого вывода включает три этапа: введение нечеткости (фазификация), механизм нечеткого вывода и приведение к четкости, или дефазификация (рис. 22).

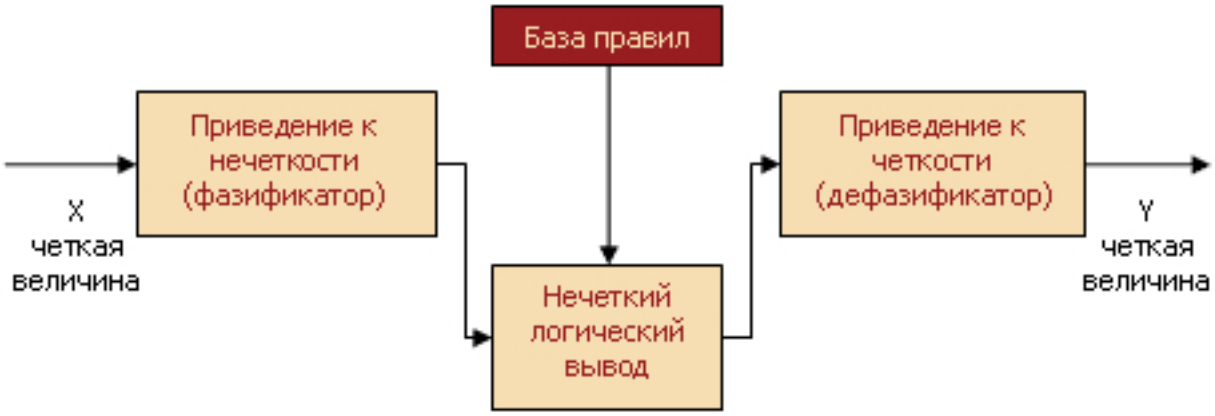


Рис. 22. Система нечеткого логического вывода

Наиболее известные методы дефазификации:

- метод среднего максимума;
- метод первого максимума;
- метод последнего максимума;
- метод центра тяжести;
- метод центра сумм;
- метод высот.

Заметим, что механизм нечеткого вывода можно рассматривать как вычисление функции принадлежности для отношения. Действительно, каждое правило представляет собой импликацию, левая часть которой – агрегация нечетких множеств, правая – просто нечеткое множество. Таким образом, каждое правило есть отношение. База правил – объединение отношений, то есть также отношение. В этом отношении встречаются три оператора: конъюнкции, дизъюнкции и импликации. Соответственно, механизмы нечеткого вывода могут различаться в трех элементах:

- вычисление степени выполнения каждого правила в отдельности (применяемая *t*-норма);
- вычисление функции принадлежности заключения для каждого правила (применяемый оператор импликации);
- вычисление результирующей функции принадлежности выходного значения (применяемая *t*-конорма).

В табл. 15 приведен пример алгоритма нечеткого вывода.

Таблица 15. Пример алгоритма нечеткого вывода

Название параметра	Значение параметра
Агрегация условий правил	Оператор *min*
Оператор импликации	Импликация Мамдани
Аккумуляция	Оператор *max*

1.3.4. Нечеткие модели

В разд. 1.3.3 рассматривался исторически первый и наиболее часто используемый тип нечеткой модели – модель Мамдани. Другим важным типом моделей являются модели Такаги-Сугено [102]. От моделей Мамдани эти модели отличаются формой правил (табл. 16).

Таблица 16. Отличия моделей Мамдани и Такаги-Сугено

Модель	Форма правил
Модель Мамдани	$\text{IF}\,(x_1 = A_1) \wedge \ldots \wedge (x_n = A_n)\,\text{THEN}\; y = B$
Модель Такаги-Сугено	$\text{IF}\,(x_1 = A_1) \wedge \ldots \wedge (x_n = A_n)\,\text{THEN}\; y = f(x_1, ..., x_n)$

Обычно функция f – линейна ($f(x_1, \ldots, x_n) = a_0 + a_1\, x_1 + \ldots + a_n\, x_n$).

Модели Такаги-Сугено являются обобщением моделей Мамдани – они сложнее. Однако за счет большей сложности они могут лучше аппроксимировать моделируемую систему, и, следовательно, позволяют упростить ценой некоторой потери точности алгоритм нечеткого вывода. Пусть задано *m* правил:

$$
\begin{array}{llll}
R_1: & \text{IF} & x_1 = A_{11} \wedge \ldots \wedge x_n = A_{1n} & \text{THEN} \quad y = f_1(x_1, ..., x_n) \\
\ldots & & & \\
R_i: & \text{IF} & x_1 = A_{i1} \wedge \ldots \wedge x_n = A_{in} & \text{THEN} \quad y = f_i(x_1, ..., x_n) \\
\ldots & & & \\
R_m: & \text{IF} & x_1 = A_{m1} \wedge \ldots \wedge x_n = A_{mn} & \text{THEN} \quad y = f_m(x_1, ..., x_n)
\end{array}
$$

и значения входных переменных $(x_1, x_2, \ldots, x_n)$, тогда выходное значение может быть вычислено как взвешенная сумма выходных значений для каждого правила

$$y = \frac{\sum_{i=1}^{m} MF_{R_i}\, f_i(x_1, \ldots, x_n)}{\sum_{i=1}^{m} f_i(x_1, \ldots, x_n)},$$

где весовой коэффициент правила MF_{R_i} есть его степень выполнения.

Еще одним типом нечетких моделей являются реляционные модели [94]. В отличие от рассмотренных выше моделей, в реляционных моделях правила не считаются абсолютно истинными, а им присваивается какой-то коэффициент доверия. Преимуществом данного типа моделей является возможность настройки коэффициентов доверия в процессе работы системы, недостатком – экспоненциальная от числа входных параметров сложность модели.

1.3.5. Построение нечетких моделей

В предыдущих разделах было описано, как работают нечеткие модели, однако нечеткую модель еще надо построить. Способы построения можно разделить на три класса:

- построение нечетких моделей на основе экспертных знаний;
- построение **самонастраивающихся** нечетких моделей;
- построение **саморганизующихся** и **самонастраивающихся** нечетких моделей.

Определение 9. Самонастраивающейся нечеткой моделью называется модель с фиксированными правилами и нечеткими множествами, но настраивающемися параметрами и типами функций принадлежности.

Определение 10. Самоорганизующейся нечеткой моделью называется модель, имеющая собственные автоматические процедуры определения оптимального числа и формы правил и нечетких множеств, использующихся для описания всех переменных модели.

Построение нечетких моделей на основе экспертных знаний не требует применения алгоритмов машинного обучения, а следовательно, является наиболее точным и наименее требовательным к машинным ресурсам способом. Однако оно не всегда возможно. Если для механических и электрических схем иногда удается построить точные модели [65], то для тепловых

или химических систем модели, построенные на основе экспертных знаний, как правило, менее точны. Еще сложнее построение моделей биологических, экономических и социальных систем.

Самонастройка и самоорганизация нечеткой модели производятся на основе измерений входов и выходов системы. По понятным соображениям для самоорганизации модели требуются больше информации. Важнейшей подзадачей самоорганизации сети является определение существенных входов системы, что позволяет сократить сложность модели.

Задача самонастройки сети является задачей оптимизации параметров и может решаться различными оптимизационными методами. Например, таких, как

- **Методы, основанные на использовании нейронечетких сетей**. Они связаны с преобразованием нечеткой модели в нейронечеткую сеть.
- **Поисковые методы**. Эти методы делятся на *упорядоченные* и *неупорядоченные*. Наиболее часто используемым методом упорядоченного поиска является метод, основанный на применении генетических алгоритмов.
- **Методы, основанные на кластеризации**. Эти методы сочетают в себе настройку параметров модели и ее структуризацию.

1.4. Искусственные нейронные сети

В данном разделе представлены основные сведения об искусственных нейронных сетях. [5, 11, 20].

1.4.1. Общие сведения

Прототипом искусственных нейронных сетей являются биологические нейронные сети, состоящие из множества нейронов – нервных клеток, соединенных между собой посредством электрохимических связей. Устройство биологического нейрона достаточно сложно и до сих пор не полностью изучено, но имеющихся знаний достаточно для того, чтобы построить несложную математическую модель нейрона. Такая модель называется искусственным нейроном. Одна из наиболее часто используемых реализаций искусственного нейрона выглядит следующим образом (рис. 23).

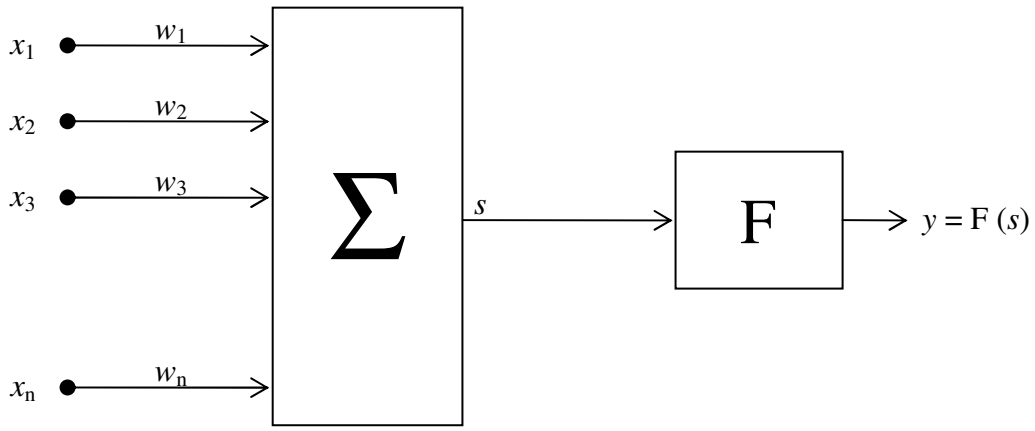


Рис. 23. Искусственный нейрон

На этом рисунке x_i – входные сигналы нейрона, w_i – веса входов, соответствующие силам синаптических связей биологического прототипа. Сигналы, поступающие на вход нейрона, умножаются на соответствующие веса и суммируются, что дает суммарный сигнал s. Этот сигнал

затем преобразуется так называемой активационной функцией *F* и в преобразованном виде поступает на выход нейрона. Таким образом, формально можно представить работу нейрона следующим образом:

$$y = F\left(\sum_i x_i \cdot w_i\right).$$

Активационная функция *F* для случая двоичных сигналов (0, 1) представляет собой пороговую функцию и имеет следующий вид:

$$F(x) = \begin{cases} 0, x < T \\ 1, x > T \end{cases}.$$

Для случая непрерывных (аналоговых) сигналов, активационная функция может иметь различный вид: от самого простого – линейного ($F(x) = k \cdot x$) до, теоретически, любого другого, лишь бы она оставалась непрерывной.

Однако обычно, по аналогии с биологическими нейронами, функция *F* принимается линейной в некотором диапазоне, и затем переходящей в насыщение при возрастании или убывании сигнала. Такая активационная функция, сжимающая весь диапазон суммарного сигнала до конечного интервала, называется *сигмоидной функцией*. Варианты формального задания такой функции могут быть различны, например $F(x) = th(x)$, или $F(x) = 1/(1 + e^{-x})$. При этом возможен сдвиг и масштабирование.

Следует понимать, что искусственный нейрон не повторяет, а лишь моделирует свой биологический прототип. Ряд особенностей биологических нейронов остаются при этом неучтенными, в частности, данная модель не рассматривает временную динамику изменения сигналов, их частотные свойства и многое другое. Однако, несмотря на всю простоту этой модели, сети, построенные из таких нейронов, демонстрируют многие замечательные свойства, находящие самое разнообразное практическое применение.

Существуют различные способы построения нейронных сетей из отдельных нейронов. В принципе, нейроны в сети могут быть связаны друг с другом произвольным образом, вплоть до предельного случая полносвязных сетей, когда на вход каждого нейрона поступают сигналы с выхода всех нейронов сети, включая его самого. Однако для удобства анализа чаще всего рассматриваются сети, организованные послойно. При этом сигналы с выхода нейронов предыдущего слоя поступают на входы следующего. Такая «слоистая» сеть показана на рис. 24.

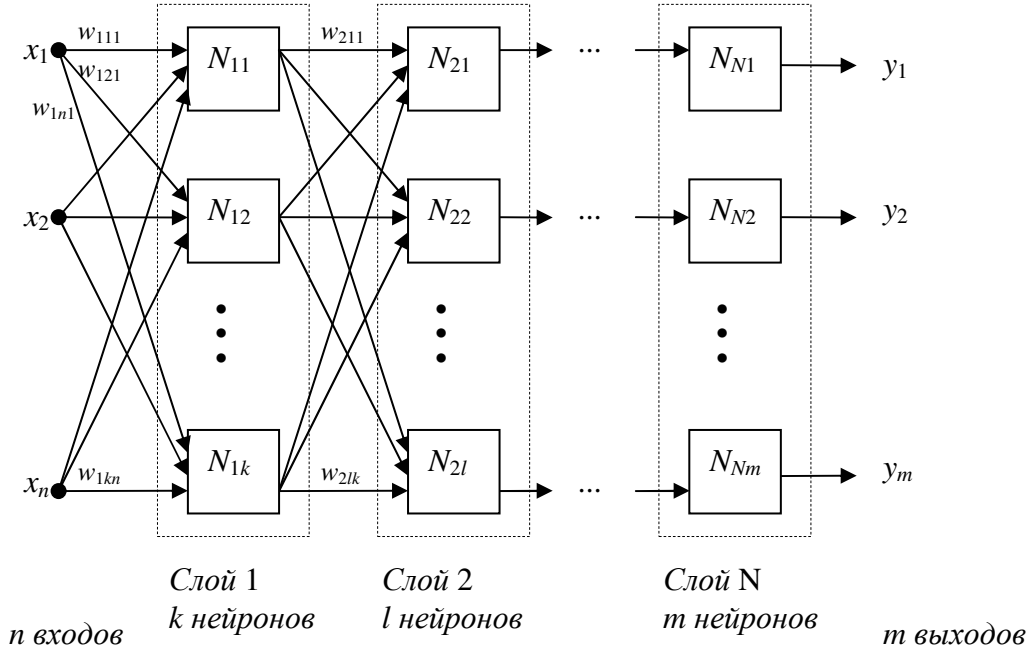


Рис. 24. Нейронная сеть слоистой архитектуры

Для удобства программной реализации входы нейронной сети часто представляют как еще один дополнительный слой из *n* нейронов. В этом случае он становится первым (нулевым), а номера остальных слоев увеличиваются на единицу. Из-за этого существует некоторое разногласие по вопросу о процедуре подсчета числа слоев сети, так как иногда этот входной слой учитывается при подсчете, а иногда нет. Поэтому самая простая из возможных сетей получается либо двух–, либо однослойной.

Сеть слоистой архитектуры не реализует всех возможностей нейронных сетей. Сети более общего вида имеют также обратные связи, соединяющие ее выходы с входами. Если в сетях без обратной связи, называемых также «сети прямого распространения», выход сети полностью определяется текущими значениями ее входов, то в сетях с обратными связями («рекуррентных») состояние сети зависит также и от предыдущих значений ее выходов – от предыстории. Возникающий эффект часто ассоциируют с кратковременной памятью человека.

Но, пожалуй, основным свойством нейронных сетей является их способность к обучению. Обучение состоит в том, что при предъявлении входных данных нейронная сеть изменяет значения весовых коэффициентов своих нейронов в соответствии с используемым алгоритмом обучения таким образом, чтобы данные, получаемые на выходе сети, соответствовали требуемым.

1.4.2. Методы обучения

Существует несколько основных видов алгоритмов обучения нейронных сетей. Различают алгоритмы обучения «с учителем» и «без учителя».

Обучение «с учителем» предполагает, что для каждого входного вектора $\{x_i\}$ существует целевой вектор $\{y_{D\,i}\}$, представляющий собой требуемый выход. Вместе они называются обучающей парой. Сеть обучается на некотором множестве таких обучающих пар. При обучении на вход сети предъявляется входной вектор, сеть вычисляет выходной вектор, и этот вектор сравнивается с соответствующим целевым вектором. Разность выходного и целевого векторов (ошибка сети) с помощью обратной связи подается в сеть, и веса нейронов изменяются в соответствии с алгоритмом,

стремящимся минимизировать ошибку. Векторы обучающего множества предъявляются последовательно, вычисляются ошибки, и веса нейронов подстраиваются до тех пор, пока ошибка по всему обучающему множеству не достигнет приемлемо низкого уровня. Основной алгоритм обучения «с учителем» – это обратное распространение ошибки, далее он будет рассмотрен более подробно.

Несмотря на многочисленные прикладные достижения, обучение «с учителем» представляется биологически неправдоподобным. Трудно вообразить обучающий механизм в мозге, который бы сравнивал желаемые и действительные значения выходов, выполняя коррекцию с помощью обратной связи. Даже если допустить наличие подобного механизма, остается непонятным, откуда возникают желаемые значения выходов. Обучение «без учителя» является намного более правдоподобной моделью обучения в биологической системе. Развитая Кохоненом и другими исследователями, эта модель не нуждается в целевом векторе для выходов и, следовательно, не требует сравнения с предопределенными идеальными ответами. Обучающее множество состоит лишь из входных векторов.

В *сети Кохонена* обучающий алгоритм подстраивает веса сети так, чтобы получались согласованные выходные векторы – чтобы предъявление достаточно близких входных векторов давало одинаковые выходы. Обучение *сети Кохонена* (она имеет только один слой) происходит следующим образом: на вход подается вектор $\{x_i\}$, после этого среди всех нейронов выбирается один, сигнал на выходе которого наибольший. Веса этого нейрона настраиваются по правилу:

$$w_i(n+1) = w_i(n) + \alpha(x_i - w_i(n)),$$

где $w_i(n)$ – значение веса до подстройки, $w_i(n+1)$ – его значение после подстройки, α – коэффициент скорости обучения. Таким образом веса нейрона-победителя приближаются к компонентам входного вектора. Для сильно различающихся входных векторов победителями становятся разные нейроны. Если же входные векторы сходны, они возбуждают сильнее всего один и тот же нейрон. Таким образом, каждый нейрон учится распознавать «свои» входные векторы. Процесс обучения выделяет статистические свойства обучающего множества и группирует сходные векторы в классы. Предъявление на вход вектора из данного класса даст определенный выходной вектор, но до обучения невозможно предсказать, какой выход будет производиться данным классом входных векторов. Следовательно, выходы подобной сети должны трансформироваться в некоторую понятную форму, обусловленную процессом обучения. Это не является серьезной проблемой. Обычно несложно идентифицировать связь между входом и выходом, установленную сетью.

Большинство современных алгоритмов обучения выросло из концепций Хебба. Им предложена модель обучения «без учителя», в которой сила (вес) синаптической связи между двумя нейронами возрастает, если активированы оба нейрона: источник и приемник. При этом часто используемые сетью пути усиливаются, и феномен привычки и обучения через повторение получает свое объяснение. В искусственной нейронной сети, использующей обучение по Хеббу, наращивание весов определяется произведением уровней возбуждения передающего и принимающего нейронов. Это можно записать как

$$w_{ij}(n+1) = w_{ij}(n) + \alpha y_i \cdot y_j,$$

где $w_{ij}(n)$ – значение веса до подстройки, $w_{ij}(n+1)$ – его значение после подстройки, α – коэффициент скорости обучения, y_i и y_j – выходы нейронов i и j соответственно. Сети, использующие обучение по Хеббу, конструктивно развивались, однако за последние 20 лет были развиты более эффективные и скоростные алгоритмы обучения.

1.4.3. Обратное распространение ошибки

Наиболее известным методом обучения нейронных сетей «с учителем» является процедура обратного распространения ошибки. Обратное распространение – это систематический метод для обучения многослойных искусственных нейронных сетей. Он имеет математическое обоснование, изложенное, например в работе [5], где обучение по методу обратного распространения ошибки рассматривается как задача оптимизации. Так как такое обучение можно свести к задаче оптимизации, для него оказываются применимы различные методы оптимизации, разработанные для других целей. В частности, в работе [79] описывается базовый алгоритм работы процедуры обратного распространения и различные его модификации, направленные на увеличение скорости сходимости алгоритма.

Алгоритм обучения нейронной сети по методу обратного распространения приведен на рис. 25.

1. Подготовить набор пар «входной вектор – выходной вектор» $\{x, y_D\}$.
2. Основной цикл обучения.

   Обнулить приращения весов dw всех нейронов.

   Цикл по всем парам векторов.

   Подать очередной входной вектор на вход сети.

   Вычислить выход сети для каждого нейрона, слой за слоем, начиная с первого. При этом сохранять промежуточный результат суммирования s для каждого нейрона.

   $$s = \sum_i x_i \cdot w_i \ , \ y = F(s)$$

   Выход последнего слоя и будет выходом сети.

   Сравнить полученный выход сети y с выходным вектором обучающего множества y_D. Разница полученного и требуемого выходов даст ошибку нейронов выходного слоя $e_y = y - y_D$.

   Цикл по всем слоям, начиная с последнего.

   Посчитать внутреннюю ошибку нейронов по формуле

   $e_s = e_y \cdot F'(s)$,

   где $F'(s)$ – производная активационной функции в точке s.

   Посчитать ошибку нейронов предыдущего слоя по формуле

   $$e_{y_j} = \sum_i e_{si} \cdot w_{ij} \ ,$$

   где индекс j относится к нейрону предыдущего слоя, а суммирование ведется по связям w_{ij} между нейроном предыдущего и нейронами текущего слоя.

   Изменить приращения весов всех нейронов по формуле

   $$dw_{ij} = dw_{ij} + e_{si} \cdot y_j \ ,$$

   где e_{si} – внутренняя ошибка нейрона текущего слоя, y_j – выход нейрона предыдущего слоя.

   Для всех нейронов изменить их веса w в соответствии с посчитанными в цикле приращениями dw и заданной скоростью обучения α: $w = w + \alpha \cdot dw$.

   Если максимальная по всем парам обучающих векторов ошибка сети оказалась достаточно малой, можно за-

вершить процесс обучения.

3. Сеть обучена.

Рис. 25. Алгоритм обучения по методу обратного распространения ошибки

Можно показать, как это делается в работе [5], что данный базовый алгоритм обучения нейронной сети на самом деле реализует ни что иное, как оптимизацию по методу наискорейшего спуска. Переменными при этом являются w – веса нейронов сети, а получаемая на каждом шаге основного цикла величина dw – это антиградиент функции ошибки, то направление изменения вектора w, вдоль которого среднеквадратичная ошибка сети убывает быстрее всего.

В работе [79] описаны также различные модификации базового метода, позволяющие увеличить скорость его сходимости. Это метод с автоподстройкой скорости обучения, «мягкий» метод обратного распространения, где скорость обучения задается для каждого веса отдельно, а также методы сопряженного градиента и методы второго порядка (квазиньютоновские).

1.4.4. Нейронные сети в управлении

Выбор в пользу нейронных сетей для их применения в управлении динамическими системами обосновывается тем, что они естественным образом работают с данными большой размерности (две и более входных переменных) и являются удобным инструментом для аппроксимации функций. Здесь необходимо сослаться на результат, приводимый в работе [5]: *обобщенная аппроксимационная теорема Стоуна.*

Пусть $E \subseteq C(X)$ – замкнутое линейное подпространство в $C(X)$, $1 \in E$, функции из E разделяют точки в X, и E замкнуто относительно нелинейной унарной операции $f \in C(\mathbf{R})$. Тогда $E=C(X)$.

Таким образом, утверждается, что с помощью линейных операций и каскадного соединения можно из произвольных нелинейных элементов получить любой требуемый результат с любой наперед заданной точностью. Это, в свою очередь, означает, что нейронная сеть, в которой в качестве активационной функции нейронов используется произвольная нелинейная функция, различающая точки на вещественной оси (например, сигмоидная функция), может сколь угодно точно аппроксимировать любую непрерывную функцию.

Однако теоретический результат не говорит ничего о практическом применении нейронных сетей. Практика же показывает, что нейронные сети, состоящие из небольшого числа нейронов, могут с приемлемой степенью точности аппроксимировать гладкие функции с медленно меняющейся производной благодаря особенностям своего построения. Именно такие функции часто возникают в задачах управления.

1.5. Построение систем управления мобильными роботами

В настоящем разделе излагаются методы построения систем управления интеллектуальными мобильными роботами. Под интеллектуальным роботом подразумевается машина, способная получать знания из окружающей среды и использовать их для своей работы.

Система управления мобильным роботом – интеллектуальное звено, соединяющее «чувства» и действия.

Мобильные роботы подразделяются на три основных класса:

- беспилотные летательные аппараты;
- безэкипажные наземные роботы;
- необитаемые подводные аппараты.

Назначение робота определяет выбор набора сенсоров, которыми снабжается робот. Выбранный набор сенсоров впоследствии играет существенную роль при выборе системы управления.

Состояние робота – набор определенных параметров робота (обусловленных внешними и внутренними факторами), однозначно определяющих его поведение и способ функционирования в данный момент времени. Все множество состояний называется пространством состояний [7]. Состояния подразделяются на следующие три группы:

- наблюдаемые (робот знает свое состояние все время);
- скрытые (робот не знает своего состояния);
- частично наблюдаемые (робот знает свое состояние лишь частично).

Степень интеллектуальности робота напрямую зависит от того, насколько быстро и точно он может определять внутренние и внешние состояния системы.

Отметим, что внутреннее состояние может быть использовано для запоминания информации о мире (целеполагание, карта местности и т.д.). Таким образом, внутреннее состояние также включает в себя внутреннюю модель мира, от которой, по сути, и зависит то, насколько сложной может быть система управления роботом.

Робот оказывает воздействия на внешний мир при помощи эффекторов, действия которых подразделяются на два типа: локомоции (перемещение робота) и манипуляции (перемещение других предметов роботом) [16].

Поведение робота – определенный набор выходных воздействий на внешний мир в ответ на входные воздействия.

Система управления роботом предназначена для стабилизации, слежения и решения других задач управления процессами функционирования робота, что предусматривает поддержание желаемых законов изменения регулируемых переменных или переменных состояния с заданными показателями качества.

Имеет место классификация стратегий и принципов управления в зависимости от структуры связей [7, 8]:

- разомкнутое управление;
- замкнутое управление;
- комбинированное управление;

Разомкнутое управление вводит в состав системы контур прямой связи по задающему воздействию:

$$u = U(y^*),$$

где $U(\cdot)$ – функциональный оператор, который должен выбираться из условия получения заданного закона изменения выходной переменной.

Замкнутое управление (или управление по отклонению) вводит в структуру системы контур обратной связи:

$$u = K(y^* - y),$$

где оператор $K(\cdot)$ выбирается из условия уменьшения отклонения в процессе работы системы. Так как в данном случае поведение объекта управления корректируется в зависимости от текущего значения отклонения, то управление по отклонению обеспечивает устойчивость системы и уменьшение влияния возмущений.

Абсолютная точность решения задачи управления может быть достигнута при помощи комбинированного управления, предусматривающего использование как прямых, так и обратных связей: $u = U(y^*) + K(y^* - y)$.

Помимо приведенной классификации, системы управления роботами могут быть разделены на следующие категории [8]:

- реактивное управление;
- иерархическое управление;
- гибридное управление;
- управление на основе поведения.

Реактивное управление – такой тип управления, который тесно связывает входные и выходные воздействия для получения быстрого отклика системы управления в быстроизменяющемся и неструктурированном мире [8, 54]. Данный тип управления основан на цикле «sense-act» (рис. 26).

Рис. 26. Реактивное управление

При таком типе управления не используется модель. Кроме того, для выработки управляющих воздействий не осуществляется алгоритмический вывод и не производится поиск. В процессе эксплуатации подобного контроллера решающую роль в выработке выходного воздействия играет обратная связь от среды. Поведение, проявляющееся в результате взаимодействия такой системы управления и среды, часто называют эмерджентным поведением (поведение, которое не планируется, а обусловлено ситуацией). Отметим, что многие из животных в большой степени реактивны.

Недостатками данного метода являются:

- как правило, минимальный набор состояний;
- отсутствие памяти;
- отсутствие обучения;
- отсутствие внутренних моделей мира;
- невозможность планирования.

Иерархическое управление – такой тип управления, который действует через цикл «sense-plan-act» (рис. 27) – в отличие от реактивного управления имеет место этап планирования.

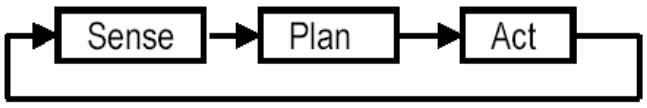


Рис. 27. Цикл работы системы иерархического управления

К недостаткам данного метода можно отнести:

- планирование требует поиска решений, который происходит медленно;
- поиск требует модели внешнего мира, которую достаточно сложно получить;
- модель внешнего мира быстро устаревает.

Гибридные системы – системы управления, которые являются комбинацией двух рассмотренных выше типов [8, 95, 31].

Исследования в области искусственного интеллекта были традиционно основаны на предположении, что робот в ходе решения какой-либо задачи должен собрать информацию о среде, построить на основе этой информации модель (репрезентацию) среды, затем разработать план своих будущих действий на основе этой модели и, наконец, приступить к исполнению плана. Соответственно, архитектура агента должна состоять из иерархии модулей, соответствующих перечисленным этапам (рис. 28).

Рис. 28. Многоуровневая система управления

Альтернативный подход был предложен во второй половине 80-х годов и состоит в том, чтобы имитировать те способы принятия решений, которые предположительно используются животными в их естественной среде. Было постулировано, что агент должен состоять из отдельных модулей, каждый из которых независимо управляет отдельной формой поведения без какого-либо моделирования среды или планирования действий: действия запускаются в ответ на внешние сигналы или даже просто спонтанно. Эти элементарные действия могут представлять собой ненаправленное блуждание, движение к цели, поворот в сторону от препятствия, схватывание какого-либо объекта и т.п. Результирующее адаптивное поведение создается в этом случае конкуренцией модулей (рис. 29).

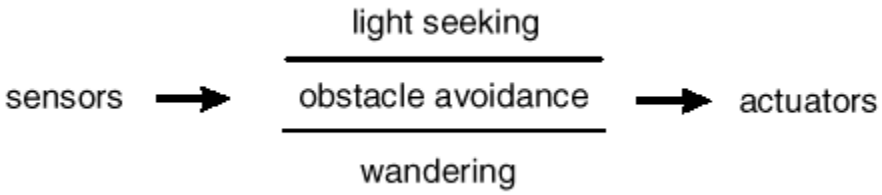


Рис. 29. Система управления на основе поведения

Один из вариантов этой схемы, так называемая «поглощающая архитектура», предполагает, что модули не равноправны: одни из них перекрывают модулям низшего ранга доступ к эффекторам, если получают соответствующий сигнал (рис. 30).

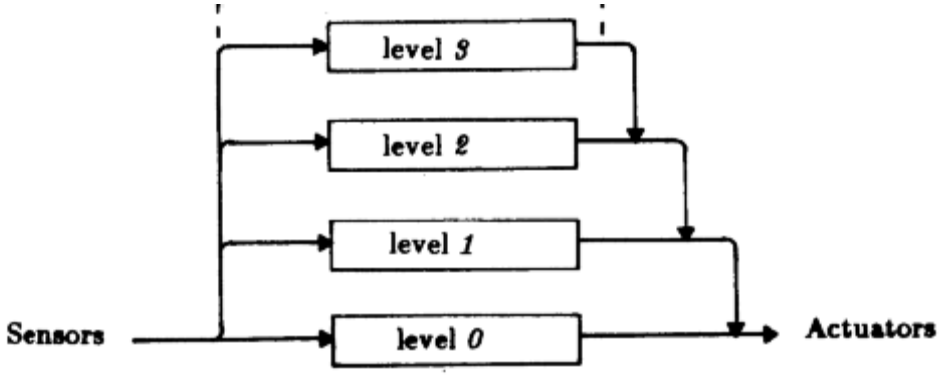


Рис. 30. Поглощающая архитектура

Например, пока агент не воспринимает каких-либо специфических сигналов, он может блуждать по местности для того, чтобы обнаружить заданную цель. Когда цель обнаружена, становится активным модуль движения к цели, который блокирует доступ к эффекторам модулям ненаправленного движения. Если на пути к цели обнаружено препятствие, то активным становится модуль, задающий поворот и движение в сторону, причем этот модуль подавляет все остальные модули. Когда агент оказывается в стороне от препятствия, снова активизируется модуль движения к цели.

Несмотря на крайнюю простоту такого рода схем, они дали хорошие результаты при создании роботов, способных к адаптивному поведению – способных выполнять осмысленные задачи, несмотря на препятствия [33]. Так, эксперименты с навигацией роботов на пересеченной местности показали, что «поглощающая архитектура» действительно способна обеспечить достижение цели при полном отсутствии модели внешней среды и плана действий.

В табл. 17 приводится обзорное сравнение рассмотренных методов.

Таблица 17. Сравнение способов построения систем управления

Метод	Наличие внутренней модели мира	Скорость реакции на входное воздействие	Адаптивность
Реактивное управление	Отсутствует	Быстрая	Отсутствует
Иерархическое управление	Присутствует	Медленная	Зависит от устройства модели мира
Гибридное управление	Присутствует, но может быть слабой	Зависит от входного воздействия	Зависит от устройства модели мира
Управление на основе поведения	Неявно выраженная	Средняя	Присутствует

1.6. Управление человекоподобным роботом

1.6.1. Общие сведения

Создание человекоподобных роботов – одно из наиболее актуальных и перспективных направлений науки. Роботы такого типа могут эффективно выполнять различные задачи в мире, приспособленном для человека.

Написание управляющей программы для человекоподобного робота – трудоемкая задача, поскольку роботу необходимо воспринимать большой объем информации и выполнять сложные и точные движения. Условно в управляющей программе робота можно выделить несколько модулей (рис. 31):

- зрительная система (получение данных от сенсоров робота, обработка изображения);
- локализация (определение положения робота в пространстве относительно набора ориентиров);
- модуль принятия решений о дальнейших действиях;
- двигательная система.

Рис. 31. Структура управляющей программы

Основой двигательной системы является набор базовых действий: шаг вперед, шаг назад, поворот влево, вправо, подъем из положения лежа на спине и из положения лежа на животе. В то же время задать эти действия весьма непросто.

1.6.2. Способы задания движений

Возможны следующие способы задания движения для последующего выполнения его роботом:

- в виде числовой последовательности;
- в виде набора функций.

1.6.2.1. Задание движения в виде числовой последовательности

При описании движения для робота часто используется следующий прием.

Движение разбивается на некоторое число фаз. Каждая фаза характеризуется временем ее начала относительно начала движения и углами для всех сервомоторов, участвующих в движении. Шаг по времени может быть произвольным, но, как правило, используются движения с фиксированным шагом по времени (одинаковым для всех фаз). В случае программирования модели робота с использованием виртуальной среды, шаг по времени часто принимается равным шагу моделирования. Полное движение формируется путем установки сервомоторов робота в указанные для них положения в соответствующие моменты времени.

В случае задания движения таким способом, удобно записывать данные в виде таблицы (табл. 18).

Таблица 18. Пример задания движения в виде числовой последовательности

Время, мс	LHipRoll	LHipPitch	LKneePitch
0	0.027	-0.505	1.042
40	0	-0.524	1.047
80	0	-0.524	1.047
120	0	-0.523	1.047
160	-0.001	-0.523	1.047
200	-0.003	-0.523	1.047
240	-0.004	-0.523	1.047
280	-0.007	-0.522	1.047
320	-0.01	-0.522	1.047
360	-0.016	-0.521	1.046
400	-0.021	-0.52	1.046
440	-0.027	-0.518	1.045
480	-0.034	-0.517	1.044
520	-0.049	-0.513	1.041
560	-0.063	-0.509	1.038
600	-0.09	-0.5	1.03

1.6.2.2. Задание движения в виде набора функций

Пусть в движении участвует *n* сервомоторов. Сопоставим каждому из них целочисленный индекс из промежутка $[0, n-1]$. Очевидно, что значение угла каждого сервомотора представимо в виде функции от времени $\alpha_i = F_i(t)$, где $i \in \overline{0, n-1}$, время t задано относительно времени начала движения.

В случае задания движения в виде числовой последовательности, в файл движения записываются непосредственно значения функций, а исходная функция нигде не фигурирует. Более того, возможно, что и при построении числовой последовательности функция не использовалась в явном виде, а последовательность была получена каким-то другим способом [115, 117].

Тем не менее, часто функции выводятся в явном виде или генерируются управляющей программой в режиме online [33, 71]. В таком случае и сама числовая последовательность может генерироваться «на лету». Часто это дает дополнительную гибкость в управлении движением. Например, появляется возможность изменять скорость ходьбы.

1.6.2.3. Методы, основанные на построении траекторий точек

Широкое распространение получили методы, основанные на построении траекторий определенных точек робота *(trajectory tracking methods)*. В качестве таких точек обычно выступают «суставы» робота или точка нулевого момента *(zero moment point)*. Траектории строятся заранее (offline) путем решения уравнений движения.

Точка нулевого момента – точка на поверхности земли, над которой сумма моментов всех действующих на робота сил равна нулю. Если точка нулевого момента попадает в выпуклую оболочку точек соприкосновения стоп робота с полом *(support polygon)*, то положение робота динамически стабильно [106].

Такой подход обладает следующими недостатками:

- построение движения невозможно без точной физической модели робота;

- решение уравнений движения для роботов с большим числом степеней свободы процесс трудоемкий (даже с применением численных методов решения);
- готовые движения жестко заданы, в ходе выполнения повлиять на них никак нельзя (отсутствует обратная связь).

Подход, основанный на построении траекторий, используется, например, в работе [41].

Отметим, что критерий стабильности робота во время движения, использующий положение точки нулевого момента, широко применяется не только в методах данной категории, но и во многих других методах, использующих оптимизацию.

1.6.2.4. Методы, основанные на анализе движения человека

Один из наиболее простых для понимания (но не для реализации) является подход, использующий данные о движениях людей в качестве отправной точки построения движений для роботов.

Данные об относительном положении различных частей тела человека во время движения могут быть записаны с помощью специальных устройств. После некоторой предварительной обработки, они могут быть преобразованы в формат, пригодный для робота. Затем производится попытка «улучшения» движения одним из методов оптимизации, например, при помощи генетических алгоритмов [58].

К недостаткам подхода следует отнести сложность в реализации, связанную с необходимостью использования специальных устройств. Кроме того, при использовании оптимизационных методов возникают трудности с оценкой качества движений.

1.6.2.5. Методы, использующие центральные генераторы паттернов

Многие виды движений животных содержат в своей основе циклические процессы. В том числе при ходьбе человека наблюдается повторяющаяся смена положений рук и ног. Поэтому для моделирования такого рода движений используют осцилляторные модели нейронных сетей.

Осцилляторные нейронные сети (*ОНС*) – нейронные сети, основными структурными единицами которых являются осцилляторы. Функционируют ОНС за счет колебаний отдельных элементов или групп элементов и их взаимодействия.

Центральным генератором паттернов (ЦГП), central pattern generators, называют осцилляторную нейронную сеть, которая способна демонстрировать различные наборы динамических паттернов активности, соответствующих тем или иным типам движений.

Использование ЦГП является широко распространенным подходом к построению движений для роботов. При этом многое зависит от выбора модели ЦГП. При этом можно добиться того, что построенный метод не будет нуждаться в динамической модели робота или окружающей среды и, в отличии от методов предыдущей группы, появится возможность корректировки движения во время его выполнения.

Двигательную активность животных контролируют схожие механизмы [1].

1.7. Мультиагентные системы

В настоящем подразделе описывается применение мультиагентного подхода в разработке систем управления.

Вычислительные мультиагентные модели применяются для имитации поведения, действий и взаимодействий автономных индивидуумов (агентов), объединенных в систему. Целью моделирования является оценка влияния поведения каждого агента на систему в целом. Имитация действий каждого агента позволяет воспроизводить и предсказывать поведение системы.

Предполагается, что каждый агент действует исключительно в своих интересах, стремится максимизировать свою функцию выигрыша (utility function). Это может проявляться в стремлении

повысить свою экономическую прибыль (в экономических моделях), или свой социальный статус (при социологическом моделировании).

Каждый агент обладает ограниченными знаниями об окружающей среде. В процессе моделирования агенты могут накапливать опыт, обучаться и самовоспроизводиться.

В основе агент-ориентированных моделей (АОМ) лежат три основные идеи:

- объектная ориентированность;
- простое поведение агентов;
- обучаемость агентов (или их эволюция);

Несмотря на простоту поведения каждого моделируемого агента, вся система может демонстрировать весьма сложное поведение.

Мультиагентные системы состоят из:

- динамически взаимодействующих по определенным правилам агентов;
- среды, в которой агенты взаимодействуют (которая может быть достаточно сложной);

Основные свойства агентов АОМ:

- интеллектуальность (это свойство должно быть ограниченным);
- наличие жизненной цели (функции выигрыша, utility function);
- расположение во времени и пространстве.

В последнем пункте имеется ввиду некоторая «среда обитания», которая может быть представлена как в виде решетки (например, игра «Жизнь»), так и в виде гораздо более сложной структуры. Результатом взаимодействия агентов в «среде обитания» может быть:

- равновесие;
- непрекращающийся процесс эволюции;
- бесконечный цикл без определенного результата или прогресса.

Считается, что мультиагентное моделирование дополняют традиционные аналитические методы. Последние позволяют охарактеризовать состояние равновесия системы, в то время как моделирование дает возможность исследования способов получения такого состояния.

Принято считать, что мультиагентные модели берут свое начало от вычислительных машин Джон фон Неймана (Von Neumann), являющихся теоретическими машинами, способными к воспроизведению. Фон Нейман предложил использовать машины, которые следуют детальным инструкциям для создания точных копий самих себя [91].

Впоследствии подход был усовершенствован другом фон Неймана – Станиславом Уламом, который предложил изображать машину на бумаге в качестве набора клеток на решетке. Данный подход стал началом развития клеточных автоматов. Наиболее известной реализацией конечного автомата стала игра «Жизнь», предложенная Джоном Хортоном Конвеем (John Horton Conway) в работе [35], отличающаяся от машины фон Неймана крайне простыми правилами поведения агентов.

Использование мультиагентного моделирования для социальных систем берет свое начало от программиста Крега Рейнолдса (Craig Reynolds). Им была предпринята попытка моделирования деятельности живых биологических агентов (модель «Artificial life» – «Искусственная жизнь»).

С середины 1990-х годов, мультиагентные системы стали использоваться для решения множества коммерческих и технологических проблем. Примерами могут послужить задачи:

- оптимизации сети поставщиков и логистика;
- моделирования потребительского поведения (в том числе социальные сети);
- распределенных вычислений;
- менеджмента трудовых ресурсов;
- управления транспортом;
- управления инвестиционными портфелями.

В этих и других приложениях стратегии поведения определяются с учетом поведения множества индивидуальных агентов-атомов и их взаимодействий. Таким образом, мультиагентные модели могут помочь в изучении влияния индивидуального поведения агентов на эволюцию всей системы.

Мультиагентное моделирование может помочь в объяснении причин возникновения таких явлений как террористические организации, войны, обрушения рынка акций и т. д. В идентификации критических моментов времени, после наступления которых чрезвычайные последствия будут иметь необратимый характер.

Существуют исследования, устанавливающие связи между использованием нечеткой логики и мультиагентным подходом. Часто встречается совместное использование нечеткой логики и теории автоматических переговоров. Так, например, в работе [109] описан способ реализации программных агентов, способных вести торги на онлайн-рынке. Модель переговоров агентов основана на использовании нечеткой логики. По мнению авторов, это позволяет одновременно снизить сложность переговорного процесса и работать с нечеткой и рассеянной информацией, которую можно найти в интернете.

В работе [42] предложен еще один вариант применения технологии нечетких вычислений и программных агентов в экономике: в рамках этих исследований была построена интеллектуальная система управления большим объемом инвестиций (giga-investments). Агенты в этой системе делятся на две основные группы. К первой относятся агенты, ответственные за сканирование внешних данных (внешние базы данных, интернет), выделение актуальной информации, и ее преобразование во внутреннее представление. Ко второй – ответственные за анализ внутренних данных, общение с пользователем и предпринимаемые действия. Все перечисленные агенты руководствуются нечеткими инструкциями.

Совместное использование нечеткой логики и программных агентов также находит широкое применение в семантическом вебе. Например, в работе [45] описана рекомендательная система, использующая агентов с нечетким поведением. Система состоит из агентов пользовательского интерфейса, фильтрации, заданий, пользовательского профиля и рекомендующего агента. Агенты проводят переговоры, осуществляют поиск потенциально интересных пользователю документов и реализуют обратную связь (обновляют данные в профиле пользователя, исходя из высказанных им предпочтений).

В табл. 19 приводится обзорное сравнение указанных работ.

Таблица 19. Сравнение работ по мультиагентным системам

Название работы	Решаемая задача	Технологии
Based Intelligent Negotiation Agent (FINA) in Ecommerce	Ведение торгов на онлайн бирже	Автоматические переговоры агентов
Fuzzy Logic and Intelligent Agents: Towards the Next Step of Capital Budgeting Decision Support	Задача управления большим объемом инвестиций	Сбор информации в web, автоматические переговоры, нечеткая логика
Gathering information on the Web using fuzzy linguistic agents and Semantic Web technologies	Построение рекомендующей системы	Сбор информации в web, автоматические переговоры, нечеткая логика, машинное обучение

Выводы по главе 1

1. Искусственный интеллект в настоящее время включает такие разделы как мультиагентные системы, обучение с подкреплением, искусственные нейронные сети, робототехника, методы поиска при условиях ограничений, представление знаний, логический вывод, планирование действий, методы вероятностных рассуждений и рассуждений в условиях неопределенности, в том числе нечеткие множества и нечеткая логика, обработка естественного языка, машинное зрение.
2. Методы поиска в условиях ограничений находят свое применение при разработке программных систем при решении задач генерации тестов, разбиения на модули, тестировании времени отклика. Наиболее важными компонентами таких методов являются метод представления решений и функция приспособленности.
3. На примере задачи о флибах и дилеммы узника показано, что генетическое программирование более эффективно для построения управляющих систем на основе конечных автоматов, чем генетические алгоритмы.
4. Одной из важных областей применения методов искусственного интеллекта является построения программных систем для мобильных роботов, в том числе человекоподобных.
5. При построении управляющих программных систем могут также применяться другие методы искусственного интеллекта: искусственные нейронные сети, нечеткая логика, мультиагентные системы.

2. ВЫБОР И ОБОСНОВАНИЕ ОПТИМАЛЬНОГО ВАРИАНТА НАПРАВЛЕНИЯ ИССЛЕДОВАНИЙ

В настоящем разделе представлено обоснование оптимального варианта проведения исследований.

На основании результатов выполненного аналитического обзора предлагается следующий вариант направления исследований: методы искусственного интеллекта будут применяться для автоматизации процесса построения управляющих программных систем на основе конечных автоматов. При этом особое внимание будет уделено построению систем управления для мобильных роботов. Будет также рассмотрено применение метода коэволюции для построения систем управляющих конечных автоматов, так как для ряда задач целесообразным является построение системы конечных автоматов. Кроме того, этот метод может применяться для построения мультиагентных систем, в которых каждый из агентов представлен конечным автоматом.

На этапе теоретических исследований будет разработан метод применения генетических алгоритмов для построения системы управления виртуальным человекоподобным роботом (на примере управления ходьбой), метод применения генетического программирования для построения систем управления виртуальными роботами для упрощенной модели игры в футбол, метод применения коэволюции для построения взаимодействующих управляющих конечных автоматов, метод совместного применения конечных автоматов и нейронных сетей, метод применения верификации моделей программ при построении управляющих конечных автоматов с помощью генетического программирования.

Экспериментальное исследование разработанных методов будет проводиться сначала на программных моделях, а затем – непосредственно на малоразмерном мобильном роботе. На основании результатов экспериментов будут определены достоинства и недостатки разработанных методов.

На основании результатов теоретических и экспериментальных исследований также будет разработана виртуальная лаборатория обучения генетическим алгоритмам для языков программирования *Java*, *C++*, C#. С помощью этой виртуальной лаборатории будут выполнены не менее 20 студенческих лабораторных работ, отчеты по которым будут опубликованы в сети Интернет.

По итогам работ будут зарегистрированы пять программ для ЭВМ и опубликованы пять статьей в журналах из перечня ВАК.

Выводы по главе 2

Выбран и обоснован оптимальный вариант направления исследований.

3. ПЛАН ПРОВЕДЕНИЯ ТЕОРЕТИЧЕСКИХ И ЭКСПЕРИМЕНТАЛЬНЫХ ИССЛЕДОВАНИЙ

В настоящем разделе представлен план проведения теоретических и экспериментальных исследований.

3.1. План проведения первого этапа теоретических и экспериментальных исследований

На первом этапе проведения исследований планируется провести следующие работы:

- разработка, программная реализация и экспериментальное исследование метода применения генетических алгоритмов для построения системы управления виртуальным человекоподобным роботом (на примере управления ходьбой) (20.11.2009 – 25.11.2009):
  - разработка структуры хромосом;
  - разработка алгоритма скрещивания хромосом;
  - разработка метода вычисления функции приспособленности;
  - апробация метода на примере построения системы управления ходьбой человекоподобного робота.

По итогам первого этапа планируется подготовить и подать заявку на регистрацию программы для ЭВМ.

Результаты работ первого этапа:

- метод применения генетических алгоритмов для построения системы управления виртуальным человекоподобным роботом;
- свидетельство о регистрации программы для ЭВМ;
- научно-технический отчет.

3.2. План проведения второго этапа теоретических и экспериментальных исследований

На втором этапе проведения исследований планируется провести следующие работы:

- разработка, программная реализация и экспериментальное исследование метода применения генетического программирования для построения систем управления виртуальными роботами для упрощенной модели игры в футбол (01.01.2010 – 28.02.2010):
  - разработка структуры хромосом алгоритма генетического программирования;
  - разработка алгоритма скрещивания хромосом;
  - разработка метода вычисления функции приспособленности;
  - апробация метода на примере построения системы управления роботом для игры «Виртуальный футбол».
- разработка, программная реализация и экспериментальное исследование метода применения коэволюции для построения взаимодействующих управляющих конечных автоматов (01.03.2010 – 31.03.2010);
- разработка, программная реализация и экспериментальное исследование метода совместного применения конечных автоматов и нейронных сетей (01.05.2010 – 30.06.2010);
- разработка метода применения верификации моделей программ при построении управляющих конечных автоматов с помощью генетического программирования (01.05.2010 – 30.06.2010);

По итогам второго этапа работ планируется подготовить для публикации три статьи в журналах из перечня ВАК, а также подготовить и подать две заявки на регистрацию программы для ЭВМ.

Результаты работ второго этапа:

- метод применения генетического программирования для построения систем управления виртуальными роботами для упрощенной модели игры в футбол;
- метод применения коэволюции для построения взаимодействующих управляющих конечных автоматов;
- метод совместного применения конечных автоматов и нейронных сетей;
- метод применения верификации моделей программ при построении управляющих конечных автоматов с помощью генетического программирования;
- три статьи в журналах из перечня ВАК;
- два свидетельства о регистрации программы для ЭВМ;
- научно-технический отчет.

3.3. План проведения третьего этапа теоретических и экспериментальных исследований

На третьем этапе проведения исследований планируется провести следующие работы:

- разработка программы, позволяющей увеличить объем знаний для более глубокого понимания изучаемого предмета исследования и пути применения новых явлений, механизмов или закономерностей (виртуальная лаборатория для обучения генетическим алгоритмам для языков программирования *Java, C++, C#*) (01.01.2011 – 31.01.2011);
- экспериментальные исследования модифицированных методов на примере построения системы управления программной моделью малоразмерного мобильного робота (16.02.2011 – 20.03.2011);
- экспериментальные исследования модифицированных методов на примере построения системы управления малоразмерным мобильным роботом (21.03.2011 – 30.04.2011);
- обобщение и оценка результатов исследований – оценка полноты решения задачи и достижения поставленных целей (01.06.2011 – 09.07.2011);
- обобщение и оценка результатов исследований – сопоставление и обобщение результатов анализа научно-информационных источников и теоретических и экспериментальных исследований (01.06.2011 – 09.07.2011);
- обобщение и оценка результатов исследований – оценка эффективности полученных результатов в сравнении с современным научно-техническим уровнем (01.06.2011 – 09.07.2011);
- разработка рекомендаций по использованию результатов НИР при создании научно-образовательных курсов (01.06.2011 – 09.07.2011).

По итогам третьего этапа работ планируется подготовить для публикации две статьи в журналах из перечня ВАК и подать две заявки на регистрацию программы для ЭВМ.

Результаты работ третьего этапа:

- виртуальная лаборатория для обучения генетическим алгоритмам для языков программирования *Java, C++, C#*;
- две статьи в журналах из перечня ВАК, содержащие основные результаты НИР;
- два свидетельства о регистрации программы для ЭВМ;
- рекомендации по использованию результатов НИР при создании научно-образовательных курсов;

- протоколы экспериментов;
- научно-технический отчет.

Выводы по главе 3

1. Разработан план проведения теоретических и экспериментальных исследований.
2. На первом этапе теоретических исследований будет проведена разработка метода применения генетических алгоритмов для построения системы управления виртуальным человекоподобным роботом (на примере управления ходьбой).

4. РАЗРАБОТКА, ПРОГРАММНАЯ РЕАЛИЗАЦИЯ И ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ МЕТОДА ПРИМЕНЕНИЯ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ ПОСТРОЕНИЯ СИСТЕМЫ УПРАВЛЕНИЯ ВИРТУАЛЬНЫМ ЧЕЛОВЕКОПОДОБНЫМ РОБОТОМ (НА ПРИМЕРЕ УПРАВЛЕНИЯ ХОДЬБОЙ)

Данный раздел посвящен рассмотрению вопроса применения генетических алгоритмов для генерации движений человекоподобных роботов. При этом в качестве примера применения разработанного метода рассматривается задача выполнения роботом шага вперед. В начале раздела приводится обзор используемых инструментов – среды симуляции *Webots* и модели робота *Nao*. Кроме того, приведен обзор существующих методов генерации движений и возможных способов задания движений.

Цель данного этапа – установить возможность генерации движений на основании информации, предоставляемой только датчиками самого робота, причем такими, какими оснащены практически все роботы данного типа, без использования данных о динамической модели робота. Такой метод, в случае если он будет получен, может быть применен и для других человекоподобных роботов.

В заключение раздела анализируются полученные результаты, и приводится ряд открытых вопросов и направлений для дальнейших исследований в этой области.

4.1. СРЕДА СИМУЛЯЦИИ *WEBOTS*

Швейцарская компания *Cyberbotics Ltd*. разработала *Webots* – среду для моделирования, программирования и симуляции мобильных роботов [81]. Поставляемые вместе с *Webots* библиотеки позволяют запускать написанные в симуляторе управляющие программы для исполнения на реальных роботах.

Несмотря на то, что конечной целью большинства исследований в робототехнике являются настоящие роботы, на практике часто возникает потребность в симуляторе, с помощью которого можно быстро проверить новые идеи. Моделирование поведения робота в симуляторе часто выполняется быстрее, следить за нужными параметрами на нем удобнее, а сам симулятор обходится гораздо дешевле настоящего робота.

На рис. 32 показаны основные этапы разработки управляющей программы для робота с помощью симулятора *Webots*.

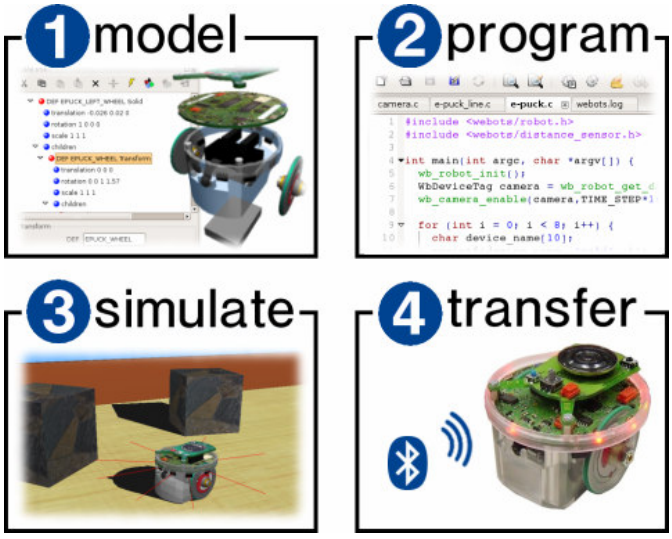


Рис. 32. Этапы разработки управляющей программы для реального робота в *Webots*

На первом этапе необходимо создать модель самого робота и окружающую его среду. Для создания модели робота предусмотрен набор различных настраиваемых устройств (видеокамеры, датчики расстояния, света, прикосновения, *GPS*, сервомоторы, эмиттеры, ресиверы, светодиоды и др.). Создание окружающей обстановки включает в себя создание сцены из набора стандартных примитивов или импорта объектов в формате *VRML97*, настройки освещенности, наложения текстур и дополнительных эффектов. Далее в данном разделе будет использоваться модель робота *Nao* (*NaoV3R*), входящая в поставку *Webots 6.1.2* и стандартная сцена из файла *nao1.wbt*, также поставляемого вместе с симулятором. В состав этой сцены входит один робот, футбольное поле, полностью соответствующее полю для стандартной лиги *RoboCup*, и мяч. Дополнительно к стандартной сцене добавлен супервизор. Внешний вид среды *Webots* с открытым в нем файлом сцены показан на рис. 33.

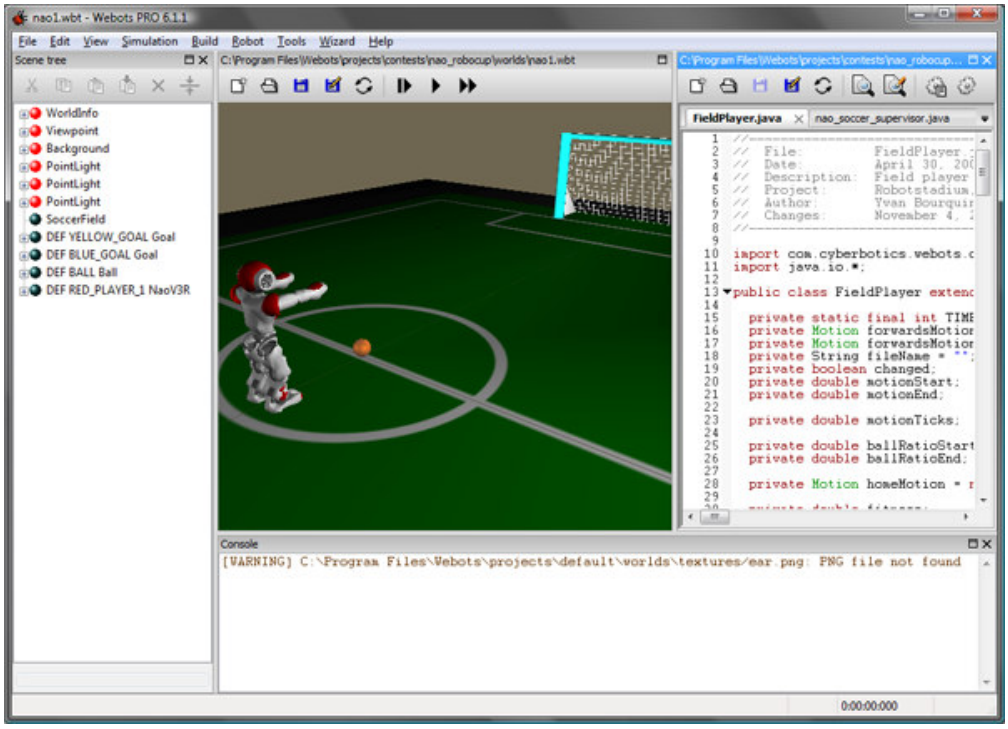


Рис. 33. Внешний вид среды симуляции *Webots*

Управляющая программа для робота (контроллер) может быть написана на языках *C/C++*, *Java, Python* или *URBI*. Кроме того, через *TCP/IP* из кода контроллера можно подключаться и использовать функциональность программ сторонних разработчиков (например, *MatLab* или *LabView*).

Часто возникает необходимость в контроле над процессом моделирования, отслеживании определенных параметров, перемещении объектов в пределах сцены и т.п. Для решения таких задач на сцену может быть добавлен специальный объект – супервизор. Супервизор представляет собой программу, написанную на одном из перечисленных выше языков программирования, и управляющую ходом моделирования.

Подсистема симуляции *Webots* использует модель виртуального времени, что позволяет запускать симуляцию в быстром режиме. Максимальная скорость моделирования зависит от сложности сцены и мощности компьютера, и для сцены, используемой в данной работе, при запуске симуляции на компьютере с процессором *Intel Core 2 Quad Q9300* достигает 1:11.

4.2. Модель робота *Nao*

Модель *Nao*, используемая в *Webots*, оснащена 22 сервомоторами. Их расположение показано на рис. 34 (сервомоторы *RWristYaw*, *LWristYaw*, *RHand* и *LHand* в симуляторе не используются).

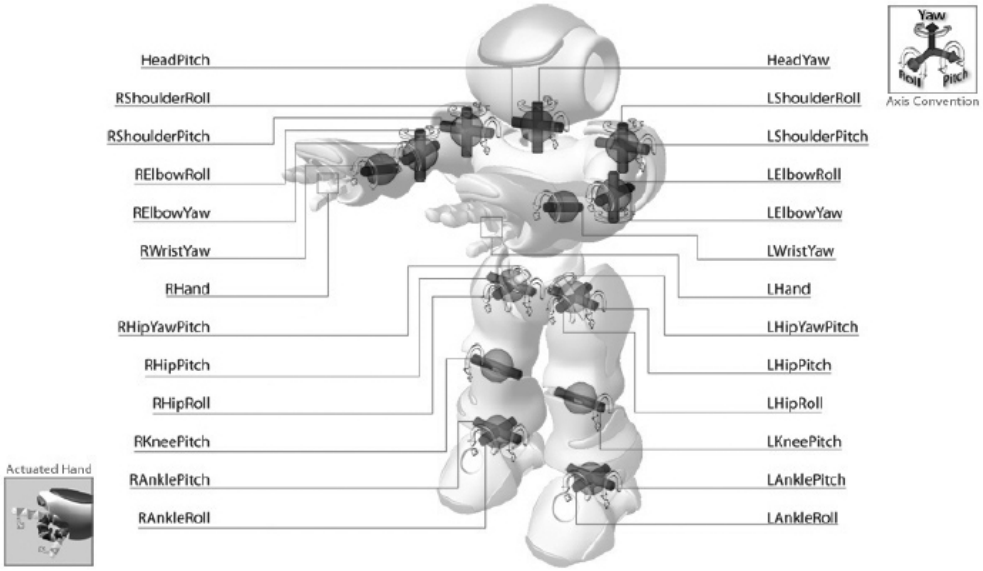


Рис. 34. Расположение сервомоторов робота

Для анализа информации из окружающей среды робот оснащен следующими датчиками:

- видеокамера;
- акселерометр;
- гироскоп;
- датчики давления на ногах *(Force Sensitive Resistors)*;
- датчики соприкосновения с полом на ногах (*Foot Bumpers*, по два на каждой ноге робота);
- ультразвуковые датчики для определения расстояния до ближайшего препятствия;
- эмиттер;
- ресивер.

4.2.1. Видеокамера

Модель робота оснащена одной видеокамерой (в отличие от настоящего робота, обладающего двумя видеокамерами) и позволяет получать картинку в виде массива из 160×120 *RGB*-значений. Кроме того, для нее можно задать одно из двух положений – верхнее или нижнее, что позволяет имитировать наличие двух видеокамер, так как у настоящего робота *Nao* они расположены одна под другой.

4.2.2. Акселерометр

Трехосевой акселерометр позволяет получить значения ускорения тела робота по трем осям и может быть использован для определения падения робота и его положения после падения («на спине» или «на животе»).

4.2.3. Гироскоп

Гироскоп определяет угловую скорость (в рад/с) тела робота относительно осей *OX* и *OY* (центр системы координат совпадает с центром тела робота и находится в районе талии робота, там же расположен и сам гироскоп). Это устройство возвращает вектор из трех вещественных чисел, первые два равны угловой скорости по соответствующей оси и позволяют определить начало падения робота и его направление (вперед-назад или вправо-влево), третье число не используется.

4.2.4. Эмиттер и ресивер

Эмиттер и ресивер – устройства для обмена информацией между роботами. В данной работе эти устройства активно используются для обмена информацией между роботом и супервизором.

4.3. Постановка задачи

Цель исследования – разработка метода генерации движения вперед для модели робота *Nao* с учетом следующих ограничений:

1. Метод должен базироваться только на информации предоставляемой средой симуляции. Это ограничение имеет важное практическое значение. Часто исследователи не могут позволить себе проведение экспериментов на настоящих роботах, так как они весьма дороги. Поэтому логично проводить эксперименты с виртуальной моделью робота, а затем уже адаптировать полученные движения для реального робота.
2. Во время проигрывания полученного движения вперед, робот должен максимально сместиться вперед, минимально отклониться и/или повернуться в сторону.
3. Для упрощения задачи будем считать, что только следующие 10 сервомоторов участвуют в выполнении шага: *RHipPitch, RHipRoll, RKneePitch, RAnklePitch, RAnkleRoll, LHipPitch, LHipRoll, LKneePitch, LAnklePitch, LAnkleRoll* (все они управляют ногами робота, рис. 35).

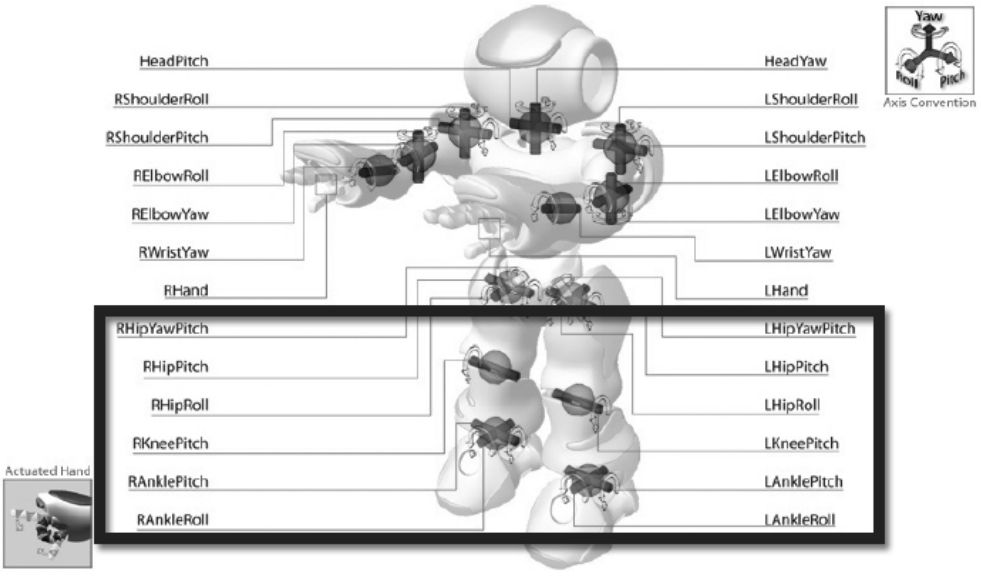


Рис. 35. Сервомоторы робота, участвующие в выполнении движения вперед

В качестве основы для метода генерации движения вперед были выбраны генетические алгоритмы. При этом изучается вопрос выбора параметров генетического алгоритма и зависимость между этими параметрами и качеством получаемых движений.

Вместе с *Webots* распространяется набор примеров движений, среди которых есть и шаг вперед. С ним и будет проведено сравнение полученных при помощи предложенного метода движений.

4.4. Основы предложенного метода

Метод основан на использовании центральных генераторов паттернов. На рис. 36 приведены графики зависимостей углов сгиба суставов от времени при выполнении шага вперед человеком [88].

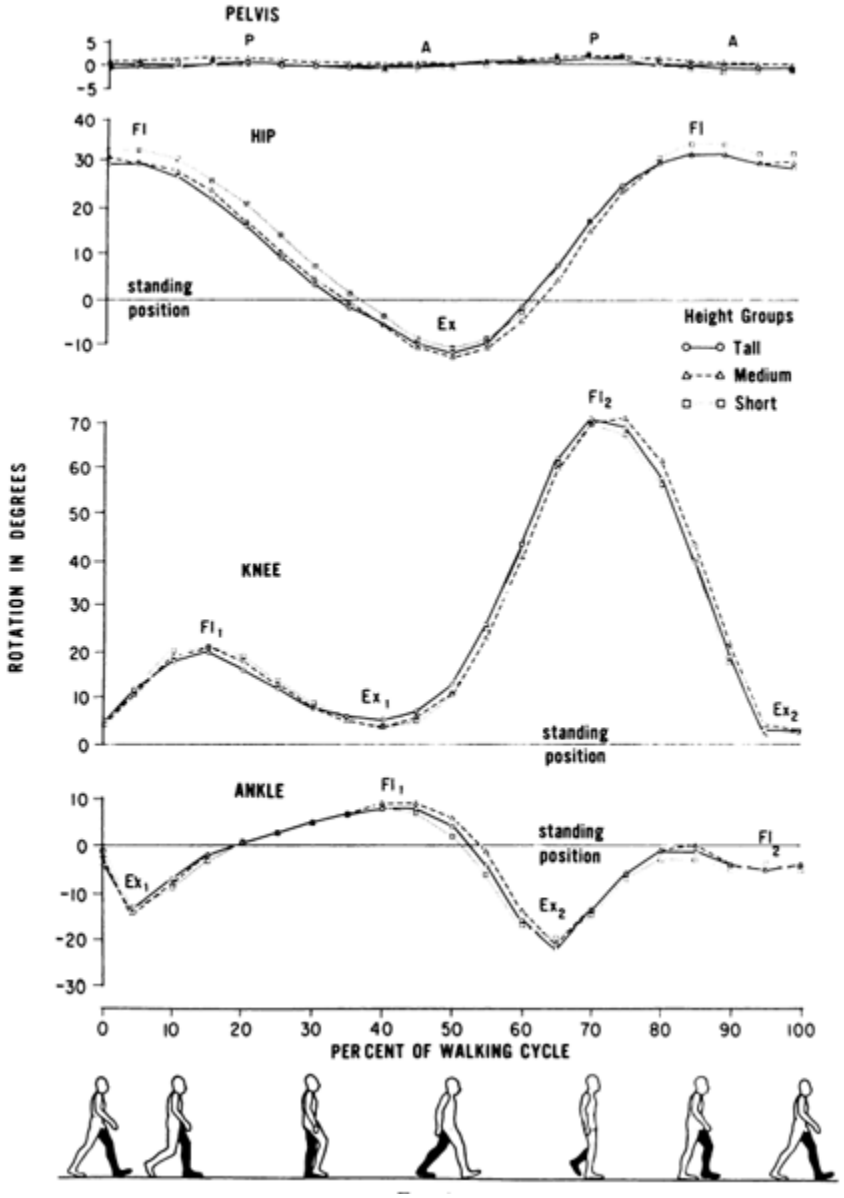


Рис. 36. Выполнение человеком шага вперед

Задача генерации движения вперед эквивалентна задаче поиска аналогичных траекторий для каждого из сервомоторов робота. Чтобы упростить задачу, будем искать траектории углов сервомоторов робота в виде синусоид:

$$\alpha_i(t) = A_i \sin(\omega t) + \alpha_{i0},$$

здесь

- $\alpha_i(t)$ – угол *i*-го сервомотора в момент времени *t*;
- A_i – амплитуда *i*-го сервомотора;
- ω – частота;

- α_{i0} – начальное положение *i*-го сервомотора.

В работах [85, 106] показано, что подход, основанный на использовании синусоид, позволяет получать неплохие результаты для аналогичных роботов. Таким образом, решение поставленной задачи сводится к подбору значений ω, A_i и α_{i0} для всех участвующих в движении сервомоторов. Для того чтобы сузить пространство поиска, введем дополнительные связи между искомыми значениями.

Рассмотрим сначала выполнение движения в дорсовентральной плоскости, и заметим, что во время движения вес тела переносится на опорную ногу (рис. 37).

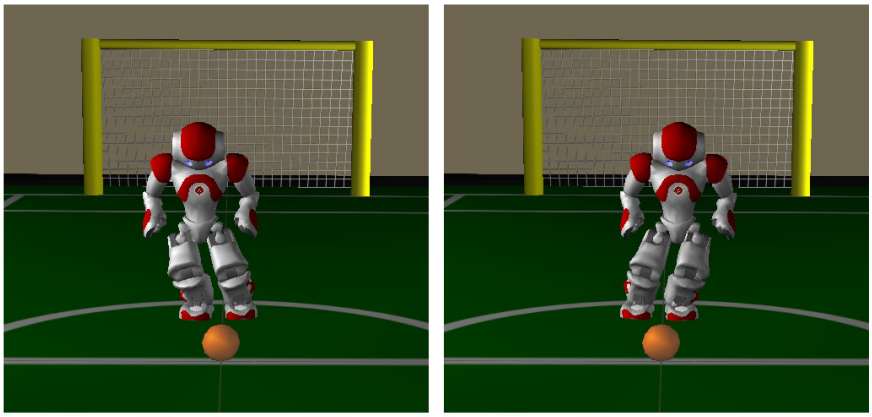


Рис. 37. Перенос веса тела с одной ноги на другую

Подобное движение может быть смоделировано изменением положений сервомоторов *HipRoll* и *AnkleRoll* в соответствии с уравнениями:

$$\alpha_{HipRoll}(t) = A_{HipRoll}\sin(\omega t)$$
$$\alpha_{AnkleRoll}(t) = -A_{AnkleRoll}\sin(\omega t).$$

Почему используется знак «минус» в выражении для сервомотора *AnkleRoll* становится понятно из правой части рис. 38. Выражения для «левых» и «правых» сервомоторов в данном случае эквивалентны.

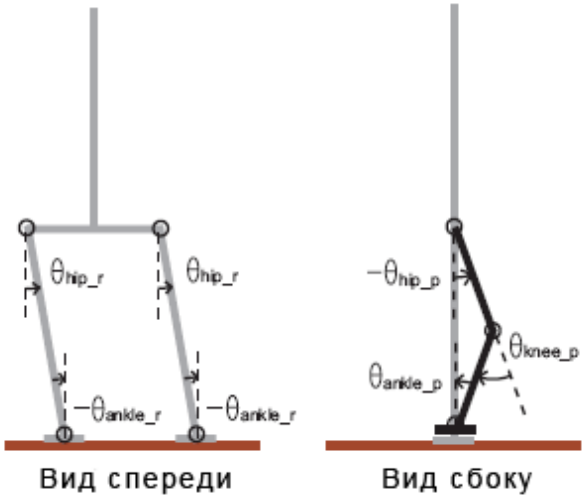


Рис. 38. Положения сервомоторов во время шага вперед

Далее рассмотрим движение вперед в латеральной (боковой) плоскости. В этой плоскости движение осуществляется за счет сервомоторов *HipPitch, KneePitch и AnklePitch*. В работе [106] для них используются выражения:

$$\alpha_{HipPitch}(t) = A_{Pitch}\sin(\omega t) + \alpha_{HipPitch\,0}$$
$$\alpha_{KneePitch}(t) = -2A_{Pitch}\sin(\omega t) + \alpha_{KneePitch\,0}$$
$$\alpha_{AnklePitch}(t) = A_{Pitch}\sin(\omega t) + \alpha_{AnklePitch\,0}$$

В ходе экспериментов было установлено, что такие выражения подходят для использования с описанной моделью робота, но коэффициенты были уточнены. Таким образом, уравнение для *AnklePitch* приобрело вид:

$$\alpha_{AnklePitch}(t) = 0.5A_{Pitch}\sin(\omega t) + \alpha_{AnklePitch\,0}$$

Кроме того, было замечено, что коэффициент при *Apitch* в уравнении для сервомоторов *AnklePitch* слишком велик. Было произведено несколько попыток подобрать его вручную, и оказалось, что для разных наборов *Apitch, AHipRoll* и *AAnkleRoll* оптимальными являются различные его значения из диапазона [1, 2]. Поэтому он был добавлен к параметрам оптимизации:

$$\alpha_{KneePitch}(t) = -kA_{Pitch}\sin(\omega t) + \alpha_{KneePitch\,0}, k \in [1,2]$$

Для того, чтобы обеспечить симметричность движения, «левые» и «правые» сервомоторы, двигающиеся в латеральной плоскости, принимают значения со сдвигом по фазе, равной половине периода.

Несмотря на то, что уже с использованием таких простых выражений для положений сервомоторов, удается получить неплохие результаты, в них были внесены некоторые изменения, полученные эвристически. Было замечено, что при использовании одних только синусоид, робот часто оказывается неспособен полностью оторвать ногу от пола, так как начинает это делать еще до того, как вес тела будет перенесен на другую ногу (рис. 39). В результате робот не шагает в традиционном понимании этого слова, а скорее скользит, что отрицательно сказывается на скорости движения (а иногда и вовсе приводит к падению).

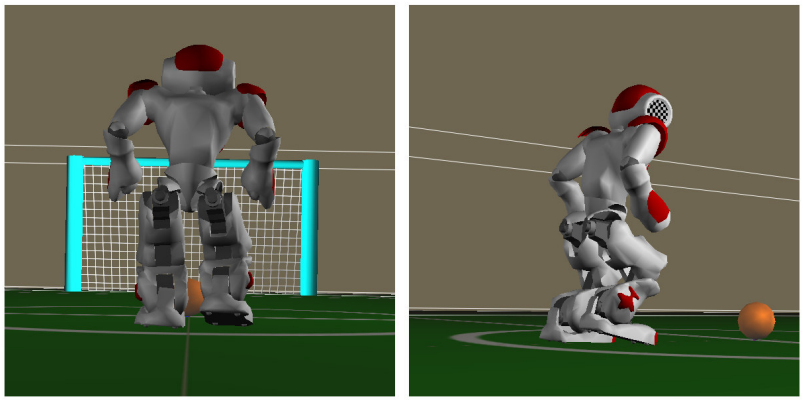


Рис. 39. Вес тела перенесен на опорную ногу, но свободная нога касается пола

Для того чтобы этого избежать, положения сервомоторов *HipPitch* и *AnklePitch* для временных отрезков вида $\left[nT + \phi, T(n + \frac{1}{4}) + \phi\right]$, где $T = \frac{2\pi}{\omega}$, ϕ – сдвиг по фазе для соответствующей ноги

(либо 0, либо $\frac{T}{2}$) принимаются равными $\alpha_{HipPitch0}$ и $\alpha_{AnklePitch0}$ соответственно. Графики соответствующих зависимостей показаны на рис. 40.

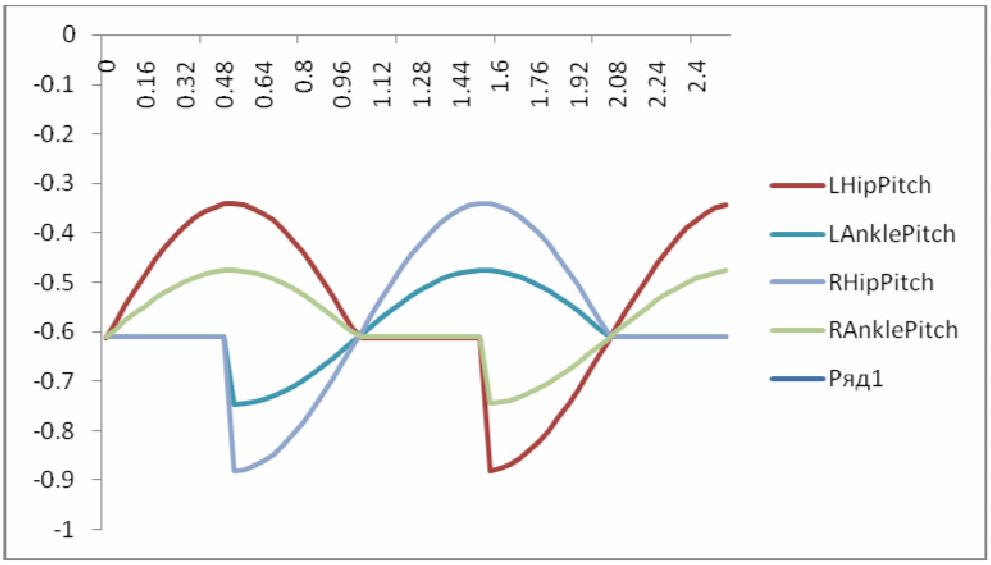


Рис. 40. Графики зависимостей положений сервомоторов *LHipPitch, RHipPitch, LAnklePitch и RAnklePitch* от времени

Положения углов сервомоторов в начальный момент времени были определены вручную. Таким образом, для полного задания движения требуется пять параметров: A_{Pitch}, $A_{AnkleRoll}$, $A_{HipRoll}$, k и ω. Они и будут оптимизироваться средствами генетического программирования.

4.5. Описание генетического алгоритма

Для работы генетического алгоритма необходимо реализовать кодирование особей (движений робота) в хромосомы (числовые наборы), задать генетические операторы над хромосомами выбранного вида и алгоритм вычисления функции-приспособленности.

В качестве реализации генетических алгоритмов было использовано инструментальное средство, описанное в работе [12].

4.5.1. Структура хромосомы

Роль хромосомы в данной работе выполняет вектор из пяти вещественных чисел, соответствующих значениям A_{Pitch}, $A_{AnkleRoll}$, $A_{HipRoll}$, k и ω. Диапазон допустимых значений для параметров оптимизации приведен в табл. 20.

4.5.2. Создание начального поколения

Так как генетические алгоритмы – методы оптимизационные, то возникает необходимость в выборе стартовой точки – начального движения, которое будет оптимизироваться.

В данной работе параметры начального движения были подобраны вручную. Они приведены в табл. 20.

Таблица 20. Параметры начального движения и ограничения на параметры оптимизации

Параметр оптимизации	Минимальное значение	Максимальное значение	Начальное значение
A_{hip_roll}	0	0.3	0.1
A_{ankle_roll}	0	0.3	0.1
A_{hip_pitch}	0	0.4	0.15
K	1	2	1
T	1	6	1

Начальное поколение заполняется N особями, каждая из которых является результатом применения мутации к начальной особи.

4.5.3. Генетические операторы

Оператор скрещивания получает на вход две особи из предыдущего поколения, которые выбираются по принципу «рулетки», и возвращает также две особи. Были использованы два оператора скрещивания: арифметический оператор скрещивания и смешанный оператор скрещивания [59].

Арифметический оператор скрещивания. Пусть хромосомы выбраны оператором селекции для проведения скрещивания. Создаются два потомка: $H_1 = (h_1^1,..,h_n^1)$ и $H_2 = (h_1^2,..,h_n^2)$, такие что $h_i^1 = \eta c_i^1 + (1-\eta)c_i^2$, $h_i^2 = \eta c_i^2 + (1-\eta)c_i^1$, $i = \overline{1,n}$, η – случайное число из интервала *[0,1]*.

Смешанный кроссовер (BLX-alpha crossover). Генерируются два потомка $H_i = (h_1^i,..,h_n^i), i = \overline{1,2},$

$h_k \in [c_{\min} - \alpha I, c_{\max} + \alpha I], c_{\min} = \min(c_k^1, c_k^2), c_{\max} = \max(c_k^1, c_k^2), I = c_{\max} - c_{\min}$.

Мутация. Оператор мутации получает на вход одну особь, выбранную по принципу «рулетки», и меняет один из ее параметров на случайную величину по правилу: $h_i = h_i + \eta(h_{\max} - h_{\min})$, где h_i – выбранный для изменения параметр, η – случайное число из интервала [-0.25, 0.25]. Границы указанного интервала могут являться параметрами алгоритма, но в данной работе они были зафиксированы.

Результатом работы оператора также является одна особь.

4.5.4. Вычисление приспособленности

Для вычисления приспособленности особи движение моделируется в среде *Webots*. Для этого модуль, отвечающий за тестирование особей, записывает движение в файл в виде числовой последовательности. Файл помещается в специальную директорию. В то же время супервизор, запущенный в *Webots*, постоянно сканирует указанную директорию на предмет новых файлов движений. Как только такой файл появляется, робот устанавливается в начальную позицию и с помощью эмиттера ему пересылается сообщение с именем файла.

Робот проигрывает полученное движение до тех пор, пока оно не закончится, или он не упадет. После этого вычисляется приспособленность данного движения. Она записывается в другой файл, который считывается и обрабатывается тестером движений.

Возможно множество вариантов функций приспособленности. В данной случае использовалась функция для максимизации скорости движения (или, что эквивалентно, расстояния, проходимого за фиксированное время):

$$F(individual) = \begin{cases} 100\left(x_{after} - x_{before}\right), x_{after} > x_{before}, \\ 50\left(x_{after} - x_{before}\right), x_{after} > x_{before}, \\ 0, x_{after} \le x_{before} \end{cases}$$

При этом первая формула использовалась для устойчивого движения, а вторая – в случае падения. Здесь x_{before} и x_{after} – координаты робота (в метрах) до и после выполнения движения, отложенные по оси, совпадающей с направлением движения вперед.

4.5.5. Отбор особей для следующего поколения

После того, как для всех особей очередного поколения вычислена их приспособленность, необходимо заполнить новое поколение.

Для этого сначала выбирается k наиболее приспособленных особей из предыдущего поколения, которые переносятся в следующее поколение в неизменном виде (элитизм).

Далее по принципу рулетки отбираются особи для кроссовера и мутации, и формируется промежуточное поколение, включающее в себя предыдущее поколение и созданных потомков. Недостающие особи для нового поколения добираются из промежуточного.

Вероятность использования мутации равна p (параметр алгоритма), $0 \le p \le 1$, скрещивания, соответственно, $1 - p$.

4.5.6. Критерий останова

Оценить максимально возможное значение фитнес-функции не представляется возможным. Есть несколько вариантов для критерия останова: останов через определенное число поколений, останов в случае, если за последние k поколений максимальная приспособленность не изменилась или изменилась на малую величину.

Во всех проведенных экспериментах останов осуществлялся, в случае если максимальная приспособленность не изменялась на протяжении 30 поколений.

4.5.7. Пример применения описанного генетического алгоритма

На рис. 41 показан график роста приспособленности в случае использования генетического алгоритма со следующими параметрами:

- 30% – доля мутации в формировании нового поколения;
- 20% – процент особей, переходящих в следующее поколение в неизменном виде (элитизм);
- смешанный оператор скрещивания;
- размер поколения – 30 особей;
- максимальное время на тестирование одной особи – 20 с.

Рис. 41. График роста приспособленности

Лучшая особь, полученная в этом эксперименте, имела приспособленность 69.88, а соответствующее движение – скорость 1.18 м/мин, с сильным поворотом и смещением вправо.

4.6. Программная реализация

4.6.1. Обзор инструментального средства *GAAP*

В качестве реализации генетических алгоритмов было использовано инструментальное средство, описанное в работе [12], далее именуемое *GAAP* (*Genetic Algorithms for Automata-based Programming*). Основной частью данного инструментального средства является модуль поддержки генетических алгоритмов, который включает в себя:

- Репозиторий особей для сохранения промежуточных и конечных результатов работы. Его можно использовать в качестве общего хранилища особей при реализации метода композитного генетического программирования.
- Реализацию классического генетического алгоритма с блоком подбора и адаптации параметров алгоритма и генетических операторов под решаемую задачу.
- Средства для различных типов визуализаций и отладки процесса выполнения генетического алгоритма.
- Различные операторы и способы представления хромосом (бинарное и вещественное кодирование) для задач численной оптимизации.

Применение *GAAP* предполагает реализацию кодирования особи, механизма скрещивания и мутации, а также способа вычисления приспособленности по данной особи. Все остальные необходимые функции реализованы в самом *GAAP*.

4.6.2. Задание особи

Ранее в качестве хромосомы был выбран вектор из пяти вещественных чисел, соответствующих значениям A_{Pitch}, $A_{AnkleRoll}$, $A_{HipRoll}$, k и ω. На языке *Java* структура, описывающая хромосому, представляет из себя класс со следующими полями:

`values` – массив из пяти элементов типа `double`, хранящий значения параметров движения в следующем порядке: $A_{HipRoll}$, $A_{AnkleRoll}$, A_{Pitch}, k и ω.

`min`, `max` – статические массивы из пяти элементов типа `double`, заполненные минимальными и максимальными значениями для каждого из пяти параметров движения соответственно;

`defaultValues` – статический массив из пяти элементов типа `double`, заполненный значениями по умолчанию для каждого из пяти параметров движения;

Начальное поколение заполняется хромосомами, для создания которых используется следующий метод:

```
public static CPGMotion createIndividualFromDefault() {
    CPGMotion temp = new CPGMotion();

    for (int i = 0; i < temp.getValues().length; i++) {
        temp.getValues()[i] = defaultValues[i] + 0.05 * (0.25 -
        random.nextDouble() / 2) * (max[i] - min[i]);
        temp.getValues()[i] = Math.max(temp.getValues()[i],
        min[i]);
        temp.getValues()[i] = Math.min(temp.getValues()[i],
        max[i]);
    }
    return temp;
}
```

Этот метод генерирует новую особь так, чтобы значение каждого из пяти параметров отличалось от значения по умолчанию на небольшую случайную величину. При этом попадало в интервал между минимальным и максимальным допустимым значением.

Также для особей реализован метод `save`, который записывает движение в файл в специальном формате. Имя файла впоследствии передается в *Webots*, где движение считывается и моделируется.

4.6.3. Операторы скрещивания

Для реализации механизма скрещивания использовались два кроссовера: арифметический и смешанный.

Класс, реализующий арифметический кроссовер, содержит метод `crossover`, который получает на вход и возвращает две особи. В нем создаются две особи-потомка и для них каждый из параметров движения принимается равным `m * value1 + (1 – m)*value2`, где `m` – случайное вещественное число, `value1` и `value2` значения соответствующего параметра у родительских особей (тех, которые были получены на входе). При этом число *m* генерируется один раз при входе в метод *crossover* и затем используется для инициализации всех параметров обеих особей.

На языке *Java* описанное поведение записывается следующим образом:

```
public CPGMotion[] crossover(Individual mo, Individual fa) {
    CPGMotion mother = mo.getChromosome();
    CPGMotion father = fa.getChromosome();
    CPGMotion child1 = new CPGMotion();
    CPGMotion child2 = new CPGMotion();

    double m = random.nextDouble();

    for (int i = 0; i < 5; i++) {
```

```
            child1.getValues()[i] =
            generateNew(mother.getValues()[i],
            father.getValues()[i], m);
            child2.getValues()[i] =
            generateNew(father.getValues()[i],
            mother.getValues()[i], m);
        }

        return new CPGMotion[]{child1, child2};
    }

    private double generateNew(double value1, double value2, double
            m) {
        return m * value1 + (1 - m) * value2;
    }
```

Класс, реализующий смешанный кроссовер, устроен сходным образом. Различие состоит лишь в том, как по двум параметрам движений родительских особей вычисляется параметр для потомков. Соответствующий программный код имеет вид:

```
    private double generateNew(double value1, double value2, double
            min, double max) {
        double cmin = Math.min(value1, value2);
        double cmax = Math.max(value1, value2);
        double l = cmax - cmin;
        double res = cmin - l * ALPHA + random.nextDouble() * (cmax
            + l * ALPHA - cmin + l * ALPHA);
        res = Math.min(res, max);
        res = Math.max(res, min);
        return res;
    }
```

В проводимых экспериментах коэффициент `ALPHA` принимался равным 0.5.

4.6.4. Оператор мутации

Класс `CPGSimpleMutation`, реализующий механизм мутации, содержит метод `mutate`, получающий на вход и возвращающий одну особь. При этом один из параметров исходной особи изменяется на случайную величину, а остальные копируются в новую особь без изменений. При этом случайное число, которое прибавляется или вычитается, не превышает 25% от разности максимального и минимального допустимых значений для соответствующего параметра. Кроме того, новое значение параметра гарантированно попадет в допустимый интервал благодаря дополнительной проверке.

На языке *Java* это выглядит следующим образом:

```
    public CPGMotion mutate(Individual individual) {
        CPGMotion ind = individual.getChromosome();
        CPGMotion temp = new CPGMotion();

        for (int i = 0; i < 5; i++) {
            temp.getValues()[i] = ind.getValues()[i];
        }
```

```
        int m = random.nextInt(5);
        temp.getValues()[m] = temp.getValues()[m] + (0.25 -
            random.nextDouble() / 2) * (CPGMotion.max[m] -
            CPGMotion.min[m]);
        temp.getValues()[m] = Math.max(temp.getValues()[m],
            CPGMotion.min[m]);
        temp.getValues()[m] = Math.min(temp.getValues()[m],
            CPGMotion.max[m]);

        return temp;
    }
```

4.6.5. Вычисление приспособленности

После того, как генетический алгоритм сформирует новое поколение, для каждой его особи должно быть сопоставлено значение фитнесс-функции. Для этого движения, соответствующие особям, передаются в *Webots*, где каждое движение проверяется на роботе. Передача осуществляется посредством записи информации о движениях в файлы в специальном формате. Файлы записываются в указанную заранее директорию. Каждый файл получает уникальное имя, сопоставленное с особью, которой он соответствует. В это время в *Webots* уже запущено моделирование. Специальный контроллер, написанный для *Webots*, считывает информацию о движениях, обрабатывает ее и выводит значение фитнесс-функции в выходной файл, имя которого также привязано к оцениваемой особи. Из этих выходных файлов посчитанные значения приспособленности поступают обратно в генетический алгоритм. Процесс продолжается до тех пор, пока не будут обработаны все особи текущего поколения.

4.6.6. Реализация взаимодействия со средой моделирования *Webots*

Взаимодействие со средой *Webots* происходит следующим образом. Для *Webots* подготовлен отдельный файл со сценой, содержащей стоящего в исходном положении робота (значения углов всех сервомоторов равны нулю). За сценой закреплен специальный контроллер – программа на языке *Java*, которая будет автоматически запущена при запуске моделирования, и которая сможет отслеживать состояние робота, взаимодействовать с его управляющей программой, принимать описания движений, генерируемых генетическим алгоритмом, и выводить данные о результате их исполнения роботом. Кроме того, роботу сопоставлена управляющая программа, которая также запускается при запуске моделирования.

Процесс моделирования должен быть запущен на протяжении всего процесса эволюции. Контроллер проверяет, не появились ли новые файлы с движениями. Как только такой файл появится, путь к нему передается управляющей программе робота. Это делается с использованием эмиттера и ресивера. Они представляют собой передающее и принимающее устройства, имеющиеся как у робота, так и у контроллера, при помощи которых они могут взаимодействовать в рамках моделируемого мира.

Управляющая программа, получив путь к файлу, открывает его и считывает движение. Затем робот пытается выполнить это движение. Как только движение будет полностью проиграно или робот упадет, управляющая программа вычислит фитнесс-функцию по соответствующей формуле и отправит результат контроллеру. В свою очередь, контроллер запишет полученное значение в выходной файл и приступит к обработке следующего движения. Кроме того, контроллер установит робота в исходное положение.

4.7. Выбор оптимальных параметров генетического алгоритма

Для получения лучших результатов и оптимальной скорости работы генетического алгоритма необходимо правильно выбрать следующие параметры:

- процент особей, переходящих в следующее поколение в неизменном виде (элитизм);
- соотношение между мутацией и кроссовером;
- наиболее эффективный оператор кроссовера;
- размер поколения;
- время на тестирование одной особи.

Для выбора перечисленных параметров был проведен ряд экспериментов.

4.7.1. Выбор оптимального элитизма

Для того чтобы определить оптимальный процент элитизма, были проведены эксперименты с переменным элитизмом и зафиксированными остальными параметрами:

- 70% потомков формируются посредством кроссовера, 30% – с помощью мутации;
- использовался арифметический кроссовер;
- размер поколения – 30 особей;
- максимальное время на тестирование одной особи – 10 с.

Результаты проведенных экспериментов приведены в табл. 21, 22.

Таблица 21. Связь между максимальной приспособленностью и элитизмом

№ п/п	Элитизм, %	Максимальная приспособленность			
		Эксперимент 1	Эксперимент 2	Эксперимент 3	В среднем
1	5	34.04	53.10	38.39	41.85
2	10	39.74	37.42	29.20	35.46
3	20	35.21	51.57	34.91	40.56
4	30	35.74	33.88	32.72	34.11
5	40	15.30	35.67	30.48	27.15

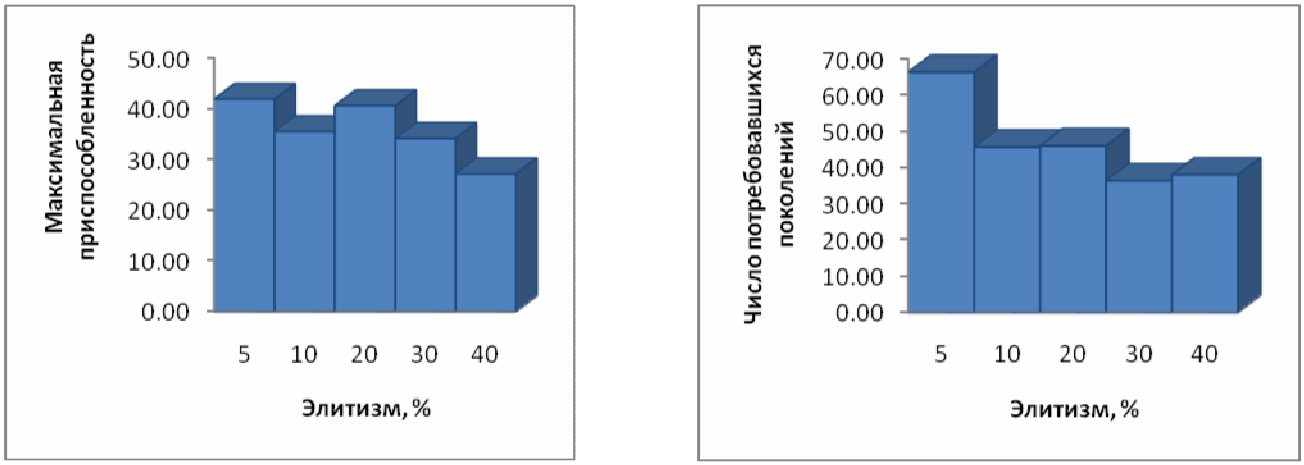
Таблица 22. Связь между числом поколений и элитизмом

№ п/п	Элитизм, %	Число потребовавшихся поколений			
		Эксперимент 1	Эксперимент 2	Эксперимент 3	В среднем
1	5	91	20	88	66.33
2	10	30	85	22	45.67
3	20	50	39	49	46.00
4	30	83	8	18	36.33
5	40	34	58	22	38.00

На рис. 42 показаны усредненные зависимости между процентом лучших особей, переходящих в новое поколение, и полученной максимальной приспособленностью (слева) и числом поколений (справа). Видно, что лучшие результаты дает элитизм от пяти до 20 процентов. При этом

для 10 и 20 процентов требуется меньшее число поколений. На рис. 43 показано отношение максимальной приспособленности к числу поколений.

В дальнейших экспериментах для элитизма будет использовано значение 20 процентов, так как оно оптимально и по полученной приспособленности, так и по числу поколений.



а б

Рис. 42. Связь между максимальной приспособленностью и элитизмом

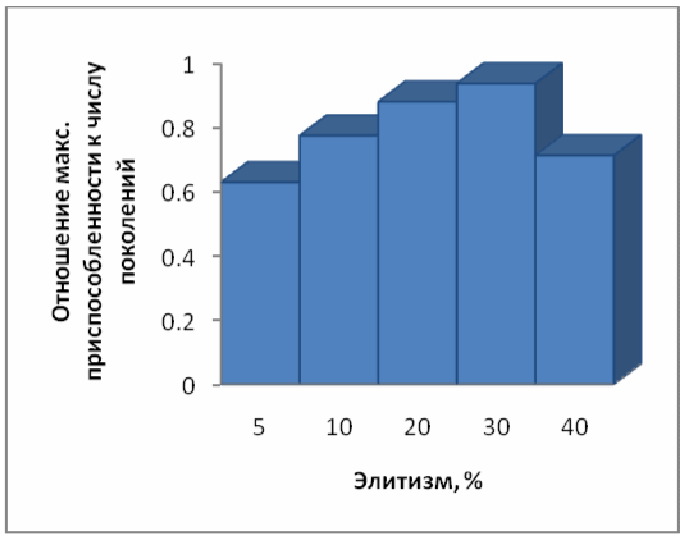


Рис. 43. Отношение максимальной приспособленности к числу поколений в соответствующем эксперименте

4.7.2. Выбор оптимального соотношения кроссовера и мутации

Для следующей группы экспериментов использовались следующие параметры:

- переменное соотношение кроссовера и мутации;
- 20% – процент особей, переходящих в следующее поколение в неизменном виде (элитизм);
- арифметический кроссовер;
- размер поколения – 30 особей;

- максимальное время на тестирование одной особи – 20 с.

Время, отведенное на тестирование одной особи, было увеличено по сравнению с предыдущим экспериментом. Результаты экспериментов приведены в табл. 23.

Таблица 23. Зависимость максимальной приспособленности от доли мутации при формировании нового поколения

№ п/п	Кроссовер, %	Мутация, %	Максимальная приспособленность
1	95	5	25.00
2	90	10	26.87
3	80	20	27.49
4	70	30	49.65
5	60	40	50.10
6	50	50	41.72
7	40	60	44.61

Из рис. 44 видно, что наибольшая максимальная приспособленность достигается при доле мутации 30%.

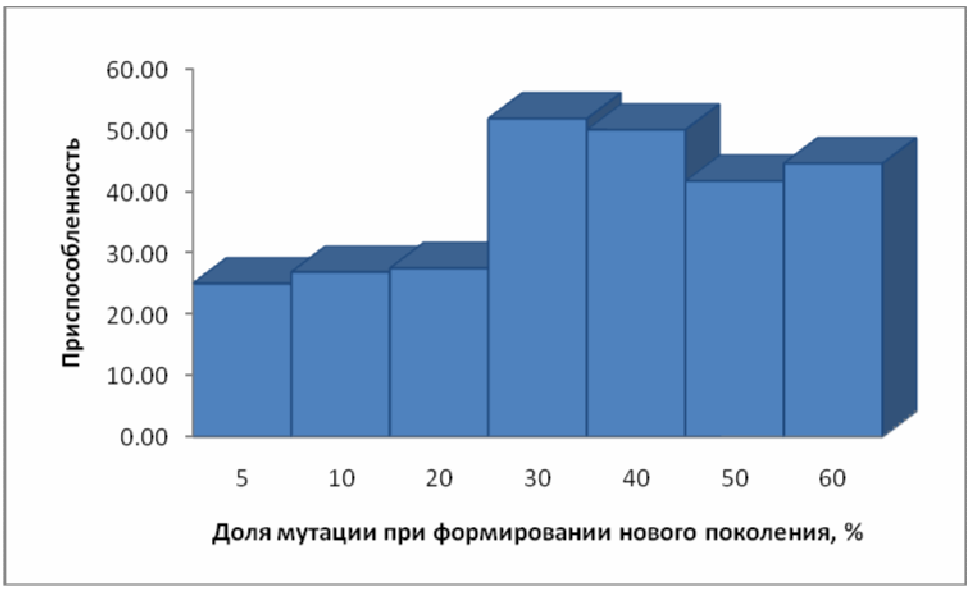


Рис. 44. Зависимость максимальной приспособленности от доли мутации при формировании нового поколения

4.7.3. Выбор наиболее эффективного оператора кроссовера

Для следующей группы экспериментов использовались следующие параметры:

- 20% – процент особей, переходящих в следующее поколение в неизменном виде (элитизм);
- при формировании нового поколения 70% потомков формировались при помощи оператора кроссовера, остальные 30% – при помощи мутации;

- размер поколения – 30 особей;
- максимальное время на тестирование одной особи – 20 с.

Как видно из табл. 24 и рис. 45, максимальная приспособленность в поколении, полученная с использованием смешанного оператора скрещивания, заметно превосходит аналогичную величину, полученную с использованием арифметического кроссовера.

Таблица 24. Зависимость максимальной приспособленности от используемого оператора скрещивания

№ п/п	Оператор скрещивания	Максимальная приспособленность
1	Арифметический	51.92
2	Смешанный	69.88

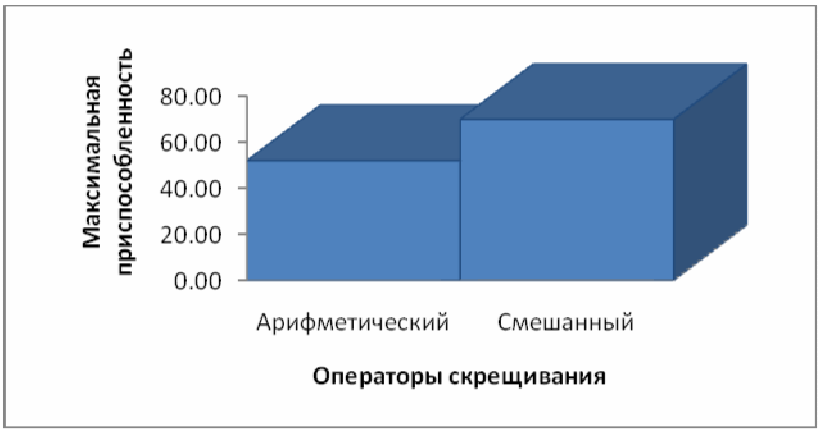


Рис. 45. Зависимость максимальной приспособленности от используемого оператора скрещивания

4.7.4. Выбор оптимального размера поколения

Для следующей группы экспериментов использовались следующие параметры:

- 20% – процент особей, переходящих в следующее поколение в неизменном виде (элитизм);
- при формировании нового поколения 70% потомков формировались при помощи оператора кроссовера, остальные 30% – при помощи мутации;
- смешанный оператор скрещивания;
- максимальное время на тестирование одной особи – 20 с.

Результаты соответствующих экспериментов приведены в табл. 25.

Таблица 25. Зависимость максимальной приспособленности от размера популяции

№ п/п	Размер популяции	Максимальная приспособленность	Число поколений
1	20	48.67	106
2	30	69.88	57
3	40	69.34	49
4	50	70.06	69

Отметим, что хотя для экспериментов с большим размером популяции, и приспособленность была больше, но число поколений, потребовавшихся для достижения такой приспособленности, не сильно отличается. Однако время, затрачиваемое на вычисление приспособленности 50 особей, по сравнению с 30 особями, существенно больше. Поэтому использование популяций больше, чем из 30 особей, не имеет смысла.

4.8. Анализ результатов

Было произведено сравнение движения, полученное предложенным методом с использованием выбранных оптимальных параметров, с движением из файла `Forwards.motion`, поставляемого вместе с *Webots* в качестве примера. Результаты приведены в табл. 26.

Таблица 26. Сравнение полученного методом движения со стандартным

Движение	Скорость, м/мин	Отклонение в сторону, м/мин	Поворот в сторону, °/мин
Стандартное движение (`Forwards.motion`)	1.64	0.15	4.43
Начальное движение	0.47	0.02	3.65
Полученное движение	2.11	0.19	0.68

Таким образом, полученное предложенным методом движение, не уступает аналогам, полученным другими методами, а по скорости – превосходит примерно на 30%.

Выводы по главе 4

1. Разработан метод применения генетических алгоритмов для построения системы управления виртуальным человекоподобным роботом (на примере управления ходьбой).
2. Выполнена программная реализация разработанного метода на языке программирования *Java*.
3. Проведено экспериментальное исследование разработанного метода на задаче построения системы управления шагом вперед человекоподобного робота – построенное движение по скорости превосходит примерно на 30% движения, построенные другими методами.

ЗАКЛЮЧЕНИЕ

В результате исследований, выполненных на первом этапе работ по контракту, были проведены следующие работы:

- выполнение аналитического обзора;
- выбор и обоснование направления исследований;
- подготовка плана проведения теоретических и экспериментальных исследований;
- разработка, программная реализация и экспериментальное исследование метода применения генетических алгоритмов для построения системы управления виртуальным человекоподобным роботом (на примере управления ходьбой).

Аналитический обзор был выполнен по следующим направлениям:

- методы поиска при условиях ограничений в разработке программного обеспечения;
- обучение с подкреплением;
- нечеткие множества и нечеткая логика;
- искусственные нейронные сети;
- построение систем управления мобильными роботами;
- управление человекоподобными роботами;
- мультиагентные системы.

В результате выполнения аналитического обзора разработан план проведения теоретических и экспериментальных исследований. В соответствии с этим планом в рамках теоретических исследований будут разработаны следующие методы генетического программирования:

- метод применения генетического программирования для построения систем управления виртуальными роботами для упрощенной модели игры в футбол;
- метод применения коэволюции для построения взаимодействующих управляющих конечных автоматов;
- метод совместного применения конечных автоматов и нейронных сетей;
- метод применения верификации моделей программ при построении управляющих конечных автоматов с помощью генетического программирования.

В соответствии с планом экспериментальных исследований будут проведены эксперименты на программной модели малоразмерного мобильного робота, а также непосредственно с самим роботом. Кроме этого, будет разработана виртуальная лаборатория для обучения генетическим алгоритмам для языков программирования *Java*, *C++*, *C#*. В заключение работы будет проведено обобщение и оценка результатов исследований.

Результаты выполненных работ позволяют утверждать, что научно-технический уровень исследований превышает уровень исследований в рассматриваемой области, проводимых в лучших исследовательских центрах мира.

ИСТОЧНИКИ

1. *Абарбанель Г.Д., Рабинович М.И., Селверстон А. и др.* Синхронизация в нейронных ансамблях. http://ufn.ru/ufn96/ufn96_4/Russian/r964b.pdf
2. *Воронин О., Дьюдни А.* Дарвинизм в программировании //Мой компьютер. 2004. № 35. http://www.mycomp.kiev.ua/text/7458
3. *Гач П., Курдюмов Г. Л., Левин Л. А.* Одномерные однородные среды, размывающие конечные острова // Проблемы передачи информации. 1976. 14, 3.
4. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит, 2006.
5. *Горбань А. Н., Дунин-Барковский В. Л., Кирдин А. Н. и др.* Нейроинформатика. Новосибирск: Наука. 1998. http://algolist.manual.ru/ai/neuro/neuroinform.zip
6. *Заде Л.* Понятие лингвистической переменной и его применение к принятию приближенных решений. М.: Мир, 1976.
7. *Заде Л., Дезоер Ч.* Теория линейных систем. Метод пространства состояний. М.: Наука, 1970.
8. *Клебан В. О., Шалыто А. А.* Использование автоматного программирования для построения многоуровневых систем управления мобильными роботами / Сборник тезисов 19 Всероссийской научно-технической конференции «Экстремальная робототехника». СПб: ЦНИИ РТК. 2008, с. 85–87.
9. *Колесов Ю. Б., Сениченков Ю.Б.* Моделирование систем. Динамические и гибридные системы. СПб.: БХВ-Петербург, 2006.
10. *Лобанов П. Г., Шалыто А. А.* Использование генетических алгоритмов для автоматического построения конечных автоматов в задаче о «Флибах» / Сборник докладов 4-й Всероссийской научной конференции «Управление и информационные технологии» (УИТ-2006). СПбГЭТУ «ЛЭТИ». 2006, с.144–149. http://is.ifmo.ru/works/flib
11. *Люгер Дж. Ф.* Искусственный интеллект: стратегии и методы решения сложных проблем. М: Вильямс, 2003.
12. *Мандриков Е.А., Кулев В.А.* Разработка инструментального средства для генерации конечных автоматов с использованием генетических алгоритмов / Материалы V межвузовской конференции молодых ученых.
13. *Непейвода Н. Н.* Стили и методы программирования. М.: Интернет-Университет Информационных технологий, 2005.
14. *Николенко С. И.* Лекции по генетическим алгоритмам. http://logic.pdmi.ras.ru/~sergey/teaching/ml/
15. *Новак В., Перфильева И.* Математические принципы нечеткой логики. М.: Физматлит, 2006.
16. *Ослэндер Д., Риджли Д., Рингенберг Д.* Управляющие программы для механических систем. Объектно-ориентированное проектирование систем реального времени. М.: БИНОМ. Лаборатория знаний. 2004.
17. *Пегат А.* Нечеткое моделирование и управление. М.: Бином, 2009.
18. *Полимеразная цепная реакция.* http://en.wikipedia.org/wiki/Polymerase_chain_reaction
19. *Рассел С., Норвиг П.* Искусственный интеллект. Современный подход. М.: Вильямс, 2006.
20. *Терехов В. А.* Нейросетевые системы управления. М: Высшая школа, 2002.
21. *Туккель Н. И., Шалыто А. А.* Система управления дизель-генератором (фрагмент). Программирование с явным выделением состояний. Проектная документация. http://is.ifmo.ru/projects/dg/

22. *Хопкрофт Дж., Мотвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.
23. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб: Наука, 1998. http://is.ifmo.ru/books/switch/1
24. *Albus J. S.* Data Storage in the Cerebellar Model Articulation Controller (CMAC) // Journal of Dynamic Systems, Measurement and Control, 1975 № 4, pp. 228–233.
25. *Andre D., Bennet F., Koza J.* Discovery by Genetic Programming of a Cellular Automata Rule that is Better than any Known Rule for the Majority Classification Problem. 1996. http://citeseer.ist.psu.edu/andre96discovery.html
26. *Angeline P., Pollack J.* Evolutionary Module Acquisition / Proceedings of the Second Annual Conference on Evolutionary Programming. Cambridge: MIT Press. 1993, pp.154–163. http://www.demo.cs.brandeis.edu/papers/ep93.pdf
27. *Armstrong, D.* A programmed algorithm for assigning internal codes to sequential machines // IRE Transactions on Electronic Computers. EC.11. 1962. I.4, pp.466–472.
28. *Axelrod R.* The Evolution of Cooperation. NY: Basic Books, 1984.
29. *Axelrod R.* The Evolution of Strategies in the Iterated Prisoner's Dilemma /In Genetic Algorithms and Simulated Annealing. London: Pitman, 1987, pp. 32–41. http://www-personal.umich.edu/~axe/research_papers.html
30. *Back T.* Evolutionary Algorithms in Theory and Practice. Oxford University Press, 1996.
31. *Bak T., Bendsen J.* Hybrid Control Design for a Wheeled Mobile Robot
32. *Barto A. G., Sutton R. S., Anderson C. W.* Neuron-like elements that can solve difficult learning control problems //IEEE Transactions on Systems, Man and Cybernetics. 1983, №13, pp. 835–846.
33. *Behnke S.* Online Trajectory Generation for Omnidirectional Biped Walking /In Proceedings of IEEE International Conference on Robotics and Automation (ICRA'06). 2006. Orlando. Florida, pp. 1597–1603. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.8947
34. *Bellman R.* Dynamic Programming, Princeton. NJ: Princeton University Press, 1957.
35. *Berlekamp, E. R.; Conway, John Horton; Guy, R.K.*, Winning Ways for your Mathematical Plays. 2001: A K Peters Ltd.
36. *Belz A., Eskikaya B.* A Genetic Algorithm for Finite State Automata Induction with Application to Phonotactics / Proceedings of the ESSLLI–98 Workshop on Automated Acquisition of Syntax and Parsing. Saarbruecken. 1998, pp. 9–17. http://www.itri.brighton.ac.uk/~Anja.Belz/Publications/A_GA_for_FSA_induction_with_an_application_to_phonotactics.ps
37. *Benson K.* Evolving Finite State Machines with Embedded Genetic Programming for Automatic Target Detection. http://ieeexplore.ieee.org/iel5/6997/18853/00870838.pdf
38. *Brave S.* Evolving Deterministic Finite Automata Using Cellular Encoding / Proceeding of First Annual Conference (Genetic Programming–1996), pp. 39–44. http://citeseer.ist.psu.edu/131538.html
39. *Brooks R. A.* A Robust Layered Control System for a Mobile Robot //IEEE Journal of Robotics and Automation. 1986. 2, pp.14–23.
40. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volumes I. II, III. CRC Press, 1999.
41. *Changjiu Zhou, Lingyun Hu, Carlos A A Acosta, Pik Kong Yue* Humanoid Soccer Gait Generation and Optimization Using Probability Distribution Models, 2006. http://www.dei.unipd.it/~emg/whs2006/papers/HSR-110.pdf
42. *Collan, Mikael and Liu, Shuhua*, Fuzzy Logic and Intelligent Agents: Towards the Next Step of Capital Budgeting Decision Support 2002: Working Papers 398, IAMSR, Abo Akademi.

43. *Das R., Mitchell M., Crutchfield J.P.* A genetic algorithm discovers particle-based computation in cellular automata //Lecture Notes in Computer Science. 1994. www.santafe.edu/research/publications/workingpapers/94-03-015.pdf
44. *Driankov D., Hellndoorn H. and Reinfrank M.* An introduction to fuzzy control. Berlin: Springer-Verlag, 1993.
45. *Enrique Herrera-Viedma. Eduardo Peis. José M. Morales-del-Castillo* Gathering information on the Web using fuzzy linguistic agents and Semantic Web technologies 2005 //4th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT 2005).
46. *Ernst D., Geurts P., Wehenkel L.* Tree-Based Batch Mode Reinforcement Learning // Journal of Machine Learning Research 2005. № 6, pp. 503–556.
47. *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
48. *Fogel D.* The Evolution of Intelligent Decision Making in Gaming // Cybernetics and Systems. 2001. 22, pp. 223–236.
49. *Fogel L.* Autonomous Automata // Industrial Research. 1962. V.4, pp. 14–19.
50. *Fogel L., Owens A., Walsh M.* Artificial Intelligence through Simulated Evolution. NY: Wiley. 1966.
51. *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
52. *Ghnemat R., Khatatneh K., Oqeili S., Bertelle C., Duchamp G.* Automata-based adaptive behavior for economic modeling using game theory. 2005. http://arxiv.org/abs/cs/0510089
53. *Glover F.* Tabu search: A tutorial. Interfaces 20 (1990), pp. 74–94.
54. *Gold E. M.* Complexity of automaton identification from given data. // Information and Control. 1978, № 37, pp. 302–320.
55. *Goldberg D.* A Note on Boltzmann Tournament Selection for Genetic Algorithms and Population-Oriented Simulated Annealing // Complex Systems. 1990. V. 4, I. 4, pp. 445–460.
56. *Harel D., Pnueli A.* On the development of reactive systems / In «Logic and Models of Concurrent Systems». NATO Advanced Study Institute on Logic and Models for Verification and Specification of Concurrent Systems. Springer Verlag, 1985. pp. 477–498.
57. *Harman N., Hierons R., Proctor M.* A new representation and crossover operator for search-based optimization of software modularization /In GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference (New York, 9-13 July 2002), Morgan Kauffman Publishers, pp. 1351–1358.
58. *Ha S., Han Y., Hahn H.* Adaptive Gait Pattern Generation of Biped Robot based on Human's Gait Pattern Analysis. http://www.waset.org/ijmsse/v1/v1-2-14.pdf
59. *Herrera F., Lozano M., Verdegay J.L.* Tackling real-coded Genetic algorithms: operators and tools for the behaviour analysis. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.56.7704
60. *Hillis W.* Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure / In Artificial Life II. MA: Addison-Wesley 1992, pp. 313–324.
61. *Holland J. H.* Adaptation of Natural and Artificial Systems. MIT Press, Ann Arbor, 1975.
62. *Holland J.* ECHO: Explorations of Evolution in a Miniature World / Proceedings of the Second Conference on Artificial Life. Redwood City. CA: Addison-Wesley, 1990.
63. *Horihan J., Yung-Hsiang Lu.* Improving FSM evolution with progressive fitness functions / In GLSVLSI '04: Proceedings of the 14th ACM Great Lakes symposium on VLSI. NY: ACM Press. 2004, pp. 123–126.

64. *Huang D.* MS Thesis Preproposal: Adaptive Incremental Fitness Evaluation in Genetic Algorithms. 2005. NY: Rochester.
http://www.cs.rit.edu/~dxh6185/downloads/MS_Thesis/Documents/Presentation.pdf
65. *Isermann*. On fuzzy logic applications for automatic control, supervision and fault diagnostic. /Proceedings of the International Conference EUFIT'96. 1996. Vol. 2, pp. 738-753.
66. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A.* The Genesys System: Evolution as a Theme in Artificial Life /Proceedings of Second Conference on Artificial Life. MA: Addison-Wesley. 1992, pp.549–578.
www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html
67. *Jones B., Sthamer H.-H., Eyres D.* Automatic structural testing using genetic algorithms //The Software Engineering Journal. 1996. 11, pp. 299–306.
68. *Jones B., Eyers D., Sthamer H.-H.* A strategy for using genetic algorithms to automate branch and fault-based testing //The Computer Journal. 41. 1998. 2 , pp. 98–107.
69. *Juillie H., Pollack J.* Coevolving the «Ideal» Trainer: Application to the Discovery of Cellular Automata Rules. 1998. http://citeseer.ist.psu.edu/16712.html
70. *Kirsopp C., Shepperd M., Hart J.* Search heuristic, case-based reasoning and software project effort prediction /In GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference (New York, 2002). Morgan Kauffman Publishers, pp. 1367–1374.
71. *Koza J.R.* Genetic Evolution and Co-Evolution of Computer Programs / Proceedings of Second Conference on Artificial Life. Redwood City. CA: Addison-Wesley. 1992, pp. 603–629. http://citeseer.ist.psu.edu/177879.html
72. *Koza J.R.* Genetic programming. On the Programming of Computers by Means of Natural Selection. MA: The MIT Press, 1998.
73. *Levent Akin H., Mericli C., Mericli T.* Cerberus'08 Team Report. http://www.tzi.de/spl/pub/Website/Teams2008/Cerberus.pdf
74. *Levy S.* Artificial Life: The Quest for a New Creation. NY: Pantheon, 1992.
75. *Linton R.* Adapting binary fitness functions in genetic algorithms / Proceedings of the 42nd annual Southeast regional conference. NY: ACM Press. 2004. pp. 391–395.
76. *Lucas M., Reynolds T. J.* Learning DFA: Evolution versus Evidence Driven State Merging.
77. *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
78. *Mamdani, E.H.* Application of fuzzy algorithms for control of simple dynamic plant. / Proceedings IEEE. 1974. №121, pp.1585–1588.
79. *MathWorks Inc.* Neural Network Toolbox User's Guide. http://www.mathworks.com/access/helpdesk/help/pdf_doc/nnet/nnet.pdf
80. *Metropolis N., Rosenbluth A., Rosenbluth M., Teller A., Teller E.* Equation of state calculations by fast computing machines // Journal of Chemical Physics. 21 (1953), pp. 1087–1092.
81. *Michel O.* Cyberbotics Ltd – WetsTM: Professional Mobile Robot Simulation. //International Journal of Advanced Robotic Systems. V. 1. N 1 (2004), pp. 39–42. http://www.cyberbotics.com/publications/ars.pdf
82. *Miller J.* The Coevolution of Automata in the Repeated Prisoner's Dilemma. Working Paper. Santa Fe Institute. 1989 // Journal of Economic Behavior & Organization. 1996. V. 29. I. 1, pp. 87–112.
83. *Mitchell B. S.* A Heuristic Search Approach to Solving the Software Clustering Problem. PhD Thesis. Drexel University., Philadelphia, PA, Jan. 2002.

84. *Mitchell B. S., Mancordis S.* Using heuristic search techniques to extract design abstractions from source code //In GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference (New York, 2002)/ Morgan Kauffman Publishers, pp. 1375–1382.
85. *Mitchell M.* An Introduction to Genetic Algorithms. MA: The MIT Press, 1996.
86. *Mitchell M., Crutchfield J. P., Hraber P. T.* Evolving cellular automata to perform computations //Physica D. 75. 1993, pp. 361–391, http://web.cecs.pdx.edu/~mm/mech-imped.pdf
87. *Morimoto J., Endo G., Nakanishi J.* Modulation of simple sinusoidal patterns by a coupled oscillator model for biped walking.
88. *Murray M.P., Drought A.B., Kory R.C.* Walking Patterns of Normal Men. http://www.ejbjs.org/cgi/content/abstract/46/2/335
89. *Naidoo A., Pillay N.* The Induction of Finite Transducers Using Genetic Programming /Proceedings of Euro GP. Springer. 2007. http://saturn.cs.unp.ac.za/~nelishiap/papers/eurogp07.pdf
90. *Nedjah N., Mourelle L.* Mealy Finite State Machines: An Evolutionary Approach // International Journal of Innovative Computing, Information and Control. 2006. V.2, I. 4.
91. *von Neumann, John. A. Burks. ed.* The Theory of Self-reproducing Automata. Urbana. IL: Univ. of Illinois Press, 1993.
92. *Ormoneit D., Glynn P.* Kernel-based reinforcement learning in average-cost problems / IEEE Transactions on Automatic Control. 2002. № 47(10), pp. 624–636.
93. *Pargas R. P., Harrold M. J., Peck R. R.* Test-data generation using genetic algorithms //The Journal of Software Testing, Verification and Reliability. 1999. 9, pp. 263–282.
94. *Pedrycz W.* An identification algorithm in fuzzy relational systems // Fuzzy Sets and Systems. 1984. №13, pp. 153–167.
95. *Pettersson S., Lennartson B.* Modelling, Analysis & Synthesis of Hybrid Systems http://citeseer.ist.psu.edu/187615.html
96. *Ray T.* An Approach to the Synthesis of Life / In Artificial Life II. MA: Addison-Wesley, 1992, pp. 371–408.
97. *Reynolds C. W.* Competition, Coevolution and the Game of Tag / Proceedings of Artificial Life IV. Cambridge. MA: MIT Press. 1994, pp. 59–69. http://www.red3d.com/cwr/papers/1994/alife4.html
98. *Rummery G., Niranjan M.* On-line Q-learning using Connectionist systems. Technical report №166. University of Cambridge. Engineering Department, 1994.
99. *Sutton R. S.* Learning to predict by methods of temporal differences //Machine Learning. 1988. 3, pp. 9–44.
100. *Sutton R. S., Barto A. G.* Reinforcement Learning: An Introduction. MIT Press, 1998.
101. *Sutton R. S.* Implementation details of TD(λ) procedure for case of vector predictions and backpropagation. TN87-509.1, GTE Laboratories, 1989.
102. *Takagi T., Sugeno M.* Fuzzy identification of systems and its applications to modeling and control. / IEEE Transactions on Systems, Man and Cybernetics. 1985. Vol. 15/ №1, pp.116–132.
103. *Tham C. K., Prager K. W.* Reinforcement Learning for Multi-linked Manipulator Control, 1994.
104. *Tracey N., Clark J., Mander K.* Automated program flaw finding using simulated annealing //In International Symposium on Software Testing and Analysis (March 1998). ACM/SIGSOFT, pp. 73–81.
105. *Tracey N., Clark J., Mander K.* The way forward for unifying dynamic test-case generation: The optimisation-based approach //In International Workshop on Dependable Computing and Its Applications (DCIA) (January 1998). IFIP, pp. 169–180.

106. *Tu M., Wolff E., Lamersdorf W.* Genetic Algorithms for Automated Negotiations: A FSM-based Application Approach /Proceedings of the 11-th International Conference on Database and Expert Systems. 2000. http://ieeexplore.ieee.org/iel5/7035/18943/00875153.pdf

107. *Van den Kieboom J.* Biped Locomotion and Stability a Practical Approach.http://scripties.fwn.eldoc.ub.rug.nl/FILES/scripties/Kunstmatigeintellige/Master/2009/Kieboom.J.van.den./AI-MAI-2009-J.VANDENKIEBOOM.pdf

108. *Vukobratovic M., Borovac B.* Zero-moment point – thirty five years of its life. http://www.cs.cmu.edu/~cga/legs/vukobratovic.pdf

109. *Wang, Shen, Georganas* A Fuzzy Logic Based Intelligent Negotiation Agent (FINA) //In Ecommerce 2006: Electrical and Computer Engineering. CCECE '06. 2006. Canadian Conference

110. *Watkins C.* Learning from Delayed Rewards, University of Cambridge, 1989.

111. *Wegener J., Baresel A., Sthamer H.* Evolutionary test environment for automatic structural testing //Information and Software Technology Special Issue on Software Engineering using Metaheuristic Innovative Algorithms. 2001. 43, 14, pp. 841–854.

112. *Wegener J., Grimm K., Grochtmann M., Sthamer H., Jones B. F.* Systematic testing of real-time systems // In 4th International Conference in Software Testing Analysis and Review (EuroSTAR 96). 1996.

113. *Wegener J., Sthamer H., Jones B. F., Eyres D. E.* Testing real-time systems using genetic algorithms // Software Quality. 1997. 6, pp. 127–135.

114. *Whitley D.* The GENITOR Algorithm and Selective Pressure / Proceedings of the 3rd International Conference on Genetic Algorithms. Colorado State: Morgan Kaufmann. 1989, pp. 116–121. http://www.genalgo.com/index.php?option=com_content&task=view&id=80&Itemid=30

115. *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

116. *Yager R., Filev D.* Essentials of fuzzy modeling and control. New York: John Wiley and Sons, 1995.

117. *Zomorodian A.* Context-free Language Induction by Evolution of Deterministic Push-Down Automata Using Genetic Programming. http://www.cs.dartmouth.edu/~afra/papers/aaai96/aaai96.pdf

118. *Zykov V., Bongard J., Lipson H.* Evolving Dynamic Gaits on a Physical Robot. http://ccsl.mae.cornell.edu/papers/GECCO04_Zykov.pdf