грокаем

Алгоритмы искусственного интеллекта

Ришал Харбанс

Карта книги «Трокаем алгоритмы искусственного интеллекта»

Понимание ИИ
Алгоритмы обрабатывают данные для решения сложных задач. Различные алгоритмы лучше подходят для решения различных типов задач. Для решения комплексных задач можно использовать комбинации алгоритмов.

Основы поиска
Алгоритмы неинформированного поиска находят лучшее решение путем перебора всех вариантов, но требуют больших вычислительных затрат. Они помогают увидеть полезные структуры данных для использования другими алгоритмами ИИ.

Умный поиск
Алгоритмы информированного поиска используют эвристику, чтобы задать направление поиска. Они часто применяются при решении состязательных задач, где нужно учитывать наличие другого агента.

Роевой интеллект: муравьи
В основе генетических алгоритмов лежит концепция эволюционного развития. Возможные решения кодируются и совершенствуются на протяжении нескольких поколений для выявления наиболее эффективных.

Эволюционные алгоритмы
Алгоритмы неинформированного поиска находят лучшее решение путем перебора всех вариантов, но требуют больших вычислительных затрат. Они помогают увидеть полезные структуры данных для использования другими алгоритмами ИИ.

Роевой интеллект: частицы
Метод роя частиц вдохновлен перемещением птичьих стай. Для перемещения по области решений частицы используют собственное лучшее положение и лучшее положение роя.

Машинное обучение
Алгоритмы регрессии и классификации изучают закономерности в данных для предсказания различных значений или принадлежности образцов к определенным категориям. Ключ к удачной модели МО — продуманные и хорошо подготовленные данные.

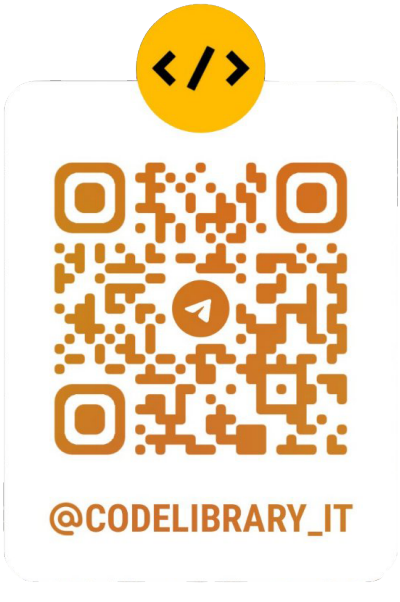
Искусственные нейронные сети
Искусственные нейронные сети симулируют работу головного мозга и нервной системы. Нейронная сеть получает сигналы, обрабатывает их, взвешивает и формирует на выходе результат с учетом корреляций, найденных во входных сигналах.

Обучение с подкреплением с помощью Q-обучения
Обучение с подкреплением применяет метод проб и ошибок с наградами и штрафами за совершаемые действия в заданном окружении. Таким образом алгоритм обучается совершать правильные действия для достижения поставленной цели.

grokking

Artificial Intelligence Algorithms

Rishal Hurbans



MANNING
SHELTER ISLAND

грокаем

Алгоритмы искусственного интеллекта

Ришал Харбанс

ПИТЕР®

Санкт-Петербург • Москва • Минск

2023

ББК 32.973.2-018+32.813
УДК 004.021+004.8
Х20

Ришал Харбанс

Х20 Грокаем алгоритмы искусственного интеллекта. — СПб.: Питер, 2023. — 368 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-2924-9

Искусственный интеллект — часть нашей повседневной жизни. Мы встречаемся с его проявлениями, когда занимаемся шопингом в интернет-магазинах, получаем рекомендации «вам может понравиться этот фильм», узнаем медицинские диагнозы…

Чтобы уверенно ориентироваться в новом мире, необходимо понимать алгоритмы, лежащие в основе ИИ.

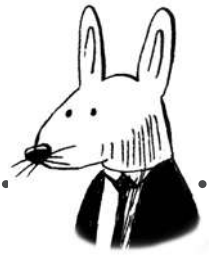
«Грокаем алгоритмы искусственного интеллекта» объясняет фундаментальные концепции ИИ с помощью иллюстраций и примеров из жизни. Все, что вам понадобится, — это знание алгебры на уровне старших классов школы, и вы с легкостью будете решать задачи, позволяющие обнаружить банковских мошенников, создавать шедевры живописи и управлять движением беспилотных автомобилей.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

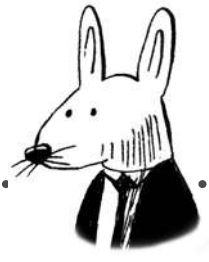
ББК 32.973.2-018+32.813
УДК 004.021+004.8

ISBN 978-1617296185 англ.
ISBN 978-5-4461-2924-9

Краткое содержание



Оглавление



Посвящается моим родителям,
Пранилу и Рекхе — спасибо, что помогли мне
добиться успеха.

Предисловие

В предисловии рассказывается о развитии технологии, о нашей потребности в автоматизации, а также о необходимой ответственности за принятие этических решений при использовании искусственного интеллекта.

Одержимость технологиями и стремление к автоматизации

На протяжении всей своей истории человечество неустанно искало способы решать задачи, прикладывая как можно меньше ручного труда и усилий. Люди всегда стремились создавать инструменты и автоматизировать повторяющиеся действия, чтобы выживать и сохранять энергию. Здесь некоторые могут поспорить, заявив, что человек умен, он ищет возможности совершенствования, решая задачи творчески или создавая произведения литературы, музыки и других искусств. Но эта книга не ставит целью обсуждение философской сути бытия. Она дает обзор методов искусственного интеллекта (ИИ), которые можно применять для решения прикладных задач. Решая эти задачи, мы облегчаем себе жизнь, делая ее более безопасной, здоровой, полной и радостной. Все технологические достижения человечества, которые вы видите сегодня, включая ИИ, удовлетворяют потребности отдельных людей, их сообществ и целых наций.

Чтобы создавать будущее, необходимо понимать ключевые вехи прошлого. Во многих переломных моментах истории инновации меняли образ жизни человека и определяли способы взаимодействия с миром и его восприятие. И человечество продолжит меняться по мере развития инструментов, которые открывают перед нами все больше возможностей (рис. 0.1).

Этот краткий исторически-философский экскурс нужен исключительно затем, чтобы заложить основу для понимания технологий и ИИ, а также заставить задуматься об ответственном принятии решений при реализации собственных проектов.

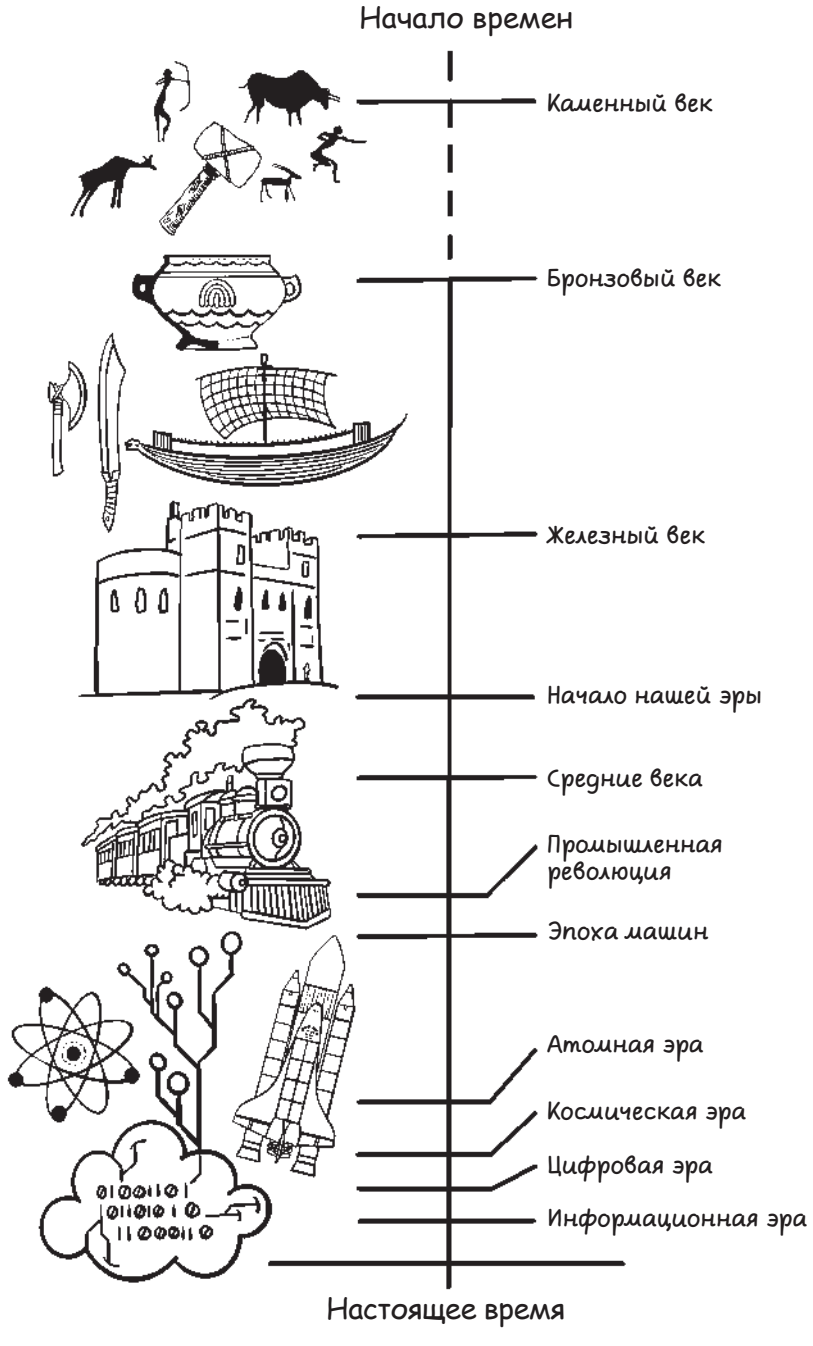


Рис. 0.1. Краткая хронология технологического развития человечества

Обратите внимание, что чем ближе к настоящему, тем быстрее происходят смены этапов развития. К самым значимым технологическим прорывам последних 30 лет можно отнести усовершенствование микросхем, широкое распространение персональных компьютеров, бум сетевых устройств и переход всех отраслей в цифровую среду, которая разрушила физические ограничения, объединив весь мир. Все это послужило причиной роста важности и актуальности искусственного интеллекта.

- Интернет объединил мир и сделал возможным собирать огромные объемы данных практически обо всем.
- Развитие вычислительного оборудования предоставило возможности обрабатывать с помощью этих данных ранее известные алгоритмы и одновременно с этим открывать новые.
- Различные отрасли ощутили потребность в применении данных и алгоритмов, чтобы принимать лучшие решения, решать более сложные задачи и оптимизировать условия жизни (что люди и делали с самого зарождения человечества).

Мы склонны рассматривать технологический прогресс в линейной перспективе. Но если обратиться к истории, можно заметить, что он есть и будет экспоненциальным (рис. 0.2). Технологические достижения появляются все чаще, потребность в освоении все новых инструментов и техник растет, но в основе всего этого лежат именно *принципы решения задач*.

В этой книге мы расскажем о фундаментальных концепциях, которые помогают решать непростые задачи; но при этом постараемся описать даже самые сложные концепции в простой форме.

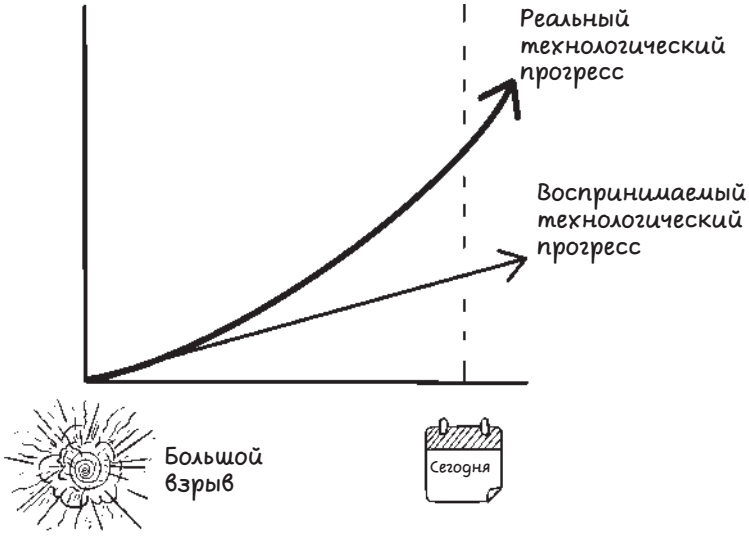


Рис. 0.2. Реальный и воспринимаемый технологический прогресс

Разные люди по-разному воспринимают понятие автоматизации. Для технического специалиста она может означать написание скриптов, которые упрощают процессы разработки, развертывания и дистрибуции ПО и помогают избежать ошибок. Для инженера — расширение производственной линии, увеличение ее пропускной способности и снижение процента брака. Фермер может считать автоматизацию средством повышения урожайности сельхозкультур с помощью тракторов и систем ирригации. Автоматизация — это любое решение, которое снижает потребность в человеческой энергии для увеличения производительности или ценности в сравнении с ручным трудом (рис. 0.3).

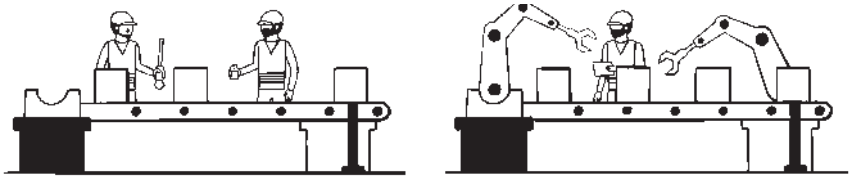


Рис. 0.3 Ручной и автоматизированный процесс

Если подумать о возможных причинах для отказа от автоматизации, то одной из наиболее существенных можно назвать способность человека выполнять задачу более качественно, с меньшим риском ошибки и более точно. Это касается задач, в которых требуется оценка и понимание сразу нескольких перспектив, когда нужен абстрактный творческий подход, либо при анализе социальных взаимодействий и природы людей.

Медсестры не просто выполняют какие-то задачи, они общаются с пациентами и заботятся о них. Исследования показывают, что человеческая забота — важная составляющая процесса лечения. Преподаватели не просто передают знания, они находят творческие способы подать, объяснить материал, а также помогать студентам с учетом их способностей, личностных качеств и интересов. Таким образом, в нашей жизни есть место и для технологий, и для людей. При современном уровне развития технологий автоматизация становится близким спутником любой профессии.

Этика, правовой аспект и ответственность

Зачем вообще в этой книге раздел про этику и ответственность, спросите вы. Дело в том, что по мере приближения к миру, в котором технологии тесно вплетены в повседневную жизнь, создатели технологий получают намного большую власть, чем могут предполагать. Даже небольшое воздействие может приводить к ощутимой отдаче. Поэтому важно, чтобы наши намерения были благими, а результаты работы не причиняли вреда (рис. 0.4).

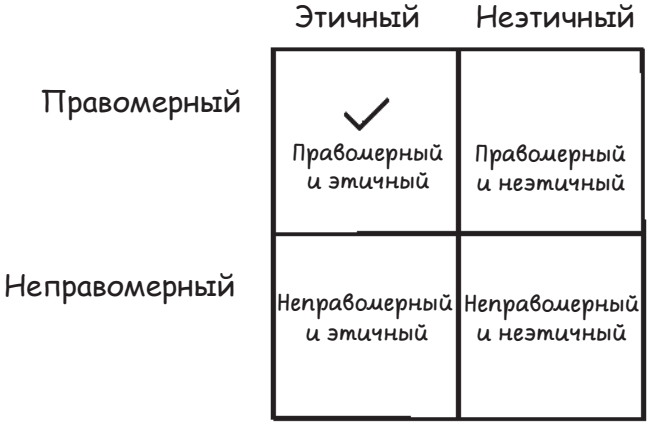


Рис. 0.4. Стремление к этичному и правомерному применению технологий

Намерение и его воздействие: понимание ви́дения и целей

При создании чего-либо, будь то новый материальный продукт, сервис или программа, всегда важен мотив. Несет ли создаваемое ПО положительный эффект или же намерения разработчика нечисты и корыстны? Учитывает ли он влияние создаваемого продукта? Бизнес всегда находит способы увеличить прибыль и распространить власть, в чем и состоит весь его смысл. Предприниматели строят стратегии нахождения лучших путей победы над конкурентами, расширения клиентской базы и сфер влияния. С учетом сказанного бизнесменам следует спросить себя, чисты ли их намерения. Причем не только с точки зрения выживания самого бизнеса, но и с позиции пользы для своих клиентов и общества в целом. Многие известные ученые, инженеры и технологи не раз указывали на необходимость управления и контроля технологий ИИ, чтобы они не применялись во вред. Мы тоже как отдельные личности обязаны действовать в рамках этики и морали и руководствоваться своими базовыми ценностями. Когда вас просят делать нечто противоречащее вашим принципам, важно озвучить эти принципы и придерживаться их.

Непредусмотренное использование: защита от злого умысла

Очень важно выявлять возможность непредусмотренного применения разработки и препятствовать этому. Хотя задача кажется очевидной и простой, на самом деле сложно предугадать, как люди будут использовать создаваемый продукт, а еще сложнее предсказать, насколько такое использование согласуется с вашими ценностями и ценностями организации.

В качестве примера можно привести громкоговоритель, изобретенный Питером Дженсеном (Peter Jensen) в 1915 году. Изначально он был назван Magnavox и использовался в Сан-Франциско, чтобы транслировать оперную музыку

широкой публике — достаточно благой способ применения. Но нацисты в Германии нашли для него другое применение: они поместили громкоговорители в общественных местах, так чтобы все жители волей-неволей слушали речи Гитлера. Поскольку избежать этого было невозможно, люди попали под влияние идей нацизма, и этот режим получил обширную поддержку в стране. Дженсен явно не предполагал, что его изобретение будет использоваться таким образом, но сделать с этим ничего не мог.

Времена изменились, и теперь у людей есть больше контроля над создаваемыми продуктами, особенно в области ПО. Невозможно заранее представить все варианты использования той или иной технологии, но почти с уверенностью можно сказать, что кто-нибудь да найдет способ применить ее не так, как задумывалось изначально, вызвав либо позитивные, либо негативные последствия. С учетом этого и мы как профессионалы в сфере технологий, и организации, с которыми мы работаем, должны препятствовать злонамеренному использованию наших разработок, насколько это возможно.

Непредусмотренные предубеждения: создание универсальных решений

В процессе создания ИИ-систем мы применяем свои знания контекстов и предметных областей. Мы также используем алгоритмы для поиска закономерностей в данных и выполнения соответствующих действий. При этом нельзя отрицать, что на нас влияют предубеждения. Предубеждение — это предрассудок в отношении человека или группы людей на основе их пола, расы, вероисповедания и других критериев. Многие из подобных предрассудков обусловлены коллективным поведением социума, историческими событиями, а также политическими и культурными взглядами на мир. Все они оказывают влияние на собираемые нами данные. И поскольку алгоритмы ИИ в итоге работают с этими предвзятыми (иначе говоря, смещенными) данными, то они и «усваивают» их. С технической точки зрения мы можем спроектировать совершенную систему ИИ, но в конечном счете с ней будут взаимодействовать люди, и наша задача — минимизировать возможное смещение (предвзятость) модели, насколько это возможно. Используемые нами алгоритмы хороши ровно в той степени, в которой хороши данные, на которых они обучаются. Понимание этих данных и контекста, в котором они применяются, — первый шаг в борьбе со смещением. Это также поможет вам лучше ориентироваться в предметной области. Сбалансированные данные с минимальным смещением позволяют разрабатывать наиболее эффективные решения.

Закон, защита частной жизни и персональных данных: важность базовых ценностей

Правовой аспект нашей работы также чрезвычайно важен. Закон определяет, что мы можем и чего не можем делать в интересах общества в целом. Многие законы были написаны во времена, когда компьютеры и интернет не являлись

важной частью нашей жизни. Поэтому сейчас вскрывается множество неоднозначных трактовок того, как мы должны разрабатывать технологии и что нам дозволено с ними делать.

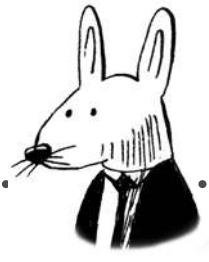
Мы рискуем своими персональными данными практически постоянно, когда взаимодействуем с компьютерами, смартфонами и другими устройствами. Мы передаем огромное количество информации о себе, иногда очень личной. Как эти данные обрабатываются и хранятся? Эти факты необходимо учитывать при создании собственных проектов. Люди должны иметь возможность выбирать, какие из их данных будут собираться, обрабатываться и храниться; как эти данные будут использоваться и кто может получить к ним доступ. По своему опыту могу сказать, что люди обычно соглашаются передавать свои данные, если это улучшает необходимые им продукты. При этом очень важно понимать, что люди гораздо более сговорчивы, когда им предоставляют выбор и уважают его.

Сингулярность: исследование неизвестного

Сингулярность — это идея создания ИИ, который будет настолько умен, что сможет самосовершенствоваться и расширяться до уровня сверхинтеллекта. Беспокойство вызывает тот факт, что люди не могут понять явление таких масштабов: явление, которое окажется способным изменить мир, каким мы его знаем, по причинам, которые нам не удается даже представить. Некоторые люди боятся, что интеллект такого уровня может увидеть в них угрозу. Другие предполагают, что сверхинтеллект может относиться к людям так, как мы относимся к муравьям. Мы не обращаем внимания на муравьев и не озадачиваемся пониманием их жизни, но, если они нас раздражают, мы от них избавляемся.

Независимо от того, в какой степени эти предположения реально отражают будущее, мы должны нести ответственность за принимаемые решения и обдумывать их, поскольку в конечном итоге они оказывают воздействие на человека, группы людей или даже на весь мир.

Благодарности

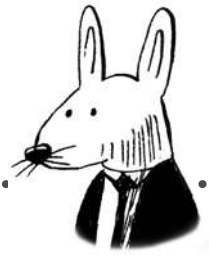


Написание этой книги стало одним из наиболее трудных, но при этом стóящих начинаний в моей жизни. Мне приходилось выискивать время, когда его не было, создавать необходимый настрой, работая одновременно с несколькими задачами, а также поддерживать мотивацию, когда меня поглощали жизненные обстоятельства. Все это мне бы не удалось, если бы не помощь удивительных людей. Этот опыт многому меня научил. Благодарю тебя, Берт Бейтс (Bert Bates), за наставничество и фантастическую редакторскую помощь. Я очень многое узнал от тебя об эффективном обучении и письменной коммуникации. Наши споры и обсуждения, а также твоя дружеская поддержка помогли мне сделать эту книгу такой, какая она есть. Каждому проекту требуется организатор, который бы держал руку на пульсе и следил за тем, чтобы все шло как запланировано. За это я благодарю моего редактора-консультанта Элешу Хайд (Elesha Hyde). Работать с тобой было абсолютным удовольствием. Ты всегда подсказывала нужное направление и делилась полезными наблюдениями о моей работе. Нам всегда нужны люди, с которыми можно обсудить идеи, а для этого лучше всего подходят друзья. Особенно хочу поблагодарить Хэнни Бринк (Hennie Brink) за то, что была надежной опорой и терпеливым слушателем. Также выражаю свою признательность Франсесу Буонотемпо (Frances Buontempo) и Кшиштофу Камычеку (Krzysztof Kamyczek) за конструктивную критику и объективную обратную связь по техническим и языковым вопросам. Ваш вклад помог мне заполнить пробелы и сделать книгу более доступной для понимания. Выражаю также благодарность Дейдре Хиам (Deirdre Hiam), моему менеджеру проектов; рецензенту Ивану Мартиновичу (Ivan Martinovic), выпускающему редактору Киеру Симпсону (Kier Simpson), а также корректору Джейсону Эверетту (Jason Everett).

В заключение искренне благодарю всех консультантов, которые вычитывали мои рукописи и давали ценную обратную связь: Андре Вайнера (Andre Weiner), Арава Агарвала (Arav Agarwal), Чарльза Соетана (Charles Soetan), Дэна Шейха (Dan Sheikh), Дэвида Джейкобса (David Jacobs), Дхивью Шивасубраманяна

(Dhivya Sivasubramanian), Доминго Салазара (Domingo Salazar), Ганди Раджана (Gandhi Rajan), Хелен Мэри Баррамеду (Helen Mary Barrameda), Джеймса Чжицзюнь Лю (James Zhijun Liu), Джозефа Фридмана (Joseph Friedman), Джузефа Мурада (Jousef Murad), Карана Ниха (Karan Nih), Кельвина Д. Микса (Kelvin D. Meeks), Кена Бирна (Ken Byrne), Кшиштофа Камычека (Krzysztof Kamyczek), Кайла Петерсона (Kyle Peterson), Линду Ристевски (Linda Ristevski), Мартина Лопеса (Martin Lopez), Питера Брауна (Peter Brown), Филипа Паттерсона (Philip Patterson), Родольфо Альендеса (Rodolfo Allendes), Теджаса Джайна (Tejas Jain) и Вейрана Дена (Weiran Deng).

О книге



Книга «Грокаем алгоритмы искусственного интеллекта» была написана и оформлена так, чтобы облегчить рядовому техническому специалисту понимание и реализацию алгоритмов ИИ, а также способы их применения. Для этого используются аналогии, практические примеры и визуальные пояснения.

Для кого эта книга

Книга — для разработчиков ПО и всех связанных с этой индустрией людей, которые хотят понять принципы и алгоритмы искусственного интеллекта на практических примерах и наглядных пояснениях, сопровождаемых теоретическими основами и математическими выкладками.

Книга рассчитана на тех, кто имеет базовое представление о таких понятиях программирования, как переменные, типы данных, массивы, условные конструкции, итераторы, классы и функции; кто имеет опыт разработки на любом языке; а также тех, кто знаком с основными математическими понятиями: переменной величиной, способами задания функций, отображением функций и данных на графиках.

Путеводитель по книге

Книга разделена на 10 глав, каждая из которых посвящена отдельному алгоритму ИИ или подходу. В начале рассматриваются основные алгоритмы и понятия, чтобы подготовить вас для изучения более сложных тем.

- *Глава 1* «Понятие искусственного интеллекта» рассказывает о фундаментальных понятиях в области данных, типов задач, категорий алгоритмов и парадигм, а также общих случаях использования алгоритмов ИИ.

- *Глава 2* «Основы поиска» содержит описание ключевых принципов структур данных и простейших алгоритмов поиска, а также примеры их использования.
- *Глава 3* «Умный поиск» переходит от простых поисковых алгоритмов к оптимальным поисковым решениям, а также к нахождению решений в соревновательной среде.
- *Глава 4* «Эволюционные алгоритмы» посвящена работам в области генетических алгоритмов, где решения задач генерируются итеративно и совершенствуются по принципу эволюции в природе.
- *Глава 5* «Продвинутые эволюционные подходы» является продолжением главы 4, но описывает продвинутые принципы, включающие способы подстройки этапов развития алгоритмов для более оптимального решения различных задач.
- *Глава 6* «Роевой интеллект: муравьи» раскрывает суть реализации роевого интеллекта и показывает, как муравьиный алгоритм использует принципы жизнедеятельности муравьев и способы решения этими насекомыми сложных задач.
- *Глава 7* «Роевой интеллект: частицы» продолжает предыдущую тему, переходя к рассмотрению задач оптимизации и способов их решения с помощью метода роя частиц, который ищет лучшие решения в обширных пространствах поиска.
- *Глава 8* «Машинное обучение» знакомит с рабочим процессом машинного обучения, включающим подготовку данных, их обработку, моделирование и тестирование (для решения регрессионных задач с помощью линейной регрессии, а также задач классификации с помощью деревьев решений).
- *Глава 9* «Искусственные нейронные сети» знакомит с понятием искусственных нейронных сетей. В ней описываются логические шаги и математические вычисления, выполняемые в ходе обучения, для нахождения закономерностей в данных и формирования прогнозов. Эта глава также определяет место нейронных сетей в машинном обучении.
- *Глава 10* «Обучение с подкреплением с помощью Q-Learning» знакомит читателей с понятием обучения с подкреплением на основе поведенческой психологии и демонстрирует алгоритм Q-Learning, с помощью которого агенты в среде путем стимуляции обучаются отличать неудачные решения от успешных.

Читать книгу рекомендуется по порядку, от начала и до конца. Все концепции излагаются последовательно. После прочтения каждой главы полезно обращаться к коду Python в репозитории, чтобы попрактиковаться и разобраться в реализации соответствующих алгоритмов.

О коде

В этой книге содержится псевдокод, который поможет понять логические принципы и идеи, лежащие в основе алгоритмов. Он доступен каждому вне зависимости от предпочитаемого языка программирования. Псевдокод — это неформальный способ описать инструкции в коде. Он более удобен для человеческого восприятия, поэтому его легче читать и понимать.

Для всех описываемых в книге алгоритмов доступны рабочие примеры кода Python в репозитории GitHub (*https://github.com/rishal-hurbans/Grokking-Artificial-Intelligence-Algorithms*). Исходный код сопровождается инструкциями по настройке и комментариями, которые будут помогать вам в ходе обучения. Рекомендуем обращаться к коду после прочтения каждой главы. Это позволит закреплять полученные знания об алгоритмах.

Исходный код Python призван показать варианты реализации этих алгоритмов. Его примеры оптимизированы ДЛЯ ОБУЧЕНИЯ, а не для ПРАКТИЧЕСКОГО использования. Код намеренно был написан в качестве вспомогательного обучающего материала. Для практических проектов рекомендуется использовать установленные библиотеки и фреймворки, так как они обычно оптимизированы для высокой производительности, прошли надлежащее тестирование и имеют профессиональную поддержку.

Форум liveBook

Приобретая книгу «Грокаем алгоритмы искусственного интеллекта», вы получаете бесплатный доступ к закрытому веб-форуму издательства Manning (на английском языке), на котором можно оставлять комментарии о книге, задавать технические вопросы и получать помощь от автора и других пользователей. Чтобы получить доступ к форуму, откройте страницу *https://livebook.manning.com/book/grokking-artificial-intelligence-algorithms/welcome/v-5/discussions*. Информацию о форумах Manning и правилах поведения на них см. на *https://livebook.manning.com/#!/discussion*.

В рамках своих обязательств перед читателями издательство Manning предоставляет ресурс для содержательного общения читателей и авторов. Эти обязательства не подразумевают конкретную степень участия автора, которое остается добровольным (и неоплачиваемым). Задавайте автору хорошие вопросы, чтобы он не терял интереса к происходящему! Форум и архивы обсуждений доступны на веб-сайте издательства, пока книга продолжает издаваться.

Дополнительные ресурсы

Исходный код для «Грокаем алгоритмы искусственного интеллекта»: *https://github.com/rishal-hurbans/Grokking-Artificial-Intelligence-Algorithms*.

Сайт автора: *https://rhurbans.com*

Об авторе

Ришал с самого детства увлекается компьютерами, технологиями и различными безумными идеями. На протяжении своей карьеры он руководил командами и проектами, занимался практической разработкой программного обеспечения, стратегическим планированием и решениями полного цикла для международных компаний. Он активно внедряет культуру прагматизма, а также пропагандирует постоянное обучение и расширение навыков в компании, сообществе и всей индустрии.

Ришал страстно увлечен бизнес-механиками и стратегиями, развитием людей и команд, а также искусственным интеллектом и философией. Он создал немало различных цифровых продуктов для повышения эффективности работы, чтобы люди и компании могли сосредоточиться на наиболее важных вещах. Помимо этого, он неоднократно выступал на международных конференциях, объясняя сложные понятия доступным языком и помогая людям совершенствовать знания и навыки.

От издательства

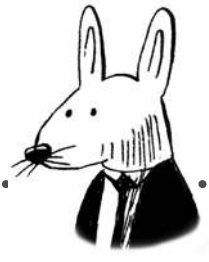
Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Понятие искусственного интеллекта

1



В этой главе

- ✓ Определение ИИ, каким мы его знаем
- ✓ Представление применимых к ИИ концепций
- ✓ Виды задач в компьютерной науке и ИИ, а также описание их свойств
- ✓ Обзор ИИ-алгоритмов, с которыми вы познакомитесь в книге
- ✓ Реальные примеры использования ИИ

Что такое искусственный интеллект?

Интеллект — это загадка, понятие, для которого не существует единого определения. Философы, психологи, ученые и инженеры по-разному определяют, что это такое и откуда происходит. Мы видим интеллект в окружающей природе, например в групповом взаимодействии живых существ. Мы также видим его проявление в способах мышления и поступках людей. Если говорить в общем, то объекты, являющиеся автономными, но при этом умеющие приспосабливаться, считаются наделенными интеллектом. *Автономные* означает, что им не требуются постоянные инструкции. А *приспосабливающиеся* подразумевает способность менять поведение в соответствии с изменениями среды или пространства задачи. При рассмотрении живых организмов и машин мы понимаем, что основой для их действий выступают данные. Данными служат образы, которые мы видим; звуки, которые мы слышим; характеристики вещей и явлений, которые нас окружают. Мы получаем все эти данные, обрабатываем и делаем выводы. Поэтому для понимания алгоритмов искусственного интеллекта очень важно понимание концепций, связанных с данными.

Определение ИИ

Некоторые люди утверждают, что мы не понимаем суть ИИ, потому что не можем определить само понятие интеллекта. Сальвадор Дали верил, что признаком интеллекта является амбиция. Он говорил: «Ум без амбиции — все равно что птица без крыльев». Альберт Эйнштейн считал, что весомым показателем интеллекта является воображение. Он говорил: «Не знание является истинным признаком интеллекта, а воображение». А вот слова Стивена Хокинга: «Интеллект — это способность адаптироваться», что подразумевает способность подстраиваться под изменения в мире. Эти три великих ума рассматривали явление интеллекта с различных позиций. До сих пор не имея единого истинного определения этого понятия, мы основываем понимание интеллекта на том, что человек является доминирующим и наиболее разумным видом на земле.

С позиции здравого смысла и для согласованности с практическими примерами из данной книги мы дадим примерное определение ИИ как синтетической системы, демонстрирующей «умное» поведение. Вместо того чтобы пытаться разобрать, что есть ИИ, а что им не является, мы будем обращаться к ИИ-подобию рассматриваемых явлений. Некая сущность может проявлять определенные признаки интеллекта, так как помогает нам решать сложные задачи, а также привносит ценность и пользу. Обычно реализации ИИ, симулирующие зрение, слух и другие органы чувств, рассматриваются как ИИ-подобные. Решения, которые способны обучаться автономно и в то же время приспосабливаться к новым данным и средам, также считаются ИИ-подобными.

Вот некоторые примеры ИИ-подобных явлений:

- Система, способная играть в разнообразные сложные игры.
- Система обнаружения раковых опухолей.
- Система, генерирующая произведения искусства на основе небольшого количества входных данных.
- Беспилотный автомобиль.

Как сказал Дуглас Хофштадтер (Douglas Hofstadter): «ИИ — это то, что еще не разработано в достаточной степени». В приведенных выше примерах беспилотные автомобили могут выглядеть как умные, потому что еще недостаточно усовершенствованы. А компьютер, выполняющий сложение чисел, еще не так давно считался умным, но теперь этим никого не удивишь.

Суть в том, что *ИИ* — это неоднозначный термин, который для разных людей, индустрий и дисциплин означает разные понятия. Представленные в этой книге алгоритмы в прошлом или настоящем были классифицированы как алгоритмы ИИ, и не имеет значения, соответствуют ли они конкретному определению ИИ. Важно то, что они решают сложные задачи.

Данные как ключевая составляющая ИИ-алгоритмов

Данные — это путь к волшебным алгоритмам, которые позволяют получать в некотором смысле магические результаты. Если данных недостаточно, они неверно выбраны или нерепрезентативны, алгоритмы работают плохо. Качество работы алгоритмов напрямую определяется качеством исходных данных. Мир заполнен информацией, и эта информация существует даже в таких формах, которые мы не способны почувствовать. Данные могут представлять значения, измеряемые численно, например текущую температуру воздуха в Арктике, количество рыбы в водоеме или ваш возраст в днях. Все эти примеры подразумевают подбор точных численных значений на основе фактов. Такие данные сложно интерпретировать с ошибкой. Температура в конкретном месте в конкретное время абсолютно верна и не подвержена смещению. Такие данные называются *количественными*.

Данные также могут представлять показания наблюдений, например запах цветка или степень чьего-либо согласия с политикой партии. Такие данные называются *качественными* и иногда трудно интерпретируются, потому что представляют не абсолютную истину, а чье-то личное восприятие истины. На рис. 1.1 показано несколько примеров окружающих нас количественных и качественных данных.

Данные — это сырые факты о вещах, поэтому их запись, как правило, не имеет смещения. Тем не менее в реальном мире люди собирают, записывают и сопоставляют данные в конкретном контексте и с конкретным понимани-

ем того, как их можно использовать. Формирование на основе данных неких осмысленных идей для ответа на определенный вопрос рождает *информацию*. Далее использование информации на основе опыта и сознательное ее применение формируют *знания*. Отчасти именно это мы и стремимся симулировать с помощью алгоритмов ИИ.

Рис. 1.1. Знакомые нам примеры данных

На рис. 1.2 показано, как количественные и качественные данные можно интерпретировать. Такие стандартизированные инструменты, как часы, калькуляторы и весы, обычно используются для измерения количественных данных, в то время как наши обоняние, вкус, слух, осязание, зрение, а также оценочные суждения служат для формирования качественных данных.

Разные люди могут по-разному интерпретировать данные, информацию и знания в зависимости от их уровня понимания соответствующей области и взгляда на мир. Этот факт влияет на качество решений, делая научный аспект в создании технологии крайне важным. Соблюдая повторяемые научные процессы сбора данных, выполнения экспериментов и регистрирования результатов, можно обеспечить их повышенную точность и находить лучшие решения задач при обработке данных с помощью алгоритмов.

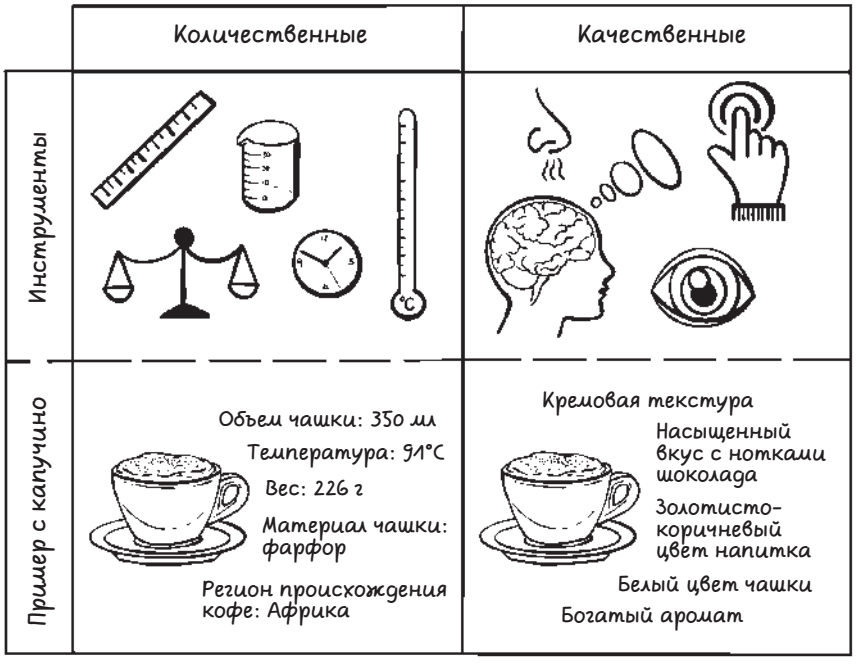


Рис. 1.2. Качественные и количественные данные

Алгоритмы как рецепты

Итак, мы знаем примерное определение ИИ и понимаем, как важны данные. Теперь надо разобраться, что именно из себя представляет алгоритм. *Алгоритм* — это набор инструкций и правил, представленных в качестве спецификации для достижения конкретной цели. Как правило, алгоритмы получают входные данные, и после нескольких этапов, на которых алгоритмы меняют их состояния, производятся выходные данные.

Даже такое простое занятие, как чтение книги, можно представить в виде алгоритма. Вот пример этапов чтения:

1. Найти книгу «Грокаем алгоритмы искусственного интеллекта».
2. Открыть книгу.
3. Пока остаются непрочтенные страницы:
   а) прочесть страницу;
   б) перелистнуть;
   в) обдумать полученные знания.
4. Подумать, как эти знания можно применить на практике.

На рис. 1.3 алгоритм представлен в виде рецепта. Входными данными служат ингредиенты и необходимый инвентарь, а выходными — блюдо, приготовленное в соответствии с инструкциями.

Алгоритмы используются в различных решениях. Например, можно реализовать видеочат с помощью алгоритмов сжатия или перемещаться по городам с помощью сим-карт на смартфоне, использующих алгоритмы построения маршрутов в реальном времени. Даже простая программа «Hello World» задействует множество алгоритмов, чтобы перевести понятный человеку язык программирования в машинный код и выполнить его инструкции в аппаратной части устройства. Алгоритмы можно встретить повсюду, нужно лишь повнимательнее присмотреться.

Рис. 1.3. Алгоритм подобен рецепту

Для более наглядной демонстрации принципа действия алгоритмов на рис. 1.4 приводится блок-схема алгоритма игры по угадыванию чисел. Компьютер генерирует в заданном диапазоне случайное число, а игрок старается это число угадать. Обратите внимание, что алгоритм содержит дискретные шаги, которые выполняют действие либо определяют решение до перехода к следующей операции.

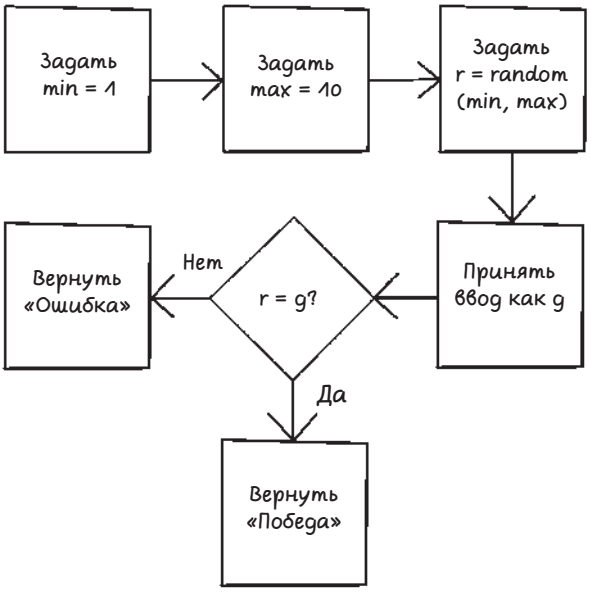


Рис. 1.4. Блок-схема игры по угадыванию чисел

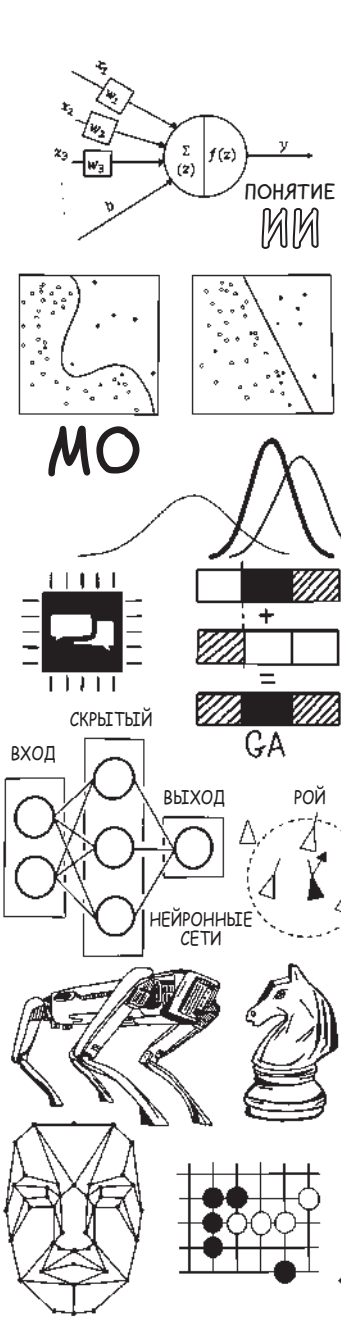
На основе нашего понимания технологии, данных, интеллекта и алгоритмов можно определить алгоритмы ИИ как наборы инструкций, которые используют данные для создания систем, демонстрирующих разумное поведение и решающих сложные задачи.

Краткая история искусственного интеллекта

Краткий обзор этапов становления индустрии ИИ поможет понять, что старые техники можно успешно совмещать с новыми идеями для решения задач инновационными способами. Сама по себе технология ИИ не нова. История наполнена мифами о механических людях и автономных «мыслящих» машинах. Оглядываясь назад, мы обнаруживаем, что стоим на плечах гигантов. Возможно, и мы сможем внести свою лепту в общий поток знаний.

Анализ прошлых достижений подчеркивает важность понимания основ ИИ. Алгоритмы, разработанные несколько десятилетий назад, чрезвычайно важны в современных реализациях ИИ. Эта книга начинается с разбора именно основополагающих алгоритмов, которые помогут вам выработать понимание, как решать задачи с их помощью и постепенно продвигаться к более интересным и современным подходам.

На рис. 1.5 приведен далеко не исчерпывающий список достижений в области ИИ — это просто небольшой набор примеров. История исполнена множества великих прорывов и открытий!



1950-е
- Введен термин «искусственный интеллект»
- Введено понятие искусственной нейросети
- Создана модель перцептрона
- Создан язык программирования LISP

1960-е
- Разработаны прогнозные модели машинного обучения
- Робот Unimate начинает работать на линии сборки автомобилей
- Разработан робот Шейки, способный анализировать задачи и совершать движения
- Издана книга, описывавшая недостатки концепции перцептрона и заставившая усомниться в ее целесообразности

1970-е
- Программа BKG выиграла в нарды (с долей везения)
- Повышается интерес к эволюционным алгоритмам
- Робот Freddy получил способность использовать зрительную систему
- Создан язык программирования Prolog

1980-е
- Создание вычислительных лисп-машин
- Возврат к потенциалу нейросетей благодаря разработке метода обратного распространения ошибки
- Повышение интереса к роевому интеллекту

1990-е
- Программа TD-Gammon демонстрирует успех обучения с подкреплением
- Работа над созданием автомобилей с автономным управлением
- Суперкомпьютер IBM Deep Blue становится чемпионом мира по шахматам
- Развитие технологий интернет-ботов и поиска

2000-е
- ИИ научился играть в шашки
- Распознавание лиц нейросетью
- Суперкомпьютер IBM Watson одержал победу в Jeopardy!
- Игровой контроллер XBox Kinect распознает движения тела
- В крупных технических корпорациях запускают умные голосовые помощники
- Программа Google Alpha Go обыграла чемпиона в игре го
- Разработка специализированного ПО на основе ИИ и устройств интернета вещей
- ИИ обнаруживает опухоли лучше врачей
- Беспилотные автомобили

Рис. 1.5. Эволюция ИИ

Типы задач и парадигмы их решения

Возможности алгоритмов ИИ очень велики, но при этом они не являются универсальным средством, способным решать любые задачи. А о каких задачах тогда идет речь? В этом разделе я расскажу о нескольких типах, которые чаще всего встречаются в компьютерной науке, и помогу сформировать о них базовое понимание. Это позволит вам выявлять похожие задачи в реальной жизни и выбирать подходящие для их решения алгоритмы.

В компьютерной науке и области ИИ для описания и классификации задач используются термины *контекст* и *цель*.

Задачи поиска: найти путь к решению

Задача поиска подразумевает ситуацию, в которой есть несколько возможных решений. Каждое из них — последовательность шагов (путь) к нужной цели. Некоторые решения содержат пересекающиеся поднаборы путей; одни оказываются более подходящими, чем другие; при этом некоторые менее затратные, а другие наоборот. «Лучшее» решение определяется конкретной решаемой задачей. Термин «менее затратные» означает менее требовательные к вычислительным ресурсам для выполнения. Пример: определение на карте кратчайшего пути между городами. Доступно множество дорог, каждая со своей протяженностью и условиями движения, но при этом некоторые дороги окажутся предпочтительными. Многие алгоритмы ИИ основаны на принципе поиска в пространстве решений.

Задачи оптимизации: найти хорошее решение

Задача оптимизации подразумевает ситуацию с множеством подходящих решений, среди которых сложно найти абсолютно лучшее. Пример — разместить чемоданы в багажнике автомобиля так, чтобы максимально полно задействовать его объем. Здесь возможно множество комбинаций, и чем эффективнее заполняется багажник, тем больше в него войдет вещей.

> ***Подумайте об этом***
>
> Так как задачи оптимизации предполагают множество решений, и поскольку эти решения существуют в различных точках пространства поиска, возникает понятие локального и глобального лучшего. *Локальное лучшее решение* — это наилучшее решение внутри определенной зоны пространства поиска, а *глобальное лучшее* — это наилучшее решение во всем пространстве. Как правило, существует много локальных лучших решений и одно глобальное. Возьмем, например, поиск лучшего ресторана. Можно найти лучший ресторан в вашем городе, но он не обязательно окажется лучшим рестораном в стране или во всем мире.

Задачи прогнозирования и классификации: обучение на закономерностях в данных

Задача прогнозирования — это задача, в которой у нас есть данные о чем-либо и требуется найти в них закономерности. Например, у нас есть данные о разных транспортных средствах (ТС) и объемах их двигателей, а также количестве потребляемого этими ТС топлива. Можем ли мы спрогнозировать потребление топлива у новой модели ТС, исходя из объема ее двигателя? Если в имеющихся данных присутствует корреляция между объемами двигателей и потреблением топлива, то прогнозирование окажется возможным.

Задача классификации похожа на задачу прогнозирования, но вместо поиска конкретного прогноза, как в случае с потреблением топлива, мы стараемся определить категорию чего-либо на основе его признаков. Например, имея данные о размере ТС, объеме его двигателя и количестве сидений, можем ли мы спрогнозировать, чем это ТС является: мотоциклом, седаном или спорткаром? Задачи классификации требуют обнаружения закономерностей в данных, которые объединяют примеры в категории. При поиске таких закономерностей важным принципом является интерполяция, потому что здесь на основе известных данных мы вычисляем их новые точки.

Задачи кластеризации: определение закономерностей в данных

Задача кластеризации включает сценарии, в которых тренды и связи выявляются из анализа данных. Примеры группируются различными способами на основе различных свойств данных. Например, при наличии информации о стоимости блюд и расположении ресторанов можно выяснить, что молодежь чаще посещает места с более низкими ценами.

Кластеризация позволяет находить связи в данных, даже когда не сформулирован точный вопрос. Это полезно для лучшего понимания данных и того, какие действия с ними можно совершать.

Детерминированные модели: один и тот же результат при каждом вычислении

Детерминированные модели — это модели, которые при получении конкретных вводных данных возвращают одинаковый результат. Например, мы ожидаем, что в полдень в одном и том же городе всегда будет светло, а в полночь темно. Конечно, этот простой пример не учитывает нестандартную продолжительность светового дня в районе полюсов.

Стохастические/вероятностные модели: потенциально разные результаты при каждом вычислении

Вероятностные модели — это модели, которые при получении конкретных вводных данных возвращают один результат из набора возможных. Такие модели обычно содержат элемент контролируемой случайности, который оказывает

воздействие на набор возможных результатов. Например, при условии дневного времени можно ожидать солнечную погоду, облачную или дождливую. Для этого времени нет фиксированного варианта погоды.

Концепции искусственного интеллекта

ИИ — это «горячая» тема, равно как машинное и глубокое обучение. Разобраться в этих различных, но в то же время схожих понятиях непросто. К тому же, в области ИИ отличия существуют и между уровнями интеллектуальности.

В этом разделе мы разберем некоторые понятия ИИ, а также представим план книги.

Начнем со знакомства с тремя уровнями ИИ, представленными на рис. 1.6.

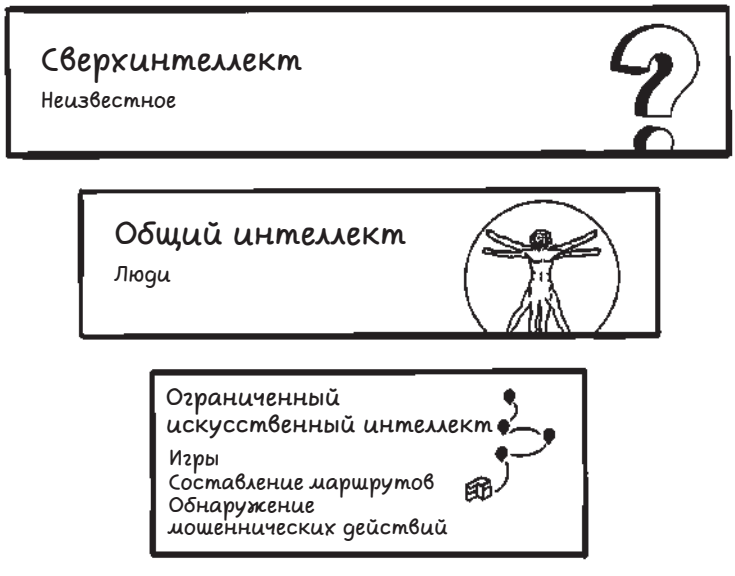


Рис. 1.6. Уровни ИИ

Ограниченный искусственный интеллект: решения конкретных задач

Системы с *ограниченным ИИ* решают задачи в конкретном контексте или предметной области. Как правило, они не способны решать задачу в одном контексте и затем применять то же знание в другом. Например, система, разработанная для понимания взаимодействия с клиентом и его модели расходов, не сможет распознавать кошек на изображении. Обычно для эффективного решения какой-либо задачи требуется специализация именно в этой области, что усложняет адаптацию к другим видам задач.

Различные системы ограниченного интеллекта можно совмещать, создавая варианты с расширенным спектром возможностей. Рассмотрим, к примеру, голосовой помощник. Эта система может понимать естественный язык, что само по себе описывает ее как узконаправленную. Однако интеграция с другими системами, например интернет-поиском и рекомендательным движком для музыки, делает ее уже более обобщенной.

Общий интеллект: человекоподобные решения

Общий интеллект подобен человеческому. Будучи людьми, мы способны обучаться на различном опыте и взаимодействиях с миром, а также применять полученные знания для решения различных задач. Например, если вы обожглись в детстве и почувствовали боль, то экстраполируете этот опыт и будете знать, что другие горячие предметы также могут причинить боль. Тем не менее общий интеллект у людей — это больше чем просто рассуждение наподобие «горячие объекты могут причинить вред». Общий интеллект подразумевает память, пространственный анализ на основе визуального восприятия, использование знаний и другие аспекты. Достичь уровня общего интеллекта у машин в обозримой перспективе невозможно, но успехи в области квантовых вычислений, обработки данных и алгоритмов ИИ могут сделать это реальностью в более отдаленном будущем.

Сверхинтеллект: великое неизвестное

Некоторые идеи сверхинтеллекта встречаются в научно-фантастических фильмах про постапокалиптические миры, где все машины связаны, способны рассуждать о выходящих за грани нашего понимания вещах и доминируют над людьми. Существует множество философских рассуждений на тему, способен ли человек изобрести нечто более разумное, чем он сам. А если даже способен, то знает ли он об этом? Сверхинтеллект — это великое неизвестное, и еще долгое время любые его определения будут не более чем гипотетическими.

Старый и новый ИИ

Иногда используются выражения «старый ИИ» и «новый ИИ». *Старый ИИ* зачастую понимается как система, где люди закодировали действия, благодаря которым алгоритм демонстрирует разумное поведение — исходя из глубокого знания задачи или путем проб и ошибок. Примером является человек, вручную создающий дерево решений, а также правила и варианты во всем этом дереве. *Новый ИИ* подразумевает создание алгоритмов и моделей, которые обучаются на данных и создают собственные правила, выполняемые настолько же точно, что и заложенные человеком, или даже точнее. Разница этих двух ИИ в том, что последний может находить в данных важные закономерности, которые человек может либо не найти совсем, либо искать очень долго. Поисковые алгоритмы

зачастую рассматриваются как старый ИИ, но понимание, как они работают, полезно для изучения более сложных подходов. В этой книге приводятся наиболее популярные алгоритмы ИИ и постепенно раскрываются лежащие в их основе концепции. На рис. 1.7 показана связь между некоторыми концепциями искусственного интеллекта.

Рис. 1.7. Категоризация концепций в области ИИ (поисковые алгоритмы, алгоритмы, вдохновленные природой, машинное обучение, глубокое обучение)

Поисковые алгоритмы

Поисковые алгоритмы полезны в решении задач, в которых для достижения цели требуется совершить несколько шагов. Например, поиск пути через лабиринт или определение наилучшего хода в игре. Эти алгоритмы оценивают будущие состояния и пытаются найти оптимальный путь к наиболее сто́ящей цели. Как правило, у нас есть слишком много возможных решений, чтобы механически перебирать их. В таких случаях поиски лучшего решения даже в небольшой области могут занимать тысячи часов вычислений. Алгоритмы поиска помогают оценить область поиска. Они используются в поисковых движках интернета, картографических приложениях и даже в игровых агентах.

Алгоритмы, вдохновленные природой

Когда мы смотрим на окружающий мир, то замечаем удивительные вещи в поведении и функционировании живых организмов. К примерам можно отнести кооперацию муравьев при сборе пищи, образование стай птиц при

их миграции, работу головного мозга, а также эволюцию организмов для выведения более сильного потомства. Через наблюдение и изучение различных явлений мы получаем знания о том, как эти органические системы работают и как простые правила приводят к появлению разумного поведения. Некоторые из этих явлений легли в основу алгоритмов, которые оказались эффективными в области ИИ, а именно эволюционных алгоритмов и алгоритмов роевого интеллекта.

Эволюционные алгоритмы построены на основе теории эволюции, сформулированной Чарльзом Дарвином. Она гласит, что в результате смешения генов и мутаций в процессе воспроизводства новые особи в популяции рождаются более приспособленными к среде обитания. *Роевой интеллект* — это группа на первый взгляд «глупых» особей, демонстрирующих разумное поведение. В книге будут рассмотрены два наиболее популярных алгоритма оптимизации этого типа: муравьиный алгоритм и алгоритм роя частиц.

Алгоритмы машинного обучения

В машинном обучении (МО) для обучения моделей на основе данных используется статистический подход. Машинное обучение включает в себя множество алгоритмов, которые можно задействовать для понимания связей в данных, принятия решений и формирования прогнозов на основе этих данных.

В этой области можно выделить три основных подхода:

- *Обучение с учителем*, то есть обучение моделей с помощью алгоритмов в условиях, когда обучающие данные содержат известные результаты для заданного вопроса. Например, определение вида фрукта на основе набора данных, включающего вес, цвет, фактуру поверхности, а также метку фрукта для каждого примера.
- *Обучение без учителя* раскрывает скрытые связи и структуры внутри данных, которые помогают подобрать для датасета подходящие вопросы. Алгоритм может находить закономерности в свойствах схожих фруктов и соответствующим образом эти свойства группировать, на основе чего формируются конкретные вопросы к данным. Эти ключевые принципы и алгоритмы закладывают основу для изучения продвинутых алгоритмов.
- *Обучение с подкреплением* основано на поведенческой психологии. Если кратко, то оно описывает схему вознаграждения агента при выполнении им эффективного действия и, наоборот, штрафа в обратном случае. Например, ребенок получает награду за хорошую отметку в дневнике и наказание за плохую, что стимулирует его приносить хорошие отметки. Обучение с подкреплением полезно для изучения способов взаимодействия программ или роботов с динамической средой. Приведем в пример робота, получившего задачу открывать двери. Если в нужный момент он этого не делает, то получает наказание, если же справляется

правильно — вознаграждение. С течением времени, после множества попыток, робот «заучивает» последовательность действий, необходимую для открывания двери.

Алгоритмы глубокого обучения

Глубокое обучение, которое является ответвлением общей дисциплины машинного обучения, включает в себя обширное семейство подходов и алгоритмов для формирования ограниченного интеллекта и стремления к достижению общего. Как правило, цель глубокого обучения — решить задачу наиболее общим способом, наподобие пространственного мышления; либо оно применяется к задачам, требующим большего обобщения, таким как компьютерное зрение и распознавание речи. К общим задачам относятся те, в решении которых преуспели люди. Например, мы можем сопоставлять визуальные паттерны практически в любом контексте. Глубокое обучение, как и МО, включает в себя обучение с учителем, без учителя, а также обучение с подкреплением. Используемые в нем подходы обычно реализуют множество слоев искусственных нейронных сетей. При использовании различных слоев интеллектуальных компонентов каждый слой решает отдельные задачи, а все вместе они реализуют более сложные задачи для достижения более значимой цели. Определение любого объекта на изображении, например, является общей задачей, но ее можно разделить на понимание цвета, распознавание форм объектов и выявление связей между этими объектами для достижения конечной цели.

Применение алгоритмов искусственного интеллекта

Сферы применения техник ИИ практически безграничны. Везде, где есть данные и задачи, которые нужно решить, потенциально уместно применение ИИ. В условиях постоянно меняющейся среды, а также изменения межличностных взаимодействий и потребностей людей и индустрий ИИ можно использовать для решения широкого спектра прикладных задач и проблем. В этом разделе описаны варианты применения искусственного интеллекта в различных областях.

Сельское хозяйство: оптимизация роста культур

Сельское хозяйство — это одна из наиболее важных сфер жизни, поддерживающая наше существование. Его задача — выращивать качественные и дешевые культуры для массового потребления. Многие фермеры занимаются этим в коммерческих масштабах, и их продукция доступна в магазинах. Разные культуры требуют разных условий для роста и развития — содержания пита-

тельных веществ в почве, влаги, климата и других факторов. Цель — собрать как можно больше урожая высокого качества за сезон, потому что определенные виды культурных растений растут только в течение конкретного времени года.

У фермеров и сельхозпредприятий за много лет накопились данные об их деятельности. С помощью этих данных мы задействуем машины для поиска закономерностей и связей среди переменных, наблюдаемых в процессе выращивания культур, чтобы определить факторы, оказывающие наибольшее влияние на урожайность. Более того, с помощью современных цифровых датчиков можно в реальном времени регистрировать погодные условия, параметры почвы, качество воды, а также скорость роста растений. При внесении всей этой информации в алгоритм можно в реальном времени генерировать рекомендации и корректировки для оптимизации роста (рис. 1.8).

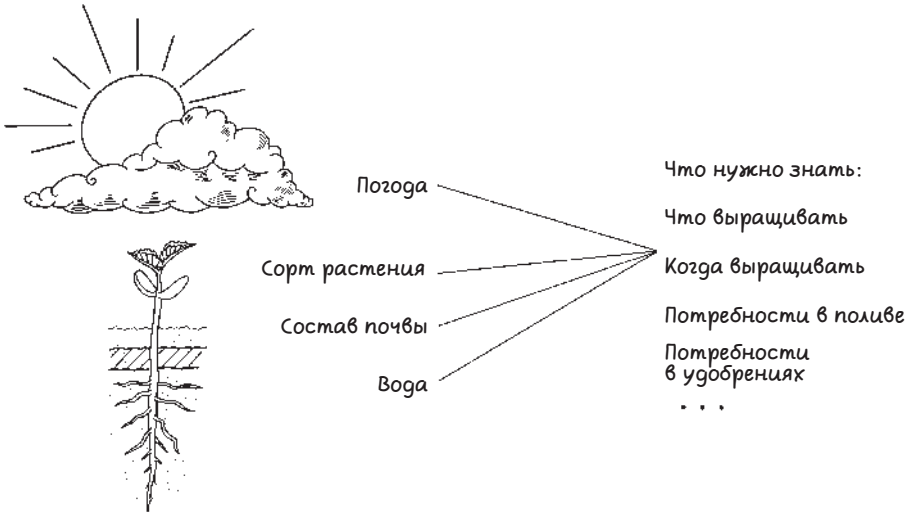


Рис. 1.8. Использование данных для оптимизации выращивания сельхозкультур

Банковская система: обнаружение мошеннических действий

Необходимость в банках стала очевидной, когда людям потребовалось найти единую валюту для обмена товарами и услугами. С течением времени банковская система несколько изменилась и начала предлагать различные варианты обслуживания: хранение денег, вклады и совершение платежей. Что не изменилось, так это стремление людей находить способы обмануть систему. Одной из самых серьезных проблем не только банковской сферы, но и большинства других финансовых учреждений является мошенничество. *Мошенничеством*

можно назвать ситуацию, когда кто-либо обманывает или совершает незаконные действия для получения выгоды. Реализуются мошеннические схемы через неучтенные лазейки в процессах либо с помощью «развода» неосведомленной стороны, которая разглашает чувствительную информацию. Так как финансовые услуги часто оказываются через интернет и персональные устройства, операции с деньгами все чаще совершаются в сети, а не лично. На основе большого объема данных о выполненных транзакциях можно в реальном времени выявлять поведенческие паттерны расходов для конкретного человека и определять, когда они становятся нетипичными. Эти данные помогают сохранить финансовым организациям огромное количество денег и защитить ничего не подозревающих клиентов.

Кибербезопасность: обнаружение атак и противодействие им

Один из интересных побочных эффектов стремительного развития интернета — кибербезопасность. Мы постоянно отправляем и получаем чувствительную информацию по сети — мгновенные сообщения, данные кредитных карт, письма и другую конфиденциальную информацию, которая, оказавшись в руках злоумышленников, может быть использована нам во вред. Тысячи серверов по всему миру получают данные, обрабатывают их и хранят. Хакеры стремятся скомпрометировать эти системы для получения доступа к данным, устройствам или даже промышленным объектам.

С помощью ИИ мы можем определять и блокировать потенциальные атаки на серверы. Некоторые крупные интернет-компании хранят данные о взаимодействии пользователей с их сервисами, включая ID пользовательских устройств, геолокацию и специфику поведения. В случае обнаружения отклонения от привычного поведения происходит, например, ограничение доступа. Некоторые компании также блокируют и перенаправляют вредоносный трафик в процессе распределенной атаки типа «отказ в обслуживании» (ddos-атаки), которая подразумевает перегрузку службы фиктивными запросами с целью вывести ее из строя или лишить доступа подлинных пользователей. Такие фиктивные запросы можно обнаруживать и перенаправлять для минимизации влияния атаки.

Здравоохранение: постановка диагноза

Здравоохранение — постоянная задача на протяжении всей истории человечества. Нам нужен доступ к диагностике и лечению различных недугов в разных точках планеты и в разные сроки, чтобы предотвратить ухудшение состояния и возможный летальный исход. При изучении диагноза можно использовать огромный багаж знаний о человеческом теле, его известных проблемах, опыте решения этих проблем, а также множество сопутствующих диагностических снимков. Традиционно для обнаружения опухолей докторам

требовалось проанализировать рентгеновские снимки, но такой подход позволял определить наличие только крупных и уже развитых новообразований. Современные достижения в области глубокого обучения позволили добиться большей точности. Теперь доктора могут выявлять рак на ранних стадиях, что позволяет назначить пациентам своевременное лечение и повысить шансы на выздоровление.

Более того, ИИ можно использовать для поиска закономерностей в симптоматике, заболеваниях, наследственных генах, месте проживания и т. д. Мы можем узнать о высокой вероятности возникновения определенного заболевания у определенного человека, что позволит подготовиться и начать лечение на самых ранних стадиях. На рис. 1.9 показан пример распознавания симптомов на снимке мозга с помощью глубокого обучения.

Рис. 1.9. Использование машинного обучения для обнаружения симптомов на снимках мозга

Логистика: построение и оптимизация маршрутов

Индустрия логистики — это огромный рынок различных видов транспортных средств (ТС), выполняющих доставку всевозможных товаров в разные точки земного шара. При этом требуется соблюдать различные требования и сроки. Представьте себе сложность планирования доставки товаров от крупного онлайн-продавца. Будь то товары широкого потребления, оборудование для строительства, детали для машин или топливо, система стремится к оптимизации процесса, стараясь обеспечить выполнение требований и минимизировать затраты.

Возможно, вы слышали о так называемой задаче коммивояжера: торговцу для выполнения работы нужно посетить несколько городов, и цель — найти кратчайший путь. Задачи логистики аналогичны, но обычно намного сложнее из-за динамического изменения окружающей среды. С помощью ИИ мы можем находить оптимальные по времени и расстоянию маршруты. Более того, можно находить наилучшие маршруты с учетом схем организации дви-

жения, строительных заграждений и даже типов дорог. Вдобавок мы можем вычислить лучший способ загрузки каждого ТС для оптимизации условий доставки.

Телекоммуникации: оптимизация сетей

Отрасль телекоммуникаций сыграла огромную роль в объединении мира. Работающие в этой сфере компании прокладывают дорогостоящие кабели, сооружают вышки и запускают спутники для создания сети, которую затем используют клиенты и организации для коммуникаций через интернет или частные сети. Обслуживание этой инфраструктуры довольно затратно. Оптимизация сети позволяет устанавливать больше подключений, давая возможность большему числу людей использовать высокоскоростные соединения. В данном случае ИИ можно использовать для мониторинга поведения в сети и оптимизации маршрутизации. Помимо этого сети ведут запись запросов и ответов. На основе собранных данных о загрузке от отдельных пользователей, участков и локальных сетей можно проводить оптимизацию. Эти же данные можно использовать, чтобы понять, где находятся пользователи и кем они являются, применяя полученную информацию для городского планирования.

Игры: создание ИИ-агентов

Игры стали популярны еще на заре становления всей индустрии ЭВМ. Можно вспомнить аркадные устройства, ТВ-приставки, а также ПК с игровыми возможностями. Сейчас ИИ стал доминировать в шахматах, нардах и других играх. Если игра относительно простая, компьютер может просчитать все возможности и принять решение быстрее человека. Не так давно компьютер смог победить чемпиона в стратегической игре го. Го предполагает простые правила для контроля территории, но обладает очень высокой сложностью в плане выбора решений для достижения победы. Компьютер не способен сгенерировать все вероятности для победы над лучшим игроком-человеком, потому что пространство поиска огромно. Вместо этого он задействует более общий алгоритм, который может «мыслить» абстрактно, продумывать стратегию и планировать ходы, ведущие к цели. Этот алгоритм уже изобретен и преуспел в победе над чемпионами мира. Он был адаптирован и для других вариантов применения, например для игр Atari и современных многопользовательских продуктов. Называется эта система Alpha Go.

Несколько исследовательских компаний разработали ИИ-агенты, способные играть в особо сложные игры лучше, чем люди и их команды. Целью этой работы было создание общих подходов, которые смогут адаптироваться к различным контекстам. На первый взгляд игровые ИИ-алгоритмы не представляют особой важности, но по факту этот подход можно эффективно применять и в других областях. На рис. 1.10 показано, как алгоритм обучения с подкреплением может научиться играть в классическую видеоигру наподобие Mario.

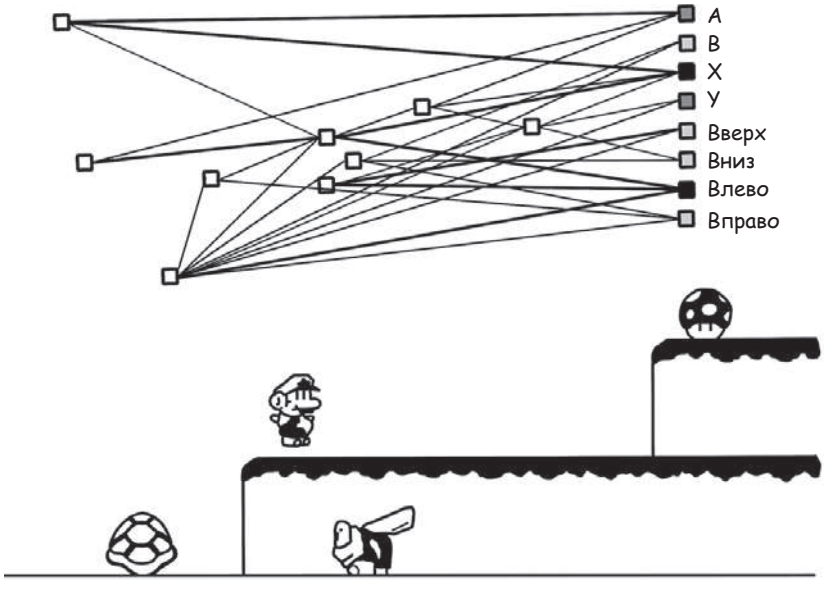


Рис. 1.10. Использование нейронных сетей для обучения алгоритма играть в игры

Искусство: создание шедевров

Уникальные, интересные художники создают множество прекрасных картин. При этом каждого художника отличает собственный стиль изображения окружающего мира. Здесь можно назвать и другие категории творцов, например композиторов, которые создают популярные произведения. В обоих случаях продукт творчества нельзя оценить количественно. Вместо этого он измеряется качественно, а именно степенью удовольствия, испытываемого людьми от его восприятия. Лежащие в основе этого факторы сложно понять и охватить, так как оценка основывается на эмоциях и чувствах.

Над созданием ИИ, генерирующего произведения искусства, работают многие исследователи. Используемый для этого принцип подразумевает обобщение. Алгоритм должен иметь широкое и обобщенное понимание предмета для создания чего-то, вписывающегося в его параметры. К примеру, ИИ Ван Гога потребовалось бы понять все работы этого мастера и извлечь из них стиль, а также «чувство», чтобы затем применить эти данные для создания новых картин. То же касается и извлечения скрытых закономерностей в таких областях, как медицина, кибербезопасность и финансы.

Теперь, когда у нас есть образное представление о том, что такое ИИ, на какие он делится категории, какие решает задачи и в каких случаях используется, мы перейдем к одной из старейших и простейших его форм, а именно алгоритмам поиска. Поисковые алгоритмы помогут нам разобраться в некоторых принципах, которые используются в более сложных алгоритмах ИИ.

Краткий обзор главы «Понятие искусственного интеллекта»

Дать определение ИИ достаточно сложно, и единого мнения на этот счет нет.

Различные прикладные решения проявляют признаки интеллектуального поведения.

ИИ включает несколько дисциплин

В решениях на основе ИИ всегда существует возможность ошибки. Учитывайте возможные последствия.

Качество и подготовка данных имеют большое значение.

У искусственного интеллекта чрезвычайно широкие возможности применения. Изобретайте!

Снимок мозга

Снимок мозга с распознаванием симптомов

Разработка новых технологий должна быть ответственной.

2 Основы поиска

В этой главе

- ✓ Понятие планирования и поиска
- ✓ Определение задач, для решения которых подходят поисковые алгоритмы
- ✓ Представление областей задач подходящим для обработки алгоритмами способом
- ✓ Понимание и проектирование базовых алгоритмов поиска для решения задач

Что мы планируем и ищем?

Если подумать о том, какие существуют составляющие интеллекта, то в первую очередь необходимо назвать способность к планированию. Мы планируем путешествие по незнакомой стране, начало нового проекта, написание функции в коде и бесчисленное множество других ситуаций. *Планирование* происходит на разных уровнях детализации и в разных контекстах, что позволяет добиваться наиболее эффективного результата (рис. 2.1).

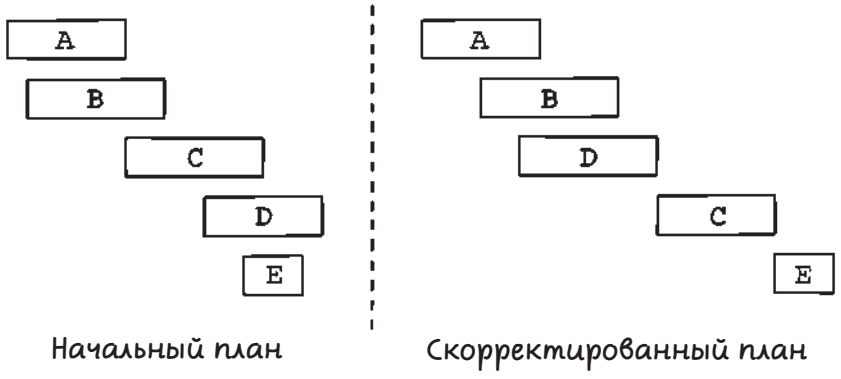


Рис. 2.1. Пример изменения планов проекта

Планы редко реализуются так, как было изначально задумано. Мы живем в постоянно меняющейся среде, поэтому просто невозможно учесть все переменные и неизвестные. Мы практически всегда отклоняемся от начального плана из-за изменений в предметной области. Нам приходится менять планы, и иногда не раз, когда мы уже прошли несколько этапов, но происходят неожиданные события, которые приходится учитывать для достижения цели. В результате фактический план, как правило, отличается от начального.

Поиск — это способ управлять планированием путем разделения плана на этапы. Например, когда мы планируем путешествие, то ищем интересующие нас маршруты, оцениваем остановки на пути и сопутствующие возможности. Мы также ищем, где жить и чем заниматься, исходя из предпочтений и бюджета. В зависимости от результатов всех этих поисков меняется и план.

Предположим, что мы задумали поездку на побережье, расстояние до которого около 500 км, и решили сделать две остановки: одну в зоопарке и вторую в пиццерии. По прибытии мы планируем переночевать в домике на берегу и поучаствовать в трех мероприятиях. Путь до места назначения займет приблизительно 8 часов. Мы также решаем сократить путь по платной автостраде, съезд на которую будет после ресторана. Но эта дорога открыта только до 2:00.

Путешествие начинается, и все идет по плану. Мы приезжаем в зоопарк и знакомимся с чудесными животными. Затем путь продолжается, и уже подходит время перекусить в ресторане, но по приезде оказывается, что он недавно

закрылся. Теперь нужно скорректировать план и найти поблизости другой ресторан или кафе, которые нам подойдут.

Проехав еще какое-то расстояние, мы находим ресторан, с удовольствием съедаем пиццу и возвращаемся на дорогу. При подъезде к частной автостраде мы понимаем, что время уже 2:20 и она закрыта. И здесь снова приходится менять план. Мы ищем объездной путь и выясняем, что он добавит к поездке еще 120 км. Это значит, что нам нужно найти другое место для ночлега еще до прибытия на побережье. Мы находим такое место и меняем маршрут. Из-за потраченного времени мы успеваем поучаствовать только в двух из трех мероприятий. Нам пришлось искать альтернативные решения, и поэтому наш первоначальный план серьезно изменился, но в конечном итоге поездка обернулась очень интересным приключением, и мы прекрасно провели время.

Этот пример показывает, как поиск используется для планирования и влияет на его итог для достижения намеченной цели. Меняются обстоятельства — меняются и цели. При этом путь к ним также неизбежно придется корректировать (рис. 2.2). Изменения в планах практически всегда внезапны и вносятся по мере необходимости.

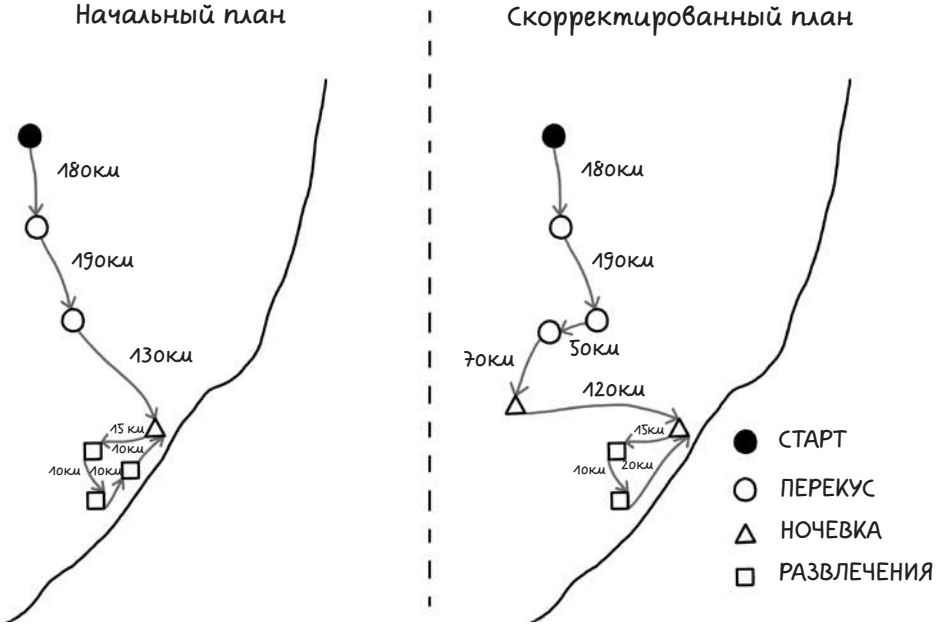


Рис. 2.2. Исходный план поездки и его измененный конечный вариант

Поиск подразумевает оценку будущих состояний на пути к цели для нахождения оптимального пути из этих состояний к намеченной точке. Эта глава посвящена различным подходам к поиску, различающимся по типу решаемых

задач. Поиск — не новый, но мощный инструмент разработки интеллектуальных алгоритмов.

Стоимость вычислений: причина использования умных алгоритмов

В программировании функции состоят из операций, и согласно принципу работы современных компьютеров разные функции требуют разного количества времени на обработку. Чем больше требуется вычислений, тем более дорогостоящей получается функция. Для описания сложности функции или алгоритма используется *нотация «О большое»*. Эта нотация моделирует количество операций, необходимых при возрастающем размере ввода. Вот некоторые примеры и связанные с ними сложности:

- *Операция, выводящая на экран надпись* `Hello World`, — это единичная операция, следовательно, стоимость вычисления равна O(1).
- *Функция, перебирающая список и выводящая каждый его элемент*, — количество операций зависит от количества элементов в списке. Стоимость будет составлять O(n).
- *Функция, сравнивающая каждый элемент списка с каждым элементом другого списка*, — стоимость операции $O(n^2)$.

На рис. 2.3 отражена различная стоимость алгоритмов. Те из них, которые требуют увеличения количества операций по мере увеличения входных данных, выполняются медленнее всех. При этом алгоритмы, для которых требуется постоянное количество операций по мере роста количества входных данных, выполняются быстрее.

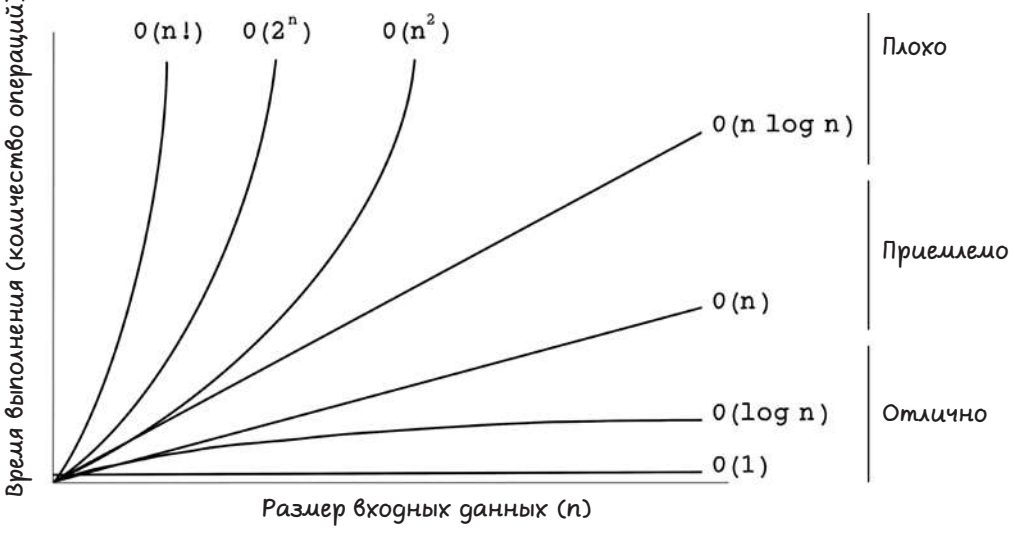


Рис. 2.3. Сложность «О большое»

Понимание того, что различные алгоритмы имеют различную вычислительную стоимость, очень важно, потому что в ее минимизации и состоит сама цель интеллектуальных алгоритмов, способных справляться с задачами быстро и эффективно. Теоретически почти любую задачу можно решить перебором возможных решений, пока не будет найдено лучшее. Но в реальности на подобные вычисления ушли бы часы, недели и даже годы, поэтому они не подходят для практического применения.

Задачи, подходящие для алгоритмов поиска

Практически любую задачу, требующую очередности решений, можно реализовать с помощью поисковых алгоритмов, которые подбираются согласно типу задачи и размеру области поиска. В зависимости от выбранного алгоритма и конфигурации можно найти хорошее или лучшее решение. Другими словами, можно найти хорошее, но не обязательно лучшее решение. Когда мы говорим «хорошее решение» или «лучшее решение», то подразумеваем степень его эффективности в достижении поставленной цели.

Возьмем, к примеру, сценарий игрока в лабиринте, который стремится найти кратчайший путь к цели. Предположим, что он находится в квадратном лабиринте размером 10 × 10 клеток, как показано на рис. 2.4. На карте лабиринта звездой отмечена интересующая нас цель, а также добавлены препятствия. Задача — найти путь к звезде, обойдя препятствия и затратив на это как можно меньше шагов. Перемещаться можно на север, юг, восток и запад, но не по диагонали.

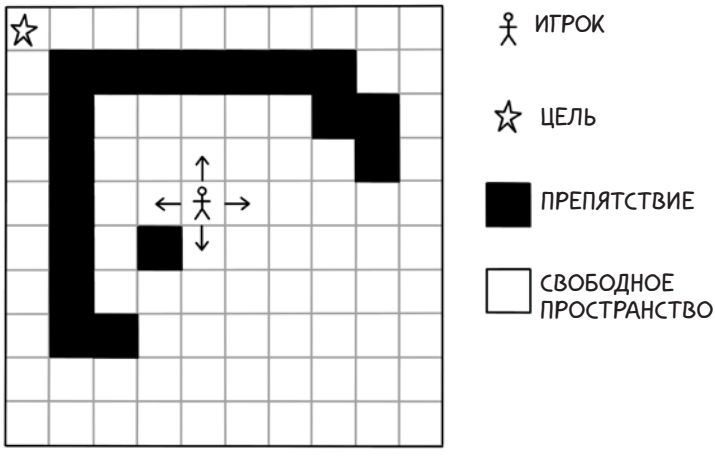


Рис. 2.4. Пример задачи с лабиринтом

Как найти кратчайший путь и обойти препятствия? Оценивая задачу с позиции человека, можно попробовать каждый вариант и подсчитать количество ходов. Таким образом, методом проб и ошибок можно найти кратчайшие пути, учитывая, что лабиринт относительно мал.

На рис. 2.5 показаны некоторые из возможных путей к цели, но обратите внимание, что в первом варианте мы ее не достигаем.

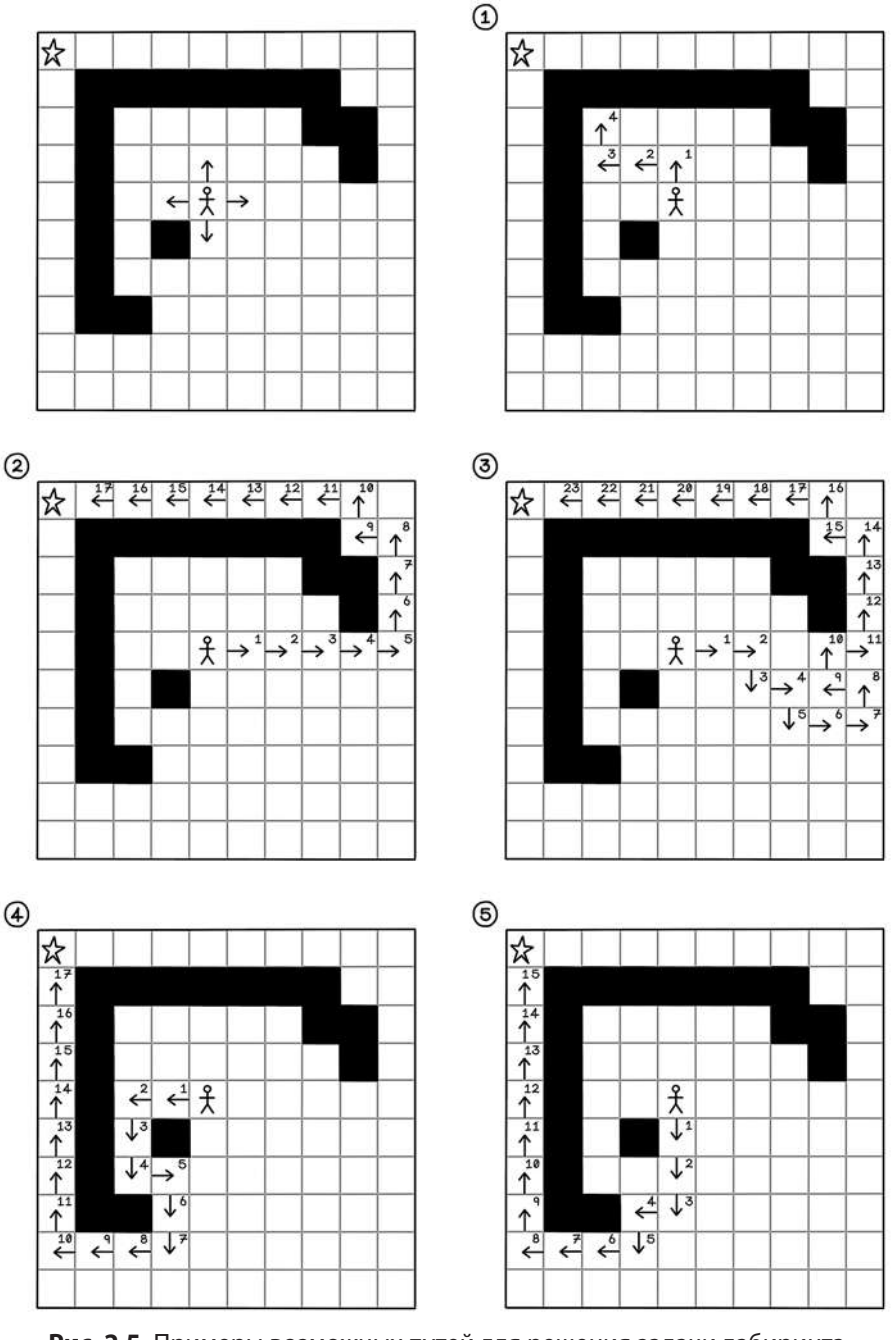


Рис. 2.5. Примеры возможных путей для решения задачи лабиринта

Рассматривая лабиринт и подсчитывая количество клеток в различных направлениях, можно найти несколько решений задачи. Чтобы найти четыре успешных варианта из неизвестного количества возможных, мы предприняли пять попыток. Для вычисления же всех возможных решений потребовалось бы очень много усилий:

- Попытка 1 не является подходящим решением. В ней выполняется 4 хода и цель при этом не достигается.
- Попытка 2 решает задачу, на что уходит 17 ходов.
- Попытка 3 решает задачу и требует совершения 23 ходов.
- Попытка 4 решает задачу снова за 17 ходов.
- Попытка 5 оказывается наилучшим доступным решением, так как требует всего 15 ходов. Несмотря на то что эта попытка оптимальна, она была найдена случайно.

Если бы лабиринт был намного больше, как, например, на рис. 2.6, то, чтобы найти кратчайший путь вручную, потребовалось бы огромное количество времени. Тут-то и пригодятся поисковые алгоритмы.

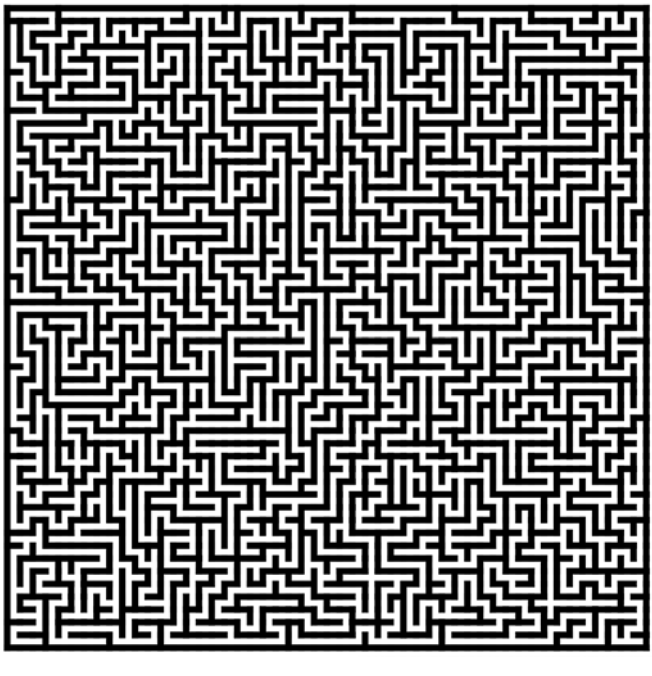


Рис. 2.6. Пример задачи с большим лабиринтом

Наше преимущество как людей состоит в том, что мы способны визуально воспринять задачу, понять ее и найти решение на основе заданных параметров.

Мы понимаем и интерпретируем данные абстрактно. Компьютер пока что не способен понимать обобщенную информацию в столь же естественной форме, как это делаем мы. Поэтому пространство задачи нужно представить в виде, который будет пригоден для вычисления и обработки поисковым алгоритмом.

Представление состояния: создание структуры для представления пространства задачи и решений

При выражении данных и информации в понятном для компьютера виде необходимо кодировать их логически, чтобы это понимание было объективным. Несмотря на то что данные будет субъективно кодировать человек, выполняющий эту задачу, необходимо предусмотреть краткий и согласованный способ их представления.

Проясним разницу между данными и информацией. *Данные* — это сырые факты о чем-либо, а *информация* — это интерпретация этих фактов, их анализ для конкретной области. Чтобы информация стала значимой, требуется контекст и обработка данных. Вот пример: каждое отдельное пройденное в лабиринте расстояние представляет собой данные, а сумма всех расстояний является информацией. В зависимости от точки восприятия, уровня детализации и желаемого результата классификация чего-либо как данных или информации может быть субъективной для контекста и человека либо команды (рис. 2.7).

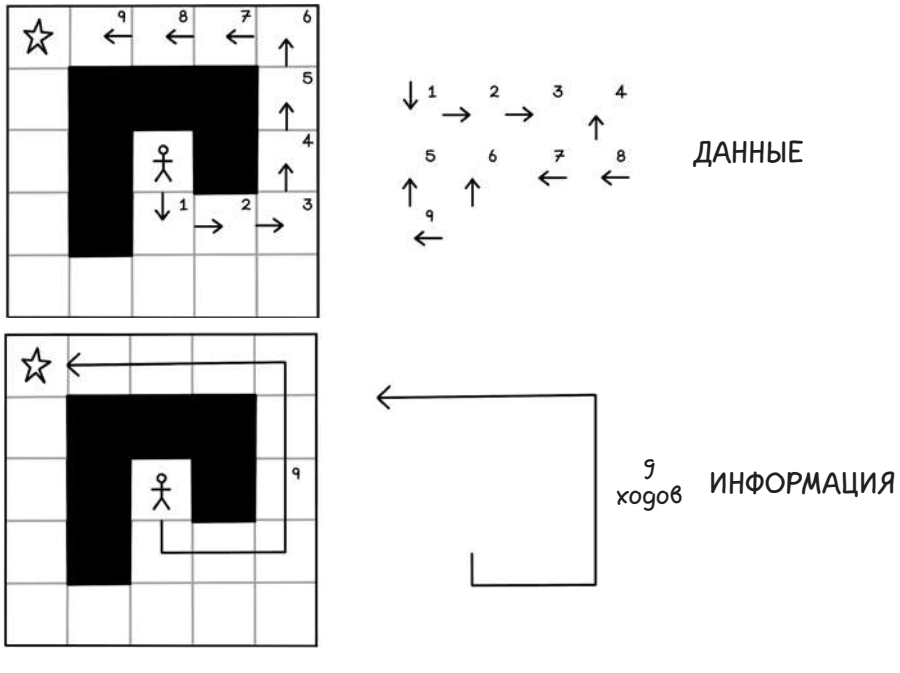


Рис. 2.7. Данные и информация

Структуры данных — это понятие компьютерной науки; с помощью структуры данные представляются в таком виде, чтобы их можно было обрабатывать с применением алгоритмов. Это абстрактный тип данных, состоящий из данных и операций, организованных определенным образом. При этом используемая структура определяется контекстом задачи и целью.

Примером такой структуры является *массив*, который представляет собой просто набор данных. Разные типы массивов обладают различными свойствами, которые делают их более подходящими для тех или иных целей. В зависимости от используемого языка программирования массив может содержать значения разных типов либо только одного типа; кроме того, в нем могут быть запрещены повторяющиеся значения. Различные виды массивов, как правило, называются тоже по-разному. Особенности и ограничения различных структур данных ведут к более эффективным вычислениям (рис. 2.8).

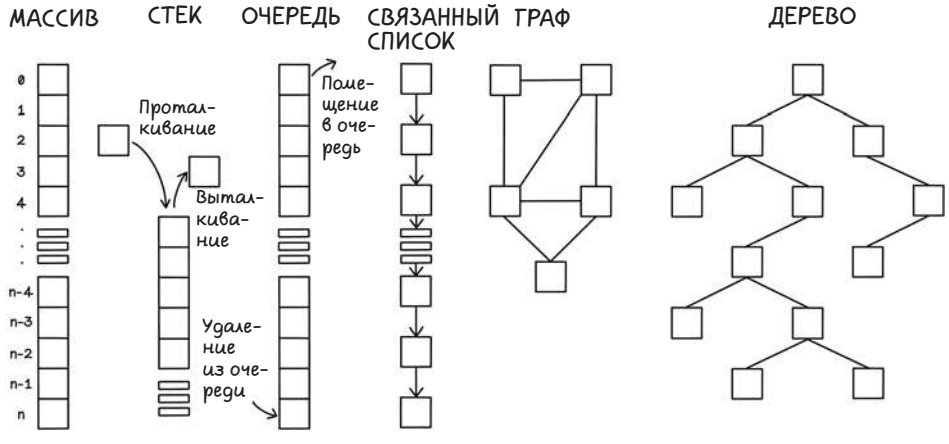
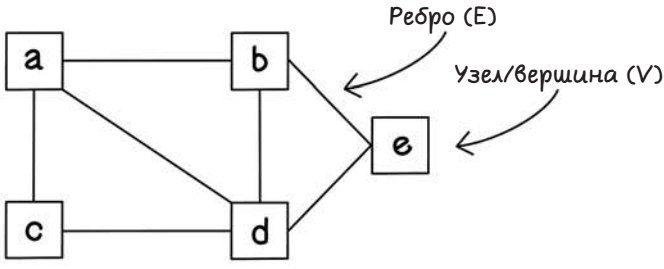


Рис. 2.8. Структуры данных, используемые с алгоритмами

Структуры дерева и графа — идеальная организация данных для поисковых алгоритмов.

Графы: представление задач поиска и решений поиска

Граф — это структура данных, содержащая несколько связанных между собой состояний. Каждое состояние называется *вершиной* (или *узлом*), а связь между двумя состояниями — *ребром*. Понятие графа происходит из теории графов в математике и используется для моделирования связей между объектами. Это полезные структуры данных, которые благодаря простоте визуального представления и логической природе легко понятны человеку, что очень удобно для обработки с помощью различных алгоритмов (рис. 2.9).



V = {a, b, c, d, e}

E = {ab, ac, ad, bd, be, cd, de}

Рис. 2.9. Обозначения, используемые для записи графов

На рис. 2.10 приведен граф поездки на побережье, о которой мы недавно говорили. Каждая остановка представлена в виде вершины, ребра между вершинами отражают точки, между которыми мы перемещались, а веса каждого ребра указывают пройденное расстояние.

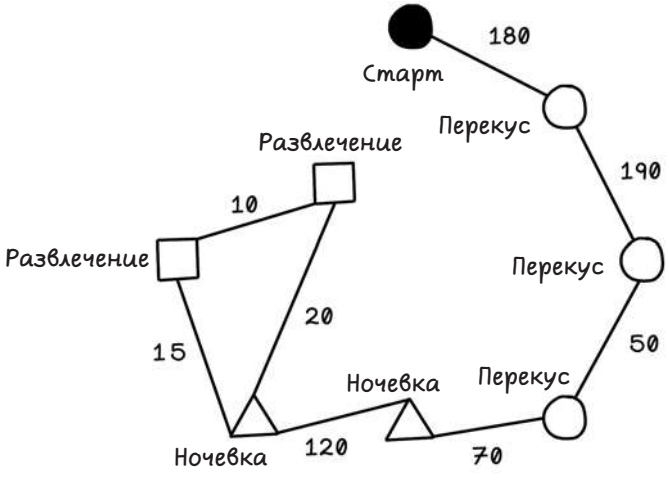


Рис. 2.10. Пример описания поездки в виде графа

Представление графа как конкретной структуры данных

Для эффективной обработки алгоритмами граф можно представить несколькими способами. По сути, его можно отразить с помощью массива массивов, указывающего связи между вершинами (рис. 2.11). Иногда полезно создать еще один массив, просто перечисляющий все вершины графа, чтобы отдельные вершины не приходилось выводить на основе связей.

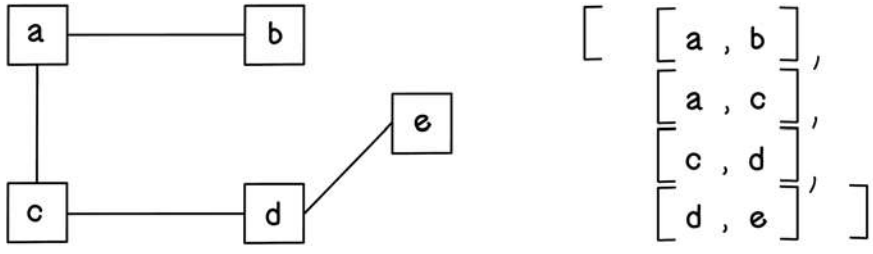
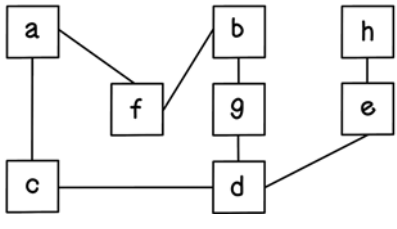


Рис. 2.11. Представление графа как массива массивов

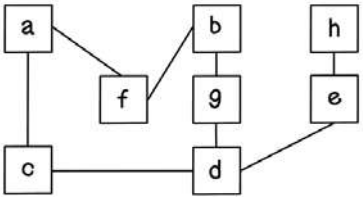
К другим вариантам представлений графов относятся матрица инцидентности, матрица смежности и список смежности. Из их названий понятно, что здесь важна смежность вершин. *Смежная вершина* — это вершина, которая напрямую связана с другой вершиной.

УПРАЖНЕНИЕ: ПРЕДСТАВЬТЕ ГРАФ В ВИДЕ МАТРИЦЫ

Как бы вы представили этот граф, используя массивы ребер?



ОТВЕТ: ГРАФ КАК МАТРИЦА




```
[ [a, c],
  [a, f],
  [b, g],
  [b, f],
  [c, d],
  [d, g],
  [d, e],
  [e, h] ]
```

Массив ребер

	a	b	c	d	e	f	g	h
a	0	0	1	0	0	1	0	0
b	0	0	0	0	0	1	1	0
c	1	0	0	1	0	0	0	0
d	0	0	1	0	1	0	1	0
e	0	0	0	1	0	0	0	1
f	1	1	0	0	0	0	0	0
g	0	1	0	1	0	0	0	0
h	0	0	0	0	1	0	0	0

Матрица смежности

Дерево: структура, используемая для представления решений поиска

Дерево — это популярная структура данных, которая симулирует иерархию значений или объектов. *Иерархия* — это упорядоченность элементов, в которой один объект связан с несколькими другими нижестоящими объектами. Дерево — это *связный ациклический граф*, то есть граф, где каждая вершина связана ребром с другой вершиной без образования цикла.

В дереве значение или объект, представленный в конкретной точке, называется *узлом*. Как правило, у деревьев есть один корневой узел с некоторым количеством дочерних узлов, которые могут содержать поддеревья. Давайте разберемся еще с несколькими терминами. В структуре, где узел связан с другими узлами, корневой узел называется *родителем*. Этот принцип применяется рекурсивно ко всему дереву. Дочерний узел может иметь собственные дочерние узлы, которые также могут содержать поддеревья. У каждого дочернего узла есть только один родительский узел. Узел, не имеющий потомков, называется листом или концевым узлом (вершиной).

У деревьев также есть общая высота. При этом уровень конкретных узлов называется их *глубиной*.

Терминология, описывающая связи внутри семейства узлов, активно используется при работе с деревьями. Так что рекомендую удерживать ее в памяти, поскольку это поможет связывать понятия в структуре данных дерева. Обратите внимание на рис. 2.12, где высота и глубина проиндексированы, начиная с корневого узла, то есть с 0.

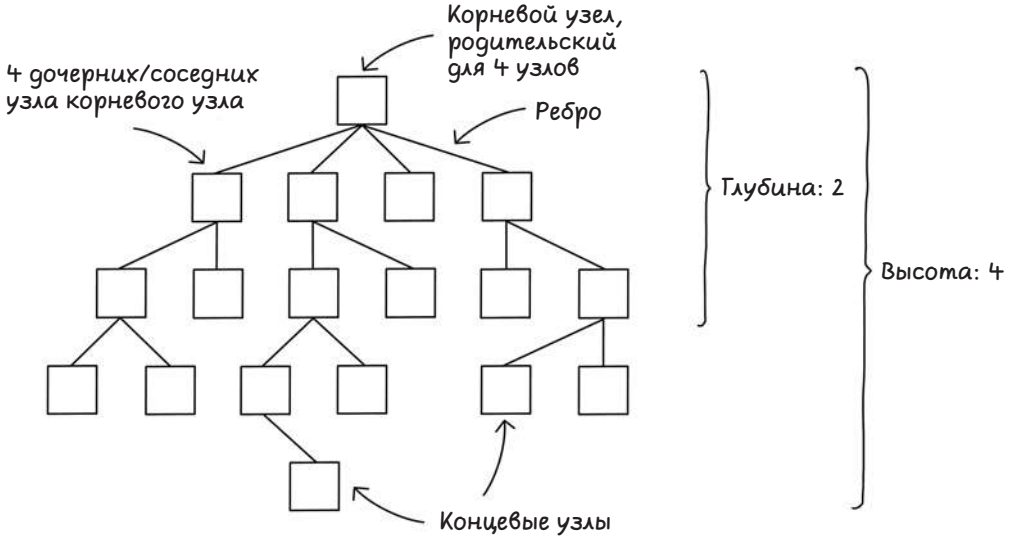


Рис. 2.12. Основные атрибуты дерева

Верхний узел называется *корневым*. Узел, непосредственно связанный с одним или более узлом, называется *родительским*. Узлы, связанные с родительским, называются *дочерними* **или** *соседними*. Узлы, имеющие одного родителя, называются *братьями*. Связь между двумя узлами называется *ребром (или ветвью)*.

Путь — это последовательность узлов и связывающих их ребер, которые не соединены напрямую. Узел, связанный с другим узлом по исходящему от корневого узла пути, называется *потомком*. И наоборот, узел, связанный с другим узлом через путь, ведущий к корню дерева, называется *предком*. Узел, не имеющий потомков, называется *концевым узлом (листом)*. Для описания количества потомков узла используется термин *степень*, из чего следует, что степень концевого узла равна 0.

На рис. 2.13 показан путь от начальной точки до цели в задаче с лабиринтом. Этот путь содержит девять узлов, представляющих совершаемые ходы.

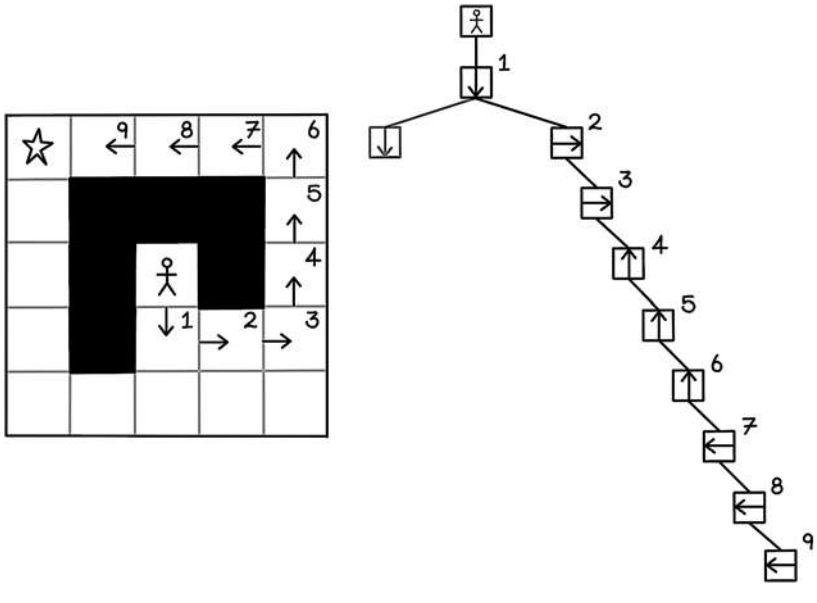


Рис. 2.13. Решение задачи с лабиринтом, представленное в виде дерева

Деревья — это фундаментальная структура данных для поисковых алгоритмов, к которым мы перейдем далее. Для более эффективных вычислений и решения конкретных задач применяется также сортировка алгоритмов. Если вы хотите узнать о ней больше, обратитесь к книге *«Grokking Algorithms»* (Manning Publications)[1].

[1] *Бхаргава А.* «Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих». СПб., Издательство «Питер».

Неинформированный поиск: слепой поиск решений

Неинформированный поиск известен также как *слепой поиск* или *поиск методом грубой силы (брут-форс)*. В неинформированных поисковых алгоритмах отсутствует информация о пространстве решения задачи, помимо представления самой задачи, которое обычно является деревом.

Подумайте, например, как вы изучаете что-то новое. Кто-то берет несколько разных тем и изучает основы каждой, а другие выбирают одну узкую тематику и углубляются во все ее подразделы. В этом заключается разница между поиском в ширину (breadth-first search, BFS) и поиском в глубину (depth-first search, DFS). *Поиск в глубину* подразумевает исследование конкретного пути от начальной точки вглубь до достижения цели. *Поиск в ширину*, в свою очередь, исследует все возможные варианты на определенной глубине пути, после чего переходит на следующий уровень и повторяет перебор вариантов.

Вернемся к лабиринту (рис. 2.14). В попытке найти оптимальный путь к цели введем следующее простое ограничение, которое позволит предотвратить застревание в бесконечном цикле и формирование циклов в дереве: *игрок не может перемещаться в клетку, которую он уже занимал*. Поскольку неинформированные алгоритмы перебирают каждый возможный вариант каждого узла, зацикливание приведет к фатальному сбою.

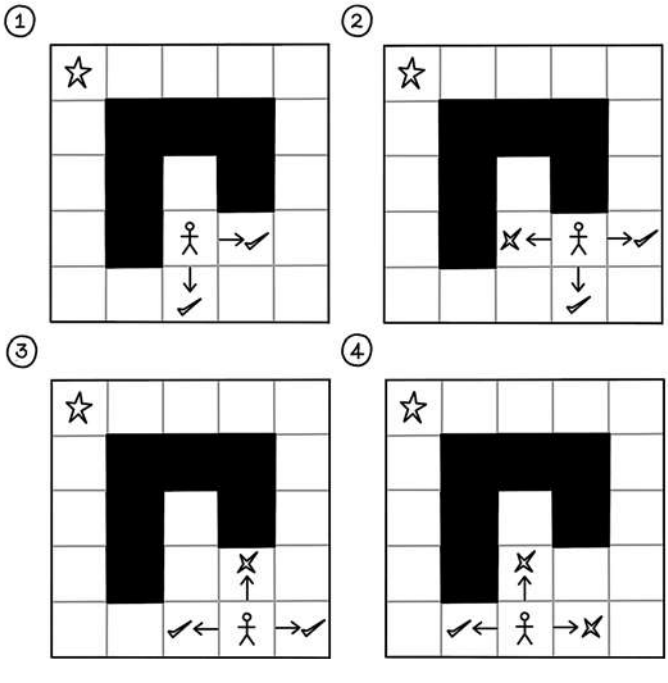


Рис. 2.14. Ограничение для задачи с лабиринтом

Заданное ограничение исключает возникновение циклов на пути к цели. Но при этом оно может вызвать проблему, если в другом лабиринте существуют другие правила, по которым подобное перемещение необходимо, чтобы найти лучшее решение.

На рис. 2.15 представлены все возможные пути и все доступные варианты. Это дерево содержит семь путей, которые ведут к цели, и один, ведущий к неверному решению. При этом работает условие запрета на возврат к ранее занятой клетке. Для такого небольшого лабиринта можно изобразить все варианты сразу. Тем не менее задача поискового алгоритма — выводить деревья поиска поочередно, поскольку генерировать все дерево возможностей сразу неэффективно ввиду высоких вычислительных затрат.

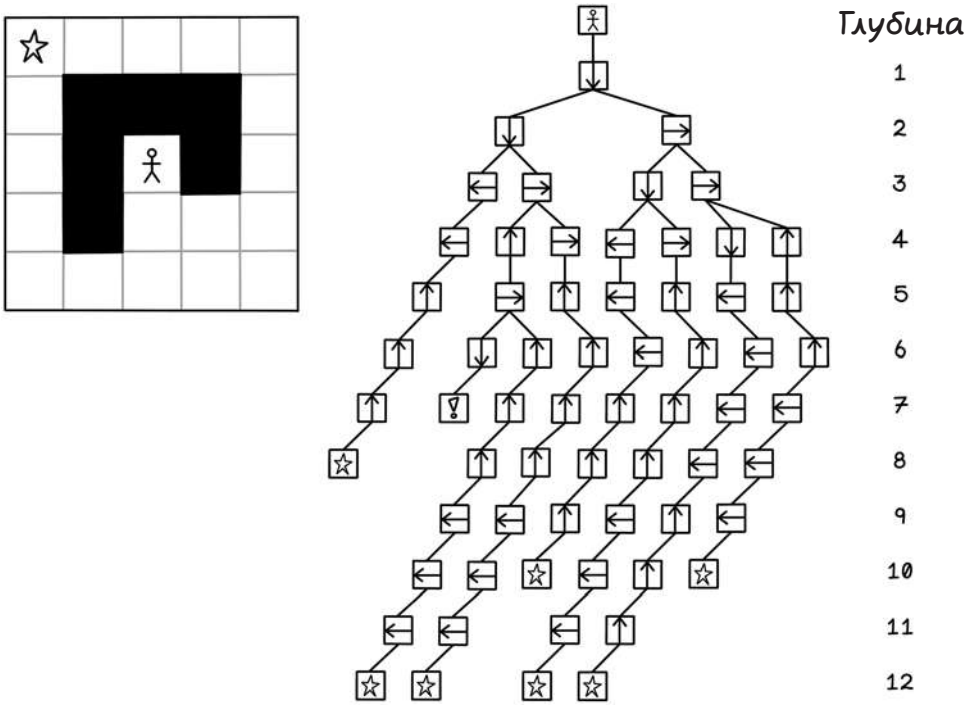


Рис. 2.15. Все возможные варианты перемещения, представленные в виде дерева

Также важно отметить, что термин *посещение* используется для обозначения разных понятий. Игрок посещает клетку в лабиринте. Алгоритм также посещает узлы в дереве. Порядок выбранных игроком шагов будет влиять на порядок посещения узлов дерева. В примере с лабиринтом приоритетным порядком перемещения является север, юг, восток, а затем запад.

Теперь, когда мы разобрали лежащие в основе организации деревьев принципы и пример лабиринта, рассмотрим, как алгоритмы поиска могут генерировать деревья, которые ищут лучший путь к цели.

Поиск в ширину: смотреть вширь, прежде чем смотреть вглубь

Поиск в ширину — это алгоритм, используемый для исследования или генерации дерева. Он начинает поиск от конкретного узла, называемого *корнем*, и исследует каждый узел на глубине корня, после чего переходит к узлам на следующем уровне. По сути, он посещает все дочерние узлы на конкретной глубине, прежде чем перейти к следующему уровню узлов-потомков, повторяя этот процесс вплоть до нахождения *целевого* конечного узла.

Алгоритм поиска в ширину лучше всего реализовывать с помощью очереди «первым вошел — первым вышел» (FIFO, first in — first out), в которой обрабатываются узлы на текущей глубине дерева, а их дочерние узлы помещаются в очередь для последующей обработки. Именно такой порядок обработки нам и нужен при реализации этого алгоритма.

На рис. 2.16 приводится блок-схема, отражающая последовательность шагов алгоритма поиска в ширину.

Ниже приводятся пояснения к каждому шагу процесса:

1. *Добавить в очередь корневой узел.* Алгоритм поиска в ширину лучше всего реализуется с помощью очереди. Объекты обрабатываются в той последовательности, в какой добавляются в очередь. Этот процесс еще известен как «первым вошел — первым вышел». Первый шаг — это добавление в очередь корневого узла, который будет представлять начальную позицию игрока на карте.
2. *Отметить корневой узел как посещенный.* После добавления корневого узла в очередь для обработки он отмечается как посещенный, что исключает его повторное посещение.

3. *Очередь пуста?* Если очередь оказывается пуста (спустя множество итераций все узлы были обработаны) и на 12-м шаге алгоритма не был возвращен конечный путь, значит, путь к цели отсутствует. Если же в очереди остались узлы, алгоритм может продолжить ее поиск.
4. *Вернуть "No path to goal" («Путь к цели отсутствует»).* Это сообщение — возможный выход из алгоритма в случае отсутствия пути к цели.
5. *Удалить текущий узел из очереди.* Извлекая следующий объект из очереди и рассматривая его как текущий узел, изучить его возможности. В начале выполнения алгоритма текущим узлом является корневой узел.
6. *Получить следующего соседа текущего узла.* Совершение следующего возможного хода в лабиринте, для чего определяются доступные направления движения: на север, юг, восток или запад.
7. *Посещался ли этот сосед?* Если текущий сосед не посещался, значит, он еще не изучен и может быть обработан.
8. *Отметить соседа как посещенного.* Этот шаг указывает, что соседний узел был посещен.
9. *Установить текущий узел как родителя соседа.* Установка узла-источника как родителя текущего соседа. Этот шаг важен для отслеживания пути от текущего соседа к корневому узлу. С позиции карты источник — это клетка, из которой игрок перемещается, а текущий сосед — это клетка, куда он перемещается.
10. *Добавить соседа в очередь.* Соседний узел ставится в очередь, что позволяет далее исследовать его потомков. Такой механизм очереди позволяет обрабатывать узлы на каждом уровне глубины в нужном порядке.
11. *Цель достигнута?* Этот шаг определяет, содержит ли текущий сосед искомую алгоритмом цель.
12. *Вернуть путь к данному соседу.* Поочередно ссылаясь на родителя текущего соседа, затем его родителя и т. д., описывается путь от цели к корню. Корневой узел при этом окажется узлом без родителя.
13. *У текущего узла еще есть соседи?* Если от текущего узла возможны еще ходы — перейти к шагу 6 и сделать ход.

Теперь рассмотрим, как все это должно выглядеть в простом дереве. Обратите внимание, что по мере исследования дерева и добавления узлов в очередь FIFO эти узлы обрабатываются в нужном порядке, устанавливаемом очередью (рис. 2.17–2.18).

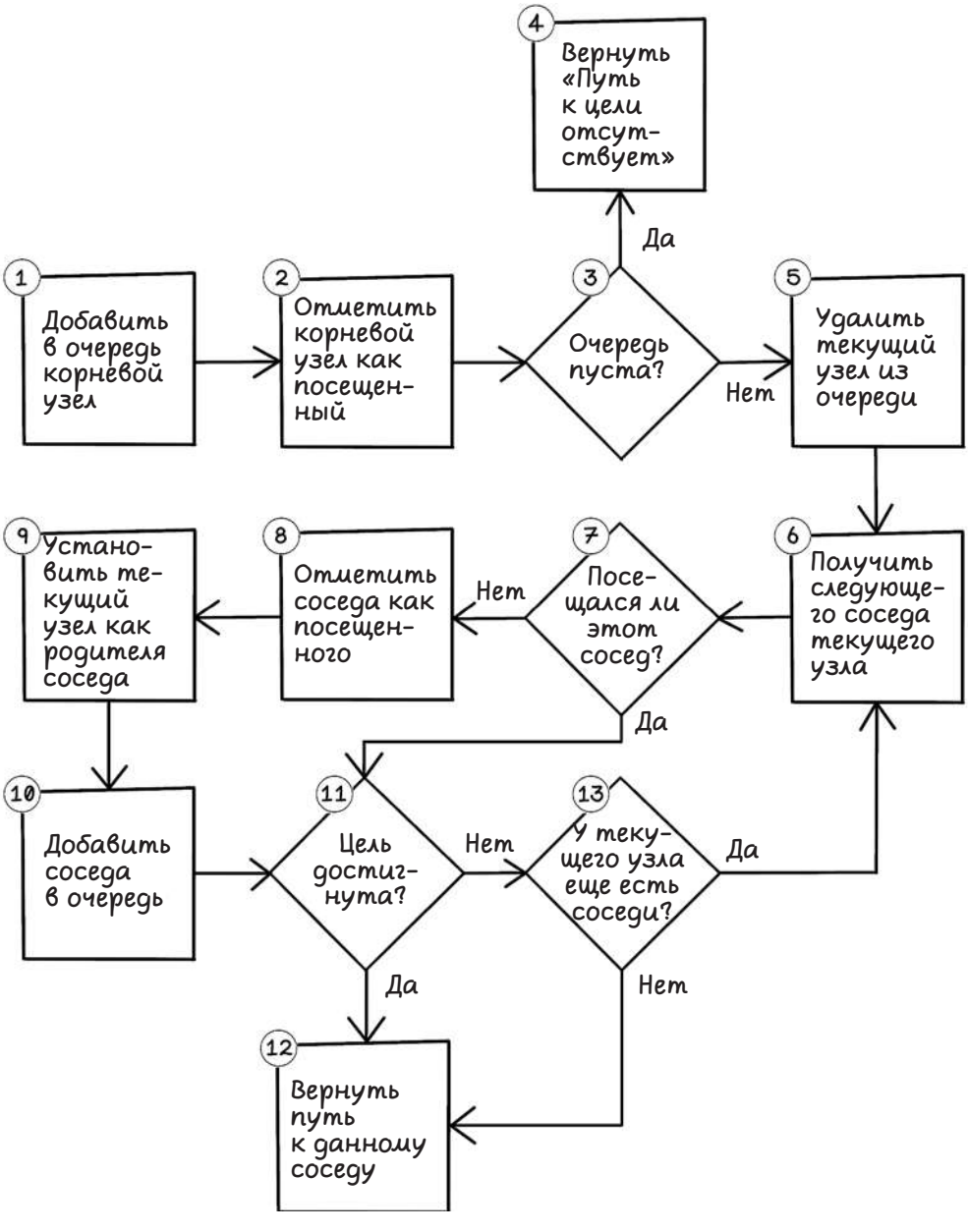


Рис. 2.16. Поток алгоритма поиска в ширину

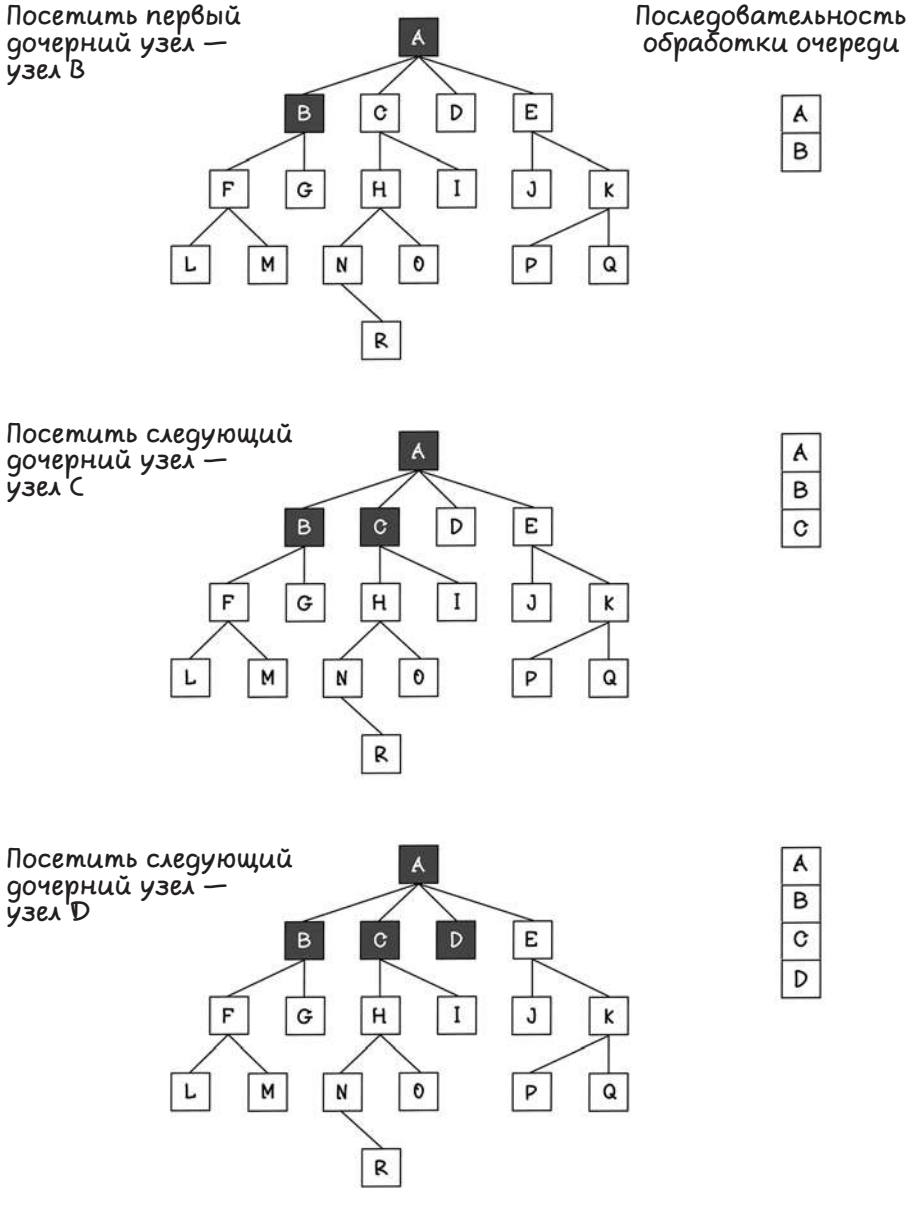


Рис. 2.17. Последовательность обработки дерева поиском в ширину (часть 1)

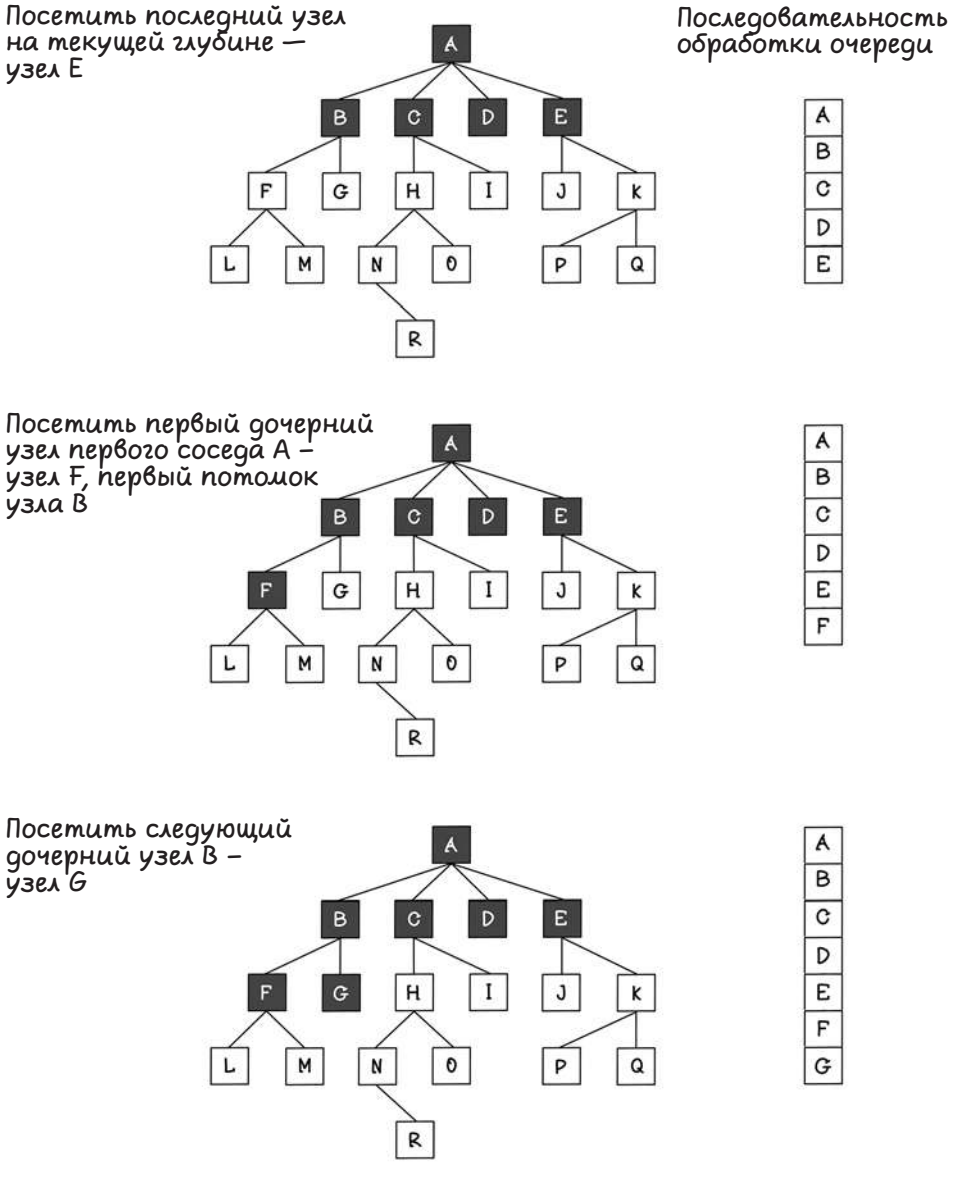
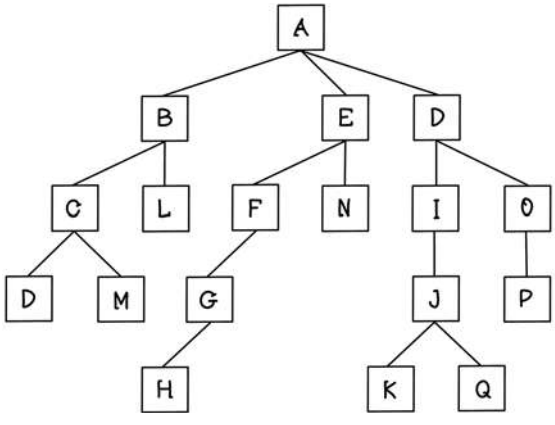


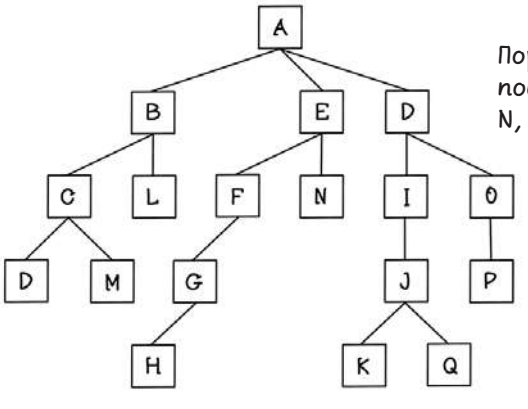
Рис. 2.18. Последовательность обработки дерева поиском в ширину (часть 2)

УПРАЖНЕНИЕ: НАЙДИТЕ ПУТЬ К РЕШЕНИЮ

Каким будет порядок исследования узлов следующего дерева при поиске в ширину?



ОТВЕТ: ПУТЬ К РЕШЕНИЮ



Порядок исследования узлов при поиске в ширину: A, B, E, D, C, L, F, N, I, O, D, M, G, J, P, H, K, Q

В примере с лабиринтом алгоритму нужно понять текущую позицию игрока, оценить все возможные направления и повторить эту логику для каждого направления, пока не будет достигнута цель. Таким образом, алгоритм генерирует дерево с единственным путем к цели.

Важно понять, что с помощью посещений происходит генерация узлов в дереве. Мы просто находим через этот механизм связанные узлы.

> ***Заметка***
>
> В примере с лабиринтом все алгоритмы поиска завершаются при обнаружении решения. Можно позволить им находить несколько решений, внеся небольшую доработку, но лучше всего вести поиск именно одного пути, поскольку перебор всех возможных вариантов будет дорогостоящей процедурой.

Каждый путь к цели состоит из серии ходов. Количество ходов является расстоянием до цели по этому пути, которое мы будем называть *стоимостью*. Количество ходов также равняется количеству посещаемых на пути узлов, начиная от корневого и заканчивая концевым, в котором находится цель. Алгоритм перемещается вниз по дереву, переходя все глубже и глубже, пока не находит цель. Затем в качестве решения он возвращает первый найденный путь, который его к ней привел. При этом может существовать и более оптимальный маршрут, но поскольку поиск неинформированный, нет гарантии, что будет найден именно он.

На рис. 2.19 показана генерация дерева в процессе перемещения по лабиринту. Так как для этого используется поиск в ширину, каждая глубина генерируется полностью, и только после этого происходит переход на следующий уровень (рис. 2.20).

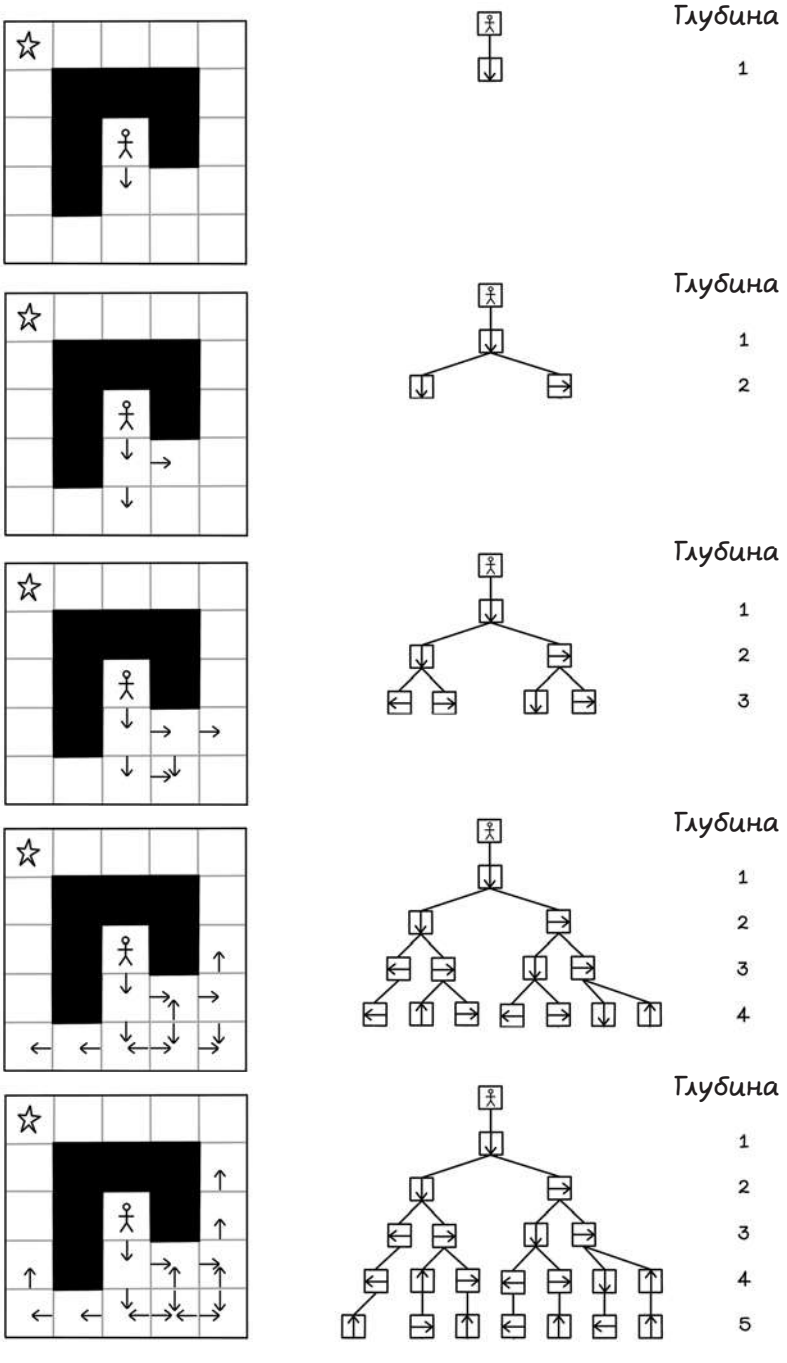


Рис. 2.19. Генерация дерева при перемещении по лабиринту с помощью поиска в ширину

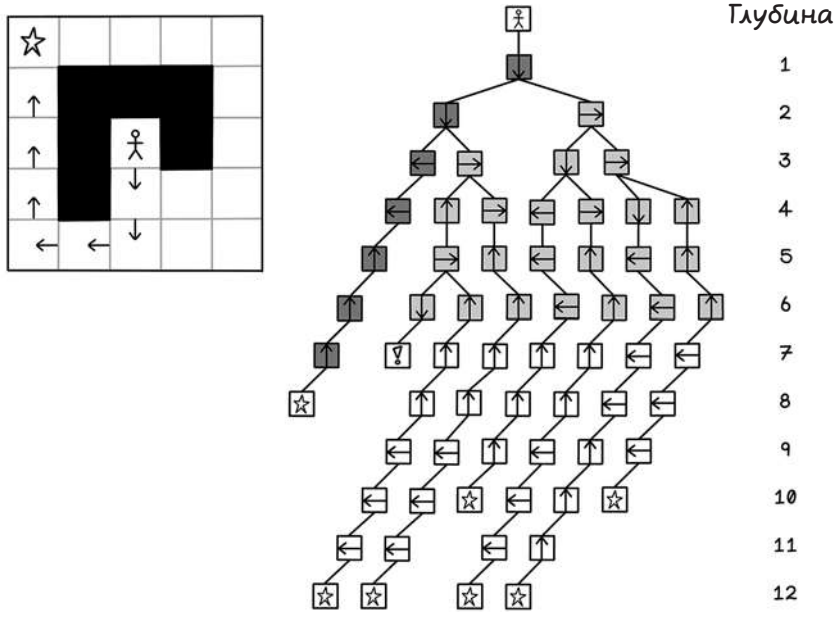


Рис. 2.20. Узлы, посещенные во всем дереве на момент завершения поиска

Псевдокод

Как уже говорилось, алгоритм поиска в ширину использует очередь, генерируя по одному уровню глубины за раз. Наличие структуры для хранения посещенных узлов очень важно для предотвращения зацикливания. Кроме того, чтобы определить путь от начальной позиции к цели, важно устанавливать родителя для каждого узла:

```
run_bfs(maze, current_point, visited_points):
  let q equal a new queue
  push current_point to q
  mark current_point as visited
  while q is not empty:
    pop q and let current_point equal the returned point
    add available cells north, east, south, and west to a list neighbors
    for each neighbor in neighbors:
      if neighbor is not visited:
        set neighbor parent as current_point
```

```
        mark neighbor as visited
        push neighbor to q
        if value at neighbor is the goal:
          return path using neighbor
return "No path to goal"
```

Поиск в глубину: обход дерева в глубину, а затем — в ширину

Поиск в глубину — это еще один алгоритм, используемый для перемещения по дереву или генерации в нем узлов и путей. Данный алгоритм рекурсивно исследует пути от первого потомка определенного узла до тех пор, пока не достигнет самого удаленного концевого узла. После этого он возвращается и исследует другие пути к концевым узлам через ранее посещенных потомков. На рис. 2.21 показан общий поток этого алгоритма.

Рассмотрим все эти шаги подробнее:

1. *Добавить корневой узел в стек.* Алгоритм поиска в глубину можно реализовать с помощью стека, в котором последний добавленный объект обрабатывается первым. Этот процесс иначе называется «последним пришел — первым ушел» (LIFO, last in — first out). Первым шагом будет добавление в стек корневого узла.
2. *Стек пуст?* Если стек пуст и на шаге 8 никакой путь не возвращался, значит, маршрут к цели отсутствует. Если же в стеке остаются узлы, алгоритм может продолжить поиск.
3. *Вернуть «No path to goal» («Путь к цели отсутствует»).* Это сообщение — возможный выход из алгоритма в случае отсутствия пути к цели.
4. *Извлечь из стека узел на место текущего.* Извлекая следующий объект из стека и рассматривая его как текущий узел, изучить его возможности.
5. *Текущий узел посещался?* Если текущий узел не посещался, значит, он еще не исследован и может быть обработан.
6. *Отметить текущий узел как посещенный.* Этот шаг помечает узел как посещенный, предотвращая ненужное повторение его обработки.
7. *Цель достигнута?* Этот шаг определяет, содержит ли текущий узел искомую алгоритмом цель.
8. *Вернуть путь к текущему узлу.* Поочередно ссылаясь на родителя текущего узла, затем его родителя и т. д., описывается путь от корня к конечной цели. У корневого узла в этом случае родитель будет отсутствовать.
9. *У текущего узла еще есть сосед?* Если от текущего узла есть другие возможные ходы, очередной ход добавляется в стек для обработки. В противном случае алгоритм переходит к шагу 2, где начинается обработка следу-

ющего объекта в стеке, если стек содержит объекты. Принцип стека LIFO позволяет алгоритму обрабатывать все узлы вплоть до концевого узла, прежде чем возвращаться для обхода других потомков корневого узла.

10. *Установить текущий узел как родитель соседа*. Установить узел-источник как родитель текущего соседа. Этот шаг важен для отслеживания пути от текущего соседа к корневому узлу. На карте источник — это клетка, из которой перемещается игрок, а текущий сосед — клетка, куда он перемещается.
11. *Добавить соседа в стек*. Соседний узел добавляется в стек для дальнейшего исследования. Опять же, этот механизм стека позволяет обрабатывать самые глубокие узлы, после чего переходить к менее глубоким.

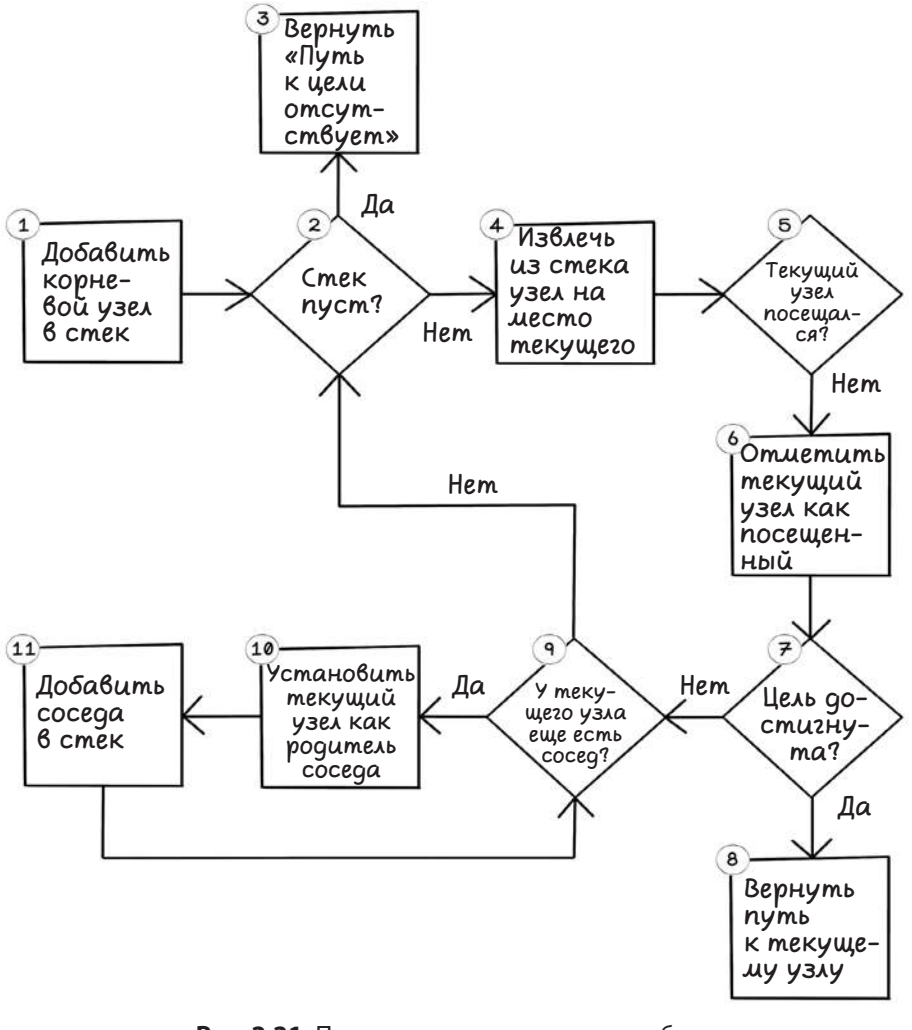


Рис. 2.21. Поток алгоритма поиска в глубину

На рис. 2.22 и 2.23 показано, как стек LIFO используется для обхода узлов в требуемом для поиска в глубину порядке. Обратите внимание, как узлы проталкиваются в стек по мере углубления, а затем выталкиваются из стека.

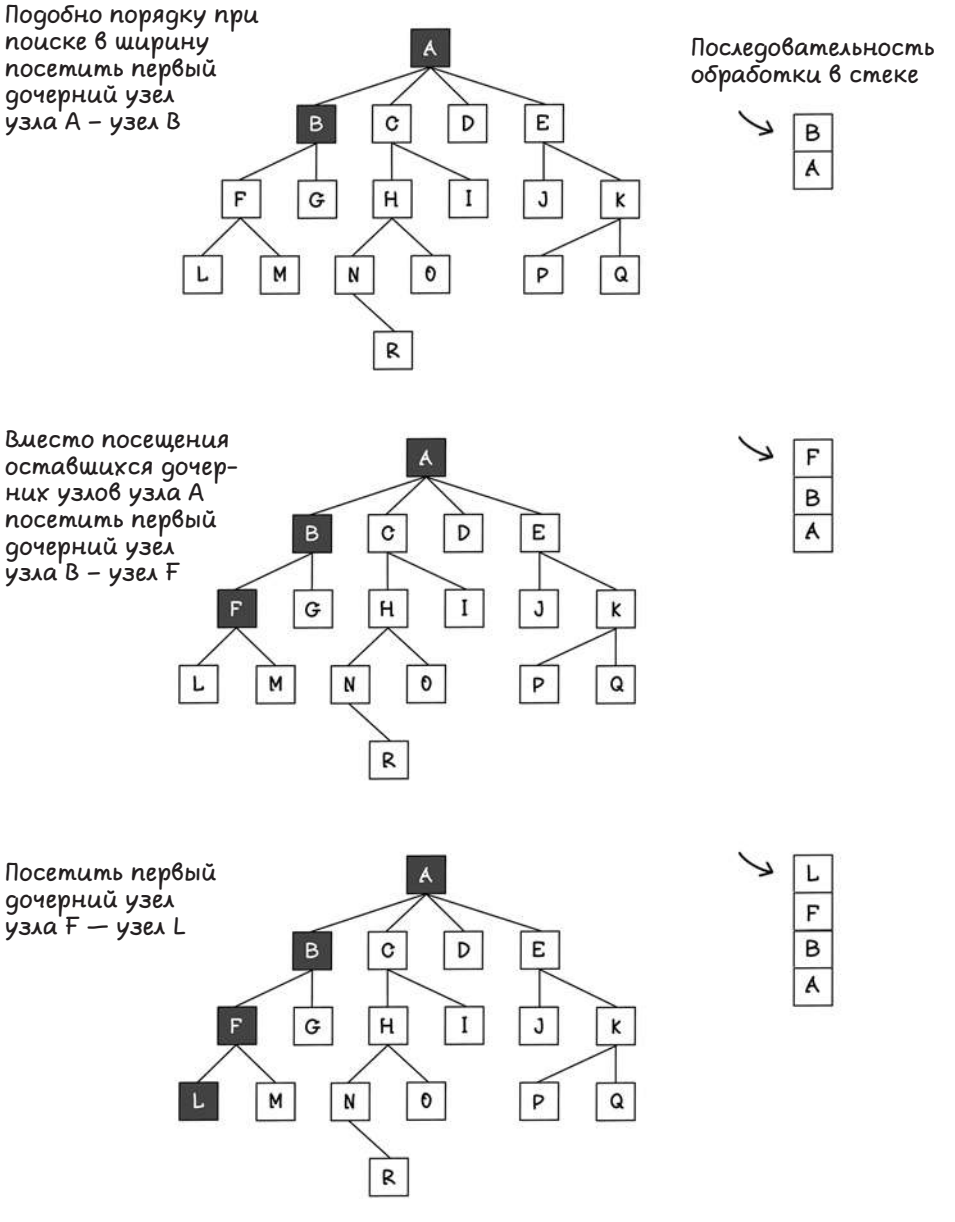


Рис. 2.22. Последовательность обработки дерева поиском в глубину (часть 1)

Термин *проталкивание (push)* означает добавление объектов в стек, а термин *выталкивание (pop)* означает их удаление из стека.

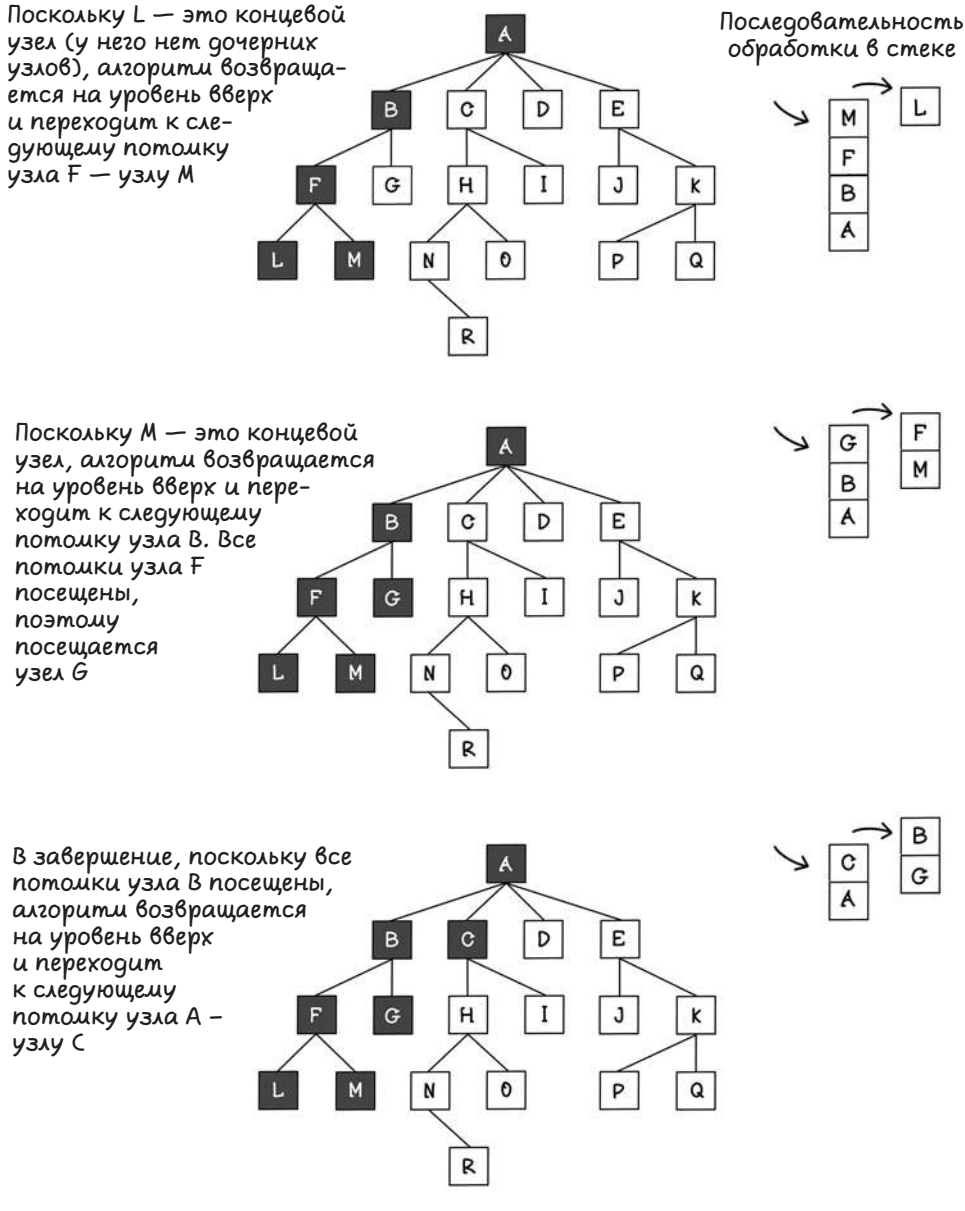
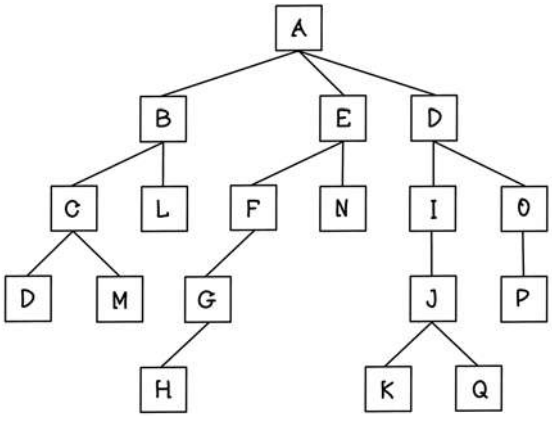


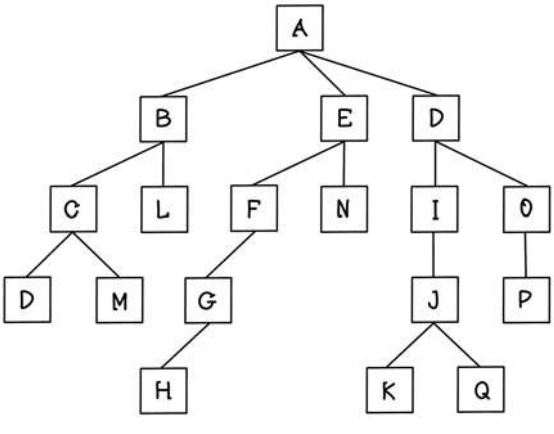
Рис. 2.23. Последовательность обработки дерева поиском в глубину (часть 2)

УПРАЖНЕНИЕ: ОПРЕДЕЛИТЕ ПУТЬ К РЕШЕНИЮ

Каким будет порядок посещения узлов при поиске в глубину в следующем дереве?



ОТВЕТ: ПУТЬ К РЕШЕНИЮ



Порядок посещения узлов при поиске в глубину: A, B, C, D, M, L, E, F, G, H, N, D, I, J, K, Q, O, P

Важно понимать, что при использовании поиска в глубину большое значение имеет порядок потомков, поскольку этот алгоритм начинает с первого потомка, пока не достигнет концевой вершины, после чего возвращается.

В примере с лабиринтом порядок ходов (север, юг, восток и запад) влияет на определенный в итоге путь к цели. Изменение этой последовательности приведет к нахождению другого решения. Ответвления, показанные на рис. 2.24 и 2.25, не имеют значения. В примере лабиринта важен лишь порядок выбора ходов.

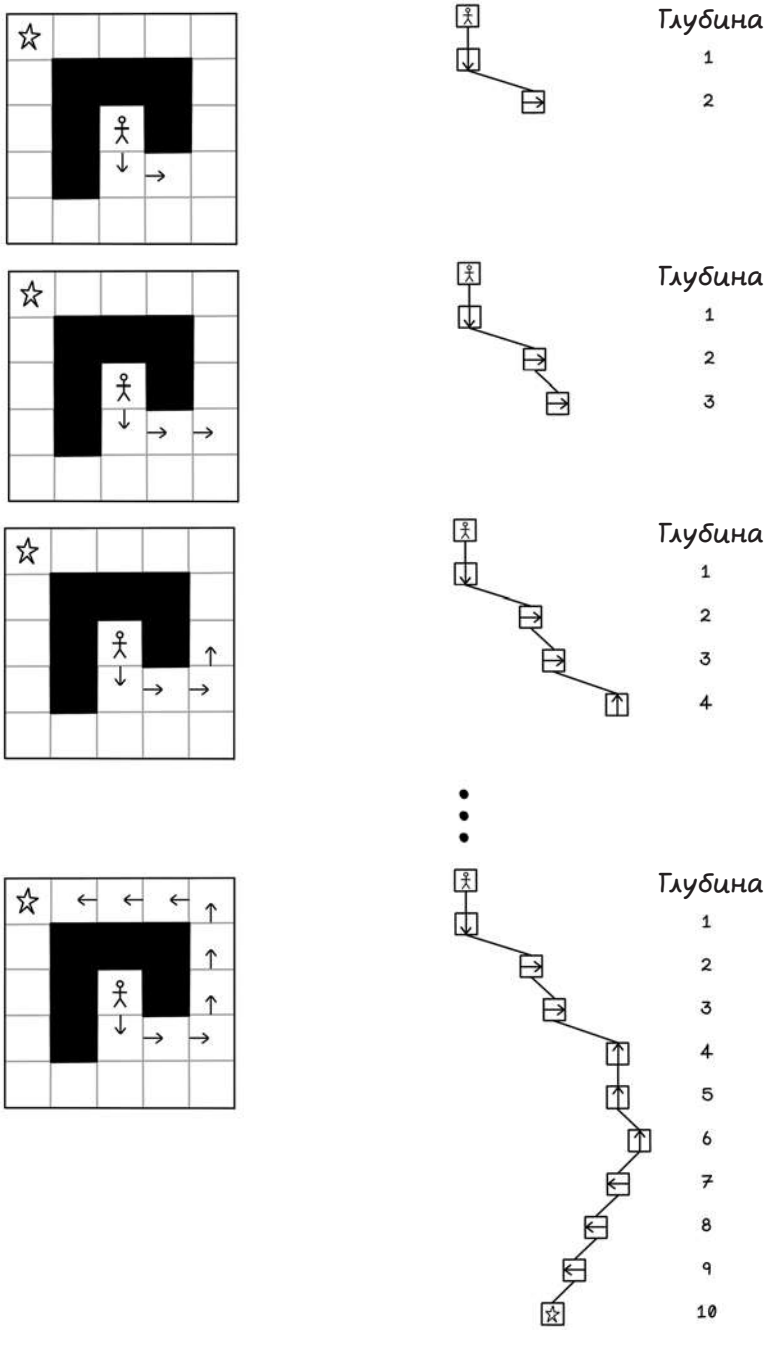


Рис. 2.24. Генерация дерева перемещения по лабиринту с использованием поиска в глубину

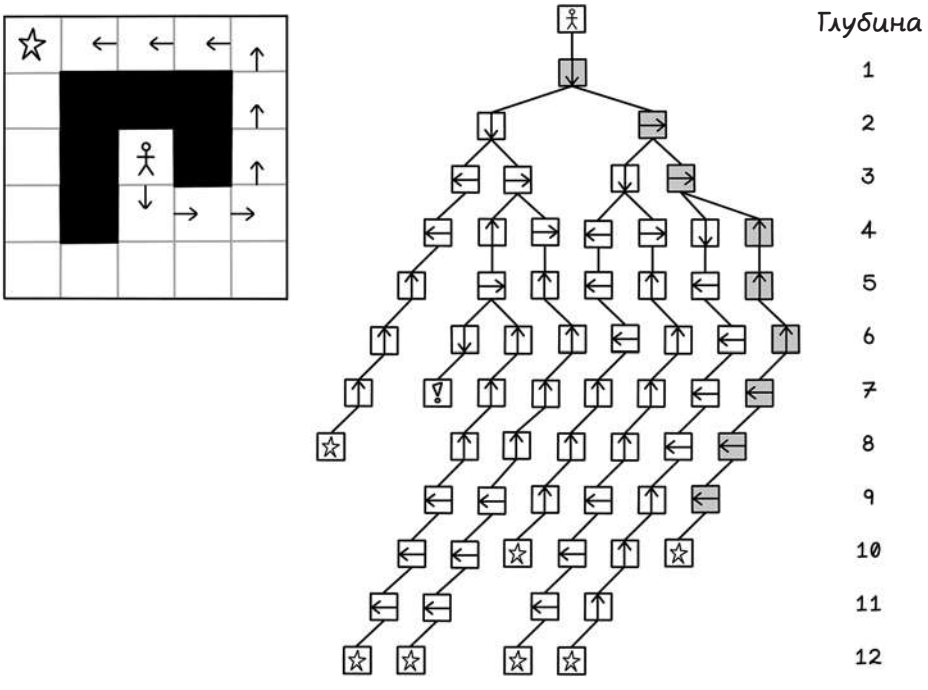


Рис. 2.25. Узлы, посещенные по всему дереву, после поиска в глубину

Псевдокод

Несмотря на то что алгоритм поиска в глубину можно реализовать с помощью рекурсивной функции, мы рассматриваем реализацию, которая достигается с помощью стека. Это позволяет лучше представить последовательность посещения и обработки узлов. В этом процессе очень важно отслеживать посещенные точки, чтобы не посещать повторно одни и те же узлы, создавая замкнутые циклы:

```
run_dfs(maze, root_point, visited_points):
  let s equal a new stack
  add root_point to s
  while s is not empty
    pop s and let current_point equal the returned point
    if current_point is not visited:
```

```
    mark current_point as visited
    if value at current_node is the goal:
      return path using current_point
    else:
      add available cells north, east, south, and west to a list neighbors
      for each neighbor in neighbors:
        set neighbor parent as current_point
        push neighbor to s
return "No path to goal"
```

Применение алгоритмов неинформированного поиска

Неинформированные алгоритмы поиска универсальны, и их можно использовать на практике в разнообразных ситуациях, например:

- *Нахождение путей между узлами в сети* — когда двум компьютерам нужно установить связь по сети, соединение устанавливается через множество других компьютеров и устройств. Поисковые алгоритмы в данном случае используются для соединения двух устройств в сети.
- *Сканирование веб-страниц* — веб-поиск позволяет находить информацию среди огромного количества веб-страниц. Для индексации этих страниц поисковые роботы считывают содержимое каждой, при этом также рекурсивно проходя по всем содержащимся на них ссылкам. Поисковые алгоритмы здесь применяются для создания поисковых роботов и структур метаданных, а также связей между контентом.
- *Поиск связей в социальных сетях* — приложения социальных сетей содержат данные о многих людях и их связях. К примеру, Боб может быть другом Элис, но не быть другом знакомого ей Джона. Таким образом, Боб и Джон связаны через Элис косвенно. Механизм социальной сети может предложить Бобу и Джону стать друзьями, так как у них есть общая знакомая — Элис.

Дополнительно: подробнее о видах графов

Графы используются во многих задачах компьютерной науки и математики. А так как каждый их вид имеет свои особенности, то и применяются к ним

различные принципы и алгоритмы. Категория графа определяется на основе его общей структуры, количества вершин и ребер, а также взаимосвязей между вершинами.

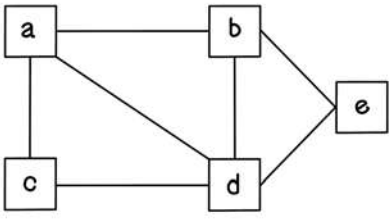
Знание вида графа весьма полезно, так как графы встречаются в поисковых и других алгоритмах ИИ. Перечислим основные виды:

- *Неориентированный граф*. Ребра такого графа не имеют определенного направления. Связь между вершинами является двунаправленной. В качестве аналогии можно привести двустороннее движение по шоссе между городами.
- *Ориентированный граф*. Ребра указывают направление. В этом случае связи между вершинами являются явными. В таком графе вершины разделяются на родителей и потомков, и потомок уже не может выступить в качестве родителя своего родителя.
- *Несвязный граф*. Одна или более вершин не связаны ребрами с остальными. Например, в графе, представляющем сухопутные пути между континентами, соединены будут не все вершины, так как некоторые континенты имеют связь по суше, а другие отделены океаном.
- *Ациклический граф*. Граф, который не содержит циклов. Например, подобно течению времени, такой граф не возвращается назад ни в какую из пройденных ранее точек.
- *Полный граф*. Каждая вершина связана ребром с каждой другой вершиной. Примером будет взаимодействие членов одной команды, когда все общаются и сотрудничают друг с другом.
- *Полный двудольный граф*. Подразумевает разбиение вершин, то есть их группировку на две доли. При разбиении каждая вершина из одной доли ребрами соединяется с каждой вершиной другой доли. Пример — дегустация сыра, когда каждый дегустатор пробует каждый сорт сыра.
- *Взвешенный граф*. Граф, в котором ребра между вершинами имеют веса. Как в случае с расстояниями (весами) между городами — чем больше расстояние, тем больше вес ребра на графе.

Полезно понимать различные виды графов, чтобы уметь составлять полное описание задачи и использовать наиболее эффективный алгоритм для обработки (рис. 2.26). Некоторые из этих категорий будут рассмотрены далее, например в главе 6, посвященной муравьиному алгоритму, и главе 8, освещающей тему нейронных сетей.

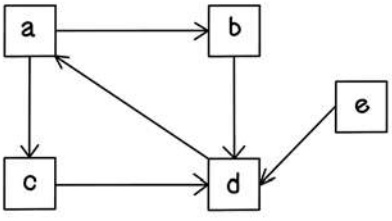
НЕОРИЕНТИРОВАННЫЙ

Ребра не имеют направления. Связь между вершинами двунаправленная



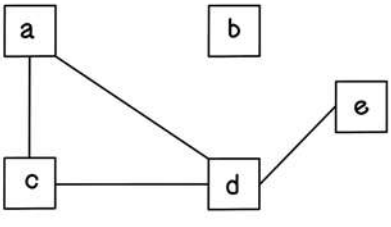
ОРИЕНТИРОВАННЫЙ

Ребра указывают направление. Связи между вершинами явно выражены



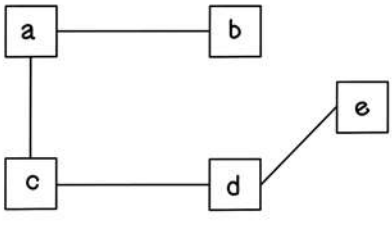
НЕСВЯЗНЫЙ

Одна или более вершин не связаны ребрами с остальными



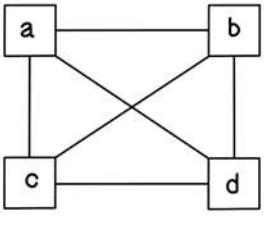
АЦИКЛИЧЕСКИЙ

Граф, который не содержит циклов



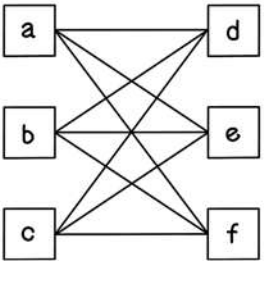
ПОЛНЫЙ

Каждая вершина связана ребром с каждой другой вершиной



ПОЛНЫЙ ДВУДОЛЬНЫЙ

Каждая вершина из одной доли ребрами соединяется с каждой вершиной другой доли



ВЗВЕШЕННЫЙ

Граф, в котором ребра между вершинами имеют веса

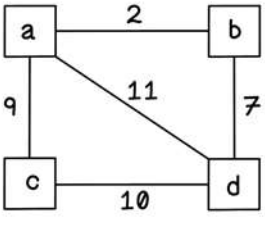


Рис. 2.26. Виды графов

Дополнительно: другие способы представления графа

В зависимости от контекста и используемого языка программирования можно применять и другие представления графов.

Матрица инцидентности

Матрица инцидентности (incidence matrix) — это матрица, в которой высота представляет количество вершин графа, а ширина — количество ребер. Каждая строка выражает связи вершин с конкретным ребром. Если вершина не связана с определенным ребром, записывается значение `0`. Если же вершина имеет связь с конкретным ребром как вершина-получатель (в случае ориентированного графа), то сохраняется значение `1`. Если вершина связана с ребром как вершина-источник или является частью неориентированного графа, то сохраняется значение `-1`. Матрицу инцидентности можно использовать для представления как ориентированных, так и неориентированных графов (рис. 2.27).

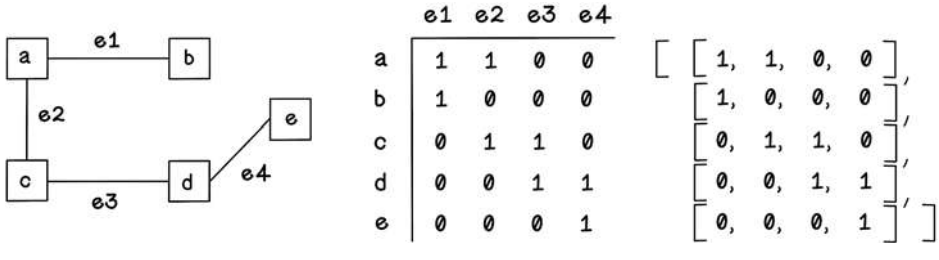


Рис. 2.27. Представление графа как матрицы инцидентности

Список смежности

Список смежности (adjacency list) представляет связанные списки, в которых размер исходного списка определяется количеством вершин в графе и каждое значение представляет количество братьев конкретной вершины. Список смежности можно использовать для представления как ориентированных, так и неориентированных графов (рис. 2.28).

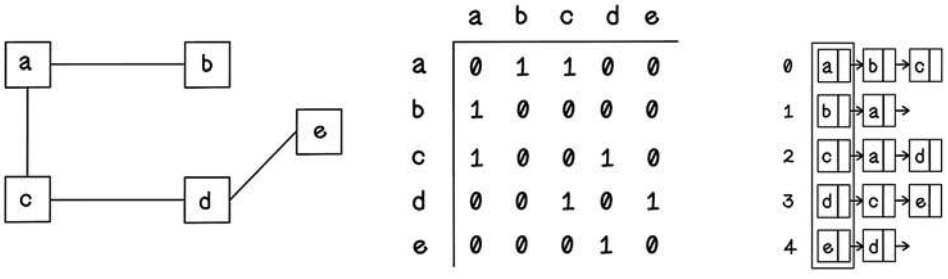


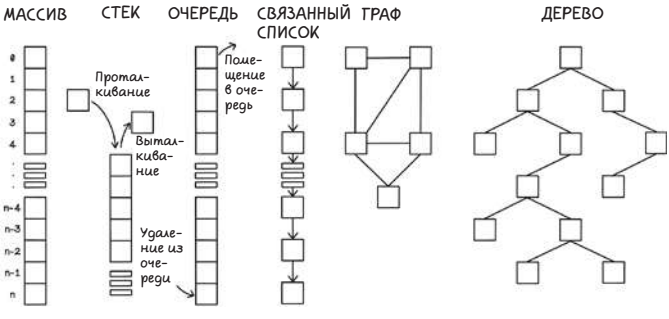
Рис. 2.28. Представление графа в виде списка смежности

Графы — интересная и полезная структура данных, потому что их можно легко представить как математические уравнения, которые лежат в основе всех используемых нами алгоритмов. Далее в книге мы раскроем эту тему более подробно.

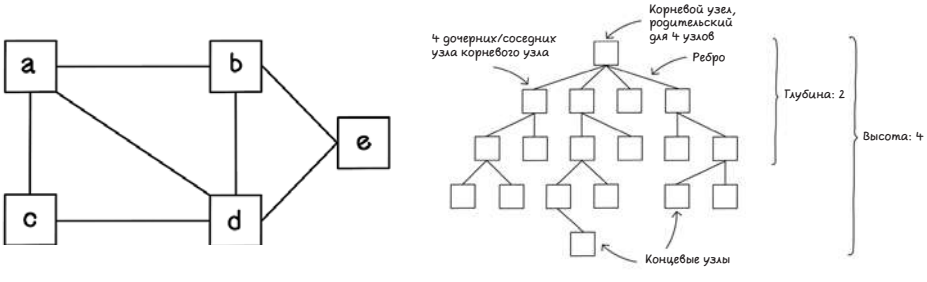
Краткий обзор главы «Основы поиска»

Структуры данных важны для решения задач.

Поисковые алгоритмы эффективны в планировании и поиске решений в некоторых изменяющихся средах

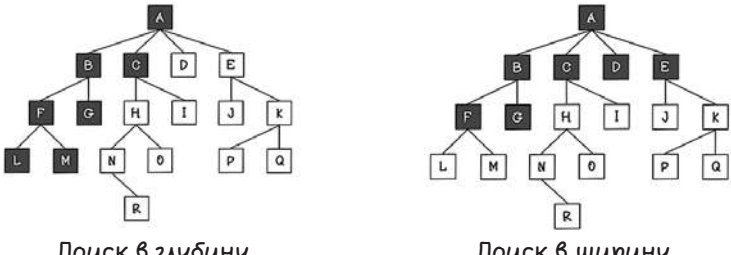


В ИИ применяются структуры графа и дерева



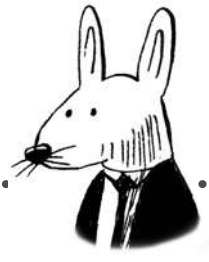
Неинформированный поиск является слепым и может иметь высокую стоимость вычислений. Чтобы этого избежать, необходимо использовать подходящую структуру данных

Поиск в глубину ведется по вертикали, а поиск в ширину — по горизонтали



Поиск в глубину *Поиск в ширину*

3 Умный поиск



В этой главе

- ✓ Что такое эвристика для управляемого поиска и как она создается.
- ✓ Какие задачи можно решать с помощью техник управляемого поиска.
- ✓ Описание и построение алгоритма управляемого поиска.
- ✓ Разработка алгоритма поиска для игры с двумя игроками.

Эвристика: создание обоснованных предположений

Уяснив из главы 2 принципы работы алгоритмов неинформированного поиска, можно научиться их улучшать, если мы соберем больше информации о задаче. Для этого мы задействуем информированный поиск. *Информированный поиск* означает, что у алгоритма есть определенный контекст задачи. Эвристика — это способ представить такой контекст. Нередко ее еще определяют как *правило или набор правил*, используемых для оценки состояния. С ее помощью можно выявлять критерии, которым состояние должно соответствовать, или измерять эффективность конкретного состояния. Эвристика задействуется в случае невозможности применения явного метода поиска оптимального решения. Данную технику можно описать как обоснованное предположение; ее следует рассматривать в отношении задачи скорее как руководство, а не как научную истину.

Например, когда вы заказываете в ресторане пиццу, то ваш эвристический анализ ее качества может исходить из используемых ингредиентов и основы. Если вы любите пиццу, где больше томатной пасты, сыра, грибов и ананаса, на толстом тесте с хрустящей корочкой, то чем больше пицца будет соответствовать этим критериям, тем более высокую эвристическую оценку вы ей дадите. И наоборот, продукт с меньшим соответствием желаемым качествам окажется для вас менее привлекателен и оценку получит ниже.

Еще один пример — это написание алгоритмов для решения задачи GPS-маршрутизации. Эвристика может гласить: «Маршрут считается хорошим, если время движения и пройденное расстояние минимальны» или «Маршрут считается хорошим, если на нем минимум платных автострад и наилучшие дорожные условия». Слабая эвристика GPS-навигатора определит кратчайшее расстояние между точками, просто соединив их прямой линией. Такой способ хорош для птиц и самолетов, но в нашем случае движение происходит по дорогам и тротуарам и неизбежно связано с различными препятствиями: зданиями, светофорами, пробками и т. д. Поэтому грамотная эвристика должна учитывать контекст использования.

Возьмем в качестве следующего примера проверку, принадлежит ли загруженный аудиофайл библиотеке контента с авторскими правами. Поскольку аудиофайлы представляют собой комбинацию звуковых частот, то одним из способов проверки будет сравнить каждый отрезок загруженного файла с каждой записью в библиотеке. Эта задача потребует огромного количества вычислений. Более эффективным будет задать эвристику, минимизирующую разницу между распределениями частот двух записей, как показано на рис. 3.1. Обратите внимание, что частоты идентичны, отличается только время. В результате их гистограммы распределения частот одинаковы. Такое решение не идеально, но это путь к менее дорогостоящему алгоритму.

Рис. 3.1. Сравнение аудиоклипов по распределению частот

Эвристика должна соответствовать контексту и при грамотном определении существенно помогает оптимизировать решения. Изменим сценарий с лабиринтом из главы 2, чтобы продемонстрировать принцип создания эвристики. Для этого введем интересную динамику. До сих пор все ходы рассматривались одинаково, и лучшим решением было то, при котором требовалось совершать меньше всего действий (минимальная глубина дерева). Теперь же ходы, совершаемые в разных направлениях, будут иметь различную стоимость. В нашем лабиринте произошла гравитационная аномалия, и перемещение на север или юг теперь стоит в пять раз дороже, чем движение на восток или запад (рис. 3.2).

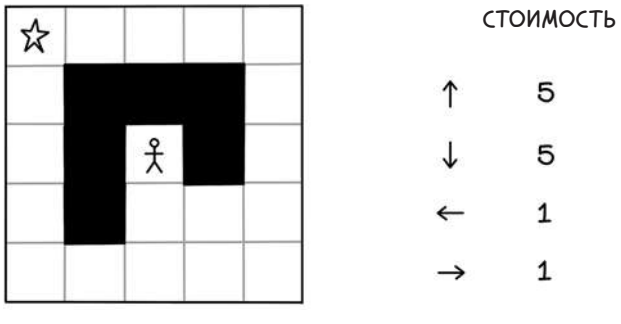


Рис. 3.2. Изменение примера лабиринта: гравитация

В измененном сценарии лабиринта факторами, определяющими лучший возможный путь к цели, являются количество сделанных ходов и сумма их стоимостей для каждого пути.

На рис. 3.3 в виде дерева представлены все возможные варианты пути и стоимость соответствующих ходов. Опять же, этот пример демонстрирует пространство поиска в простом сценарии лабиринта и не соответствует реальным ситуациям. Алгоритм будет генерировать дерево как часть процесса поиска.

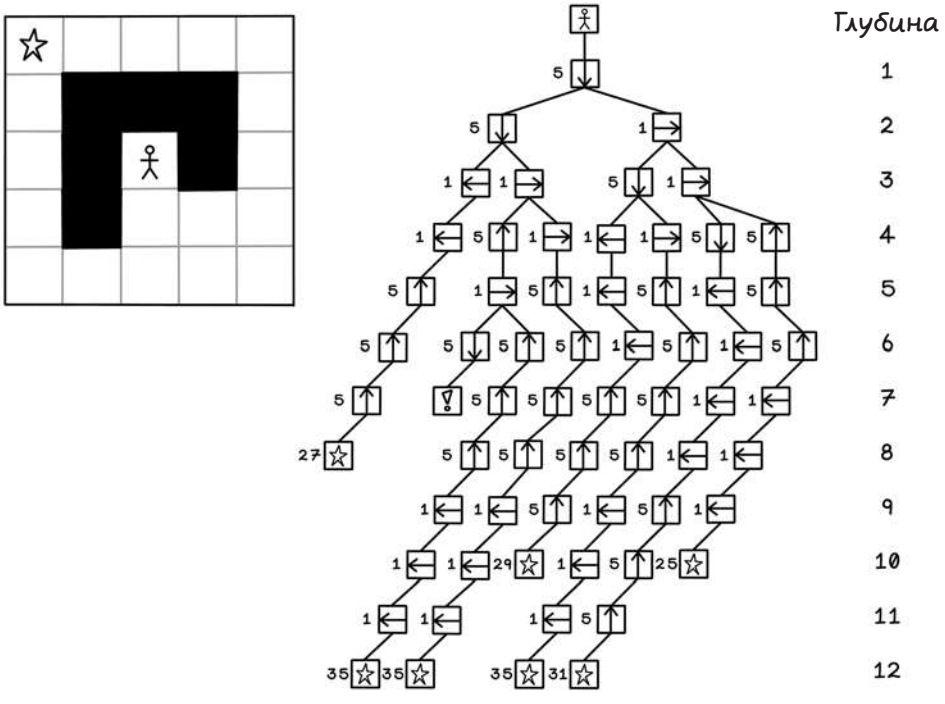


Рис. 3.3. Все возможные варианты ходов, представленные в виде дерева

Эвристику для этой задачи лабиринта можно определить так: «Путь считается хорошим, если стоимость перемещения минимальна и количество ходов до цели наименьшее». Эта простая эвристика поможет выбирать узлы для посещения, так как будет учтено знание пространства решаемой задачи.

УПРАЖНЕНИЕ: ПРЕДЛОЖИТЕ ЭВРИСТИКУ ДЛЯ СЛЕДУЮЩЕГО СЦЕНАРИЯ

Несколько горняков занимаются добычей различных видов ископаемых, а именно алмазов, золота и платины. Любой из них может работать в любой шахте, но наибольшую продуктивность они показывают в тех, которые соответствуют их

профилю. На местности расположено несколько шахт, где могут содержаться алмазы, золото и платина. При этом хранилища добытого сырья находятся на разном расстоянии от этих шахт. Если требуется распределить горняков таким образом, чтобы максимально увеличить их эффективность и уменьшить время пути до склада, то какой должна быть эвристика?

ОТВЕТ: ВОЗМОЖНОЕ РЕШЕНИЕ

Разумная эвристика предполагает назначение каждого горняка на соответствующую его профилю шахту и поручение доставлять сырье на ближайший склад. Это можно также интерпретировать как минимизацию назначения горняков на не соответствующие их профилю шахты совместно с сокращением маршрута до склада.

Информированный поиск: нахождение решений по ориентирам

Информированный поиск, иначе называемый *эвристическим поиском*, — это алгоритм, использующий техники поиска в ширину и в глубину при наличии определенной информации. Такой вид поиска опирается на эвристику, располагая при этом определенным набором данных о решаемой задаче. В зависимости от сути задачи можно применить несколько алгоритмов информированного поиска, к которым также относится жадный алгоритм, иначе известный как поиск по первому наилучшему совпадению. Тем не менее самым популярным и полезным из подобных алгоритмов является A*.

Поиск A*

*Поиск A** произносится как «поиск А звезда». Его алгоритм обычно повышает производительность на основе эвристики с целью выбора минимальной стоимости посещения следующего узла.

Общая стоимость вычисляется с помощью двух показателей: общего расстояния от начального узла до текущего и оценочной стоимости перемещения в конкретный узел на основе эвристики. Когда мы стремимся минимизировать стоимость, то наилучшее решение определяется более низким значением (рис. 3.4).

$$f(n) = g(n) + h(n)$$

g(n): стоимость пути от начального узла до узла n

h(n): стоимость перемещения в узел n, вычисленная эвристически

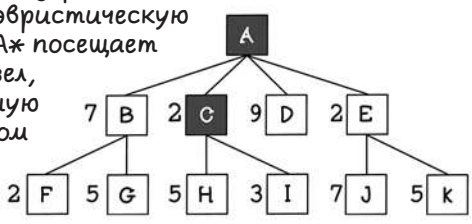
f(n): сумма стоимости пути от начального узла до узла n и стоимости перемещения в узел n, вычисленной эвристически

Рис. 3.4. Функция для алгоритма поиска A*

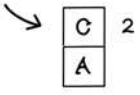
Ниже приведен абстрактный пример обхода дерева при выполнении поиска на основе эвристики. Суть заключается в эвристических вычислениях для разных узлов дерева.

При поиске в ширину посещаются все узлы одной глубины, после чего происходит переход на следующий уровень. При поиске в глубину посещаются сначала все узлы до самого глубокого уровня, затем происходит возврат к корню и начинается обход вниз по следующему пути. Поиск A* отличается тем, что не следует заданному шаблону. Порядок посещения узлов определяется их эвристической стоимостью. Обратите внимание, что в этом алгоритме стоимость всех узлов изначально не известна, она вычисляется по мере обхода или генерации дерева. При этом каждый посещенный узел добавляется в стек, в результате чего узлы, чья стоимость выше стоимости уже посещенных, игнорируются. За счет этого возникает экономия вычислительных ресурсов (рис. 3.5, 3.6 и 3.7).

Примем, что дерево содержит узлы и эти узлы имеют эвристическую стоимость. Поиск A* посещает первый дочерний узел, имеющий наименьшую стоимость. В данном случае это узел C со стоимостью 2.

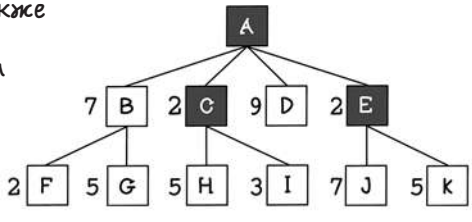


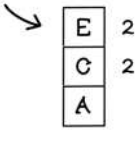
Порядок обработки стека



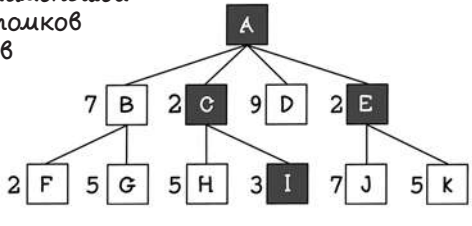
Если два узла имеют одинаковую стоимость, выбирается узел, чья стоимость была рассчитана первой.

Поскольку узел E также имеет стоимость 2 и это дочерний узел для A, он будет следующим в очереди на посещение.





Далее A* выберет для посещения узел с наименьшей стоимостью из потомков узла A или потомков уже посещенных узлов.



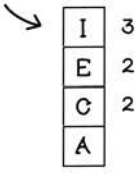
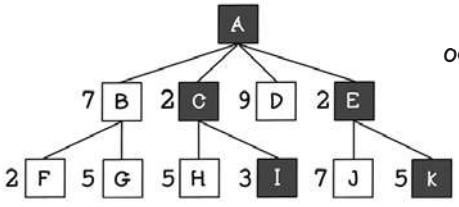
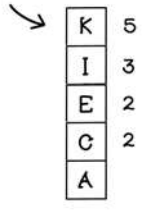


Рис. 3.5. Последовательность обработки дерева поиском A* (часть 1)

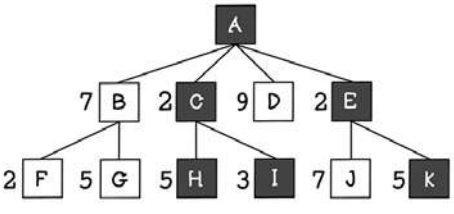
Следующий узел с наименьшей стоимостью — K, дочерний узел для E.

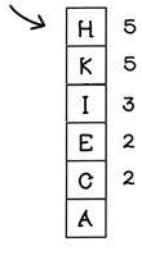


Порядок обработки стека

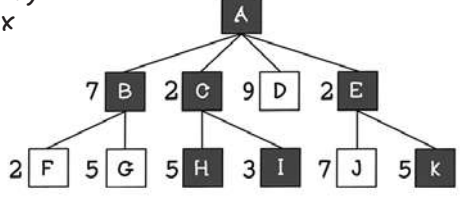


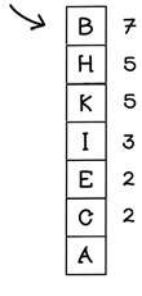
Следующий узел с наименьшей стоимостью — H, дочерний узел для C.





Посещается прямой потомок узла A, поскольку он имеет наименьшую стоимость из всех потомков узла A и потомков всех остальных уже посещенных узлов.





Узлы, имеющие более высокую стоимость, чем текущий путь к решению, игнорируются, поскольку путь через них будет стоить выше.

Рис. 3.6. Последовательность обработки дерева поиском A* (часть 2)

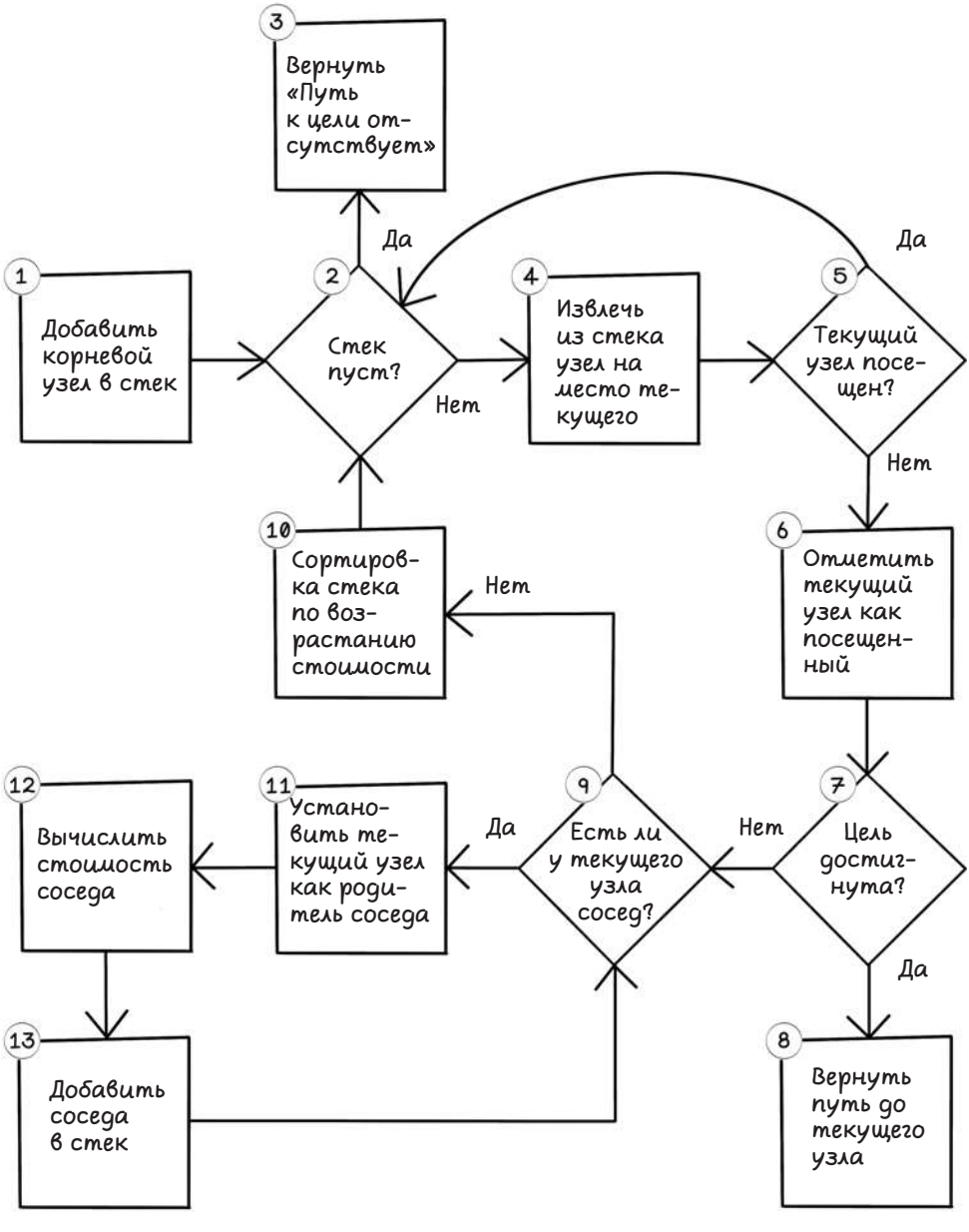


Рис. 3.7. Поток выполнения алгоритма поиска A*

Разберем действия из потока алгоритма поиска A*:

1. *Добавить корневой узел в стек.* Алгоритм поиска A* можно реализовать с помощью стека, в котором последний добавленный объект обрабатывается первым (LIFO). Первым шагом в стек добавляется корневой узел.
2. *Стек пуст?* Если стек пуст и на шаге 8 не был возвращен какой-либо путь, значит, путь к цели отсутствует. Если в очереди остаются узлы, то алгоритм может продолжать поиск.
3. *Вернуть "No path to goal" («Путь к цели отсутствует»).* Этот шаг — один из возможных выходов из алгоритма в случае отсутствия пути к цели.
4. *Извлечь из стека узел на место текущего.* Извлекая следующий объект стека и помещая его в качестве текущего узла, можно изучить его возможности.
5. *Текущий узел посещен?* Если текущий узел не посещался, его можно обработать сейчас.
6. *Отметить текущий узел как посещенный.* Этот шаг указывает, что данный узел посещался, и его ненужная повторная обработка исключается.
7. *Цель достигнута?* На этом шаге определяется, содержит ли текущий сосед искомую алгоритмом цель.
8. *Вернуть путь до текущего узла.* Поочередно ссылаясь на родителя текущего узла, затем его родителя и т. д., описывается путь от цели к корневому узлу. При этом корневой узел будет узлом без родителя.
9. *Есть ли у текущего узла сосед?* Если от текущего узла есть еще возможный ход по лабиринту, то этот ход добавляется в обработку. В противном случае алгоритм переходит к шагу 2, где из стека извлекается следующий объект при его наличии. Принцип работы стека LIFO позволяет алгоритму обрабатывать все узлы до концевой вершины, прежде чем возвращаться обратно и посещать других потомков корневого узла.
10. *Сортировка стека по возрастанию стоимости.* Когда элементы стека упорядочиваются по возрастанию их стоимости, то первым всегда обрабатывается тот, чья стоимость оказывается меньшей.
11. *Установить текущий узел как родитель соседа.* Установка исходного узла как родителя текущего соседа. Этот шаг необходим для отслеживания пути от текущего узла к корневому. На карте исходный узел — это позиция, из которой игрок переместился, а текущий сосед — это позиция, куда он переместился.
12. *Вычислить стоимость соседа.* Функция стоимости необходима для выбора направления алгоритма A*. Чтобы вычислить стоимость, нужно сложить расстояние от корневого узла с эвристической стоимостью следующего хода. Более интеллектуальная эвристика будет непосредственно влиять на алгоритм A*, повышая его эффективность.
13. *Добавить соседа в стек.* Узел-сосед добавляется в стек, чтобы впоследствии можно было исследовать его потомков. Опять же, такой механизм стека позволяет обрабатывать узлы сначала до наибольшей глубины, после чего переходить к соседям уровнями выше.

Здесь по аналогии с поиском в глубину порядок дочерних узлов влияет на выбранный путь, но не столь значительно. Если у двух узлов одинаковая стоимость, сначала посещается первый (рис. 3.8, 3.9, 3.10).

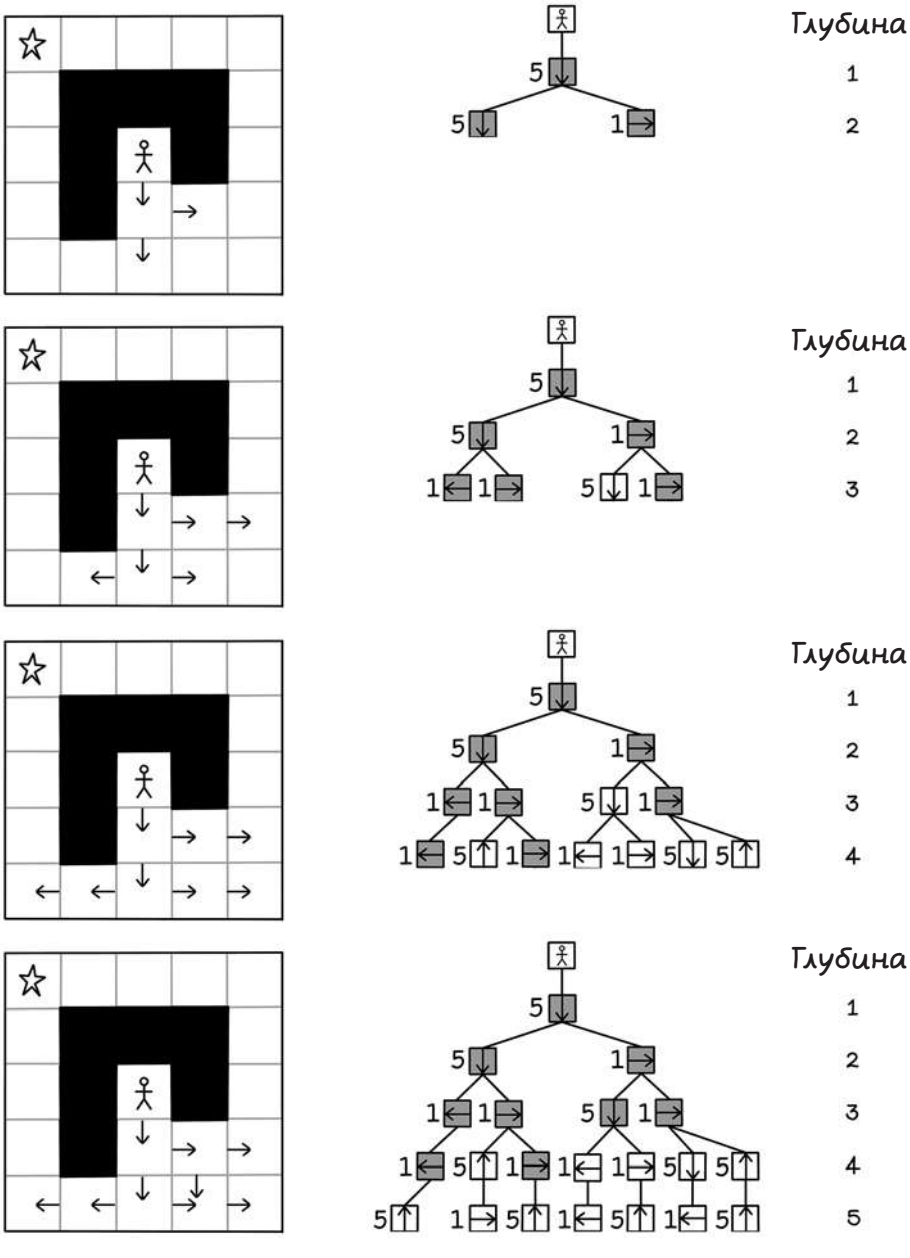


Рис. 3.8. Порядок обработки дерева поиском A* (часть 1)

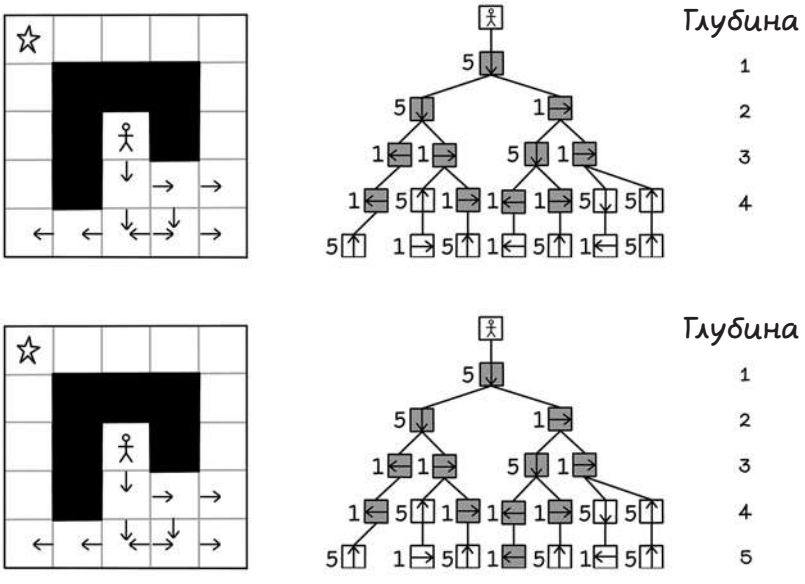


Рис. 3.9. Порядок обработки дерева поиском A* (часть 2)

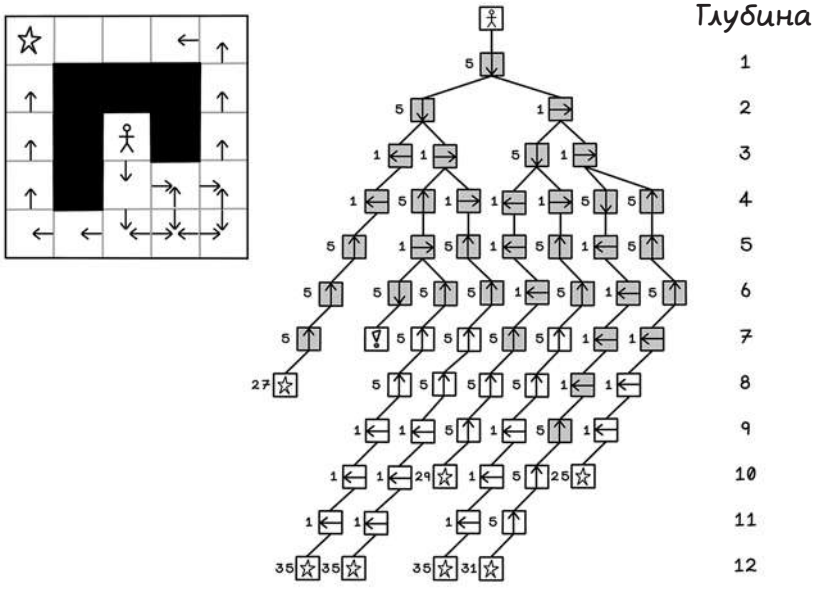


Рис. 3.10. Узлы, посещенные во всем дереве, по завершении поиска A*

Обратите внимание, что к цели есть несколько путей, но алгоритм A* находит маршрут, требующий наименьшей стоимости и наименьшего количества ходов с учетом того, что движения на север и юг стоят дороже.

Псевдокод

Алгоритм A* работает по аналогичному с поиском в глубину принципу, но при этом целенаправленно выбирает узлы, стоимость посещения которых ниже. Для обработки узлов используется механизм стека, но этот стек при каждом вычислении упорядочивается по возрастанию стоимости. Такой порядок гарантирует, что извлекаться всегда будет объект с наименьшей стоимостью:

```
run_astar(maze, root_point, visited_points):
  let s equal a new stack
  add root_point to s
  while s is not empty
    pop s and let current_point equal the returned point
    if current_point is not visited:
      mark current_point as visited
      if value at current_node is the goal:
        return path using current_point
      else:
        add available cells north, east, south, and west to a list neighbors
        for each neighbor in neighbors:
          set neighbor parent as current_point
          set neighbor cost as calculate_cost(current_point, neighbor)
          push neighbor to s
        sort s by cost ascending
  return "No path to goal"
```

Функция вычисления стоимости необходима для поиска A*. Она предоставляет информацию, на основе которой алгоритм находит путь с наименьшей стоимостью. В нашем измененном примере лабиринта более высокая стоимость ассоциируется с движением вверх или вниз. Если с функцией вычисления стоимости возникнут проблемы, то алгоритм работать не сможет.

Две приведенные ниже функции описывают вычисление стоимости. Расстояние от корневого узла прибавляется к стоимости следующего хода. В нашем гипотетическом примере стоимость перемещения на север или юг влияет на общую стоимость посещения этого узла:

```
calculate_cost(origin, target):
  let distance_to_root equal length of path from origin to target
  let cost_to_move equal get_move_cost(origin, target)
  return distance_to_root + cost_to_move


move_cost(origin, target):
  if target is north or south of origin:
    return 5
  else:
    return 1
```

Алгоритмы неинформированного поиска, такие как поиск в ширину и глубину, изучают каждую возможность и находят оптимальное решение. Поиск A*, в свою очередь, полезен, когда для процесса можно сформулировать разумную эвристику. Он более эффективен в вычислениях, поскольку игнорирует узлы, чья стоимость выше стоимости уже посещенных. Если же эвристика продумана плохо и не отражает смысл решаемой задачи и контекста, то вместо оптимальных решений будут найдены посредственные.

Применение алгоритмов информированного поиска

Информированный поиск универсален и полезен в ситуациях, когда можно задать эвристику. Например:

- *Нахождение пути для автономных игровых персонажей в видеоиграх*. Разработчики часто используют этот алгоритм для управления перемещением вражеских единиц в игре, где целью является найти игрока-человека в игровой среде.
- *Парсинг абзацев при обработке естественного языка (natural language processing, NLP)*. Содержимое абзаца может быть разделено на композиции из фраз, а те, в свою очередь, на композиции из слов разных видов (например, существительных и глаголов) с формированием структуры

дерева для дальнейшей оценки. Информированный поиск полезен для извлечения значения содержимого.

- *Маршрутизация телекоммуникационной сети.* Алгоритмы управляемого поиска используются для нахождения кратчайшего пути сетевого трафика, что позволяет повысить общую производительность сети. Серверы/сетевые узлы и соединения представляются как графы из вершин и ребер, по которым можно выполнять поиск.
- *Игры и головоломки с одним игроком.* Информированный поиск может использоваться в играх и головоломках с одним игроком, наподобие кубика Рубика, так как в них каждое движение является решением в дереве вероятностей вплоть до достижения цели.

Состязательный поиск: нахождение решений в изменяющейся среде

Поиск по лабиринту подразумевает единственного участника — игрока. Среда при этом изменяется только под его влиянием, то есть он генерирует все вероятности. До сих пор целью было максимально упростить задачу для игрока, найдя кратчайший путь к цели с наименьшей стоимостью.

Состязательный поиск характеризуется противодействием или конфликтом. Состязательные задачи для достижения цели требуют предугадывания, понимания и парирования действий противника. К примерам таких задач можно отнести пошаговые игры для двух игроков, такие как «Крестики-нолики» и «Четыре в ряд». Игроки по очереди делают ходы, изменяя состояние среды согласно своим целям. При этом возможности изменения среды определяются набором правил, которые также описывают условия завершения игры и достижения победы.

Простая состязательная задача

Представим решение состязательных задач на примере игры «Четыре в ряд». В ней (рис. 3.11) игроки по очереди помещают фишки в определенные столбцы игровой доски. В результате фишки в столбце накапливаются, и выигрывает тот игрок, кто первым сможет составить из них непрерывную последовательность по горизонтали, вертикали или диагонали. Если доска заполняется до выявления победителя, то игра оканчивается вничью.

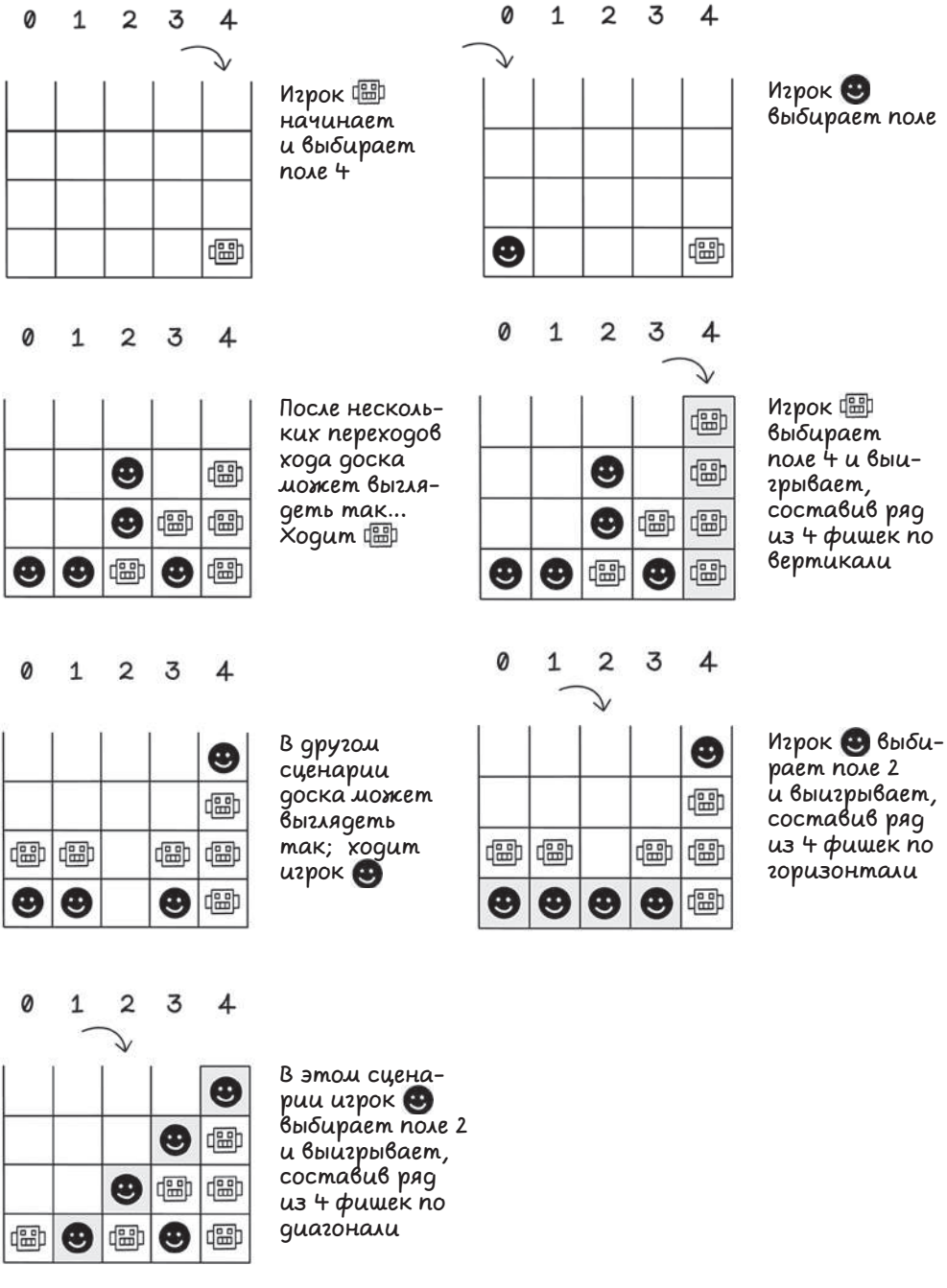


Рис. 3.11. Игра «Четыре в ряд»

Минимаксный поиск: симуляция действий и выбор лучшего исхода

Минимаксный поиск выстраивает дерево возможных исходов на основе ходов, которые мог бы сделать каждый игрок, и выбирает наиболее выгодные пути для агента, избегая предпочтительных путей для оппонента. Для этого алгоритм симулирует возможные ходы и оценивает состояние на основе эвристики после совершения соответствующего хода. Минимаксный алгоритм пытается просчитать как можно больше вероятностей. Однако из-за ограничения памяти и вычислительной мощности изучить все вероятности игры обычно невозможно, поэтому поиск производится до определенной глубины. Алгоритм симулирует переход хода игроков, поэтому определяемая глубина напрямую связана с количеством ходов обоих игроков. Например, глубина 4 означает, что каждый игрок сделал по два хода. Игрок A делает ход, игрок B тоже делает ход, затем A делает еще один ход и B также делает ход.

Эвристика

Минимаксный алгоритм принимает решения на основе эвристических значений. Эти значения не изучаются алгоритмом, а определяются заданной эвристикой. Если взять определенное состояние игры, то каждый допустимый исход из этого состояния будет дочерним узлом в дереве игры.

Предположим, что мы используем эвристику, в которой положительные значения предпочтительнее отрицательных. Путем симуляции каждого допустимого хода алгоритм старается сократить количество ходов, которые дадут противнику преимущество или позволят ему победить, и при этом увеличить количество ходов, которые, наоборот, позволят агенту получить преимущество или победить.

На рис. 3.12 показано дерево минимаксного поиска. Здесь концевые вершины являются единственными узлами, где вычисляется эвристическое значение, поскольку эти состояния определяют победителя или ничью. Другие узлы дерева указывают состояния в процессе. Начиная с глубины, где вычислено значение, и продвигаясь вверх, выбирается потомок либо с минимальным, либо с максимальным значением, что зависит от того, чей ход идет следующим в будущих симулируемых состояниях. Начиная с верхнего узла, агент старается выбрать максимальное значение, и с каждым переходом хода намерение меняется на противоположное, так как цель — получить максимальное значение для агента и минимальное для оппонента.

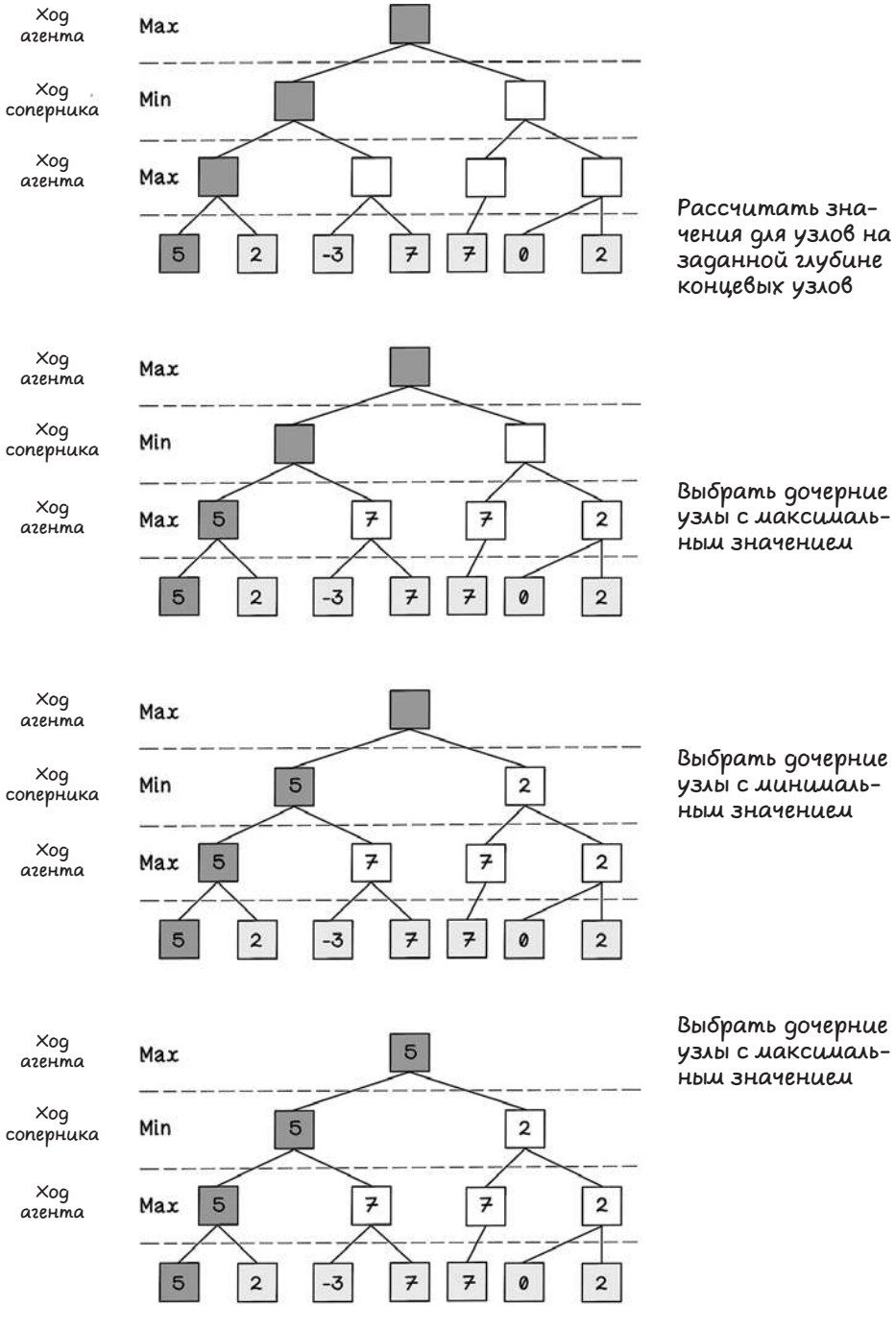
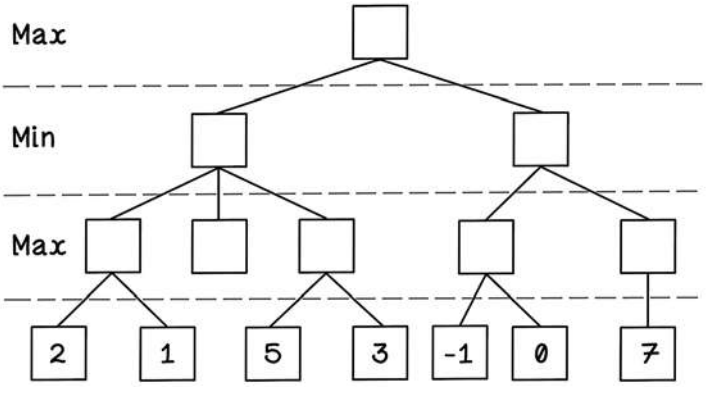
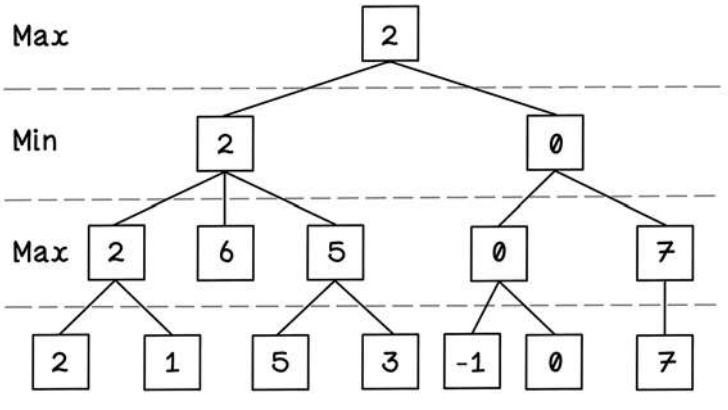


Рис. 3.12. Порядок обработки дерева минимаксным поиском

УПРАЖНЕНИЕ: КАКИЕ ЗНАЧЕНИЯ НЕОБХОДИМО ПОДСТАВИТЬ В УЗЛЫ ДЕРЕВА МИНИМАКС?



ОТВЕТ: ЗНАЧЕНИЯ ДЛЯ УЗЛОВ ДЕРЕВА МИНИМАКС



Так как минимаксный алгоритм симулирует возможные результаты, в играх, предлагающих множество выборов, количество вероятностей растет со взрывной скоростью и дерево быстро становится слишком сложным для обработки. Даже в простом примере игры «Четыре в ряд» на доске 5 × 4 количество вероятностей столь велико, что изучение возможностей всего дерева на каждом ходу становится неэффективным (рис. 3.13).

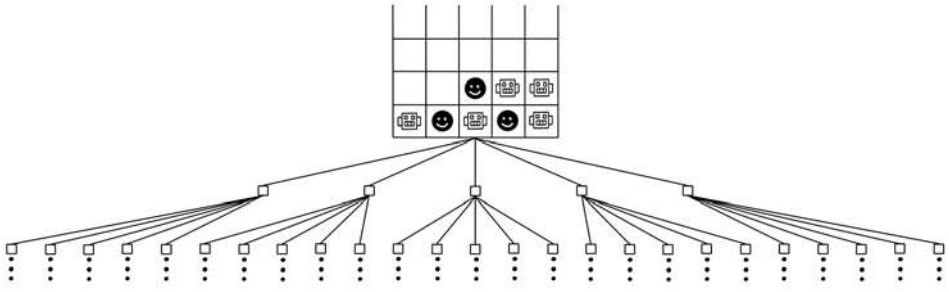


Рис. 3.13. Взрывной рост подсчета вероятностей при обходе дерева игры

При минимаксном поиске в примере «Четыре в ряд» алгоритм фактически совершает все возможные ходы из текущего состояния игры. Затем он рекурсивно определяет все возможные ходы из каждого полученного состояния до тех пор, пока не будет найден наиболее предпочтительный. Состояния игры, которые приводят к победе агента, возвращают значение 10, а состояния, приводящие к победе оппонента, — –10. Алгоритм стремится выбрать для агента максимально возможное положительное значение (рис. 3.14 и 3.15).

Рис. 3.14. Значения для агента и для противника

Несмотря на то что схема потока этого алгоритма выглядит сложной и объемной, по сути она проста. Размер схемы вызван количеством условий, которые проверяют, является ли текущее состояние MIN или MAX.

Рассмотрим шаги этого алгоритма:

1. *Дано: состояние игры, текущее состояние MAX или MIN и текущая глубина. Алгоритм начинает выполнение.* Важно понять входные данные для алгоритма, так как он рекурсивен, то есть вызывает сам себя на одном или нескольких шагах. Поэтому такой алгоритм обязательно должен иметь условие выхода, которое не позволит ему вызывать себя бесконечно.
2. *Текущее состояние конечное или глубина равна 0?* Это условие определяет, является ли текущее состояние игры заключительным, а также была

ли достигнута нужная глубина. Состояние завершения — то, в котором один из игроков победил либо игра закончилась вничью. Значение 10 представляет победу агента, –10 — победу противника, а 0 указывает на ничью. Глубина задается, поскольку обход всего дерева возможностей до всех конечных состояний слишком дорогостоящая процедура и для компьютера средней мощности потребует очень много времени. Когда глубина задана, алгоритм может просчитать несколько дальнейших ходов, чтобы определить, достигается ли в них конечное состояние.

3. *Вернуть текущее значение и последний ход.* Значение текущего состояния возвращается, если достигается конечное состояние игры или заданная глубина.

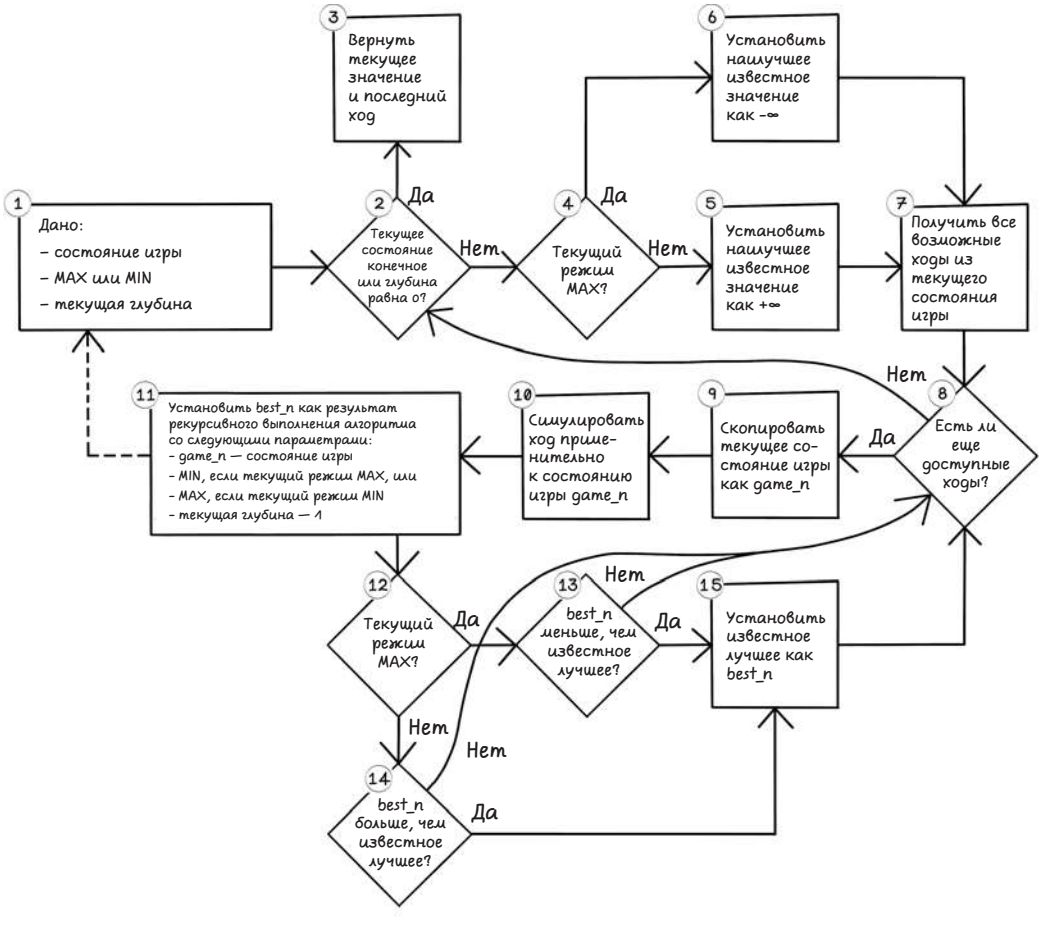


Рис. 3.15. Поток выполнения алгоритма минимаксного поиска

4. *Текущий режим MAX?* Если текущая итерация алгоритма находится в состоянии выбора максимального значения, то алгоритм стремится выбрать для агента именно максимальное значение.
5. *Установить наилучшее известное значение как* $+\infty$. Если текущий режим подразумевает выбор минимального значения, то лучшее значение задается положительной бесконечностью, потому что нам известно, что возвращаемые состояниями игры значения всегда будут меньше. В фактической реализации вместо бесконечности используется просто очень большое число.
6. *Установить наилучшее известное значение как* $-\infty$. Если текущий режим подразумевает выбор максимального значения, то лучшее значение задается отрицательной бесконечностью, так как нам известно, что возвращаемые состояниями игры значения всегда будут больше. В фактической реализации вместо бесконечности используется просто очень большое отрицательное число.
7. *Получить все возможные ходы из текущего состояния игры*. Этот шаг определяет список возможных ходов из текущего состояния игры. В процессе самой игры некоторые изначально доступные ходы утрачивают свою актуальность. Например, в игре «Четыре в ряд» столбец может заполниться, и его выбор станет невозможен.
8. *Есть ли еще доступные ходы?* Если еще не симулированы все возможные ходы и при этом не осталось доступных ходов, алгоритм возвращает лучший ход в этом экземпляре вызова функции.
9. *Скопировать текущее состояние игры как game_n*. Для симулирования возможных будущих ходов требуется копия текущего состояния игры.
10. *Симулировать ход применительно к состоянию игры game_n*. На этом этапе к скопированному состоянию игры применяется текущий рассматриваемый ход.
11. *Установить best_n как результат рекурсивного выполнения алгоритма*. Именно здесь возникает рекурсия. Переменная *best_n* используется для хранения следующего лучшего хода, и мы ставим алгоритму задачу исследовать дальнейшие возможности с этого хода.
12. *Если текущий режим MAX*. Когда рекурсивный вызов возвращает лучшего кандидата, это условие определяет, подразумевает ли текущий режим выбор максимального значения.
13. *best_n меньше, чем известное лучшее?* Этот шаг определяет, нашел ли алгоритм значение лучше предыдущего при условии, что активен режим MAX.
14. *best_n больше, чем известное лучшее?* Этот шаг определяет, нашел ли алгоритм значение лучше предыдущего при условии, что активен режим MIN.
15. *Установить известное лучшее как best_n*. Если было найдено новое лучшее значение, оно устанавливается в качестве известного лучшего.

В определенном состоянии игры минимаксный алгоритм генерирует дерево, показанное на рис. 3.16. Изучается каждый ход, начиная с начального состояния. Далее изучается каждый ход из следующего состояния до тех пор, пока не будет достигнуто заключительное — то есть либо доска будет заполнена, либо один из игроков победит.

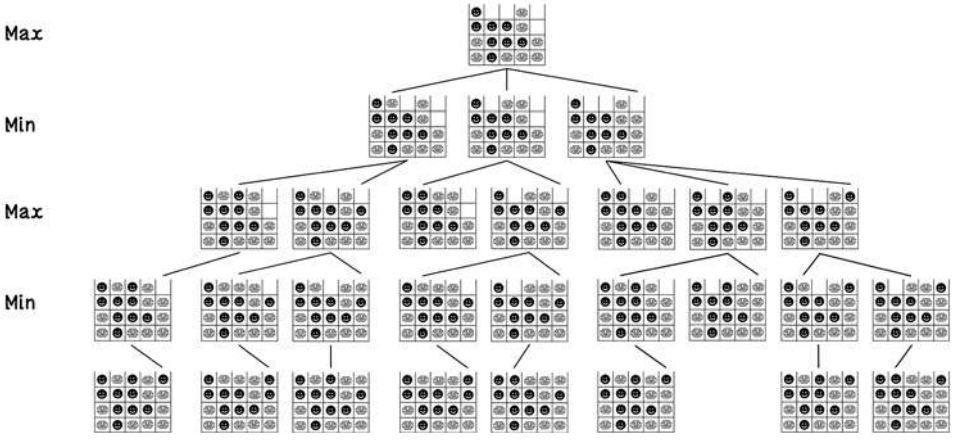


Рис. 3.16. Представление возможных состояний в игре «Четыре в ряд»

Выделенные на рис. 3.17 узлы представляют заключительное состояние, в котором ничьи обозначаются значением 0, проигрыши –10, а победы 10. Так как алгоритм стремится к максимальному значению, то для него необходимо положительное число, в то время как победа оппонента обозначается отрицательным.

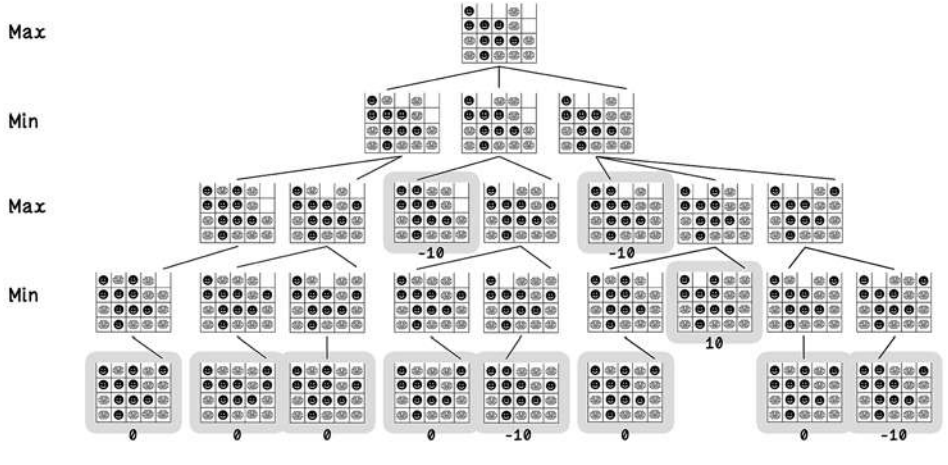


Рис. 3.17. Возможные заключительные состояния в игре «Четыре в ряд»

Когда значения известны, минимаксный алгоритм начинает с наибольшей глубины и выбирает узел с минимальным значением (рис. 3.18).

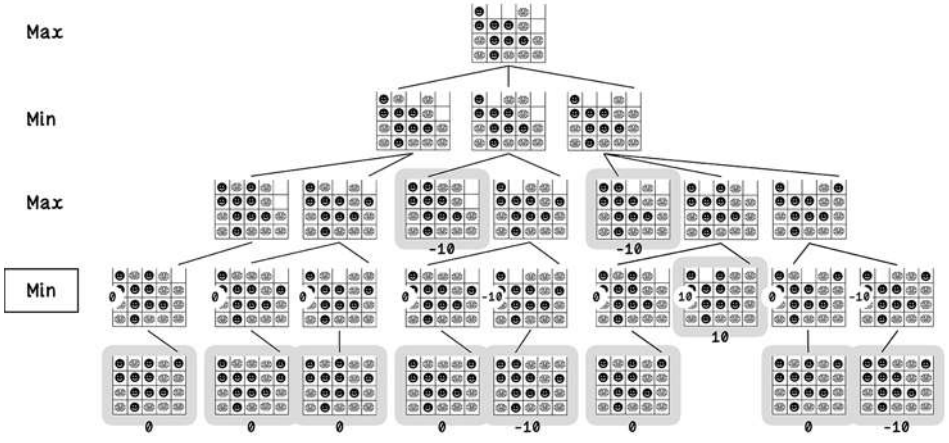


Рис. 3.18. Возможные значения для заключительных состояний в игре «Четыре в ряд» (часть 1)

Далее, на следующем уровне, алгоритм выбирает узел с максимальным значением (рис. 3.19).

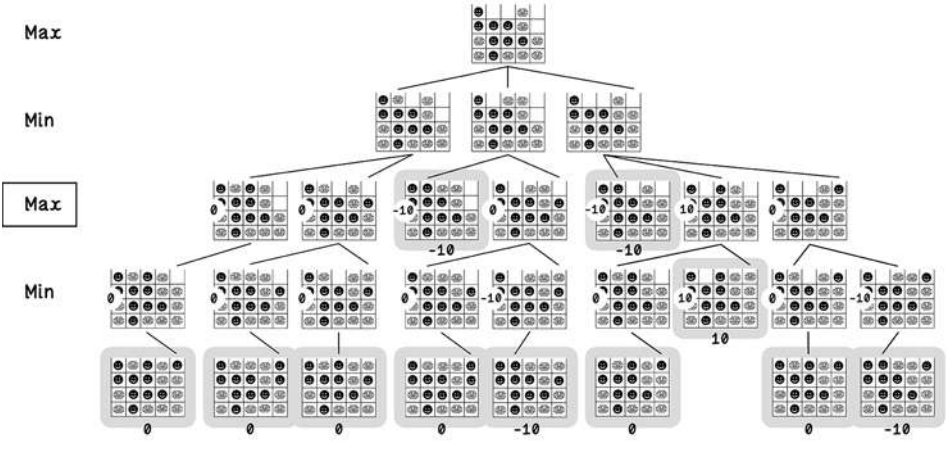


Рис. 3.19. Возможные значения заключительных состояний игры «Четыре в ряд» (часть 2)

В завершение на следующем уровне выбираются узлы с минимальным значением, а в корневом узле выбирается максимальный из вариантов. Следуя выбору узлов и значений и интуитивно разбирая задачу, мы видим, что алгоритм с целью избежать проигрыша выбирает путь к ничьей. Если он выберет путь к победе, то существует высокая вероятность проигрыша на следующем ходе. Алгоритм предполагает, что оппонент всегда будет делать самый грамотный ход для увеличения шанса на победу (рис. 3.20).

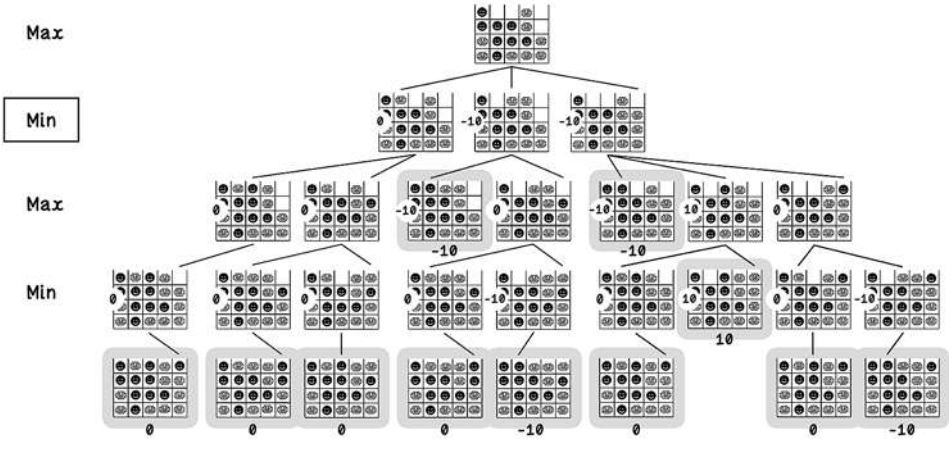


Рис. 3.20. Возможные значения заключительных состояний в игре «Четыре в ряд» (часть 3)

Упрощенное дерево на рис. 3.21 представляет результат выполнения минимаксного алгоритма поиска для заданного примера состояния игры.

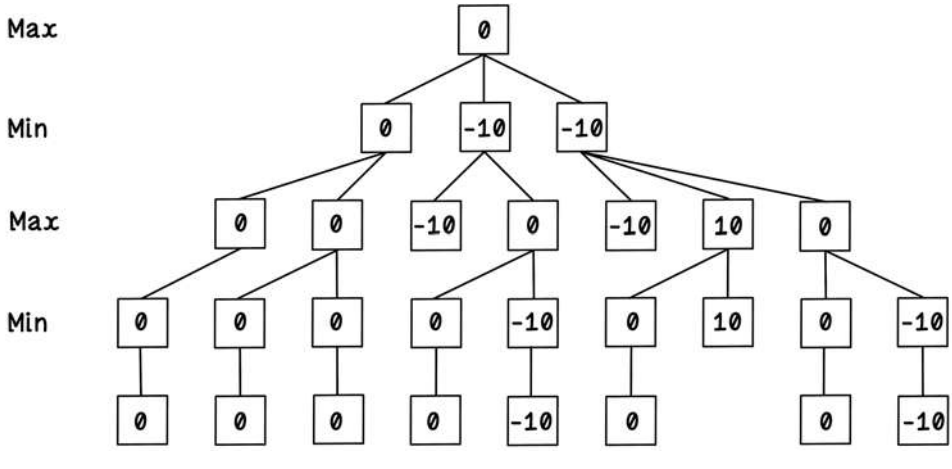


Рис. 3.21. Упрощенное дерево игры с минимаксными значениями

Псевдокод

Поисковый минимаксный алгоритм реализуется как рекурсивная функция. Этой функции передается текущее состояние, нужная глубина для поиска, режим MIN или MAX, а также последний ход. Алгоритм завершается, возвращая лучший ход и значение для каждого потомка на каждой глубине дерева. Сравнивая код с блок-схемой из рис. 3.15, мы видим, что монотонные условия проверки, является ли текущий режим MIN или MAX, не так явно выражены. В псевдокоде 1 или –1 представляет намерение выбирать максимальное или минимальное значение соответственно. Если применить нехитрую логику, то можно выбирать лучшее значение, условия и выполнять переключение состояния по принципу умножения двух отрицательных чисел, в результате которого получается положительное число. Поэтому, если –1 указывает на ход оппонента, то умножение на –1 даст 1, что укажет на ход агента. Тогда для следующего хода умножение 1 на –1 даст –1, снова указав на ход противника:

```
minmax(state, depth, min_or_max, last_move):
  let current score equal state.get_score
  if current_score is not equal to 0 or state.is_full or depth is equal to 0:
    return new Move(last_move, current_score)
  let best_score equal to min_or_max multiplied by -∞
  let best_move = -1
  for each possible choice (0 to 4 in a 5x4 board) as move:
    let neighbor equal to a copy of state
    execute current move on neighbor
    let best_neighbor equal minmax(neighbor, depth -1, min_or_max * -1, move)
    if (best_neighbor.score is greater than best_score and min_or_max is MAX)
    or (best_neighbor.score is less than best_score and min_or_max is MIN):
      let best_score = best_neighbor.score
      let best_move = best_neighbor.move
  return new Move(best_move, best_score)
```

Альфа-бета отсечение: оптимизация путем исследования только целесообразных путей

Альфа-бета отсечение — это техника, используемая совместно с алгоритмом минимаксного поиска для отсечения областей поиска в дереве игры, заведомо представляющих неудачные решения. Эта техника оптимизирует минимаксный алгоритм и экономит вычислительные ресурсы за счет игнорирования ненужных путей. С учетом того, как резко увеличивается количество путей в дереве решений игры «Четыре в ряд», очевидно, что игнорирование лишних путей намного улучшит производительность (рис. 3.22).

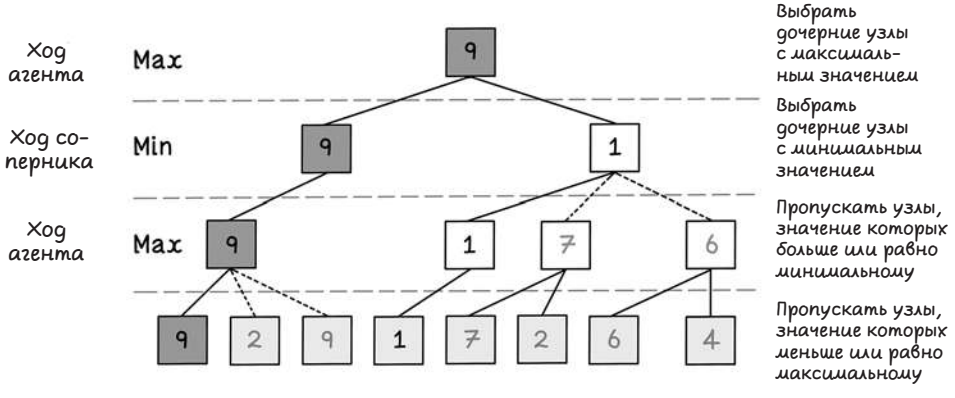


Рис. 3.22. Пример альфа-бета отсечения

Алгоритм альфа-бета отсечения сохраняет лучшее значение для игрока, идущего по максимальным значениям, и лучшее значение для игрока, идущего по минимальным значениям, как альфа и бета соответственно. Изначально альфа устанавливается как $-\infty$, а бета как $+\infty$ — худшее значение для каждого игрока. Если лучшее значение игрока, для которого предполагается минимальный ход, меньше, чем лучшее значение игрока, совершающего максимальный ход, то логично заключить, что другие дочерние пути посещенных узлов не повлияют на лучшее значение.

На рис. 3.23 показаны изменения в алгоритме минимаксного поиска, когда добавлена оптимизация альфа-бета отсечения. Добавленные шаги выделены серой заливкой.

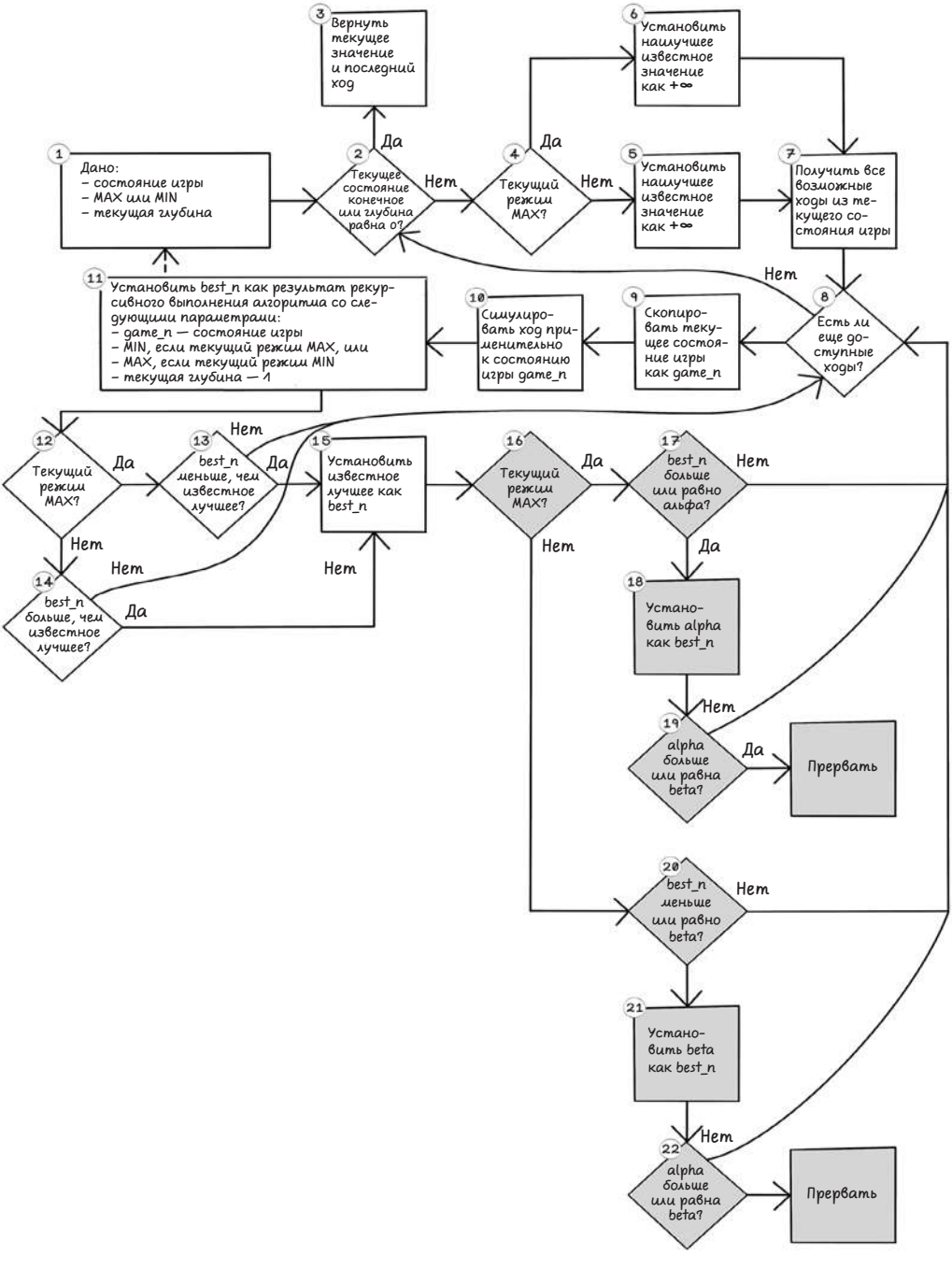


Рис. 3.23. Поток минимаксного алгоритма с альфа-бета отсечением

Описанные ниже шаги дополняют минимаксный поиск. Данные условия позволяют исключить изучение путей, на которых лучшее найденное значение не повлияет на исход:

1. *Текущий режим MAX?* Снова определяем, находится ли алгоритм на стадии выбора максимального или минимального значения.
2. *best_n больше или равно alpha?* Если текущий режим подразумевает выбор максимального значения и текущее лучшее значение больше или равно alpha, то потомки данного узла не содержат лучших значений и алгоритм может игнорировать этот узел.
3. *Установить alpha как best_n.* Устанавливаем переменную *alpha* как *best_n.*
4. *alpha больше или равна beta?* Значит, это значение не хуже, чем другие найденные значения, и дальнейшее исследование текущего узла можно прервать.
5. *best_n меньше или равно beta?* Если текущий режим подразумевает выбор минимального значения и текущее лучшее значение меньше или равно beta, то потомки этого узла не содержат более выгодных значений и узел можно игнорировать.
6. *Установить beta как best_n.* Установить переменную *beta* как *best_n.*
7. *alpha больше или равна beta?* Если да, то это значение не лучше других найденных значений и исследование остальной части узла можно прервать.

Псевдокод

Псевдокод для реализации альфа-бета отсечения в основном такой же, что и для минимаксного поиска, с единственным различием: в него добавлено отслеживание значений альфа и бета, а также их сохранение в процессе обхода дерева. Обратите внимание, что в режиме минимума (min) переменная *min_or_max* равна –1, а в режиме максимума (max) она равна 1:

```
minmax_ab_pruning(state, depth, min_or_max, last_move, alpha, beta):
  let current score equal state.get_score
  if current_score is not equal to 0 or state.is_full or depth is equal to 0:
    return new Move(last_move, current_score)
  let best_score equal to min_or_max multiplied by -∞
  let best_move = -1
  for each possible choice (0 to 4 in a 5x4 board) as move:
    let neighbor equal to a copy of state
    execute current move on neighbor
```

```
    let best_neighbor equal
      minmax(neighbor, depth -1, min_or_max * -1, move, alpha, beta)
    if (best_neighbor.score is greater than best_score and min_or_max is MAX)
    or (best_neighbor.score is less than best_score and min_or_max is MIN):
      let best_score = best_neighbor.score
      let best_move = best_neighbor.move
      if best_score >= alpha:
        alpha = best_score
      if best_score <= beta:
        beta = best_score
    if alpha >= beta:
      break
  return new Move(best_move, best_score)
```

Применение алгоритмов состязательного поиска

Алгоритмы состязательного поиска очень гибки и применяются во многих реальных сценариях:

- *Создание игровых агентов для пошаговых игр с совершенной информацией.* В некоторых играх два или более игрока действуют в одной среде. Данный алгоритм успешно применяется в шахматах, шашках и других классических играх. Игры с совершенной информацией — это те, в которых отсутствуют скрытые данные или элемент случайности.
- *Создание игровых агентов для пошаговых игр с несовершенной информацией.* Некоторые игры подразумевают неизвестные исходы, например покер или «Словодел».
- *Состязательный поиск и муравьиный алгоритм (ant colony optimization, ACO) для оптимизации маршрута.* Состязательный поиск используется совместно с алгоритмом ACO, который мы будем рассматривать в главе 6, с целью оптимизации маршрутов доставки посылок в городах.

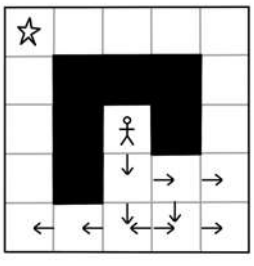
Краткий обзор главы «Умный поиск»

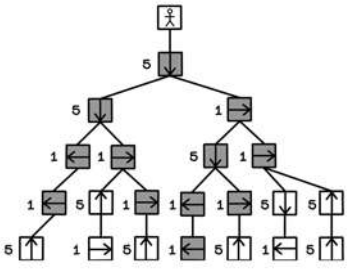
Информированный поиск делает алгоритмы интеллектуальными

Эвристику бывает трудно определить, но грамотно подобранная эвристика очень эффективна при решении задач

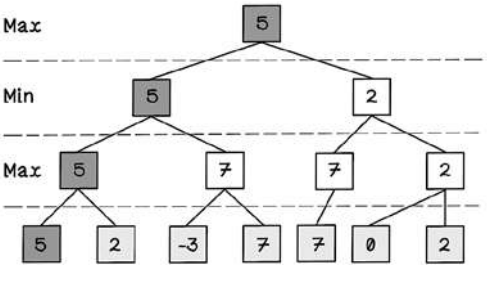
Для эффективного поиска решений алгоритм A* использует эвристику и расстояние от корневого узла

$$f(n) = g(n) + h(n)$$





Состязательный поиск, такой как минимаксный, полезен для работы в изменяющейся среде



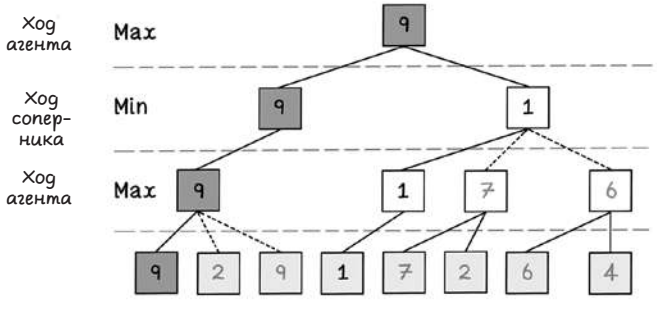
Выбрать дочерние узлы с максимальным значением

Выбрать дочерние узлы с минимальным значением

Выбрать дочерние узлы с максимальным значением

Рассчитать значения для узлов на заданной глубине концевых узлов

Альфа-бета отсечение оптимизирует минимаксный алгоритм, игнорируя неоптимальные пути



Выбрать дочерние узлы с максимальным значением

Выбрать дочерние узлы с минимальным значением

Пропускать узлы, значение которых больше или равно минимальному

Пропускать узлы, значение которых меньше или равно максимальному

4 Эволюционные алгоритмы

В этой главе

- ✓ Как появились эволюционные алгоритмы
- ✓ Решение задач с помощью эволюционных алгоритмов
- ✓ Жизненный цикл генетического алгоритма
- ✓ Проектирование и разработка генетического алгоритма для решения задач оптимизации

Что такое эволюция?

Глядя на окружающий мир, мы иногда задаемся вопросом: «Как все это появилось?» Одно из объяснений дает теория эволюции. Согласно этой теории, современные живые организмы стали такими, какими мы их видим сегодня, не сразу, а в ходе постепенных изменений, длившихся миллионы лет, на протяжении которых каждое новое поколение постепенно адаптировалось к окружающей среде. Из этого следует, что физические и когнитивные характеристики каждого живого организма — это результат наилучшей приспособленности к среде с целью выживания. Эволюция предполагает, что организмы развиваются, производя потомство со смешанными генами родителей. С учетом степени соответствия окружающей среде сильнейшие особи имеют больше шансов на выживание.

Мы часто заблуждаемся, представляя себе эволюцию как линейный процесс с очевидными изменениями в потомстве. В реальности она намного более хаотична и включает в себя расхождение видов. В результате репродукции и смешения генов создается множество различных видов. Причем изменения, заметные при сопоставлении современных представителей вида и их предков, могут проявиться лишь спустя тысячелетия. На рис. 4.1 изображен реальный процесс эволюции человека в сравнении с ошибочным представлением о нем.

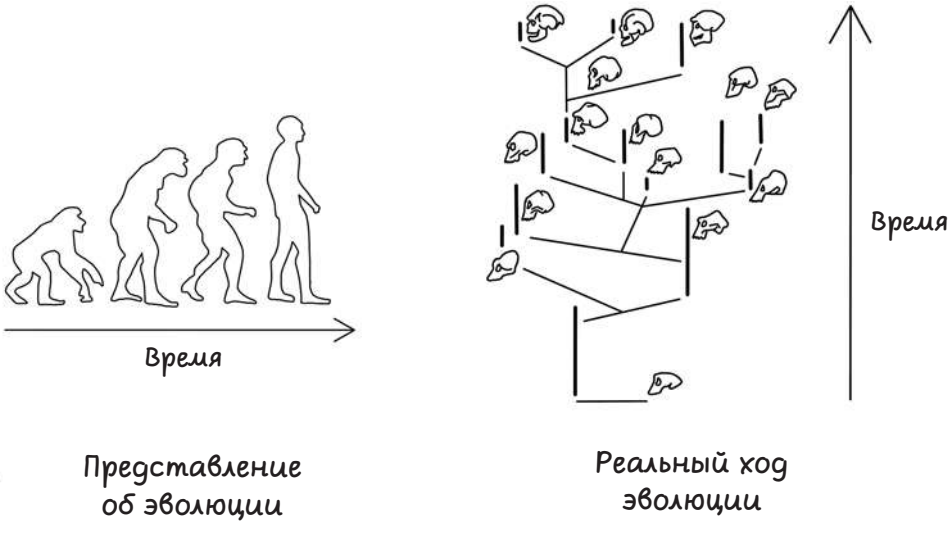


Рис. 4.1. Идея линейной эволюции человека в сравнении с реальным ходом эволюции

Чарльз Дарвин предложил теорию эволюции, основанную на естественном отборе. *Естественный отбор* — это принцип, согласно которому с большей вероятностью выживают сильнейшие особи, наиболее приспособленные к среде обитания. В результате они естественным образом производят больше потомства и, следовательно, передают по наследству признаки, способствующие выживанию, вследствие чего последующие поколения могут потенциально превзойти предков по своим качествам.

Классический пример эволюции с целью приспособления — это бабочка пяденица. Изначально это насекомое имело светлый окрас, который обеспечивал для нее хорошую маскировку от хищников, позволяя сливаться со светлыми поверхностями окружения. При этом всего 2 % популяции этих бабочек имели темную расцветку. Тем не менее после начала промышленной революции темный окрас стал характерен для почти 95 % особей. Одно из объяснений этому в том, что светлые пяденицы больше не могли столь же успешно сливаться с поверхностями, так как в результате загрязнения поверхности стали темнеть. В итоге светлые особи чаще подвергались нападениям хищников, чем темные, что позволило последним более активно размножаться и передавать свои гены потомкам.

В данном случае на самом внешнем уровне у пяденицы изменилась окраска. Тем не менее произошел этот процесс далеко не сразу, соответствующие гены передавались от предков к потомкам последовательно и постепенно.

Есть множество других примеров естественной эволюции, когда помимо простого изменения окраски можно наблюдать более серьезные изменения. Однако за всеми ними стоят глубокие генетические трансформации, которые затрагивают многие поколения (рис. 4.2).

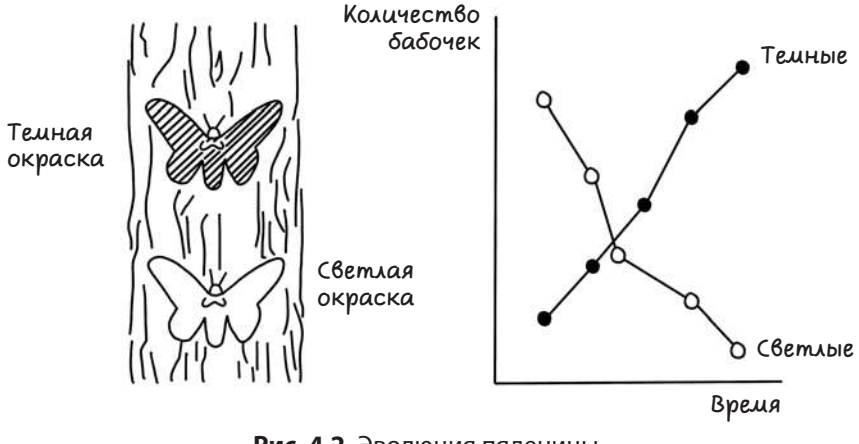


Рис. 4.2. Эволюция пяденицы

Эволюция заключает в себе идею, что размножение в популяции вида происходит путем спаривания двух особей. Генотип потомков содержит в себе комбинацию генов родителей, но при этом в нем происходят изменения, вызванные

мутациями. После рождения потомок становится частью популяции. Тем не менее не все особи в популяции доживают до старости, так как погибают вследствие болезней, травм и других причин. С большей вероятностью выживают те, кто более успешно адаптируется к окружающей среде. Отсюда и произошло выражение *выживает сильнейший*. На основе теории эволюции Дарвина популяция имеет следующие признаки:

- *Разнообразие* — особи имеют разные генетические черты.
- *Наследственность* — потомок наследует генетические особенности родителей.
- *Отбор* — механизм, который определяет, насколько особь соответствует окружающей среде. Более сильные имеют больше шансов на выживание.

Эти признаки подразумевают, что в ходе эволюции происходят следующие процессы (рис. 4.3):

- *Размножение*. Обычно для произведения потомства требуется две особи в популяции.
- *Скрещивание и мутация*. Генотип потомков представляет собой смесь генов родителей и небольшие случайные изменения в собственном генетическом коде.

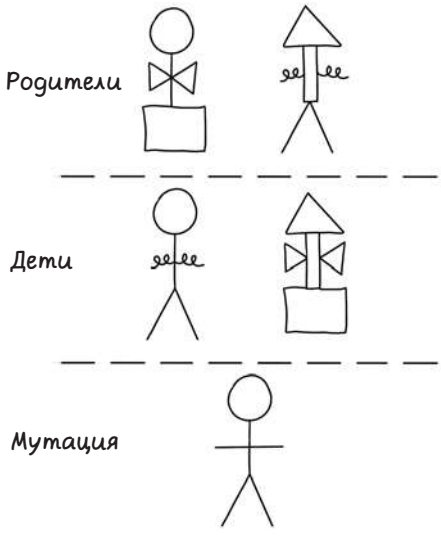


Рис. 4.3. Простой пример размножения и мутации

Если обобщить, то эволюция — это удивительная и неупорядоченная система, порождающая многообразие форм жизни, причем какие-то из них оказываются более подходящими для выполнения конкретных задач в конкретных средах. Эта теория применима и к эволюционным алгоритмам. Научные прин-

ципы биологической эволюции используются для поиска оптимальных решений практических задач путем создания разнообразия и постепенного сужения этого разнообразия до оптимального варианта через много поколений.

Эта глава и следующая глава 5 посвящены изучению эволюционных алгоритмов, представляющих собой мощные, но при этом недооцененные подходы к решению сложных задач. Их можно использовать самостоятельно или с такими конструктами, как нейронные сети. Понимание этих концепций открывает широкие возможности для решения различных современных задач.

Задачи, решаемые с помощью эволюционных алгоритмов

Эволюционные алгоритмы применимы не для всех ситуаций; лучше всего они подходят для таких задач оптимизации, где решение состоит из большого числа перестановок или выборов. Такие задачи обычно предполагают множество возможных решений, среди которых одни оказываются эффективнее других.

Рассмотрим задачу о рюкзаке, классическую задачу, используемую в компьютерной науке для изучения работы алгоритмов и оценки их эффективности. Рюкзак может выдержать определенный вес содержимого. При этом в него можно положить несколько предметов, каждый из которых имеет свой вес и ценность. Цель — положить в рюкзак как можно больше предметов, добившись максимальной общей ценности, не превышая допустимого веса. В простейшей вариации задачи физический размер и форма предметов игнорируются (рис. 4.4).

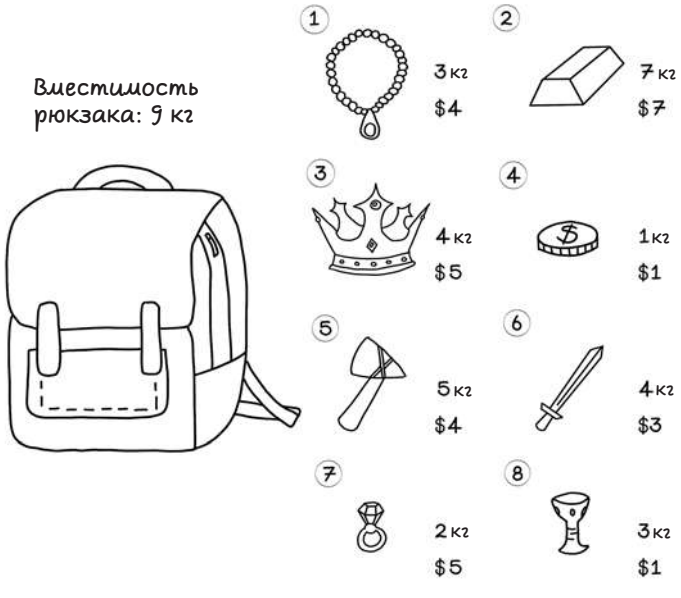


Рис. 4.4. Простой пример задачи о рюкзаке

В качестве базового примера, согласно спецификации, приведенной в табл. 4.1, рюкзак вмещает 9 кг, и положить в него можно любые из восьми предложенных предметов.

Таблица 4.1. Вместимость рюкзака: 9 кг

ID предмета	Название предмета	Вес (кг)	Ценность ($)
1	Жемчуг	3	4
2	Золото	7	7
3	Корона	4	5
4	Монета	1	1
5	Топор	5	4
6	Меч	4	3
7	Кольцо	2	5
8	Чашка	3	1

У этой задачи есть 255 вариантов решений, включая следующие (рис. 4.5):

- *Решение 1*. Положить предметы 1, 4 и 6. Общий вес получится 8 кг при ценности $8.
- *Решение 2*. Положить предметы 1, 3 и 7. Общий вес составит 9 кг при ценности $14.
- *Решение 3*. Положить предметы 2, 3 и 6. Общий вес будет 15 кг, что превысит допустимый для рюкзака предел.

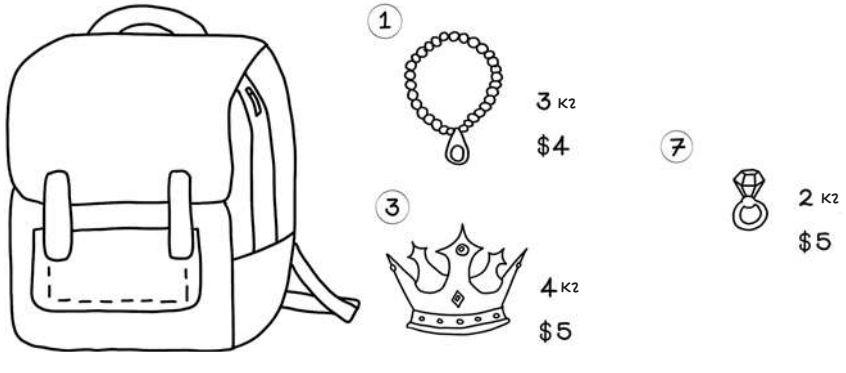


Рис. 4.5. Оптимальное решение простого варианта задачи о рюкзаке

Очевидно, что лучшим оказывается *Решение 2*. Не тратьте время на анализ того, как вычисляется количество вероятностей, лучше поймите, что при увеличении числа доступных предметов происходит взрывной рост количества этих вероятностей.

Несмотря на то что эту простую версию задачи вполне реально решить в уме, задача о рюкзаке может предполагать различные ограничения по весу, разное количество предметов, их разную ценность, что сильно усложнит и даже сделает решение невозможным, поскольку количество вариаций существенно возрастет. В таком случае решение методом подбора также окажется слишком ресурсозатратным. Поэтому мы и ищем алгоритмы, которые смогут эффективно находить приемлемое решение.

Обратите внимание, что мы определяем наилучшее решение, которое можем найти, как *приемлемое*, а не как *оптимальное*. Несмотря на то что некоторые алгоритмы стараются найти единственно верное оптимальное решение задачи о рюкзаке, их эволюционный вариант стремится к этому, но не гарантирует именно такой результат. Он найдет решение, которое будет приемлемым для конкретного случая. Понятие «приемлемое» может трактоваться субъективно, в зависимости от задачи. Например, для критически важной системы здравоохранения «достаточно хорошее» решение не подойдет, но для системы рекомендации музыки окажется вполне допустимым.

Теперь рассмотрим более обширный набор данных (огромный рюкзак) из табл. 4.2, где количество предметов, а также различие в их весе и ценности существенно усложняют задачу. Представив сложность набора данных, можно легко понять, почему многие алгоритмы компьютерной науки оцениваются по их эффективности в решении подобных задач. Эффективность означает, насколько хорошо определенное решение справляется с задачей, и не обязательно подразумевает вычислительную эффективность. В задаче о рюкзаке решение, создающее более высокую общую ценность, будет считаться более эффективным. Эволюционные алгоритмы предоставляют метод поиска решений этой задачи.

Таблица 4.2. Вместимость рюкзака: 6 404 180 кг

ID предмета	Название предмета	Вес (кг)	Ценность ($)
1	Топор	32 252	68 674
2	Бронзовая монета	225 790	471 010
3	Корона	468 164	944 620
4	Алмазная статуя	489 494	962 094
5	Изумрудный пояс	35 384	78 344
6	Кусок руды	265 590	579 152
7	Золотая монета	497 911	902 698
8	Шлем	800 493	1 686 515
9	Чернила	823 576	1 688 691
10	Шкатулка	552 202	1 056 157

ID предмета	Название предмета	Вес (кг)	Ценность ($)
11	Нож	323 618	677 562
12	Длинный меч	382 846	833 132
13	Маска	44 676	99 192
14	Ожерелье	169 738	376 418
15	Опаловая брошь	610 876	1 253 986
16	Жемчуг	854 190	1 853 562
17	Колчан	671 123	1 320 297
18	Рубиновое кольцо	698 180	1 301 637
19	Серебряный браслет	446 517	859 835
20	Хронометр	909 620	1 677 534
21	Униформа	904 818	1 910 501
22	Ядовитое зелье	730 061	1 528 646
23	Шерстяной шарф	931 932	1 827 477
24	Арбалет	952 360	2 068 204
25	Прошлогодняя книга	926 023	1 746 556
26	Чаша из цинка	978 724	2 100 851

Один из способов решения данной задачи — метод полного перебора, или брутфорс. Этот подход подразумевает вычисление каждой возможной комбинации предметов с определением ее ценности при выполнении условия по ограничению веса рюкзака, пока не будет найдено лучшее решение.

На рис. 4.6 показаны некоторые бенчмарки такого подхода. Имейте в виду, что они основаны на вычислительных возможностях среднего ПК.

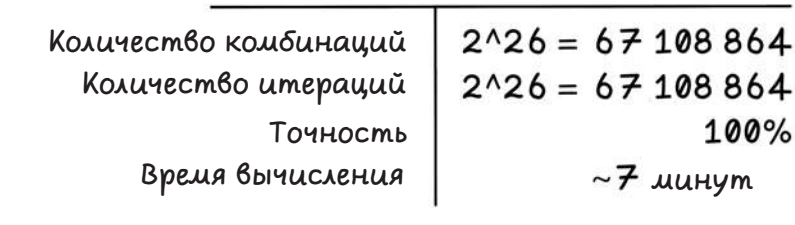


Рис. 4.6. Анализ производительности метода полного перебора для задачи о рюкзаке

Запомните эту задачу, поскольку мы еще не раз будем возвращаться к ней по мере изучения, проектирования и разработки генетического алгоритма для поиска приемлемых решений.

Примечание

Небольшое пояснение о термине «производительность». С точки зрения отдельного решения производительность означает, насколько хорошо это решение справляется с задачей. С точки зрения алгоритма она может означать то, насколько хорошо его конкретная конфигурация подходит для поиска решения. И наконец, производительность может означать вычислительные циклы. Помните, что этот термин может иметь различные значения в зависимости от контекста.

Логика использования генетического алгоритма в решении задачи о рюкзаке применима в различных ситуациях. Если, например, логистической компании нужно оптимизировать загрузку фур на основе очередности точек маршрута, то генетический алгоритм окажется весьма кстати. Если той же компании понадобится найти кратчайший путь между несколькими точками маршрута, то он также будет полезен. Если фабрика производит некую продукцию из отдельных комплектующих с использованием конвейера и порядок появления комплектующих на конвейере влияет на продуктивность, то генетический алгоритм поможет найти оптимальный порядок.

Когда мы углубляемся в логику, подход и жизненный цикл генетического алгоритма, то понимаем, где его можно использовать. Так что вы наверняка найдете ему применение и в своих разработках. Важно помнить, что этот алгоритм *стохастичен*, то есть его результат будет отличаться при каждом выполнении.

Генетический алгоритм: жизненный цикл

Каждый эволюционный алгоритм, включая генетический, имеет свои особенности. Основываются все эти алгоритмы на одном принципе эволюции, но различаются деталями жизненных циклов, что позволяет решать различные задачи. Некоторые из этих особенностей мы будем разбирать в главе 5.

Генетические алгоритмы используются для анализа больших пространств решений в поиске наиболее подходящего. Важно отметить, что этот алгоритм не гарантирует обнаружения наилучшего решения. Он стремится найти глобальное лучшее, избегая при этом локальных лучших.

Глобальное лучшее представляет наилучшее возможное решение, *локальное* же является менее оптимальным вариантом. На рис. 4.7 показаны возможные лучшие решения при условии поиска минимальных вариантов, то есть чем меньше значение, тем лучше. Если же целью будет поиск максимального значения, то условие изменится на противоположное. Алгоритмы оптимизации, такие как генетический, пошагово находят локальное лучшее, стремясь обнаружить среди них глобальное лучшее.

Рис. 4.7. Локальные и глобальное лучшие решения

Конфигурацию параметров алгоритма нужно производить очень внимательно, стремясь обеспечить разнообразие решений в начале и постепенное приближение к лучшим вариантам с каждым поколением. На старте возможные решения должны значительно различаться в отдельных генетических свойствах. Без такого начального расхождения возникает риск застревания в области локального лучшего решения (рис. 4.8).

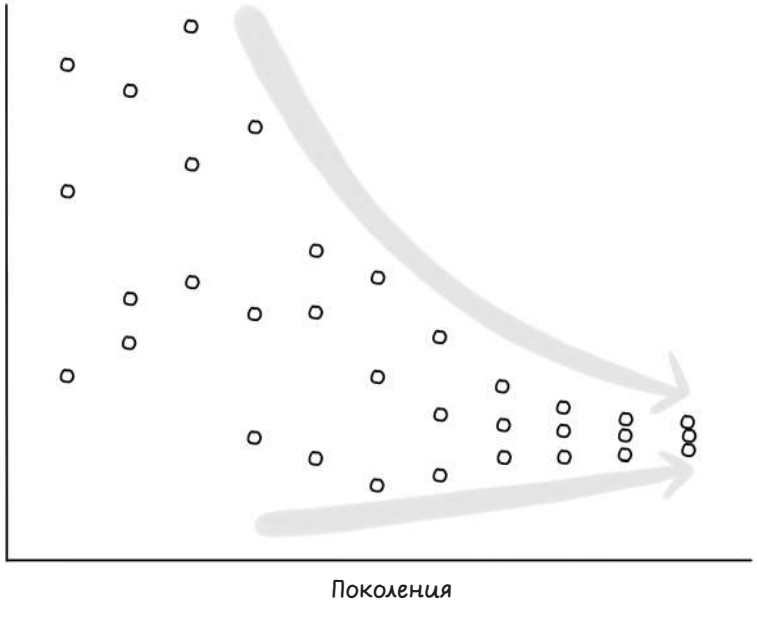


Рис. 4.8. От разнообразия к конвергенции

Конфигурация генетического алгоритма определяется областью задачи. Каждая задача имеет свой уникальный контекст и область, в которой представлены данные. В связи с этим решения оцениваются по-разному.

Общий жизненный цикл генетического алгоритма выглядит так:

- *Создание популяции*. Создание случайной популяции возможных решений.
- *Оценка приспособленности отдельных особей в популяции*. Выяснение, насколько хорошо конкретное решение. Эта задача реализуется с помощью функции приспособленности, или фитнес-функции, которая оценивает решения.
- *Выбор родителей на основе их приспособленности*. Выбор пар родителей для воспроизводства потомка.
- *Воспроизводство потомства*. Создание потомка от родителей путем смешения генетической информации и применения небольшой мутации.
- *Генерация следующего поколения*. Выбор из популяции особей и потомков, которые выживут до следующего поколения.

Реализация генетического алгоритма происходит в несколько шагов, которые включают в себя стадии его жизненного цикла (рис. 4.9).

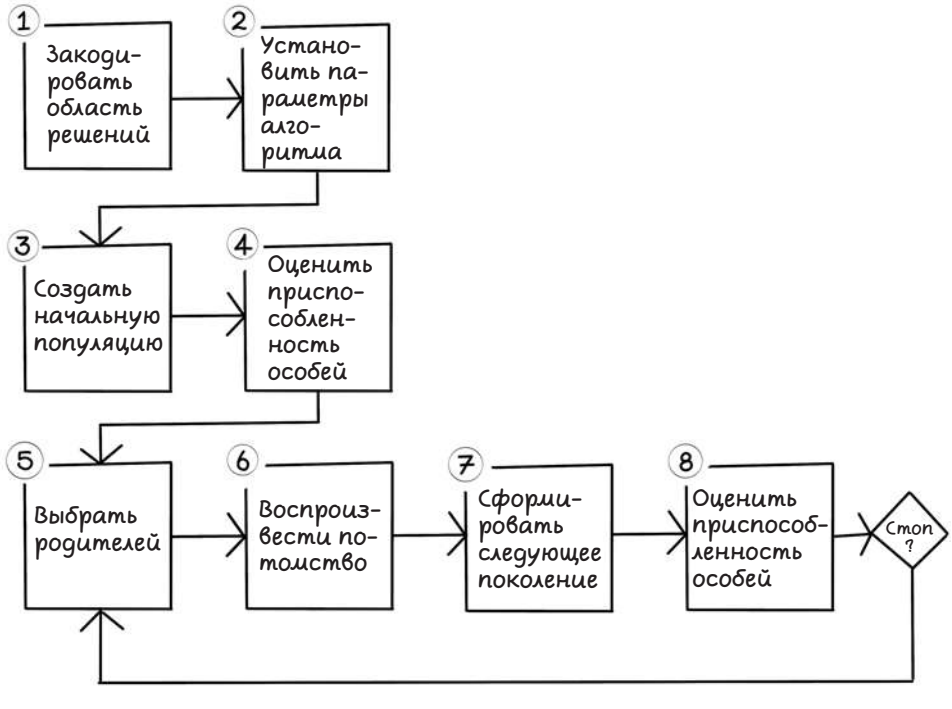


Рис. 4.9. Жизненный цикл генетического алгоритма

Если вернуться к задаче о рюкзаке, подумайте, как бы вы использовали этот алгоритм для поиска ее решений? Следующий раздел посвящен именно этому.

Кодирование пространства решений

При использовании генетического алгоритма первостепенная задача состоит в правильном кодировании, для чего нужно грамотно построить представление возможных состояний. *Состояние* — это структура данных с определенными правилами, которая представляет возможные решения задачи. Коллекция состояний формирует популяцию (рис. 4.10).

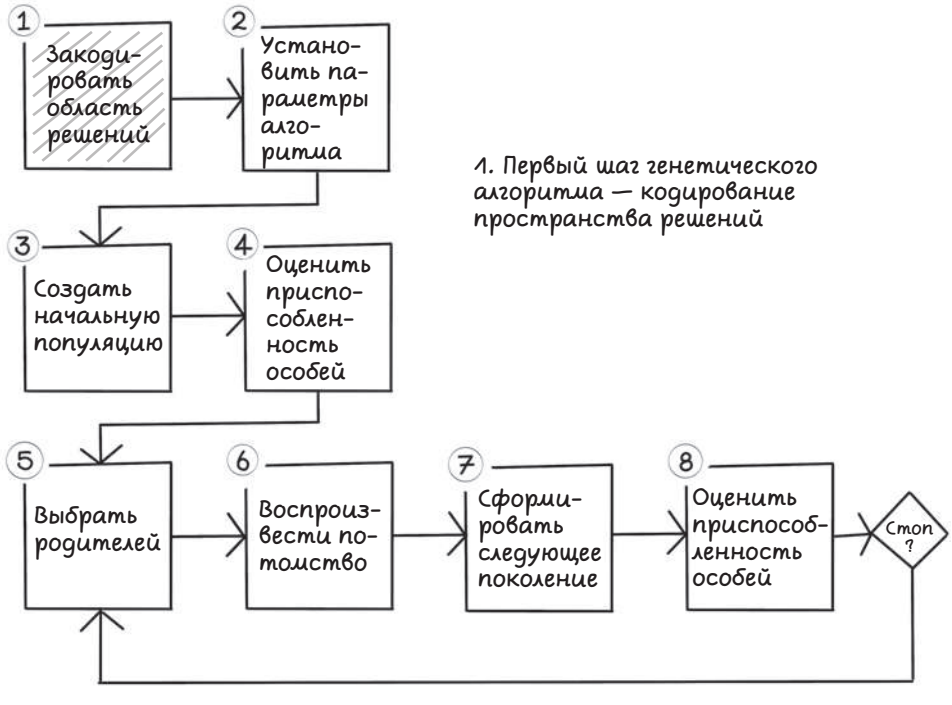


Рис. 4.10. Кодирование решения

Терминология

В эволюционных алгоритмах отдельное возможное решение называется *хромосомой*. Хромосома состоит из генов. *Ген* — это элемент логического типа, а *аллель* — фактическое хранящееся в этом элементе значение. *Генотип* — это представление решения, а *фенотип* — это само уникальное решение. Количество генов в любой хромосоме всегда неизменно. Коллекция хромосом формирует популяцию (рис. 4.11).

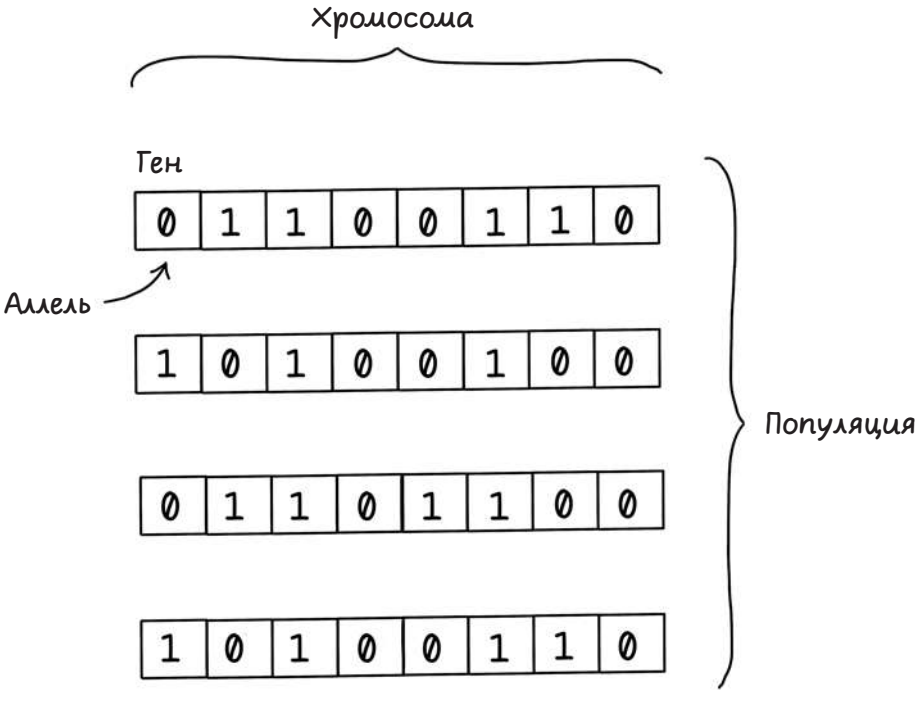


Рис. 4.11. Терминология структур данных, представляющих популяцию решений

В задаче о рюкзаке можно положить в рюкзак несколько предметов. Простым способом описать возможное решение, включающее определенные предметы, будет двоичное кодирование (рис. 4.12). *Двоичное кодирование* представляет невключенные предметы в виде 0, а добавленные в виде 1. Если, например, значение в гене по индексу 3 будет равно 1, значит, данный элемент добавлен в рюкзак. Полная двоичная строка всегда имеет одинаковый размер, соответствующий количеству доступных для выбора предметов. Тем не менее существует несколько альтернативных схем кодирования, которые описываются в главе 5.

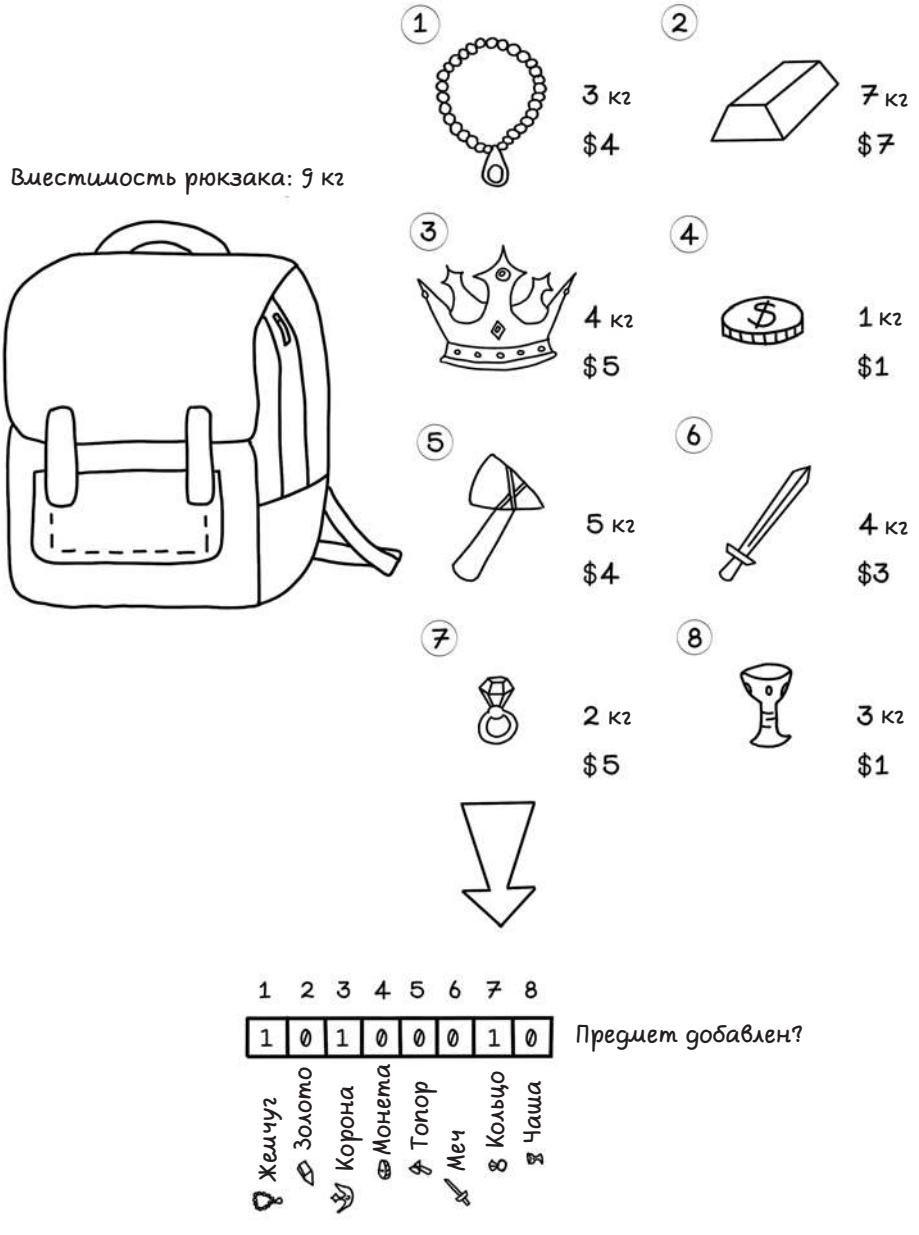


Рис. 4.12. Двоичное кодирование задачи о рюкзаке

Двоичное кодирование: представление возможных решений через 0 и 1

Двоичное кодирование представляет ген как 0 или 1. В результате хромосома являет собой строку двоичных битов. Чтобы показать наличие конкретного предмета в рюкзаке, применяют разные способы двоичного кодирования, включая кодирование численных значений в двоичном виде. Преимущество такого кодирования в том, что ввиду использования примитивных типов его производительность выше. Его применение снижает нагрузку на память компьютера, к тому же, в зависимости от используемого языка программирования, двоичные операции выполняются быстрее. Тем не менее необходимо всякий раз оценивать, насколько оправданно его применение в задаче и можно ли с его помощью должным образом представить возможные решения. В противном случае эффективность алгоритма может оказаться невысока (рис. 4.13).

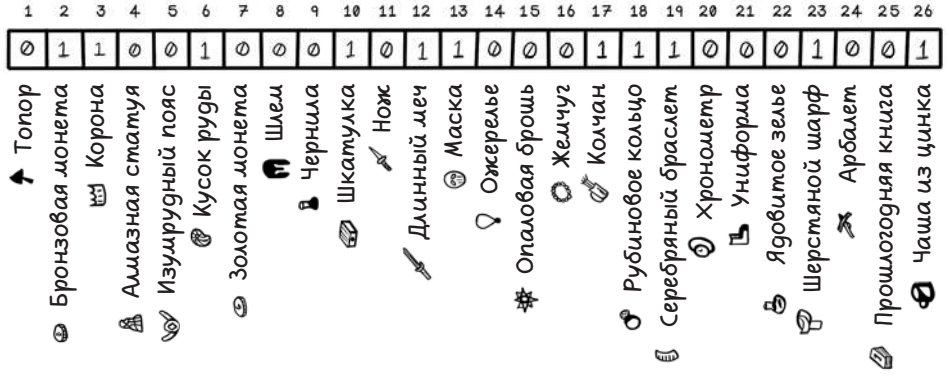


Рис. 4.13. Двоичное кодирование большого набора данных для задачи о рюкзаке

При рассмотрении задачи с набором из 26 предметов, имеющих различный вес и ценность, с помощью двоичной строки можно представить, добавлены предметы в рюкзак или нет. В итоге получится строка из 26 знаков, в которой для каждого индекса 0 означает отсутствие в рюкзаке предмета, а 1 — его наличие.

Другие схемы, включая вещественное, перестановочное и древовидное кодирование, будут рассмотрены в главе 5.

УПРАЖНЕНИЕ: ПРЕДЛОЖИТЕ ВОЗМОЖНОЕ КОДИРОВАНИЕ ДЛЯ СЛЕДУЮЩЕЙ ЗАДАЧИ.

Дано предложение. С помощью генетического алгоритма определите, какие слова из него можно удалить так, чтобы оставшиеся слова образовывали осмысленные фразы.[1]

```
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
```

Неверно

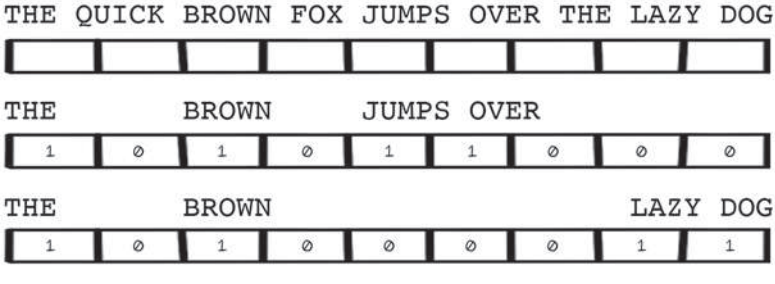
```
THE         BROWN       JUMPS OVER
    QUICK         FOX         OVER THE
THE               FOX               THE LAZY
```

Верно

```
THE QUICK         FOX
    QUICK         FOX JUMPS
THE         BROWN FOX                     DOG
THE         BROWN                   LAZY  DOG
THE QUICK                                 DOG
    QUICK                   OVER THE      DOG     * Знаки препинания
THE QUICK                             LAZY  DOG     опущены.
```

ОТВЕТ: ВОЗМОЖНОЕ КОДИРОВАНИЕ.

Так как количество возможных слов всегда одинаково и находятся они всегда в одном и том же месте, с помощью двоичного кодирования можно описать, какие слова добавлены, а какие нет. Хромосома состоит из 9 генов, каждый из которых обозначает слово из фразы.



[1] В задании использована фраза-панграмма The quick brown fox jumps over the lazy dog («Шустрая бурая лисица прыгает через ленивого пса»), содержащая все буквы английского алфавита. В целях сохранения логики задания фраза приведена без перевода. — *Примеч. ред.*

Создание популяции решений

В начале создается популяция. Первый шаг генетического алгоритма — это инициализация случайных возможных решений задачи. В этом процессе, несмотря на то что хромосомы генерируются случайно, необходимо учитывать ограничения задачи, то есть возможные решения должны быть действительными либо, в случае нарушения ограничений, получать нулевое значение приспособленности (нулевое фитнес-функции). Как говорилось в предыдущем примере упаковки предметов в рюкзак, решение, включающее добавление одного предмета более одного раза, должно считаться недопустимым и не включаться в популяцию возможных решений (рис. 4.14).

Рис. 4.14. Создание начальной популяции

В соответствии с представленным способом решения задачи о рюкзаке, алгоритм для каждого предмета случайным образом определяет, нужно ли класть его в рюкзак. При этом рассматриваться должны только те решения, которые не выходят за рамки ограничения максимального веса поклажи. Однако здесь есть проблема. Если просто перебирать предметы слева направо, определяя, нужно ли класть их в рюкзак, то возникает смещение в сторону тех, которые расположены на левом конце хромосомы, то есть вероятность их добавления возрастет. То же происходит, если начать перебирать предметы

справа налево, но в этом случае возникает смещение уже в сторону предметов на правом конце. Обойти это смещение можно, к примеру, так: генерировать отдельную особь со случайными генами, а затем определять, является ли это решение допустимым и не нарушает ли ограничений. Присваивание недопустимым решениям нулевого значения функции приспособленности решит описанную проблему (рис. 4.15).

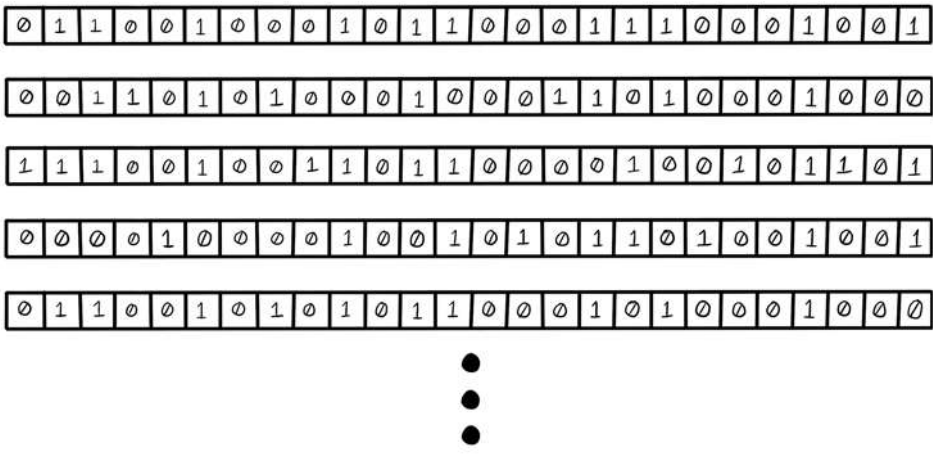


Рис. 4.15. Пример популяции решений

Псевдокод

Чтобы сгенерировать начальную популяцию возможных решений, создается пустой массив для хранения особей. Затем для каждой особи популяции также создается пустой массив, который будет содержать гены этой особи. Каждый ген случайным образом определяется как 1 или 0, указывая, добавлен ли в рюкзак предмет, соответствующий этому гену:

```
generate_initial_population (population_size, individual_size)
    let population be an empty array
    for individual in range 0 to population_size
        let current_individual be an empty array
        for gene in range 0 to individual_size
            let random_gene be 0 or 1 randomly
            append random_gene to current_individual
        append current_individual to population
    return population
```

Оценка приспособленности особей популяции

После создания популяции нужно определить приспособленность каждой ее особи, то есть выяснить, насколько хорошо работает решение. Фитнес-функция — неотъемлемая часть жизненного цикла генетического алгоритма. Если приспособленность особей оценить ошибочно или таким способом, который не обнаруживает оптимальное решение, это негативно скажется на процессе отбора родителей новых особей и последующих поколений. В результате такой алгоритм не сможет найти оптимальное решение. Функции приспособленности аналогичны эвристикам, изученным в главе 3. Они служат ориентирами при нахождении хороших решений (рис. 4.16).

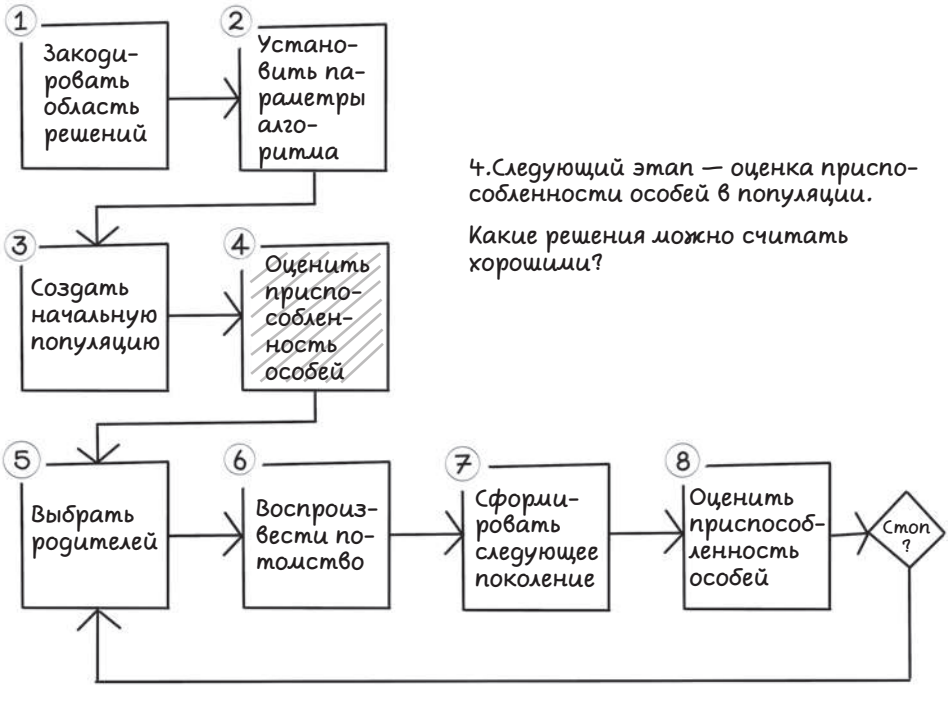


Рис. 4.16. Оценка приспособленности особей

В нашем примере решение стремится максимизировать ценность предметов в рюкзаке, учитывая при этом ограничение по его весу. Фитнес-функция измеряет общую ценность предметов в рюкзаке для каждой особи. В результате особи с более высоким показателем считаются более приспособленными. Обратите внимание, что на рис. 4.17 наглядно представлен недопустимый вариант особи с нулевым значением приспособленности, потому что в этом

решении нарушается условие максимального веса рюкзака, который равен 6 404 180 кг.

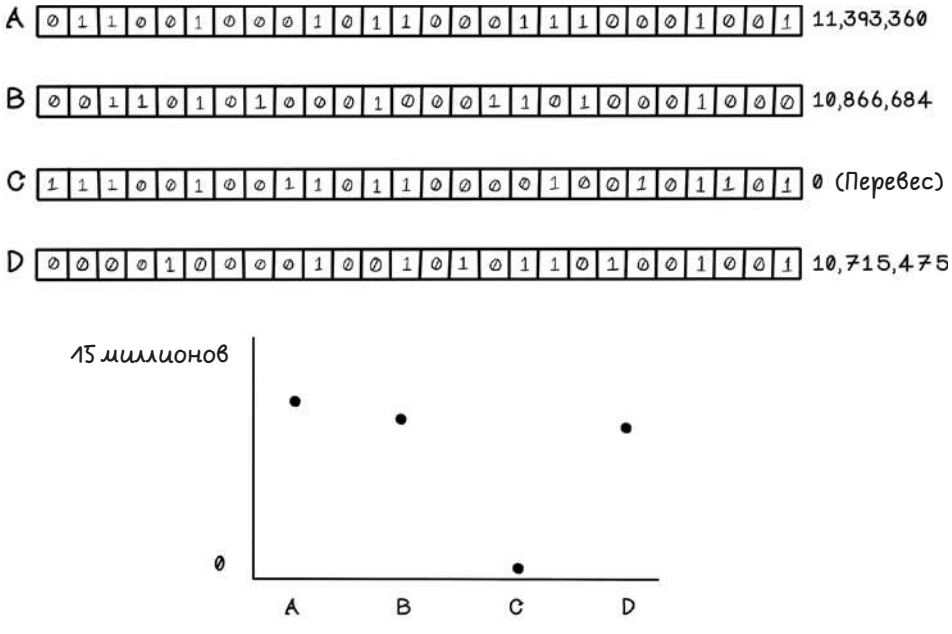


Рис. 4.17. Оценка приспособленности особей

В зависимости от решаемой задачи фитнес-функция может стремиться минимизировать или максимизировать результат. В задаче о рюкзаке можно либо максимально увеличить его содержимое в рамках ограничений, либо максимально сократить пустое пространство. Выбор одного из этих подходов будет зависеть от интерпретации задачи.

Псевдокод

Чтобы вычислить приспособленность особи в задаче о рюкзаке, нужно узнать сумму ценности предметов, принадлежащих каждой отдельной особи. Для этого начальная общая ценность устанавливается как 0, после чего выполняется перебор каждого гена, чтобы определить, добавлен ли в рюкзак соответствующий предмет. Если да, то его ценность добавляется к общей. Аналогичным образом для каждого решения вычисляют общий вес, определяя допустимость решения. Оценку приспособленности и проверку соответствия ограничениям можно разделить, чтобы явно разграничить эти задачи:

```
calculate_individual_fitness (individual,
                              knapsack_items,
                              knapsack_max_weight)
  let total_weight equal 0
  let total_value equal 0
  for gene_index in range 0 to length of individual
    let current_bit equal individual[gene_index]
    if current_bit equals 1
      add weight of knapsack_items[gene_index] to total_weight
      add value of knapsack_items[gene_index] to total_value
  if total_weight is greater than knapsack_max_weight
    return value as 0 since it exceeds the weight constraint
  return total_value as individual fitness
```

Отбор родителей на основе их приспособленности

Следующим шагом алгоритма идет отбор родителей для воспроизводства новых особей. В теории Дарвина у наиболее приспособленных особей вероятность воспроизведения больше, потому что живут они, как правило, дольше. Более того, поскольку эти особи лучше приспособлены к окружающей среде, их признаки предпочтительнее для наследования. Тем не менее некоторые особи могут воспроизводить потомство, даже не будучи лидирующими представителями вида, и обладать сильными чертами, не будучи сильными в целом.

Для каждой особи вычисляется ее приспособленность, используемая для определения вероятности выбора этой особи в качестве родителя. Это свойство делает генетический алгоритм стохастическим по своей природе (рис. 4.18).

Часто в качестве инструмента для отбора родителей на основе их приспособленности используют *колесо рулетки*. В этом подходе особям на основе степени их приспособленности выделяется определенная доля колеса. То есть чем выше приспособленность, тем большую секцию поля занимает особь, и шанс ее выбора возрастает. После этого колесо «вращается» и выбирается победитель. Этот процесс повторяется, пока не будет отобрано нужное количество родителей.

Колесо вычисляет вероятности для 16 особей с различной степенью приспособленности и выделяет секцию для каждой особи. Поскольку у многих особей приспособленность похожа, для них выделяются секции примерно одинакового размера (рис. 4.19).

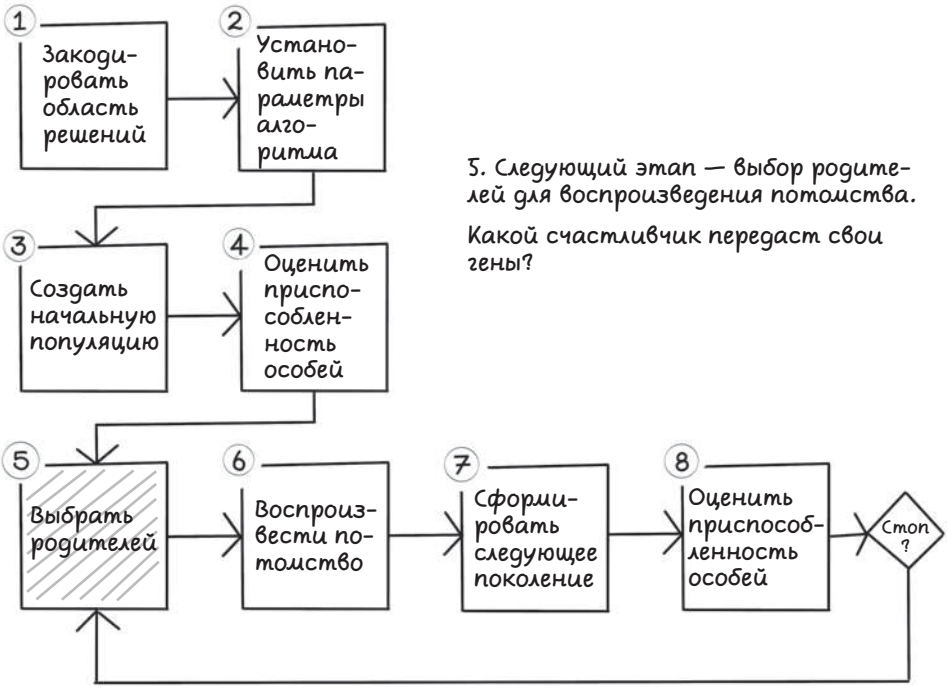


Рис. 4.18. Отбор родителей

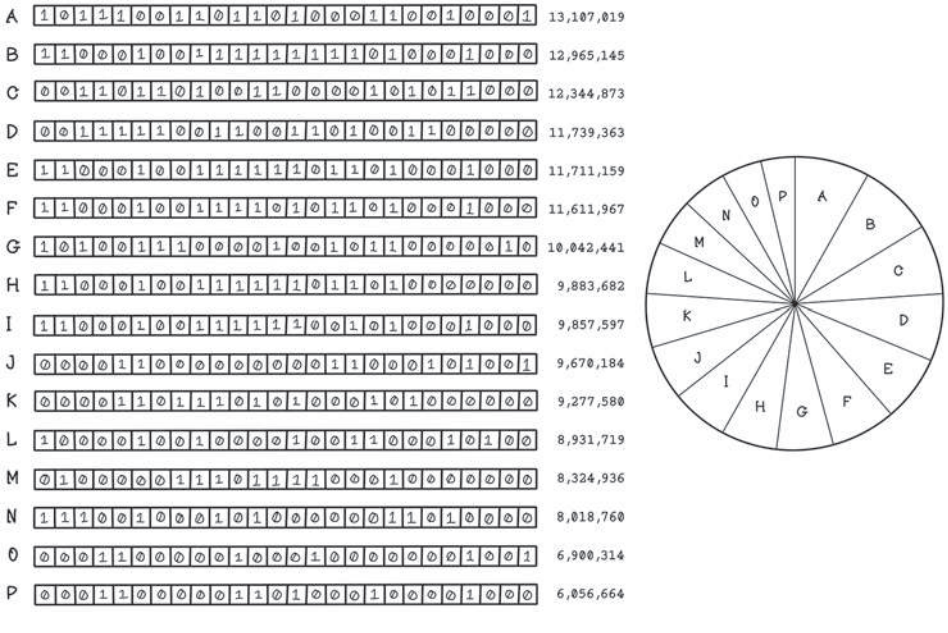


Рис. 4.19. Определение вероятности выбора для каждой особи

Количество отобранных родителей определяется согласно общему требуемому количеству потомков, которое, в свою очередь, обусловливается требуемым размером популяции для каждого поколения. Выбираются два родителя, и создается потомок. Этот процесс повторяется для разных выбранных родителей (с возможностью повторного участия особей в воспроизводстве потомства) до момента достижения нужного числа потомков. Двое родителей могут воспроизвести одного или двух смешанных потомков. Этот принцип будет пояснен подробнее несколько позже. В примере задачи о рюкзаке особи с большей степенью приспособленности — это такие, которые заполняют рюкзак с наибольшей общей ценностью, не превышая ограничения по весу.

Для контроля разнообразия популяции используются различные подходы, в частности стратегия стационарного состояния и «поколенческая» стратегия, каждая из которых имеет свои преимущества и недостатки.

Стационарное состояние: смена в каждом новом поколении только части популяции

Этот высокоуровневый подход управления популяцией не заменяет другие стратегии отбора, а служит схемой их использования. Его идея состоит в сохранении большей части популяции и замене новым потомством только небольшой группы наиболее слабых особей. Этот процесс подобен циклам жизни и смерти, в которых более слабые особи умирают и на их смену вследствие размножения приходят новые. Если популяция состоит из 100 особей, то в следующем поколении, например, 80 из них продолжат существовать, а 20 будут замещены потомством.

Поколенческая модель: полная смена популяции каждое поколение

Этот высокоуровневый подход к управлению популяцией аналогичен предыдущему и не заменяет другие стратегии отбора. В поколенческой модели создается столько же потомков, сколько особей есть в текущей популяции, и популяция полностью ими заменяется. Если популяция состоит из 100 особей, то каждое поколение будет производить 100 потомков. Стратегия стационарного состояния и поколенческая стратегия — ключевые принципы построения конфигурации алгоритма.

Колесо рулетки: отбор родителей и выживших особей

У хромосом с более высоким показателем приспособленности больше шансов быть выбранными, но это не мешает хромосомам с низким показателем также пройти отбор. Выражение *отбор колесом рулетки* взято из индустрии казино, где игровое поле рулетки разделено на секции. В начале игры колесо раскручивают, после чего в него вбрасывается шарик. По завершении вращения выигрывает секция, где в итоге остановился шарик.

Согласно такому же принципу хромосомы приравниваются к секциям колеса. Чем выше приспособленность хромосомы, тем больше окажется размер ее секции. Далее по аналогии с шариком в рулетке случайным образом выбирается выигравшая хромосома.

Эта аналогия — пример вероятностного отбора. У каждой особи есть шанс пройти отбор. В результате от этого шанса зависит конечное разнообразие популяции и скорость ее конвергенции, о которой мы говорили ранее в этой главе. Рисунок 4.19 выше также демонстрирует данный принцип.

Псевдокод

Сначала нужно определить вероятность выбора каждой особи. Эта вероятность вычисляется путем деления показателя приспособленности особи на общую приспособленность популяции. Далее можно использовать отбор рулеткой. Колесо вращается столько раз, сколько требуется выбрать особей. Для каждого отбора вычисляется случайное десятичное число между 0 и 1. Если приспособленность особи оказывается в рамках этой вероятности, особь отбирается. Для определения вероятности отбора особи можно использовать и другие подходы, включая стандартное отклонение, в котором значение особи сравнивается со средним значением группы:

```
set_probabilities_of_population (population)
  let total_fitness equal the sum of fitness of the population
  for individual in population
    let the probability_of_selection of individual...
        ...equal it's fitness/total_fitness

roulette_wheel_selection(population, number_of_selections):
  let possible_probabilities equal
      set_probabilities_of_population (population)
  let slices equal empty array
  let total equal 0
  for i in range(0, number_of_selections):
    append [i, total, total + possible_probabilities[i]]
      to slices
    total += possible_probabilities[i]
  let spin equal random(0, 1)
  let result equal [slice for slice in slices if slice[1] < spin <= slice[2]]
  return result
```

Воспроизведение особей родителями

После отбора родителей наступает этап воспроизведения ими нового потомства. Как правило, этот процесс состоит из двух шагов. Первый — это *кроссинговер (кроссовер)*, то есть смешивание части хромосомы одного родителя с частью хромосомы другого и наоборот. В результате этого процесса получаются два потомка, которые содержат инверсионную смесь генов родителей. Второй шаг — это *мутация*, подразумевающая небольшое случайное изменение потомка для увеличения разнообразия популяции (рис. 4.20).

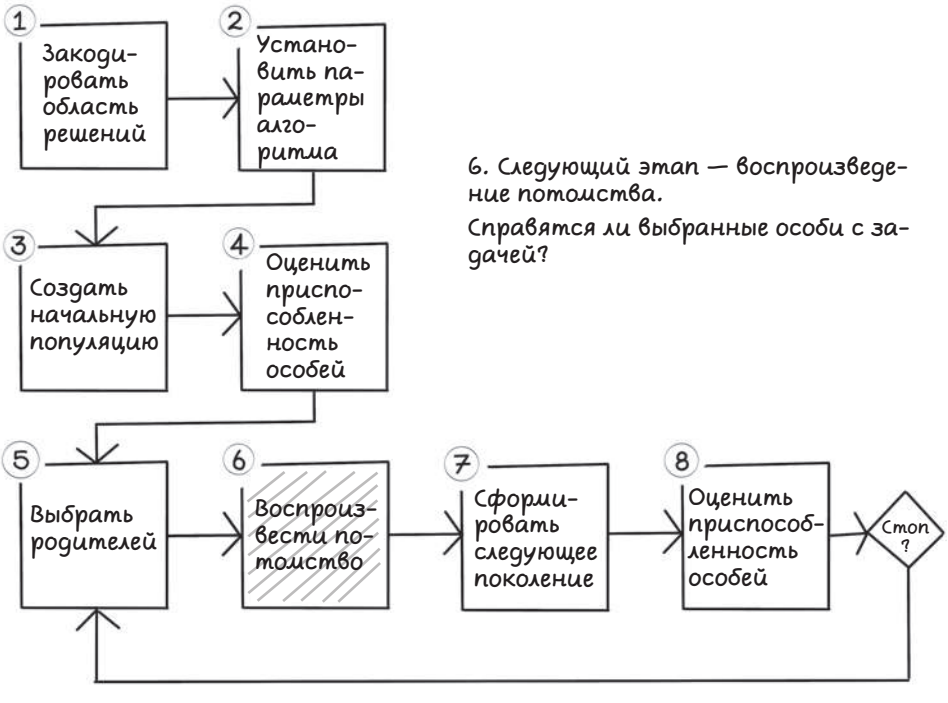


Рис. 4.20. Воспроизведение потомства

Кроссинговер

Кроссинговер — это процесс смешения генов двух особей для создания одного или более потомков. В основе этого процесса лежит принцип размножения. Потомки являются частью своих родителей, в зависимости от используемой стратегии кроссинговера, которая, в свою очередь, сильно зависит от стратегии кодирования.

Одноточечный кроссинговер: наследование одной части от каждого родителя

В этом случае выбирается одна точка в структуре хромосомы. Далее происходит обращение к обоим родителям, и от первого из них берется первая часть, а от второго вторая. В результате совмещения этих частей получается потомок. Второй потомок может получиться путем использования второй части первого родителя и первой части второго, то есть инверсионным способом.

Одноточечный кроссинговер применим к двоичному кодированию, перестановочному/порядковому или вещественному (рис. 4.21). Эти схемы кодирования будут рассматриваться в главе 5.

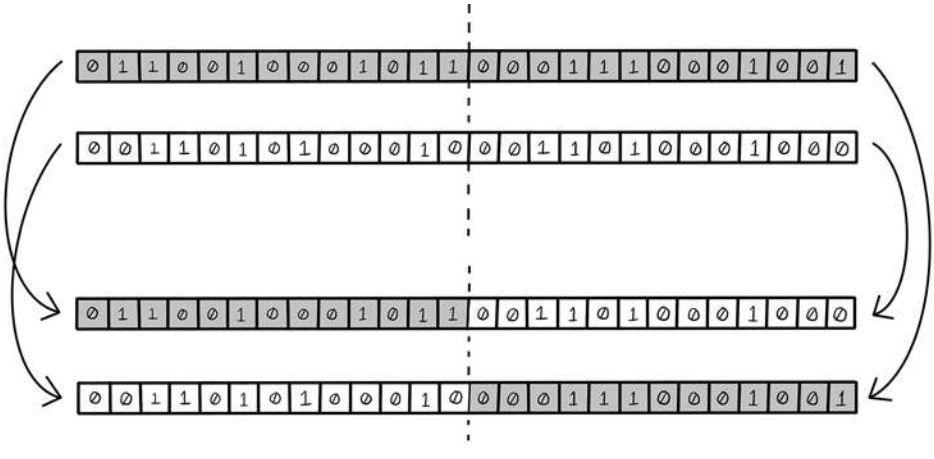


Рис. 4.21. Одноточечный кроссинговер

Псевдокод

При воспроизведении двух потомков создается пустой массив, в котором они будут храниться. Все гены от индекса 0 до нужного индекса родителя A объединяются со всеми генами от нужного индекса до конца хромосомы родителя B, создавая одного потомка. Второй потомок создается инверсионным порядком объединения частей генов родителей:

```
one_point_crossover (parent_a, parent_b, xover_point)
    let children equal empty array

    let child_1 equal genes 0 to xover_point from parent_a plus...
    ...genes xover_point to parent_b length from parent_b
    append child_1 to children
```

```
let child_2 equal genes 0 to xover_point from parent_b plus...
...genes xover_point to parent_a length from parent_a
append child_2 to children

return children
```

Двухточечный кроссинговер: наследование двух частей от каждого родителя

Выбираются две точки в структуре хромосомы. Далее через поочередное обращение к родителям выбираются их части генов, из которых и составляется потомок. Этот процесс аналогичен рассмотренному выше одноточечному кроссинговеру. Если описать его подробно, то потомок составляется из первой части первого родителя, второй части второго и третьей части первого. Можно представить двухточечный кроссинговер как разделение массивов с целью создания из их частей новых массивов. Этот вид кроссинговера применим к двоичному и вещественному кодированию (рис. 4.22).

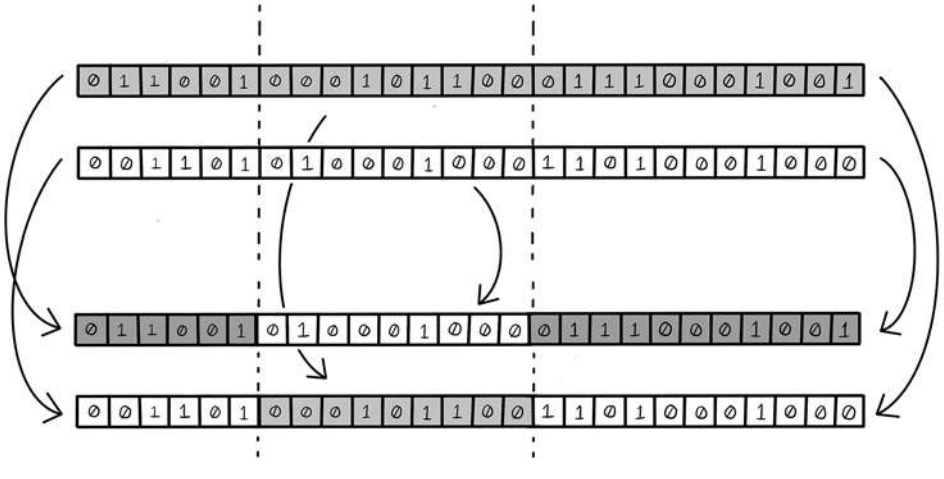


Рис. 4.22. Двухточечный кроссинговер

Однородный кроссинговер: наследование множества частей от обоих родителей

Однородный кроссинговер — это усовершенствованный двухточечный кроссинговер. Он подразумевает создание маски, представляющей, какие гены от каждого родителя будут использованы для генерации потомка. Для создания

второго потомка может применяться инверсионный процесс. Чтобы добиться максимального разнообразия, маску можно генерировать случайным образом при создании каждого потомка. Говоря в общем, в результате однородного кроссинговера рождаются более разнообразные особи, потому что признаки потомка сильно отличаются от любого из родителей. Данный кроссинговер применим к двоичному и вещественному кодированию (рис. 4.23).

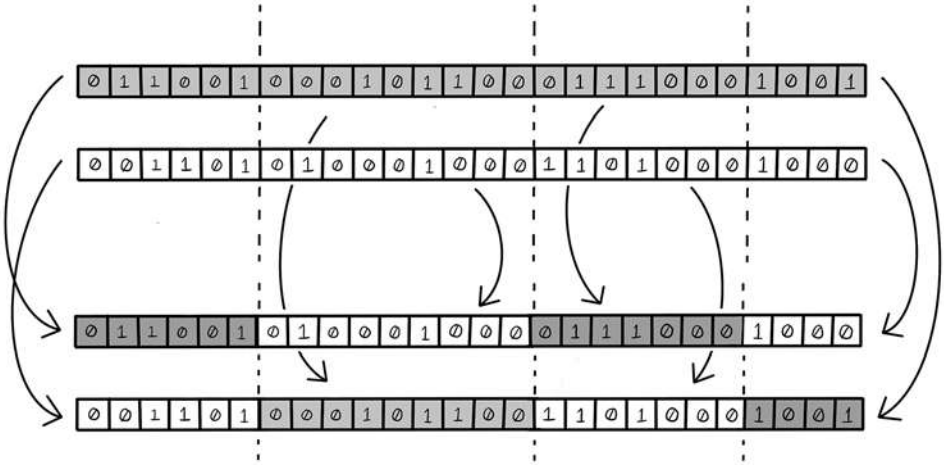


Рис. 4.23. Равномерный кроссинговер

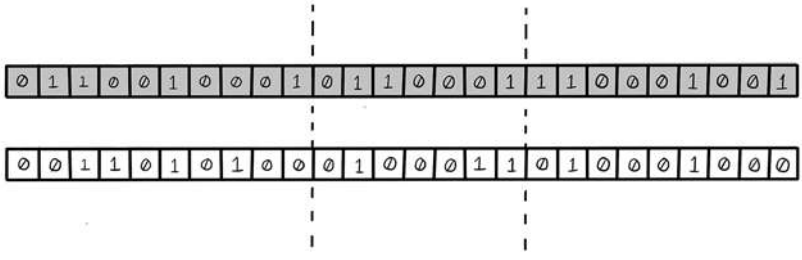
Мутация

Мутация подразумевает небольшое изменение потомков с целью разнообразия популяции. В зависимости от задачи и метода кодирования могут использоваться различные способы мутации.

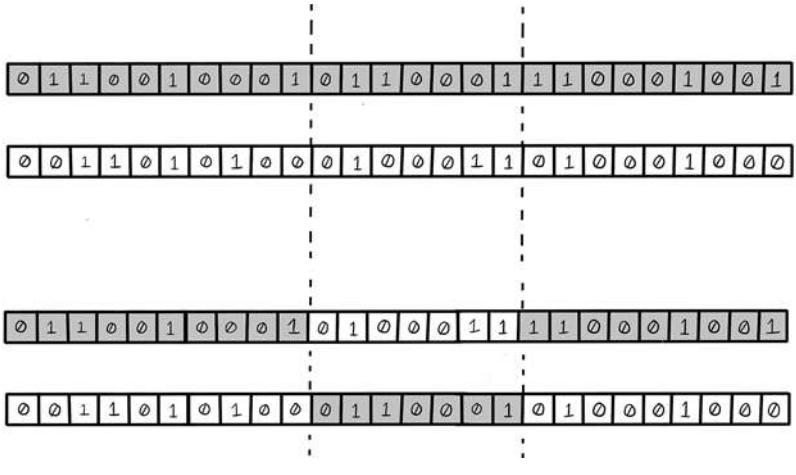
Одним из параметров мутации является ее частота — вероятность, что хромосома потомка будет изменена. По аналогии с живыми организмами некоторые хромосомы изменяются больше других. В результате генотип потомка является не точной комбинацией хромосом родителей, а содержит небольшие отличия. Мутация важна для обеспечения разнообразия в популяции и предотвращения застревания алгоритма в локальных лучших решениях.

Высокая частота мутаций означает либо высокую вероятность выбора особи для мутации, либо высокую вероятность мутации генов в хромосоме особи, что будет зависеть от выбранной стратегии мутирования. Высокий показатель мутации предполагает большее разнообразие, но нужно учитывать, что чрезмерное разнообразие может привести к отклонению от хороших решений.

УПРАЖНЕНИЕ: УКАЖИТЕ РЕЗУЛЬТАТ ОДНОРОДНОГО КРОССИНГОВЕРА ЭТИХ ХРОМОСОМ.



ОТВЕТ: РЕЗУЛЬТАТ ОДНОРОДНОГО КРОССИНГОВЕРА ХРОМОСОМ.



Мутация битовых строк для двоичного кодирования

При мутации битовых строк в двоично-кодированной хромосоме случайным образом выбирается ген и изменяется на другое допустимое значение (рис. 4.24). Если используется недвоичное кодирование, то могут применяться и другие механизмы мутации, о которых мы будем говорить в главе 5.

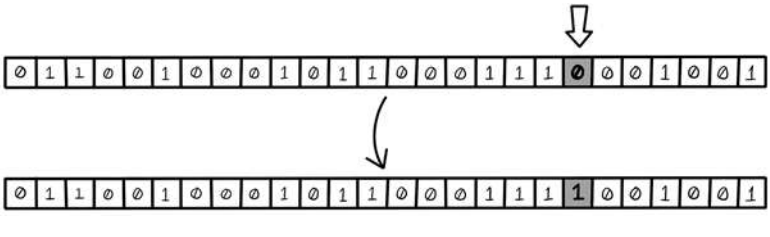


Рис. 4.24. Мутация битовой строки

Псевдокод

Для мутации одного гена хромосомы особи выбирается ген в случайном индексе. Если он представляет 1, то его значение изменяется на 0, и наоборот:

```
mutate_individual (individual, chromosome_length)
  let random_index equal a random number between 0 and chromosome_length
  if gene at index random_index of individual is equal to 1:
    let gene at index random_index of individual equal 0
  else:
    let gene at index random_index of individual equal 1
  return individual
```

Мутация инверсией битов для двоичного кодирования

В этом виде мутации все гены двоично-закодированной хромосомы инвертируются на противоположные значения. Все 1 заменяются на 0, и наоборот. Такая мутация может сильно ухудшить качество хороших решений и обычно используется в случаях, когда требуется одномоментно внести разнообразие в популяцию (рис. 4.25).

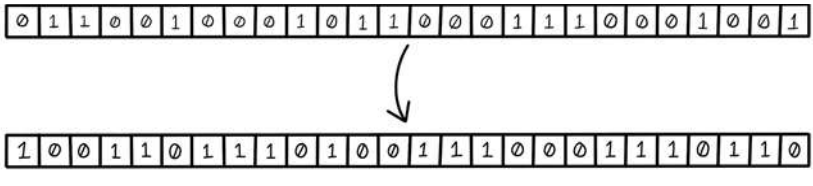


Рис. 4.25. Мутация инверсией битов

Заполнение следующего поколения

После оценки приспособленности особей популяции и воспроизведения потомства наступает этап отбора особей, которые доживут до следующего поколения. Размер популяции обычно фиксирован, а поскольку в процессе размножения появились новые особи, некоторые представители популяции должны умереть.

На первый взгляд кажется разумным взять превосходящих особей согласно требуемому размеру популяции и ликвидировать остальных. Тем не менее такой подход может привести к недостаточному разнообразию особей, если выжившие представители окажутся схожими по генетической структуре (рис. 4.26).

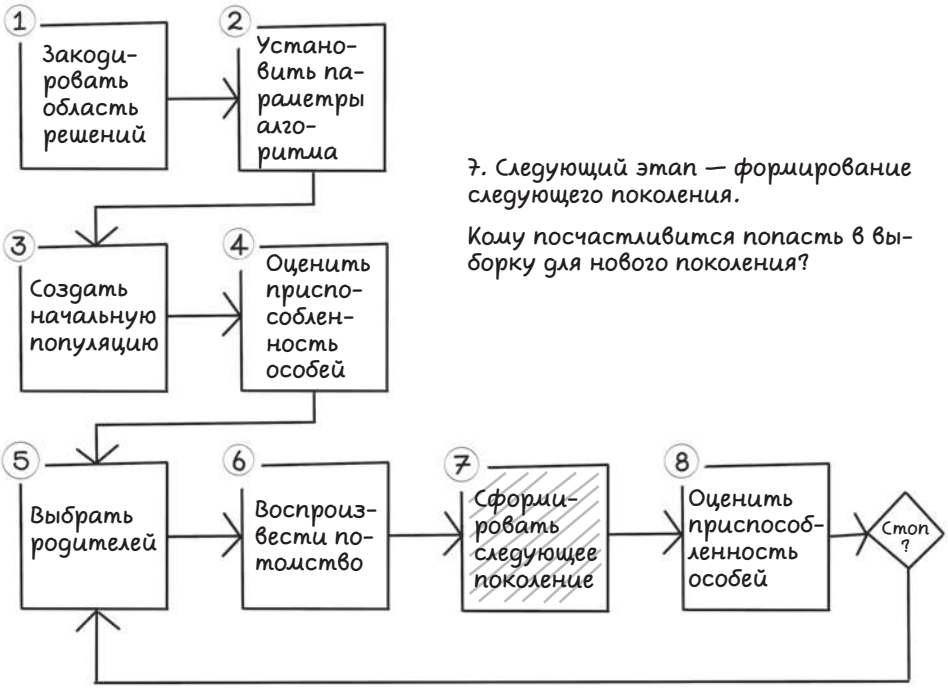


Рис. 4.26. Заполнение следующего поколения

Рассматриваемые в этом разделе стратегии отбора можно использовать для определения особей, из которых будет формироваться популяция следующего поколения.

Разведка vs эксплуатация

Выполнение генетического алгоритма всегда подразумевает поиск баланса между разведкой и эксплуатацией. В идеальной ситуации среди особей существует разнообразие, и популяция в целом ищет в пространстве поиска совершенно разные решения. Затем пространства с наиболее подходящими локальными решениями эксплуатируются для нахождения самого подходящего. Преимущество такой схемы в том, что алгоритм по максимуму исследует пространство поиска, эксплуатируя при этом сильные решения по мере развития особей (рис. 4.27).

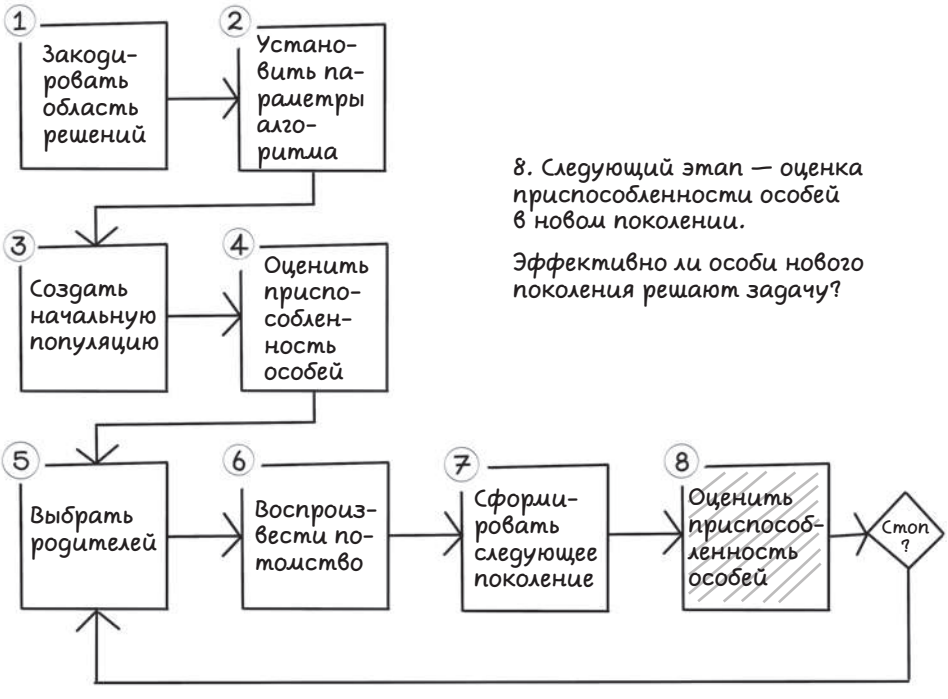


Рис. 4.27. Оценка приспособленности особей

Условия остановки

Так как генетический алгоритм ищет лучшие решения с каждым поколением итеративно, необходимо обозначать условие его остановки. В противном случае он будет выполняться бесконечно. *Условие остановки* — это условие, при выполнении которого алгоритм завершает работу, а в качестве лучшего решения выбирается сильнейшая особь в текущем поколении. Простейшим условием остановки является *константа* — постоянное значение, указывающее количество поколений для выполнения алгоритма. В качестве альтернативы можно останавливать алгоритм по достижении определенного показателя приспособленности. Этот метод используется, когда известна минимальная требуемая степень приспособленности, но неизвестно решение.

Стагнация — это проблема в эволюционных алгоритмах, при которой популяция на протяжении нескольких поколений получает схожие по силе решения. Если популяция впадает в стагнацию, вероятность генерации сильных решений в последующих поколениях сильно снижается. В данном случае условие остановки может проверять изменение приспособленности лучшей особи каждого поколения и, если эти изменения становятся незначительными, останавливать выполнение алгоритма.

Псевдокод

В основной функции, полностью описывающей жизненный цикл, используются различные шаги генетического алгоритма. К переменным параметрам относится размер популяции, количество поколений для выполнения алгоритма и вместимость рюкзака для функции приспособленности, а также переменная позиция кроссинговера и частота мутации для этапов кроссинговера и мутации:

```
run_ga (population_size, number_of_generations, knapsack_capacity):
  let best_global_fitness equal 0
  let global_population equal...
  ...generate_initial_population(population_size)
  for generation in range(number_of_generations):
    let current_best_fitness equal...
    ...calculate_population_fitness(global_population, knapsack_capacity)
    if current_best_fitness is greater than best_global_fitness:
      let best_global_fitness equal current_best_fitness
    let the_chosen equal...
    ...roulette_wheel_selection(global_population, population_size)
    let the_children equal...
    ...reproduce_children(the_chosen)
    let the_children equal...
    ...mutate_children(the_children)
    let global_population equal...
    ...merge_population_and_children(global_population, the_children)
```

Как говорилось в начале главы, задачу о рюкзаке можно решить с помощью метода прямого перебора (брутфорса), при котором потребуется сгенерировать и проанализировать 60 миллионов комбинаций. При сравнении генетических алгоритмов, решающих одну и ту же задачу, эффективность намного более высока, если параметры разведки и эксплуатации настроены верно. Помните, что в некоторых случаях генетический алгоритм выводит «достаточно хорошее» решение, которое не обязательно является лучшим возможным. Опять же, рациональность использования генетического алгоритма зависит от контекста задачи (рис. 4.28).

	Брутфорс	Генетический алгоритм
Кол-во итераций	2^26 = 67 108 864	10 000 - 100 000
Точность	100%	100%
Время вычисления	~7 минут	~3 секунды
Лучший результат	13 692 887	13 692 887

Рис. 4.28. Эффективность брутфорса и генетического алгоритма

Конфигурация параметров генетического алгоритма

При проектировании и конфигурации генетического алгоритма нужно принять несколько решений, которые повлияют на его итоговую эффективность. Общая эффективность складывается из двух составляющих: алгоритм должен успешно справляться с поиском хороших решений задачи и демонстрировать хорошую вычислительную производительность. Бессмысленно разрабатывать алгоритм, если решение задачи с его помощью потребует бóльших вычислительных затрат, чем традиционные техники. Используемый тип кодирования, выбранная функция приспособленности, а также другие алгоритмические параметры влияют на обе составляющих эффективности, определяя и качество найденного решения, и вычислительную мощность. Вот ряд параметров, которые потребуется учесть:

- *Кодирование хромосом*. Метод кодирования хромосом должен соответствовать задаче, а возможные решения — стремиться к глобальному максимуму. Схема кодирования — основа успешной работы алгоритма.
- *Размер популяции*. Это настраиваемый параметр. Чем крупнее популяция, тем больше разнообразие в возможных решениях, но при этом возрастает и вычислительная нагрузка. Иногда увеличение популяции снимает необходимость в мутациях, то есть большое разнообразие создается изначально, а не с течением поколений. Допустимо также начать с небольшой популяции и увеличивать ее, исходя из производительности.
- *Инициализация популяции*. Несмотря на то что особи инициализируются случайно, важно следить за тем, чтобы их решения были действительными, так как это оптимизирует вычислительные затраты на выполнение алгоритма и инициализацию особей с соблюдением ограничений.
- *Количество потомков*. Количество создаваемых в каждом поколении потомков — настраиваемый параметр. Учитывая, что после размножения часть популяции ликвидируется для поддержания постоянного числа особей, большее количество потомков будет означать большее разнообразие, но при этом увеличивается риск, что эти потомки заменят собой хорошие решения. Если популяция динамична, то ее размер может изменяться после каждого поколения, но в таком случае потребуется конфигурировать и контролировать большее число параметров.
- *Метод отбора родителей*. Этот метод должен основываться на задаче и желаемом соотношении разведки и эксплуатации.
- *Метод кроссинговера*. Этот метод ассоциируется с используемым методом кодирования, но также может настраиваться, повышая или понижая разнообразие популяции. Потомки по-прежнему должны воспроизводить действительные решения.
- *Частота мутаций*. Частота мутаций относится к параметрам, увеличивающим разнообразие особей и возможных решений. Повышенная частота мутаций увеличивает разнообразие, но может ухудшить свойства особей.

Этот параметр можно с течением времени изменять, повышая разнообразие в ранних поколениях и понижая его в более поздних. Это можно охарактеризовать как разведку в начале, сопровождаемую последующей эксплуатацией.

- *Метод мутации.* Метод мутации аналогичен методу кроссинговера в том, что зависит от выбранного метода кодирования. Его основная задача — продолжать воспроизводить действительное решение после модификации или получать нулевое значение приспособленности.
- *Метод заполнения поколения.* Аналогично методу отбора родителей заполнение поколения подразумевает отбор особей, которые доживут до следующего поколения. В зависимости от выбранного метода алгоритм может либо слишком быстро конвергировать и впадать в стагнацию, либо слишком долго заниматься разведкой.
- *Условие остановки.* Условие остановки алгоритма должно определяться на основе задачи и желаемого результата. Основными метриками для его определения выступают вычислительная сложность и время выполнения.

Применение эволюционных алгоритмов

Эволюционные алгоритмы находят обширное применение. Некоторые решают отдельно взятые задачи, некоторые используются совместно с другими техниками для решения комплексных задач, например:

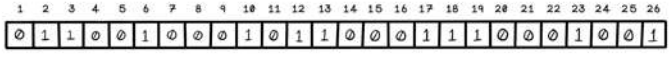
- *Прогнозирование поведения биржевых игроков.* Инвесторы ежедневно принимают решения о том, стоит ли покупать дополнительные активы, сохранять имеющиеся или же, наоборот, продавать их. Последовательность этих действий во времени можно развернуть и сопоставить с их влиянием на портфель инвестора. Далее на основе этой информации финансовые институты формируют полезные рекомендации и предлагают различные услуги.
- *Отбор признаков в машинном обучении.* Машинное обучение будет рассматриваться в главе 8, но его ключевой аспект заключается в следующем: при наличии некоторого количества признаков объекта определить, чем этот объект является. Если рассматривать дома, можно найти много связанных с ними свойств, например время постройки, материал, площадь, цвет и местоположение. Но для прогнозирования рыночной стоимости важными могут оказаться, пожалуй, только время постройки, площадь и местоположение. Генетический алгоритм может находить неявные признаки, имеющие первостепенное значение.
- *Взлом кодов и шифров.* Шифр — это сообщение, закодированное так, чтобы выглядеть как нечто другое. Обычно шифры используются для скрытия информации. Если получатель не знает, как расшифровать сообщение, то и понять его не сможет. Эволюционные алгоритмы могут генерировать множество вероятностей для изменения зашифрованного сообщения, выявляя таким образом исходное.

В главе 5 мы подробнее рассмотрим продвинутые концепции генетических алгоритмов, которые адаптируют эти алгоритмы к различным областям задач. Мы изучим различные техники кодирования, кроссинговера, мутации и отбора, а также познакомимся с их альтернативами.

Краткий обзор главы «Эволюционные алгоритмы»

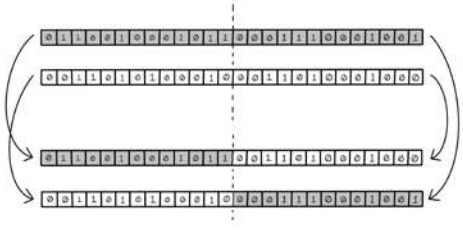
Для быстрого эффективного поиска эволюционные алгоритмы используют интеллектуальный случайный выбор.

В этом алгоритме кодирование имеет ключевое значение.

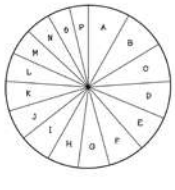


В основе нахождения лучших решений лежит фитнес-функция, или функция приспособленности.

Кроссинговер служит для воспроизведения лучших решений в каждом поколении.



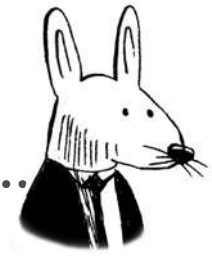
В ходе отбора преимущество получают более сильные особи, но слабые сохраняются, чтобы, возможно, воспроизвести лучшие решения в будущем.



Отбор методом рулетки

В начале — разведка, затем — эксплуатация.

5 Продвинутые эволюционные подходы



В этой главе

- ✓ Вариации этапов жизненного цикла генетического алгоритма
- ✓ Изменение генетического алгоритма для решения различных задач
- ✓ Продвинутые параметры конфигурации жизненного цикла алгоритма на основе разных сценариев, задач и наборов данных

Примечание

Текущая глава является продолжением предыдущей.

Жизненный цикл эволюционного алгоритма

Обобщенный жизненный цикл генетического алгоритма описан в главе 4. В текущей же главе рассматриваются другие задачи, которые также можно решать с помощью этого алгоритма, объясняется, почему некоторые из рассмотренных ранее подходов могут не работать, а также приводятся альтернативные варианты.

Напомним, что жизненный цикл генетического алгоритма выглядит так:

- *Создание популяции*. Создание случайной популяции возможных решений.
- *Оценка приспособленности особей популяции*. Определение, насколько хорошим является конкретное решение. Для этого используется функция приспособленности, вычисляющая показатель его эффективности.
- *Отбор родителей на основе их приспособленности*. Отбор пар родителей для воспроизведения потомства.
- *Воспроизведение особей родителями*. Создание родителями потомства путем смешивания генетической информации и применения к потомству легкой мутации.
- *Заполнение следующего поколения*. Отбор из популяции особей и потомков, которые доживут до следующего поколения.

По мере чтения главы обращайтесь к приведенной на рис. 5.1 схеме жизненного цикла алгоритма.

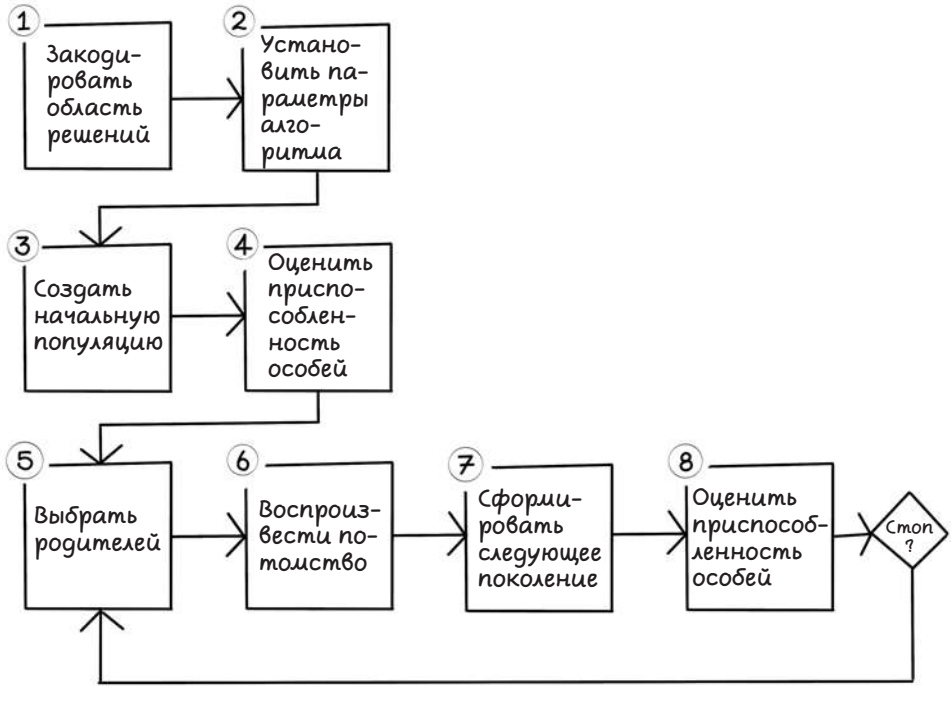


Рис. 5.1. Жизненный цикл генетического алгоритма

Начнем с изучения альтернативных стратегий отбора. Эти отдельные подходы можно подставлять в любой генетический алгоритм или извлекать из него. Далее описаны три измененных сценария задачи о рюкзаке (из главы 4), в которых демонстрируется использование альтернативных подходов кодирования, кроссинговера и мутации (рис. 5.2).

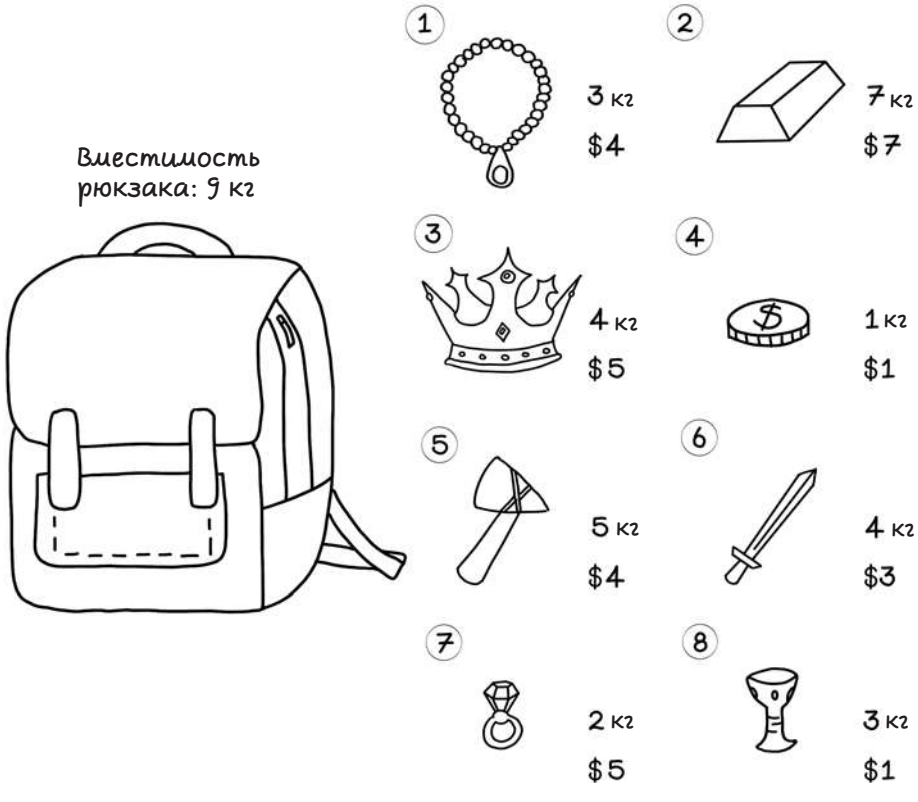


Рис. 5.2. Пример задачи о рюкзаке

Альтернативные стратегии отбора

В главе 4 мы изучили отбор с помощью колеса рулетки — это один из самых простых методов отбора особей. Описываемые далее три альтернативные стратегии позволяют свести к минимуму связанные с таким подходом проблемы. Каждая стратегия имеет как достоинства, так и недостатки, влияющие на разнообразие популяции, которое в конечном итоге определяет, будет ли найдено оптимальное решение.

Ранговый отбор: выравнивание игрового поля

Одна из проблем отбора колесом рулетки состоит в большом разбросе приспособленности хромосом. Это либо сильно смещает отбор в сторону особей с высокими значениями приспособленности, либо, наоборот, увеличивает шанс выбора слабых особей больше, чем хотелось бы. И тот и другой случай негативно сказываются на разнообразии популяции. Большее разнообразие дает больший охват пространства поиска, но это может растянуть поиск оптимальных решений на слишком большое количество поколений.

Ранговый отбор стремится справиться с этой проблемой, ранжируя особей на основе их приспособленности. Присвоенные значения затем используются для вычисления размера секции колеса рулетки. В задаче о рюкзаке это значение представлено числом от 1 до 16, так как выбор происходит между 16 особями. Несмотря на то что вероятность выбора сильных особей выше, чем вероятность выбора слабых, эти шансы усредняются, и каждая особь получает более справедливый шанс быть выбранной на основе ранга, чем на основе точной приспособленности. После ранжирования 16 особей колесо выглядит уже несколько иначе (рис. 5.3).

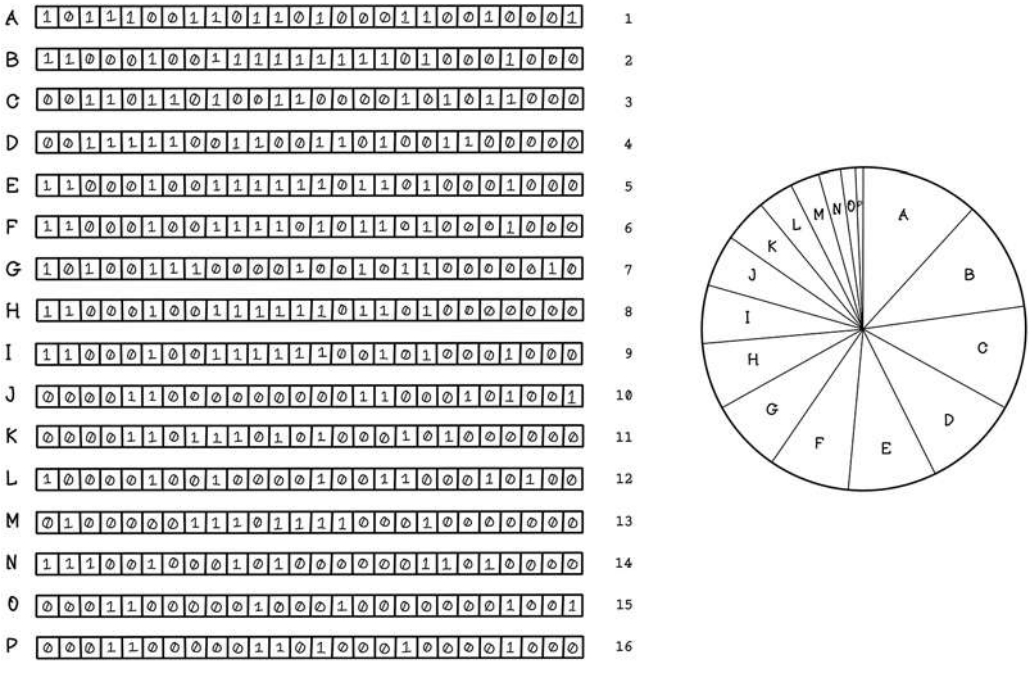


Рис. 5.3. Пример рангового отбора

На рис. 5.4 представлено сравнение отбора на основе рулетки и рангового отбора. Очевидно, что ранговый вариант повышает шанс выбора лучших решений.

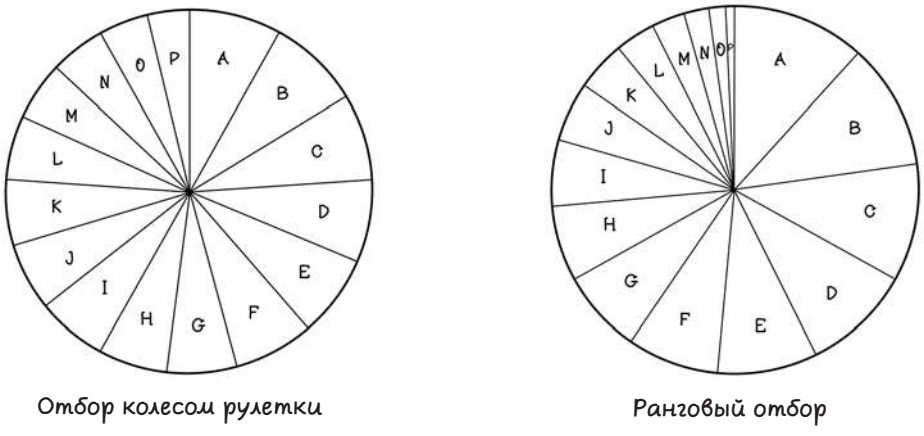


Рис. 5.4. Отбор колесом рулетки и ранговый отбор

Турнирный отбор: победит сильнейший!

В турнирном отборе хромосомы выступают друг против друга. Случайным образом выбирается определенное количество особей, которые помещаются в группу. Таким образом формируется заданное число групп, после чего в каждой группе выбирается особь с наибольшим значением приспособленности. Чем больше группа, тем меньше конечное разнообразие, так как из каждой группы отбирается только одна особь. Как и в случае с ранговым отбором, фактическое значение приспособленности каждой особи не является основным фактором в глобальном выборе особей.

После распределения 16 особей по четырем группам из каждой группы отбирается по одному сильнейшему представителю. Далее эти четыре победителя составляют пары для воспроизведения потомства (рис. 5.5).

Элитный отбор: выбираются только лучшие

Элитизм подразумевает выбор лучших особей популяции. Такой способ полезен для сохранения сильных представителей и избежания риска их утраты при

использовании других методов отбора. Недостаток метода в том, что популяция может оказаться в области локального лучшего решения, утратив достаточное разнообразие для нахождения глобальных лучших.

Элитизм часто используется вместе с рулеточной селекцией, ранговой селекцией и турнирной. Идея состоит в выборе нескольких лучших особей для воспроизведения и заполнении остальной части популяции с помощью другой стратегии отбора (рис. 5.6).

В главе 4 рассматривалась задача, предполагающая двоичную кодировку, то есть каждый предмет либо добавляли в рюкзак, либо нет. Тем не менее разные области задач требуют разных видов кодирования, поскольку двоичный подход может оказаться бесполезным. В трех последующих разделах описываются альтернативные сценарии.

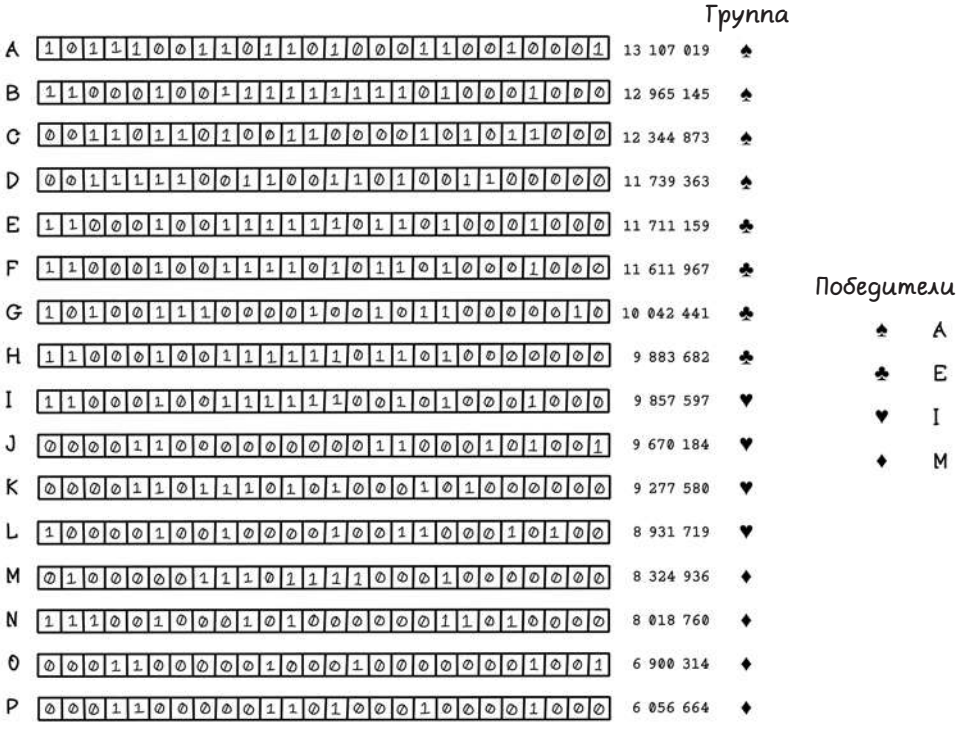


Рис. 5.5. Пример турнирного отбора

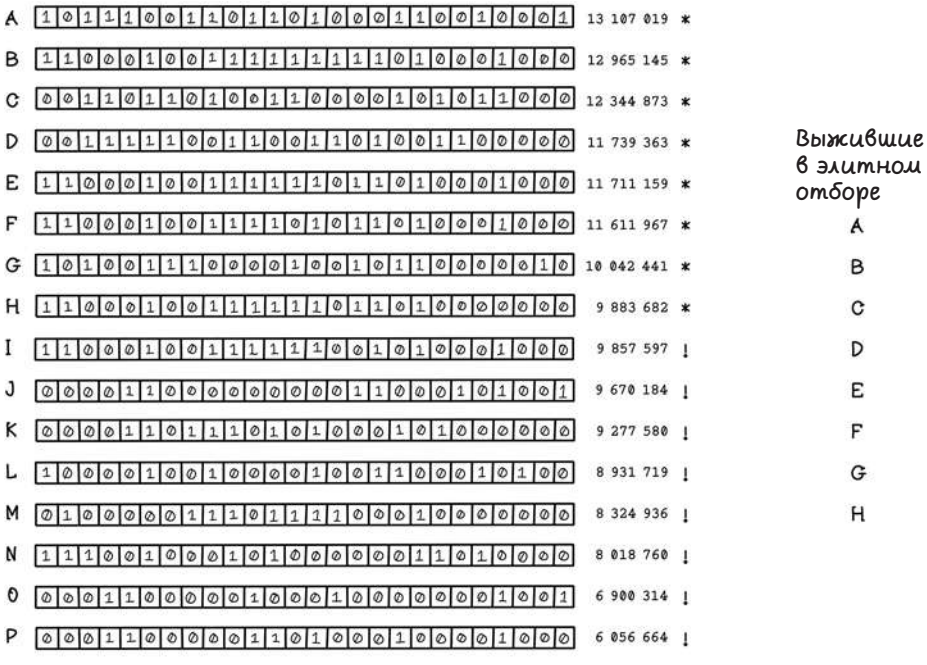


Рис. 5.6. Пример элитного отбора

Вещественное кодирование: работа с действительными числами

Рассмотрим несколько измененную вариацию задачи о рюкзаке. В ней по-прежнему нужно выбирать наиболее ценные предметы для заполнения рюкзака в соответствии с его максимальным весом, но теперь каждый предмет можно добавлять неоднократно. Как показано в табл. 5.1, вес и ценность остались неизменными, но добавилась графа «количество». После такой незначительной, казалось бы, доработки появилось огромное множество дополнительных вариантов решений, один из которых или более могут оказаться предпочтительными. В таком сценарии двоичное кодирование уже не подойдет. Вместо него для представления состояния потенциальных решений разумнее использовать вещественное кодирование.

Таблица 5.1. Вместимость рюкзака: 6 404 180 кг

ID предмета	Название предмета	Вес (кг)	Ценность ($)	Количество
1	Топор	32 252	68 674	19
2	Бронзовая монета	225 790	471 010	14
3	Корона	468 164	944 620	2
4	Алмазная статуя	489 494	962 094	9
5	Изумрудный пояс	35 384	78 344	11
6	Кусок руды	265 590	579 152	6
7	Золотая монета	497 911	902 698	4
8	Шлем	800 493	1 686 515	10
9	Чернила	823 576	1 688 691	7
10	Шкатулка	552 202	1 056 157	3
11	Нож	323 618	677 562	5
12	Длинный меч	382 846	833 132	13
13	Маска	44 676	99 192	15
14	Ожерелье	169 738	376 418	8
15	Опаловая брошь	610 876	1 253 986	4
16	Жемчуг	854 190	1 853 562	9
17	Колчан	671 123	1 320 297	12
18	Рубиновое кольцо	698 180	1 301 637	17
19	Серебряный браслет	446 517	859 835	16
20	Хронометр	909 620	1 677 534	7
21	Униформа	904 818	1 910 501	6
22	Ядовитое зелье	730 061	1 528 646	9
23	Шерстяной шарф	931 932	1 827 477	3
24	Арбалет	952 360	2 068 204	1
25	Прошлогодняя книга	926 023	1 746 556	7
26	Чаша из цинка	978 724	2 100 851	2

Суть вещественного кодирования

Вещественное кодирование представляет ген в виде численных значений, строк или символов и выражает возможные решения в естественном, соответствую-

щем задаче состоянии. Этот вид кодирования используется, когда возможные решения содержат непрерывные значения, которые трудно выразить в двоичном виде. Например, из-за того что теперь в рюкзак можно класть несколько одинаковых предметов, индекс каждого из них не может просто указывать, добавлен этот предмет или нет, и должен обозначать количество соответствующих предметов в рюкзаке (рис. 5.7).

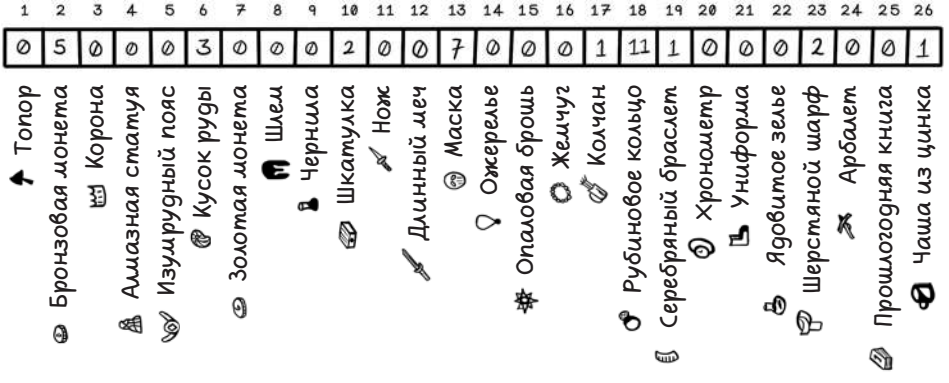


Рис. 5.7. Пример вещественного кодирования

Вследствие изменения схемы кодирования становятся доступны новые варианты кроссинговера и мутации. Что касается кроссинговера, то применимы те же его приемы, что и в двоичном кодировании, а вот способ мутации потребуется пересмотреть.

Арифметический кроссинговер: воспроизведение по математическим принципам

Арифметический кроссинговер подразумевает совершение арифметической операции, в которой каждый родитель представляется в виде переменной в выражении. Результатом такой операции с участием двух родителей является потомок. При использовании этой стратегии с двоичным кодированием важно убедиться, что в результате получилась подходящая хромосома. Арифметический кроссинговер можно применять и к двоичному, и к вещественному кодированию (рис. 5.8).

> ***Примечание***
>
> Будьте внимательны, так как потомство при этом подходе очень разнообразно, что может вызвать сложности.

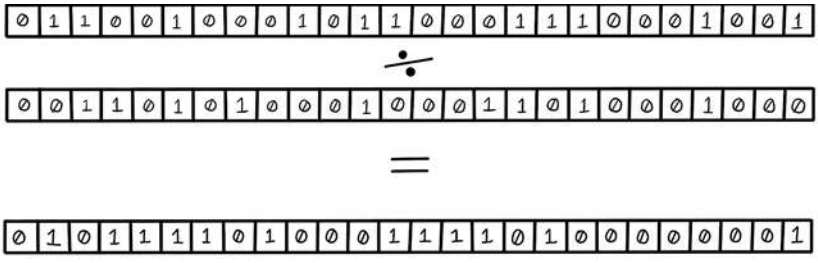


Рис. 5.8. Пример арифметического кроссинговера

Граничная мутация

В граничной мутации случайный ген вещественно закодированной хромосомы случайным образом изменяется на верхнее или нижнее действительное значение. При наличии 26 генов в хромосоме случайным образом выбирается индекс, и его значение устанавливается либо как максимальное, либо как минимальное возможное для этого предмета. Минимальные и максимальные значения могут быть одинаковыми для всех генов либо устанавливаются для каждого гена отдельно, если тому способствует знание пространства задачи. Этот подход оценивает влияние отдельных генов на хромосому.

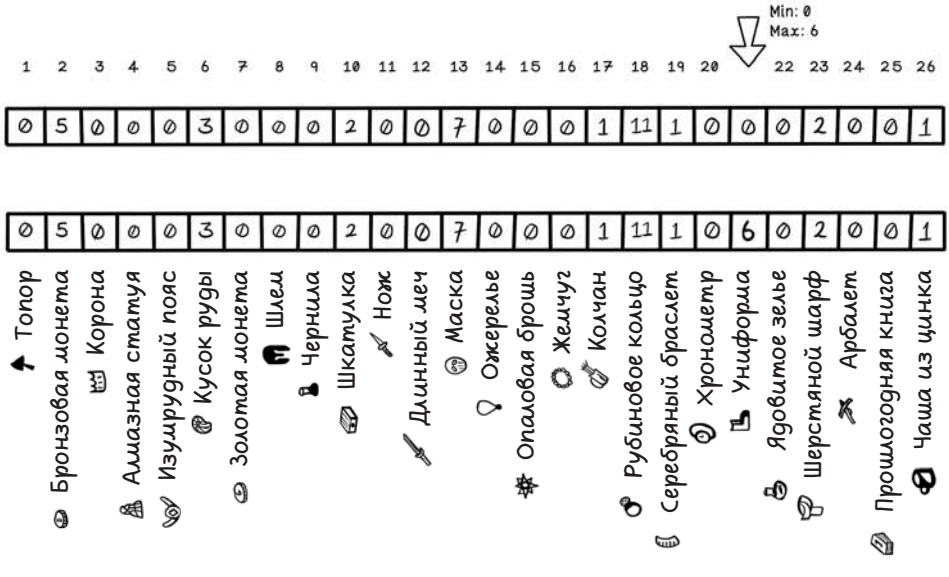


Рис. 5.9. Пример граничной мутации

Арифметическая мутация

В арифметической мутации случайно выбираемый ген в вещественно закодированной хромосоме изменяется путем прибавления или вычитания небольшого числа. Обратите внимание, что пример на рис. 5.10 включает целые числа, но эти числа могут быть и дробными.

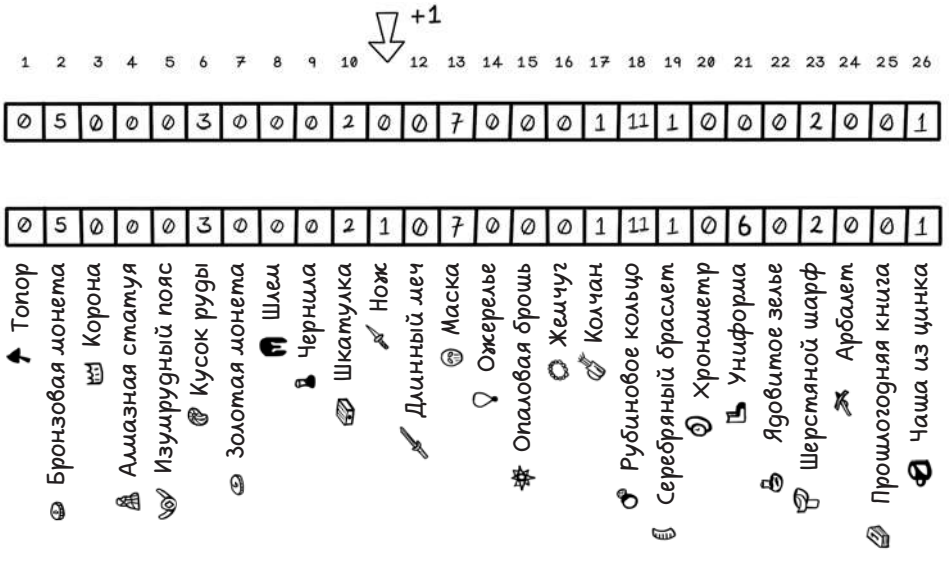


Рис. 5.10. Пример арифметической мутации

Порядковое кодирование: работа с последовательностями

Мы по-прежнему работаем с теми же предметами, что и в задаче о рюкзаке, только теперь нам не требуется определять, какие из них надо положить в рюкзак. Вместо этого все предметы (золотая монета, серебряный браслет и др.) должны пройти цикл переработки на фабрике, в ходе которого из них будет получено исходное сырье. В этом сценарии выбирать отдельные предметы не потребуется, так как в переработку пойдут все.

Помимо этого фабрика требует устойчивый показатель ценности извлекаемых материалов с учетом времени переработки каждого предмета и его ценности. Предполагается, что ценность переработанного сырья примерно равна ценности исходного предмета. Здесь мы сталкиваемся с задачей упорядочивания. В какой последовательности нужно перерабатывать предметы, чтобы поддерживать постоянный показатель ценности? В табл. 5.2 приводится тот же список предметов с указанием соответствующего времени их переработки.

Таблица 5.2. Ценность фабрики в час: 600 000

ID предмета	Название предмета	Вес (кг)	Ценность ($)	Время переработки
1	Топор	32 252	68 674	60
2	Бронзовая монета	225 790	471 010	30
3	Корона	468 164	944 620	45
4	Алмазная статуя	489 494	962 094	90
5	Изумрудный пояс	35 384	78 344	70
6	Кусок руды	265 590	579 152	20
7	Золотая монета	497 911	902 698	15
8	Шлем	800 493	1 686 515	20
9	Чернила	823 576	1 688 691	10
10	Шкатулка	552 202	1 056 157	40
11	Нож	323 618	677 562	15
12	Длинный меч	382 846	833 132	60
13	Маска	44 676	99 192	10
14	Ожерелье	169 738	376 418	20
15	Опаловая брошь	610 876	1 253 986	60
16	Жемчуг	854 190	1 853 562	25
17	Колчан	671 123	1 320 297	30
18	Рубиновое кольцо	698 180	1 301 637	70
19	Серебряный браслет	446 517	859 835	50
20	Хронометр	909 620	1 677 534	45
21	Униформа	904 818	1 910 501	5
22	Ядовитое зелье	730 061	1 528 646	5
23	Шерстяной шарф	931 932	1 827 477	5
24	Арбалет	952 360	2 068 204	25
25	Прошлогодняя книга	926 023	1 746 556	5
26	Чаша из цинка	978 724	2 100 851	10

Важность функции приспособленности

При преобразовании задачи о рюкзаке в задачу о переработке ключевое отличие состоит в оценке успешных решений. Так как фабрике необходим постоянный минимальный показатель ценности в час, точность функции приспособленности становится первостепенным критерием для нахождения оптимальных

решений. В задаче о рюкзаке пригодность решения вычисляется просто, так как подразумевает всего два действия: обеспечение соответствия максимальному весу рюкзака и суммирование ценности выбранных предметов. В задаче о переработке эта функция должна вычислять показатель получаемой ценности в час, используя время переработки предметов и их ценность. Эти вычисления уже более сложны, и ошибка в логике функции будет непосредственно влиять на качество получаемых решений.

Суть порядкового кодирования

Порядковое кодирование, также известное как пермутационное, представляет хромосому как последовательность элементов. Этот вид кодирования обычно требует наличия в хромосоме всех элементов, что подразумевает вероятность корректировки при выполнении кроссинговера и мутации с целью исключить утрату и дублирование элементов. На рис. 5.11 показано, как хромосома представляет порядок обработки доступных предметов.

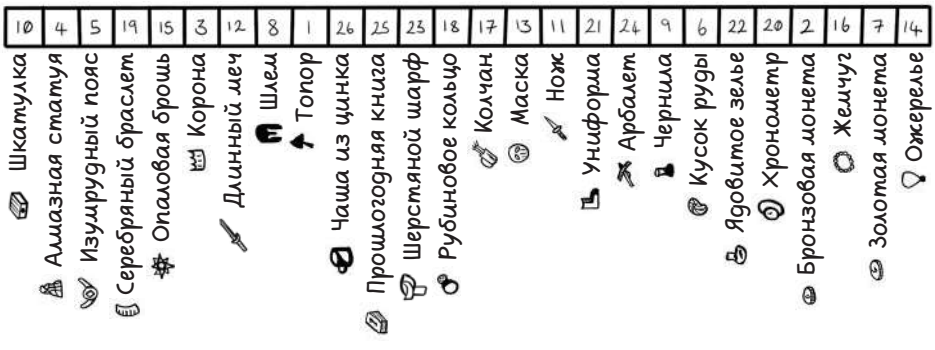


Рис. 5.11. Пример порядкового кодирования

Еще один пример, в котором разумно применить порядковое кодирование, — это представление потенциальных решений в задачах оптимизации маршрута. При наличии заданного количества пунктов назначения, каждый из которых нужно посетить не менее одного раза, стремясь при этом минимизировать пройденное расстояние, маршрут можно представить в виде строки, которая включает эти пункты назначения, расположенные в порядке их посещения. Этот пример мы используем, когда будем говорить о роевом интеллекте в главе 6.

Порядковая мутация: порядковое/пермутационное кодирование

В порядковой мутации два случайно выбранных гена в хромосоме с порядковой кодировкой меняются местами, обеспечивая разнообразие с сохранением всех предметов в хромосоме (рис. 5.12).

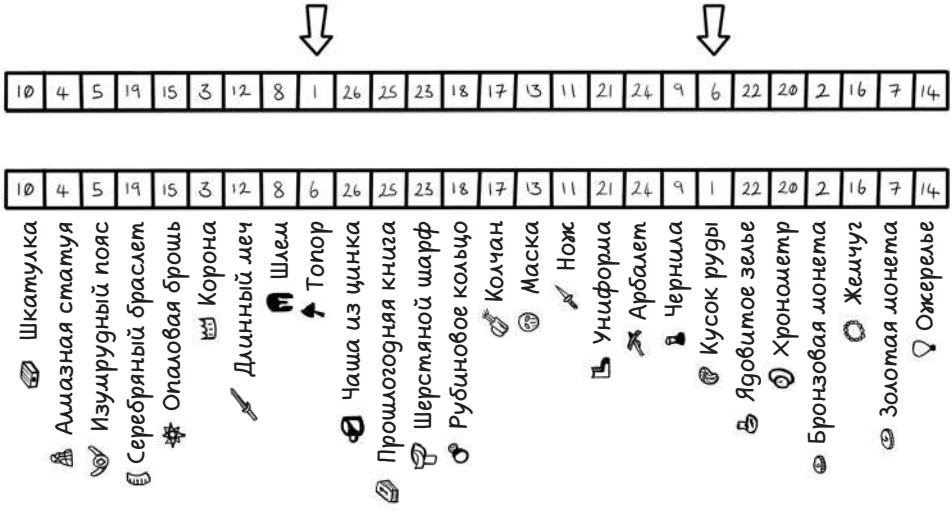


Рис. 5.12. Пример порядковой мутации

Древовидное кодирование: работа с иерархиями

Предыдущие разделы показывают, что двоичное кодирование подходит при выборе предметов из набора, вещественное — в случаях, когда для решения важны действительные числа, а порядковое эффективно для определения приоритета и последовательности. Предположим, что предметы из задачи о рюкзаке упакованы для доставки в дома, расположенные в разных точках города. Каждый фургон может вместить определенный объем груза. Задача — распределить упаковки так, чтобы пустое пространство в каждом фургоне было минимальным (табл. 5.3).

Таблица 5.3. Габариты фургона: 1000 ширина × 1000 высота

ID предмета	Название предмета	Вес (кг)	Ценность ($)	Ширина	Высота
1	Топор	32 252	68 674	20	60
2	Бронзовая монета	225 790	471 010	10	10
3	Корона	468 164	944 620	20	20
4	Алмазная статуя	489 494	962 094	30	70
5	Изумрудный пояс	35 384	78 344	30	20

ID предмета	Название предмета	Вес (кг)	Ценность ($)	Ширина	Высота
6	Кусок руды	265 590	579 152	15	15
7	Золотая монета	497 911	902 698	10	10
8	Шлем	800 493	1 686 515	40	50
9	Чернила	823 576	1 688 691	5	10
10	Шкатулка	552 202	1 056 157	40	30
11	Нож	323 618	677 562	10	30
12	Длинный меч	382 846	833 132	15	50
13	Маска	44 676	99 192	20	30
14	Ожерелье	169 738	376 418	15	20
15	Опаловая брошь	610 876	1 253 986	5	5
16	Жемчуг	854 190	1 853 562	10	5
17	Колчан	671 123	1 320 297	30	70
18	Рубиновое кольцо	698 180	1 301 637	5	10
19	Серебряный браслет	446 517	859 835	10	20
20	Хронометр	909 620	1 677 534	15	20
21	Униформа	904 818	1 910 501	30	40
22	Ядовитое зелье	730 061	1 528 646	15	15
23	Шерстяной шарф	931 932	1 827 477	20	30
24	Арбалет	952 360	2 068 204	50	70
25	Прошлогодняя книга	926 023	1 746 556	25	30
26	Чаша из цинка	978 724	2 100 851	15	25

Для простоты предположим, что объем фургонов представлен двумерным прямоугольником и упаковки также прямоугольные, а не трехмерные.

Суть древовидного кодирования

При кодировании в виде дерева хромосома представляется как дерево элементов. Это весьма гибкий вид кодирования для представления возмож-

ных решений, в котором важной или необходимой составляющей является иерархия элементов. С помощью древовидного кодирования можно также представлять функции, состоящие из дерева выражений. В итоге такое кодирование можно использовать для поэтапного развития программных функций, решающих конкретную задачу. При этом решение может работать, но выглядеть странным.

Ниже приведен пример использования древовидного кодирования. Согласно условию у нас есть фургон заданной высоты и ширины, в который нужно поместить определенное количество упаковок. Цель — заполнить фургон так, чтобы осталось как можно меньше пустого пространства. Возможные решения этой задачи можно представить в виде дерева.

На рис. 5.13 корневой узел A представляет заполнение фургона сверху вниз. Узел B, равно как узлы C и D, представляет все упаковки горизонтально. Узел E представляет упаковки, сложенные вертикально в своей части фургона.

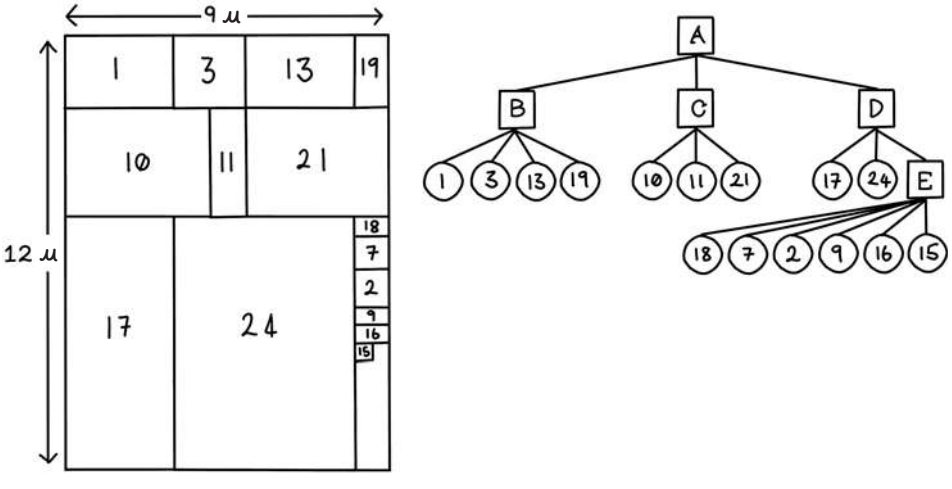


Рис. 5.13. Пример использования дерева для представления задачи загрузки фургона

Кроссинговер дерева: наследование частей дерева

Кроссинговер дерева подобен одноточечному кроссинговеру (см. главу 4) тем, что в структуре дерева также выбирается одно ребро и делится пополам, после чего из двух получившихся частей формируется потомок. Для создания второго потомка этот процесс объединения частей родителей инвертируется. Далее потомки должны пройти проверку на соответствие их решений ограничениям задачи. При этом в процессе кроссинговера может использоваться больше ребер, если это помогает найти лучшее решение (рис. 5.14).

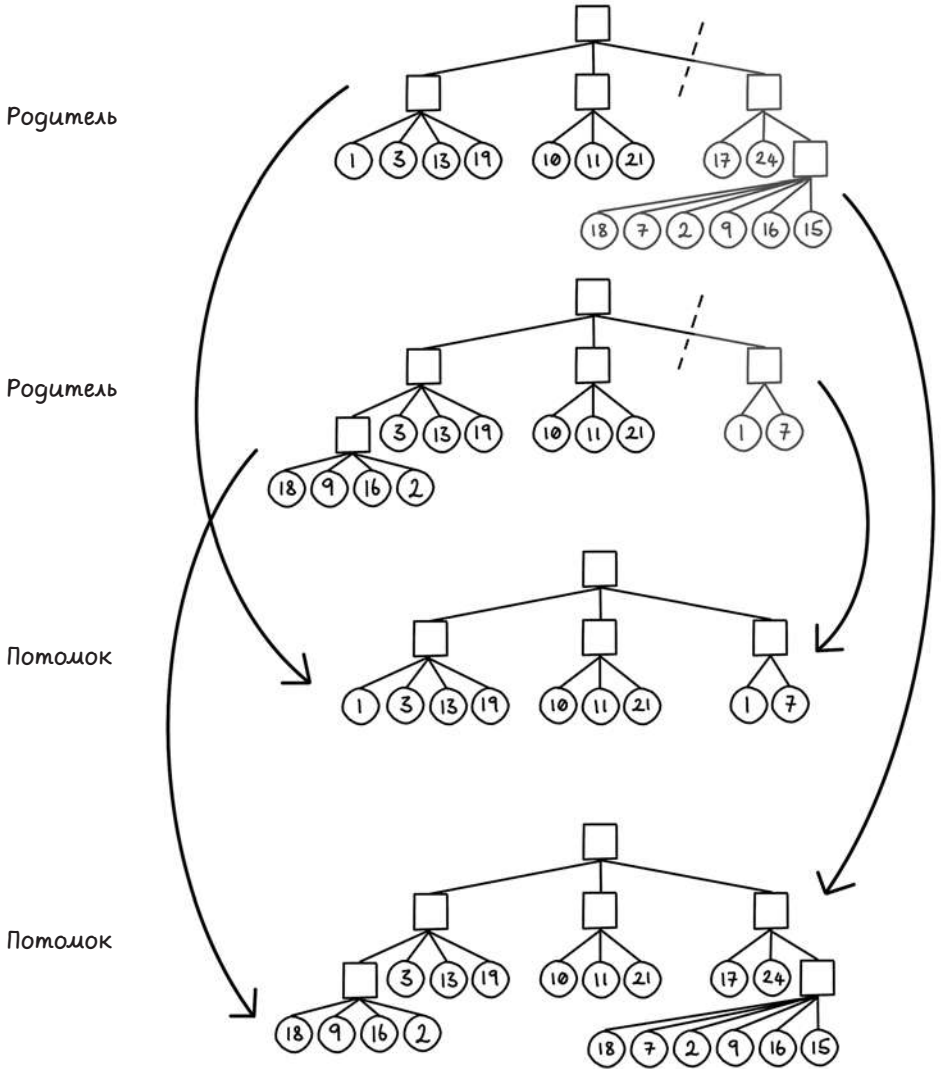


Рис. 5.14. Пример кроссинговера дерева

Мутация узла: изменение значения узла

При мутации узла значение случайно выбранного узла хромосомы в древовидной кодировке заменяется на случайно выбранный допустимый для этого узла объект. При наличии дерева, представляющего расположение предметов, можно заменить один из них на другой подходящий (рис. 5.15).

В этой и предыдущей главах рассматриваются несколько схем кодирования и кроссинговера, а также стратегии отбора. При реализации собственных генетических алгоритмов вы можете подставлять вместо них свои подходы, если это поможет найти решение задачи.

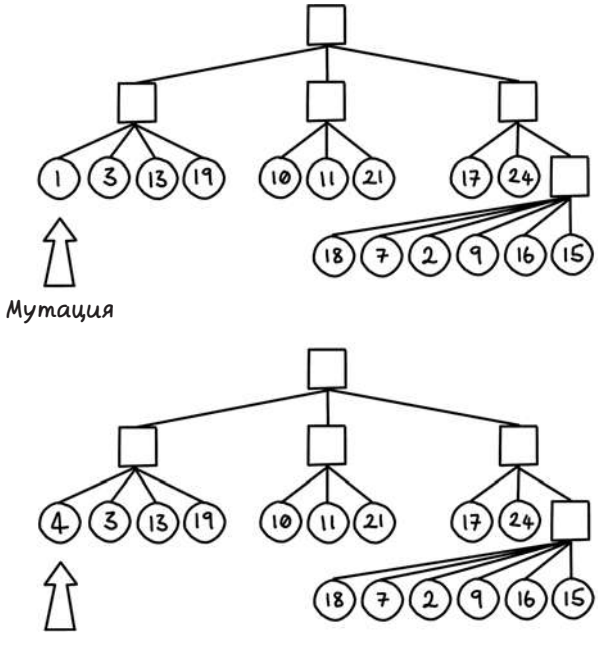


Рис. 5.15. Мутация узла в дереве

Стандартные типы эволюционных алгоритмов

В этой главе основное внимание уделяется жизненному циклу генетического алгоритма и альтернативным подходам к его реализации. Разные версии этого алгоритма подходят для решения разных задач. Теперь, когда мы разобрались, как этот алгоритм работает, пришло время рассмотреть его вариации и соответствующие случаи использования.

Генетическое программирование

Генетическое программирование подобно генетическим алгоритмам, но используется в основном для решения задач путем генерации компьютерных программ. В данном случае полностью применим описанный в предыдущем разделе процесс. Приспособленность возможных решений в алгоритме генетического программирования выражается в том, насколько хорошо созданная программа решает вычислительную задачу. С учетом этого мы видим, что метод древовидного кодирования здесь отлично работает, так как большинство компьютерных программ являются графами, состоящими из узлов, которые обозначают операции и процессы. Эти деревья логики могут развиваться, а вместе с ними развивается и компьютерная программа, решая поставленную задачу. Стоит отметить, что эти программы обычно развиваются в сложный для человеческого понимания и отладки код.

Эволюционное программирование

Эволюционное программирование похоже на генетическое, но в нем возможное решение представляет собой параметры для предварительно заданной, а не сгенерированной компьютерной программы. Если программе требуются точные исходные данные, а определение их удачной комбинации вызывает сложности, можно использовать для их получения генетический алгоритм. Приспособленность потенциальных решений в алгоритме эволюционного программирования определяется по тому, насколько хорошо фиксированная компьютерная программа выполняется на основе закодированных в особи параметров. С помощью эволюционного программирования также можно находить хорошие параметры для искусственных нейронных сетей, с которыми мы познакомимся в главе 9.

Глоссарий терминов теории эволюционных алгоритмов

Ниже приведен глоссарий терминов теории эволюционных алгоритмов, который пригодится для дальнейшего исследования и изучения материала:

- *Аллель* — значение конкретного гена в хромосоме.
- *Хромосома* — коллекция генов, представляющая возможное решение.
- *Особь* — одна хромосома в популяции.
- *Популяция* — коллекция особей.
- *Генотип* — искусственное представление популяции возможных решений в пространстве вычисления.
- *Фенотип* — фактическое представление популяции возможных решений в реальном мире.

- *Поколение* — одна итерация алгоритма.
- *Разведка (исследование)* — процесс поиска разнообразия возможных решений, из которых часть может оказаться хорошими, а часть — плохими.
- *Эксплуатация* — процесс доработки хороших решений и их многократного уточнения.
- *Функция приспособленности* — конкретный тип целевой функции.
- *Целевая функция* — функция, стремящаяся к максимизации или минимизации.

Другие случаи использования эволюционных алгоритмов

Некоторые случаи использования эволюционных алгоритмов были упомянуты ранее в главе 4, но на практике их намного больше. Примеры ниже особенно интересны, потому что в них описано применение одной или нескольких рассмотренных в этой главе концепций:

- *Подстройка весов в искусственных нейронных сетях*. Искусственные нейронные сети будут рассматриваться позже, в главе 9, но их ключевой принцип заключается в постепенной подстройке весов сети с целью изучения закономерностей и взаимосвязей в данных. Для этого используется несколько математических техник, но в подходящих сценариях эволюционные алгоритмы оказываются более эффективны.
- *Проектирование радиосхем*. Электронные схемы, состоящие из одинаковых компонентов, можно спроектировать разными способами, и какие-то из них окажутся более эффективными. Например, если два часто взаимодействующих компонента расположить ближе, это может повысить общую эффективность конфигурации. В данном случае эволюционные алгоритмы могут использоваться для построения различных вариаций схем с целью определить оптимальную схему.
- *Симуляция и проектирование молекулярных структур*. Аналогично изменению «поведения» микросхем в зависимости от конфигурации, разные молекулы также ведут себя по-разному и имеют как преимущества, так и недостатки. В этом случае с помощью эволюционных алгоритмов можно генерировать различные молекулярные структуры для стимуляции и изучения с целью определить их поведенческие особенности.

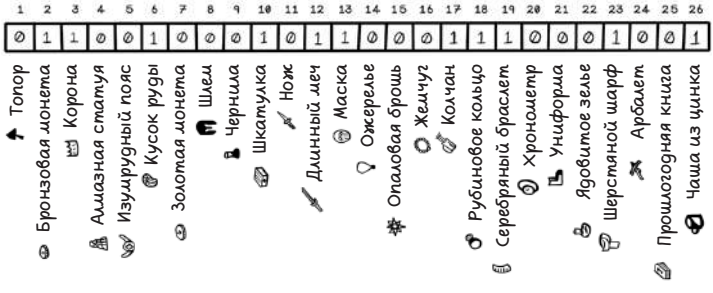
Теперь, когда вы познакомились с общей структурой жизненного цикла генетического алгоритма в главе 4, а также разобрали возможные дополнительные подходы в текущей главе, у вас достаточно знаний, чтобы применять эволюционные алгоритмы в собственных контекстах и решениях.

Краткий обзор главы «Продвинутые эволюционные подходы»

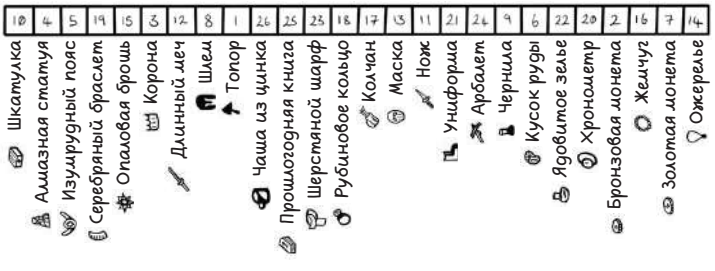
Генетические алгоритмы используются для решения широкого круга задач.

У каждой стратегии отбора есть свои преимущества и недостатки.

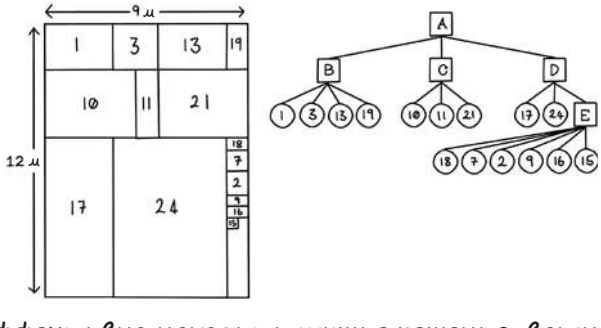
Целочисленное кодирование применяется очень широко.



Порядковое кодирование применяется, когда для решения задачи важен порядок расположения элементов.



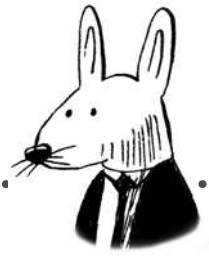
Древовидное кодирование применяется, когда для решения задачи важны связи и иерархия.



Чтобы эффективно находить лучшие решения, важно уметь настраивать все параметры алгоритма.

Роевой интеллект: муравьи

6



В этой главе

- ✓ Знакомство с основами алгоритмов роевого интеллекта
- ✓ Решение задач с помощью алгоритмов роевого интеллекта
- ✓ Проектирование и реализация муравьиного алгоритма

Что такое роевой интеллект?

Алгоритмы роевого интеллекта представляют собой подмножество эволюционных алгоритмов, о которых мы говорили в главе 5; они также известны как бионические алгоритмы. Как и в случае с теорией эволюции, в их основу легло наблюдение за поведением различных форм жизни в естественной среде. В окружающем нас мире мы наблюдаем множество, казалось бы, примитивных живых организмов, которые тем не менее при объединении в группы демонстрируют эмерджентное разумное поведение.

К примерам таких живых организмов можно отнести муравьев. Один муравей способен переносить ношу, в 10–50 раз превосходящую его по весу, и за минуту пробегать расстояние, равное 700-кратной длине своего тела.

Эти качества поистине впечатляют. Тем не менее при работе в коллективе тот же муравей может демонстрировать еще более удивительные способности. Например, группа муравьев может строить муравейники, находить и добывать пищу, а также предупреждать других муравьев, демонстрировать признание сородичей и использовать общественное воздействие на других особей в колонии. Все эти задачи они реализуют с помощью *феромонов*, по сути являющихся запахами, которые муравьи оставляют на своем пути. Другие особи могут чувствовать эти запахи и соответственным образом изменять свое поведение. В муравьином арсенале присутствует от 10 до 20 видов феромонов, с помощью которых они выражают свои намерения. И поскольку отдельные особи используют феромоны для демонстрации своих намерений и нужд, мы наблюдаем эмерджентное разумное поведение в группах муравьев.

На рис. 6.1 показан пример муравьев, дружно работающих над созданием моста между двумя точками, который позволит другим муравьям выполнять свои задачи, например добывать пищу или строительные материалы для муравейника.

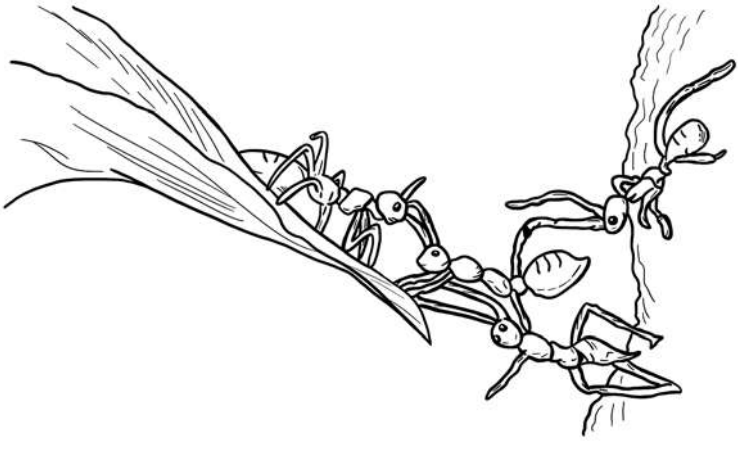


Рис. 6.1. Группа муравьев сообща работает над преодолением препятствия

Эксперимент по наблюдению за реальными муравьями-добытчиками показал, что они всегда склоняются к кратчайшему пути между муравейником и источником пищи. На рис. 6.2 показано отличие между перемещением колонии в начале и в конце, когда муравьи уже проложили пути и интенсивность феромонов на них возросла. Результат наблюдался в классическом эксперименте с асимметричным мостом, где участвовали реальные муравьи. Обратите внимание, что муравьи склоняются к перемещению по кратчайшему пути буквально через восемь минут.

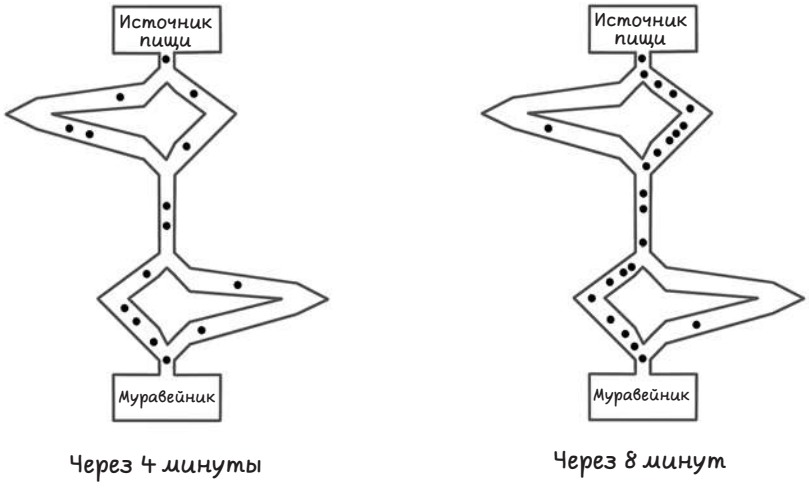


Рис. 6.2. Эксперимент с асимметричным мостом

Муравьиные алгоритмы (ant colony optimization, ACO) симулируют эмерджентное поведение, продемонстрированное в данном эксперименте. В случае поиска кратчайшего пути алгоритм сходится к тому же состоянию, которое наблюдается среди реальных муравьев.

Алгоритмы роевого интеллекта полезны для решения задач оптимизации, когда в определенной области задачи необходимо соблюсти ряд ограничений и ввиду наличия множества возможных решений сложно найти абсолютное лучшее — некоторые оказываются лучше, а некоторые хуже. Эти задачи относятся к тому же классу задач, которые способен решать генетический алгоритм. Выбор алгоритма в данном случае зависит от возможности представления и описания задачи. Мы будем подробнее знакомиться с техническими тонкостями задач оптимизации в главе 7, посвященной изучению оптимизации роем частиц. Роевой интеллект полезен в некоторых реальных сценариях, в том числе представленных на рис. 6.3.

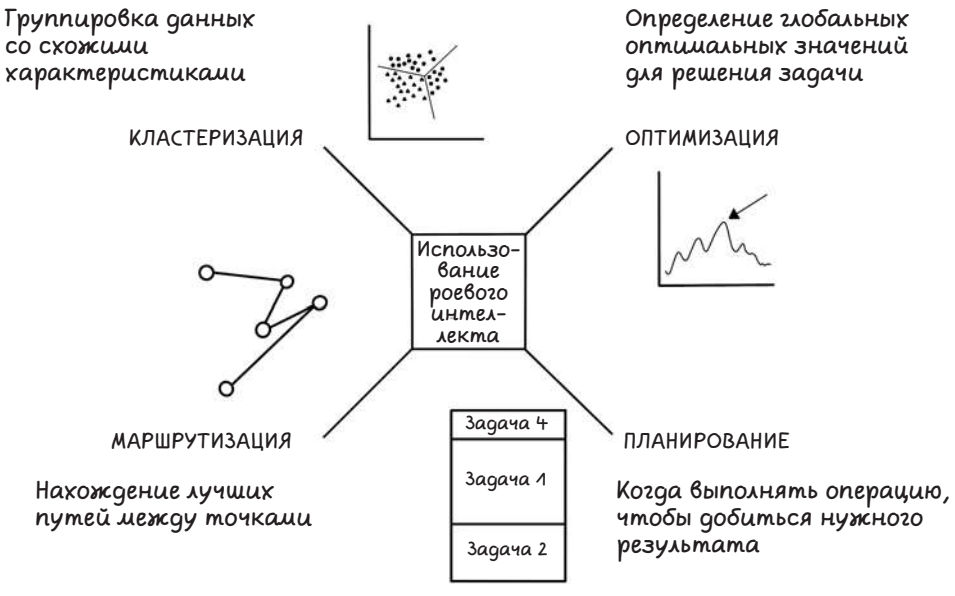


Рис. 6.3. Задачи, решаемые с помощью роевого интеллекта

С учетом общего понимания роевого интеллекта муравьев в следующих разделах описываются конкретные реализации, основанные на этих принципах. Муравьиный алгоритм построен на основе поведения муравьев, которые перемещаются между пунктами назначения, выделяя феромоны и действуя согласно другим встреченным ими феромонам. Эмерджентное поведение муравьев заключается в том, что они действуют по пути наименьшего сопротивления.

Задачи, решаемые с помощью муравьиного алгоритма

Представьте, что вы идете в парк развлечений с множеством аттракционов. Каждый аттракцион располагается отдельно от остальных, и расстояния между ними различаются. Поскольку вы не хотите тратить время на лишнюю ходьбу туда-сюда, то стараетесь найти кратчайшие пути между всеми аттракционами.

На рис. 6.4 показаны аттракционы и расстояния между ними. Обратите внимание, что выбор разных путей изменяет общий маршрут.

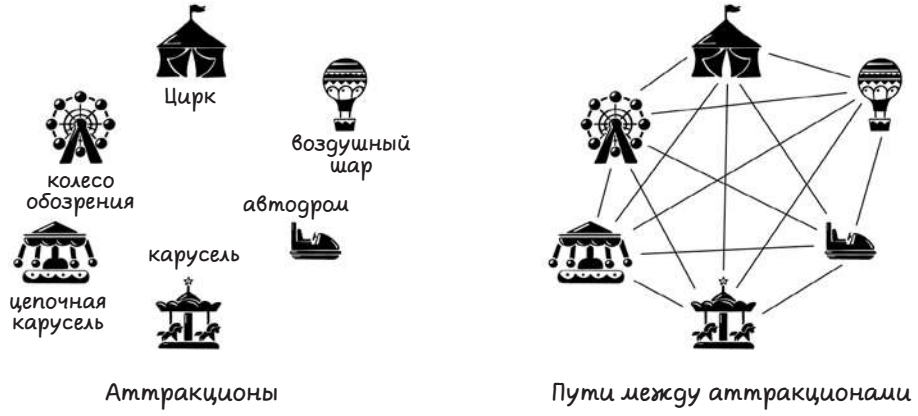


Рис. 6.4. Аттракционы и пути между ними

На рисунке показано 6 аттракционов и 15 соединяющих их путей. Данный пример должен быть вам знаком. Эта задача представляется в виде полного графа, как описывалось в главе 2. Аттракционы являются вершинами, или узлами, а пути между ними — ребрами. Для вычисления количества ребер в полном графе используется приведенная ниже формула. При увеличении количества аттракционов количество ребер резко возрастает:

$$n(n-1)/2$$

На рис. 6.5 показаны расстояния между всеми аттракционами. Здесь же справа показан возможный общий маршрут для посещения каждого аттракциона. Обратите внимание, что линии на рис. 6.5 нарисованы без учета масштаба.

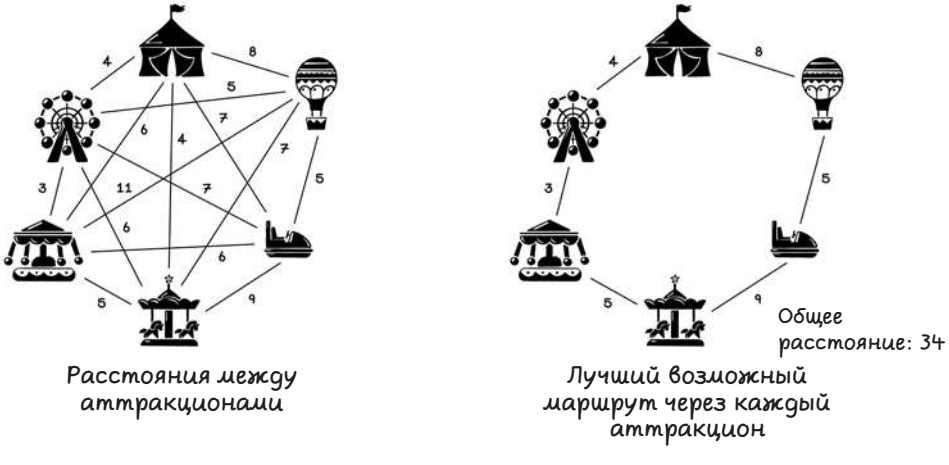


Рис. 6.5. Расстояния между аттракционами и возможный общий маршрут

Если проанализировать расстояния между всеми аттракционами, то можно обнаружить, что оптимальным будет маршрут, показанный на рис. 6.6. По нему мы посещаем аттракционы в таком порядке: цепочная карусель, колесо обозрения, цирк, карусель, воздушный шар и автодром.

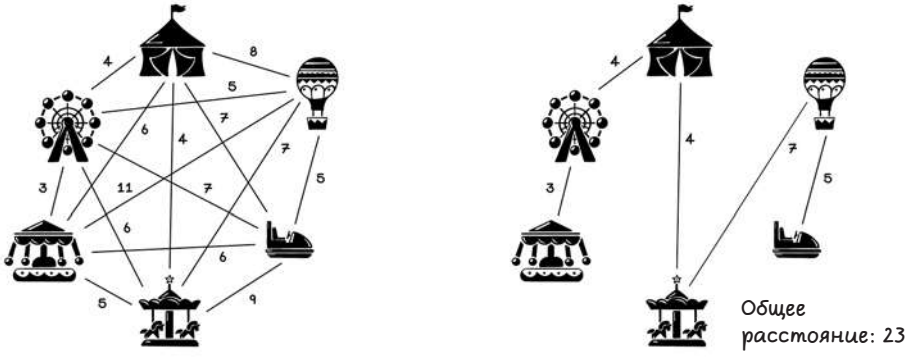


Рис. 6.6. Расстояния между аттракционами и оптимальный путь

Небольшую задачу с шестью аттракционами легко решить самому, но если увеличить их количество до 15, то число возможностей резко возрастет (рис. 6.7). Предположим, что аттракционы — это сервера, а пути между ними — сетевые подключения. Для решения подобных задач требуются интеллектуальные алгоритмы.

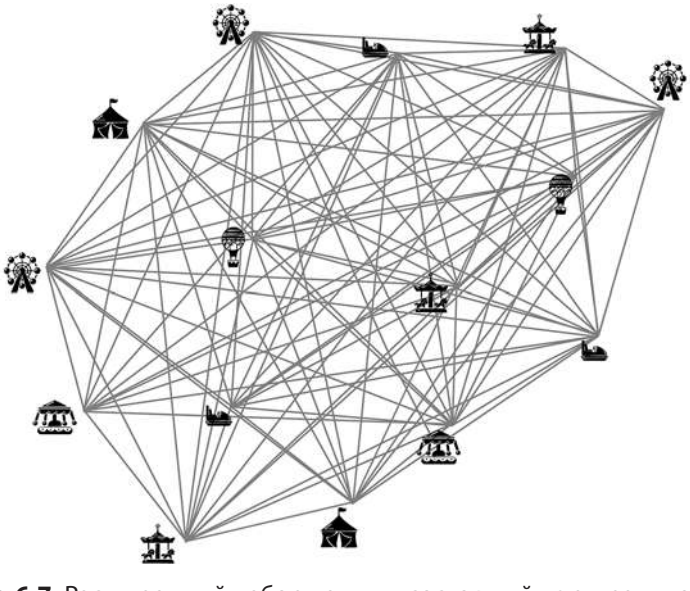
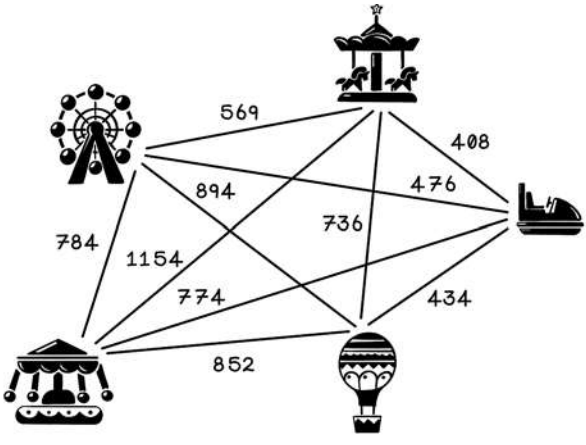
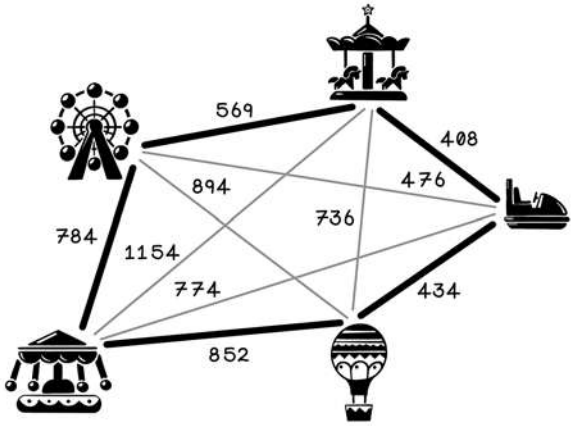


Рис. 6.7. Расширенный набор данных, состоящий из аттракционов и соединяющих их путей

УПРАЖНЕНИЕ: НАЙДИТЕ ВРУЧНУЮ КРАТЧАЙШИЙ ПУТЬ В ПРЕДЛОЖЕННОМ ПРИМЕРЕ ПАРКА РАЗВЛЕЧЕНИЙ.



ОТВЕТ: КРАТЧАЙШИЙ ПУТЬ В ПРИМЕРЕ ПАРКА РАЗВЛЕЧЕНИЙ.



Одним из способов решения этой задачи будет брутфорс: каждый вариант обхода (обход — это последовательность однократных посещений каждого аттракциона) генерируется и оценивается, пока не будет найден обход с наименьшей общей дистанцией. Опять же, это решение может показаться закономерным, но в больших наборах данных вычисление окажется очень дорогостоящим и затратным по времени. К примеру, поиск брутфорсом наилучшего решения для задачи с 48 аттракционами занимает приблизительно 10 часов.

Представление состояния: как выглядят муравьи и их пути?

Рассматривая задачу с парком развлечений, нам нужно представить ее данные так, чтобы их можно было обработать муравьиным алгоритмом. Поскольку у нас есть несколько аттракционов и нам также известны расстояния между ними, задачу можно точно представить с помощью матрицы расстояний.

Матрица расстояний — это двумерный массив, в котором индексы представляют объекты, а их значения отражают расстояние между этими объектами. Эта матрица аналогична матрице смежности, которую мы рассматривали в главе 2 (рис. 6.8 и табл. 6.1).

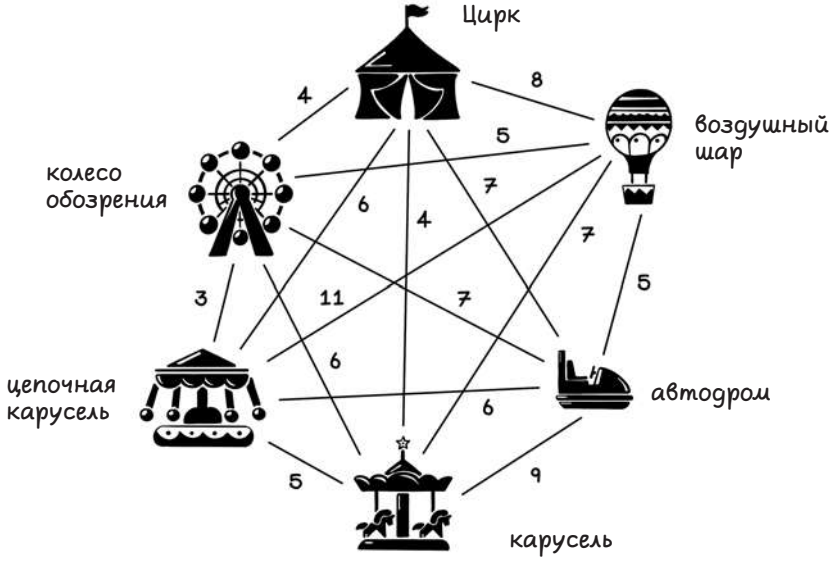


Рис. 6.8. Пример задачи с парком развлечений

Таблица 6.1. Расстояния между аттракционами

	Цирк	Воздушные шары	Автодром	Карусель	Цепочная карусель	Колесо обозрения
Цирк	0	8	7	4	6	4
Воздушные шары	8	0	5	7	11	5
Автодром	7	5	0	9	6	7
Карусель	4	7	9	0	5	6
Цепочная карусель	6	11	6	5	0	3
Колесо обозрения	4	5	7	6	3	0

Псевдокод

Расстояния между аттракционами можно представить как матрицу, то есть массив массивов, в котором индекс в позиции x, y обозначает расстояние между аттракционами x и y. Обратите внимание, что расстояние между одним и тем же аттракционом равно 0. Этот массив можно также создать программно путем перебора данных в файле и генерации каждого элемента:

```
let attraction_distances equal
    [
     [0,8,7,4,6,4],
     [8,0,5,7,11,5],
     [7,5,0,9,6,7],
     [4,7,9,0,5,6],
     [6,11,6,5,0,3],
     [4,5,7,6,3,0],
    ]
```

Далее нужно представить муравьев. Муравьи перемещаются к разным аттракционам, оставляя за собой след из феромонов. При этом они принимают решение, какой аттракцион посетить следующим, а в конце маршрута каждый из них делает вывод, какое расстояние он прошел. Вот основные характеристики муравья (рис. 6.9):

- *Память*. В алгоритме ACO это список посещенных аттракционов.
- *Лучшая приспособленность*. Это кратчайший путь между всеми аттракционами.
- *Действие*. Выбрать следующую точку для посещения и оставить по дороге след феромонами.

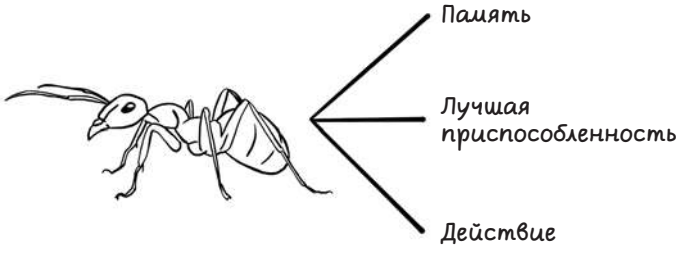


Рис. 6.9. Свойства муравья

Псевдокод

Несмотря на то что этот абстрактный принцип муравья включает память, лучшую приспособленность и действие, для решения задачи также потребуются определенные данные и функции. Логику муравья мы инкапсулируем в класс. При инициализации экземпляра класса `Ant` создается пустой массив, представляющий список аттракционов, которые он будет посещать. Помимо этого, в качестве начальной точки для конкретного муравья будет выбираться случайный аттракцион:

```
Ant(attraction_count):
  let ant.visited_attractions equal an empty array
  append a random number between 0 and
    (attraction_count - 1) to ant.visited_attractions
```

Класс `Ant` также содержит несколько функций, отвечающих за передвижение муравья. Функции `visit_*` используются для определения, в каком направлении муравей должен продолжить движение. Функция `visit_attraction` генерирует шанс посещения случайного аттракциона. Если шанс срабатывает, то вызывается `visit_random_attraction`; в противном случае используется `roulette_wheel_selection` с вычисленным списком вероятностей. Более подробное описание будет приведено в следующем разделе:

```
Ant functions:
visit_attraction(pheromone_trails)
visit_random_attraction()
visit_probabilistic_attraction(pheromone_trails)
roulette_wheel_selection(probabilities)
get_distance_traveled()
```

Последняя функция `get_distance_traveled` на основе списка посещенных отдельным муравьем аттракционов вычисляет общее пройденное им расстояние. Это расстояние нужно минимизировать, чтобы найти кратчайший путь, и для муравьев оно выступает в качестве показателя приспособленности:

```
get_distance_travelled(ant):
  let total_distance equal 0
  for a in range(1, length of ant.visited_attractions):
    total_distance += distance between ant.visited_attractions[a - 1] and
                                       ant.visited_attractions[a]
  return total_distance
```

Заключительная структура данных — это принцип формирования следов феромонов. По аналогии с расстояниями между аттракционами интенсивность феромонов по каждому пути можно представить в виде матрицы. Только вместо расстояний она будет содержать значения интенсивности феромонов. На рис. 6.10 более толстые линии указывают на более интенсивные следы, а в табл. 6.2 отражена их интенсивность между аттракционами.

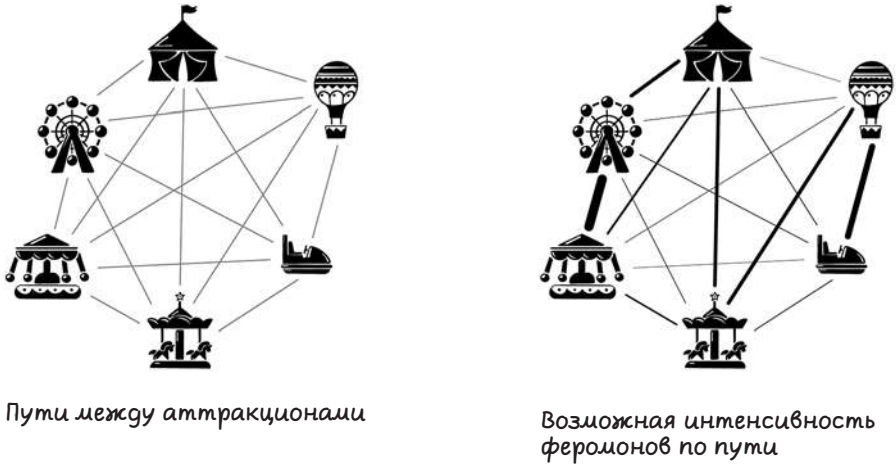


Рис. 6.10. Пример интенсивности феромонов на разных путях

Таблица 6.2. Интенсивность феромонов между аттракционами

	Цирк	**Воздушные шары**	**Автодром**	**Карусель**	**Цепочная карусель**	**Колесо обозрения**
Цирк	0	2	0	8	6	8
Воздушные шары	2	0	10	8	2	2
Автодром	2	10	0	0	2	2
Карусель	8	8	2	0	2	2
Цепочная карусель	6	2	2	2	0	10
Колесо обозрения	8	2	2	2	10	0

Жизненный цикл муравьиного алгоритма

Разобравшись с необходимыми структурами данных, можно переходить к реализации самого муравьиного алгоритма. Подход к его разработке всегда основывается на области решаемой задачи. У каждой задачи есть свой уникальный контекст и предметная область, в которой представлены данные, но принципы всегда остаются одинаковы.

С учетом сказанного перейдем к рассмотрению настройки алгоритма для решения задачи парка развлечений. В общем виде его жизненный цикл будет следующим:

- *Инициализация следов феромонов*. Создание принципа формирования следов феромонов между аттракционами и инициализация значений их интенсивности.
- *Настройка популяции муравьев*. Создание популяции муравьев, которые начинают свой путь от разных аттракционов.
- *Выбор следующей точки посещения*. Выбор следующего аттракциона, пока не будут пройдены все.
- *Обновление следов феромонов*. Обновление значения интенсивности феромонов с учетом количества перемещений по ним муравьев, а также фактора испаряемости.
- *Обновление лучшего решения*. Обновление лучшего решения с учетом общего расстояния, пройденного каждым муравьем.
- *Определение критерия остановки*. Процесс посещения муравьями аттракционов повторяется несколько раз. Одна итерация — это однократное посещение каждым муравьем всех аттракционов. Критерий остановки определяет общее число итераций. Чем их больше, тем легче муравьям найти на основе следов феромонов лучшие решения.

Рисунок 6.11 описывает общий жизненный цикл муравьиного алгоритма оптимизации.

Инициализация следов феромонов

Это первый этап алгоритма. Так как муравьи еще не совершили ни одного обхода аттракционов, то пути феромонов инициализируются со значением 1. При установке этого значения для каждого пути ни один из них не получает преимущества перед другими. Важный аспект здесь — это определение надежной структуры данных, которая будет содержать значения следов. Ее мы и рассмотрим далее (рис. 6.12).

Этот принцип можно применить и к другим задачам, в которых вместо расстояний между точками интенсивность феромонов определяется другой эвристикой.

На рис. 6.13 эвристикой выступает расстояние между аттракционами.

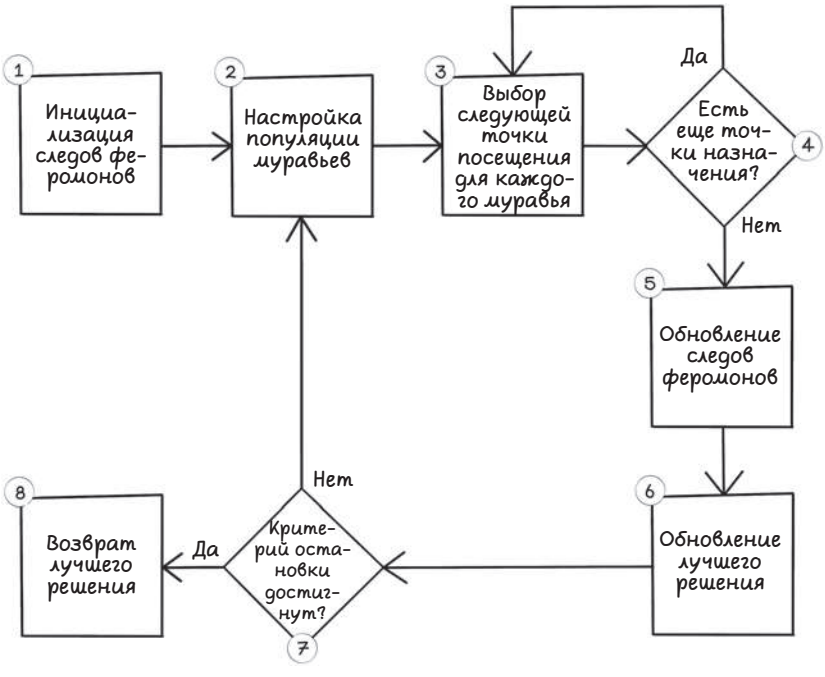


Рис. 6.11. Жизненный цикл муравьиного алгоритма

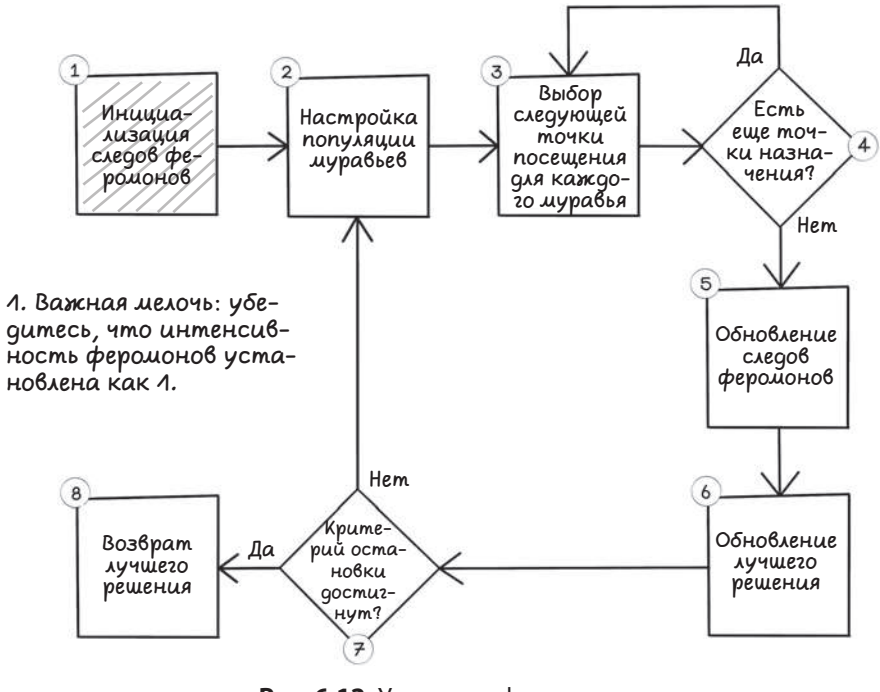


Рис. 6.12. Установка феромонов

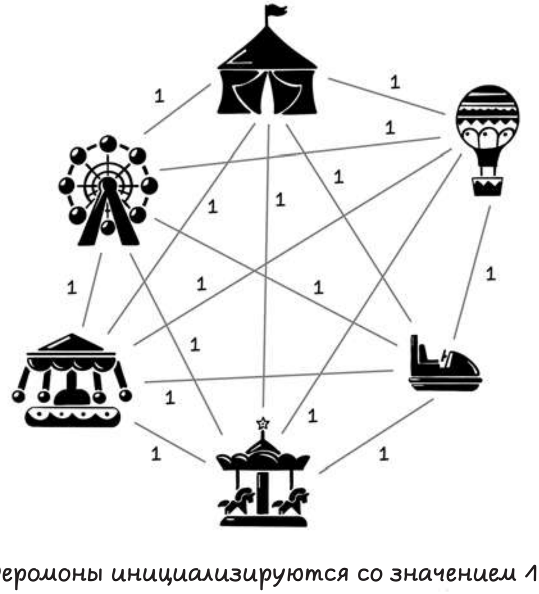


Рис. 6.13. Инициализация феромонов

Псевдокод

По аналогии с расстояниями между аттракционами следы феромонов можно представить в виде матрицы, но в этом случае индексы в точках x, y будут указывать интенсивность феромонов на пути между x и y. Начальная интенсивность устанавливается как 1. Значения для всех путей должны инициализироваться одинаково, чтобы исключить начальное смещение:

```
let pheromone_trails equal
    [
     [1,1,1,1,1,1],
     [1,1,1,1,1,1],
     [1,1,1,1,1,1],
     [1,1,1,1,1,1],
     [1,1,1,1,1,1],
     [1,1,1,1,1,1]
    ]
```

Настройка популяции муравьев

Следующий шаг — создание популяции муравьев, которые будут перемещаться между аттракционами и оставлять за собой следы феромонов (рис. 6.14).

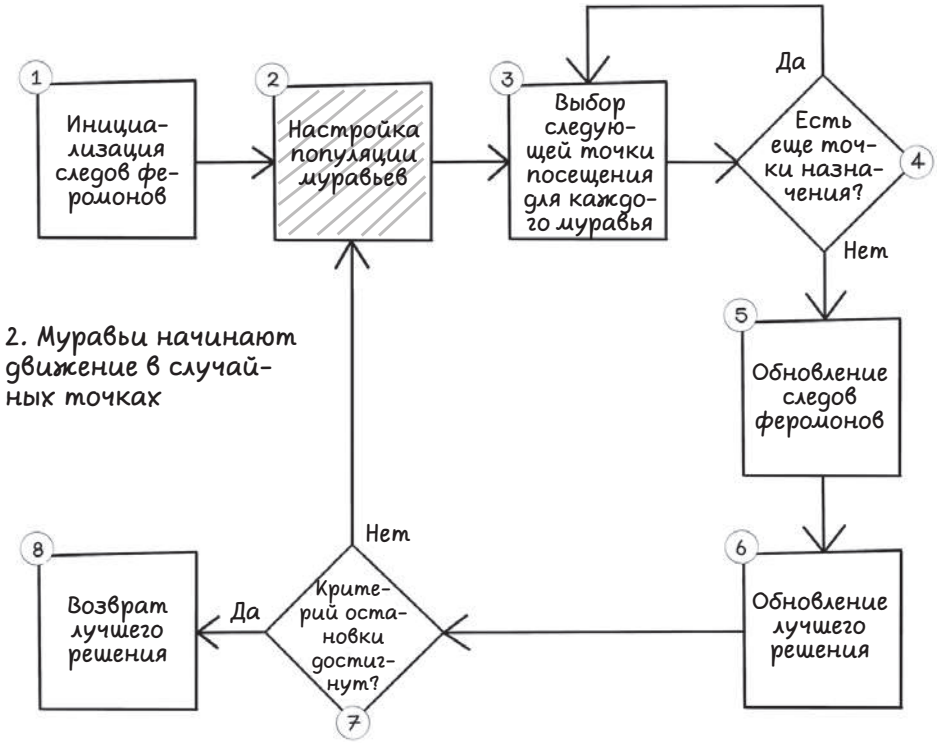


Рис. 6.14. Настройка популяции муравьев

Муравьи будут начинать маршрут от случайно назначаемых аттракционов (рис. 6.15), то есть в случайной точке потенциальной последовательности, поскольку муравьиный алгоритм можно применять к задачам, в которых фактическое расстояние не существует. После обхода всех аттракционов муравьи возвращаются на начальные позиции.

Этот принцип можно адаптировать и для других случаев. Например, в задаче планирования очередности заданий каждый муравей начинает со случайного задания.

Рис. 6.15. Муравьи начинают маршрут от случайных аттракционов

Псевдокод

Настройка колонии муравьев включает инициализацию нескольких особей и их добавление в список, через который к ним можно будет обращаться позднее. Помните, что функция инициализации в классе `Ant` выбирает для начала случайный аттракцион:

```
setup_ants(attraction_count, number_of_ants_factor):
  let number_of_ants equal round(attraction_count * number_of_ants_factor)
  let ant_colony equal to an empty array
  for i in range(0, number_of_ants):
    append new Ant to ant_colony
  return ant_colony
```

Выбор следующей точки посещения

Муравьям нужно выбирать аттракционы для посещения. В течение обхода они поочередно переходят к каждому новому объекту развлечений, пока не посетят все. Выбор очередного аттракциона происходит на основе двух факторов (рис. 6.16):

- *Интенсивности феромонов* — показателя интенсивности феромонов среди всех доступных путей.
- *Эвристического значения* — наилучшего значения эвристики среди всех доступных путей, которое в примере с парком развлечений представляет собой расстояние между аттракционами.

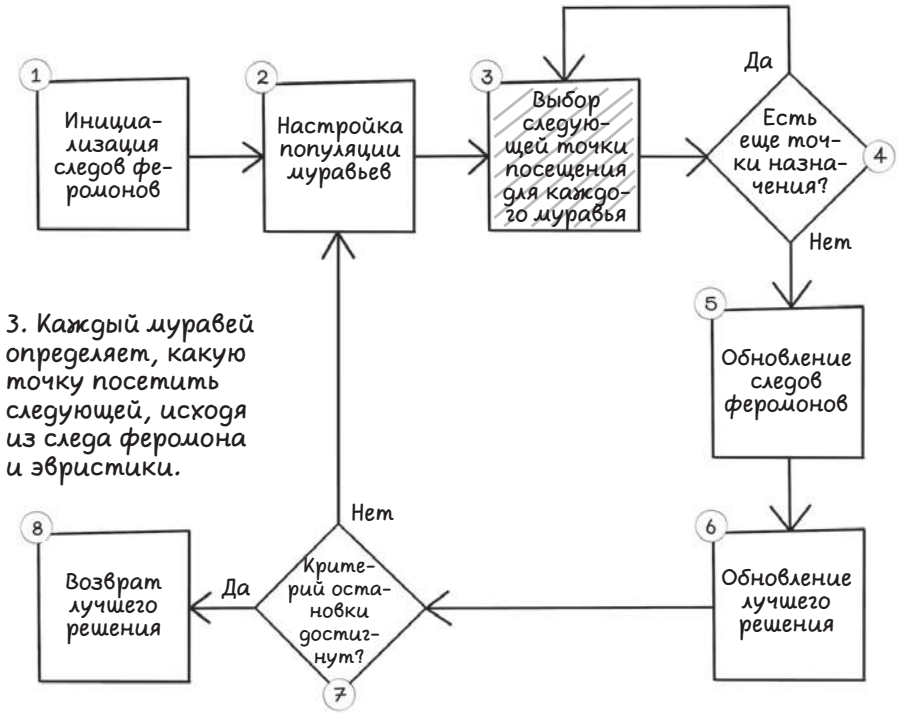


Рис. 6.16. Выбор следующего места посещения

Муравьи не возвращаются к уже посещенным в текущем обходе аттракционам.

Стохастическая природа муравьев

В муравьином алгоритме присутствует элемент случайности. Его смысл в том, чтобы дать муравьям возможность изучать менее оптимальные ближайшие пути, которые могут привести к сокращению общего маршрута обхода.

Изначально для муравья определяется вероятность принятия решения выбрать случайную точку посещения, например 10 %. Далее можно генерировать случайное число между 0 и 1, и при результате 0,1 (вероятность 10 %) или меньше муравей выберет случайное место назначения среди всех доступных.

Выбор точки посещения на основе эвристики

Когда для муравья наступает момент выбора следующего места посещения в неслучайном порядке, он определяет интенсивность феромонов каждого пути и значение эвристики по следующей формуле:

$$\frac{(\text{феромоны на пути x})^a * (1 / \text{эвристика для пути x})^b}{\text{Сумма n доступных точек } ((\text{феромоны на пути n})^a * (1 / \text{эвристика для пути n})^b)}$$

После применения этой функции ко всем доступным путям муравей выбирает тот, для которого было получено лучшее значение. На рис. 6.17 показаны возможные перемещения от цирка с соответствующими расстояниями и интенсивностями феромонов.

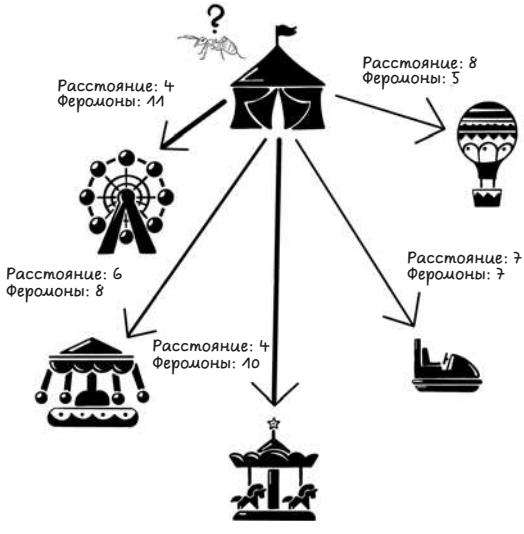


Рис. 6.17. Пример возможных перемещений от цирка

Далее мы разберем эту формулу, чтобы понять вычисления и влияние результатов на принятие решения (рис. 6.18).

$$\underbrace{(\text{феромоны на пути x})^a}_{\text{Влияние феромонов}} * \underbrace{(1 / \text{эвристика для пути x})^b}_{\text{Влияние эвристики}}$$

Рис. 6.18. Влияние показателя феромонов и эвристики в формуле

Переменные *alpha (a)* и *beta (b)* используются для увеличения влияния феромонов или эвристики. С помощью этих переменных можно регулировать баланс между выбором муравьем решения на основе собственных знаний или на основе следов феромонов, которые представляют знания колонии о том или ином пути. Эти параметры задаются предварительно и в процессе выполнения алгоритма, как правило, не корректируются.

В следующем примере рассматривается каждый начинающийся от цирка путь и вычисляются вероятности перемещения к каждому соответствующему аттракциону.

- *a (alpha)* устанавливается как 1.
- *b (beta)* устанавливается как 2.

Так как *b* больше, чем *a*, в этом примере преобладает влияние эвристики.

Далее разберем пример вычислений, используемых для определения вероятности выбора конкретного пути (рис. 6.19).

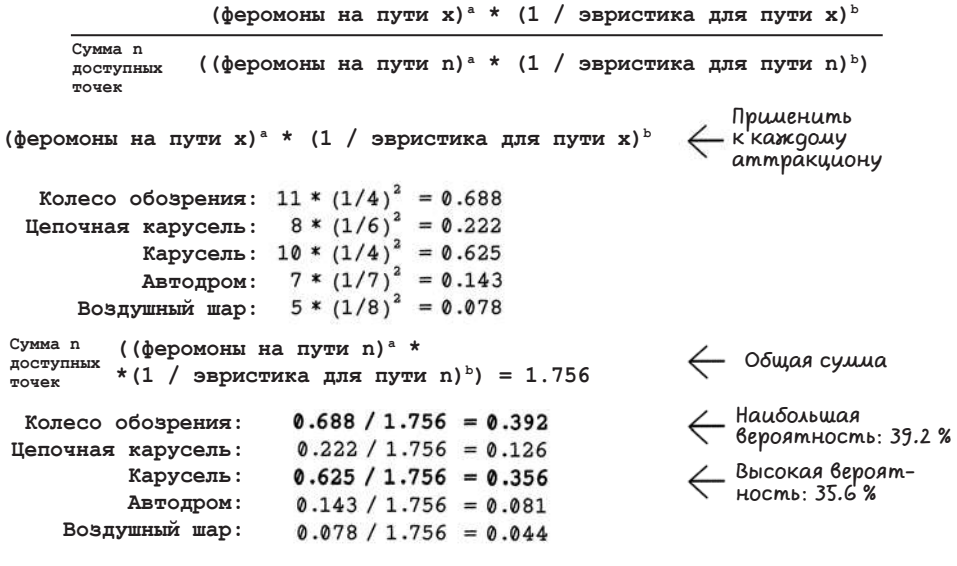


Рис. 6.19. Вычисление вероятностей выбора пути

После этих вычислений с учетом доступных точек назначения перед муравьем возникает выбор, показанный на рис. 6.20.

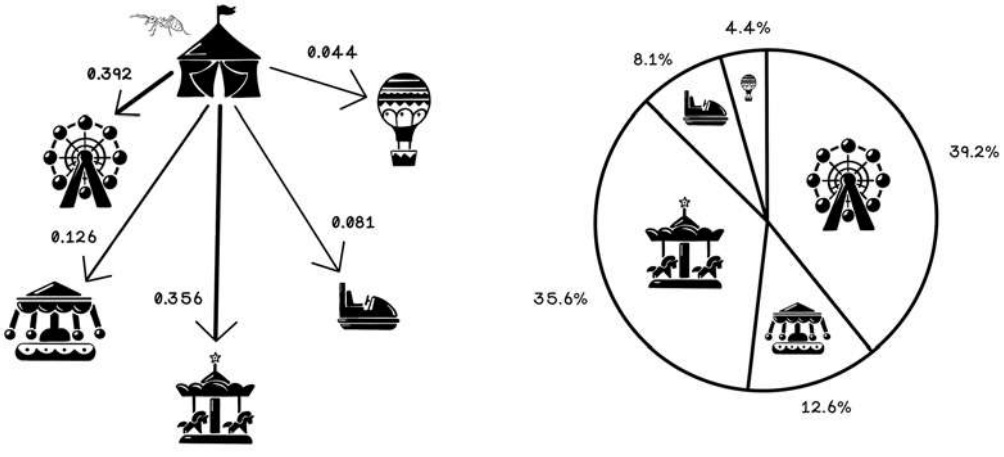


Рис. 6.20. Итоговая вероятность выбора каждого аттракциона

Помните, что рассматриваются только доступные пути, которые еще не были исследованы. На рис. 6.21 показаны возможные перемещения от цирка, за исключением колеса обозрения, так как оно уже посещалось. На рис. 6.22 приводятся вычисления вероятностей выбора одного из путей.

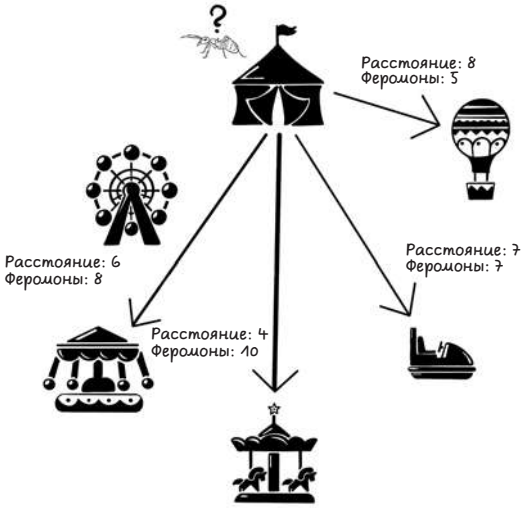


Рис. 6.21. Пример возможных путей от цирка, за исключением уже изученных

$$\frac{(\text{феромоны на пути x})^a * (1 / \text{эвристика для пути x})^b}{\text{Сумма n доступных точек } ((\text{феромоны на пути n})^a * (1 / \text{эвристика для пути n})^b)}$$

(феромоны на пути x)^a * (1 / эвристика для пути x)^b ← Применить к каждому аттракциону

цепочная карусель: 8 * $(1/6)^2$ = 0.222
карусель: 10 * $(1/4)^2$ = 0.625
автодром: 7 * $(1/7)^2$ = 0.143
воздушный шар: 5 * $(1/8)^2$ = 0.078

Сумма n доступных точек ((феромоны на пути n)^a * *(1 / эвристика для пути n)^b) = 1.068 ← Общая сумма

цепочная карусель: 0.222 / 1.068 = 0.208
карусель: **0.625 / 1.068 = 0.585** ← Наибольшая вероятность: 58.5 %
автодром: 0.143 / 1.068 = 0.134
воздушный шар: 0.078 / 1.068 = 0.073

Рис. 6.22. Вычисление вероятностей выбора одного из путей

На рис. 6.23 показано, каковы вероятности решений, принятых муравьем.

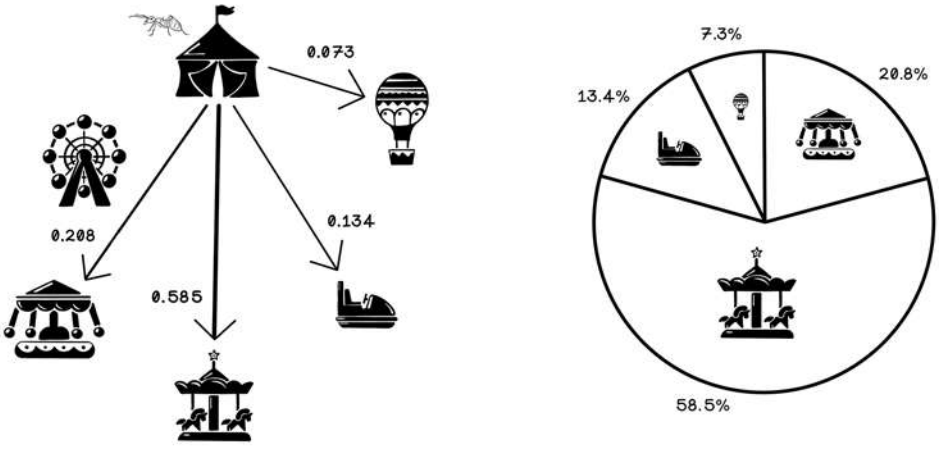


Рис. 6.23. Итоговая вероятность выбора каждого аттракциона

Псевдокод

Псевдокод для вычисления вероятностей посещения доступных аттракционов тесно связан с математическими функциями, которые мы рассмотрели. Вот ряд интересных особенностей реализации:

- *Определение доступных для посещения аттракционов.* Так как муравей уже посетил некоторые аттракционы, он не должен на них возвращаться. Массив `possible_attractions` (возможные аттракционы) хранит значения доступных аттракционов, удаляя `visited_attractions` (посещенные аттракционы) из полного списка `all_attractions` (все аттракционы).
- *Использование трех переменных для хранения результатов вычисления вероятностей.* `possible_indexes` хранит индексы аттракционов; `possible_ probabilities` хранит вероятности для соответствующих индексов; `total_ probabilities` хранит сумму всех вероятностей, которая на момент завершения функции должна равняться 1. Для большей чистоты кода эти три структуры данных можно представить с помощью класса.

```
visit_probabilistic_attraction(pheromone_trails, attraction_count, ant
                               alpha, beta):
  let current_attraction equal ant.visited_attractions[-1]
  let all_attractions equal range(0, attraction_count)
  let possible_attractions equal all_attractions - ant.visited_attractions

  let possible_indexes equal empty array
  let possible_probabilities equal empty array
  let total_probabilities equal 0

  for attraction in possible_attractions:
    append attraction to possible_indexes
    let pheromones_on_path equal
      math.pow(pheromone_trails[current_attraction][attraction], alpha)
    let heuristic_for_path equal
      math.pow(1/attraction_distances[current_attraction][attraction], beta)
```

```
    let probability equal pheromones_on_path * heuristic_for_path
    append probability to possible_probabilities
    add probability to total_probabilities
  let possible_probabilities equal [probability / total_probabilities
    for probability in possible_probabilities]
  return [possible_indexes, possible_probabilities]
```

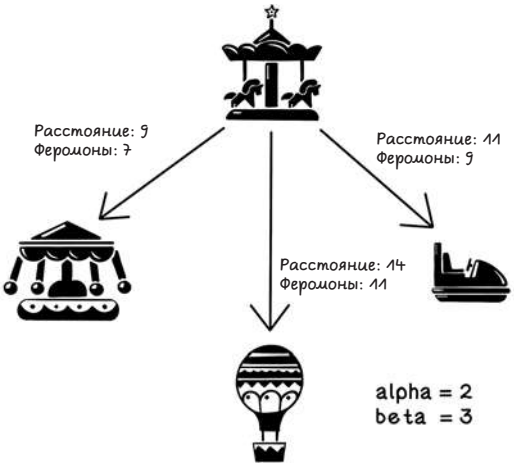
Мы снова встречаемся с отбором методом рулетки. Эта функция получает в качестве ввода возможные вероятности и индексы аттракционов. Она генерирует список срезов, каждый из которых включает индекс аттракциона в элементе 0, начало среза в индексе 1 и конец среза в индексе 2. Значения начала и конца каждого среза находятся в диапазоне от 0 до 1. Далее генерируется случайное число между 0 и 1, и срезы, в диапазон которых попадает это число, выбираются победителями:

```
roulette_wheel_selection(possible_indexes, possible_probabilities,
                         possible_attraction_count):
  let slices equal empty array
  let total equal 0
  for i in range(0, possible_attractions_count):
    append [possible_indexes[i], total, total + possible_probabilities[i]]
      to slices
    total += possible_probabilities[i]
  let spin equal random(0, 1)
  let result equal [slice for slice in slices if slice[1] < spin <= slice[2]]
  return result
```

Теперь, когда у нас есть вероятности выбора аттракциона для посещения, используем отбор рулеткой.

Коротко говоря, при отборе рулеткой, рассмотренном в главах 3 и 4, разным вероятностям выделяются разные секции колеса на основе их приспособленности: чем выше приспособленность, тем большая секция колеса выделяется (рис. 6.23). Затем колесо «вращается» и выбирается «особь». Процесс выбора и посещения аттракционов продолжается для всех муравьев, пока каждый из них не побывает на всех аттракционах.

УПРАЖНЕНИЕ: ОПРЕДЕЛИТЬ ВЕРОЯТНОСТИ ПОСЕЩЕНИЯ АТТРАКЦИОНОВ НА ОСНОВЕ СЛЕДУЮЩЕЙ ИНФОРМАЦИИ



ОТВЕТ: ВЕРОЯТНОСТИ ПОСЕЩЕНИЯ АТТРАКЦИОНОВ

$$\frac{(\text{феромоны на пути x})^a * (1 / \text{эвристика для пути x})^b}{\text{Сумма n доступных точек } ((\text{феромоны на пути n})^a * (1 / \text{эвристика для пути n})^b)}$$

$$(\text{феромоны на пути x})^a * (1 / \text{эвристика для пути x})^b$$

Цепочная карусель: $7^2 * (1/9)^3 = 0.067$
Автодром: $9^2 * (1/11)^3 = 0.061$
Воздушный шар: $11^2 * (1/14)^3 = 0.044$

$$\text{Сумма n доступных точек } ((\text{феромоны на пути n})^a * *(1 / \text{эвристика для пути n})^b) = 0.172$$

Цепочная карусель: 0.067 / 0.172 = 0.39
Автодром: 0.061 / 0.172 = 0.355
Воздушный шар: 0.044 / 0.172 = 0.256

Обновление следов феромонов

Пройдя по всем аттракционам, муравьи оставили за собой феромоны, изменив таким образом следы (рис. 6.24).

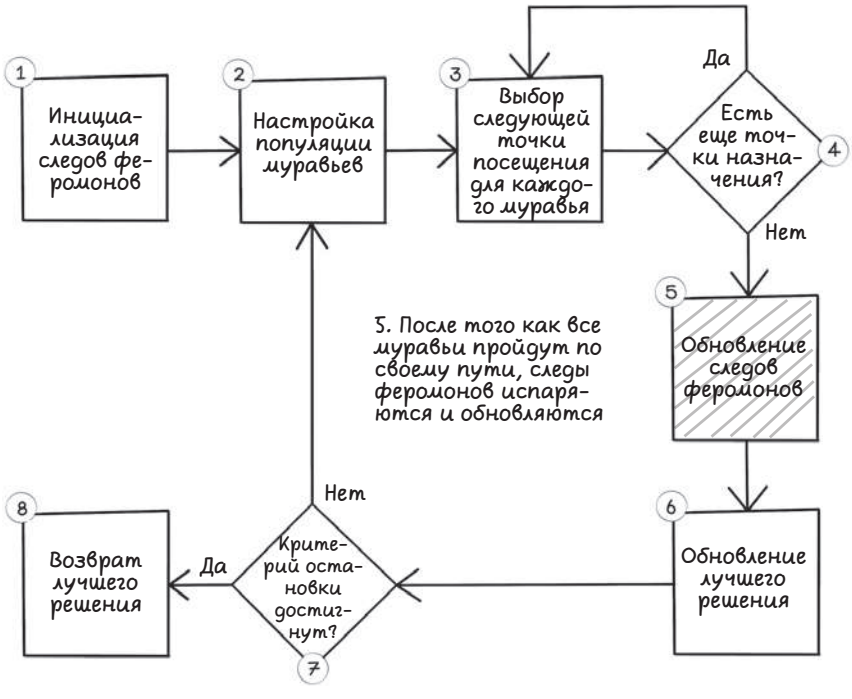
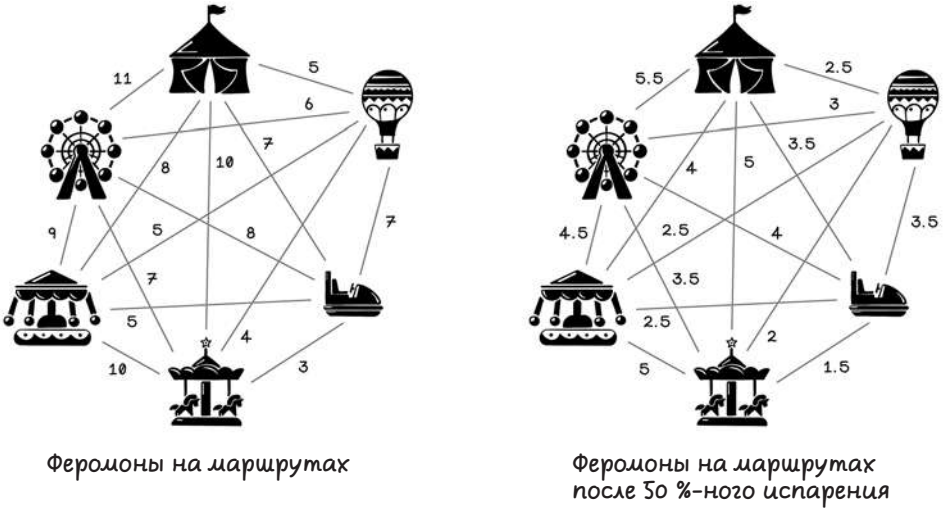


Рис. 6.24. Обновление следов феромонов

Обновление следов происходит в два этапа: испарение старых феромонов и выделение новых.

Испарение феромонов

Принцип испарения работает аналогично природному, то есть с течением времени следы утрачивают свою интенсивность. Феромоны обновляются путем умножения их соответствующих текущих значений на фактор испарения — параметр, который можно подстраивать, изменяя результативность алгоритма в плане разведки и эксплуатации. На рис. 6.25 показаны обновленные следы после испарения.



Феромоны на маршрутах

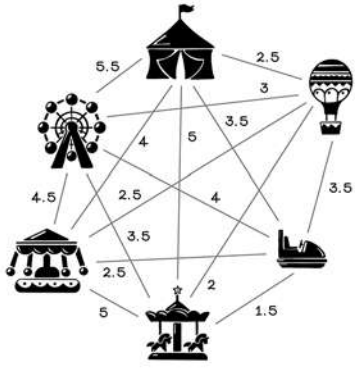
Феромоны на маршрутах после 50 %-ного испарения

Рис. 6.25. Пример обновления следов феромонов после испарения

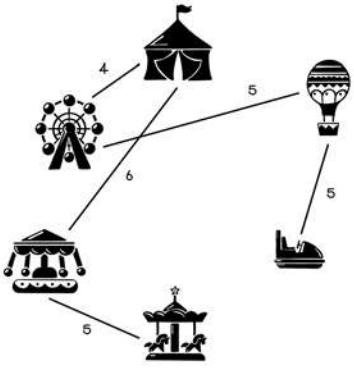
Обновление следов на основе перемещений муравьев

Интенсивность феромонов каждого пути обновляется также на основе количества прошедших по нему муравьев — чем их больше, тем интенсивность выше.

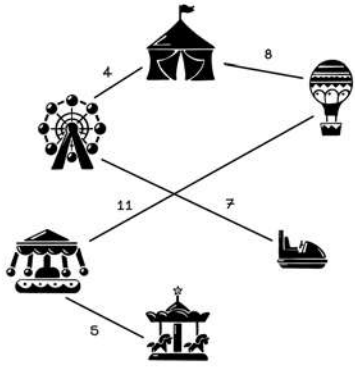
Значение фитнес-функции каждого муравья определяет количество оставляемых им на путях феромонов. В результате муравьи с лучшими вариантами маршрутов оказывают больше влияния на лучшие пути. На рис. 6.26 показаны следы феромонов, обновленные по этому принципу.



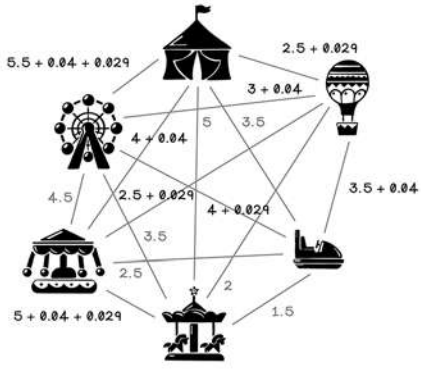
Феромоны на маршрутах после испарения



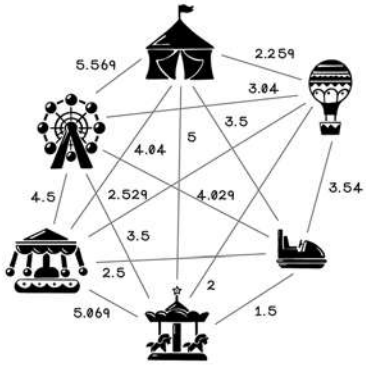
Муравей A, всего: 25
1/25 = 0.04



Муравей B, всего: 35
1/35 = 0.029



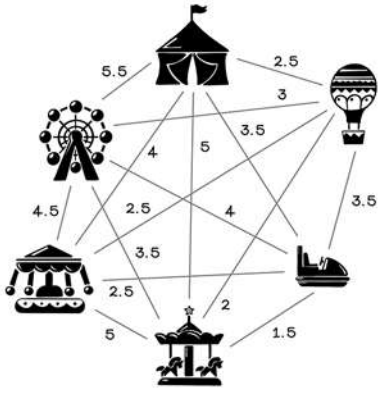
Добавление феромонов после обновления



Феромоны после обновления

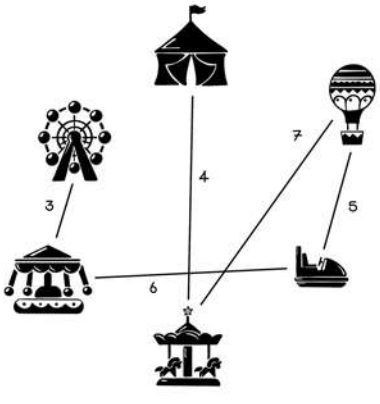
Рис. 6.26. Обновление следов на основе перемещений муравьев

УПРАЖНЕНИЕ: ВЫЧИСЛИТЕ ОБНОВЛЕНИЕ ФЕРОМОНОВ ДЛЯ СЛЕДУЮЩЕГО СЦЕНАРИЯ

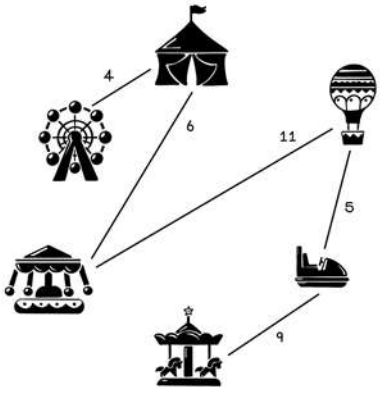


Расчет испарения 50 %

Феромоны на маршрутах

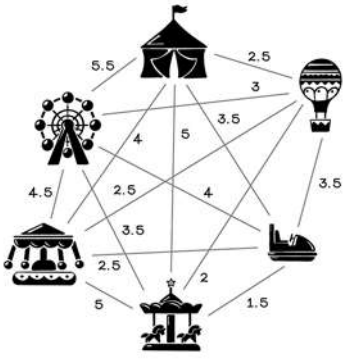


Путь муравья A

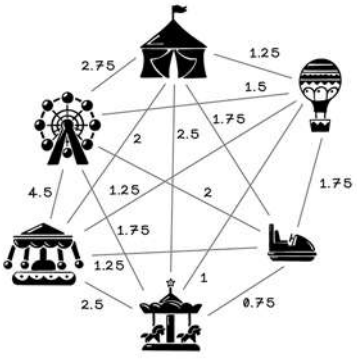


Путь муравья B

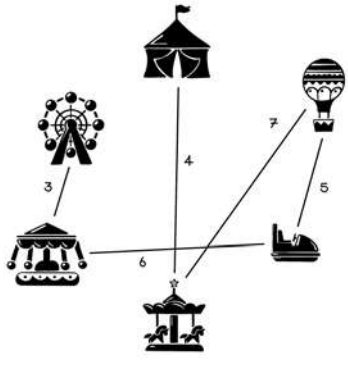
ОТВЕТ: ОБНОВЛЕНИЕ ФЕРОМОНОВ



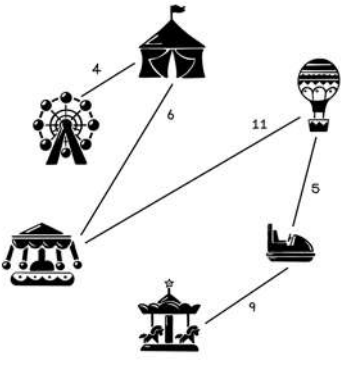
Феромоны на маршрутах



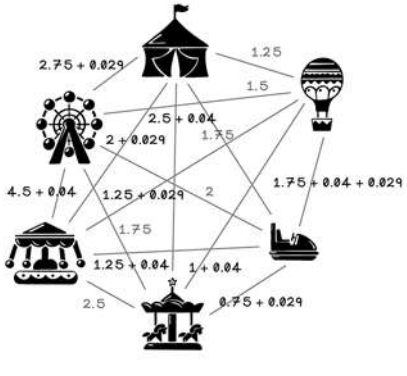
Феромоны на маршрутах после испарения 50 %



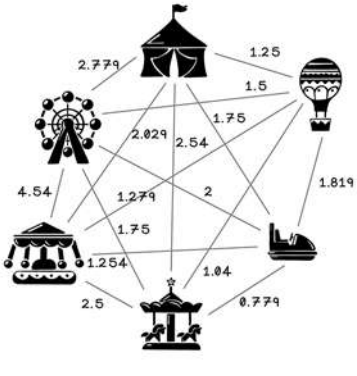
Муравей A, всего: 25
1/25 = 0.04



Муравей B, всего: 35
1/35 = 0.029



Добавление феромонов после обновления



Феромоны после обновления

Псевдокод

Функция `update_pheromones` применяет к следам феромонов два важных принципа. Сначала на основе показателя испаряемости уменьшается интенсивность следов. Если показатель равен, например, 0.5, то интенсивность уменьшается вдвое. Количество оставляемых каждым муравьем феромонов определяется значением его фитнес-функции, которым в данном случае является общее пройденное расстояние.

```
update_pheromones(evaporation_rate, pheromone_trails, attraction_count):
  for x in range(0, attraction_count):
    for y in range(0, attraction_count):
      let pheromone_trails[x][y] equal
        pheromone_trails[x][y] * evaporation_rate
      for ant in ant_colony:
        pheromone_trails[x][y] += 1 / ant.get_distance_traveled()
```

Обновление лучшего решения

Лучшее решение — это последовательность посещенных аттракционов с наименьшим общим расстоянием (рис. 6.27).

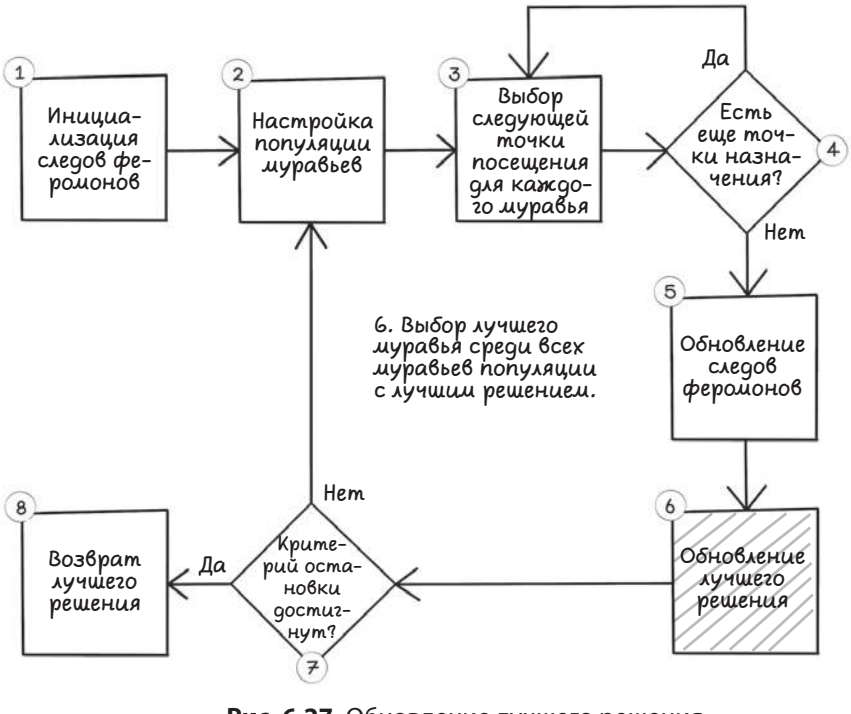


Рис. 6.27. Обновление лучшего решения

Псевдокод

По завершении итерации, когда все муравьи посетили каждый аттракцион, определяется, какой из них нашел кратчайшую дорогу. Этот муравей устанавливается как лучший в колонии:

```
get_best(ant_population, previous_best_ant):
  let best_ant equal previous_best_ant
  for ant in ant_population:
    let distance_traveled equal ant.get_distance_traveled()
    if distance_traveled < best_ant.best_distance:
      let best_ant equal ant
  return best_ant
```

Определение критерия остановки

Алгоритм останавливается после нескольких итераций: по сути, количества обходов, совершаемых муравьями. К примеру, десять итераций будут означать, что каждый муравей сделает 10 обходов, в каждом из которых он посетит все аттракционы по одному разу (рис. 6.28).

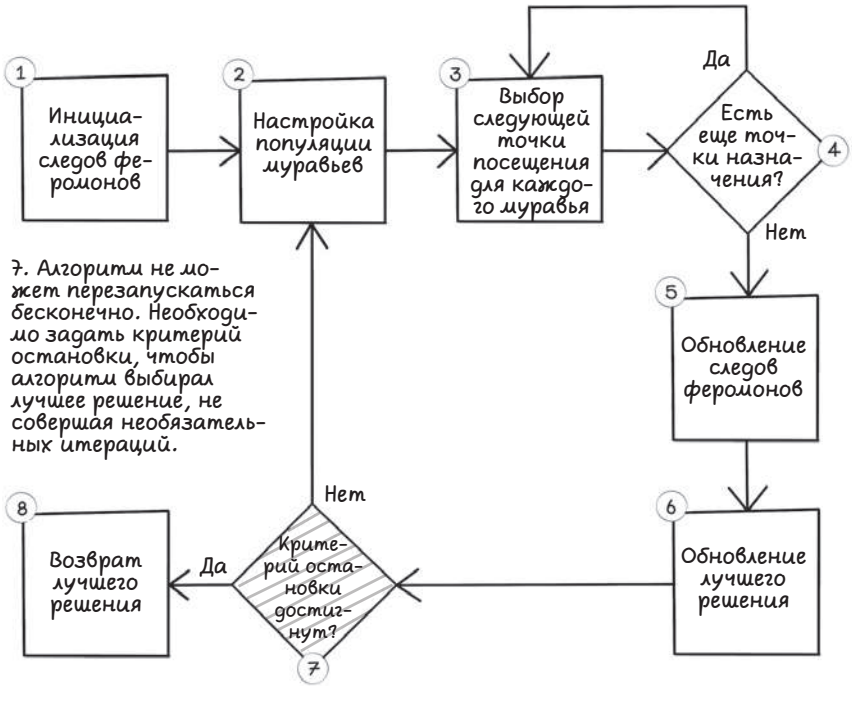


Рис. 6.28. Достигнуто ли условие остановки?

Критерий остановки муравьиного алгоритма может различаться в зависимости от предметной области решаемой задачи. В некоторых случаях известны его реалистичные пределы. Если же они не известны, то возможны следующие опции:

- *Остановка по достижении заданного числа итераций.* В этом сценарии задается общее число итераций выполнения алгоритма. Если установлено значение 100, то каждый муравей сделает 100 обходов, после чего алгоритм завершится.
- *Остановка при стагнации лучшего решения.* В этом сценарии после каждой итерации решение сравнивается с лучшим решением из предыдущей итерации. Если текущее решение не улучшается в течение заданного количества итераций, алгоритм останавливается. Например, если 20-е повторение дало результат со значением фитнес-функции 100 и этот результат продолжает сохраняться вплоть до 30-го повторения, то, скорее всего (но не обязательно), лучшего решения не существует.

Псевдокод

Функция `solve` связывает все воедино и должна дать вам лучшее представление последовательности операций и всего цикла алгоритма. Обратите внимание, что алгоритм выполняется в течение заданного числа итераций. При этом в начале каждой итерации колония устанавливается в исходное состояние, а по завершении выбирается лучший муравей:

```
solve(total_iterations, evaporation_rate, number_of_ants_factor,
      attraction_count):
  let pheromone_trails equal setup_pheromones()
  let best_ant equal Nothing
  for i in range(0, total_iterations):
    let ant_colony equal setup_ants(number_of_ants_factor)
    for r in range(0, attraction_count - 1):
      move_ants(ant_colony)
    update_pheromones(evaporation_rate,
                      pheromone_trails,
                      attraction_count)
    let best_ant equal get_best(ant_colony)
```

С помощью изменения некоторых параметров можно настраивать соотношение выполняемой алгоритмом разведки и эксплуатации. Эти параметры влияют на то, как долго алгоритм будет искать хорошее решение. Разведке в этом случае способствует элемент случайности. Изменение весового баланса между эвристиками и феромонами влияет на то, применяют ли муравьи жадный поиск (при перевесе в сторону эвристики) или же больше доверяют феромонам. Скорость испарения также влияет на этот баланс. Количество муравьев и общее число итераций алгоритма определяют качество найденного решения. Чем больше муравьев и итераций задействуется, тем выше вычислительные затраты. В зависимости от решаемой задачи время вычисления может влиять на эти параметры (рис. 6.29):

```
Установить вероятность выбора муравьями случайного аттракциона
для посещения (0.0-1.0) (0%-100%).
RANDOM_ATTRACTION_FACTOR = 0.3

Установить вес феромонов на пути для выбора муравьями.
ALPHA = 4

Установить вес эвристики пути для выбора муравьями.
BETA = 7

Установить процент муравьев в колонии на основе общего количества
аттракционов.
NUMBER_OF_ANTS_FACTOR = 0.5

Установить, какое количество путей должны пройти муравьи.
TOTAL_ITERATIONS = 1000

Установить коэффициент испарения феромонов (0.0-1.0) (0%-100%).
EVAPORATION_RATE = 0.4
```

Рис. 6.29. Настраиваемые параметры муравьиного алгоритма

Теперь вы имеете представление о работе муравьиного алгоритма и о том, как его можно использовать для решения задачи с парком развлечений. Следующий раздел описывает ряд других сценариев его применения. Возможно, эти примеры помогут вам найти подходящие варианты использования этого алгоритма в своей работе.

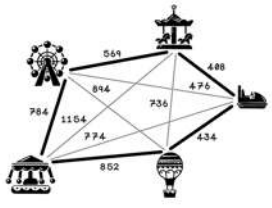
Применение муравьиного алгоритма

Муравьиные алгоритмы оптимизации очень гибкие и применяются во многих сферах деятельности, как правило, для решения комплексных задач по оптимизации, включая следующие:

- *Оптимизация маршрутов*. Задачи маршрутизации обычно подразумевают несколько пунктов назначения, которые требуется посетить с учетом ряда факторов. В примере с логистикой к важным факторам могут относиться расстояние между пунктами назначения, дорожные условия, вид доставляемых грузов и время суток. Все эти факторы следует принимать в расчет, чтобы настроить процесс доставки оптимальным для бизнеса образом. Как раз здесь и можно использовать муравьиный алгоритм. Данная задача похожа на задачу с парком развлечений, но ее эвристическая функция будет уже более сложна и ориентирована на соответствующий контекст.
- *Составление рабочего графика*. Составление графиков актуально практически для любой индустрии. Правильная организация смен медсестер важна для предоставления качественных медицинских услуг. Вычислительные задачи на серверах также должны иметь оптимальное расписание выполнения, чтобы максимально повысить эффективность оборудования. Все эти и многие другие задачи можно решать с помощью муравьиного алгоритма. Вместо посещения объектов рассматривается посещение муравьями задач в определенных последовательностях. При этом эвристическая функция включает ограничения (факторы) и соответствующие контексту задач правила. Например, медсестрам нужны выходные для отдыха, а на серверах предпочтения должны отдаваться задачам с высоким приоритетом.
- *Обработка изображений*. Муравьиный алгоритм можно использовать для обнаружения границ при обработке изображений. Изображение состоит из ряда смежных пикселей, и муравьи перемещаются от пикселя к пикселю, оставляя следы феромонов. При этом чем выше интенсивность цвета пикселя, тем более выраженный след оставляют муравьи, в результате чего вдоль границ объектов феромонов остается больше всего. По сути, алгоритм очерчивает контуры изображения. Иногда для такой процедуры предварительно требуется перевести изображение в оттенки серого, чтобы значения цветов пикселей можно было сравнивать более унифицированно.

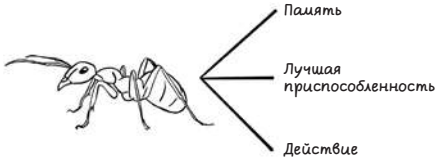
Краткий обзор главы «Роевой интеллект: муравьи»

В муравьиных алгоритмах используются феромоны и эвристика.

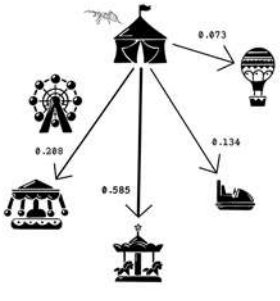


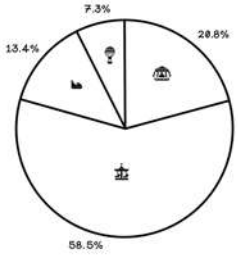
Муравьиные алгоритмы эффективны при решении таких задач оптимизации, как нахождение кратчайшего пути или составление расписаний.

Муравьи обладают памятью и производительностью и могут совершать действия.

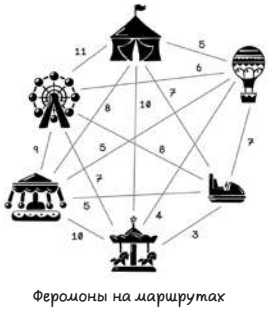


При расчете вероятности выбора используются веса эвристики и феромонов.

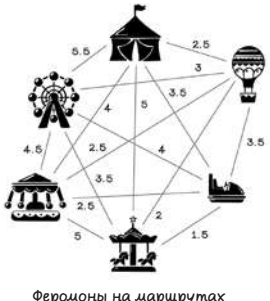




Каждый муравей оставляет феромоны пропорционально своей производительности. Феромоны могут испаряться.



Феромоны на маршрутах



Феромоны на маршрутах после 50 %-ного испарения

7 Роевой интеллект: частицы

В этой главе

- ✓ Природа алгоритмов оптимизации роем частиц
- ✓ Знакомство с задачами по оптимизации и их решение
- ✓ Разработка и реализация алгоритма оптимизации роем частиц

Что такое оптимизация роем частиц?

Метод роя частиц — это еще один роевой алгоритм. Роевой интеллект подразумевает эмерджентное поведение группы особей с целью коллективного решения задач. В главе 6 мы видели, как муравьи находят кратчайшие пути между точками назначения с помощью феромонов.

Еще одним отличным примером роевого интеллекта в природе являются стаи птиц. Во время полета отдельная птица применяет различные маневры и техники с целью сохранения энергии, например подъем вверх и плавное парение по воздуху либо использование попутных потоков ветра, помогающих перемещаться в нужном направлении. Это поведение указывает на наличие у отдельной особи примитивного интеллекта. Но, как мы знаем, многим птицам также приходится мигрировать при смене сезонов года, поскольку зимой становится меньше насекомых и другой пищи, а также ухудшаются условия для гнездования. В связи с этим птицы образуют стаи для перелета в более теплые регионы, где повышается их шанс на выживание. Как правило, такая миграция очень длительна. Приходится преодолевать тысячи километров, чтобы в конечном счете попасть в места с пригодными условиями обитания. В таких случаях птицы сбиваются в стаи, что дает им больше возможностей противостоять хищникам, а также позволяет экономить энергию. При этом сохранение определенной формации стаи несет в себе ряд общих преимуществ. Например, крупная сильная особь занимает лидирующее положение, и взмахами своих крыльев она генерирует воздушную подъемную силу для летящих сзади, что позволяет им затрачивать намного меньше энергии. Лидер стаи может сменяться при смене направления, или если он устает. Когда отдельная птица выбивается из стаи, ей становится сложнее справляться с повышенным сопротивлением ветра, и она корректирует свое движение для возвращения в стаю. На рис. 7.1 показана формация стаи птиц. Вы наверняка видели ее.

Рис. 7.1. Пример формации стаи птиц

В 1987 году Крэйг Рейнольдс (Craig Reynolds) разработал программу-симулятор для лучшего понимания признаков эмерджентного поведения в стаях птиц, использовав для управления движением группы перечисленные ниже правила. Эти правила были сформулированы на основе наблюдения за реальными стаями птиц:

- *Равнение* — особь должна держать курс в соответствии с курсом соседей, чтобы вся группа перемещалась в одном направлении.
- *Связность* — особь должна придерживаться средней позиции между своими соседями, чтобы сохранять формацию группы.
- *Разделение* — особь должна избегать скучивания или столкновения с соседями, чтобы группа не нарушалась.

При симуляции роевого поведения используются и другие варианты правил. На рис. 7.2 показано поведение особи в различных сценариях, а также направление, в котором она должна перемещаться, подчиняясь соответствующему правилу. Изменение направления движения отражает баланс трех проиллюстрированных принципов.

Рис. 7.2. Правила, координирующие рой

При оптимизации роем частиц задействуется группа особей, которые находятся в разных точках области решений и на основе роевых принципов из реальной жизни стремятся найти среди возможных решений оптимальное. В этой главе рассматривается внутреннее устройство алгоритма роя частиц, а также приводятся примеры его использования для решения прикладных задач. Представьте себе рой пчел, которые распределяются по поляне цветов и постепенно сосредотачиваются в области с наибольшей плотностью цветов. Чем больше пчел находит это скопление цветов, тем больше других пчел к нему стремится. В своей сути этот пример демонстрирует принцип действия алгоритма оптимизации роем частиц (рис. 7.3).

Задачи оптимизации уже упоминались в нескольких главах. Поиск оптимального пути в лабиринте, выбор оптимальных предметов для рюкзака, а также кратчайшего пути между аттракционами в парке развлечений — все это примеры задач оптимизации. Но мы проработали их, не погружаясь в детали. Тем не менее, начиная с этой главы, мы будем разбирать их более подробно. Следующий раздел научит вас легче распознавать среди всех задач именно задачи оптимизации.

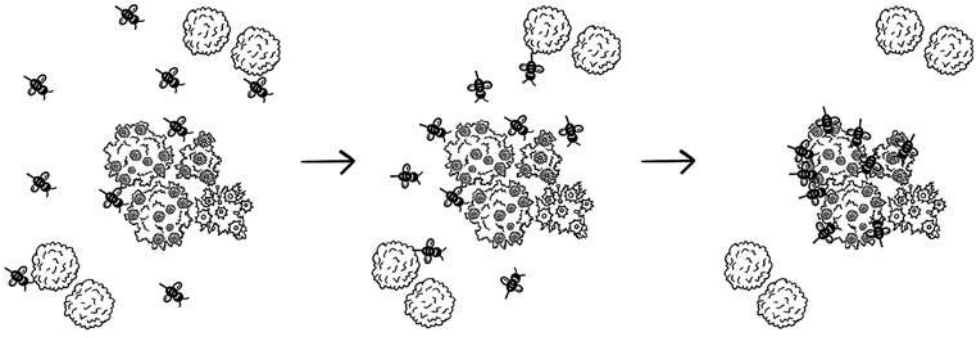


Рис. 7.3. Рой пчел, сходящихся к общей цели

Задачи оптимизации с технической точки зрения

Предположим, что у нас есть несколько перцев разного размера. Как правило, маленькие перцы оказываются более жгучими, чем крупные. Если нарисовать их на графике, распределив согласно размеру и жгучести, то получится рис. 7.4.

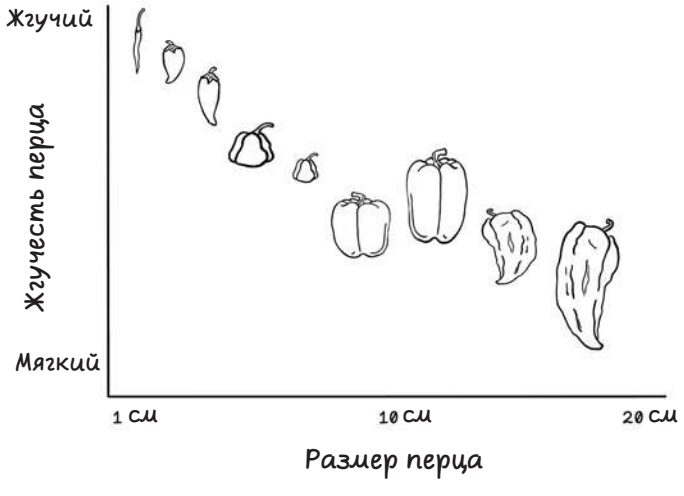


Рис. 7.4. Жгучесть и размер перца

Рисунок отражает размер и степень жгучести каждого перца. Заменим изображения перцев точками данных и нарисуем между ними возможную кривую, получив рис. 7.5. Если бы перцев у нас было больше, то и точек получилось бы больше, а кривая выглядела бы более точной.

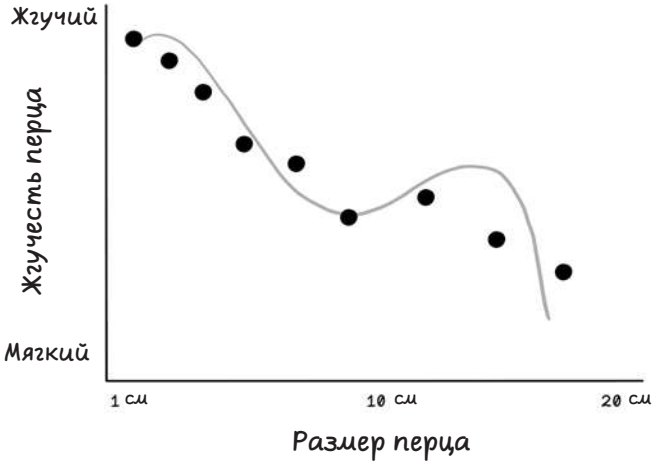


Рис. 7.5. Изменение жгучести и размера перца

Этот пример мог бы стать задачей оптимизации. Например, мы ищем минимум слева направо и постепенно встречаем точки, каждая из которых оказывается ниже предыдущей, но при этом посередине встречается точка, находящаяся выше. Нужно ли нам остановиться? Если мы остановимся, то можем упустить реальный минимум, который представлен последней точкой, называемой *глобальным минимумом*.

Эту аппроксимированную кривую/линию тренда можно выразить функцией, как показано на рис. 7.6. Данную функцию можно интерпретировать как жгучесть перца, равную результату функции при условии, что x — это размер перца.

$$f(x) = -(x - 4)(x - 0.2)(x - 2)(x - 3) + 5$$

Рис. 7.6. Пример функции для определения жгучести перца относительно его размера

В реальных же задачах обычно присутствуют тысячи точек данных, и получить минимум функции становится не так просто, как в этом примере. В таких случаях пространство поиска огромно и найти решение вручную невозможно.

Обратите внимание, что для создания точек данных мы использовали только два свойства перца, в результате чего получилась простая кривая. Если же добавить еще одно свойство, например цвет, то представление данных существенно изменится. Теперь график нужно рисовать в 3D, а тренд превращается из кривой в поверхность. Поверхность напоминает скомканное в трех измерениях одеяло (рис. 7.7). Представляется она также в виде функции, но уже более сложной.

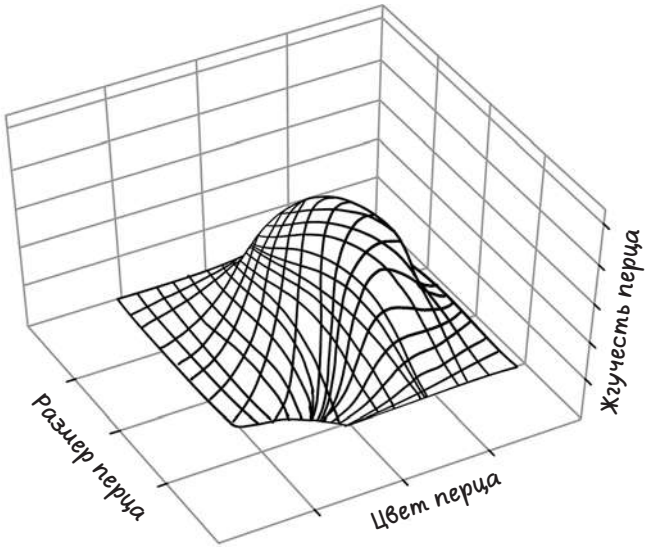


Рис. 7.7. Жгучесть, размер и цвет перца

Более того, трехмерное пространство поиска может выглядеть простым, как на рис. 7.7, либо настолько сложным, что визуально найти в нем глобальный минимум будет практически нереально (рис. 7.8).

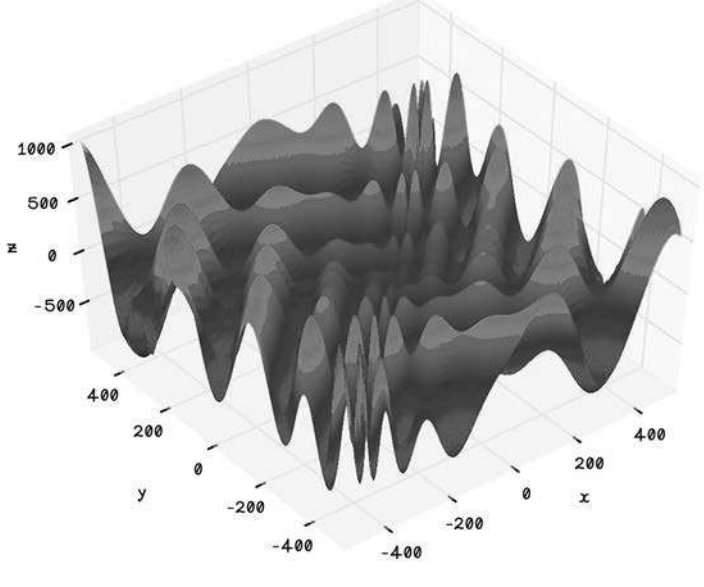


Рис. 7.8. Функция, визуализированная как поверхность в 3D-пространстве

На рис. 7.9 показана функция, представляющая эту поверхность.

$$f(x,y) = -(y+47)\sin\sqrt{\left|\frac{x}{2}+(y+47)\right|} - x\sin\sqrt{|x-(y+47)|}$$

Рис. 7.9. Функция, представляющая поверхность на рис. 7.8

Становится еще интереснее! Мы рассмотрели три свойства перца: размер, цвет и степень жгучести. В результате поиск уже происходит в трех измерениях. А что, если нам нужно добавить место, где он был выращен? Это свойство еще больше усложнит визуализацию и понимание данных, так как искать мы уже будем в четырех измерениях. А если добавить еще и возраст перца вместе с видом использованных для его выращивания удобрений, то образуется огромное шестимерное пространство, и представить, как будет выглядеть поиск по нему, мы уже не сможем. Он будет также выражен в виде функции, но, опять же, слишком сложной для решения человеком.

С решением сложных задач оптимизации особенно хорошо справляются алгоритмы роя частиц. В них частицы распределяются по многомерному пространству поиска и работают сообща для поиска хороших максимумов или минимумов.

В частности, эти алгоритмы эффективны в следующих сценариях:

- *Большие пространства поиска*. Когда присутствует множество точек данных и возможных комбинаций.
- *Пространства поиска с большим количеством измерений*. Для нахождения хорошего решения требуется учесть много измерений.

УПРАЖНЕНИЕ: СКОЛЬКО ИЗМЕРЕНИЙ БУДЕТ ИМЕТЬ ПРОСТРАНСТВО ПОИСКА ДЛЯ СЛЕДУЮЩЕГО СЦЕНАРИЯ?

Клиент агентства недвижимости, который не любит холодный климат, просит определить подходящий для проживания город на основе средней минимальной годовой температуры. Также важно, чтобы в городе проживало не более 700 000 человек, потому что большие скопления людей вызывают неудобства. При этом средняя стоимость жилья должна быть минимально возможной, а вот количество поездов как можно большим.

ОТВЕТ: КОЛИЧЕСТВО ИЗМЕРЕНИЙ В ПРОСТРАНСТВЕ ПОИСКА

Задача в этом сценарии имеет пять измерений:

- средняя температура;
- численность населения;
- средняя стоимость жилья;
- количество поездов;
- результат всех этих свойств, который будет определять решение.

Применение оптимизации роем частиц

Представьте, что мы разрабатываем дрон; для изготовления его корпуса и пропеллера требуется два вида материалов. С помощью ряда исследований мы обнаружили, что разное соотношение этих материалов дает разную эффективность дрона в плане стабильности взлета и сопротивления сильному ветру. Речь идет об алюминии для шасси и пластике для лопастей. Их чрезмерное или недостаточное количество негативно сказывается на качестве готового дрона. При этом несколько комбинаций дают хороший результат и только одна — превосходный.

На рис. 7.10 показано, какие компоненты изготавливаются из пластика, а какие из алюминия. Стрелки отражают влияющие на качество дрона силы. Коротко говоря, нам нужно найти хорошее соотношение пластика и алюминия для создания такого дрона, который меньше подвержен отклонению при взлете и колебанию на ветру. Пластик и алюминий выступают в качестве входных данных, а на выходе получается стабильность дрона. Опишем идеальную стабильность как снижение отклонения при взлете и колебаний от ветра.

Рис. 7.10. Пример оптимизации дрона

Здесь будет важна точность соотношения пластика и алюминия, причем диапазон возможностей очень велик. Исследователи разработали для данного сценария функцию вычисления этого соотношения. Ее мы используем в симулированной виртуальной среде, которая будет тестировать итоговое отклонение и колебание для нахождения наилучшего количества каждого материала перед изготовлением очередного прототипа дрона. Нам также известно, что максимальное и минимальное количество каждого материала равны 10 и –10 соответственно. Функция приспособленности будет похожа на эвристическую.

Данная функция приводится на рис. 7.11. Результатом ее служит показатель качества дрона на основе отклонения при взлете и колебаний от ветра с учетом входных значений x и y.

$$f(x, y) = (x + 2y - 7)^2 + (2x + y - 5)^2$$

Рис. 7.11. Пример функции для оптимизации соотношения алюминия (*x*) и пластика (*y*)

Как найти необходимое количество алюминия и пластика для изготовления качественного дрона? Один из способов — это перебрать все комбинации их значений в поиске наилучшего соотношения материалов. Но если брать каждое возможное число, количество вычислений для них будет едва ли не бесконечным. Нам нужно вычислить результат для элементов из табл. 7.1. Может показаться странным, что в ней также присутствуют отрицательные значения. Не удивляйтесь: они нужны для демонстрации функции приспособленности, используемой для оптимизации этих значений.

Таблица 7.1. Возможные значения для комбинаций алюминия и пластика

Сколько частей из алюминия? (*x*)	**Сколько частей из пластика? (*y*)**
–0.1	1.34
–0.134	0.575
–1.1	0.24
–1.1645	1.432
–2.034	–0.65
–2.12	–0.874
0.743	–1.1645
0.3623	–1.87
1.75	–2.7756
…	…
–10 ≥ Алюминий ≥ 10	*–10 ≥ Пластик ≥ 10*

Это вычисление будет продолжаться для каждого возможного числа в заданных границах и окажется весьма дорогостоящим, поэтому решать эту задачу методом брутфорса практически нереально. Здесь требуется другой подход.

Благодаря алгоритму роя частиц можно вести поиск в большой области без необходимости проверять каждое значение в каждом измерении. В задаче с дроном алюминий представляет одно измерение, пластик — второе, а собираемый из них дрон — третье.

В следующем разделе мы определим структуры данных, необходимые для представления частицы, включая информацию о задаче, которую она будет в себе нести.

Представление состояния: как выглядят частицы?

Поскольку мы будем рассматривать перемещение по области поиска частиц, сначала нужно определить их свойства (рис. 7.12).

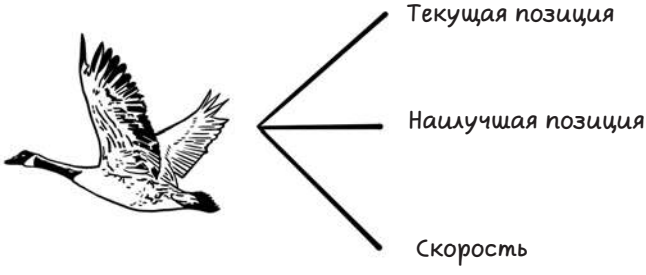


Рис. 7.12. Свойства частицы

Частицы будут иметь три свойства:

- *Текущая позиция* — положение частицы во всех измерениях.
- *Наилучшая позиция* — наилучшее положение, найденное с помощью функции приспособленности.
- *Скорость* — текущая скорость перемещения частицы.

Псевдокод

Для реализации этих трех характеристик в конструкторе частицы потребуется использовать приведенные ниже свойства, что позволит алгоритму выполнять различные связанные с ней операции. Пока не обращайте внимания на инерцию, когнитивный компонент и социальный. Мы поговорим о них позже.

```
Particle(x, y, inertia, cognitive_constant, social_constant):
  let particle.x equal to x
  let particle.y equal to y
  let particle.fitness equal to infinity
  let particle.velocity equal to 0
  let particle.best_x equal to x
  let particle.best_y equal to y
  let particle.best_fitness equal to infinity
  let particle.inertia equal to inertia
  let particle.cognitive_constant equal to cognitive_constant
  let particle.social_constant equal to social_constant
```

Жизненный цикл алгоритма роя частиц

Подход к проектированию алгоритма роя частиц основывается на тематике решаемой задачи. У каждой задачи есть свой уникальный контекст и область данных. Решения для различных задач также оцениваются по-разному. Далее мы рассмотрим вариант проектирования алгоритма для конкретной задачи сборки дрона.

Общий жизненный цикл алгоритма будет выглядеть так (рис. 7.13):

1. *Инициализация популяции частиц.* Определение количества используемых частиц и инициализация каждой из них в случайно выбранном месте позиции в области поиска.
2. *Вычисление приспособленности каждой частицы.* С учетом положения каждой частицы определяется ее приспособленность.
3. *Обновление позиции каждой частицы.* Повторяющееся обновление расположения всех частиц по принципам роевого интеллекта. Частицы будут исследовать область поиска и в итоге сходиться к хорошим решениям.
4. *Определение критерия остановки.* Определение того, когда должно прекращаться обновление положения частиц и выполнение алгоритма.

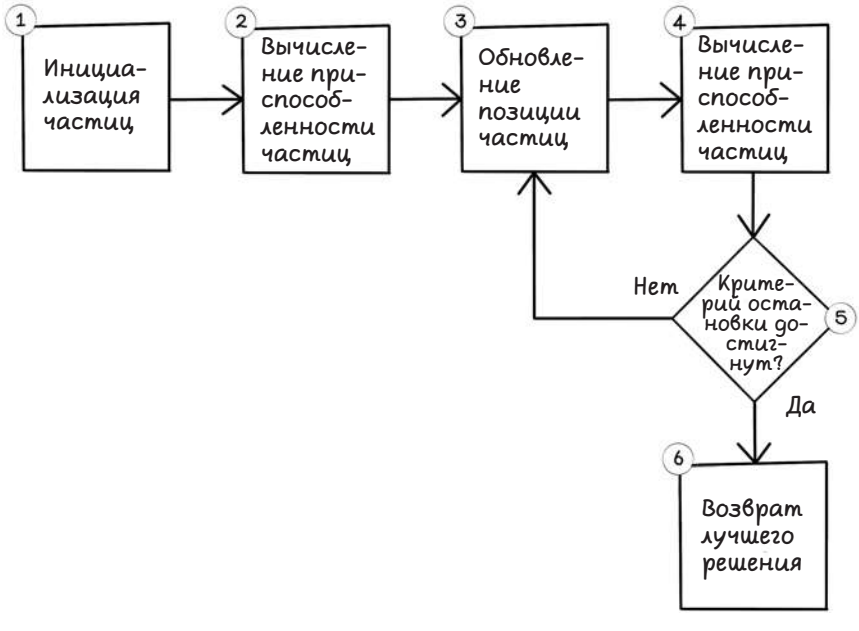


Рис. 7.13. Жизненный цикл алгоритма роя частиц

В целом алгоритм роя частиц достаточно прост, сложность представляет лишь его третий этап. Далее рассмотрим каждый этап в отдельности и подробно разберем детали.

Инициализация популяции частиц

Алгоритм начинается с создания определенного количества частиц, которое будет сохраняться на протяжении всего жизненного цикла (рис. 7.14).

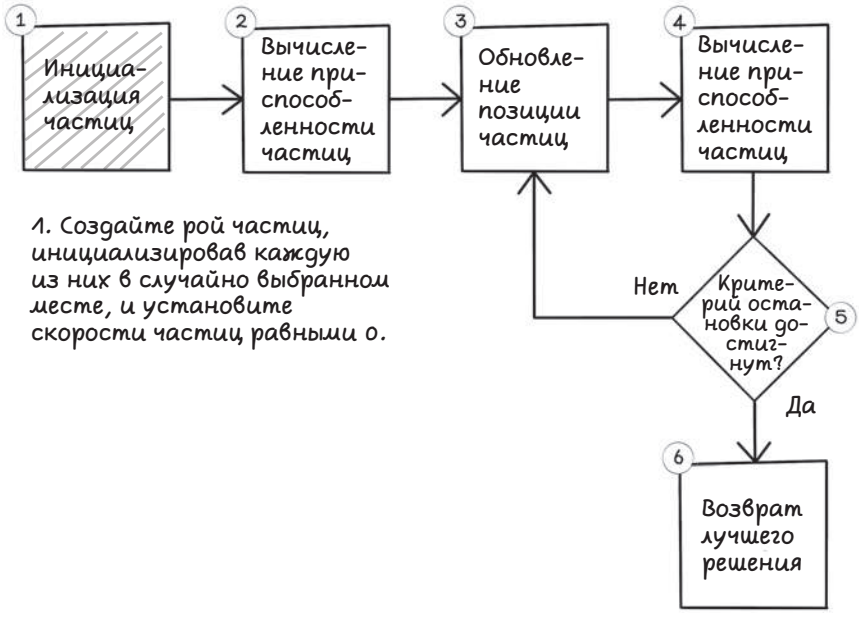


Рис. 7.14. Создание частиц

В процессе инициализации частиц необходимо учитывать три важных фактора (рис. 7.15):

- *Количество частиц.* Количество частиц влияет на сложность вычислений. Чем их больше, тем выше вычислительные затраты. Помимо этого, увеличение количества частиц будет означать более длительный процесс схождения к глобальному лучшему решению, потому что больше частиц будет склоняться к локальным лучшим. Ограничения задачи также влияют на количество частиц. Например, в обширном пространстве поиска для нахождения решения потребуется больше частиц. В зависимости от контекста количество может варьировать от всего лишь 4 до 1000. Как правило, чтобы найти хорошее решение без лишних вычислительных затрат, достаточно 50–100 частиц.
- *Начальное положение каждой частицы.* Начальным положением частицы должна быть случайная точка относительно всех измерений пространства поиска. Важно, чтобы частицы распределялись по всей области равномерно. Если же большинство частиц окажется в одной зоне, то им будет сложно найти решения, выходящие за ее пределы.

- *Начальная скорость каждой частицы.* Изначально скорость частиц устанавливается равной 0, так как влияние на них еще не оказывается. В качестве наглядной аналогии можно привести птиц, которые перед началом полета находятся неподвижно на земле.

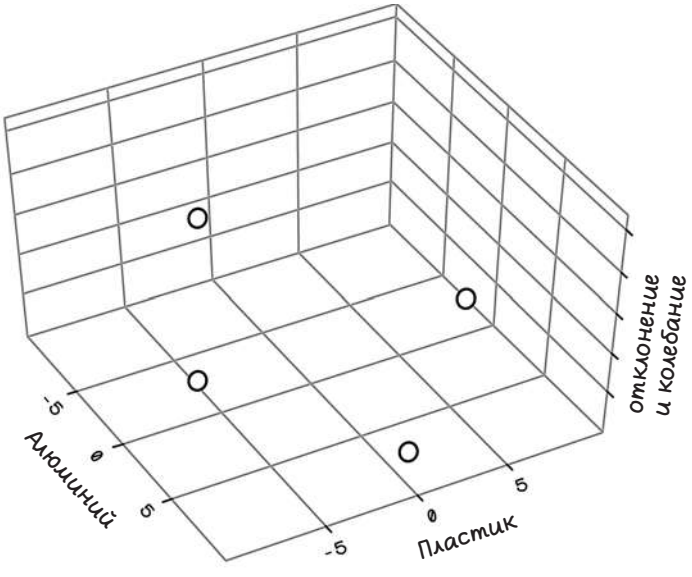


Рис. 7.15. Визуализация начального положения частиц на 3D-поверхности

В табл. 7.2 приводятся данные каждой частицы на этапе инициализации алгоритма. Обратите внимание, что скорость равна 0. Текущее и лучшее фитнес-значения также равны 0, поскольку они еще не вычислялись.

Таблица 7.2. Атрибуты данных для каждой частицы

Частица	Скорость	Текущее количество алюминия (*x*)	Текущее количество пластика (*y*)	Текущее значение фитнес-функции	Лучшее количество алюминия (*x*)	Лучшее количество пластика (*y*)	Лучшее значение фитнес-функции
1	0	7	1	0	7	1	0
2	0	–1	9	0	–1	9	0
3	0	–10	1	0	–10	1	0
4	0	–2	–5	0	–2	–5	0

Псевдокод

Метод генерации роя включает создание пустого списка и добавление в него частиц. Ключевые задачи:

- Обеспечить возможность настройки количества частиц.
- Обеспечить единообразие генерации случайных чисел. Числа распределяются по пространству поиска в рамках ограничений. Эта реализация зависит от свойств используемого генератора случайных чисел.
- Обеспечить рамки ограничений пространства поиска: в данном случае –10 и 10 для значений x и y частицы.

```
generate_swarm(number_of_particles):
  let particles equal an empty list
  for particle in range(number_of_particles):
    append Particle(random(-10, 10), random(-10, 10), INERTIA,
                    COGNITIVE_CONSTANT, SOCIAL_CONSTANT) to particles
  return particles
```

Вычисление значения фитнес-функции частицы

Далее идет вычисление значения фитнес-функции каждой частицы в ее текущем положении. Этот показатель вычисляется после каждой смены положения роя (рис. 7.16).

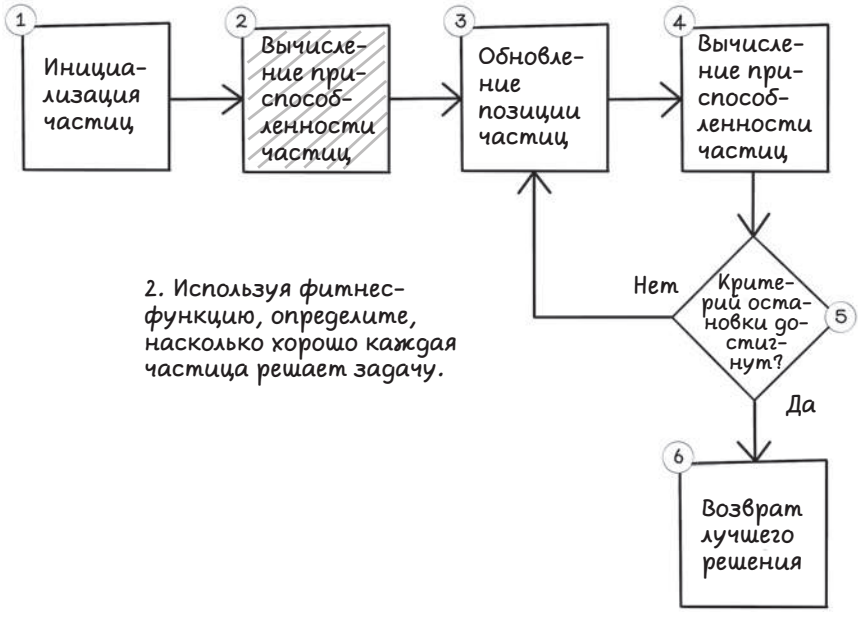


Рис. 7.16. Вычисление значения фитнес-функции частиц

Для сценария с дроном ученые предложили функцию, дающую в качестве результата показатель отклонения и колебания с учетом конкретного количества алюминия и пластика. В текущем примере алгоритма роя частиц эта функция используется в качестве фитнес-функции (рис. 7.17).

$$f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2$$

Рис. 7.17. Пример функции для оптимизации соотношения алюминия (*x*) и пластика (*y*)

С учетом того, что *x* представляет алюминий, а y — пластик, можно вычислить значение фитнес-функции каждой частицы, как показано на рис. 7.18.

$$f(7,1) = (7 + 2(1) - 7)^2 + (2(7) + 1 - 5)^2 = 104$$

$$f(-1,9) = (-1 + 2(9) - 7)^2 + (2(-1) + 9 - 5)^2 = 104$$

$$f(-10,1) = (-10 + 2(1) - 7)^2 + (2(-10) + 1 - 5)^2 = 801$$

$$f(-2,-5) = (-2 + 2(-5) - 7)^2 + (2(-2) - 5 - 5)^2 = 557$$

Рис. 7.18. Вычисление значения фитнес-функции каждой частицы

Теперь таблица частиц содержит фитнес-значения для каждой частицы (табл. 7.3). Эти значения также установлены в качестве лучших, поскольку на данный момент являются единственными известными. После первой итерации в качестве лучшей приспособленности каждой частицы будет выбираться лучшая приспособленность за все итерации этой частицы.

Таблица 7.3. Атрибуты данных для каждой частицы

Частица	Скорость	Текущее количество алюминия (*x*)	Текущее количество пластика (*y*)	Текущее значение фитнес-функции	Лучшее количество алюминия (*x*)	Лучшее количество пластика (*y*)	Лучшее значение фитнес-функции
1	0	7	1	296	7	1	296
2	0	–1	9	104	–1	9	104
3	0	–10	1	80	–10	1	80
4	0	–2	–5	365	–2	–5	365

УПРАЖНЕНИЕ: РАССЧИТАЙТЕ ЗНАЧЕНИЕ ФИТНЕС-ФУНКЦИИ ДРОНА СО СЛЕДУЮЩИМИ ИСХОДНЫМИ ДАННЫМИ

$$f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2$$

Частица	Скорость	Текущее количество алюминия (*x*)	Текущее количество пластика (*y*)	Текущее фитнес-значение	Лучшее количество алюминия (*x*)	Лучшее количество пластика (*y*)	Лучшее фитнес-значение
1	0	5	–3	0	5	–3	0
2	0	–6	–1	0	–6	–1	0
3	0	7	3	0	7	3	0
4	0	-1	9	0	–1	9	0

ОТВЕТ: ЗНАЧЕНИЕ ФИТНЕС-ФУНКЦИИ ДРОНА

$$f(5,-3) = (5 + 2(-3) - 7)^2 + (2(5) - 3 - 5)^2 = 68$$

$$f(-6,-1) = (-6 + 2(-1) - 7)^2 + (2(-6) - 1 - 5)^2 = 549$$

$$f(7,3) = (7 + 2(3) - 7)^2 + (2(7) + 3 - 5)^2 = 180$$

$$f(-1,9) = (-1 + 2(9) - 7)^2 + (2(-1) + 9 - 5)^2 = 104$$

Псевдокод

Фитнес-функция (функция приспособленности) представляется в коде математической функции. Все необходимые операции, например возведение в степень или извлечение квадратного корня, имеются в любой математической библиотеке:

```
calculate_fitness(x, y):
  return power(x + 2 * y - 7, 2) + power(2 * x + y - 5, 2)
```

Функция для обновления приспособленности частицы также весьма проста. Она определяет, превосходит ли только что вычисленное значение известное лучшее, и сохраняет полученную информацию:

```
update_fitness(x, y):
  let particle.fitness equal the result of calculate_fitness(x, y)
  if particle.fitness is less than particle.best_fitness:
    let particle.best_fitness equal particle.fitness
    let particle.best_x equal x
    let particle.best_y equal y
```

Функция для определения лучшей частицы в рое выполняет перебор всех частиц, обновляет их значение на основе новых позиций и находит ту частицу, значение которой для функции приспособленности наименьшее. В этом случае мы стремимся к минимизации, поэтому выбирается меньшее значение:

```
get_best(swarm):
  let best_fitness equal infinity
  let best_particle equal nothing
  for particle in swarm:
    update fitness of particle
    if particle.fitness is less than best_fitness:
      let best_fitness equal particle.fitness
      let best_particle equal particle
  return best_particle
```

Обновление положения каждой частицы

Шаг обновления самый сложный, поскольку именно здесь работает основной механизм алгоритма. На этом этапе природные свойства роевого интеллекта преобразуются в математическую модель, которая позволяет улучшать хорошие решения в процессе исследования пространства поиска (рис. 7.19).

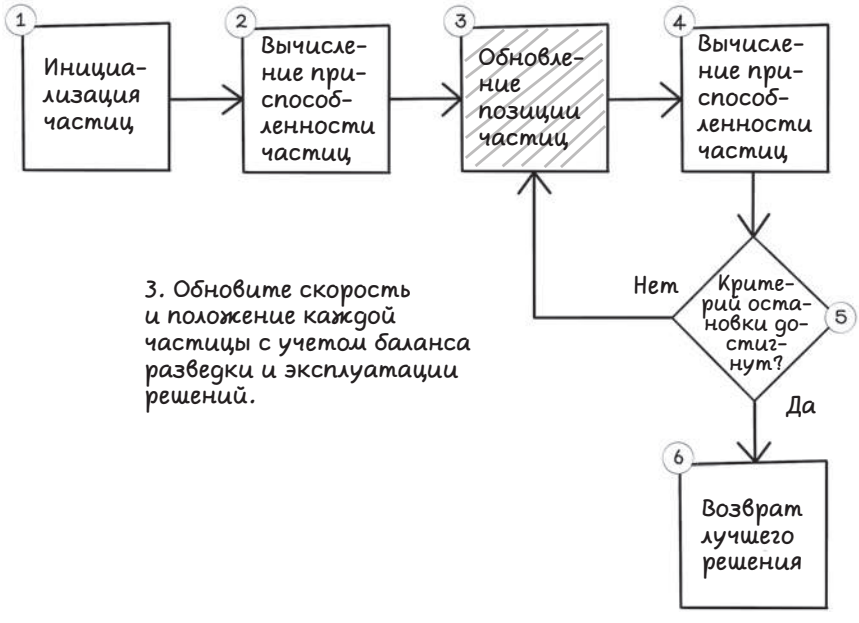


Рис. 7.19. Обновление положений частиц

Частицы роя обновляют свое положение с учетом когнитивной способности и факторов окружающей среды, таких как инерция и действия роя. Эти факторы влияют на скорость и положение каждой частицы. Первым шагом будет понимание принципа обновления скорости. Скорость также определяет направление движения частицы.

Частицы роя движутся к различным точкам пространства поиска для нахождения лучших решений. Каждая из них обращается к собственной памяти о хорошем решении, а также к знаниям роя о лучшем решении. На рис. 7.20 проиллюстрировано перемещение частиц роя по мере обновления их позиций.

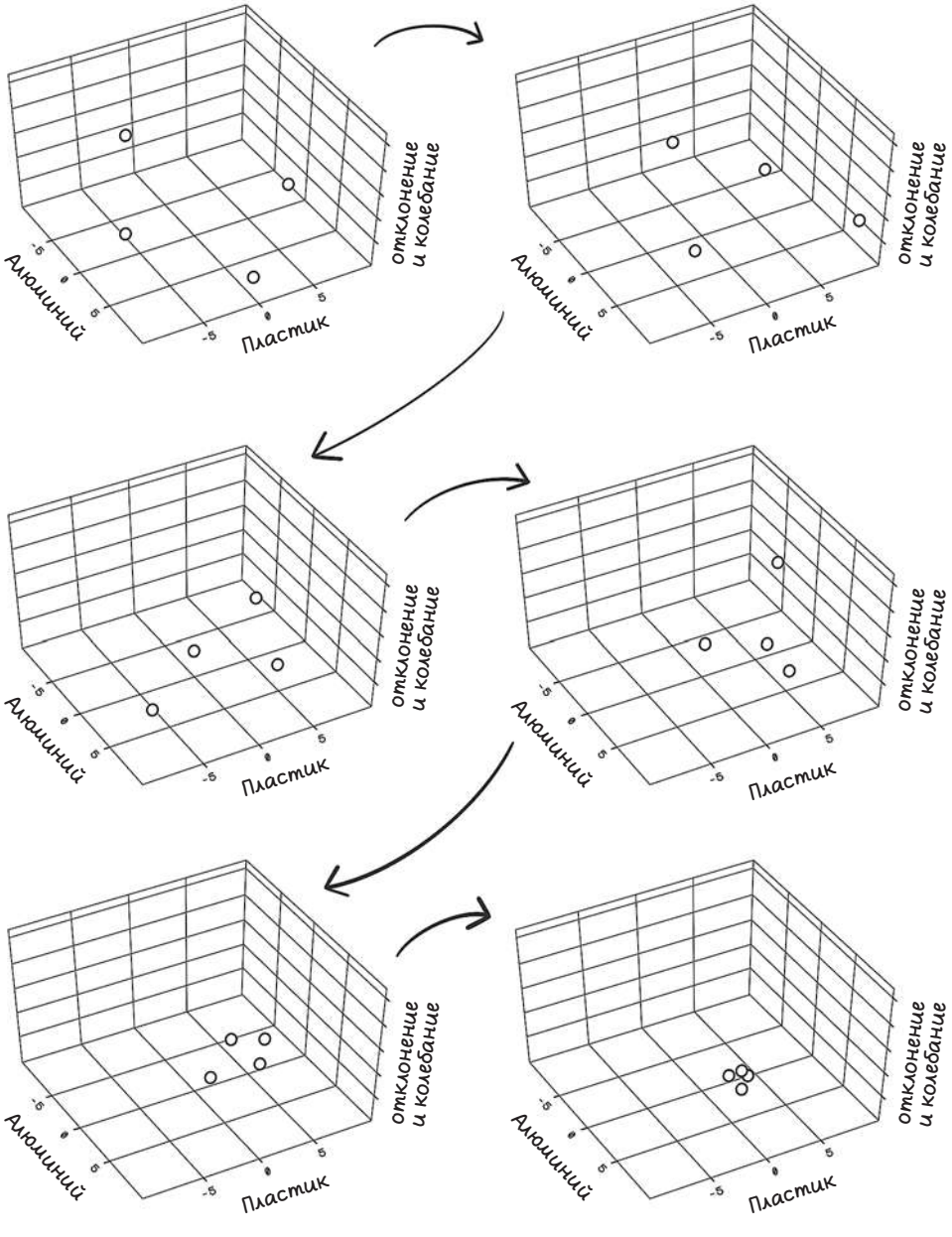


Рис. 7.20. Перемещение частиц в течение пяти итераций

Составляющие обновления скорости

Для вычисления новой скорости каждой частицы используются три составляющие: инерция, когнитивная составляющая и социальная составляющая. Каждая из них влияет на движение частицы. Сначала мы рассмотрим их по отдельности, после чего перейдем к их взаимодействию для обновления скорости и в конечном итоге положения частицы:

- *Инерция* — сопротивление движению или изменению направления конкретной частицы, которое влияет на ее скорость. Складывается она из двух значений: величины инерции и текущей скорости частицы. Итоговое значение инерции выражается в диапазоне от 0 до 1.

```
Инерциальная составляющая:
инерция * текущая скорость
```

  - Коэффициент инерции, близкий к 0, приводит к менее обширному исследованию, потенциально занимающему больше итераций.
  - Коэффициент ближе к 1 приводит к более обширному исследованию с меньшим числом итераций.

- *Когнитивная составляющая* — внутренняя когнитивная способность отдельной частицы. Когнитивная способность — это понимание частицы, знающей свое лучшее положение и использующей это знание при выборе направления движения. Когнитивная константа — это число большее 0 и меньшее 2. Чем ее значение больше, тем выше степень эксплуатации решений частицами.

Когнитивная составляющая:

```
когнитивное ускорение * (лучшее положение частицы – текущее положение)

  когнитивное ускорение = когнитивная константа * случайное когнитивное
                                                  число
```

- *Социальная составляющая* — способность частицы взаимодействовать с роем. Частица узнает лучшее коллективное решение и учитывает эту информацию при расчете перемещения. Социальное ускорение определяется путем умножения константы на случайное число. Социальная константа остается одинаковой весь жизненный цикл, а случайный множитель вводится для изменения степени предпочтения социальной составляющей при перемещении.

Социальная составляющая:

```
социальное ускорение * (лучшее положение роя – текущее положение)

  социальное ускорение = социальная константа * случайное социальное
                                                 число
```

Чем больше социальная константа, тем выше степень исследования, так как влияние социального компонента на перемещение частицы возрастает. Значение социальной константы меняется от 0 до 2. Чем выше это значение, тем активнее исследование.

Обновление скорости

После изучения инерции, а также когнитивного и социального компонентов можно рассмотреть их взаимосвязь при обновлении скорости частиц (рис. 7.21).

Новая скорость:

```
инерциальная составляющая + социальная составляющая + когнитивная
                                                       составляющая
  (инерция * текущая скорость)
    (социальное ускорение * (лучшее положение роя – текущее положение))
      (когнитивное ускорение * (лучшее положение частицы – текущее
                                                          положение))
```

Рис. 7.21. Формула вычисления скорости

Рассматривая математическую интерпретацию, будет сложно понять воздействие отдельных компонентов функции на скорость частицы. Визуально же их влияние представлено на рис. 7.22.

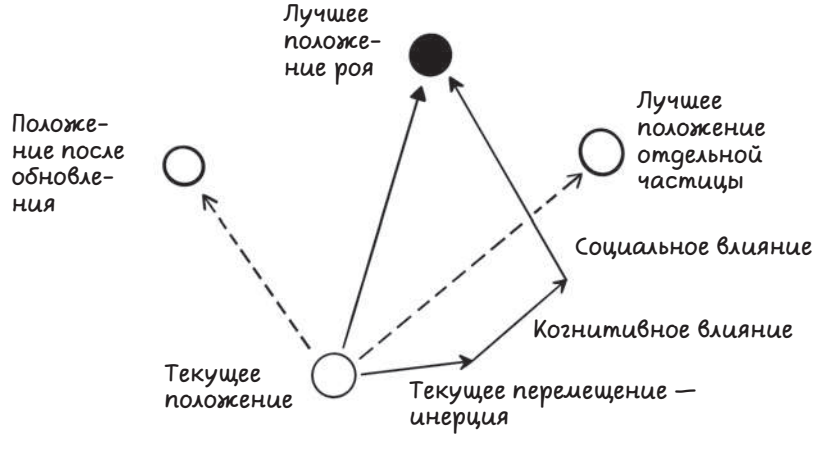


Рис. 7.22. Влияние компонентов функции на обновление скорости

В табл. 7.4 приводятся свойства каждой частицы после вычисления ее значения фитнес-функции.

Таблица 7.4. Атрибуты данных для каждой частицы

Частица	Скорость	Текущее количество алюминия	Текущее количество пластика	Текущее значение фитнес-функции	Лучшее количество алюминия	Лучшее количество пластика	Лучшее значение фитнес-функции
1	0	7	1	296	2	4	296
2	0	–1	9	104	–1	9	104
3	0	–10	1	80	–10	1	80
4	0	–2	–5	365	–2	–5	365

Далее перейдем к вычислениям обновления скорости частицы, используя приведенные выше формулы.

Для текущего сценария установлены следующие значения констант:

- *Инерция = 0.2*. Эта настройка определяет более медленное исследование.
- *Когнитивная константа = 0.35*. Поскольку она меньше, чем социальная, социальный компонент преобладает над когнитивным компонентом отдельной частицы.
- *Социальная константа = 0.45*. Так как она выше, чем когнитивная, предпочтение отдается социальному компоненту и частицы придают большее значение лучшим решениям, найденным роем.

На рис. 7.23 показаны вычисления инерции, а также когнитивного и социального компонентов для формулы обновления скорости.

Инерциальная составляющая:

```
инерция * текущая скорость
= 0.2 * 0
= 0
```

Когнитивная составляющая:

```
когнитивное ускорение = когнитивная константа * случайное когнитивное
число
= 0.35 * 0.2
= 0.07

когнитивное ускорение * (лучшее положение частицы — текущее положение)
= 0.07 * ([7,1] - [7,1])
= 0.07 * 0
= 0
```

Социальная составляющая:

```
социальное ускорение = социальная константа * случайное социальное число
= 0.45 * 0.3
= 0.135

социальное ускорение * (лучшее положение роя – текущее положение)
= 0.135 * ([-10,1] - [7,1])
= 0.135 * sqrt((-10 - 7)^2 + (1 - 1)^2)
= 0.135 * 17
= 2.295
```

Формула расчета расстояния:

$$\sqrt{((x1 - x2)^2 + (y1 - y2)^2)}$$

Новая скорость:

```
инерциальная составляющая + социальная составляющая + когнитивная
составляющая
= 0 + 0 + 2.295
= 2.295
```

Рис. 7.23. Процесс вычисления скорости частицы

По завершении вычислений для всех частиц скорость каждой из них обновляется, как показано в табл. 7.5.

Таблица 7.5. Атрибуты данных для каждой частицы

Частица	Скорость	Текущее количество алюминия	Текущее количество пластика	Текущее значение фитнес-функции	Лучшее количество алюминия	Лучшее количество пластика	Лучшее значение фитнес-функции
1	2.295	7	1	296	7	1	296
2	1.626	–1	9	104	–1	9	104
3	2.043	–10	1	80	–10	1	80
4	1.35	–2	–5	365	–2	–5	365

Обновление положения

Разобравшись, как обновлять скорость, можно перейти к обновлению текущего положения каждой частицы, используя вычисленное значение скорости (рис. 7.24).

Новое положение:

```
текущее положение + новая скорость
```

Положение:

```
текущее положение + новая скорость
= ([7,1]) + 2.295
= [9.295, 3.295]
```

Рис. 7.24. Вычисление нового положения частицы

Зная текущее положение и новую скорость, мы определяем новое положение каждой частицы и обновляем таблицу их свойств новыми значениями скоростей. Далее снова вычисляется значение фитнес-функции каждой частицы и запоминается лучшее положение (табл. 7.6).

Таблица 7.6. Атрибуты данных для каждой частицы

Частица	Скорость	Текущее количество алюминия	Текущее значение пластика	Текущее значение фитнес-функции	Лучшее количество алюминия	Лучшее количество пластика	Лучшее значение фитнес-функции
1	2.295	9.925	3.325	721.286	7	1	296
2	1.626	0.626	10	73.538	0.626	10	73.538
3	2.043	7.043	1.043	302.214	–10	1	80
4	1.35	–0.65	–3.65	179.105	–0.65	–3.65	179.105

Вычислить начальную скорость каждой частицы в первой итерации достаточно просто, потому что отсутствуют предыдущие лучшие позиции частиц и в расчет берется только лучшее положение роя, влияющее на социальный компонент.

Посмотрим, как будет выглядеть вычисление обновления скорости с новой информацией о лучшем положении каждой частицы и новом лучшем положении роя. На рис. 7.25 приводится пример вычислений для первой частицы в списке.

Инерциальная составляющая:

инерция * текущая скорость

= 0.2 * 2.295

= 0.59

Когнитивная составляющая:

когнитивное ускорение = когнитивная константа * случайное когнитивное число

= 0.35 * 0.2

= 0.07

Примечание. Случайные числа остаются без изменений исключительно для простоты понимания.

когнитивное ускорение * (лучшее положение частицы – текущее положение)

= 0.07 * ([7,1] – [9.925,3.325])

= 0.07 * $sqrt((7 - 9.925)^2 + (1 - 3.325)^2)$

= 0.07 * 3.736

= 0.266

Социальная составляющая:

социальное ускорение = социальная константа * случайное социальное число

= 0.45 * 0.3

= 0.135

социальное ускорение * (лучшее положение роя – текущее положение)

= 0.135 * ([0.626,10] – [9.925,3.325])

= 0.135 * $sqrt((0.626 - 9.925)^2 + (10 - 3.325)^2)$

= 0.135 * 11.447

= 1.545

Новая скорость:

инерциальная составляющая + социальная составляющая + когнитивная составляющая

= 0.59 + 0.266 + 1.545

= 2.401

Рис. 7.25. Процесс вычисления скорости частицы

Здесь в обновлении скорости задействованы и когнитивный, и социальный компоненты, в то время как в примере из рис. 7.23 значение имел только социальный, так как это была первая итерация.

Частицы перемещаются на протяжении нескольких итераций. Рисунок 7.26 демонстрирует их передвижение и итоговую сходимость к решению.

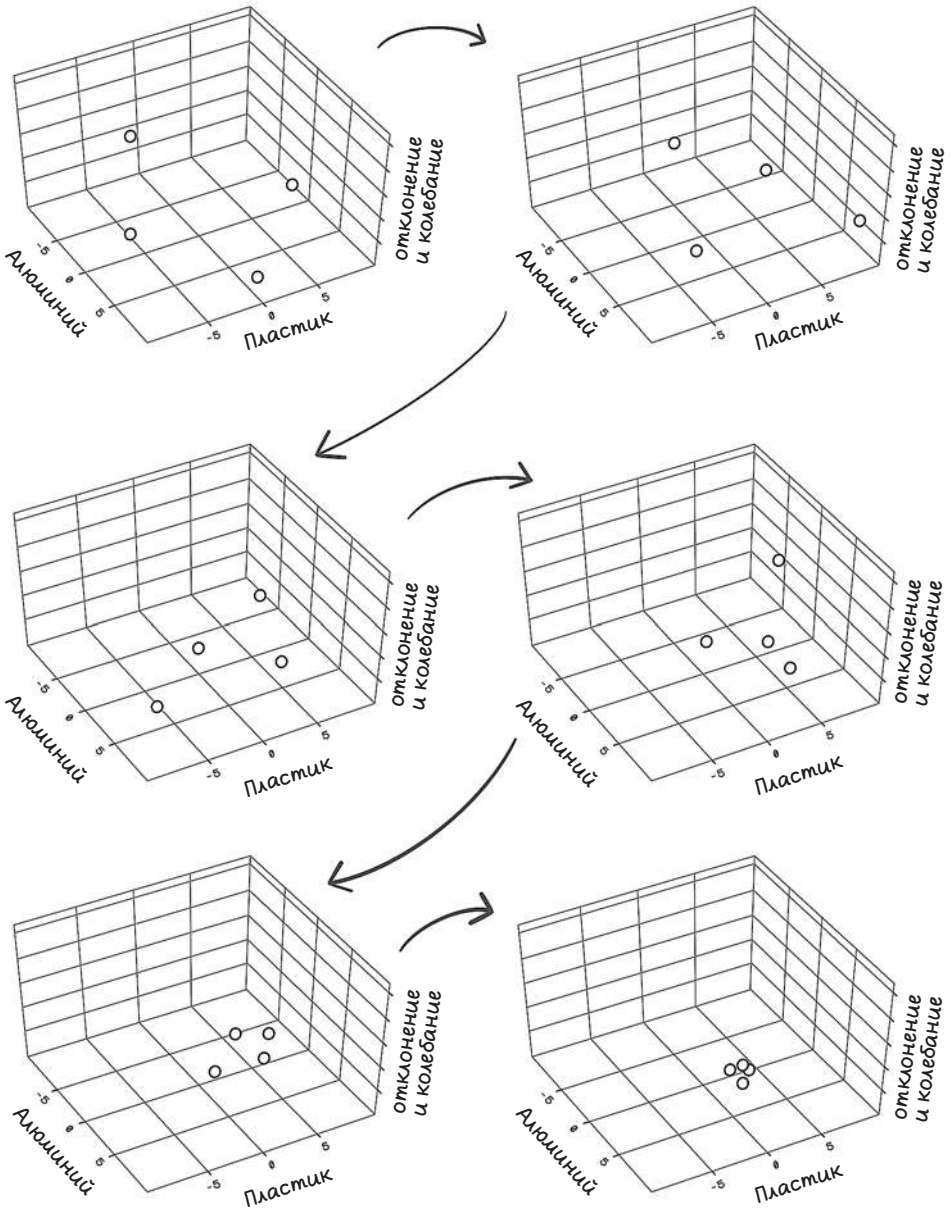


Рис. 7.26. Визуализация перемещения частиц в пространстве поиска

На последней схеме рис. 7.26 все частицы сосредоточились в одной зоне пространства поиска. В качестве окончательного будет принято лучшее решение в рое. В реальных задачах оптимизации невозможно визуализировать всю область поиска (иначе необходимость в алгоритмах оптимизации попросту отпадет). Мы же использовали для примера с дроном функцию, которая называется функцией Бута. При ее отображении на трехмерную поверхность в системе декартовых координат можно видеть, что частицы в самом деле сходятся к минимальной точке пространства поиска (рис. 7.27).

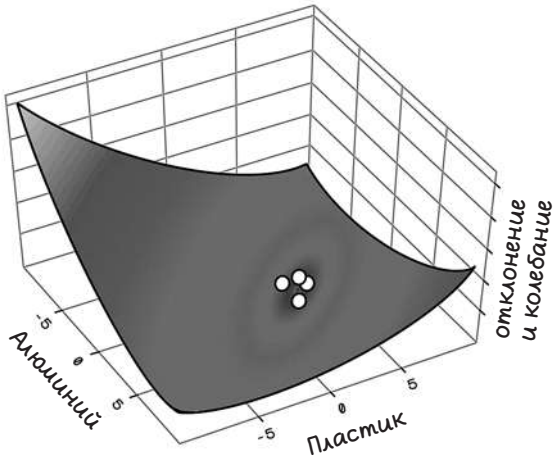


Рис. 7.27. Визуализация известной поверхности и сходимости частиц на ней

Использовав алгоритм роя частиц для примера с дроном, мы определили, что оптимальное соотношение алюминия и пластика с целью минимизации отклонения и колебания составляет 1:3, то есть 1 часть алюминия и 3 части пластика. При передаче этих значений в функцию приспособленности мы получаем значение 0, являющееся минимальным значением функции.

Псевдокод

Этап обновления может оттолкнуть своей сложностью, но если составляющие разделены на простые функции, то писать, использовать и понимать код становится намного проще. Сначала описывается функция вычисления инерции, а также функции определения когнитивного и социального ускорения. Далее требуется функция для измерения расстояния между двумя точками, которое получается извлечением квадратного корня из суммы квадрата разности значений x и квадрата разности значений y.

```
calculate_inertia(inertia_constant, velocity):
  return inertia_constant * current_velocity

calculate_cognitive_acceleration(cognitive_constant):
  return cognitive_constant * random number between 0 and 1

calculate_social_acceleration(social_constant):
  return social_constant * random number between 0 and 1

calculate_distance(best_x, best_y, current_x, current_y):
  return square_root(
              power(best_x - current_x), 2) + power(best_y - current_y), 2)
                  )
```

Когнитивная составляющая вычисляется путем нахождения когнитивного ускорения с помощью функции, которую мы определили в предыдущем разделе, и расстояния между лучшим и текущим положениями частицы:

```
calculate_cognitive(cognitive_constant,
                    particle_best_x, particle_best_y
                    particle_current_x, particle_current_y):
  let acceleration equal cognative_acceleration(cognitive_constant)
  let distance equal calculate_distance(particle_best_x,
                                        particle_best_y
                                        particle_current_x,
                                        particle_current_y)
  return acceleration * distance
```

Социальный компонент вычисляется путем нахождения социального ускорения с использованием приведенной ранее функции и расстояния между лучшим положением роя и текущим положением частицы:

```
calculate_social(social_constant,
                 swarm_best_x, swarm_best_y
                 particle_current_x, particle_current_y):
  let acceleration equal social_acceleration(social_constant)
  let distance equal calculate_distance(swarm_best_x,
                                        swarm_best_y
                                        particle_current_x,
                                        particle_current_y)
  return acceleration * distance
```

Функция обновления включает в себя все определенные критерии, выполняя фактическое обновление скорости и позиции частицы. Скорость вычисляется на основании инерции, когнитивного и социального компонентов. Позиция вычисляется путем добавления полученной скорости к текущему положению частицы:

```
update_particle(cognitive_constant,social_constant,particle_velocity,
                particle_best_x,particle_best_y,
                swarm_best_x,swarm_best_y,
                particle_current_x,particle_current_y)
  let inertia equal calculate_inertia(inertia_constant,
                                      particle_constant)
  let cognitive equal calculate_cognitive(cognitive_constant,
                                      particle_best_x,particle_best_y
                                      particle_current_x,particle_current_y)
  let social equal calculate_social(social_constant,
                                      swarm_best_x,swarm_best_y
                                      particle_current_x,particle_current_y)
  let particle.velocity equal inertia + cognitive + social
  let particle.x equal particle.x + velocity
  let particle.y equal particle.y + velocity
```

УПРАЖНЕНИЕ: ВЫЧИСЛИТЬ НОВУЮ СКОРОСТЬ И ПОЛОЖЕНИЕ ЧАСТИЦЫ 1 С УЧЕТОМ СЛЕДУЮЩЕЙ ИНФОРМАЦИИ

- Инерция равна 0.1.
- Когнитивная константа равна 0.5, а когнитивное случайное число равно 0.2.
- Социальная константа равна 0.5, а когнитивное случайное число равно 0.5.

Частица	Скорость	Текущее количество алюминия	Текущее количество пластика	Текущее значение фитнес-функции	Лучшее количество алюминия	Лучшее количество пластика	Лучшее значение фитнес-функции
1	3	4	8	721.286	7	1	296
2	4	3	3	73.538	0.626	10	73.538
3	1	6	2	302.214	–10	1	80
4	2	2	5	179.105	–0.65	–3.65	179.105

ОТВЕТ: НОВАЯ СКОРОСТЬ И ПОЛОЖЕНИЕ ЧАСТИЦЫ

Инерциальная составляющая:

инерция * текущая скорость

= 0.1 * 3

= 0.3

Когнитивная составляющая:

когнитивное ускорение = когнитивная константа * случайное когнитивное число

= 0.5 * 0.2

= 0.1

когнитивное ускорение * (лучшее положение частицы — текущее положение)

= 0.1 * ([7,1] – [4,8])

= 0.1 * $\text{sqrt}((7-4)^2 + (1-8)^2)$

= 0.1 * 7.616

= 0.7616

Социальная составляющая:

социальное ускорение = социальная константа * случайное социальное число

= 0.5 * 0.5

= 0.25

социальное ускорение * (лучшее положение роя — текущее положение)

= 0.25 * ([0.626,10] – [4,8])

= 0.25 * $\text{sqrt}((0.626-4)^2 + (10-8)^2)$

= 0.25 * 3.922

= 0.981

Новая скорость:

инерциальная составляющая + социальная составляющая + когнитивная составляющая

= 0.3 + 0.7616 + 0.981

= 2.0426

Определение критерия остановки

Частицы роя не могут обновляться и осуществлять поиск бесконечно. Необходимо задать критерий остановки, который определит для алгоритма разумное количество итераций для нахождения подходящего решения (рис. 7.28).

Количество итераций влияет на несколько аспектов поиска решений, включая следующие:

- *Разведка (исследование)*. Частицам требуется время на изучение пространства поиска и нахождение зон с наилучшими решениями. На разведку также влияют константы, определенные в функции обновления скорости.
- *Эксплуатация*. Частицы должны сходиться к хорошему решению после разумной продолжительности исследования.

Стратегия для остановки алгоритма заключается в проверке лучшего решения роя на предмет его стагнации. Стагнация происходит, когда значение лучшего решения не изменяется или изменяется незначительно. В таком случае выполнение дополнительных итераций не поможет найти новые лучшие варианты. Когда лучшее решение впадает в стагнацию, можно скорректировать параметры функции обновления, повысив склонность частиц к разведке. Если требуется более активная разведка, то корректировка обычно подразумевает увеличение числа итераций. Стагнация может означать, что было найдено хорошее решение или что рой застрял в области локального лучшего. Если в начале была произведена достаточная разведка и рой постепенно впадает в стагнацию, значит, найдено хорошее решение (рис. 7.29).

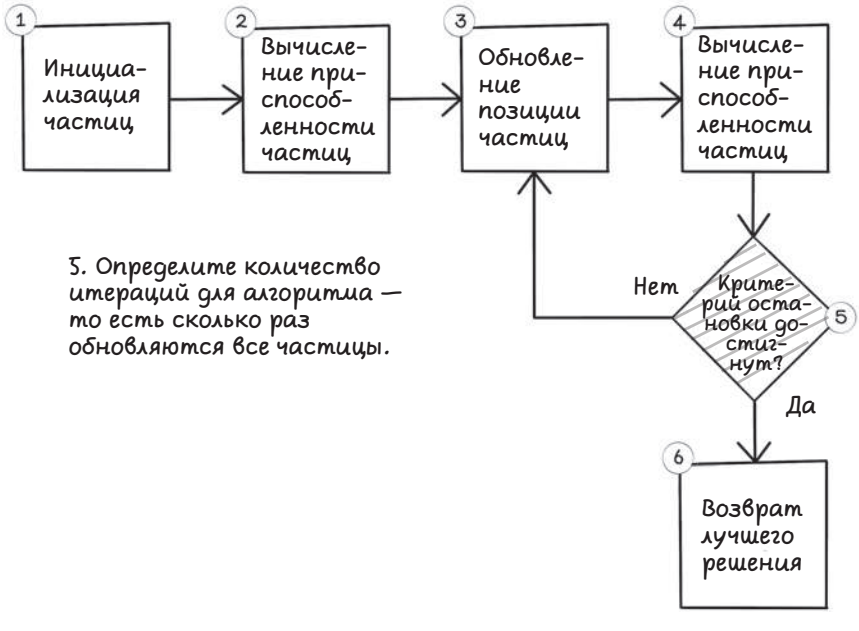


Рис. 7.28. Достиг ли алгоритм условия остановки?

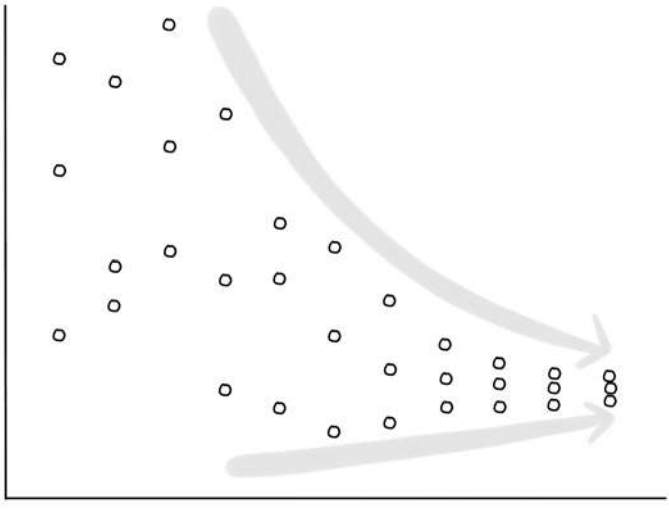


Рис. 7.29. Схождение от разведки к эксплуатации

Применение алгоритма роя частиц

Алгоритмы роя частиц интересны тем, что симулируют естественное поведение в природе, поэтому их проще понимать. При этом они могут использоваться в широком спектре задач разного уровня абстракции. Так, в этой главе была рассмотрена задача оптимизации сборки дрона. Алгоритм роя частиц также можно применять совместно с другими алгоритмами, например искусственными нейронными сетями.

Одним из интересных вариантов применения алгоритма роя частиц является глубокая стимуляция мозга. Ее принцип подразумевает внедрение в человеческий мозг зондов с электродами для лечения, например, болезни Паркинсона. В каждом зонде находятся электроды, которые можно ориентировать в разных направлениях, чтобы оказать нужное воздействие на пациента. Исследователи из университета Миннесоты разработали алгоритм роя частиц для оптимизации выбора направления каждого электрода, чтобы максимально охватить интересующую область, как можно меньше затрагивая нежелательную область и снижая потребление энергии. Так как частицы эффективны при поиске в многомерных пространствах вариантов, алгоритм роя частиц подходит для нахождения оптимальных конфигураций электродов в зондах (рис. 7.30).

К другим областям применения алгоритма роя частиц относятся:

- *Оптимизация весов в искусственных нейронных сетях*. Искусственные нейронные сети моделируются по принципу работы человеческого мозга. В них сигналы передаются от одного нейрона к другому, а нейроны, в свою очередь, эти сигналы корректируют. Задача таких сетей заключается в поиске нужного баланса весов для определения закономерностей в связях между данными. Корректировка весов требует высоких вычислительных затрат, поскольку пространство поиска оказывается огромным. Представьте, например, необходимость перебрать все возможные комбинации десятичных чисел для 10 весов. Этот процесс займет годы.

  Не волнуйтесь, если сказанное кажется для вас непонятным. Мы изучим принцип работы искусственных нейронных сетей в главе 9. В случае нейронных сетей алгоритм роя частиц можно использовать для ускоренной корректировки весов, так как частицы находят оптимальные значения в пространстве решений, не прибегая к перебору каждого возможного веса.

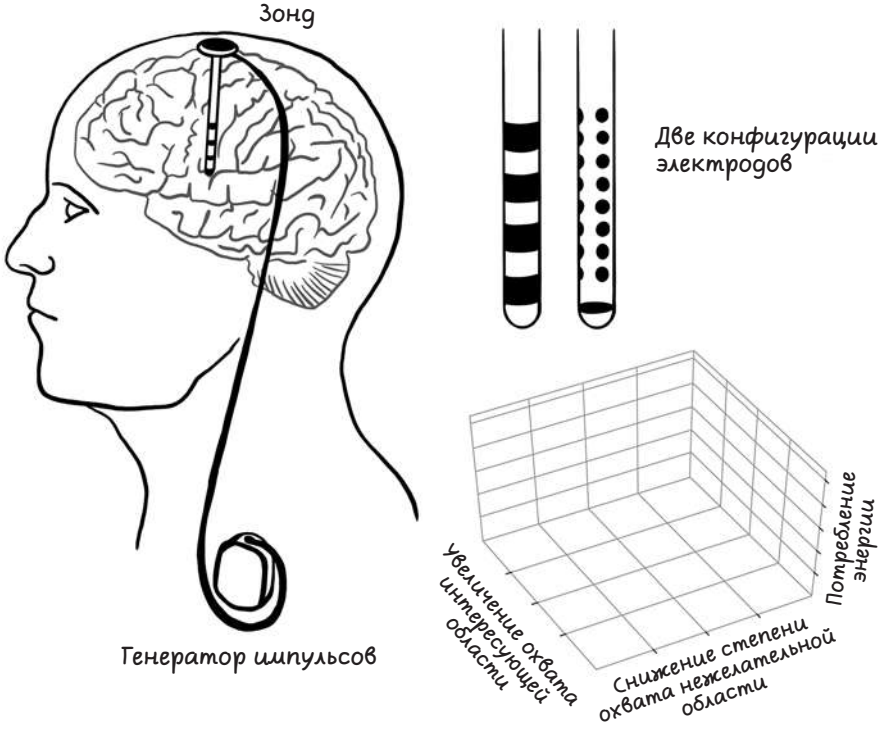
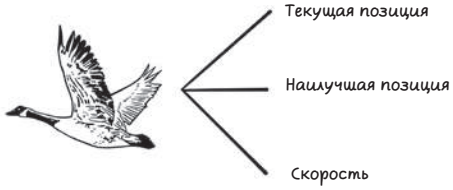


Рис. 7.30. Пример факторов, определяющих эффективность зонда, используемого в глубокой стимуляции мозга

- *Отслеживание движения на видео*. Отслеживание движения людей представляет одну из трудных задач компьютерного зрения. Его цель состоит в обнаружении поз людей и предугадывании движений только на основе информации из видео. Несмотря на одинаковую подвижность суставов, движения каждого человека уникальны. И поскольку изображения содержат множество признаков, пространство поиска становится огромным, поскольку содержит множество измерений для прогнозирования движения человека. Опять же, в многомерных пространствах решений хорошо работают алгоритмы роя частиц, в связи с чем их можно использовать для повышения эффективности отслеживания движений и их прогнозирования.
- *Повышение разборчивости речи в аудиозаписях*. Работать с аудиозаписями довольно сложно. На них всегда присутствует фоновый шум, который может мешать распознаванию записанной речи. Решается это избирательным удалением шума, для чего используется фильтрация частот и сравнение записанных звуков с образцами. Тем не менее это сложное решение, так как снижение определенных частот может положительно сказаться на одной части записи и негативно на другой. Поэтому требуется высокая точность поиска и сопоставления звуков. Традиционные методы справляются с этой задачей медленно ввиду большого объема пространства поиска. А вот алгоритм роя частиц в этом случае позволяет существенно ускорить процесс и добиться более качественного удаления шумов.

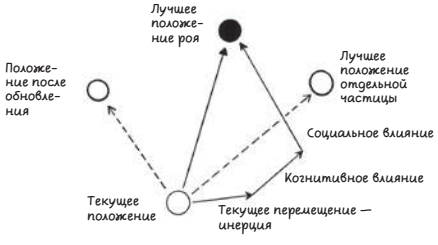
Краткий обзор главы «Роевой интеллект: частицы»

Оптимизация роем частиц широко используется в различных областях.



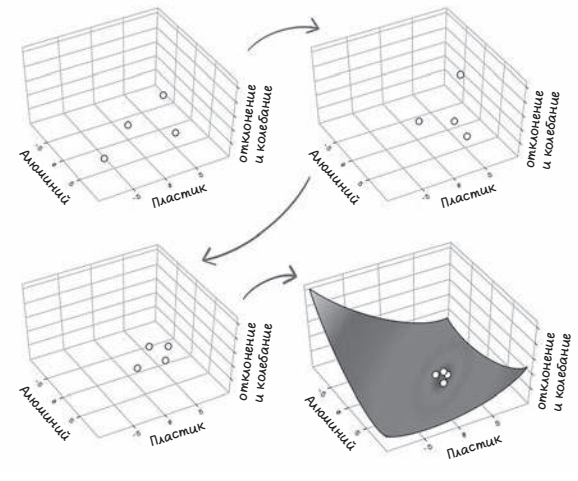
Для перемещения по области решений частицы используют собственное лучшее положение и лучшее положение роя.

Важнейший этап алгоритма оптимизации роем частиц — изменение скорости частиц с учетом влияния инерции, когнитивной составляющей и социальной составляющей.



Новая скорость:

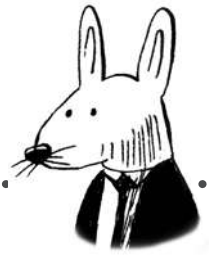
```
инерциальная составляющая + социальная составляющая + когнитивная
                                                       составляющая
(инерция * текущая скорость)
   (социальное ускорение * (лучшее положение роя – текущее положение))
   (когнитивное ускорение * (лучшее положение частицы – текущее
                                                       положение))
```



Частицы перемещаются по области решений, находя возможные хорошие решения, и собираются вместе в зоне общего лучшего решения.

Машинное обучение

8



В этой главе

- ✓ Решение задач с помощью алгоритмов машинного обучения
- ✓ Знакомство с жизненным циклом машинного обучения, подготовка данных и выбор алгоритмов
- ✓ Изучение и реализация алгоритма линейной регрессии для задач прогнозирования
- ✓ Изучение и реализация алгоритма дерева принятия решений для задач классификации
- ✓ Выработка представления о других алгоритмах машинного обучения и их использовании

Что такое машинное обучение?

Машинное обучение (МО) может казаться сложным для освоения и применения. Тем не менее при правильном построении и понимании процесса, а также используемых в нем алгоритмов, работа в этой области может стать очень даже интересной.

Представьте, что вы хотите арендовать квартиру. Вы советуетесь с родственниками и друзьями, после чего решаете поискать предложения в интернете. При этом вы замечаете, что цены на жилье отличаются в зависимости от района города. Изучив рынок, вы получаете следующие данные:

- Однокомнатная квартира в центре (рядом с работой) — $5000/месяц.
- Двухкомнатная квартира в центре — $7000/месяц.
- Однокомнатная квартира в центре с гаражом $6000/месяц.
- Однокомнатная квартира на окраине, откуда вам придется добираться на работу — $3000/месяц.
- Двухкомнатная квартира на окраине — $4500/месяц.
- Однокомнатная квартира на окраине с гаражом — $3800/месяц.

Здесь наблюдаются определенные закономерности. Квартиры в центре стоят дороже, и диапазон их цен колеблется между $5000 и $7000 в месяц. Квартиры на окраине, наоборот, дешевле. Увеличение количества комнат также влечет за собой повышение стоимости на $1500–$2000 в месяц. Помимо этого, наличие гаража прибавляет от $800 до $1000 (рис. 8.1).

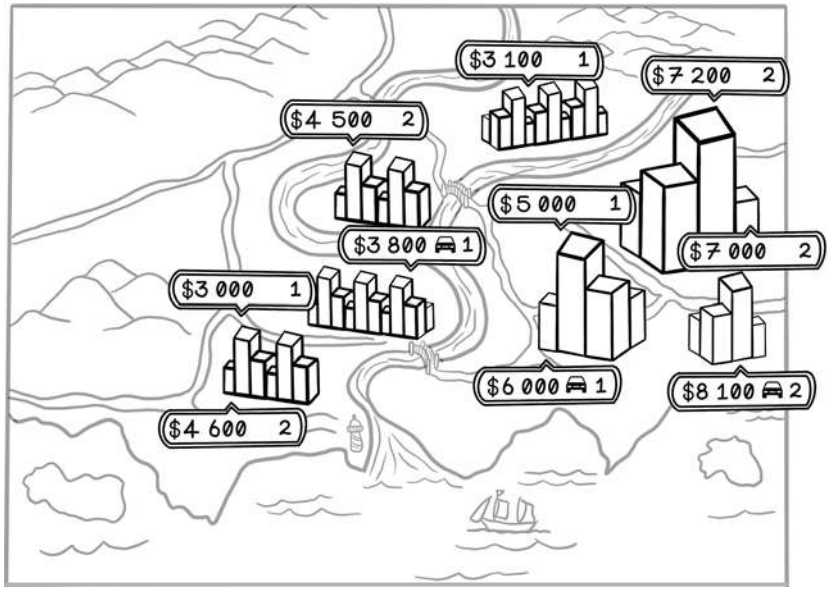


Рис. 8.1. Стоимость жилья в разных частях города с учетом разных критериев

Этот пример показывает, как мы используем данные для выявления закономерностей и принятия решений. Если вы встретите предложение двухкомнатной квартиры в центре с гаражом, то логично будет предположить, что ее стоимость составит в среднем $8000 в месяц.

Машинное обучение ставит целью нахождение закономерностей в данных, что делает его полезным инструментом в реальных жизненных ситуациях. Мы смогли выделить закономерность в небольшом наборе данных, но МО способно выделять множество разных закономерностей в объемных сложных датасетах. На рис. 8.2 показаны связи между разными атрибутами данных. Каждая точка представляет отдельное свойство.

Обратите внимание, что рядом с центром расположено больше точек и что наблюдается отчетливая закономерность в стоимости квартир: при увеличении расстояния до центра города цена постепенно снижается. Стоимость также зависит и от количества комнат. Разрыв между нижней группой точек и верхней указывает на значительное увеличение цены. Можно легко предположить, что этот эффект вызван отдаленностью от центра города, и в таком случае алгоритм МО поможет подтвердить или опровергнуть это предположение. В текущей главе вы узнаете о том, как эти алгоритмы работают.

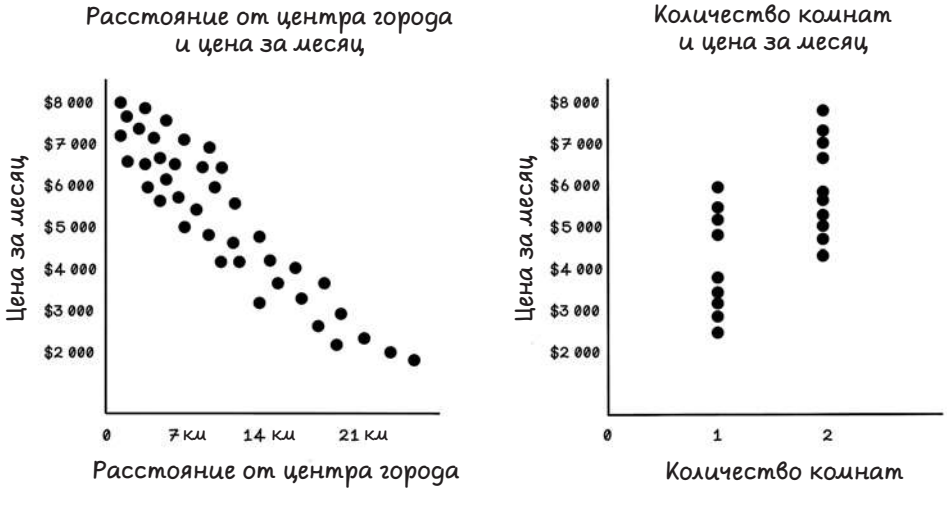


Рис. 8.2. Пример визуализации связей в данных

Как правило, данные представляются в виде таблиц. Столбцы содержат *признаки* данных, а строки — *образцы*. Когда мы сравниваем признаки, то оцениваемый признак обычно обозначается как y, а признаки, на основе которых делается прогноз, как x. По ходу изучения разных задач эта терминология станет более понятна.

Применение машинного обучения

Машинное обучение применимо только тогда, когда у вас имеются данные и есть вопросы, на которые эти данные могут ответить. Алгоритмы МО находят закономерности в данных, но далеко не просты в работе. Различные виды таких алгоритмов используют различные подходы для соответствующих сценариев, чтобы давать ответы на различные вопросы. Речь идет о таких обширных категориях МО, как обучение с учителем, обучение без учителя и обучение с подкреплением (рис. 8.3).

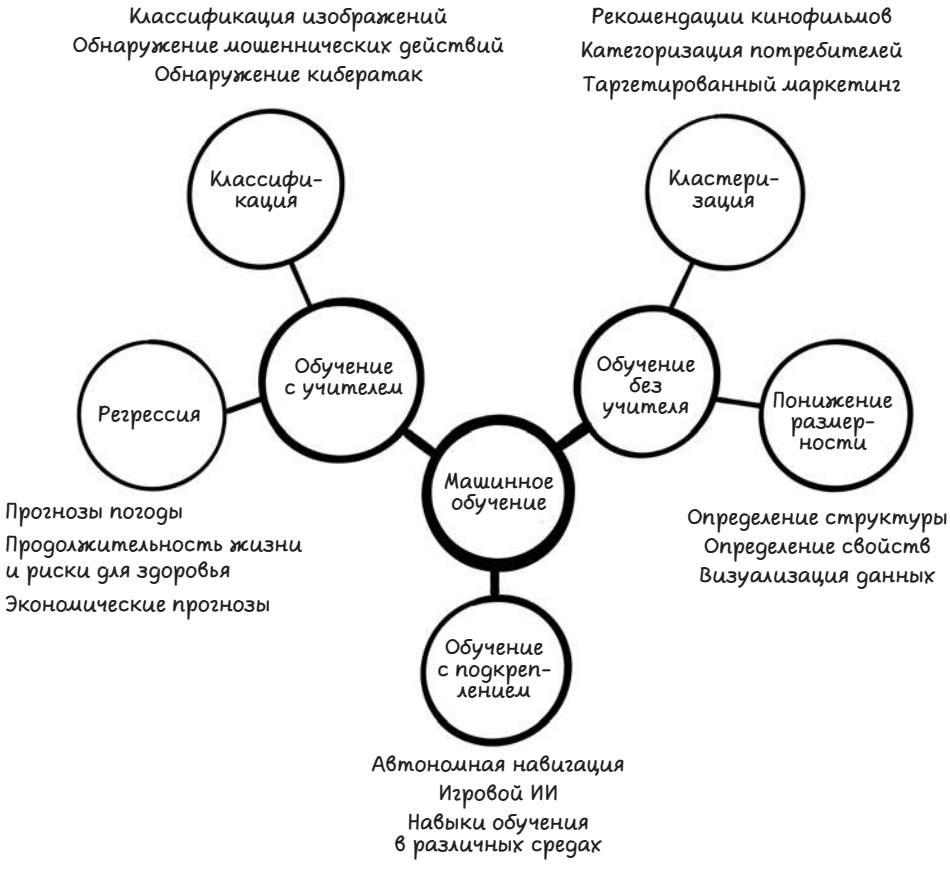


Рис. 8.3. Категоризация машинного обучения и способов его применения

Обучение с учителем

Обучение с учителем — одна из наиболее распространенных техник традиционного машинного обучения. В этом случае мы рассматриваем данные, обнаруживаем в них закономерности и связи, после чего прогнозируем результаты для новых примеров данных в том же формате. Задача с поиском жилья для аренды — пример обучения с учителем, в котором происходит поиск закономерностей. К его примерам также относится автозавершение вводимых в поисковом запросе фраз или рекомендации музыкальными сервисами новой музыки на основе истории прослушивания и предпочтений. Обучение с учителем делится на две подкатегории: регрессию и классификацию.

Регрессия подразумевает рисование линии через набор точек данных таким образом, чтобы она максимально соответствовала общей форме данных. Эта техника может использоваться, например, для построения тенденций взаимосвязей между маркетинговыми акциями и продажами. (Есть ли прямая связь между продвижением товаров через онлайн-рекламу и фактическими продажами?) Ее также можно использовать для определения влияющих на что-либо факторов. (Есть ли прямая связь между временем и стоимостью криптовалют и будет ли цена криптовалюты расти с течением времени экспоненциально?)

Классификация подразумевает прогнозирование категорий образцов на основе их признаков. (Можно ли определить, чем является объект — легковой машиной или грузовиком, — на основе количества колес, веса и максимальной скорости?)

Обучение без учителя

Обучение без учителя подразумевает поиск внутренних закономерностей в данных, которые сложно обнаружить путем самостоятельного анализа. Эта техника полезна для кластеризации данных, имеющих схожие признаки, а также обнаружения в данных важных признаков. К примеру, на сайте онлайн-магазина товары могут быть сгруппированы на основе покупательского поведения. Если многие клиенты приобретают мыло, мочалки и полотенца в одном наборе, то возрастает вероятность того, что и другие захотят купить такую же комбинацию товаров. В итоге мыло, мочалки и полотенца группируются (кластеризуются) вместе и рекомендуются новым клиентам сайта.

Обучение с подкреплением

Обучение с подкреплением основано на поведенческой психологии и работает путем вознаграждения и штрафов алгоритма на основе его действий в среде. Оно имеет ряд сходств и отличий по сравнению с техниками обучения с учителем и без учителя. Обучение с подкреплением стремится обучить агента в условиях среды на основе наград и штрафов. Представьте, что вы даете лакомство домашнему питомцу за хорошее поведение. Чем больше наград он получает за конкретное поведение, тем больше он будет это поведение проявлять. Подробнее обучение с подкреплением будет рассматриваться в главе 10.

Рабочий процесс машинного обучения

Структура процесса машинного обучения строится не только из алгоритмов. На деле в него входит контекст данных, их подготовка, а также постановка вопросов, на которые нужно получить ответы.

Есть два варианта подхода к задаче:

- Задачу можно решить с помощью машинного обучения и для этого необходимо собрать соответствующие данные. Предположим, что у банка есть огромное количество данных о правомерных и мошеннических транзакциях, и его служба безопасности хочет обучить модель, отвечая на следующий вопрос: «Можем ли мы обнаруживать мошеннические транзакции в реальном времени?»
- Есть данные в конкретном контексте, и нужно определить, как их можно использовать для решения нескольких задач. Например, у производителя сельхозпродукции есть данные о погоде и составе почвы в разных регионах, а также о требуемых различным растениям удобрениях. Вопрос может стоять так: «Какие корреляции и связи присутствуют между перечисленными типами данных?» Обнаруженные связи помогут сформулировать более точный вопрос: «Можно ли определить лучший регион для выращивания конкретного растения на основе характерных для этого региона погодных условий и состава почвы?»

На рис. 8.4 показано упрощенное представление этапов типичного процесса машинного обучения.

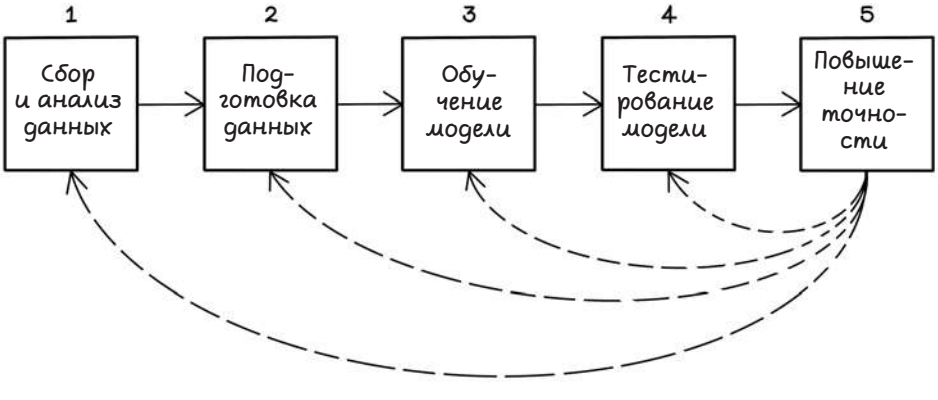


Рис. 8.4. Рабочий процесс машинного обучения: проекты и эксперименты

Сбор и анализ данных: изучение контекста

Чтобы создать успешную модель МО, в первую очередь необходимо собрать и проанализировать данные, с которыми предстоит работать. Если вы работаете, например, в сфере финансов, то знание соответствующей терминологии, внутренних процессов и данных будет важно для сбора максимально эффективной информации, которая позволит решить поставленную задачу. Если вы хотите создать систему обнаружения мошеннических действий, важно понять, какие именно данные хранятся в транзакциях и что они означают. При этом также потребуется собрать и объединить данные из различных систем. Иногда для повышения точности собранные данные дополняются данными сторонних организаций. В текущем разделе мы разберем схему организации машинного обучения и познакомимся с различными алгоритмами на примере набора данных о бриллиантах (рис. 8.5).

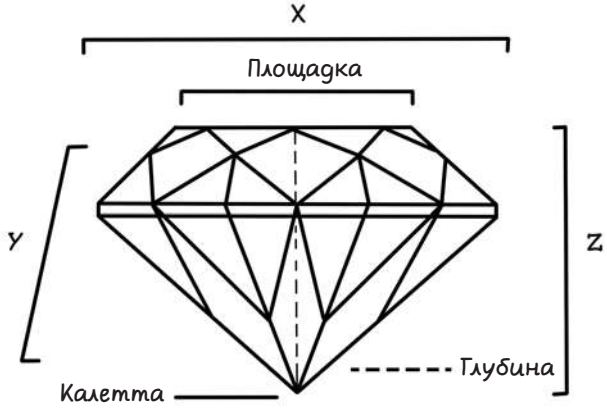


Рис. 8.5. Терминология для оценки бриллианта

В табл. 8.1 перечислены несколько бриллиантов и их свойства. Оси X, Y и Z описывают размер бриллианта в трех пространственных измерениях. Для образцов используется только поднабор данных.

Таблица 8.1. Набор данных о бриллиантах

	Каратность	Огранка	Цвет	Чистота	Глубина	Площадка	Цена	X	Y	Z
1	0.30	Хорошая	J	SI1	64.0	55	339	4.25	4.28	2.73
2	0.41	Идеальная	I	SI1	61.7	55	561	4.77	4.80	2.95
3	0.75	Очень хорошая	D	SI1	63.2	56	2760	5.80	5.75	3.65
4	0.91	Удовлетворительная	H	SI2	65.7	60	2763	6.03	5.99	3.95
5	1.20	Удовлетворительная	F	I1	64.6	56	2809	6.73	6.66	4.33
6	1.31	Премиальная	J	SI2	59.7	59	3697	7.06	7.01	4.20
7	1.50	Премиальная	H	I1	62.9	60	4022	7.31	7.22	4.57
8	1.74	Очень хорошая	H	I1	63.2	55	4677	7.62	7.59	4.80
9	1.96	Удовлетворительная	I	I1	66.8	55	6147	7.62	7.60	5.08
10	2.21	Премиальная	H	I1	62.2	58	6535	8.31	8.27	5.16

Набор данных состоит из 10 столбцов, которые называются *признаками*. В полной версии набор содержит более 50 000 строк. Вот значения каждого из признаков:

- *Каратность* — вес бриллианта. Для интересующихся: 1 карат равен 200 мг.
- *Огранка* — качество бриллианта по возрастанию: удовлетворительное, хорошее, очень хорошее, премиальное и идеальное.
- *Цвет* — цвет бриллианта в диапазоне от D до J, где D — это лучший цвет, а J — худший. D означает высокую прозрачность, а J указывает на мутность изделия.
- *Чистота* — степень несовершенства бриллианта по убыванию: FL, IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1, I2 и I3. (Не стремитесь понять эти кодовые значения; они просто представляют различные уровни совершенства.)
- *Глубина* — процент глубины, измеряемый от калетты до площадки. Отношение площадки к глубине главным образом определяет характеристику блеска бриллианта.
- *Площадка* — доля плоской стороны бриллианта в отношении измерения X.
- *Цена* — стоимость бриллианта при продаже.
- *X* — измерение x в миллиметрах.

- Y — измерение y в миллиметрах.
- Z — измерение z в миллиметрах.

На примере этого датасета мы разберем подготовку данных и их последующую обработку алгоритмом МО.

Подготовка данных: очистка и сортировка

Собранные данные никогда не бывают идеальными. Они могут быть получены из разных систем и организаций, в которых наверняка используются различные стандарты и правила сохранения их целостности. Данные могут отсутствовать, могут быть несогласованными или иметь сложный для обработки формат.

В примере набора данных о бриллиантах из табл. 8.2 важно понять, что столбцы представляют *признаки*, а строки *образцы*.

Таблица 8.2. Набор данных о бриллиантах с недостающими значениями

	Каратность	Огранка	Цвет	Чистота	Глубина	Площадка	Цена	X	Y	Z
1	0.30	Хорошая	J	SI1	64.0	55	339	4.25	4.28	2.73
2	0.41	Идеальная	I	si1	61.7	55	561	4.77	4.80	2.95
3	0.75	Очень хорошая	D	SI1	63.2	56	2760	5.80	5.75	3.65
4	0.91	—	H	SI2	—	60	2763	6.03	5.99	3.95
5	1.20	Удовлетворительная	F	I1	64.6	56	2809	6.73	6.66	4.33
6	1.21	Хорошая	E	I1	57.2	62	3144	7.01	6.96	3.99
7	1.31	Премиальная	J	SI2	59.7	59	3697	7.06	7.01	4.20
8	1.50	Премиальная	H	I1	62.9	60	4022	7.31	7.22	4.57
9	1.74	Очень хорошая	H	i1	63.2	55	4677	7.62	7.59	4.80
10	1.83	удовлетворительная*	J	I1	70.0	58	5083	7.34	7.28	5.12
11	1.96	Удовлетворительная	I	I1	66.8	55	6147	7.62	7.60	5.08
12	—	Премиальная	H	i1	62.2	—	6535	8.31	—	5.16

* Пояснение см. на с. 252.

Недостающие данные

В образце 4 отсутствуют значения признаков огранки и глубины, а в образце 12 нет значений каратности, площадки и Y. Чтобы сравнивать образцы, необходимо полное понимание данных, и отсутствующие значения усложняют этот процесс. Цель проекта машинного обучения может заключаться в приблизительной оценке (прогнозировании) этих значений. Прогнозирование мы будем рассматривать чуть позже. Предположим, что недостающие данные вносят определенные сложности и не позволяют получить полезные результаты. Вот как можно решить эту проблему:

- *Удаление*. Удалить образцы, в которых отсутствуют значения признаков, — в данном случае это образцы 4 и 12 (табл. 8.3). Данные таким образом становятся более надежными, так как в них отсутствуют предположения. Тем не менее удаленные образцы могли быть важны для достижения цели.

Таблица 8.3. Набор данных о бриллиантах с недостающими значениями: удаление образцов

	Каратность	Огранка	Цвет	Чистота	Глубина	Площадка	Цена	X	Y	Z
1	0.30	Хорошая	J	SI1	64.0	55	339	4.25	4.28	2.73
2	0.41	Идеальная	I	si1	61.7	55	561	4.77	4.80	2.95
3	0.75	Очень хорошая	D	SI1	63.2	56	2760	5.80	5.75	3.65
4	0.91	—	H	SI2	—	60	2763	6.03	5.99	3.95
5	1.20	Удовлетворительная	F	I1	64.6	56	2809	6.73	6.66	4.33
6	1.21	Хорошая	E	I1	57.2	62	3144	7.01	6.96	3.99
7	1.31	Премиальная	J	SI2	59.7	59	3697	7.06	7.01	4.20
8	1.50	Премиальная	H	I1	62.9	60	4022	7.31	7.22	4.57
9	1.74	Очень хорошая	H	i1	63.2	55	4677	7.62	7.59	4.80
10	1.83	удовлетворительная	J	I1	70.0	58	5083	7.34	7.28	5.12
11	1.96	Удовлетворительная	I	I1	66.8	55	6147	7.62	7.60	5.08
12	—	Премиальная	H	i1	62.2	—	6535	8.31	—	5.16

- *Среднее или медиана*. Другой вариант — подставить вместо недостающих значений средние или медианные величины соответствующих признаков.

Среднее вычисляется путем сложения всех соответствующих значений и деления полученной суммы на количество значений. *Медиана* вычисляется путем

упорядочивания значений в порядке возрастания и выбора значения, которое находится посередине.

Использовать среднее достаточно просто и эффективно, но оно не учитывает возможные корреляции между признаками. Этот подход нельзя применять к таким категориальным признакам, как огранка, чистота и глубина (табл. 8.4).

Таблица 8.4 Набор данных о бриллиантах с недостающими значениями: подстановка средних значений

	Каратность	Огранка	Цвет	Чистота	Глубина	Площадка	Цена	X	Y	Z
1	0.30	Хорошая	J	SI1	64.0	55	339	4.25	4.28	2.73
2	0.41	Идеальная	I	si1	61.7	55	561	4.77	4.80	2.95
3	0.75	Очень хорошая	D	SI1	63.2	56	2760	5.80	5.75	3.65
4	0.91	—	H	SI2	—	60	2763	6.03	5.99	3.95
5	1.20	Удовлетворительная	F	I1	64.6	56	2809	6.73	6.66	4.33
6	1.21	Хорошая	E	I1	57.2	62	3144	7.01	6.96	3.99
7	1.31	Премиальная	J	SI2	59.7	59	3697	7.06	7.01	4.20
8	1.50	Премиальная	H	I1	62.9	60	4022	7.31	7.22	4.57
9	1.74	Очень хорошая	H	i1	63.2	55	4677	7.62	7.59	4.80
10	1.83	удовлетворительная	J	I1	70.0	58	5083	7.34	7.28	5.12
11	1.96	Удовлетворительная	I	I1	66.8	55	6147	7.62	7.60	5.08
12	1.19	Премиальная	H	i1	62.2	57	6535	8.31	—	5.16

Чтобы вычислить среднее для признака, нужно сложить все доступные значения и разделить их сумму на количество этих значений:

```
Среднее значение площадки = (55 + 55 + 56 + 60 + 56 + 62 + 59 + 60 + 55 + 58 + 55) / 11
Среднее значение площадки = 631 / 11
Среднее значение площадки = 57.364
```

Для недостающих значений имеет смысл использовать среднее значение площадки, поскольку оно незначительно различается для разных образцов. Тем не менее могут присутствовать корреляции, которых мы не видим, например отношение между размером площадки и шириной бриллианта, представленной измерением X.

С другой стороны, использовать среднее значение каратности бессмысленно, потому что если построить график, то между признаками каратности и цены корреляция будет очевидной — рост цены при увеличении значения карата.

- *Наиболее частое*. Заполнение пустых граф наиболее часто встречающимися значениями для данного признака, которые называются *модой* данных. Такой подход отлично работает с категориальными признаками, но не учитывает возможные корреляции между признаками, в связи с чем может появляться смещение из-за использования наиболее частых значений.
- *(Продвинутые) Статистические подходы*. Использование k-ближайшего соседа или нейронных сетей. Техника k-ближайшего соседа находит значение на основе множества признаков данных. Подобно этой технике, нейронные сети могут точно прогнозировать недостающие значения при наличии достаточного количества данных. Оба этих алгоритма для определения недостающих значений требуют высоких вычислительных затрат.
- *(Продвинутый) Ничего не делать*. Некоторые алгоритмы обрабатывают недостающие данные без подготовки, например XGBoost, но для алгоритмов, которые будем изучать мы, такой метод не подойдет.

Неопределенные значения

Еще одна проблема — это одни и те же значения, которые представлены по-разному. Посмотрите на строки 2, 9, 10 и 12 в таблице. Эти значения огранки и чистоты начинаются с нижнего регистра, а не с верхнего, как остальные. Обратите внимание: мы смогли заметить это только потому, что понимаем эти признаки и их возможные значения. Без этого понимания мы могли бы воспринимать «Удовлетворительно» и «удовлетворительно» как разные категории. Чтобы исправить неопределенность, можно привести все значения к верхнему или нижнему регистру (табл. 8.5).

Таблица 8.5. Набор данных о бриллиантах с неопределенными значениями: стандартизация значений

	Каратность	Огранка	Цвет	Чистота	Глубина	Площадка	Цена	X	Y	Z
1	0.30	Хорошая	J	SI1	64.0	55	339	4.25	4.28	2.73
2	0.41	Идеальная	I	si1	61.7	55	561	4.77	4.80	2.95
3	0.75	Очень хорошая	D	SI1	63.2	56	2760	5.80	5.75	3.65
4	0.91	—	H	SI2	—	60	2763	6.03	5.99	3.95
5	1.20	Удовлетворительная	F	I1	64.6	56	2809	6.73	6.66	4.33
6	1.21	Хорошая	E	I1	57.2	62	3144	7.01	6.96	3.99
7	1.31	Премиальная	J	SI2	59.7	59	3697	7.06	7.01	4.20
8	1.50	Премиальная	H	I1	62.9	60	4022	7.31	7.22	4.57
9	1.74	Очень хорошая	H	i1	63.2	55	4677	7.62	7.59	4.80
10	1.83	удовлетворительная	J	I1	70.0	58	5083	7.34	7.28	5.12
11	1.96	Удовлетворительная	I	I1	66.8	55	6147	7.62	7.60	5.08
12	1.19	Премиальная	H	i1	62.2	57	6535	8.31	—	5.16

Кодирование категориальных данных

Поскольку компьютеры и статистические модели работают с численными значениями, возникает сложность с моделированием строковых и категориальных типов данных, таких как «Удовлетворительно», «Хорошо», «Sl1» и «l1». Эти категориальные значения нужно представить как численные одним из следующих способов:

- *Прямое унитарное кодирование (one-hot encoding)*. Этот вид кодирования напоминает переключатели, которые, за исключением одного, находятся в положении «Выкл.». Если требуется представить огранку с помощью прямого унитарного кодирования, то этот признак преобразуется в пять разных признаков, каждый со значением 0, за исключением одного, который и представляет истинное значение огранки соответствующего образца. Обратите внимание, что в табл. 8.6 другие признаки были удалены в целях экономии пространства.

Таблица 8.6. Набор данных о бриллиантах с закодированными значениями

	Каратность	Огранка: удовлетворительная	Огранка: хорошая	Огранка: очень хорошая	Огранка: премиальная	Огранка: идеальная
1	0.30	0	**1**	0	0	0
2	0.41	0	0	0	0	**1**
3	0.75	0	0	**1**	0	0
4	0.91	0	0	0	0	0
5	1.20	**1**	0	0	0	0
6	1.21	0	**1**	0	0	0
7	1.31	0	0	0	**1**	0
8	1.50	0	0	0	**1**	0
9	1.74	0	0	**1**	0	0
10	1.83	**1**	0	0	0	0
11	1.96	**1**	0	0	0	0
12	1.19	0	0	0	**1**	0

- *Кодирование меток (label encoding)*. Представление каждой категории как числа между 0 и общим количеством категорий. Эта техника подходит только для рейтингов или связанных с рейтингом таблиц. В противном случае обучаемая модель предположит, что число указывает на вес образца, а это вызовет нежелательное смещение.

УПРАЖНЕНИЕ: НАЙДИТЕ И ИСПРАВЬТЕ НЕДОЧЕТЫ В ПРЕДСТАВЛЕНИИ ДАННЫХ В ПРЕДЛОЖЕННОМ ПРИМЕРЕ

Определите, какие техники подготовки данных можно использовать для исправления следующего датасета. Определите, какие строки удалить, для каких значений использовать среднее арифметическое и как закодировать категориальные значения. Обратите внимание, что датасет немного отличается от рассмотренного выше.

	Каратность	Происхождение	Глубина	Таблица	Цена	X	Y	Z
1	0.35	Южная Африка	64.0	55	450	4.25		2.73
2	0.42	Канада	61.7	55	680		4.80	2.95
3	0.87	Канада	63.2	56	2689	5.80	5.75	3.65
4	0.99	Ботсвана	65.7		2734	6.03	5.99	3.95

	Каратность	Происхождение	Глубина	Таблица	Цена	X	Y	Z
5	1.34	Ботсвана	64.6	56	2901	6.73	6.66	
6	1.45	Южная Африка	59.7	59	3723	7.06	7.01	4.20
7	1.65	Ботсвана	62.9	60	4245	7.31	7.22	4.57
8	1.79		63.2	55	4734	7.62	7.59	4.80
9	1.81	Ботсвана	66.8	55	6093	7.62	7.60	5.08
10	2.01	Южная Африка	62.2	58	7452	8.31	8.27	5.16

ОТВЕТ: ИСПРАВЛЕНИЕ НЕДОЧЕТОВ В ПРЕДСТАВЛЕНИИ ДАННЫХ

Возможный способ исправить предложенные данные состоит из трех этапов:

- *Удалить строку 8, поскольку значение происхождения отсутствует*. Однако нам не известно, для чего будет использоваться этот датасет. Если признак происхождения окажется важен, то отсутствие этой строки может вызвать проблемы. Как вариант, значение этой строки можно спрогнозировать, если оно связано с другими признаками.
- *Закодировать значение столбца «Происхождение» в прямой унитарной кодировке*. В примере, который рассматривался выше, мы преобразовывали строковые значения в численные с помощью кодирования меток. Этот подход работал, потому что значения выражали степень качества огранки, чистоты или цвета. В случае же с происхождением значение указывает место добычи алмаза. Используя кодирование меток, мы вносим смещение, поскольку ни одно из мест происхождения не лучше другого.
- *Найти среднее для недостающих значений*. В строках 1, 2, 4 и 5 отсутствуют значения для Y, X, площадки и Z соответственно. Использование среднего — хороший вариант, так как нам известно, что размерность и площадка бриллианта взаимосвязаны.

Данные для обучения и контроля

Прежде чем перейти к обучению модели линейной регрессии, нужно убедиться, что у нас есть необходимые данные, а также создать отдельный контрольный набор данных, по которым мы будем определять, насколько эффективно модель прогнозирует новые образцы. Вернемся к примеру стоимости жилья. После выяснения влияния признаков на стоимость можно делать ее прогноз на основе известного расстояния до центра города и количества комнат. Для этого примера в качестве обучающих данных мы возьмем табл. 8.7, а остальные данные используем в дальнейшем.

Обучение модели: прогнозирование с помощью линейной регрессии

Выбор алгоритма в основном опирается на два фактора: поставленный вопрос и природу доступных данных. Если вопрос состоит в прогнозировании цены бриллианта определенного веса, то будет уместно использование алгоритмов регрессии. Выбор алгоритма также зависит от количества признаков в датасете и связей между ними. Если в данных много измерений, то есть для прогнозирования нужно учитывать множество признаков, то можно выбирать из нескольких алгоритмов и подходов.

Регрессия означает прогнозирование непрерывного значения, такого как цена или вес бриллианта в каратах. Непрерывность означает, что значения могут быть представлены любым числом диапазона. Например, цена $2271 является непрерывным значением между 0 и максимальной ценой любого бриллианта, которую может спрогнозировать регрессия.

Линейная регрессия — это один из простейших алгоритмов МО. Он находит связи между двумя переменными и позволяет прогнозировать одну переменную на основе другой. Пример — прогнозирование цены бриллианта на основе значения его веса в каратах. После рассмотрения множества образцов известных бриллиантов, включая их значения цены и веса, модель обучается найденным связям и может делать оценочные прогнозы.

Подгонка линии регрессии к данным

Начнем с поиска тренда в данных и попытки сделать прогноз. При изучении линейной регрессии мы задаем два вопроса: «Есть ли связь между весом бриллианта в каратах и его ценой?» и «Если таковая связь есть, то можно ли сделать точные прогнозы?»

Сначала выделим два интересующих нас признака и отразим эти данные на графике. Так как нам нужно найти цену исходя из значения каратности, то каратность мы обозначим x, а цену — y. Почему мы выбрали именно этот подход?

- *Каратность как независимая переменная (x). Независимая переменная* — это переменная, которая изменяется в ходе эксперимента для определения ее влияния на зависимую. В данном примере значение каратности будет корректироваться для выявления цены бриллианта с этим значением.
- *Цена как зависимая переменная (y). Зависимая переменная* — это определяемая переменная. На нее оказывают влияние как сама независимая переменная, так и изменения, связанные с изменениями значений независимой переменной. В текущем примере нас интересует цена с учетом конкретного значения каратности.

На рис. 8.6 показан график данных каратности и цен, а табл. 8.7 содержит фактические данные.

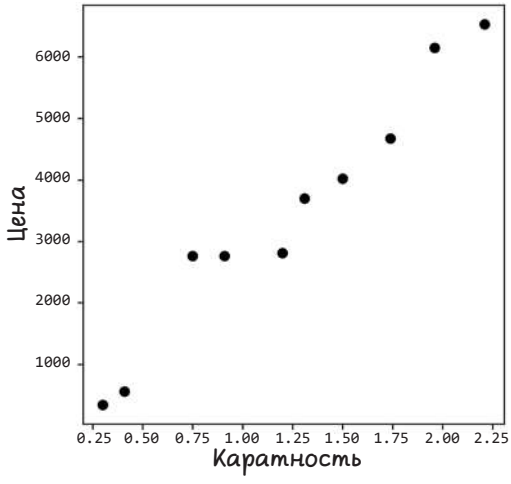


Таблица 8.7. Данные каратности и цены

	Каратность (*x*)	**Цена (*y*)**
1	0.30	339
2	0.41	561
3	0.75	2760
4	0.91	2763
5	1.20	2809
6	1.31	3697
7	1.50	4022
8	1.74	4677
9	1.96	6147
10	2.21	6535

Рис. 8.6. Диаграмма рассеяния данных цены и каратности

Заметьте, что по сравнению с величиной цены значения каратности очень малы. Цены измеряются тысячами, а каратность единицами или долями единицы. Чтобы упростить понимание вычислений (исключительно в целях обучения), масштабируем значения каратности до сопоставимых с ценой величин. Умножив каждое значение каратности на 1000, мы получим числа, которые будет легче вычислять вручную в дальнейшем разборе. Обратите внимание, что при масштабировании всех строк мы не нарушаем связи в данных, поскольку к каждому образцу применяется одна и та же операция. Полученные данные (рис. 8.7) приводятся в табл. 8.8.

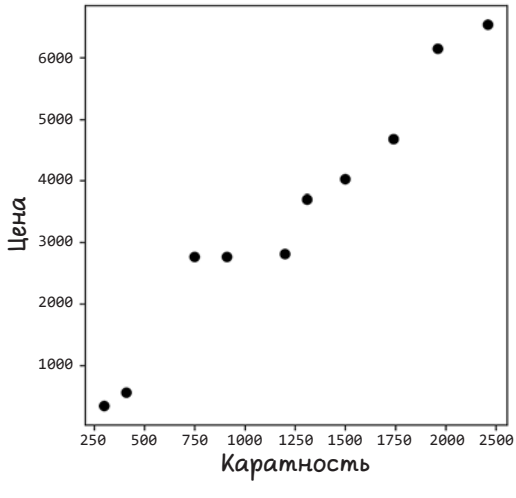


Таблица 8.8. Данные после корректировки значений каратности

	Каратность (*x*)	**Цена (*y*)**
1	300	339
2	410	561
3	750	2760
4	910	2763
5	1200	2809
6	1310	3697
7	1500	4022
8	1740	4677
9	1960	6147
10	2210	6535

Рис. 8.7. График распределения данных цены и каратности

Поиск среднего значения признаков

Чтобы построить линию регрессии, первым делом нужно найти среднее для каждого признака. Еще раз напомню, что среднее — это сумма всех значений, деленная на их количество. Для каратности среднее будет равно 1229 (вертикальная линия, перпендикулярная оси *x* на рис. 8.8). Среднее для цены будет равно $3431 (горизонтальная линия, перпендикулярная оси *y* на рис. 8.8).

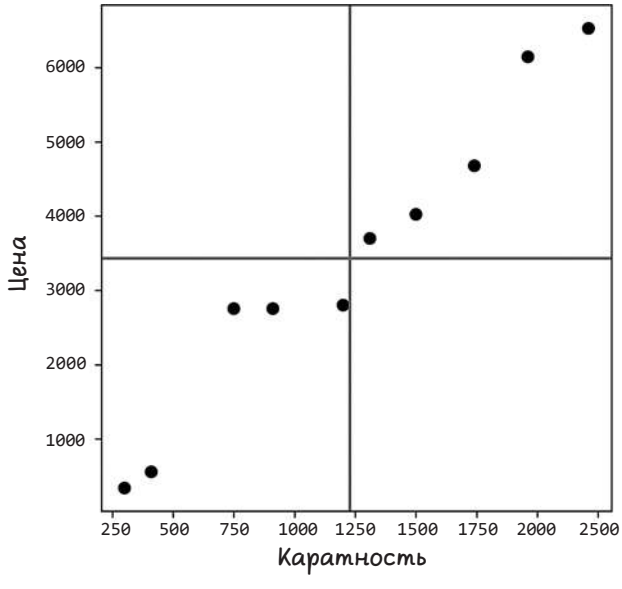


Рис. 8.8. Средние значения *x* и *y*, представленные горизонтальной и вертикальной линиями

Среднее значение важно, так как математически любая находимая нами линия регрессии будет проходить через пересечение среднего *x* и среднего *y*. Через эту точку может проходить множество линий. Некоторые из них будут соответствовать данным лучше других. Здесь зачастую применяется метод *наименьших квадратов*, который подразумевает нахождение линии с наименьшим расстоянием от всех точек датасета. На рис. 8.9 показаны примеры линий регрессии.

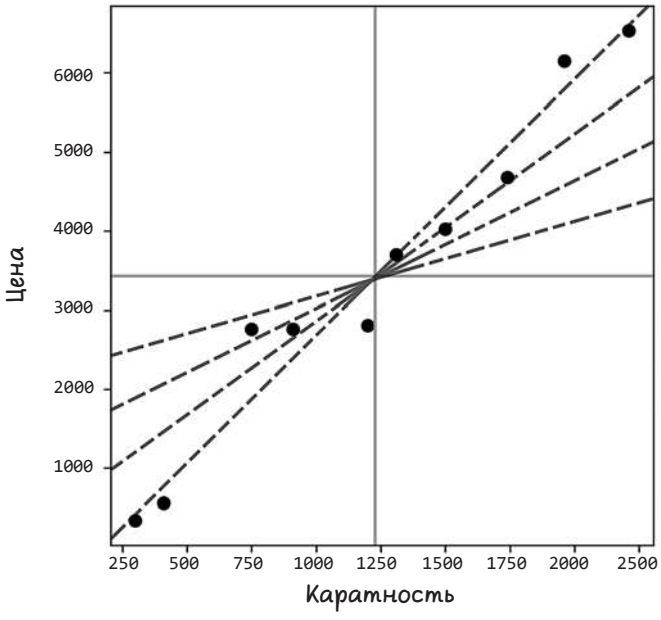


Рис. 8.9. Возможные линии регрессии

Поиск линий регрессии с помощью метода наименьших квадратов

А каково вообще назначение линий регрессии? Предположим, что мы строим метро, станции которого должны находиться как можно ближе к главным офисным зданиям города. Будет нецелесообразным прокладывать линию, которая проходит через все здания, так как станций получится очень много и стоимость проекта окажется слишком высока. Поэтому мы пробуем создать такой прямой маршрут, расстояние от линии которого до каждого здания будет минимальным. Одним пассажирам придется дольше идти до работы, чем другим, но в целом прямая линия будет оптимизирована для всех офисов. Тот же смысл заключается и в линии регрессии. Здания — это точки данных, а линия — это прямой подземный путь (рис. 8.10).

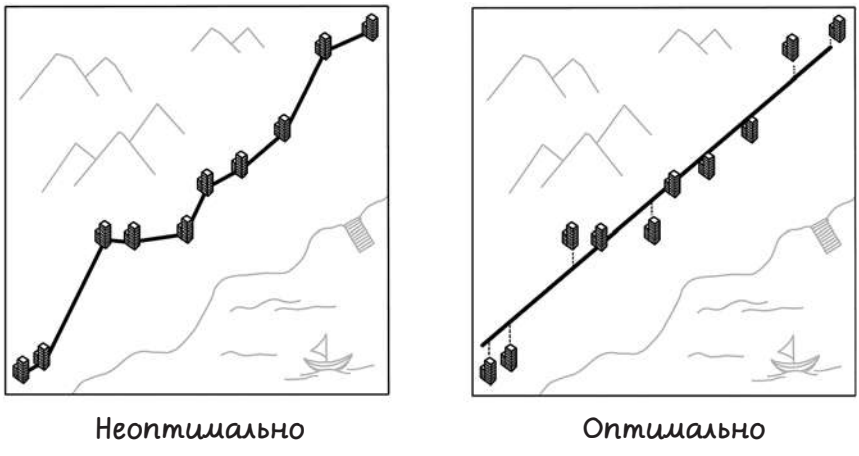


Рис. 8.10. Наглядное пояснение линии регрессии

Линейная регрессия всегда будет находить такую прямую, общее расстояние до которой от всех точек данных будет минимально. При этом важно понять уравнение прямой, так как мы будем учиться находить значения переменных, которые ее описывают.

Прямая выражается уравнением $y = c + mx$ (рис. 8.11):

- *y:* зависимая переменная;
- *x:* независимая переменная;
- *m:* угловой коэффициент прямой;
- *c:* значение *y* в точке пересечения линией оси *y*.

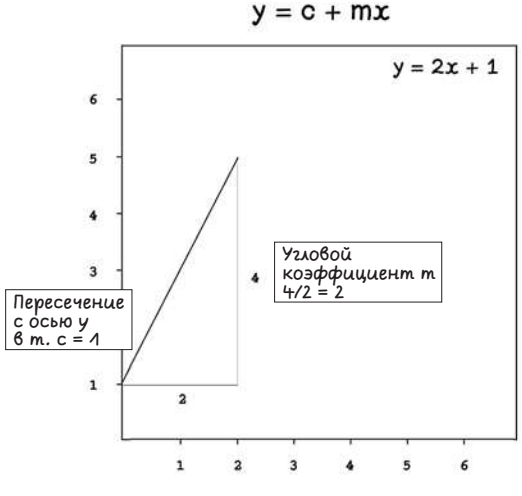


Рис. 8.11. Разбор уравнения, представляющего прямую

Как уже говорилось, линию регрессии мы ищем с помощью метода наименьших квадратов. На верхнем уровне этот процесс состоит из шагов, представленных на рис. 8.12. Для определения ближайшей к данным прямой мы находим разность между фактическими и прогнозируемыми значениями данных. Для всех точек разность будет отличаться. В некоторых случаях она будет большой, в других малой. Иногда разность будет представлена отрицательными значениями, а иногда положительными. Возводя разности в квадрат и суммируя их, мы учитываем все разности для всех точек данных. Минимизация общей разности подразумевает получение наименьшей разности квадратов и дает оптимальную линию регрессии. Не пугайтесь, если рис. 8.12 покажется вам сложным. Мы разберем каждый его шаг.

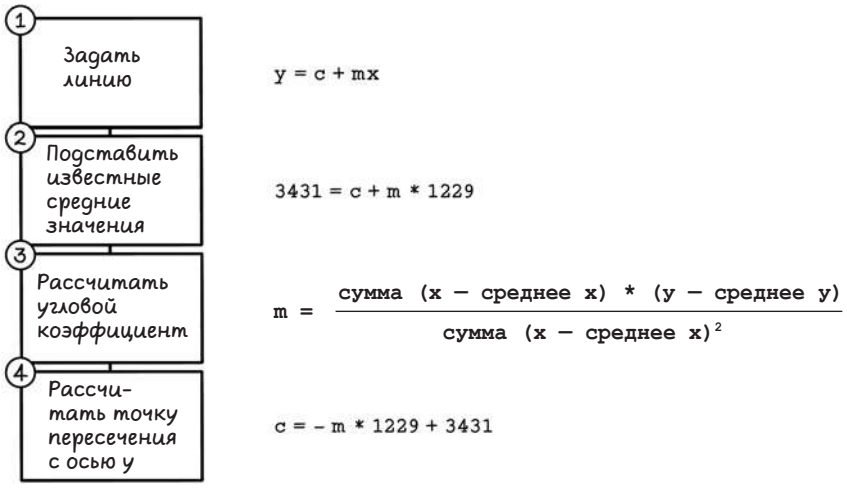


Рис. 8.12. Основные этапы вычисления линии регрессии

Сейчас у прямой есть две известные переменные. Мы знаем, что значение x равно 1229, а y равно 3431, как показано на шаге 2.

Далее вычислим разность между каждым значением каратности и ее средним значением, а также между каждым значением цены и средним значением цены, получая *(x – среднее x)* и *(y – среднее y)*, которые используются на шаге 3 (табл. 8.9).

Таблица 8.9. Набор данных о бриллиантах и вычисления, часть 1

	Каратность (*x*)	**Цена** (*y*)	*x* – **среднее** *x*		*y* – **среднее** *y*	
1	300	339	300 — 1229	–929	339 — 3431	–3092
2	410	561	410 — 1229	–819	561 — 3431	–2870
3	750	2760	750 — 1229	–479	2760 — 3431	–671
4	910	2763	910 — 1229	–319	2763 — 3431	–668
5	1200	2809	2100 — 1229	–29	2809 — 3431	–622
6	1310	3697	1310 — 1229	81	3697 — 3431	266
7	1500	4022	1500 — 1229	271	4022 — 3431	591
8	1740	4677	1740 — 1229	511	4677 — 3431	1246
9	1960	6147	1960 — 1229	731	6147 — 3431	2716
10	2210	6535	2210 — 1229	981	6535 — 3431	3104
	1229	3431				
	Средние					

Для шага 3 нам также нужно вычислить квадрат разности между каждым значением каратности и ее средним значением, чтобы найти *(x – среднее x)^2*. Помимо этого, для дальнейшей минимизации нужно сложить эти значения, получив в данном случае 3 703 690 (табл. 8.10).

Таблица 8.10. Набор данных о бриллиантах и вычисления, часть 2

	Карат-ность (*x*)	**Цена** (*y*)	*x* – **среднее** *x*		*y* – **среднее** *y*		(*x* – **среднее** *x*)**^2**
1	300	339	300 — 1229	–929	339 — 3431	–3092	863 041
2	410	561	410 — 1229	–819	561 — 3431	–2870	670 761
3	750	2760	750 — 1229	–479	2760 — 3431	–671	229 441
4	910	2763	910 — 1229	–319	2763 — 3431	–668	101 761
5	1200	2809	2100 — 1229	–29	2809 — 3431	–622	841
6	1310	3697	1310 — 1229	81	3697 — 3431	266	6561
7	1500	4022	1500 — 1229	271	4022 — 3431	591	73 441
8	1740	4677	1740 — 1229	511	4677 — 3431	1246	261 121
9	1960	6147	1960 — 1229	731	6147 — 3431	2716	534 361
10	2210	6535	2210 — 1229	981	6535 — 3431	3104	962 361
	1229	3431					3 703 690
	Средние						Суммы

Для уравнения на шаге 3 не хватает только одного значения *(x – среднее x)*(y – среднее y)*. Опять же, потребуется сумма этих значений, которая составит 11 624 370 (табл. 8.11).

Таблица 8.11. Набор данных о бриллиантах и вычисления, часть 3

	Каратность (*x*)	**Цена (*y*)**	***x* – среднее *x***		***y* – среднее *y***		**(*x* – среднее *x*)^2**	**(*x* – среднее *x*) * (*y* – среднее *y*)**
1	300	339	300 — 1229	–929	339 — 3431	–3092	863 041	2 872 468
2	410	561	410 — 1229	–819	561 — 3431	–2870	670 761	2 350 530
3	750	2760	750 — 1229	–479	2760 — 3431	–671	229 441	321 409
4	910	2763	910 — 1229	–319	2763 — 3431	–668	101 761	213 092
5	1200	2809	2100 — 1229	–29	2809 — 3431	–622	841	18 038
6	1310	3697	1310 — 1229	81	3697 — 3431	266	6561	21 546
7	1500	4022	1500 — 1229	271	4022 — 3431	591	73 441	160 161
8	1740	4677	1740 — 1229	511	4677 — 3431	1246	261 121	636 706
9	1960	6147	1960 — 1229	731	6147 — 3431	2716	534 361	1 985 396
10	2210	6535	2210 — 1229	981	6535 — 3431	3104	962 361	3 045 024
	1229	3431					3 703 690	11 624 370
	Средние						Суммы	

Теперь подставим полученные значения в формулу наименьших квадратов, чтобы вычислить *m*:

```
m = 11624370 / 3703690
m = 3.139
```

Получив *m*, можно вычислить *c*, подставив средние значения *x* и *y*. Помним, что все линии регрессии будут проходить через эту точку, поэтому она является известной точкой прямой.

```
y = c + mx

3431 = c + 0.3186x
3431 = c + 391.5594
3431 – 391.5594 = c
c = 3,039.4406
```

Полное уравнение линии

```
y = 3039.4406 + 0.3186x
```

Наконец, начертим прямую, сгенерировав ряд значений для каратности в диапазоне между минимальным и максимальным значениями, а затем подставив их в уравнение (рис. 8.13).

```
x (Carat) minimum = 300
x (Carat) maximum = 2210

Значения между минимумом и максимумом с интервалом 500:
x = [300, 2210]

Подстановка значений x в уравнение линии регрессии
y = [-426 + 3.139(300) = 515.7,
     -426 + 3.139(2210) = 6511.19]

Все значения x и y:
x = [300, 2210]
y = [3981, 9975]
```

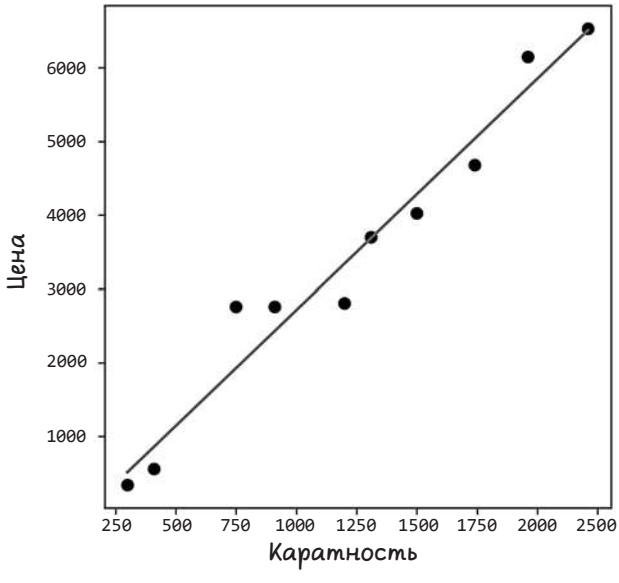


Рис. 8.13. Линия регрессии с точками данных

Итак, на основе датасета мы построили прямую линию регрессии, которая точно подходит для имеющихся данных. Тем самым мы вручную смоделировали машинное обучение.

УПРАЖНЕНИЕ: ПОСТРОИТЬ ЛИНИЮ РЕГРЕССИИ, ИСПОЛЬЗУЯ МЕТОД НАИМЕНЬШИХ КВАДРАТОВ

Следуя описанным шагам и используя предложенный датасет, постройте линию регрессии с помощью метода наименьших квадратов.

	Каратность (*x*)	**Цена (*y*)**
1	320	350
2	460	560
3	800	2760
4	910	2800
5	1350	2900
6	1390	3600
7	1650	4000
8	1700	4650
9	1950	6100
10	2000	6500

ОТВЕТ: ЛИНИЯ РЕГРЕССИИ МЕТОДОМ НАИМЕНЬШИХ КВАДРАТОВ

Сначала вычислим среднее для каждого измерения. Получаем 1253 для x и 3422 для y. На следующем шаге вычислим разность между каждым значением и его средней величиной. Далее найдем и просуммируем все квадраты разности между x и средним x, что дает 3 251 610. Наконец, разность между x и средним x умножим на разность между y и средним y и просуммируем; получим 10 566 940.

	Каратность (*x*)	**Цена (*y*)**	***x* – среднее *x***	***y* – среднее *y***	**(*x* – среднее *x*)^2**	**(*x* – среднее *x*) * (*y* – среднее *y*)**
1	320	350	–933	–3072	870 489	2 866 176
2	460	560	–793	–2862	628 849	2 269 566
3	800	2760	–453	–662	205 209	299 886
4	910	2800	–343	–622	117 649	213 346
5	1350	2900	97	–522	9409	–50 634
6	1390	3600	137	178	18 769	24 386
7	1650	4000	397	578	157 609	229 466
8	1700	4650	447	1228	199 809	548 916
9	1950	6100	697	2678	485 809	1 866 566
10	2000	6500	747	3078	558 009	2 299 266
	1253	3422			3 251 610	10 566 940

Используем эти значения для вычисления углового коэффициента *m*:

```
m = 10 566 940 / 3 251 610
m = 3.25
```

Вспомним уравнение прямой:

```
y = c + mx
```

Подставим средние для *x* и *y*, а также полученного *m*:

```
3422 = c + 3,35 * 1253
c = -775.55
```

Подставим минимальное и максимальное значения *x*, чтобы вычислить точки прямой:

```
Точка 1: используем минимальное значение для карат: x = 320

y = 775.55 + 3,25 * 320
y = 1815.55

Точка 2: используем максимальное значение для карат: x = 2000

y = 775.55 + 3.25 * 2000
y = 7275.55
```

Разобравшись с вычислениями линейной регрессии и ее использованием, можно переходить к псевдокоду.

Псевдокод

Этот код похож на рассмотренные выше шаги. Единственное, что представляет интерес, — это два цикла `for`, вычисляющие суммы значений путем перебора каждого элемента датасета:

```
fit_regression_line(carats, prices):
  let mean_X equal mean(carats)
  let mean_Y equal mean(price)
  let sum_x_squared equal 0
  for i in range(n):
    let ans equal (carats[i] - mean_X) ** 2
    sum_x_squared equal sum_x_squared + ans
  let sum_multiple equal 0
  for i in range(n):
    let ans equal (carats[i] - mean_X) * (price[i] - mean_Y)
    sum_multiple equal sum_multiple + ans
  let b1 equal sum_multiple / sum_x_squared
```

```
let b0 equal mean_Y - (b1 * mean_X)
let min_x equal min(carats)
let max_x equal max(carats)
let y1 equal b0 + b1 * min_x
let y2 equal b0 + b1 * max_x
```

Выражение для первой точки линии регрессии, заданной уравнением y = c + mx

Выражение для второй точки линии регрессии, заданной уравнением y = c + mx

Тестирование модели: определение ее точности

После определения линии регрессии ее можно использовать для прогнозирования цен на бриллианты при других значениях каратности. Измерять эффективность полученной линии регрессии мы будем с помощью новых образцов с известной ценой.

Тестировать эту модель можно на тех же данных, на которых она обучалась. Однако это не имеет смысла, невзирая на высокую точность. Обученную модель правильнее тестировать на реальных данных, которых она еще не видела.

Разделение данных на обучающие и контрольные

Обычно данные разделяются на обучающую и контрольную выборки в соотношении 80/20 (80 % на обучение и 20 % на тестирование). Используется процентное соотношение, поскольку сложно назвать точное количество образцов для обучения и тестирования. В зависимости от контекста и от задаваемых вопросов данных может требоваться как больше, так и меньше.

На рис. 8.14 и в табл. 8.12 представлен набор контрольных данных для примера с бриллиантами. Напомню, что мы масштабировали значения каратности (умножив их на 1000), чтобы сопоставить их со значениями цен. Это упростило дальнейшую работу. Точки представляют контрольные данные, а линия — обученную модель линейной регрессии.

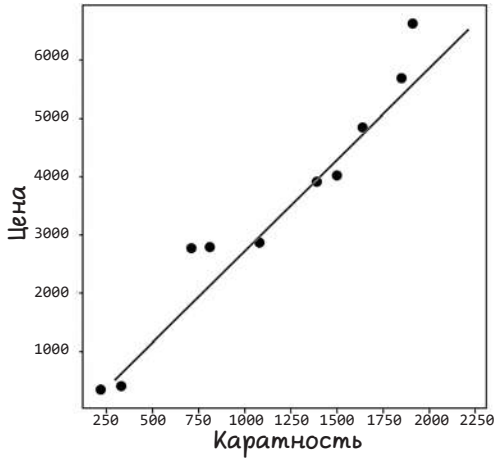


Таблица 8.12. Данные каратности и цены

	Каратность (*x*)	**Цена (*y*)**
1	220	342
2	330	403
3	710	2772
4	810	2789
5	1080	2869
6	1390	3914
7	1500	4022
8	1640	4849
9	1850	5688
10	1910	6632

Рис. 8.14. Линия регрессии и точки данных

Тестирование модели включает в себя прогнозирование на основе неизвестных обучающих данных и последующее сравнение прогнозов модели с фактическими значениями. В примере с бриллиантами нам известны фактические значения цен, так что мы определим прогнозы модели и проанализируем отличия.

Оценка эффективности предсказания прямой

В линейной регрессии для оценки точности модели вычисляется R2. R2 используется для определения расхождения между фактическим значением и спрогнозированным. Вычисляется показатель R2 следующим уравнением:

$$R^2 = \frac{\text{сумма (у прогнозное} - \text{среднее фактическое у)}^2}{\text{сумма (у фактическое} - \text{среднее фактическое у)}^2}$$

Первым делом так же, как и на этапе обучения, необходимо вычислить среднее фактических значений цены, расстояния между фактическими значениями цены и средним этих цен, а затем рассчитать квадраты полученных значений. Мы используем значения точек с рис. 8.14 (табл. 8.13).

Таблица 8.13. Набор данных о бриллиантах и вычисления, часть 1

	Каратность (*x*)	**Цена (*y*)**	***y* – среднее *y***	**(*y* – среднее *y*)^2**
1	220	*342*	–3086	9 523 396
2	330	*403*	–3025	9 150 625
3	710	*2772*	–656	430 336
4	810	*2789*	–639	408 321
5	1080	*2869*	–559	312 481
6	1390	*3914*	486	236 196
7	1500	*4022*	594	352 836
8	1640	*4849*	1421	2 019 241
9	1850	*5688*	2260	5 107 600
10	1910	*6632*	3204	10 265 616
		3428		37 806 648
		Среднее		Сумма

Далее вычисляем прогнозируемое значение цены для каждого значения каратности, возводим результаты в квадрат и складываем их (табл. 8.14).

Таблица 8.14. Набор данных о бриллиантах и вычисления, часть 2

	Каратность (*x*)	**Цена (*y*)**	***y* – среднее *y***	**(*y* – среднее *y*)^2**	**Прогнозируемое *y***	**Прогнозируемое *y* – среднее *y***	**(Прогнозируемое *y* – среднее *y*)^2**
1	220	342	–3086	9 523 396	264	–3164	10 009 876
2	330	403	–3025	9 150 625	609	–2819	7 944 471
3	710	2772	–656	430 336	1802	–1626	2 643 645
4	810	2789	–639	408 321	2116	–1312	1 721 527
5	1080	2869	–559	312 481	2963	–465	215 900
6	1390	3914	486	236 196	3936	508	258 382
7	1500	4022	594	352 836	4282	854	728 562
8	1640	4849	1421	2 019 241	4721	1293	1 671 748
9	1850	5688	2260	5 107 600	5380	1952	3 810 559
10	1910	6632	3204	10 265 616	5568	2140	4 581 230
		3428		37 806 648			33 585 901
		Среднее		Сумма			Сумма

Используя сумму квадратов разности между прогнозируемой ценой и средним, а также сумму квадратов разности между фактической ценой и средней, можно вычислить показатель R^2:

$$R^2 = \frac{\text{сумма (y прогнозное} - \text{среднее фактическое y)}^2}{\text{сумма (y фактическое} - \text{среднее фактическое y)}^2}$$

$$R^2 = 33585901 / 37806648$$
$$R^2 = 0.88$$

Результат 0.88 означает, что модель на 88 % точно прогнозирует новые данные. Это говорит о достаточно высокой точности модели. Для примера с бриллиантами результат можно считать приемлемым. Определение, достаточна ли точность для той или иной задачи, зависит от предметной области решаемой задачи. Эффективность моделей машинного обучения мы будем рассматривать в следующем разделе.

Для более полного ознакомления с подгонкой линий регрессии к данным предлагаем прочесть соответствующую главу книги «Math for Programmers» издательства Manning по ссылке http://mng.bz/Ed5q.

Связи между несколькими значениями, например каратности, цены и огранки бриллиантов, можно найти с помощью так называемой *множественной регрессии*. Этот процесс несколько усложняет вычисления, но основной принцип сохраняется.

Повышение точности

После обучения модели и оценки ее качества в отношении новых данных у нас формируется понимание ее эффективности. Нередко получается, что модель работает не так, как хотелось, поэтому требуется ее доработка. Доработка включает в себя повторение различных шагов жизненного цикла машинного обучения (рис. 8.15).

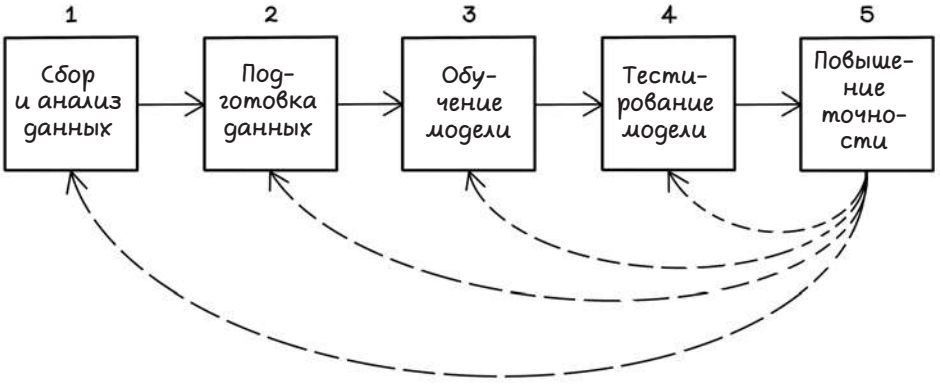


Рис. 8.15. Жизненный цикл машинного обучения

Возможно, после получения результатов мы поймем, что требуется улучшить точность модели. МО подразумевает постоянное экспериментирование, в ходе которого на разных этапах тестируются разные тактики, и только после этого выбирается лучший подход. Если бы в примере с бриллиантами наша модель дала низкий показатель точности, то для его повышения можно было бы совместно со значением каратности использовать измерение, отражающее размер бриллианта. Вот несколько способов повышения точности модели:

- *Собрать дополнительные данные*. Можно расширить датасет, дополнив его либо данными, подобными уже имеющимся, либо включив те данные, которые ранее для него не рассматривались.
- *Подготовить данные иначе*. Иногда помогает изменение подхода к подготовке данных, так как, возможно, была выбрана не самая оптимальная стратегия. Попробуйте использовать другие техники для нахождения недостающих значений, замены неоднозначных данных и кодирования категориальных.
- *Выбрать другие признаки данных*. Другие признаки из датасета могут лучше подойти для прогнозирования зависимой переменной. Например, значение измерения X может стать удачным выбором для прогнозирования значения площадки, так как они непосредственно связаны, как

видно из рис. 8.5. А вот прогнозировать чистоту на основе измерения X бессмысленно.

- *Использовать другой алгоритм для обучения модели.* Иногда выбранный алгоритм оказывается не самым подходящим для решаемой задачи или данных. Для разных целей можно пробовать разные варианты, о чем пойдет речь в следующем разделе.
- *Учесть ложноположительные тесты.* Тесты бывают обманчивы. В процессе обучения модель может показывать хороший результат, но на незнакомых данных давать низкую точность. Причиной иногда оказывается переобучение модели на конкретных данных. *Переобучение* означает, что модель слишком хорошо изучила обучающий датасет и стала недостаточно гибкой в отношении новых данных с большей дисперсией. Происходит это чаще всего в задачах классификации, с которыми мы также подробнее познакомимся в следующем разделе.

Если с помощью линейной регрессии не удалось добиться полезных результатов либо нужно найти ответ на другой вопрос, можно попробовать другие алгоритмы. Далее описаны те из них, которые используются, когда вопрос по своей природе является комплексным.

Классификация с помощью деревьев решений

Если в двух словах, задачи классификации заключаются в присвоении метки образцу на основе его свойств. По своей сути они отличаются от задач регрессии, в которых прогнозируется значение.

Задачи классификации: выбор одного из двух

Мы узнали, что регрессия — это прогнозирование значения на основе одной или более переменных. В качестве примера мы рассмотрели прогноз стоимости бриллианта на основе его веса в каратах. Классификация похожа на этот процесс тем, что стремится спрогнозировать значение, но делает это не для непрерывных величин, а для дискретных классов. Дискретные значения — это категориальные признаки. В наборе данных о бриллиантах это огранка, цвет или чистота. Вот еще один пример. Предположим, у нас есть несколько автомобилей — легковых и грузовиков. Мы будем измерять вес и количество колес каждого автомобиля. Внешние различия не будут браться в расчет. Практически все легковые машины имеют по четыре колеса, в то время как у грузовиков колес обычно больше. Как правило, грузовики тяжелее легковых авто, но крупный внедорожник может весить как небольшой грузовичок. Чтобы предсказать, какому типу принадлежит рассматриваемый автомобиль — легковой он или грузовой, — можно найти связи между его весом и количеством колес (рис. 8.16).

Рис. 8.16. Образцы автомобилей для потенциальной классификации на основе количества колес и веса

УПРАЖНЕНИЕ: РЕГРЕССИЯ ИЛИ КЛАССИФИКАЦИЯ?

Рассмотрите следующие сценарии и определите, какой из них относится к задаче регрессии, а какой к задаче классификации:

1. Среди собранных данных о крысах имеется два признака: средняя продолжительность жизни и ожирение. Задача — найти корреляцию между этими признаками.
2. Среди данных о животных есть информация о весе каждой особи и наличии у нее крыльев. Задача — определить, какие из животных являются птицами.
3. Среди данных о компьютерных устройствах есть информация о размере экрана, весе и операционной системе нескольких устройств. Задача — определить, какие устройства являются планшетами, какие ноутбуками, а какие смартфонами.
4. Среди данных о погоде представлены значения количества осадков и влажности. Задача — определить показатель влажности в разные сезоны дождей.

ОТВЕТ: РЕГРЕССИЯ ИЛИ КЛАССИФИКАЦИЯ

1. *Регрессия* — изучается связь двух переменных. Продолжительность жизни — зависимая переменная, ожирение — независимая.
2. *Классификация* — образец классифицируется как птица или не птица, для чего рассматриваются его признаки — вес и наличие крыльев.
3. *Классификация* — образец классифицируется как планшет, ноутбук или смартфон на основе имеющихся характеристик.
4. *Регрессия* — изучается связь между количеством осадков и показателем влажности. Влажность выступает зависимой переменной, а количество осадков независимой.

Основа дерева решений

В задачах классификации и регрессии используются различные алгоритмы. К наиболее популярным относится метод опорных векторов, дерево решений и случайный лес. В текущем разделе будет рассмотрено применение алгоритма дерева решений в задаче классификации.

Дерево решений — это структура, описывающая серию выборов, совершаемых на пути к итоговому решению задачи (рис. 8.17). Если мы, например, раздумываем, стоит ли идти на улицу в шортах, то для принятия окончательного решения мы принимаем серию промежуточных. Будет ли день холодным? Если нет, то задержимся ли мы на улице допоздна, когда температура понизится? В итоге мы можем решить надеть шорты в теплый день, но не в том случае, если собираемся возвращаться домой ночью.

Рис. 8.17. Пример простого дерева решений

С помощью дерева решений попробуем в примере с бриллиантом спрогнозировать качество его огранки на основе значений каратности и цены. Чтобы упростить пример, предположим, что являемся торговцем бриллиантами, которого не сильно заботит тщательность оценки качества огранки, и сгруппируем разные виды огранки в более общие категории. «Удовлетворительная» и «Хорошая» объединим в категорию «Нормальная», а «Очень хорошая», «Премиальная» и «Идеальная» — в категорию «Отличная».

1	Удовлетворительная	1	Нормальная
2	Хорошая		
3	Очень хорошая	2	Отличная
4	Премиальная		
5	Идеальная		

В табл. 8.15 показан получившийся датасет.

Таблица 8.15. Датасет для примера классификации

	Каратность	**Цена**	**Огранка**
1	0.21	327	Нормальная
2	0.39	897	Отличная
3	0.50	1122	Отличная
4	0.76	907	Нормальная
5	0.87	2757	Отличная
6	0.98	2865	Нормальная
7	1.13	3045	Отличная
8	1.34	3914	Отличная
9	1.67	4849	Отличная
10	1.81	5688	Отличная

Глядя на представленные значения в поисках закономерностей, можно кое-что заметить. Цена резко повышается после 0.98 карат, и ее повышенное значение коррелирует с отличной огранкой. При этом бриллианты с меньшими значениями каратности коррелируют с нормальным показателем качества. Тем не менее у образца 3 с отличным качеством огранки значение каратности небольшое. На рис. 8.18 показано, что получится, если задать вопросы для фильтрации данных и категоризировать их вручную. Обратите внимание, что узлы деревьев решений содержат вопросы, а вершины содержат разделяемые по категориям образцы.

На основе этого небольшого датасета получилось легко разбить бриллианты по категориям вручную. Тем не менее в реальных датасетах нужно прорабатывать тысячи образцов, возможно, с тысячами признаков, что делает невозможной ручную обработку. Именно здесь на помощь приходит алгоритм дерева решений, который способен формулировать вопросы и фильтровать образцы. Такое дерево находит закономерности, которые может упустить человек, и более точно выполняет фильтрацию.

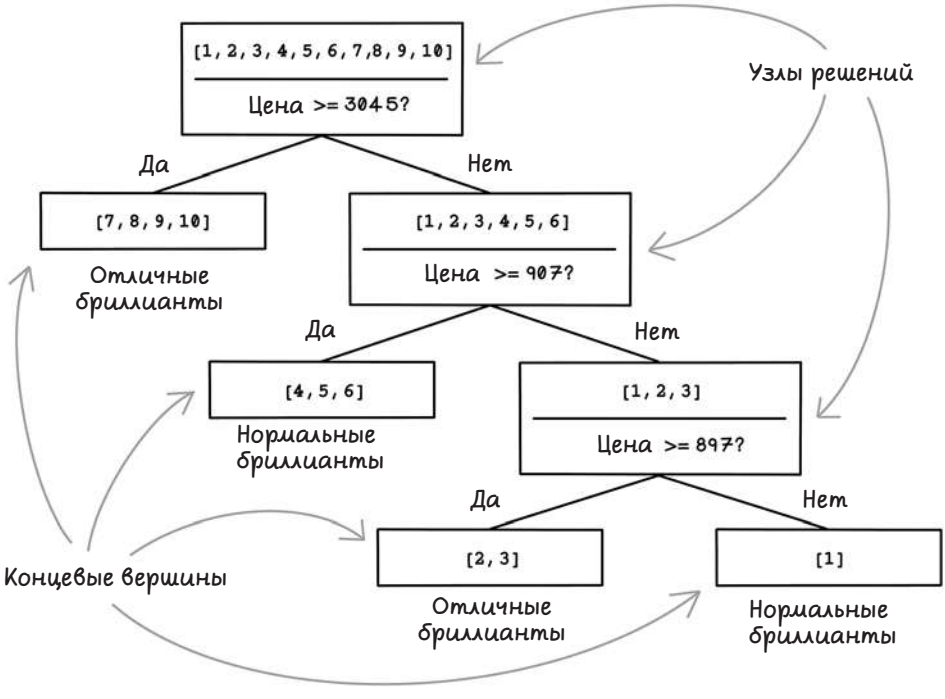


Рис. 8.18. Пример дерева решений, построенного на основе человеческого понимания

Обучение дерева решений

Чтобы создать дерево, обладающее достаточным интеллектом для принятия правильных решений в ходе классификации бриллиантов, потребуется подходящий обучающий алгоритм. Для обучения дерева решений существует целое семейство алгоритмов, и мы возьмем один из них, называемый CART (Classification And Regression Tree, дерево классификации и регрессии). В основе этого и других алгоритмов обучения деревьев лежит выяснение, какие вопросы задавать и когда, чтобы оптимально фильтровать образцы по соответствующим категориям. В примере с бриллиантами алгоритм должен выучить лучшие вопросы о значениях каратности и цены, а также понять, когда их задавать, чтобы верно выделить нормальные и отличные бриллианты.

Структуры данных для дерева решений

Чтобы понять, как дерево принимает решения, рассмотрим следующие структуры, которые организуют логику и данные подходящим для алгоритма образом:

- *Карта классов/группировка меток.* Карта представляет собой набор пар «ключ-значение» элементов, в котором не могут находиться два одина-

ковых ключа. Эта структура используется для хранения образцов, соответствующих конкретной метке, и пригодится для хранения значений, необходимых для вычисления энтропии, также известной как *неопределенность*. Вскоре мы рассмотрим это понятие подробнее.

- *Дерево узлов*. Как показано на предыдущем изображении дерева на рис. 8.18, оно строится путем объединения нескольких узлов. Этот пример может напомнить содержание одной из предыдущих глав. Узлы в дереве служат для фильтрации/распределения образцов по категориям:
  - *Узел решения* — узел, в котором датасет делится или фильтруется.
    - Вопрос: Какой задается вопрос? (см. ниже).
    - Верные образцы: образцы, дающие положительный ответ на вопрос.
    - Ложные образцы: образцы, дающие отрицательный ответ.
- *Концевая вершина* — узел, содержащий только список образцов. Все образцы в этом узле категоризованы верно.
- *Вопрос* — вопрос, который в зависимости от степени гибкости можно сформулировать по-разному. Можно спросить: «Значение каратности > 0.5 и < 1.13?» Чтобы было проще понять пример, скажем так: вопрос — это сочетание переменного признака, переменного значения и оператора >=: «Значение каратности >= 0.5?» или «Цена >= 3.045 ?»
  - *Признак* — признак, к которому задается вопрос.
  - *Значение* — постоянное значение, по отношению к которому сравниваемое значение должно быть либо больше, либо равно.

Жизненный цикл обучения дерева решений

В этом разделе рассматривается, как алгоритм фильтрует данные на основе решений, чтобы правильно классифицировать содержимое датасета. Рисунок 8.19 демонстрирует все этапы обучения дерева решений. Далее в этом разделе мы рассмотрим их подробно.

При построении дерева решений тестируются все возможные вопросы, чтобы установить, какой из них окажется лучшим в определенной точке ветвления дерева. Для проверки вопроса используется принцип *энтропии* — измерения неопределенности датасета. Если у нас есть 5 отличных бриллиантов и 5 нормальных и мы пробуем случайным образом выбрать отличный из всех десяти, то каковы шансы, что выбранный бриллиант окажется отличным (рис. 8.20)?

Возьмем начальный набор данных о бриллиантах с признаками каратности, цены и огранки. Неопределенность этого датасета можно узнать с помощью коэффициента Джини. Коэффициент Джини, равный 0, означает, что неопределенность в датасете отсутствует. Это может, например, означать, что все бриллианты отличные. Рисунок 8.21 показывает, как этот коэффициент вычисляется.

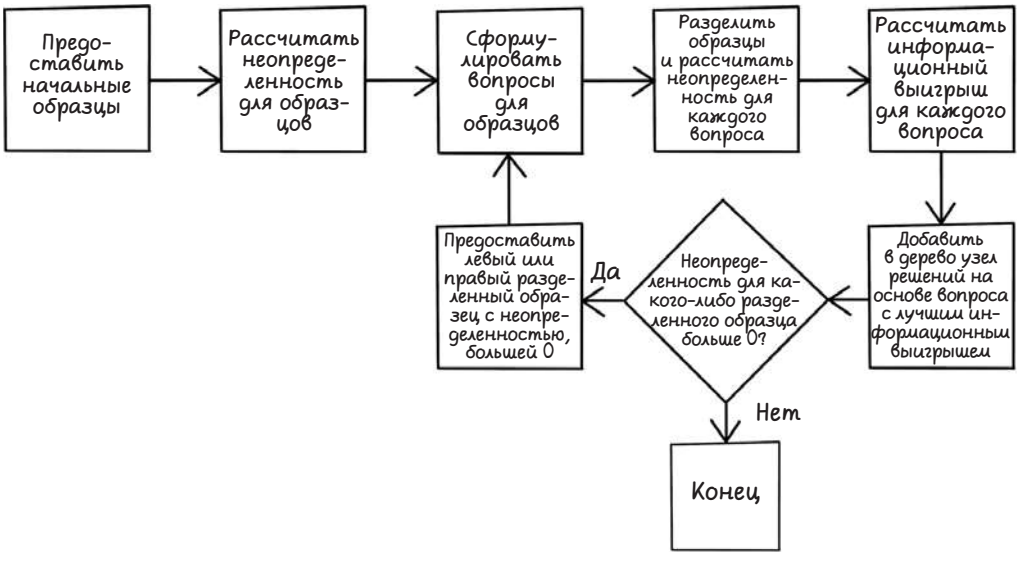
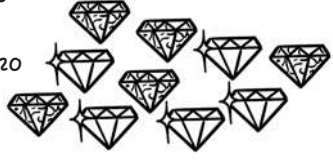


Рис. 8.19. Порядок построения дерева решений

5 нормальных бриллиантов
5 отличных бриллиантов
5 + 5 = 10 бриллиантов всего



Шанс получения отличного бриллианта:
5 / 10 = неопределенность 50 %

Рис. 8.20. Пример неопределенности

	Каратность	Цена	Огранка
1	0.21	327	Нормальная
2	0.39	897	Отличная
3	0.50	1122	Отличная
4	0.76	907	Нормальная
5	0.87	2757	Нормальная
6	0.98	2865	Нормальная
7	1.13	3045	Нормальная
8	1.34	3914	
9	1.67	4849	Отличная
10	1.81	5688	Отличная

5
5

Коэффициент Джини =
= 1 – (кол-во нормальных / общее кол-во)^2 +
+ (кол-во отличных / общее кол-во)^2

Коэффициент Джини = $1 - (5 / 10)^2 + (5 / 10)^2$
Коэффициент Джини = $1 - (0.5)^2 + (0.5)^2$
Коэффициент Джини = 1 - 0.5
Коэффициент Джини = 0.5

Рис. 8.21. Вычисление коэффициента Джини

Коэффициент Джини равен 0.5, значит, при случайном выборе есть 50%-ный шанс получить образец с неверной меткой, как показывалось на рис. 8.20.

Следующий этап — создание узла решений для деления данных. Этот узел содержит вопрос, который позволит грамотно разделить данные и снизить неопределенность. Помним, что 0 означает ее отсутствие. Мы стремимся разделить датасет на подвыборки с нулевой неопределенностью.

Для определения оптимального результата деления данных генерируется множество вопросов, основанных на каждом признаке каждого образца. Так как в нашем примере есть 2 признака и 10 образцов, всего сгенерируется 20 вопросов. На рис. 8.22 показаны некоторые из них — простые вопросы о том, является ли значение признака больше или равным определенной величине.

Неопределенность в датасете определяется *индексом Джини*, и с помощью вопросов этот показатель снижается. *Энтропия* — это еще один показатель, который, используя индекс Джини, отражает степень беспорядка конкретного разделения данных на основе заданного вопроса. Чтобы выяснить, насколько вопрос снижает неопределенность, понадобится оценить его информационный выигрыш. *Информационный выигрыш* описывает количество информации, полученной на основе конкретного вопроса. Чем выше выигрыш, тем ниже неопределенность.

Информационный выигрыш рассчитывается вычитанием значения энтропии, полученного после ответа на вопрос, из ее исходного значения. Процесс следующий:

1. Разделить датасет, задав вопрос.
2. Измерить индекс Джини для левой части.
3. Измерить энтропию для левой части в сравнении с датасетом до деления.
4. Измерить индекс Джини для правой части.
5. Измерить энтропию для правой части в сравнении с датасетом до деления.
6. Вычислить общую энтропию, сложив ее показатели для левой и правой частей.
7. Рассчитать информационный выигрыш, вычтя полученное значение общей энтропии из исходного.

На рис. 8.23 показано разделение данных и информационный выигрыш для вопроса «Цена >= 3914?»

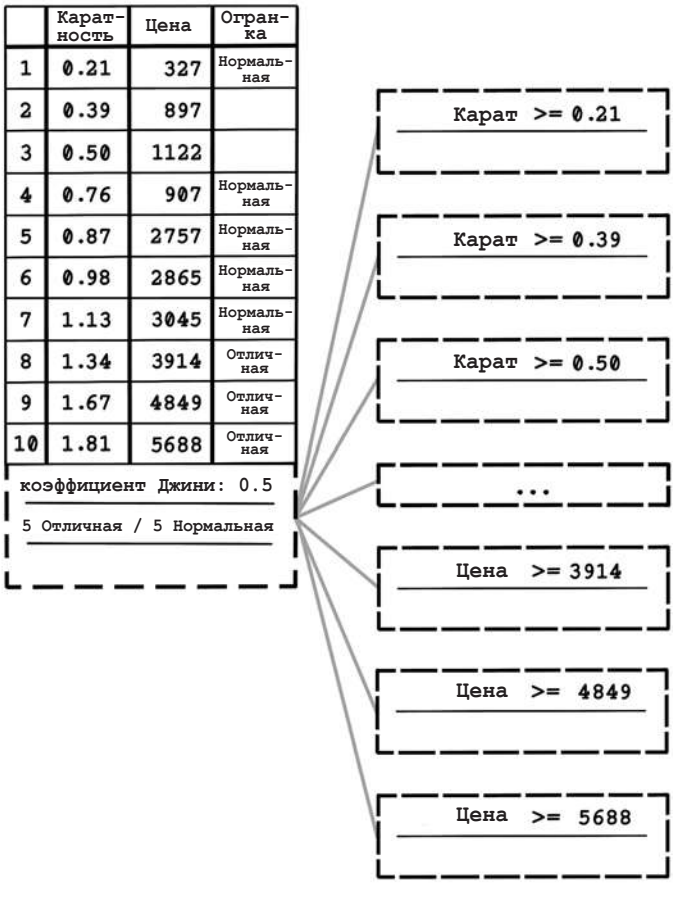
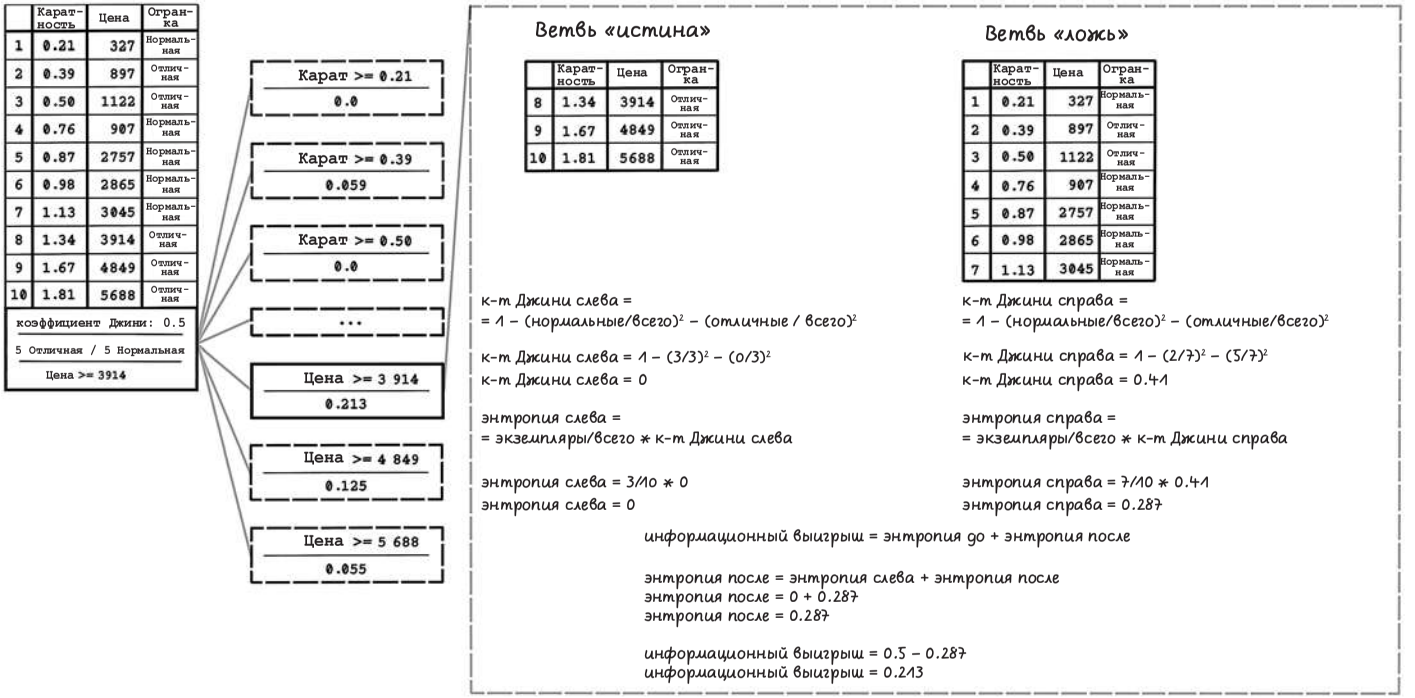


Рис. 8.22. Пример вопросов, задаваемых для деления данных с помощью дерева решений

В примере на рис. 8.23 вычисляется информационный выигрыш для всех вопросов, после чего в качестве оптимального вопроса для точки ветвления дерева выбирается тот, чей выигрыш оказывается наилучшим. Далее исходный датасет разделяется на основе этого узла решения с вопросом «Цена >= 3914?». Узел решения, содержащий этот вопрос, добавляется в дерево, и на его основе происходит деление на правую и левую ветки.



	Карат-ность	Цена	Огран-ка
1	0.21	327	Нормаль-ная
2	0.39	897	Отлич-ная
3	0.50	1122	Отлич-ная
4	0.76	907	Нормаль-ная
5	0.87	2757	Нормаль-ная
6	0.98	2865	Нормаль-ная
7	1.13	3045	Нормаль-ная
8	1.34	3914	Отлич-ная
9	1.67	4849	Отлич-ная
10	1.81	5688	Отлич-ная

	Карат-ность	Цена	Огран-ка
8	1.34	3914	Отлич-ная
9	1.67	4849	Отлич-ная
10	1.81	5688	Отлич-ная

	Карат-ность	Цена	Огран-ка
1	0.21	327	Нормаль-ная
2	0.39	897	Отлич-ная
3	0.50	1122	Отлич-ная
4	0.76	907	Нормаль-ная
5	0.87	2757	Нормаль-ная
6	0.98	2865	Нормаль-ная
7	1.13	3045	Нормаль-ная

Рис. 8.23. Демонстрация разделения данных и информационного выигрыша на основе вопроса

На рис. 8.24 после разделения датасета левая часть содержит чистую выборку отличных бриллиантов, а правая — смешанную выборку, состоящую из двух отличных и пяти нормальных образцов. Теперь, чтобы продолжать деление, к правой части нужно задать еще один вопрос, для чего на основе признаков каждого образца правой выборки снова генерируется несколько их вариантов.

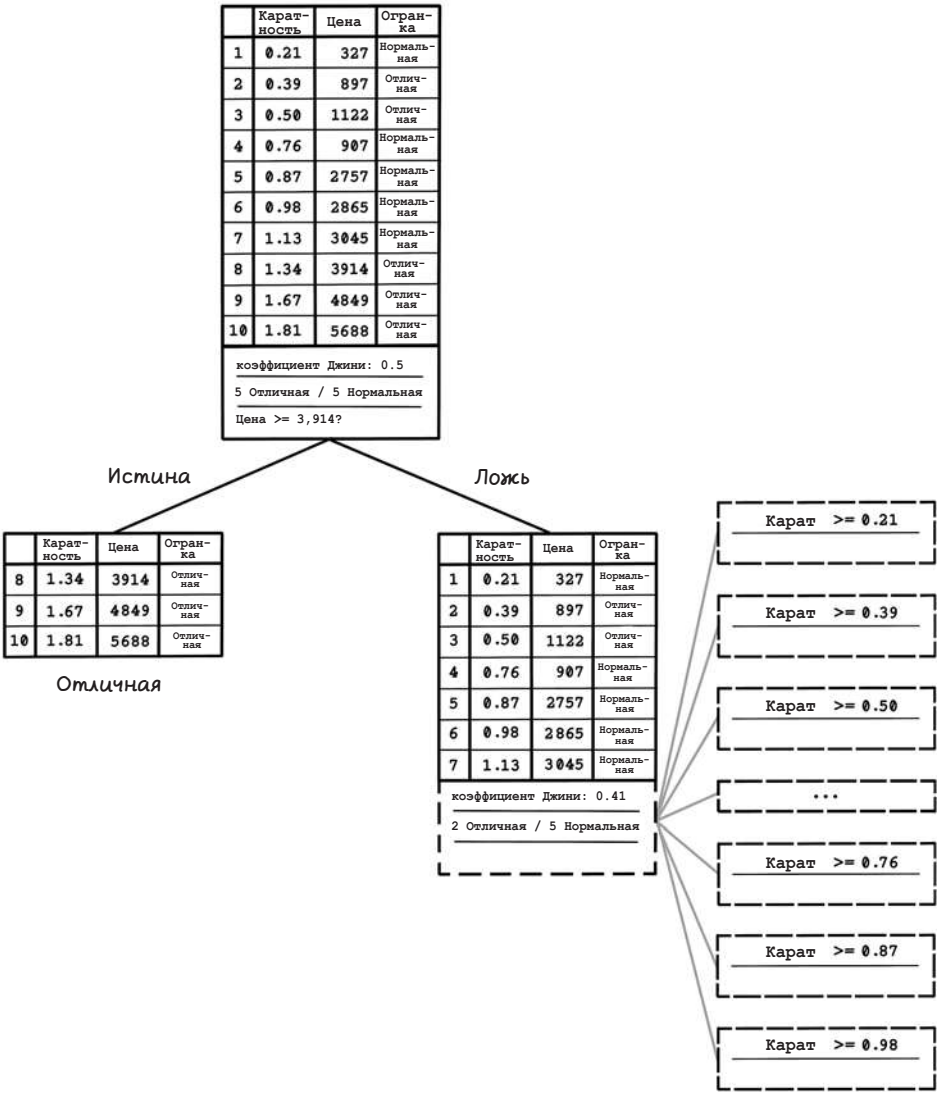


Рис. 8.24. Дерево после первого узла решений и возможных вопросов

УПРАЖНЕНИЕ: ВЫЧИСЛИТЕ НЕОПРЕДЕЛЕННОСТЬ И ИНФОРМАЦИОННЫЙ ВЫИГРЫШ ДЛЯ ВОПРОСА

Руководствуясь примером на рис. 8.23, вычислите информационный выигрыш для вопроса «Значение каратности >= 0.76?».

ОТВЕТ: НЕОПРЕДЕЛЕННОСТЬ И ИНФОРМАЦИОННЫЙ ВЫИГРЫШ

В решении, показанном на рис. 8.25, повторно использован шаблон вычислений, которые определяют энтропию и информационный выигрыш на основе вопроса. Попрактикуйтесь с другими вопросами и сравните результаты со значениями выигрыша на рисунке.

Процесс разделения данных, генерации вопросов и определения информационного выигрыша выполняется рекурсивно, пока датасет не будет полностью категоризирован. На рис. 8.26 приводится готовое дерево решений, содержащее все заданные вопросы и результаты деления.

Важно отметить, что деревья решений обычно обучаются на гораздо более крупных датасетах. Задаваемые вопросы должны быть более общими, чтобы охватывать более широкий спектр данных, в связи с чем для обучения требуется больше образцов.

Псевдокод

При программировании дерева решений с нуля необходимо начать с подсчета количества образцов в каждом классе — в данном случае количества нормальных бриллиантов и отличных:

```
find_unique_label_counts(examples):
  let class_count equal empty map
  for example in examples:
    let label equal example['quality']
    if label not in class_count:
      let class_count[label] equal 0
    class_count[label] equal class_count[label] + 1
  return class_count
```

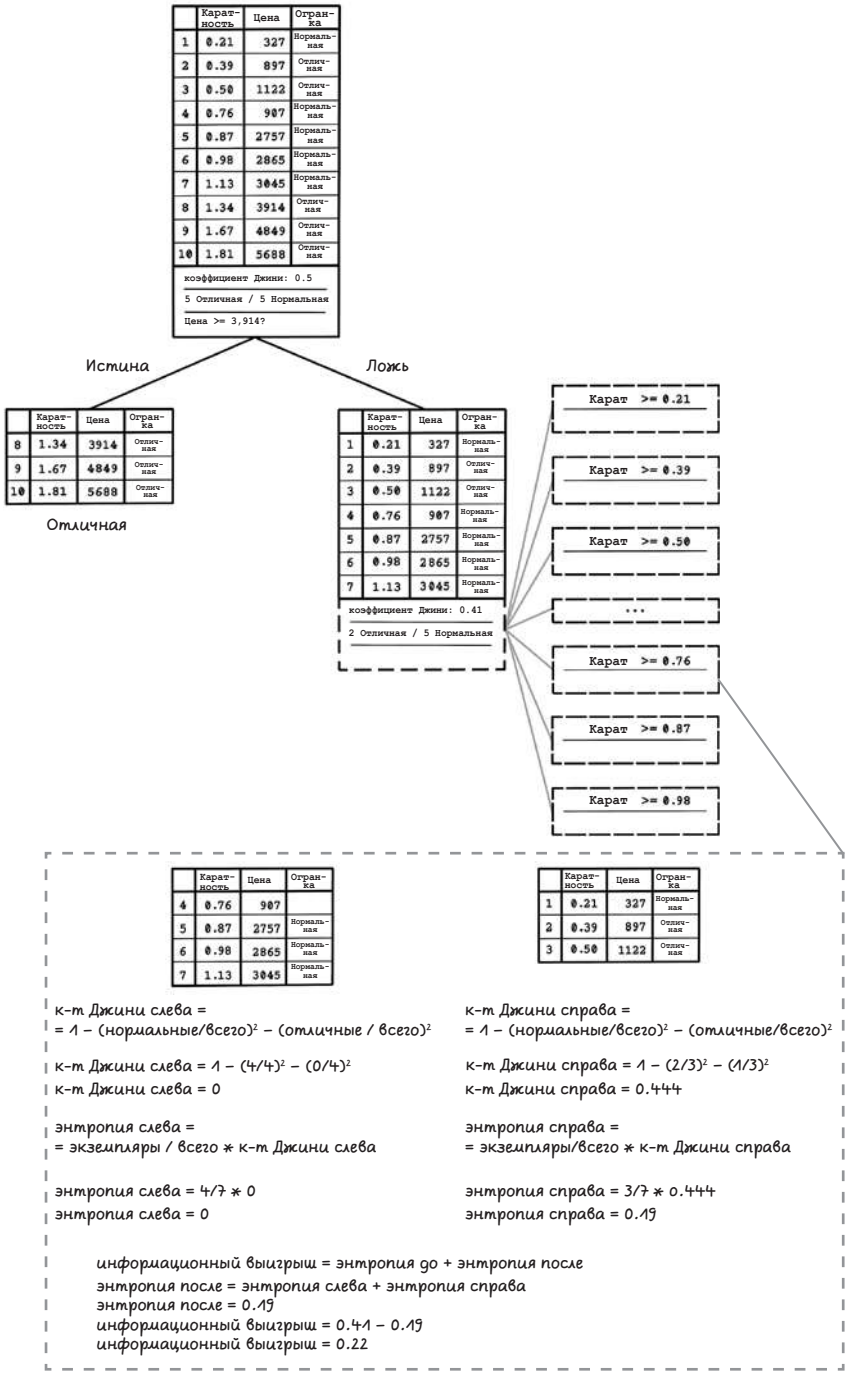


Рис. 8.25. Иллюстрация разделения данных и вычисления информационного выигрыша на основе вопроса второго уровня

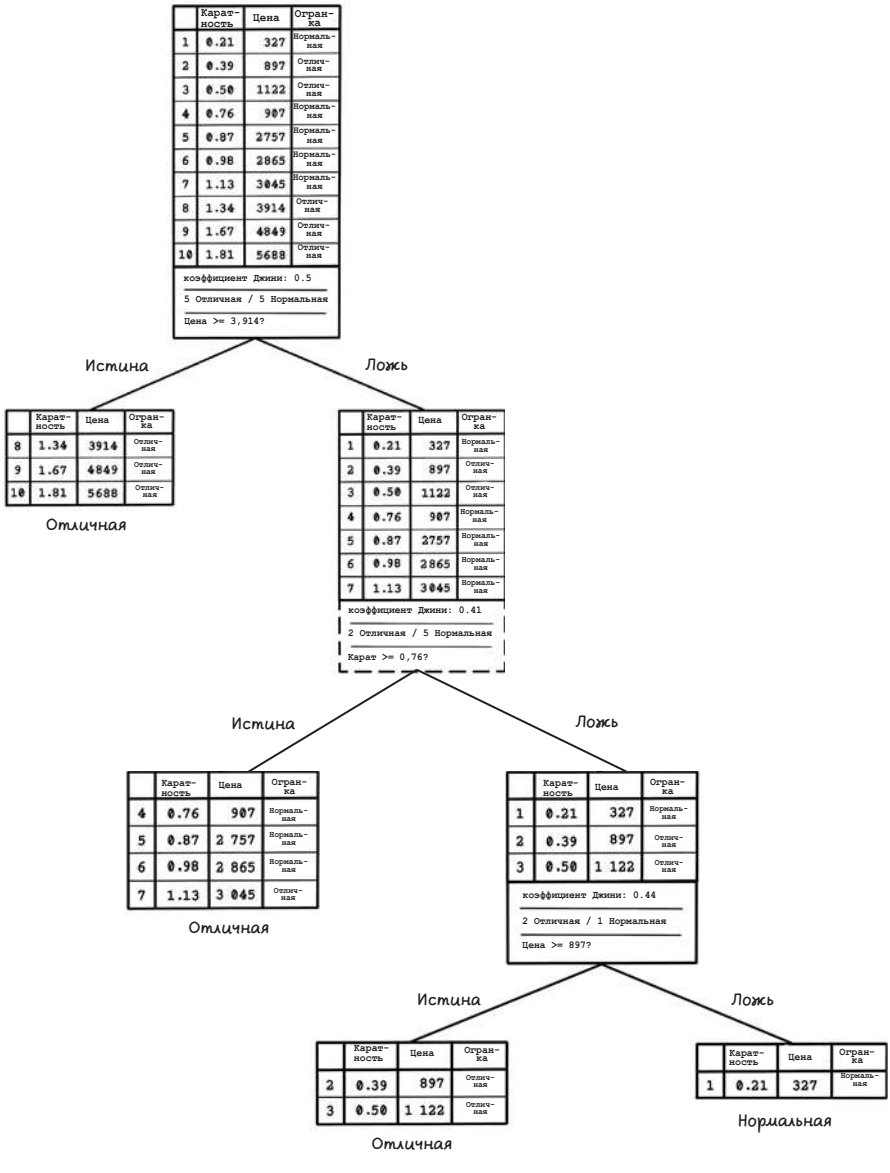


Рис. 8.26. Готовое обученное дерево решений

Далее образцы разделяются на основе вопроса. Те, которые отвечают на вопрос положительно, сохраняются в `examples_true`, а остальные в `examples_false`:

```
split_examples(examples, question):
  let examples_true equal empty array
  let examples_false equal empty array
  for example in examples:
    if question.filter(example):
      append example to examples_true
    else:
      append example to examples_false
  return examples_true, examples_false
```

Нам нужна функция, вычисляющая коэффициент Джини для набора образцов. Нижеприведенный вариант такой функции вычисляет его с помощью метода, описанного на рис. 8.23:

```
calculate_gini(examples):
  let label_counts equal find_unique_label_counts(examples)
  let uncertainty equal 1
  for label in label_counts:
    let probability_of_label equal label_counts[label] / length(examples))
    uncertainty equals uncertainty - probability_of_label ^ 2
  return uncertainty
```

Оператор `information_gain` использует левую и правую части, а также текущую неопределенность, чтобы рассчитать информационный выигрыш:

```
calculate_information_gain(left, right, current_uncertainty):
  let total equal length(left) + length(right)
  let left_gini equal calculate_gini(left)
  let left_entropy equal length(left) / total * left_gini
  let right_gini equal calculate_gini(right)
  let right_entropy equal length(right) / total * right_gini
  let uncertainty_after equal left_entropy + right_entropy
  let information_gain equal current_uncertainty - uncertainty_after
  return information_gain
```

Следующая функция может выглядеть сложной, но она просто перебирает все признаки в наборе данных и их значения, вычисляя максимальный информационный выигрыш и выбирая лучший вопрос:

```
find_best_split(examples, number_of_features):
  let best_gain equal 0
  let best_question equal None
  let current_uncertainty equal calculate_gini(examples)
  for feature_index in range(number_of_features):
    let values equal [example[feature_index] for example in examples]
    for value in values:
      let question equal Question(feature_index, value)
      let true_examples, false_examples equal
        split_examples(examples, question)
      if length(true_examples) != 0 or length(false_examples) != 0:
        let gain equal calculate_information_gain
            (true_examples, false_examples, current_uncertainty)
        if gain >= best_gain:
          best_gain, best_question equal gain, question
  return best_gain, best_question
```

Следующая функция объединяет полученные значения в дереве решений:

```
build_tree(examples, number_of_features):
  let gain, question equal find_best_split(examples, number_of_features)
  if gain == 0:
    return ExamplesNode(examples)
  let true_examples, false_examples equal split_examples(examples, question)
  let true_branch equal build_tree(true_examples)
  let false_branch equal build_tree(false_examples)
  return DecisionNode(question, true_branch, false_branch)
```

Обратите внимание, что это рекурсивная функция. Она разделяет данные и рекурсивно делит результирующие выборки, пока не перестанет получать информационный выигрыш. Это будет означать, что дальнейшее деление невозможно. Помним, что узлы решений используются для деления образцов, а концевые вершины — для хранения их отдельных наборов.

Итак, мы узнали, как строить дерево решений для классификации. Помните, что обученная модель дерева решений, подобно подходу с линейной регрессией, будет тестироваться на незнакомых данных.

Необходимо отметить, что для деревьев решений свойственна проблема переобучения, когда модель слишком хорошо обучилась на нескольких образ-

цах и дает низкие результаты на новых. Это случается, когда модель заучивает закономерности обучающих (тренировочных) данных, но, так как реальные данные оказываются отличными от уже знакомых ей примеров, она не может так же качественно их разделить. Как правило, модель со 100%-ной точностью оказывается переобученной. Такая идеальная модель классифицирует новые образцы неверно, так как она оказывается слишком подстроенной под тренировочные данные, чтобы обрабатывать другие случаи. Переобучение может произойти с любой моделью МО, не только с деревьями решений.

Рисунок 8.27 иллюстрирует, что происходит при переобучении. Недообученная модель допускает в классификации слишком много ошибок на тренировочных данных, а переобученная либо делает их слишком мало, либо не делает вообще. Норма находится где-то посередине.

Рис. 8.27. Недообучение, норма и переобучение

Классификация образцов с помощью дерева решений

После определения правильных вопросов и обучения дерева решений можно протестировать его на новых данных.

Для этого нам потребуются данные из датасета, еще не знакомые модели, включая признак огранки (рис. 8.16). При этом нужно знать их метки. Предоставив модели контрольные данные, мы оценим, насколько точно она их классифицирует.

Таблица 8.16. Набор данных о бриллиантах для классификации

	Каратность	**Цена**	**Огранка**
1	0.26	689	Отличная
2	0.41	967	Отличная
3	0.52	1012	Отличная
4	0.76	907	Нормальная
5	0.81	2650	Нормальная
6	0.90	2634	Нормальная

	Каратность	Цена	Огранка
7	1.24	2999	Отличная
8	1.42	3850	Отличная
9	1.61	4345	Отличная
10	1.78	3100	Нормальная

На рис. 8.28 показана обученная нами модель дерева решений, которая будет поочередно классифицировать образцы.

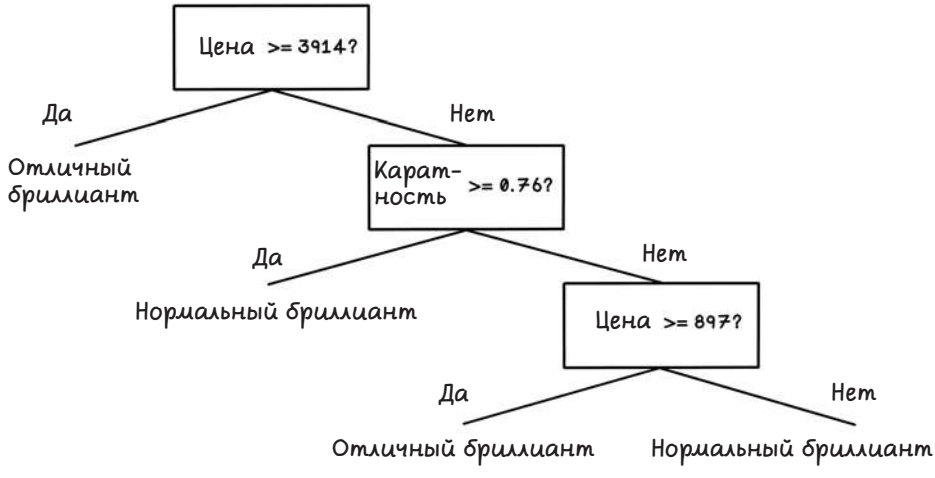


Рис. 8.28. Модель дерева решений, которая будет обрабатывать новые образцы

Результаты классификации приведены в табл. 8.17. Предположим, что целью было прогнозирование нормальных бриллиантов. Заметьте, что три образца были классифицированы неверно. Результат равен 3 из 10, то есть модель верно спрогнозировала 7 из 10 (70 %) контрольных значений. Это не самый плохой показатель точности, но он демонстрирует возможность ошибок в классификации.

Таблица 8.17. Набор данных о бриллиантах для классификации и прогнозирования

	Карат	Цена	Огранка	Прогноз	
1	0.26	689	Нормальная	Нормальная	✓
2	0.41	880	Отличная	Отличная	✓
3	0.52	1012	Отличная	Отличная	✓
4	0.76	907	Нормальная	Нормальная	✓

	Карат	Цена	Огранка	Прогноз	
5	0.81	2650	Нормальная	Нормальная	✓
6	0.90	2634	Нормальная	Нормальная	✓
7	1.24	2999	**Отличная**	**Нормальная**	†
8	1.42	3850	**Отличная**	**Нормальная**	†
9	1.61	4345	Отличная	Отличная	✓
10	1.78	3100	**Нормальная**	**Отличная**	†

Для измерения эффективности модели в отношении контрольных данных обычно используется матрица ошибок. *Матрица ошибок* описывает эффективность с помощью следующих метрик (рис. 8.29):

- *Истинно положительные (TP, true positive)* — образцы, верно классифицированные как нормальные.
- *Истинно отрицательные (TN, true negative)* — образцы, верно классифицированные как отличные.
- *Ложноположительные (FP, false positive)* — отличные образцы, классифицированные как нормальные.
- *Ложноотрицательные (FN, false negative)* — нормальные образцы, классифицированные как отличные.

	Прогнозно положительные	Прогнозно отрицательные	
Фактически положительные	Истинно положительные **TP**	Ложноотрицательные **FN**	Чувствительность **TP / TP + FN**
Фактически отрицательные	Ложноположительные **FP**	Истинно отрицательные **TN**	Специфичность **TN / TN + FP**
	Точность **TP / TP + FP**	Точность отрицательного класса	Общая точность $\frac{\mathbf{TP + TN}}{\mathbf{TP + TN + FP + FN}}$

Рис. 8.29. Матрица ошибок

Результаты тестирования модели на незнакомых образцах можно использовать для получения ряда выводов:

- *Точность* (precision) — как часто хорошие образцы классифицируются верно.
- *Точность отрицательного класса* (negative precision) — как часто идеальные образцы классифицируются верно.
- *Чувствительность* (sensitivity), или *полнота* (recall), — также известна как *частота истинно положительных* (true-positive rate). Отношение

верно классифицированных нормальных бриллиантов к общему числу нормальных бриллиантов в обучающей выборке.

- *Специфичность* (specificity) — также известна как *частота истинно отрицательных* (true-negative rate). Отношение верно классифицированных отличных бриллиантов к общему числу отличных бриллиантов в обучающем датасете.
- *(Общая) точность* (accuracy) — как часто классификатор верно определяет классы в целом.

На рис. 8.30 показана итоговая матрица ошибок с результатами для примера с набором данных о бриллиантах. Здесь важна общая точность, но остальные метрики могут дать дополнительную полезную информацию об эффективности модели.

	Прогнозно положительные	Прогнозно отрицательные	
Фактически положительные	Истинно положительные 4	Ложноотрицательные 1	Чувствительность 4/4 + 1 = 0.8
Фактически отрицательные	Ложноположительные 2	Истинно отрицательные 3	Специфичность 3/3 + 2 = 0.6
	Точность 4/5 = 0.67	Точность отрицательного класса 3/4 = 0.75	Общая точность $\frac{7}{10}$ = 0.7

Рис. 8.30. Матрица ошибок для контрольного примера с бриллиантами

На основе этой информации в жизненном цикле алгоритма МО можно принимать более информированные решения, повышая эффективность модели. Как уже упоминалось, машинное обучение подразумевает экспериментирование методом проб и ошибок, а представленные метрики служат ориентирами.

Другие популярные алгоритмы машинного обучения

Эта глава познакомила вас с двумя основными широко известными алгоритмами МО. Алгоритм линейной регрессии используется для задач регрессии, в которых обнаруживаются связи между признаками. Алгоритм дерева решений применяется для задач классификации, в которых определяются связи между признаками и категориями образцов. Тем не менее есть и многие другие алгоритмы МО, которые подходят для других контекстов и решения других задач. На рис. 8.31 показаны некоторые популярные алгоритмы и их место в общей системе машинного обучения.

Рис. 8.31. Карта популярных алгоритмов машинного обучения

Алгоритмы классификации и регрессии подходят для задач, аналогичных рассмотренным в этой главе. Обучение без учителя содержит алгоритмы, используемые на некоторых этапах подготовки данных, при поиске скрытых внутренних связей и определении, какие вопросы можно задавать в эксперименте МО.

Обратите внимание на то, что на рис. 8.31 присутствует раздел глубокого обучения. В главе 9 будет рассмотрена тема искусственных нейронных сетей (ANN — artificial neural networks) — основной принцип глубокого обучения. Эта глава поможет лучше понять типы задач, которые можно решать с помощью ANN, и познакомит с реализацией соответствующих алгоритмов.

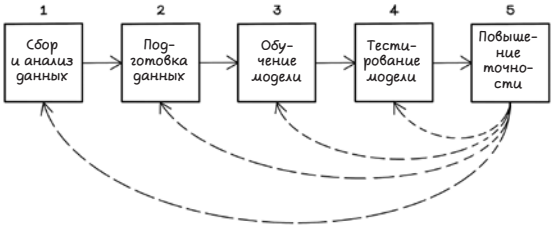
Применение алгоритмов машинного обучения

Машинное обучение можно применять практически во всех отраслях для решения разнообразных задач. При наличии подходящих данных и правильных вопросов возможности МО практически безграничны. Нам всем доводилось использовать продукты или сервисы, разработанные на основе машинного обучения и моделирования данных. Отметим основные сферы, в которых МО применяется для решения масштабных прикладных задач:

- *Обнаружение мошеннических действий и угроз.* МО используется в финансовой сфере для обнаружения и предотвращения мошеннических транзакций. Финансовые структуры собрали огромное количество информации по транзакциям, включая сообщения от клиентов о мошеннических действиях. Эти сообщения выступают в качестве входных данных для разметки и характеризации мошеннических переводов. В итоге создаваемые модели способны оценивать законность перевода денежных средств с учетом данных о месте совершения транзакции, сумме, получателе и т. д. Это позволяет защитить клиентов от кражи их средств, а финансовые организации — от страховых потерь. Та же модель может использоваться для обнаружения сетевых угроз и предотвращения атак, ориентируясь на известные данные об использовании сети и сообщения о нестандартном поведении в ней.
- *Рекомендации товаров и контента.* Многие люди приобретают товары через интернет или используют потоковые сервисы для прослушивания музыки и просмотра видео. Они получают рекомендации товаров на основе истории покупок, а рекомендации контента — на основе интересов. Эта функциональность обычно реализуется через системы МО, в которых модель покупательского поведения или предпочтения контента определяются на основе действий пользователей. Рекомендательные системы используются все чаще и чаще, чтобы увеличить объем продаж или повысить качество взаимодействия с клиентами.
- *Динамическое формирование цен на товары и услуги.* Цены на товары и услуги чаще всего формируются на основе покупательского интереса или на основе риска. К примеру, в сервисе каршеринга разумно поднимать цену, если количество доступных машин меньше, чем спрос на них. Такое повышение цены еще называют *пиковым тарифом*. В страховании цена полиса может повышаться, если степень связанного с человеком риска оценивается как высокая. В этом случае на основе динамических условий и информации об отдельном человеке с помощью МО устанавливаются признаки и связи между признаками, которые в итоге влияют на ценообразование.
- *Прогнозирование рисков для здоровья.* Система здравоохранения заинтересована, чтобы специалисты приобретали обширные знания и применяли их для успешного диагностирования и лечения пациентов. За всю историю медицинской практики было собрано очень много данных о больных: группы крови, ДНК, истории болезней в семье, места проживания, образ жизни и др. Эти данные можно использовать для установления потенциальных закономерностей, которые помогут диагностировать заболевания. Это позволит начинать лечение болезни на ранней стадии. Предоставляя той же системе МО полученные результаты, можно добиться еще более точного прогнозирования.

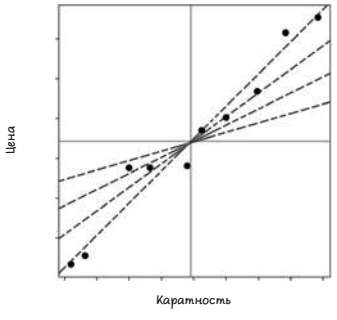
Краткий обзор главы «Машинное обучение»

Машинное обучение — это в первую очередь контекст, анализ данных и постановка верных вопросов, а уже во вторую очередь — алгоритмы.

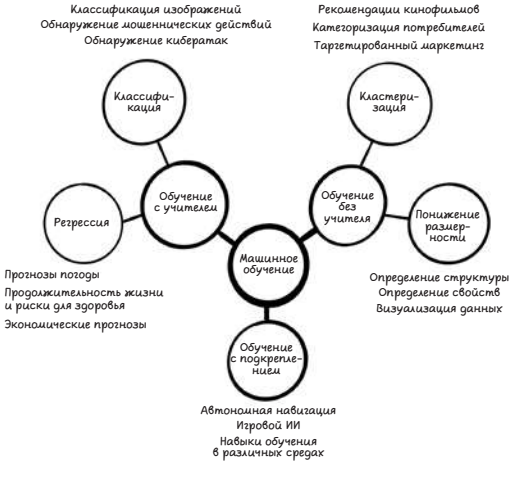


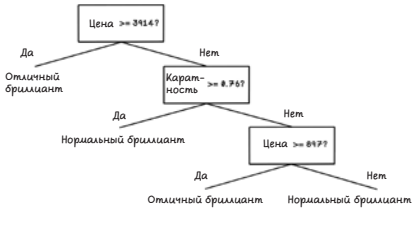
Жизненный цикл МО состоит из итераций и экспериментов

Линейная регрессия служит для нахождения лучшей линии для данных, что подразумевает минимизирование возможности ошибки в каждой точке данных.



Деревья решений делят данные на основе вопросов до тех пор, пока датасет не будет должным образом категоризирован. В первую очередь для этого необходимо снизить неопределенность датасета.





Различные алгоритмы МО используются для ответов на соответствующие вопросы и достижения целей в разных контекстах.

9 Искусственные нейронные сети

В этой главе

- ✓ Основные принципы работы искусственных нейронных сетей
- ✓ Задачи, которые можно решать с помощью искусственных нейронных сетей
- ✓ Объяснение и реализация метода прямого распространения ошибки с использованием обученной сети
- ✓ Объяснение и реализация обратного распространения ошибки для обучения сети
- ✓ Построение архитектур искусственных нейронных сетей для решения различных задач

Что такое искусственные нейронные сети?

Искусственные нейронные сети (artificial neural networks, ANN) — это мощный инструмент машинного обучения, используемый для реализации таких задач, как распознавание изображений, обработка естественного языка и игры. Обучение ANN, как и других алгоритмов МО, строится на тренировочных (обучающих) данных. Эти сети лучше всего подходят для неструктурированных данных, в которых сложно понять, как признаки связаны друг с другом. В этой главе разъясняются основы ANN, а также демонстрируется принцип работы этого алгоритма и рассматривается разработка нейронной сети для решения различных задач.

Чтобы лучше понять, какое место занимают ANN в общей карте машинного обучения, следует обратиться к взаимосвязям и категориям алгоритмов МО. *Глубокое обучение* — это разновидность алгоритмов, использующих для достижения цели ANN различных архитектур. Глубокое обучение, включая ANN, можно применять для реализации задач обучения с учителем, без учителя, а также обучения с подкреплением. На рис. 9.1 показано, как оно связано с ANN и другими концепциями МО.

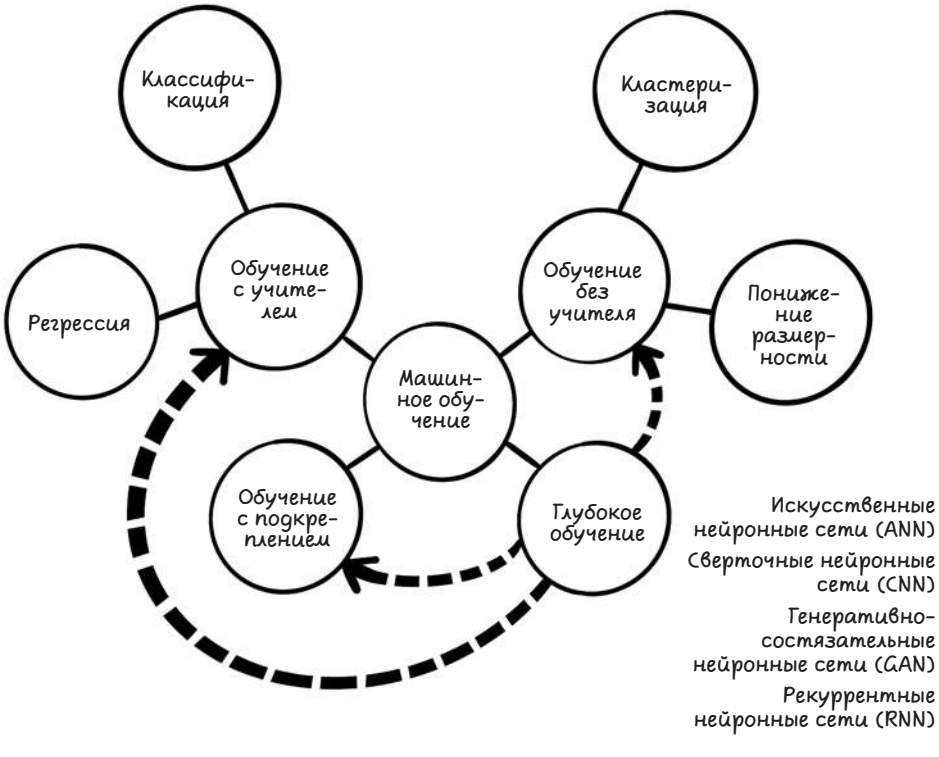


Рис. 9.1. Карта, описывающая гибкость глубокого обучения и ANN

ANN можно представить просто как другую модель в жизненном цикле машинного обучения (глава 8), который показан на рис. 9.2. Сначала определяется задача. Затем происходит сбор, анализ и подготовка соответствующих данных. В завершение обучения модель ANN тестируется и по необходимости дорабатывается.

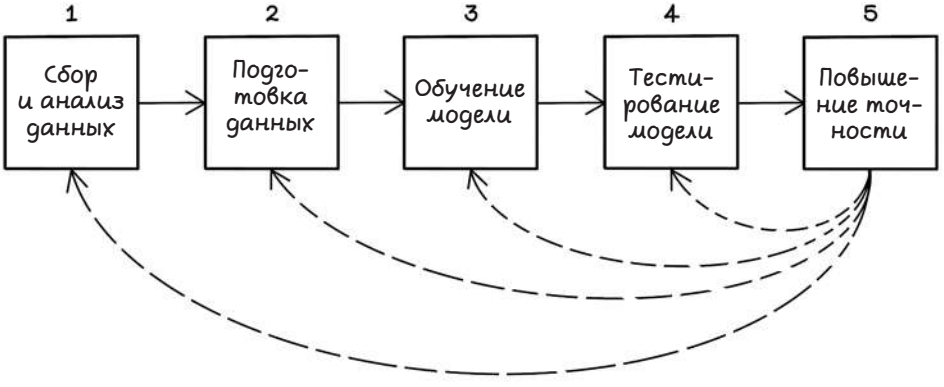


Рис. 9.2. Схема реализации экспериментов и проектов МО

Итак, мы сформировали представление о месте и роли ANN в общем ландшафте МО, а также уточнили, что ANN является еще одной моделью, обучаемой по тому же жизненному циклу. Теперь перейдем к рассмотрению ее сути и принципов работы. Подобно генетическим и роевым алгоритмам, ANN были разработаны на основе природных процессов — в данном случае работы мозга и нервной системы. Нервная система — это биологическая структура, которая позволяет испытывать ощущения и лежит в основе функционирования мозга. Человеческое тело содержит бесчисленное множество нервных окончаний и нейронов, которые действуют по единому принципу.

Нейронные сети состоят из связанных нейронов, которые передают информацию посредством электрических или химических сигналов. В процессе передачи информации между нейронами она корректируется, чтобы выполнять определенную функцию. Когда вы берете чашку и делаете глоток воды, за обработку всех связанных с этим действий и предоставление обратной связи о результате отвечают миллионы нейронов. Представьте себе детей, которые только учатся пить из чашки. Они начинают делать это неуверенно и неуклюже, часто роняя чашку. Сначала ребенок обучается брать ее двумя руками. Далее он постепенно осваивает механизм удержания чашки одной рукой и уверенно пьет из нее. Тем не менее на достижение этого результата могут уйти месяцы. В течение этого времени мозг и нервная система обучаются на практике. Рисунок 9.3 показывает упрощенную модель получения входных сигналов (стимулов), их обработки в нейронной сети и выдачу результатов (ответа).

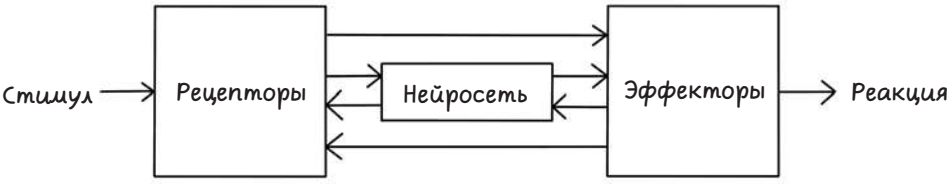


Рис. 9.3. Упрощенная модель биологической нейронной системы

Нейрон (рис. 9.4) состоит из дендритов (отростков), которые получают сигналы от других нейронов; клеточного тела и ядра, которое активирует и корректирует сигнал; аксона, передающего сигнал другим нейронам; и синапсов, которые переносят и одновременно подстраивают сигнал перед его передачей дендритам последующих нейронов. Наш мозг состоит примерно из 90 миллиардов взаимодействующих нейронов, благодаря чему мы обладаем таким высокоразвитым интеллектом.

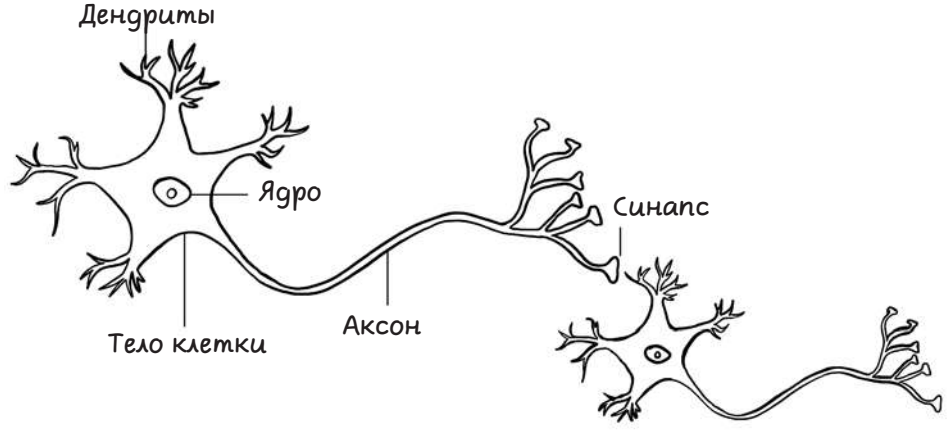


Рис. 9.4. Общая структура связей нейронов

Несмотря на то что ANN построены и функционируют по принципу биологических нейронных сетей, их нельзя назвать полными аналогами. Ученым еще многое предстоит узнать о работе мозга и нервной системы.

Восприятие: представление нейрона

Нейрон — основная составляющая нервной системы и головного мозга. Как уже говорилось, он принимает множество входящих сигналов от других нейронов, обрабатывает их и передает результат другим нейронам. ANN основаны на фундаментальном принципе *перцептрона* — логического представления одного биологического нейрона.

По аналогии с нейронами перцептрон получает входящие сигналы (роль дендритов), изменяет эти сигналы на основе весов (роль синапсов), обрабатывает взвешенные входы (роль клеточного тела и ядра) и выводит результат (роль аксонов). В общем виде перцептрон основан на принципе нейрона. Можно заметить, что синапсы изображены на схеме после дендритов, что отражает влияние синапсов на входные сигналы. Рисунок 9.5 демонстрирует логическую архитектуру перцептрона.

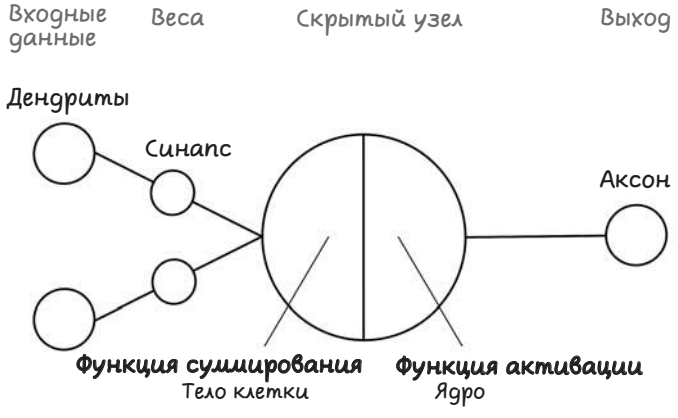


Рис. 9.5. Логическая архитектура перцептрона

Компоненты перцептрона описываются переменными, которые участвуют в вычислении результата. На первом этапе веса корректируют входные данные. Полученное значение обрабатывается скрытым узлом, и на выходе предоставляется результат.

Вот краткое описание компонентов перцептрона:

- *Входные данные* описывают входные значения. В биологическом нейроне эти значения являются входным сигналом.
- *Веса* описывают веса каждого соединения между входом и скрытым узлом. Они влияют на интенсивность входных данных и формируют взвешенный вход. В нейроне эти связи являются синапсами.
- *Скрытый узел (суммирование и активация)* суммирует взвешенный вход и применяет к результату функцию активации. Эта функция определяет активацию/выход скрытого узла/нейрона.
- *Выход* представляет результат работы перцептрона.

Чтобы понять внутренние процессы перцептрона, рассмотрим его на уже знакомом примере с поиском жилья из главы 8. Представим себя агентами по недвижимости, которым на основании размера и стоимости квартиры нужно определить, будет ли она арендована в течение месяца. Предположим, что перцептрон уже обучен, то есть его веса настроены нужным образом. Способ

обучения перцептронов и ANN рассмотрим несколько позже. Сейчас важно просто знать, что веса кодируют связи между входными данными, регулируя силы этих данных.

На рис. 9.6 представлено использование предварительно обученного перцептрона для ответа на вопрос, будет ли квартира арендована. Входные данные представлены ценой конкретного объекта жилья и его площадью. Вдобавок мы масштабируем входные данные, используя максимальные значения стоимости и площади ($8000 для стоимости и 80 кв. м для площади). Подробнее о масштабировании — в следующем разделе.

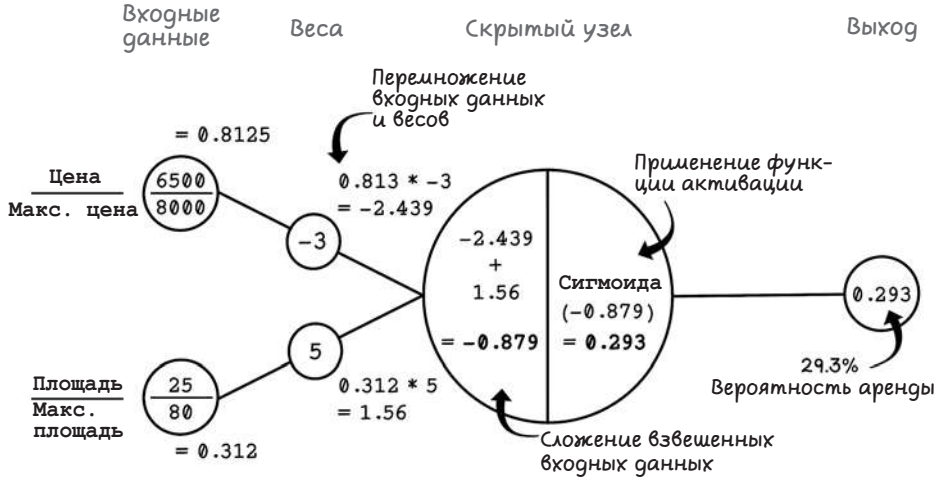


Рис. 9.6. Пример использования предварительно обученного перцептрона

Обратите внимание, что цены и площадь выступают в качестве входных данных, а шанс аренды — в качестве выхода. Ключевую же роль в прогнозировании играют веса. Они являются переменными сети, которые изучают связи между входными данными. Функции суммирования и активации используются для формирования прогноза путем обработки умноженных на веса входных значений.

Заметьте также, что в качестве функции активации используется сигмоида. Функции активации играют очень важную роль в перцептроне и ANN. В данном случае эта функция помогает решить линейную задачу. Основы такого типа задач показаны на рис. 9.7. В следующем разделе мы узнаем, как функции активации помогают получать входные данные в нелинейных сценариях.

Ниже видно, что сигмоида на основе входных значений в диапазоне от 0 до 1 превращается в кривую S от 0 до 1. Так как эта функция позволяет изменениям в x оказывать небольшое влияние на y, она подходит для постепенного обучения. Когда чуть позже мы рассмотрим устройство ANN более подробно, мы увидим, как эта функция помогает решать в том числе нелинейные задачи.

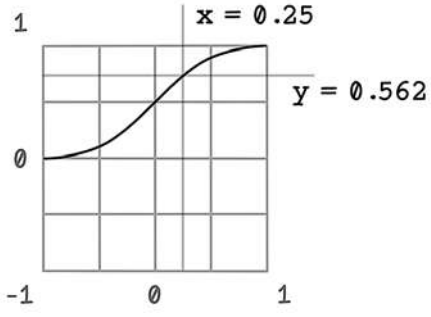


Рис. 9.7. Сигмоида

Вернемся назад, к данным, которые мы используем в перцептроне. Понимание данных, связанных с тем, была ли сдана квартира в аренду, очень важно для понимания работы перцептрона. На рис. 9.8 показаны образцы из датасета, включая цену и площадь каждой квартиры. Все квартиры маркированы по двум классам: арендованные или неарендованные. Линия, разделяющая эти два класса, представляет собой функцию, описываемую перцептроном.

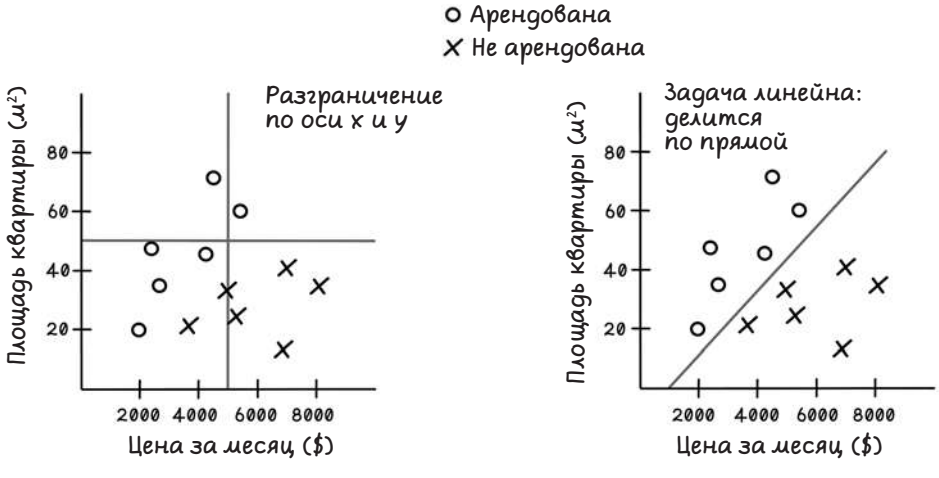


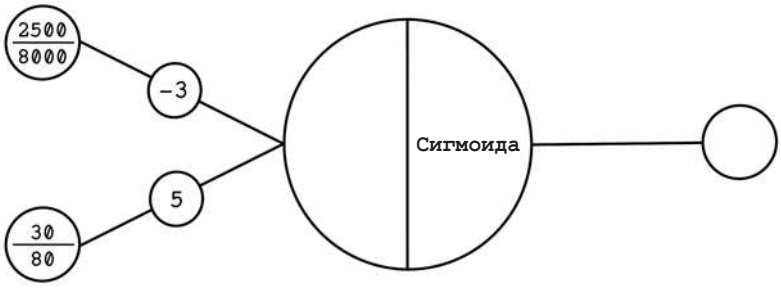
Рис. 9.8. Пример линейной задачи классификации

Перцептрон эффективен в решении линейных задач, однако не способен решать нелинейные. Если датасет нельзя разделить по прямой, перцептрон с задачей не справится.

В ANN используется принцип перцептрона в масштабе. Множество аналогичных перцептрону нейронов работают совместно над решением нелинейных задач во множестве измерений. Обратите внимание, что используемая функция активации влияет на способность ANN к обучению.

УПРАЖНЕНИЕ: ВЫЧИСЛИТЕ РЕЗУЛЬТАТ РАБОТЫ ПЕРЦЕПТРОНА НА ОСНОВЕ ПРЕДЛОЖЕННЫХ ВХОДНЫХ ДАННЫХ

На основе полученных знаний о работе перцептрона рассчитайте выход для следующего примера:



ОТВЕТ: РЕЗУЛЬТАТ РАБОТЫ ПЕРЦЕПТРОНА

Определение искусственных нейронных сетей

Перцептрон эффективен для решения простых задач, но по мере увеличения количества измерений данных он становится бесполезен. ANN работают по принципу перцептрона, но применяют уже не один, а множество скрытых нейронов.

Чтобы понять суть работы ANN, состоящих из множества нейронов, рассмотрим пример набора данных о ДТП. Предположим, что имеются данные из нескольких автомобилей в момент неожиданного появления объекта на пути их движения. Датасет содержит признаки, относящиеся к условиям и факту столкновения, включая:

- *Скорость* — скорость, с которой двигалась машина на момент появления объекта.
- *Качество дороги* — качество дорожного покрытия, по которому двигалась машина.
- *Угол обзора* — водительский угол обзора до появления объекта.
- *Водительский стаж* — общий стаж вождения водителя.
- *Произошло ли столкновение?* — Был ли факт столкновения.

На основе этих данных нужно обучить модель МО, а именно ANN, находить и изучать связи между перечисленными признаками, как показано в табл. 9.1.

Таблица 9.1. Набор данных о ДТП

	Скорость	Качество покрытия	Угол обзора	Водительский стаж	Произошло ли столкновение?
1	65 км/ч	5/10	180°	80 000 км	Нет
2	120 км/ч	1/10	72°	110 000 км	Да
3	8 км/ч	6/10	288°	50 000 км	Нет
4	50 км/ч	2/10	324°	1600 км	Да
5	25 км/ч	9/10	36°	160 000 км	Нет
6	80 км/ч	3/10	120°	6000 км	Да
7	40 км/ч	3/10	360°	400 000 км	Нет

С помощью примера архитектуры ANN на основе имеющихся признаков можно произвести классификацию, спрогнозировав вероятность столкновения. Признаки датасета будут служить входными данными для ANN, а прогнозируемый класс выступит в качестве ее выхода. В этом примере входными нейронами будут скорость, качество дорожного покрытия, угол обзора и водительский стаж. Выходной же нейрон будет показывать, произошло ли столкновение (рис. 9.9).

Как и в других алгоритмах МО, с которыми мы познакомились, для эффективной классификации важно качественно подготовить данные. Основная задача — представить их в сопоставимом виде. Будучи людьми, мы имеем представление о понятии скорости и угла обзора, но ANN контекст этих признаков не известен. Прямое сравнение 65 км/ч и 36 градусов угла обзора будет для сети бессмысленным. А вот сравнение пропорций скорости и угла обзора уже даст приемлемый результат. Здесь-то и потребуется масштабировать данные.

Стандартным способом масштабирования данных в целях их сопоставления является *минимакс*. Он подразумевает приведение данных к значениям между 0 и 1. Приведя все данные датасета в согласованный формат, мы получим возможность сопоставлять разные признаки. И поскольку ANN не знает контекста для сырых признаков, мы также удаляем смещение, вызываемое большими значениями. Например, 1000 намного больше 65, но в контексте водительского

стажа 1000 — это низкий показатель, а 65 в контексте скорости движения — уже довольно большой. Минимаксное масштабирование представляет эти данные в правильном контексте, так как берет в расчет минимальные и максимальные возможные значения каждого признака.

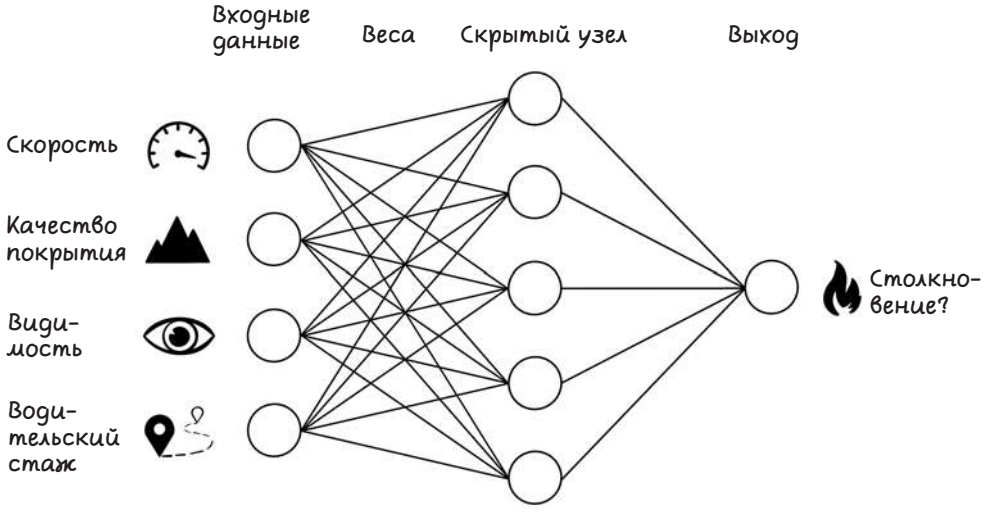


Рис. 9.9. Архитектура ANN для примера с ДТП

Для признаков набора данных о ДТП выбраны следующие минимальные и максимальные величины:

- *Скорость*. Минимальная скорость равна 0, то есть автомобиль не движется. Для максимальной скорости будет использовано значение 120, так как 120 км/ч является максимально разрешенной скоростью движения в большинстве стран. Предположим, что водитель правила не нарушает.
- *Качество дорожного покрытия*. Так как эти данные уже представлены в виде рейтинга, минимумом будет 0, а максимумом 10.
- *Угол обзора*. Известно, что максимальный угол обзора составляет 360 градусов, значит, минимальное значение будет 0, а максимальное 360.
- *Водительский стаж*. В случае отсутствия опыта вождения минимальное значение составляет 0. Максимальное же мы субъективно установим как 400 000. Логика в том, что если водитель проехал 400 000 км, то его уже можно считать опытным и любой дополнительный километраж не так важен.

При минимаксном масштабировании берутся минимальное и максимальное значения признака, и на их основе вычисляется процент фактического значения признака. Формула проста: вычесть из значения его минимум и разделить результат на разность максимального и минимального значений. На рис. 9.10 показано это вычисление для первой строки данных:

	Скорость	Качество покрытия	Угол обзора	Водительский стаж	Произошло ли столкновение?
1	65 км/ч	5/10	180°	80 000 км	Нет

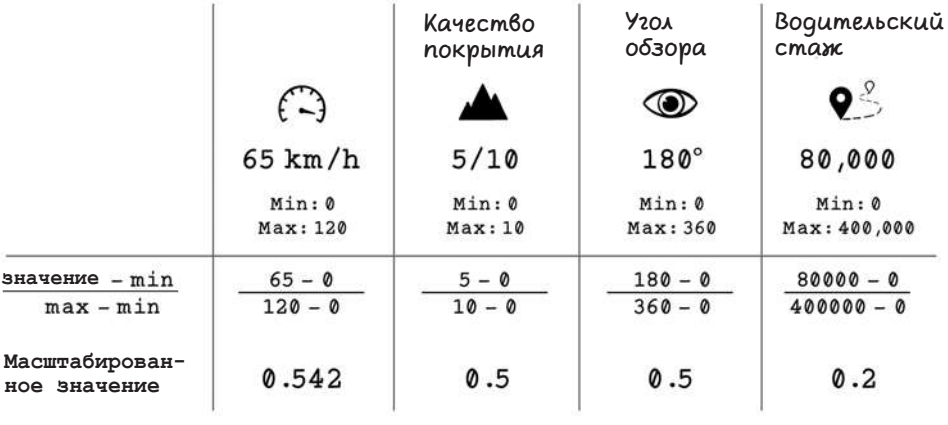


Рис. 9.10. Пример минимаксного масштабирования минимакс для данных о ДТП

Обратите внимание, что все значения находятся в диапазоне от 0 до 1 и могут сопоставляться на равных условиях. Эта же формула применяется ко всем строкам датасета, обеспечивая масштабирование каждого значения. Заметьте, что для значения признака «Произошло ли столкновение» вариант «Да» заменяется на 1, а «Нет» на 0. Все масштабированные данные представлены в табл. 9.2.

Таблица 9.2. Набор данных о ДТП после масштабирования

	Скорость	Качество покрытия	Угол обзора	Водительский стаж	Произошло ли столкновение?
1	0.542	0.5	0.5	0.200	0
2	1.000	0.1	0.2	0.275	1
3	0.067	0.6	0.8	0.125	0
4	0.417	0.2	0.9	0.004	1
5	0.208	0.9	0.1	0.400	0
6	0.667	0.3	0.3	0.015	1
7	0.333	0.3	1.0	1.000	0

Псевдокод

Код для масштабирования данных в точности использует логику и вычисления минимакс. Требуется минимум и максимум для каждого признака, а также общее количество признаков в датасете. В функции `scale_dataset` эти параметры используются для перебора каждого образца из датасета и масштабирования его значения функцией `scale_data_feature`:

```
FEATURE_MIN = [0, 0, 0, 0]
FEATURE_MAX = [120, 10, 360, 400000]
FEATURE_COUNT = 4

scale_dataset(dataset, feature_count, feature_min, feature_max):
  let scaled_data equal empty array
  for data in dataset:
    let example equal empty array
    for i in range(0, feature_count):
      append scale_data_feature(data[i], feature_min[i], feature_max[i])
        to example
    append example to scaled_data
  return scaled_data

scale_data_feature(data, feature_min, feature_max):
  return (data - feature_min) / (feature_max - feature_min)
```

После приведения данных в подходящий для ANN вид можно рассмотреть архитектуру простой нейронной сети. Помним, что признаки, используемые для прогнозирования класса, будут являться входными нейронами, а сам прогнозируемый класс — выходным.

На рис. 9.11 показана ANN с одним скрытым слоем, который представлен пятью вертикальными скрытыми нейронами. Эти слои называются *скрытыми*, потому что снаружи сети они не просматриваются. Взаимодействие происходит только с входными и выходным нейронами, в связи с чем ANN воспринимают как черные ящики. Каждый скрытый нейрон аналогичен перцептрону. Он получает входные значения и веса, после чего вычисляет функции суммирования и активации. В завершение результаты каждого скрытого нейрона обрабатываются единственным выходным нейроном.

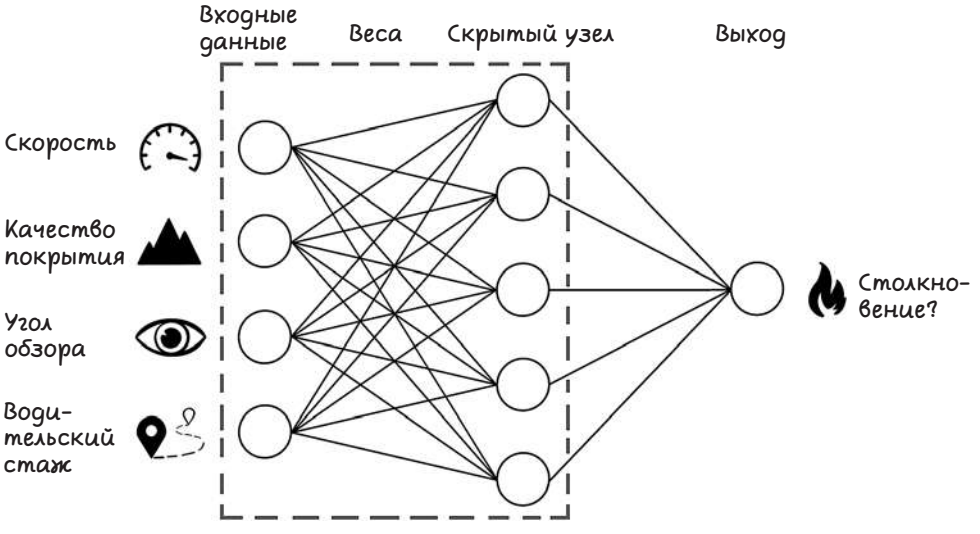


Рис. 9.11. Пример архитектуры ANN для задачи о ДТП

Прежде чем разбирать совершаемые в ANN вычисления, попробуем интуитивно понять, что делают веса в нейронной сети на высоком уровне. Так как каждый скрытый нейрон соединен с каждым входным нейроном, но при этом каждое соединение имеет свой вес, отдельные скрытые нейроны могут сосредотачиваться на определенных связях между двумя и более входными нейронами.

На рис. 9.12 изображен сценарий, в котором в первом скрытом нейроне в связях с качеством дорожного покрытия и углом обзора веса большие, а в связях

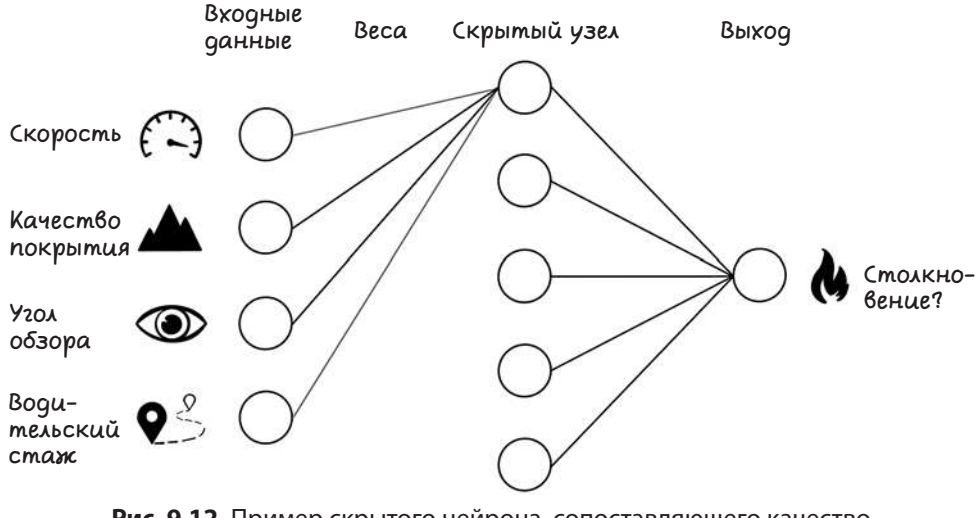


Рис. 9.12. Пример скрытого нейрона, сопоставляющего качество покрытия и угол обзора

со скоростью и опытом вождения — маленькие. Этот конкретный скрытый нейрон обрабатывает связи между качеством дорожного покрытия и углом обзора. Он может устанавливать связь двух указанных признаков и то, как эта связь влияет на возможность столкновения. К примеру, низкое качество покрытия и плохой угол обзора повышают вероятность столкновения, в противоположность хорошему качеству покрытия и среднему углу обзора. Обычно на практике эти связи более сложные, чем в описанном примере.

На рис. 9.13 второй скрытый нейрон может иметь сильные веса по связям с качеством покрытия и водительским стажем. Вероятно, между разным качеством покрытия и различием в опыте вождения есть закономерность, которая влияет на риск столкновения.

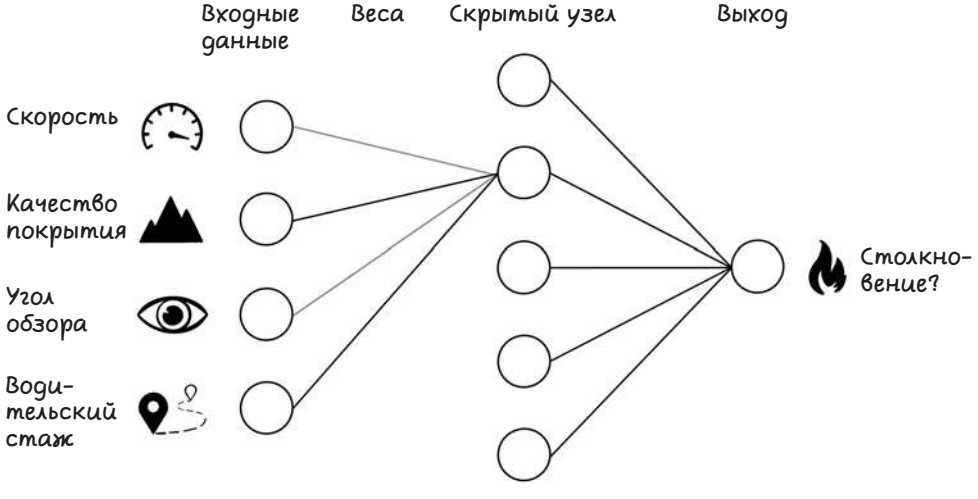


Рис. 9.13. Пример скрытого нейрона, сравнивающего качество покрытия и водительский стаж

Нейроны скрытого слоя можно сравнить с муравьями из примера в главе 6. Отдельные муравьи выполняют небольшие задачи, которые могут казаться незначительными. Но когда муравьи действуют как колония, они демонстрируют интеллектуальное поведение. Аналогичным образом отдельные скрытые нейроны вносят вклад в общую цель ANN.

Путем анализа графического представления ANN и происходящих в ней операций можно описать структуры данных, необходимые для этого алгоритма:

- *Входные нейроны*. Входные нейроны можно представить одним массивом, хранящим значения для конкретного образца. Размер массива будет равен количеству признаков датасета, используемых для прогнозирования класса. В примере с ДТП присутствует четыре входных признака, значит, размер массива будет равен 4.

- *Веса.* Веса можно представить как матрицу (двумерный массив), так как каждый входной нейрон связан с каждым скрытым нейроном, то есть имеет всего 5 соединений. Поскольку в примере задействовано 4 входных нейрона с 5 связями каждый, то в ANN получается 20 весов, ведущих к скрытому слою, и 5, ведущих к выходному, так как в скрытом слое 5 нейронов, а на выходе 1 нейрон.
- *Скрытые нейроны.* Скрытые нейроны можно представить как массив, хранящий результат активации каждого соответствующего нейрона.
- *Выходной нейрон.* Выходной нейрон — это одно значение, выражающее спрогнозированный класс для конкретного образца или шанс, что образец попадет в конкретный класс. Выход может иметь значение 0 или 1, указывая, произошло ли ДТП. Он также может иметь, например, значение 0,65, указывающее на 65%-ную вероятность ДТП.

Псевдокод

Приведенный псевдокод описывает класс, представляющий нейронную сеть. Обратите внимание, что слои выражены как свойства этого класса и все свойства являются массивами, за исключением весов, хранящихся в матрицах. Свойство `output` представляет прогнозы для заданных образцов, а свойство `expected_output` используется в процессе обучения:

```
NeuralNetwork(features, labels, hidden_node_count):
  let input equal features
  let weights_input equal a random matrix, size: features * hidden_node_count
  let hidden equal zero array, size: hidden_node_count
  let weights_hidden equal a random matrix, size: hidden_node_count
  let expected_output equal labels
  let output equal zero array,size: length of labels


let nn equal NeuralNetwork(scaled_feature_data,
                           scaled_label_data,
                           hidden_node_count)
```

Прямое распространение: использование обученной ANN

Обученная ANN — это сеть, которая на основе изученных образцов подстроила веса для наилучшего прогнозирования на новых примерах данных. Не пугайтесь возможной сложности процесса обучения и подстройки весов. Эту тему мы подробно разберем в следующем разделе. Понимание принципа прямого

распространения (называют также «прямое распространение ошибки») поможет понять технику обратного распространения ошибки, с помощью которой обучаются веса. Теперь, когда вы знаете основы общей архитектуры ANN и понимаете, как действуют нейроны в сети, пора перейти к разработке алгоритма для использования обученной ANN (рис. 9.14).

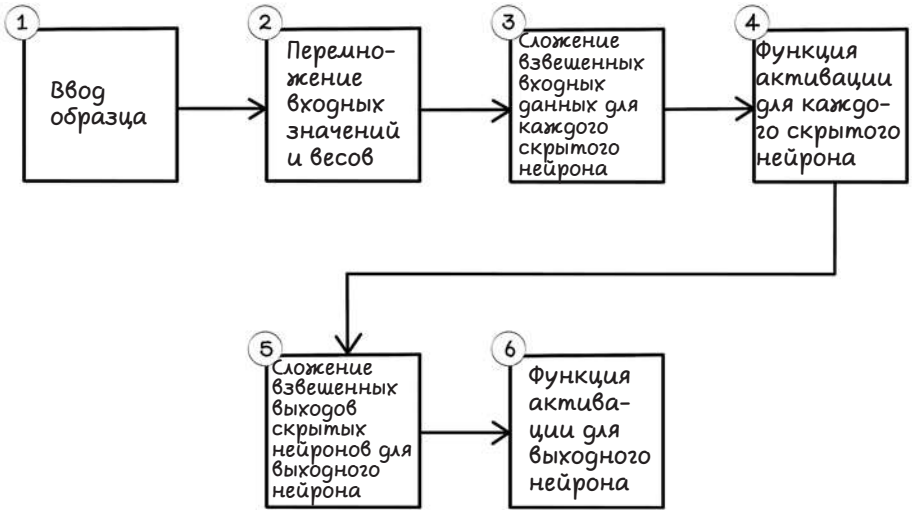


Рис. 9.14. Жизненный цикл прямого распространения в ANN

Как уже говорилось, этапы вычисления результатов нейронами в ANN аналогичны происходящим в перцептроне. Схожие операции выполняются в большом числе совместно работающих нейронов, что компенсирует недостатки перцептрона и используется для решения многомерных задач. Прямое распространение состоит из следующих процессов:

1. *Ввод образца* — предоставление одного образца из датасета, для которого нужно спрогнозировать класс.
2. *Перемножение входных значений и весов* — умножение каждого входного значения на вес каждой его связи со скрытыми нейронами.
3. *Сложение взвешенных входных данных для каждого скрытого нейрона* — суммирование результатов взвешенных входных значений в каждом скрытом нейроне.
4. *Функция активации для каждого скрытого нейрона* — применение функции активации к суммам взвешенных входных значений.
5. *Сложение взвешенных выходов скрытых нейронов для выходного нейрона* — суммирование результатов функции активации для всех скрытых нейронов.
6. *Функция активации для выходного нейрона* — применение функции активации к сумме взвешенных скрытых нейронов.

Напомню, мы предполагаем, что ANN обучена и оптимальные веса найдены. На рис. 9.15 показаны их значения для каждой связи. К примеру, в первом блоке рядом с первым скрытым нейроном указан вес 3.35, который относится ко входному нейрону для значений скорости. Вес –5.82 относится ко входному нейрону для значений качества покрытия, и т. д.

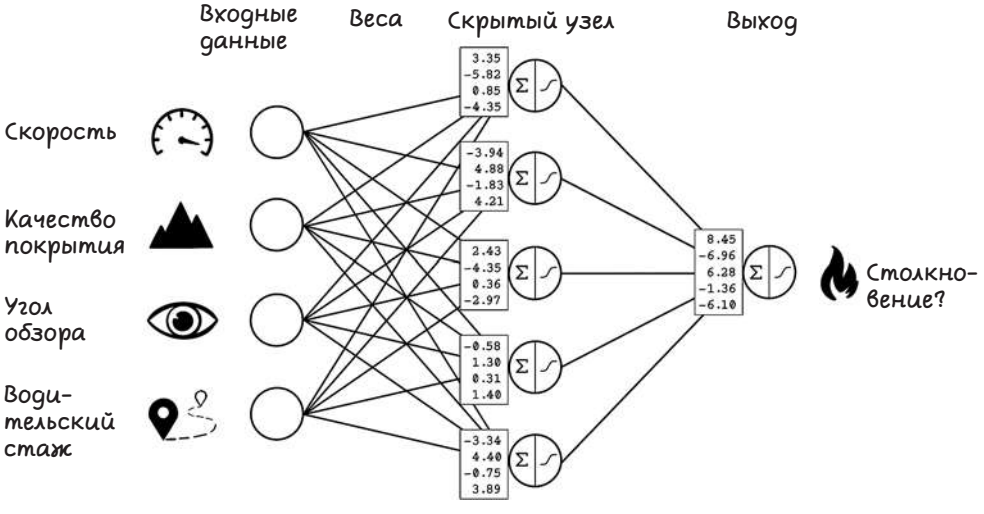


Рис. 9.15. Пример весов предварительно обученной ANN

Так как эта нейронная сеть обучена, можно сразу использовать ее для прогнозирования шанса столкновения, предоставив один образец. В табл. 9.3 для напоминания приведен используемый масштабированный датасет.

Таблица 9.3. Масштабированные данные о ДТП

	Скорость	Качество покрытия	Угол обзора	Водительский стаж	Произошло ли столкновение?
1	0.542	0.5	0.5	0.200	0
2	1.000	0.1	0.2	0.275	1
3	0.067	0.6	0.8	0.125	0
4	0.417	0.2	0.9	0.004	1
5	0.208	0.9	0.1	0.400	0
6	0.667	0.3	0.3	0.015	1
7	0.333	0.3	1.0	1.000	0

Если вы уже имели дело с ANN, то, скорее всего, встречали пугающе сложные формулы. Давайте разберем некоторые принципы, которые можно представить математически.

Входы ANN обозначаются как X. Каждая входная переменная будет обозначена X с числом в нижнем индексе: скорость — X_0, качество покрытия — X_1 и т. д. Выход сети обозначается как Y, а веса как W. Так как в сети используются два слоя — скрытый и выходной, — то присутствуют две группы весов. Первая обозначается как W_0, а вторая как W_1. В каждом весе также содержится обозначение нейронов, которые он объединяет. К примеру, вес между нейроном для значений скорости и первым скрытым нейроном — это $W_{00,0}$, а между нейроном качества для значений покрытия и первым скрытым нейроном — $W_{01,0}$. Эти обозначения необязательны для текущего примера, но их понимание пригодится в дальнейшем.

Рисунок 9.16 показывает, как следующие данные представлены в ANN.

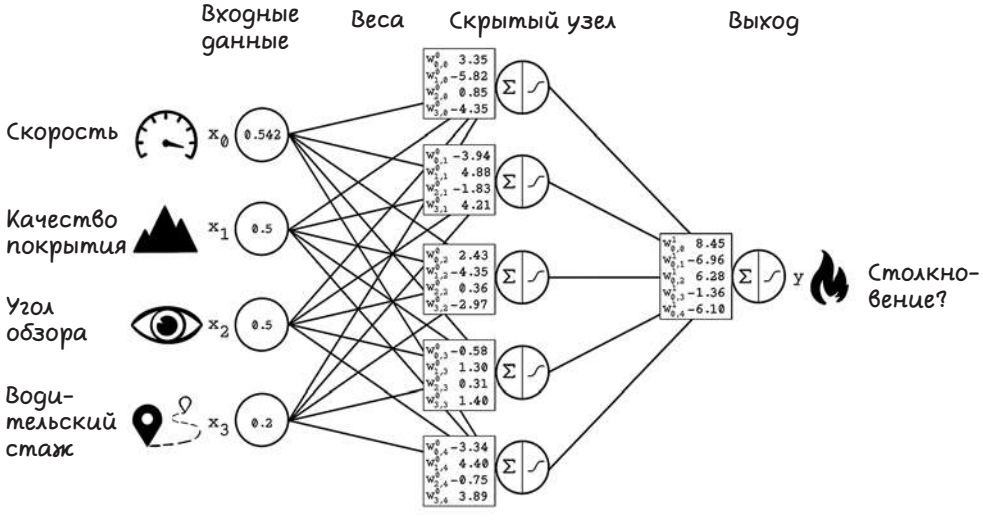


Рис. 9.16. Математическое представление ANN

Как и в случае с перцептроном, первый этап — вычисление взвешенной суммы входов и весов каждого скрытого нейрона. На рис. 9.17 каждый вход умножается на каждый вес и результаты для каждого скрытого нейрона складываются.

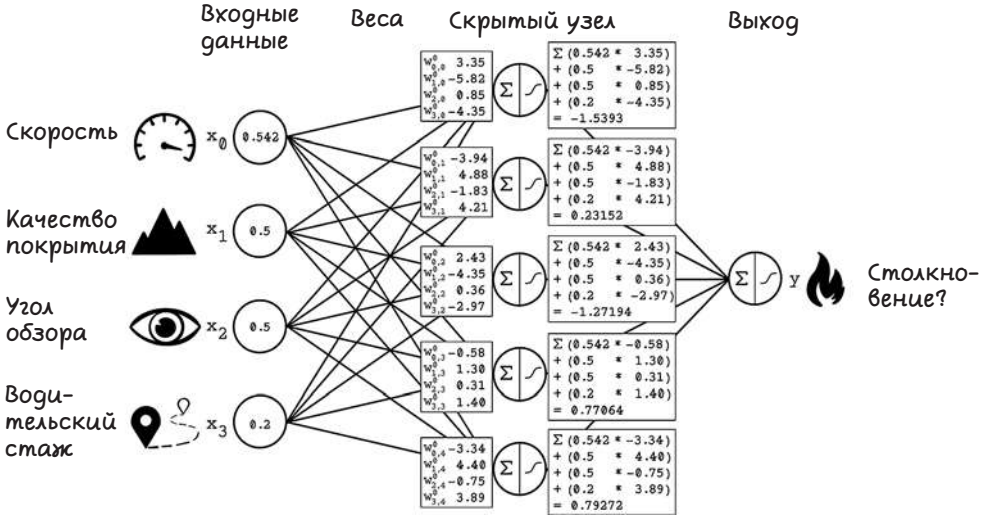


Рис. 9.17. Вычисление взвешенной суммы для каждого скрытого нейрона

Второй этап — вычисление активации каждого скрытого нейрона. Для этого используется сигмоидная функция, в качестве входа которой принимается взвешенная сумма входных значений, рассчитанная для каждого нейрона (рис. 9.18).

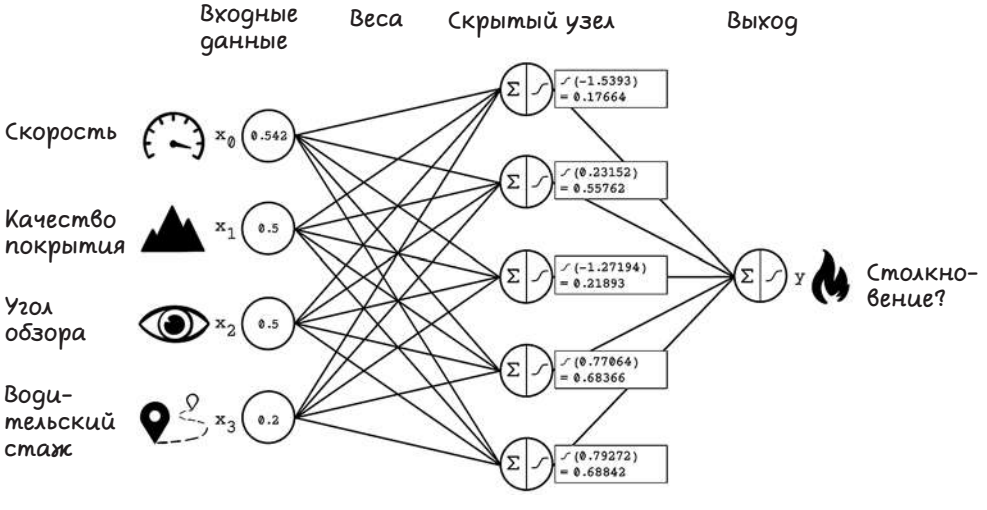


Рис. 9.18. Вычисление функции активации для каждого скрытого нейрона

Теперь у нас есть результаты активации для каждого скрытого нейрона. Если перенести этот результат на биологическую нейросеть, то он будет представлять интенсивность активации каждого естественного нейрона. Поскольку разные скрытые нейроны могут устанавливать разные связи в данных через веса, то их активации можно совмещать для определения общей активации, представляющей шанс возникновения ДТП, на основе входных значений.

На рис. 9.19 приводятся активации для каждого скрытого нейрона и веса их связей с выходным нейроном. Для получения конечного выхода процесс вычисления взвешенной суммы результатов каждого скрытого слоя повторяется, и к полученному значению применяется сигмоида.

> ***Примечание***
>
> Греческая буква сигма (Σ) в скрытых слоях означает операцию суммирования.

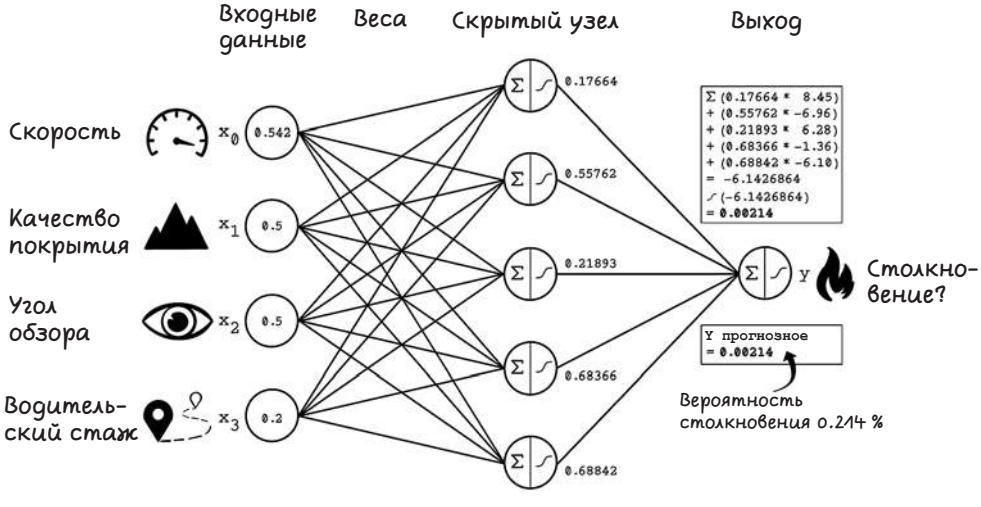


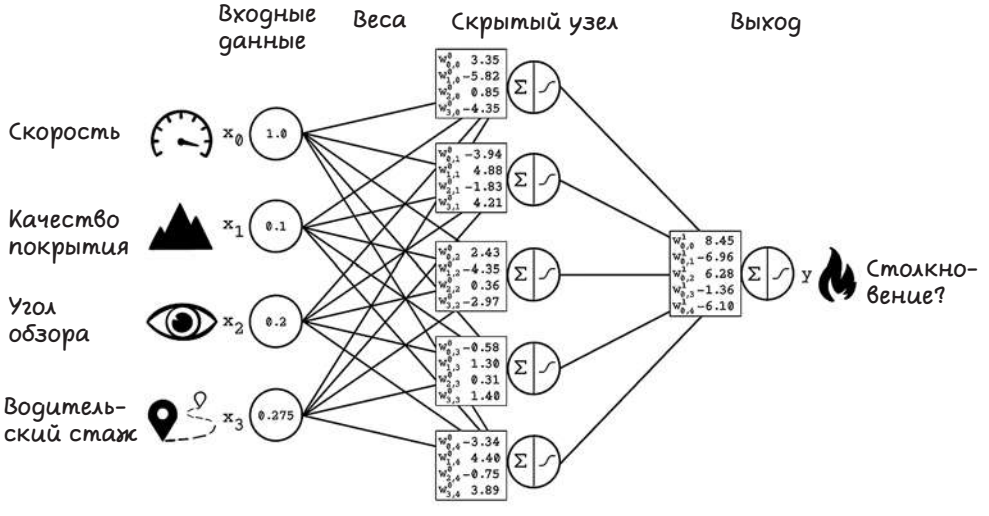
Рис. 9.19. Последнее вычисление активации для выходного нейрона

Мы рассчитали итоговый прогноз для заданного образца. Результат получился равен 0.00214, но что это значит? Выход — это значение между 0 и 1, которое представляет вероятность ДТП. В этом случае значение составляет 0.214 % (0.00214 × 100), что означает, что шанс столкновения практически равен 0.

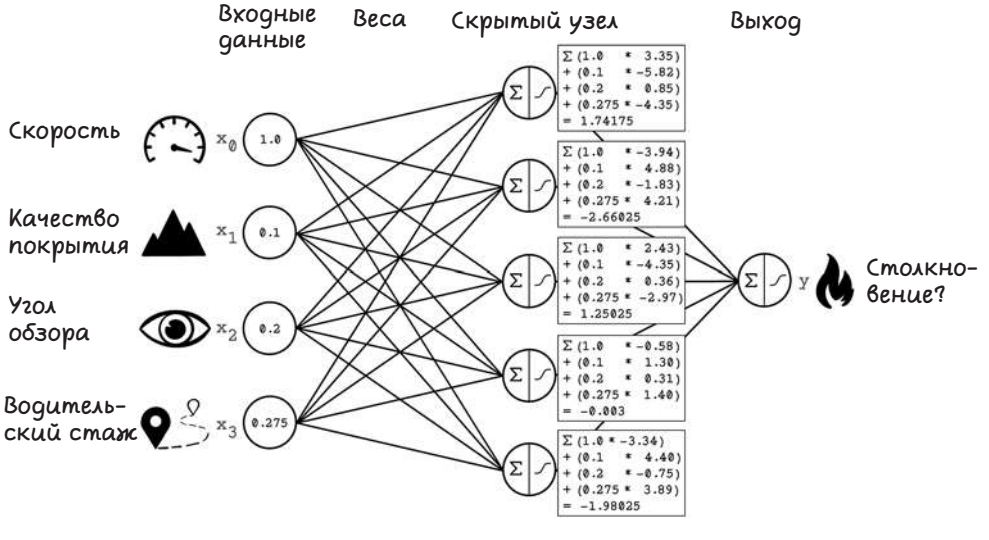
В следующем упражнении используется другой образец из датасета.

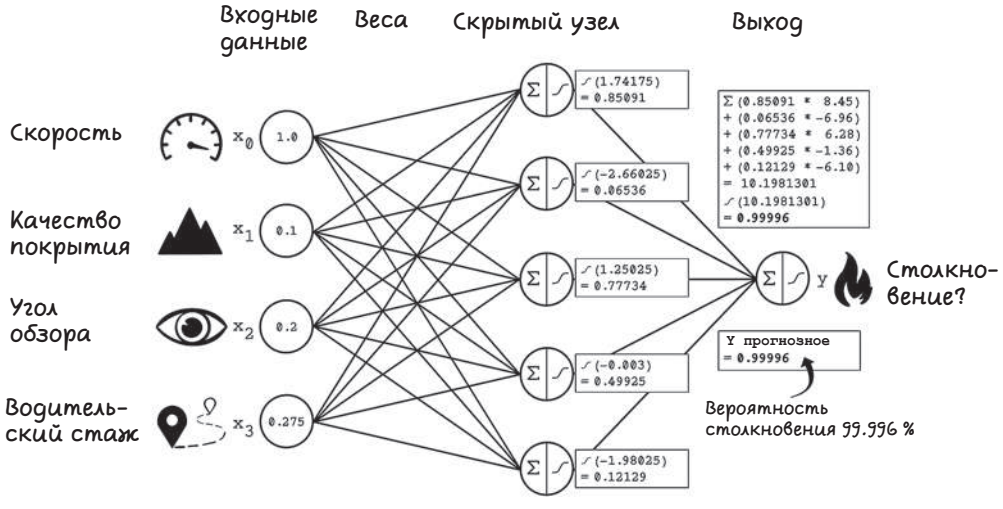
УПРАЖНЕНИЕ: ВЫЧИСЛИТЕ ПРОГНОЗ ДЛЯ ОБРАЗЦА, ИСПОЛЬЗУЯ ПРЯМОЕ РАСПРОСТРАНЕНИЕ, С ПОМОЩЬЮ СЛЕДУЮЩЕЙ ANN

	Скорость	Качество покрытия	Угол обзора	Водительский стаж	Произошло ли столкновение?
2	1.000	0.1	0.2	0.275	1



ОТВЕТ: ПРОГНОЗ ДЛЯ ОБРАЗЦА





После прогонки этого образца через обученную ANN получается результат 0.99996, иначе говоря, 99.996 % вероятности столкновения. Проанализировав этот образец с позиции человека, можно увидеть, почему вероятность столкновения столь высока. Водитель ехал на максимально разрешенной скорости по наихудшему дорожному покрытию, и угол его обзора был очень мал.

Псевдокод

Одной из важных функций активации в нашем примере выступает сигмоида. Этот метод описывает математическую функцию, которая представляет кривую S:

```
sigmoid(x):
  return 1 / (1 + exp(-x))
```

Exp — константа, число Эйлера, приблизительно равное 2.71828

Обратите внимание, что в нижеприведенном коде описывается тот же класс нейронной сети, что использовался ранее в этой главе. На этот раз в него добавлена функция `forward_propagation`. Она складывает взвешенные суммы входных значений для каждого скрытого нейрона, применяет к каждому результату сигмоиду и сохраняет полученные значения в качестве выходов нейронов скрытого слоя. Затем эта же операция совершается для выходов скрытых узлов и весов, ведущих к выходному нейрону:

```
NeuralNetwork(features, labels, hidden_node_count):
  let input equal features
  let weights_input equal a random matrix, size: features * hidden_node_count
  let hidden equal zero array, size: hidden_node_count
  let weights_hidden equal a random matrix, size: hidden_node_count
  let expected_output equal labels
  let output equal zero array,size: length of labels

  forward_propagation():
    let hidden_weighted_sum equal input · weights_input
    let hidden equal sigmoid(hidden_weighted_sum)
    let output_weighted_sum equal hidden · weights_hidden
    let output equal sigmoid(output_weighted_sum)
```

Символ · обозначает перемножение матриц.

Обратное распространение: обучение ANN

Понимание прямого распространения помогает разобраться в процессе обучения ANN. Существует также обратное распространение (часто называемое обратным распространением ошибки). Для реализации в ANN обратного распространения требуется знание жизненного цикла МО и его принципов, рассмотренных в главе 8. ANN можно рассматривать как одну из моделей МО. Нам по-прежнему нужно задавать правильный вопрос. Мы все так же собираем и анализируем данные в контексте задачи. И нам так же нужно привести эти данные в подходящий для обработки моделью вид.

Все данные будут разделены на обучающую и контрольную выборки. Помимо этого мы будем повторять и улучшать процесс, собирая дополнительные данные и меняя либо способ их подготовки, либо архитектуру и конфигурацию ANN.

Обучение ANN состоит из трех основных этапов. Этап A подразумевает настройку архитектуры сети, включая конфигурирование входов, выходов и скрытых слоев. Этап B — прямое распространение, а этап C, соответственно, обратное распространение, в ходе которого и происходит обучение (рис. 9.20).

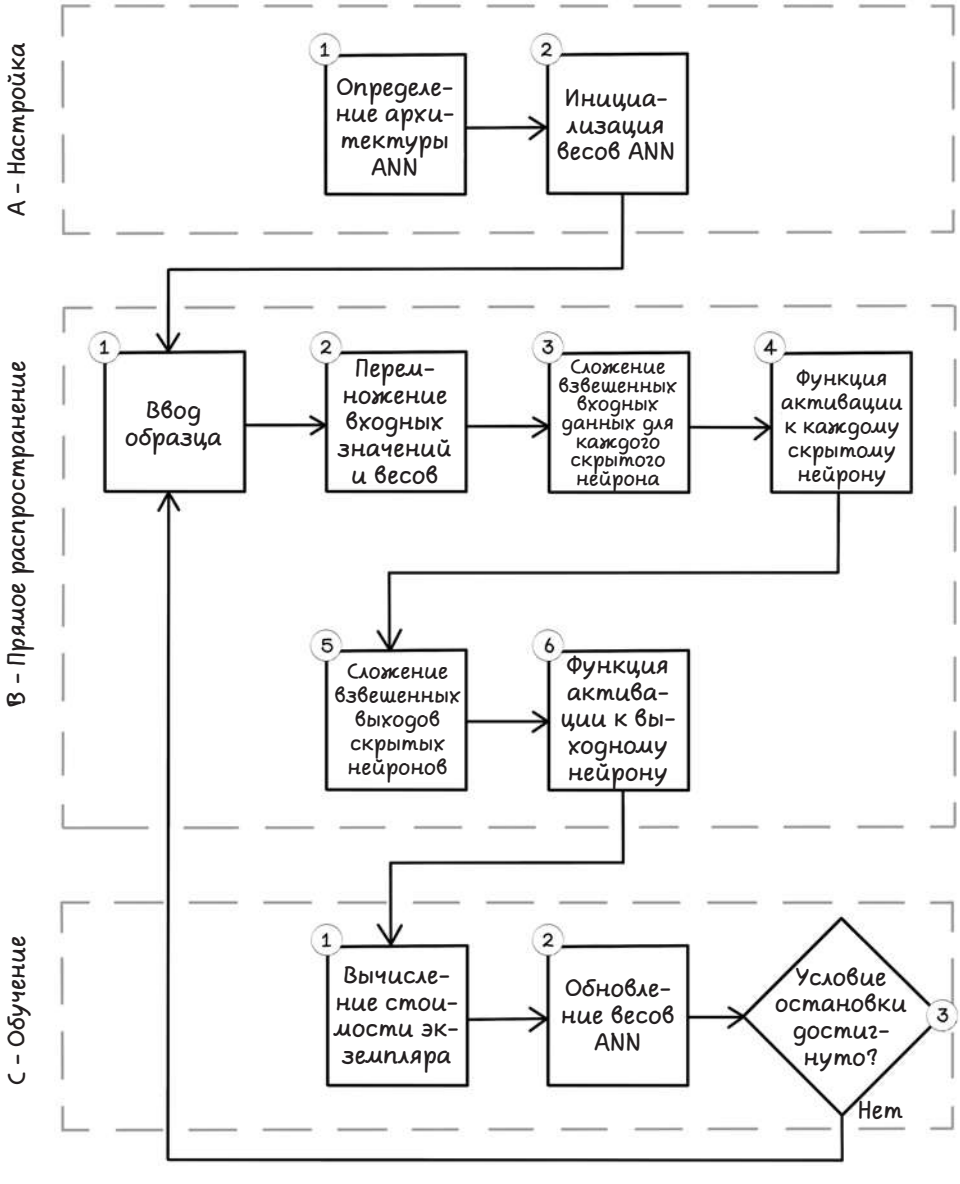


Рис. 9.20. Жизненный цикл обучения ANN

Этапы A, B и C описывают шаги и операции алгоритма обратного распространения ошибки.

Этап A: настройка

1. *Определение архитектуры ANN*. На этом шаге задаются входные нейроны, выходные нейроны, количество скрытых слоев, количество нейронов в каждом скрытом слое, функция активации и другие параметры.
2. *Инициализация весов ANN*. Веса в ANN должны инициализироваться с некоторым значением. Для этого можно использовать разные подходы. Суть в том, что по мере обучения ANN веса будут постоянно корректироваться.

Этап B: прямое распространение

Этот процесс мы подробно рассмотрели выше, разница здесь будет лишь в том, что теперь по его завершении спрогнозированный выход будет сопоставляться с фактическим классом образца, на основе чего и будет происходить обучение сети.

Этап C: обучение

1. *Вычисление стоимости*. После этапа прямого распространения определяется стоимость, которая представляет собой разность между спрогнозированным выходом и фактическим классом образца из обучающей выборки. Эта стоимость, по сути, показывает, насколько плохо ANN прогнозирует классы образцов данных.
2. *Обновление весов ANN*. Веса — это единственный параметр, который сеть может корректировать самостоятельно. Архитектура и конфигурации, которые определяются на этапе A, в процессе обучения не изменяются. Веса же, по существу, кодируют степень интеллекта сети. Они подстраиваются в сторону увеличения или уменьшения, влияя тем самым на силу входных данных.
3. *Определение условия остановки*. Обучение не может продолжаться неопределенное количество времени. Как и во многих рассмотренных нами алгоритмах, здесь требуется установить разумное условие остановки. Если обрабатывается обширный датасет, то можно, к примеру, выбрать для обучения ANN 500 образцов, которые пройдут 1000 итераций. При этом в каждой итерации будет производиться корректировка весов.

Когда мы изучали процесс прямого распространения, веса были уже настроены, так как мы использовали предобученную сеть. Сейчас же, прежде чем начать самостоятельное обучение ANN, необходимо инициализировать веса с определенным значением, после чего они будут корректироваться на основе результатов прогнозирования. Один из способов инициализации весов состоит в выборе случайных значений из нормального распределения.

На рис. 9.21 показаны случайно инициализированные веса для нашей ANN, а также вычисления для прямого распространения на примере одного обучающего образца. Здесь использован тот же образец, что и в разделе «Прямое

распространение», чтобы показать разницу в итоговом выходе, обусловленную различием весов.

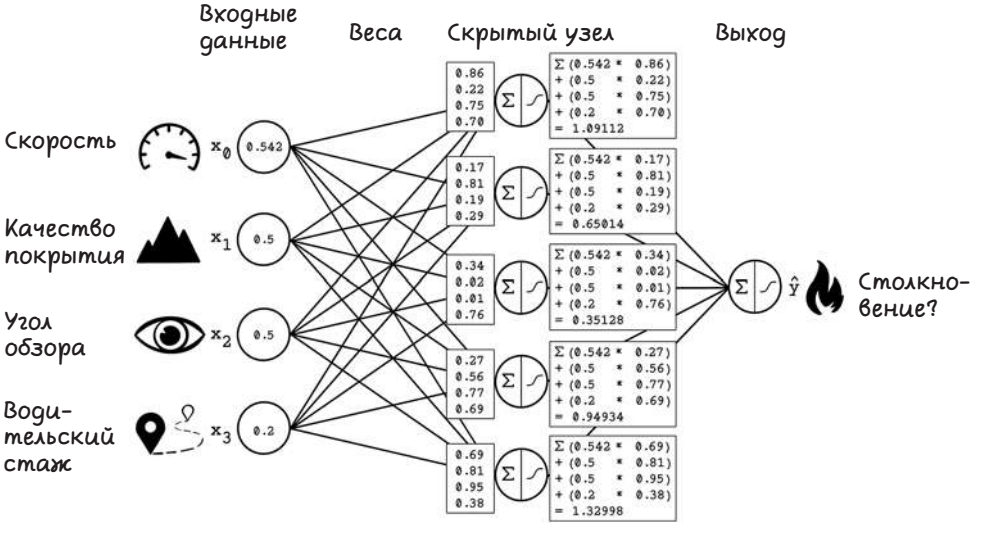


Рис. 9.21. Пример начальных весов для ANN

Следующий шаг — прямое распространение (рис. 9.22). Основные изменения — это проверка разницы между полученным прогнозом и фактическим классом.

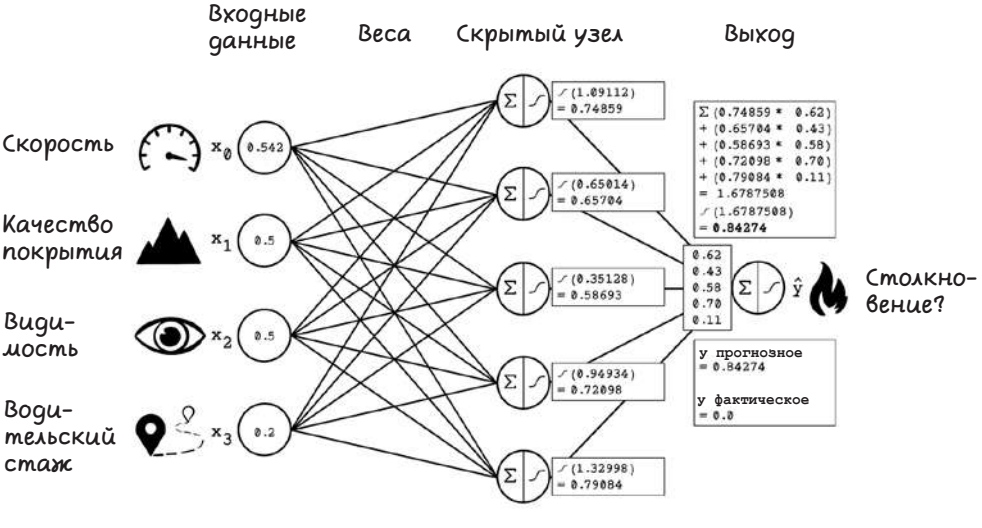


Рис. 9.22. Пример прямого распространения со случайно инициализированными весами

Сравнивая результат прогноза с фактическим классом, можно вычислить стоимость. Функция стоимости в нашем случае будет проста: разность фактического и спрогнозированного значений выхода. В этом примере 0.84274 вычитается из 0.0 и стоимость получается равной –0.84274. Данный результат показывает степень ошибочности прогноза и может использоваться для корректировки весов. В ANN при каждом вычислении стоимости веса подстраиваются на небольшое значение. Это происходит тысячи раз при использовании обучающих данных, что позволяет определить оптимальные веса, с которыми сеть делает максимально точные прогнозы. Обратите внимание, что слишком длительное обучение на одной выборке данных может привести к переобучению, о котором говорилось в главе 8.

Здесь вводится, возможно, незнакомый для вас математический аппарат, а именно правило цепи. Прежде чем использовать это правило, мы глубже разберем, что именно означают веса и как их корректировка повышает эффективность ANN.

Если перенести связь потенциальных весов и соответствующей им стоимости на график, мы получим некоторую функцию, которая представляет эти возможные веса. Некоторые точки найденной функции дают меньшую стоимость, а другие, наоборот, ее повышают. Мы же ищем те, которые дадут минимальную стоимость (рис. 9.23).

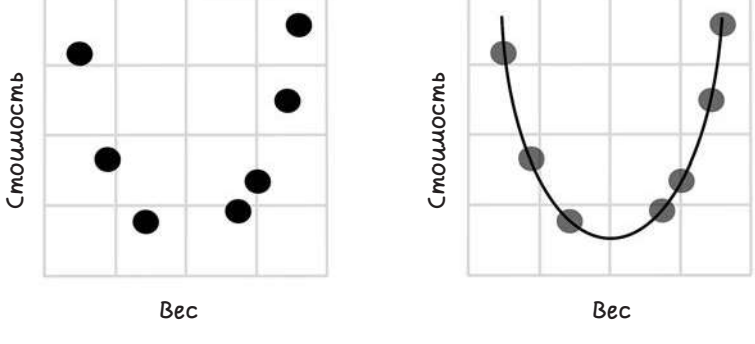


Рис. 9.23. График весов в отношении стоимости

Для этого будем использовать специальный математический инструмент, называемый *градиентным спуском*. С его помощью можно перемещать веса все ближе к минимальному значению через нахождение производной функции. Важность *производной* заключается в том, что она измеряет чувствительность функции к изменению. Например, скорость может быть производной от положения объекта в пространстве относительно времени, а производной скорости объекта относительно времени будет уже ускорение. Производные могут находить угол наклона в определенной точке функции. Градиентный спуск на основе этой информации определяет, в каком направлении нужно смещаться и насколько.

Рисунки 9.24 и 9.25 демонстрируют, как производные и угол наклона указывают направление минимумов.

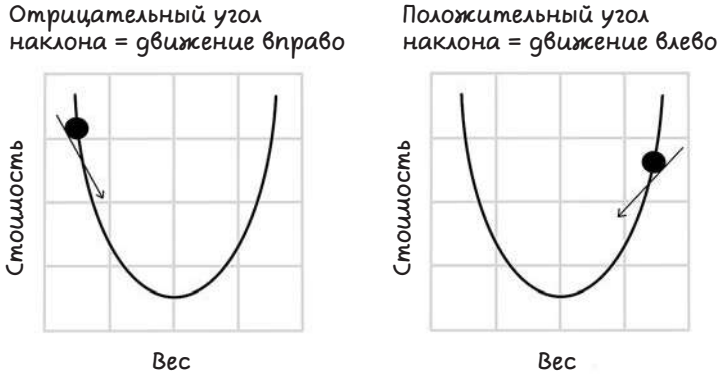


Рис. 9.24. Угол наклона производной и направление минимумов

Рис. 9.25. Пример корректировки веса с помощью градиентного спуска

При рассмотрении одного отдельного веса найти значение, минимизирующее его стоимость, может показаться простой задачей. Тем не менее итоговая стоимость в нейронной сети определяется балансированием множества весов. Некоторые веса могут быть близки к своим оптимальным точкам, некоторые нет, даже в случае, когда ANN работает хорошо.

Так как ANN формируется из многих функций, можно использовать правило цепи. Это правило представляет собой математическую теорему о вычислении производной сложной функции. В сложной функции функция `g` используется в качестве параметра функции `f`, которая сама является переменной для функции `h`, то есть, по сути, одна функция используется в качестве параметра другой функции.

На рис. 9.26 показано применение правила цепи для вычисления обновления значений весов в различных слоях ANN.

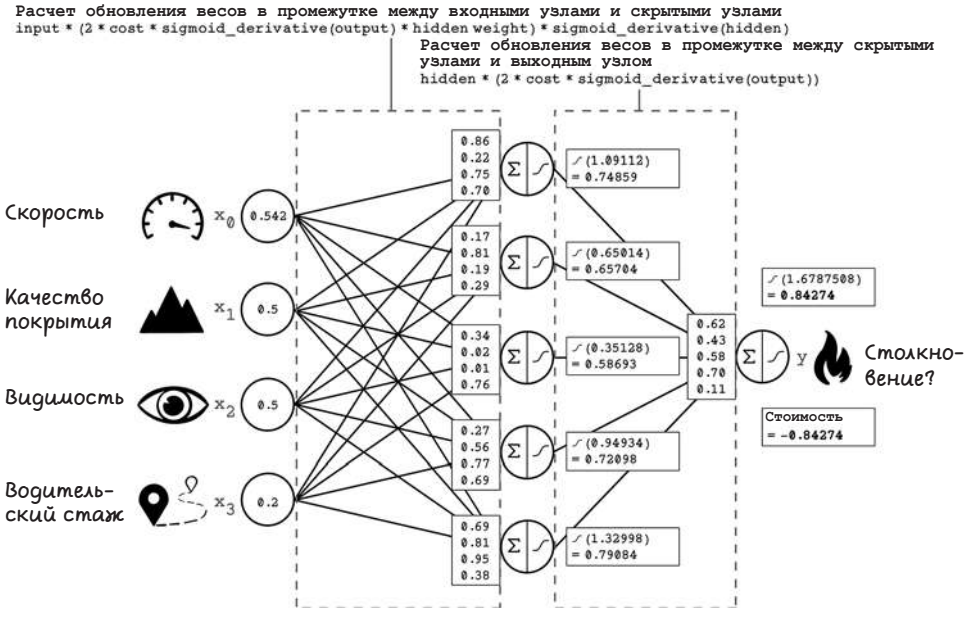


Рис. 9.26. Формула вычисления обновления весов на основе правила цепи

Обновление весов можно вычислить, подставив соответствующие значения в приведенную формулу. Эти вычисления выглядят пугающе, но обратите внимание на переменные и их роль в ANN. Несмотря на свою кажущуюся сложность, функция использует значения, которые мы уже вычисляли (рис. 9.27).

Вот подробное представление вычислений с рис. 9.27:

```
Расчет обновления весов в промежутке между входными узлами и скрытыми узлами
hidden * (2 * cost * sigmoid_derivative(output))

0.74859 * (2 * -0.84274 * sigmoid_derivative(0.84274))
= 0.74859 * (2 * -0.84274 * 0.210)
= -0.265
```

```
Расчет обновления весов в промежутке между скрытыми узлами и выходным узлом
input * (2 * cost * sigmoid_derivative(output) * hidden weight) * sigmoid_derivative(hidden)

0.542 * (2 * -0.84274 * sigmoid_derivative(0.84274) * 0.86) * sigmoid_derivative(0.74859)
= 0.542 * (2 * -0.84274 * 0.210 * 0.86) * 0.218
= -0.0360
```

После вычисления значений обновления можно прибавить их к соответствующим весам ANN. Рисунок 9.28 показывает применение результатов обновления к весам различных слоев.

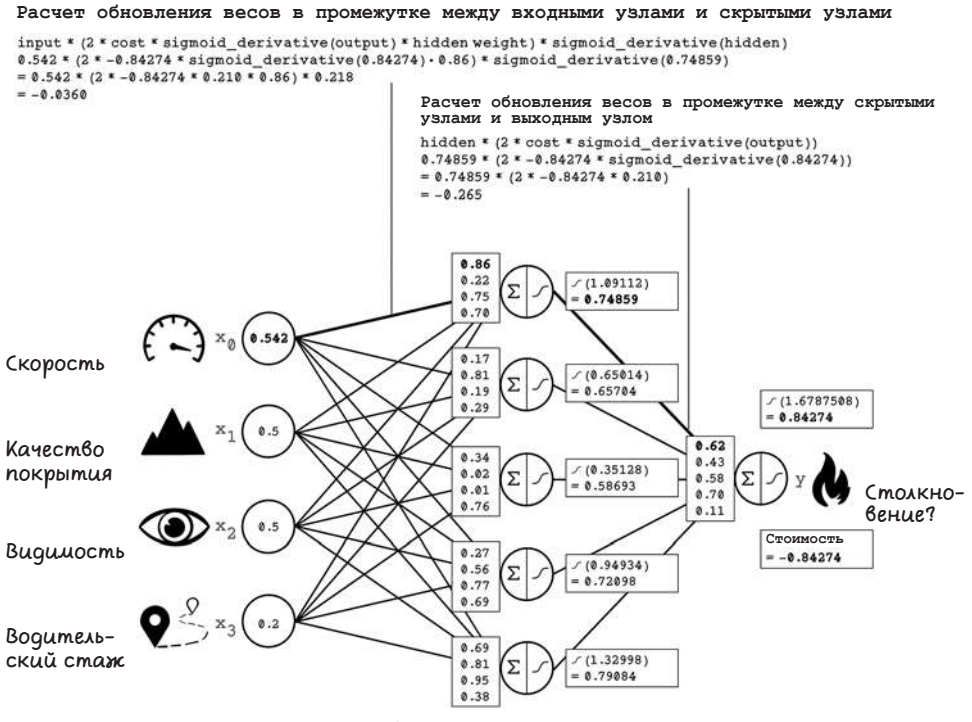


Рис. 9.27. Вычисление обновления весов с помощью правила цепи

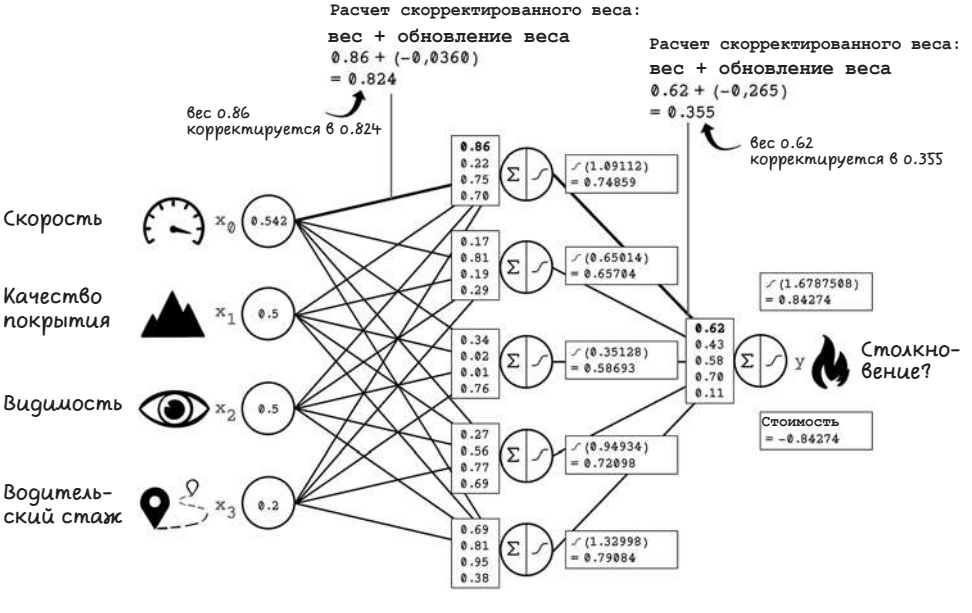
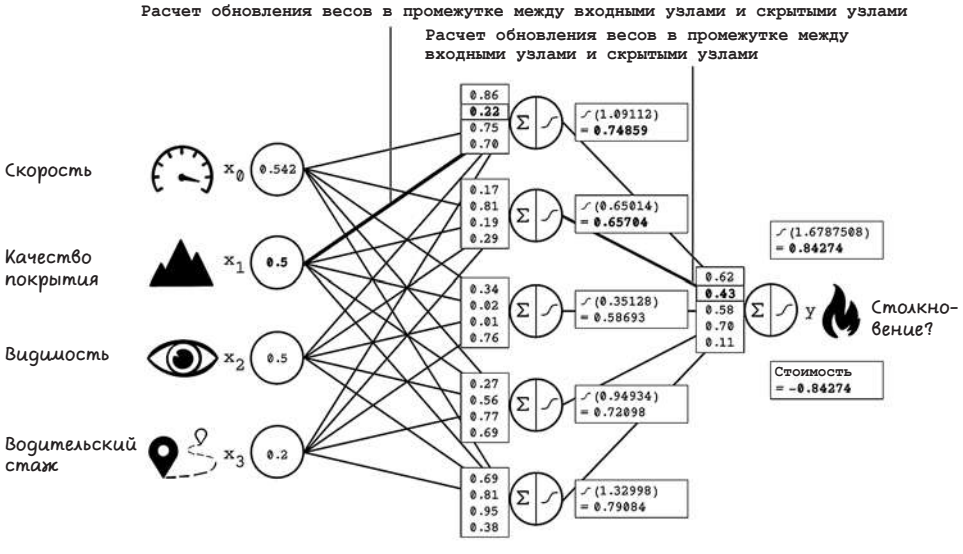
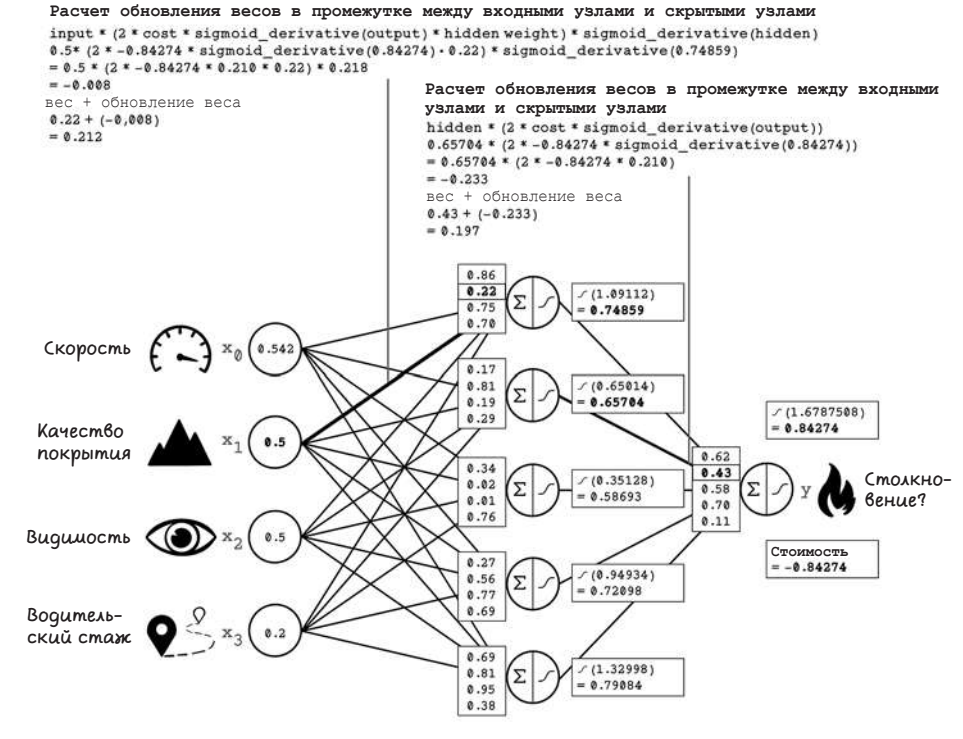


Рис. 9.28. Пример итогового обновления весов в ANN

УПРАЖНЕНИЕ: ВЫЧИСЛИТЕ НОВЫЕ ЗНАЧЕНИЯ ДЛЯ ВЫДЕЛЕННЫХ ВЕСОВ



ОТВЕТ: ВЫЧИСЛЕНИЕ НОВЫХ ЗНАЧЕНИЙ



Решаемая правилом цепи задача может напомнить вам пример с дроном из главы 7. Оптимизация роем частиц эффективна для поиска оптимальных значений в многомерных пространствах решений наподобие текущего примера, где подстраивать нужно 25 весов. Поиск весов в ANN также является задачей оптимизации. При этом для поиска можно использовать не только градиентный спуск, но и другие подходы, что зависит от контекста и самой задачи.

Псевдокод

Производная является важным элементом алгоритма обратного распространения ошибки. Следующий псевдокод описывает формулу производной сигмоидной функции, которая требуется для корректировки весов:

```
sigmoid(x):
  return 1 / (1 + exp(-x))

sigmoid_derivative(x):
  return sigmoid(x) * (1 - sigmoid(x))
```

Exp — константа, число Эйлера, приблизительно равное 2.71828

Мы снова возвращаемся к классу `NeuralNetwork`, но на этот раз он содержит функцию обратного распространения ошибки. Эта функция вычисляет стоимость, затем применяет правило цепи для получения на ее основе значения для подстройки весов, после чего прибавляет результаты к соответствующим весам. Помните, что стоимость вычисляется на основе признаков образца, спрогнозированного выхода и ожидаемого выхода. Как результат, она представляет разность между спрогнозированным выходом и ожидаемым:

```
NeuralNetwork(features, labels, hidden_node_count):
  let input equal features
  let weights_input equal a random matrix, size: features * hidden_node_count
  let hidden equal zero array, size: hidden_node_count
  let weights_hidden equal a random matrix, size: hidden_node_count
  let expected_output equal labels
  let output equal zero array,size: length of labels

  back_propagation():
    let cost equal expected_output - output
    let weights_hidden_update equal
        hidden • (2 * cost * sigmoid_derivative(output,
    let weights_input_update equal
        input • (2 * cost * sigmoid_derivative(output) * weights_hidden)
        * sigmoid_derivative(hidden)
```

Символ • обозначает перемножение матриц

```
let weights_hidden equal weights_hidden + weights_hidden_update
let weights_input equal weights_input + weights_input_update
```

У нас есть класс, представляющий нейронную сеть, функции для масштабирования данных, а также функции для прямого и обратного распространения ошибки. Все это можно объединить для обучения нейронной сети.

Псевдокод

В этом фрагменте кода присутствует функция `run_neural_network`, получающая в качестве входа `epochs` (количество эпох, то есть количество итераций обучения). Эта функция масштабирует данные и создает новую нейронную сеть с полученными масштабированными значениями, метками и определенным числом скрытых нейронов. Далее она выполняет `forward_propagation` (прямое распространение) и `back_propagation` (обратное распространение) для заданного количества `epochs`:

```
run_neural_network(epochs):
  let scaled_feature_data equal
    scale_dataset(feature_data, feature_count, features_min, features_max)
  let nn equal NeuralNetwork(scaled_feature_data,
                             scaled_label_data,
                             hidden_node_count)
  for epoch in range(epochs):
    nn.forward_propagation()
    nn.back_propagation()
```

Варианты функций активации

Этот раздел подробнее описывает функции активации и их свойства. Когда мы говорили про перцептрон и ANN, то использовали в качестве такой функции сигмоиду, что вполне подходило для наших задач. Функции активации вводят в ANN нелинейные свойства. Если их не использовать, то нейронная сеть будет вести себя аналогично регрессии, описанной в главе 8. На рис. 9.29 приводятся наиболее распространенные функции активации.

Разные функции подходят для разных сценариев и имеют свои преимущества:

- *Функция Хевисайда* — используется в качестве бинарного классификатора. Получая входное значение между –1 и 1, на выходе она дает 0 или 1. Двоичный классификатор не эффективен для обучения на данных в скрытом слое, но его можно использовать в выходном слое для бинарной классификации. К примеру, если требуется узнать, представлена в образце кошка или собака, то 0 может обозначать кошку, а 1 собаку.

- *Сигмоида* — сигмоидная функция на основе входа между –1 и 1 формирует кривую S между 0 и 1. Так как она позволяет посредством изменения x вызывать небольшие изменения в y, то ее можно использовать для решения нелинейных задач. Иногда ее применение вызывает проблему, когда при приближении значений к экстремумам производная изменяется незначительно, что приводит к неэффективному обучению. Эта проблема известна как *затухание градиента*.
- *Гиперболический тангенс* — функция гиперболического тангенса аналогична сигмоиде, но результируется в значения между –1 и 1. Ее преимущество в том, что она дает более крутые производные, положительно сказывающиеся на скорости обучения. Для этой функции также свойственна проблема затухающих градиентов.
- *Блок линейной ректификации (ReLU)* — функция ReLU дает 0 для входных значений между –1 и 0, а для значений между 0 и 1 дает линейно возрастающие величины. В обширных ANN с большим количеством нейронов при использовании функций сигмоиды или гиперболического тангенса все нейроны активируются постоянно (за исключением случаев, когда они дают 0), в связи с чем поиск решений требует объемных вычислений и тонкой подстройки множества значений. Функция ReLU же позволяет некоторым нейронам оставаться неактивными, что снижает количество вычислений и ускоряет нахождение решений.

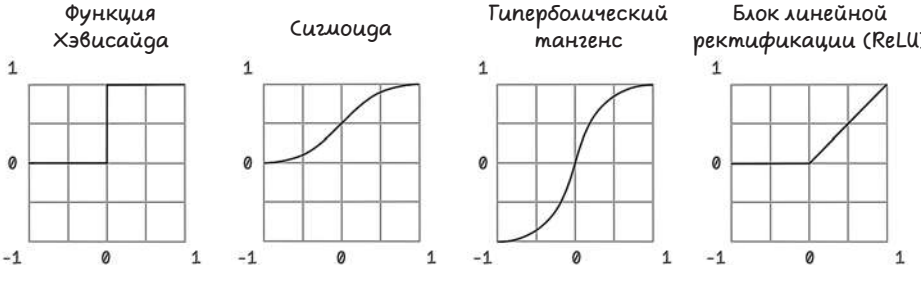


Рис. 9.29. Распространенные функции активации

Следующий раздел затрагивает некоторые вопросы проектирования ANN.

Проектирование искусственных нейронных сетей

Проектирование ANN осуществляется экспериментальным путем и зависит от решаемой задачи. Чтобы повысить эффективность прогнозирования, архитектуру и конфигурацию сети обычно меняют путем проб и ошибок. В этом разделе кратко перечисляются параметры архитектуры, которые можно изменять для повышения эффективности сети или решения отдельных проблем. На рис. 9.30

показана искусственная нейронная сеть, чья конфигурация отличается от той, что мы рассматривали прежде. Наиболее заметное отличие в том, что здесь появился новый скрытый слой и у сети теперь есть два выходных нейрона.

> ***Примечание***
>
> Как и в большинстве научных или инженерных задач, ответом на вопрос «Какова идеальная структура ANN?» обычно будет «Зависит от…». Настройка ANN требует глубокого понимания данных и решаемой задачи. Четко оформленной обобщенной схемы архитектуры и конфигурации пока не существует.

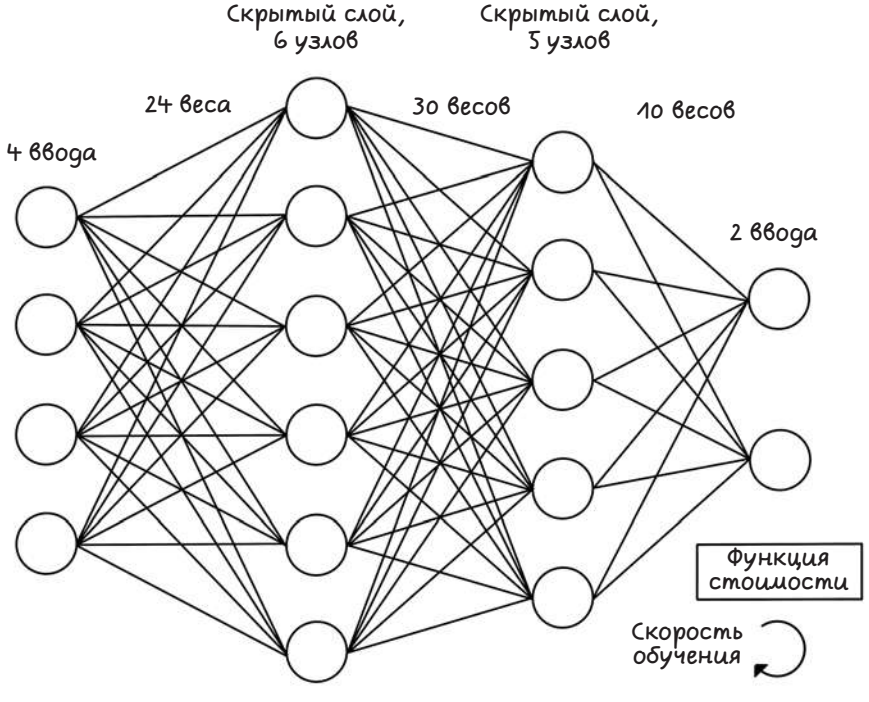


Рис. 9.30. Пример многослойной ANN с двумя выходами

Входы и выходы

Входы и выходы ANN являются фундаментальными параметрами нейронной сети. После того как модель ANN обучена, предполагается, что ее будут использовать разные люди в различных контекстах и системах. Входы и выходы определяют интерфейс сети. В этой главе мы рассматривали пример ANN с четырьмя входами, описывающими признаки сценария ДТП, и одним выходом, выра-

жающим вероятность столкновения. Однако когда входы и выходы означают разные вещи, может возникнуть проблема. К примеру, если дано изображение рукописной цифры размером 16 на 16 пикселей, можно использовать пиксели в качестве входных значений, а представляемую ими цифру — в качестве выходного. Вход будет состоять из 256 узлов, несущих значения пикселей, а выход из 10 узлов, представляющих цифры от 0 до 9, где каждый результат указывает на вероятность принадлежности изображения к соответствующей цифре.

Скрытые слои и нейроны

ANN может состоять из нескольких скрытых слоев с различным количеством нейронов в каждом. Добавление слоев позволяет решать проблемы с большим количеством измерений и повышенной сложностью линии разделения классов. В примере на рис. 9.8 простой прямой линии достаточно для точной классификации данных. Иногда линия оказывается нелинейной, но сохраняет простоту. Но что, если это будет более сложная функция со множеством кривых, потенциально проходящих через множество измерений, и мы не сможем ее визуализировать? Добавление слоев позволяет находить эти сложные функции классификации. Выбор числа слоев и нейронов обычно происходит опытным путем с внесением доработок. Однако со временем, решая однотипные задачи, можно научиться интуитивно подбирать подходящие конфигурации.

Веса

Важность инициализации весов состоит в том, что с ее помощью задается начальная точка, от которой веса будут неоднократно корректироваться. Если веса инициализируются слишком малыми значениями, это ведет к проблеме затухающего градиента, а в случае выбора слишком больших начальных значений возникает *проблема взрывающегося градиента*, подразумевающая беспорядочное блуждание весов вокруг нужного результата.

Существуют различные схемы инициализации весов, у каждой из которых есть как плюсы, так и минусы. Общее правило — обеспечить, чтобы среднее результатов активации слоя (то есть среднее всех результатов скрытых нейронов слоя) равнялось 0. Помимо этого, дисперсия результатов активации должна быть одинаковой, то есть вариативность результатов каждого скрытого нейрона должна оставаться согласованной в течение нескольких эпох.

Смещение

Смещение в ANN можно использовать, добавляя значения к взвешенной сумме входных нейронов или других слоев в сети. Смещение производит сдвиг итогового значения функции активации. При этом оно обеспечивает гибкость ANN и сдвиг функции активации влево или вправо.

Проще всего понять смещение, представив линию, которая всегда проходит через точку (0,0) на плоскости. Можно влиять на эту линию так, чтобы она

отсекала отрезки на осях координат, добавляя +1 к переменной. При этом выбираемое значение смещения будет зависеть от решаемой задачи.

Функции активации

Ранее мы кратко рассмотрели типичные функции активации, применяемые в ANN. Ключевое правило при их определении — обеспечить, чтобы у всех нейронов одного слоя была одинаковая функция. В многослойных сетях разные слои могут использовать разные функции активации, что определяется решаемой задачей. К примеру, сеть, прогнозирующая одобрение займов, должна использовать в скрытых слоях сигмоиду для определения вероятностей, а в выходном нейроне функцию Хевисайда для получения четкого решения: 0 или 1.

Функция стоимости и скорость обучения

В приведенном ранее примере мы использовали простую функцию стоимости, подразумевавшую вычитание спрогнозированного выхода из фактически ожидаемого. Тем не менее функции стоимости бывают разные. Они существенно влияют на ANN, поэтому важно подбирать для задачи и датасета именно подходящую функцию, так как она описывает цель ANN. Одна из наиболее распространенных функций — *среднеквадратичная ошибка*, аналогичная функции, использованной в главе 8 о машинном обучении. Выбирать подходящий вариант нужно на основе понимания данных, их размера, а также желаемого способа измерения точности и полноты. В ходе работы следует рассматривать различные функции стоимости.

И наконец, скорость обучения ANN описывает, насколько существенно корректируются веса в процессе обратного распространения ошибки. Низкая скорость может значительно растянуть обучение, потому что веса будут каждый раз обновляться на малые значения. Высокая же скорость может привести к заметным изменениям весов и сделать процесс обучения хаотичным. Одно из возможных решений — начать с фиксированной скорости и менять ее, если обучение будет впадать в стагнацию, не приводя к улучшению результатов. Этот повторяющийся процесс требует экспериментирования. В качестве оптимизатора можно использовать стохастический градиентный спуск. Работает он аналогично градиентному спуску, но при этом позволяет весам выпрыгивать из локальных минимумов для поиска лучших решений.

Стандартные ANN, подобные той, которую мы рассмотрели в текущей главе, применимы для нелинейных задач классификации. Если нужно категоризировать образцы на основе множества признаков, то такой вид ANN скорее всего справится с задачей.

При всем сказанном отметим, что ANN — не идеальный инструмент на все случаи жизни. Нередко во многих стандартных ситуациях более эффективны простые традиционные алгоритмы МО, описанные в главе 8. Вспомните жизненный цикл машинного обучения. Попробуйте использовать несколько моделей МО и определить самую эффективную.

Виды искусственных нейронных сетей и их применение

ANN универсальны, и разные виды их архитектур могут служить для решения соответствующих им задач. Вид архитектуры ANN можно рассматривать как фундаментальную конфигурацию нейронной сети. Разберем некоторые из возможных конфигураций на примерах.

Сверточные нейронные сети

Сверточные нейронные сети (CNN — convolutional neural networks) разрабатываются для распознавания изображений. Эти сети можно использовать для нахождения связей между разными объектами и уникальными областями изображений. При *распознавании изображений* операция свертки производится для пикселя и его соседей в определенном радиусе. Эта техника традиционно применяется для обнаружения контуров, повышения резкости, а также размытия изображения. CNN используют операцию свертки и объединения, или пулинга (pooling), для нахождения связей между пикселями изображения. *Свертка* находит признаки изображения, а *пулинг* субдискретизирует найденные паттерны, суммируя признаки, что позволяет согласованно кодировать уникальные признаки из множества обучающих образцов изображений (рис. 9.31).

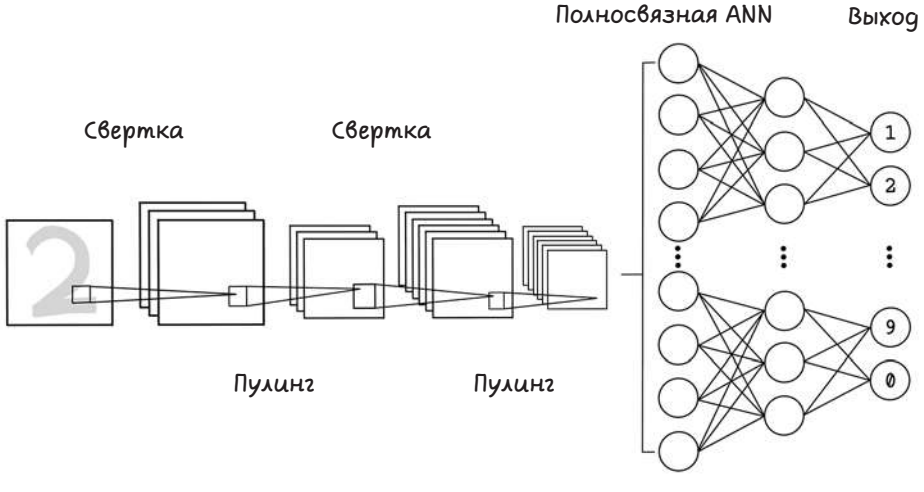


Рис. 9.31. Простой пример CNN

CNN используются для классификации изображений. Вам наверняка приходилось искать картинки онлайн. В этом процессе вы косвенно взаимодействовали именно с CNN. Эти сети также эффективны для оптического распознавания символов с целью извлечения текстовых данных из изображений. CNN нашли применение и в сфере здравоохранения, где их используют для выявления

аномалий и определения медицинских состояний по рентгеновским снимкам, МРТ и прочим диагностическим исследованиям.

Рекуррентные нейронные сети

В то время как стандартные ANN получают фиксированное количество входных данных, *рекуррентные нейронные сети* (RNN — recurrent neural networks) принимают на входе последовательность значений без предопределенной длины. Эти входные данные подобны предложениям устной речи. В RNN реализован принцип памяти, состоящей из скрытых слоев, которые представляют движение времени. Этот принцип позволяет сети удерживать информацию о связях между последовательностями входных значений. При обучении RNN веса скрытых слоев также с течением времени корректируются методом обратного распространения ошибки. Несколько весов представляют один и тот же вес в разных точках времени (рис. 9.32).

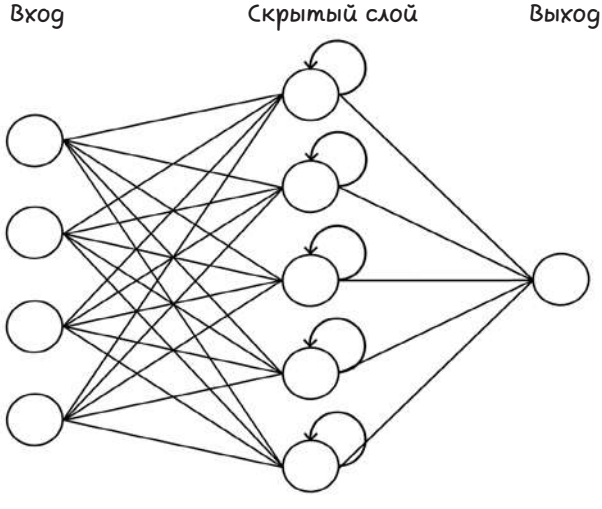


Рис. 9.32. Простой пример RNN

RNN применяются в областях, связанных с распознаванием и прогнозированием речи или текста. Они используются для автозавершения предложений в мессенджерах, а также перевода речи в текст или с одного языка на другой.

Генеративно-состязательные сети

Генеративно-состязательная сеть (GAN — generative adversarial network) состоит из сети-генератора и сети-дискриминатора. *Генератор* создает потенциальное решение, например изображение или ландшафт, а *дискриминатор* использует реальные изображения ландшафтов для определения степени реализма или верности сгенерированного образца. Ошибка или стоимость передается обрат-

но в сеть для совершенствования ее способности генерировать убедительные ландшафты и определять их правильность. Термин *состязательная* является ключевым. Его суть пояснялась на примере с генерацией дерева игры в главе 3. Два компонента соревнуются в эффективности и таким образом генерируют все более качественные решения (рис. 9.33).

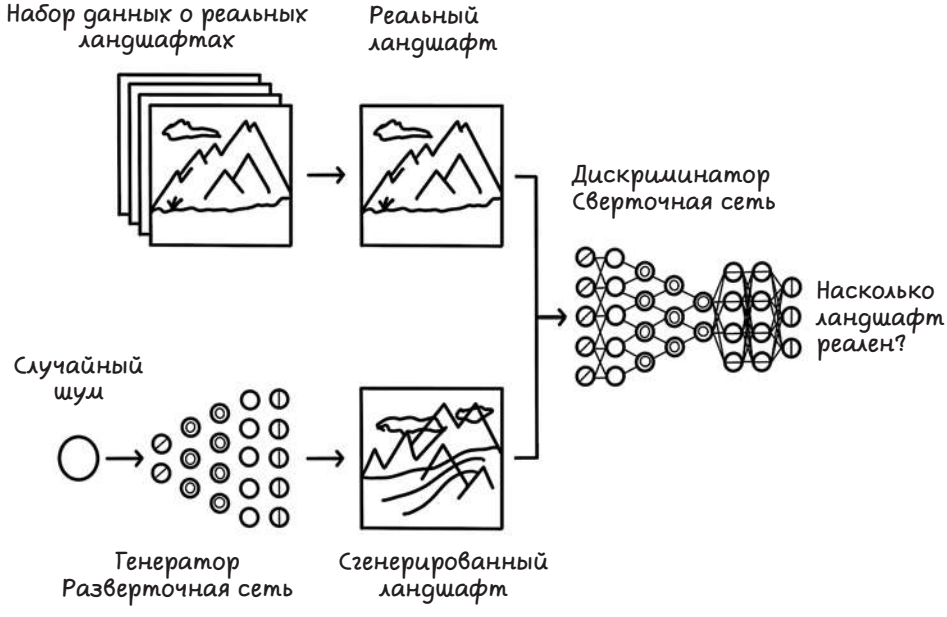


Рис. 9.33. Простой пример GAN

GAN используются для создания убедительных поддельных видео (дипфейков) известных людей, что делает актуальным вопрос подлинности информации в СМИ. Эти сети применяются и в других областях, таких как моделирование причесок или генерация трехмерных объектов из двумерных изображений, например 3D-стула из соответствующей 2D-картинки. Этот случай применения может казаться незначительным, но он свидетельствует, что сеть точно оценивает и создает информацию на основе неполного источника. Это огромный шаг в развитии ИИ и технологий в целом.

Текущая глава объединяет принципы машинного обучения и в некотором смысле загадочного мира ANN. Для дальнейшего знакомства с ANN и глубоким обучением рекомендуем книгу Grokking Deep Learning издательства Manning[1]. А в качестве практического руководства к инструменту для разработки ANN используйте другую книгу того же издательства под названием Deep Learning with Python[2].

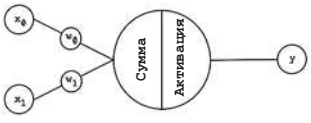
---

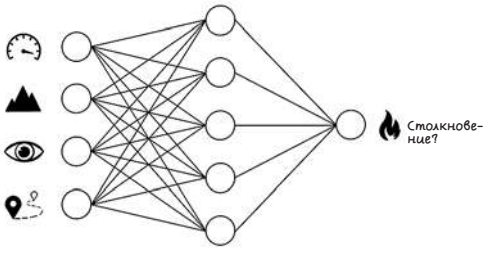
[1] *Траск Э.* Грокаем глубокое обучение. СПб, Издательство «Питер»

[2] *Шолле Ф.* Глубокое обучение на Python. СПб, Издательство «Питер»

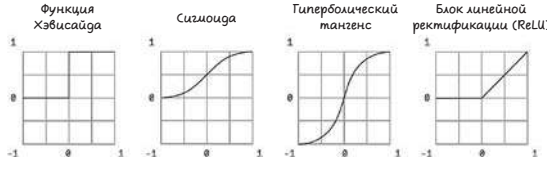
Краткий обзор главы «Искусственные нейронные сети»

Искусственные нейронные сети (ANN) основаны на принципах работы человеческого мозга, и их можно рассматривать как одну из моделей машинного обучения.



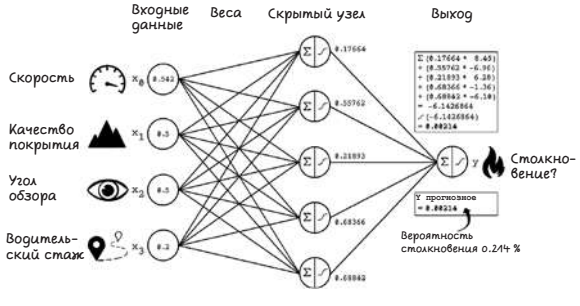


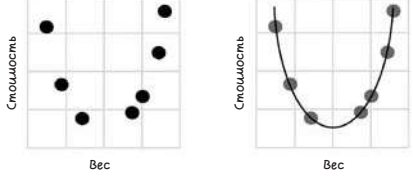
В основе работы ANN лежит принцип перцептрона.



Функция активации служит для решения нелинейных задач.

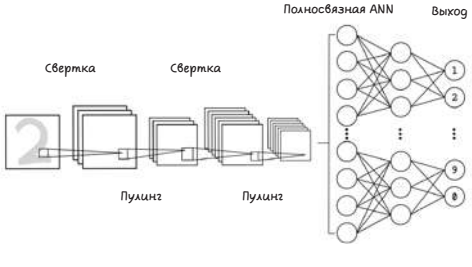
Прямое распространение используется для обучения и для составления прогнозов.





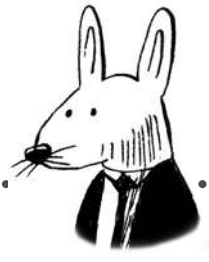
Градиентный спуск — один из вариантов оптимизации.

Искусственные нейронные сети достаточно гибкие, их можно адаптировать для решения широкого круга задач.



Обучение с подкреплением с помощью Q-Learning

10



В этой главе

- ✓ Основы обучения с подкреплением
- ✓ Какие задачи решает обучение с подкреплением
- ✓ Проектирование и реализация алгоритма обучения с подкреплением
- ✓ Различные подходы обучения с подкреплением

Что такое обучение с подкреплением?

Обучение с подкреплением (RL — reinforced learning) — это область машинного обучения, построенная на принципах поведенческой психологии. В ее основе лежит концепция суммарных наград или штрафов за действия, совершаемые агентом в динамической среде. Представьте взрослеющую собаку. Собака — это агент в среде дома. Когда мы хотим, чтобы она села, то обычно говорим: «Сидеть». Собака не понимает человеческий язык, поэтому нам нужно дать ей подсказку, слегка надавив на заднюю часть туловища. После того как она сядет, мы обычно ее гладим или даем что-нибудь вкусное. Этот процесс потребуется повторить не один раз, но спустя какое-то время мы сформируем идею сидения с помощью такого положительного подкрепления. Триггером в среде будет служить команда «Сидеть»; изученным поведением будет сидение, а наградой — поглаживание или лакомство.

Обучение с подкреплением — это еще один подход МО наряду с *обучением с учителем* и *без учителя*. В обучении с учителем прогнозирование и классификация осуществляются на основе размеченных данных. В обучении без учителя задействуются уже неразмеченные данные для поиска кластеров и закономерностей. Обучение же с подкреплением основано на результате выполняемых действий, что позволяет агенту запоминать, какие действия или их последовательности оказываются наиболее выгодными для достижения цели в тех или иных сценариях. Эта методика эффективна, когда известна цель, но не ясно, какие конкретные действия необходимы для ее достижения. На рис. 10.1 приведена карта концепций МО, на которой обозначена позиция этого вида обучения.

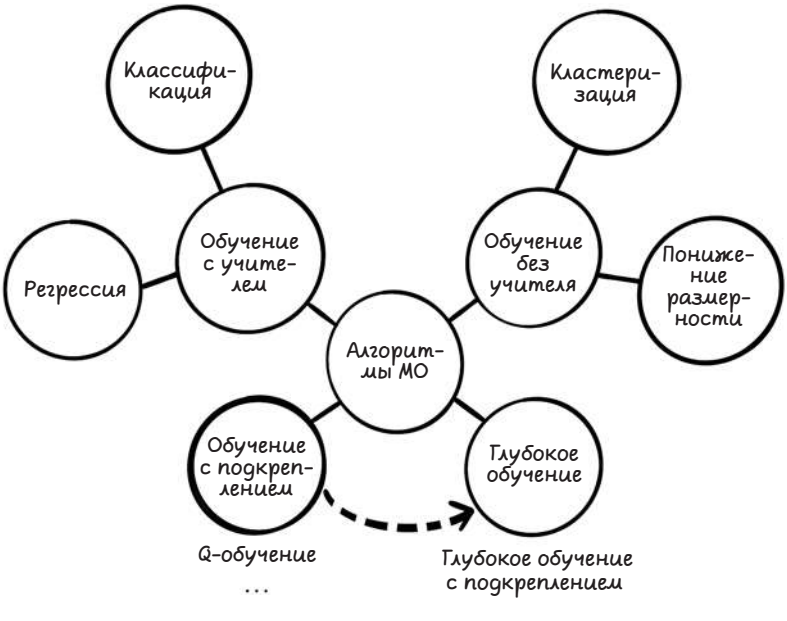


Рис. 10.1. Положение обучения с подкреплением в поле МО

Обучение с подкреплением можно реализовывать с помощью классических техник или глубокого обучения с использованием нейронных сетей. Эффективность выбранного подхода будет зависеть от области задачи.

Рисунок 10.2 схематично показывает, когда и какая техника МО может использоваться. Мы же в этой главе будем изучать обучение с подкреплением, используя классические методы.

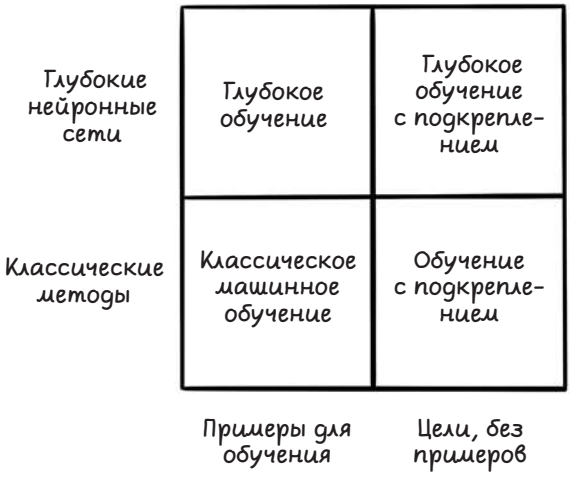


Рис. 10.2. Категоризация машинного обучения, глубокого обучения и обучения с подкреплением

Основы обучения с подкреплением

Как уже говорилось, эта техника основана на поведенческой психологии, области, изучающей поведение людей и животных. В ней поведение обычно трактуется как непроизвольные (рефлекторные) реакции или действия, выработанные на основе жизненного опыта индивидуума. Последнее как раз и относится к сути обучения с подкреплением, основанного на наградах и наказаниях, мотиваторах поведения и особенностях окружающей среды, влияющих на поведение индивидуума.

Метод проб и ошибок — один из наиболее типичных способов, с помощью которого животные определяют, что для них выгодно, а что нет. Он подразумевает попытку совершения различных потенциально неудачных действий в поиске такого, которое даст желаемый результат. Этот процесс может повторяться многократно, пока не завершится успехом, и в его основе лежит награда, мотивирующая животное продолжать попытки.

Такое поведение можно наблюдать повсюду в природе. Новорожденные цыплята, к примеру, пытаются клевать всякую мелочь, попадающуюся им на земле. В итоге путем проб и ошибок они обучаются подбирать только съедобные объекты.

Еще один пример — шимпанзе, которые на собственном опыте учатся понимать, что копать почву легче палкой, чем руками. Важными компонентами обучения с подкреплением являются цели, награды и штрафы. Цель шимпанзе — найти пищу; наградой или штрафом может стать количество выкопанных в процессе поиска ям или время, потребовавшееся для выкапывания отдельной ямы. Чем быстрее обезьяна роет землю, тем быстрее она находит еду.

Рисунок 10.3 объясняет некоторые термины из области обучения с подкреплением на простом примере дрессуры собаки.

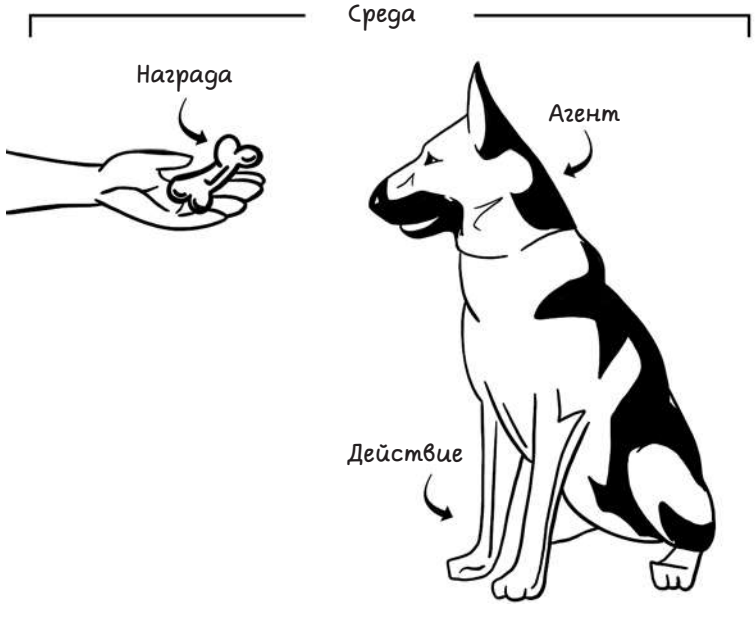


Рис. 10.3. Пример обучения с подкреплением: обучение собаки команде «сидеть» с использованием корма в качестве награды

Подкрепление бывает положительным и отрицательным. *Положительное* подразумевает получение награды за выполнение определенного действия, например, собака получает корм после того, как сядет. *Отрицательное* же означает получение штрафа за выполнение действия, например, собаку ругают, когда она дерет ковер. Положительное подкрепление повышает мотивацию на нужное поведение, а отрицательное, наоборот, отучает поступать нежелательно.

Еще один принцип обучения с подкреплением — это баланс мгновенного вознаграждения и долгосрочных последствий. Примером *мгновенного вознаграждения* можно назвать съедание плитки шоколада, которая повышает уровень глюкозы и энергии. Тем не менее, если съедать плитку шоколада каждые

полчаса, это, скорее всего, будет чревато проблемами со здоровьем, что относится к *долгосрочным последствиям*. Обучение с подкреплением направлено на получение максимальной долгосрочной выгоды, хотя краткосрочная выгода также может ей способствовать.

Обучение с подкреплением сосредоточено на долгосрочных последствиях действий в среде, в связи с чем важную роль играют время и последовательность этих действий. Предположим, что мы оказались в дикой местности и наша цель — выжить как можно дольше и в то же время проходить максимальное расстояние в надежде отыскать укрытие. Мы находимся у реки, и у нас есть два варианта: прыгнуть в воду, чтобы быстро спуститься вниз по течению, или пойти вдоль реки. Обратите внимание на лодку, стоящую возле берега, которая показана на рис. 10.4. Пустившись вплавь, мы будем двигаться быстро, но если течение понесет нас по правому рукаву реки, то лодка останется позади. Если же пойти пешком, то мы гарантированно доберемся до лодки, что существенно облегчит оставшуюся часть пути, но изначально мы этого не знаем, потому что не видим лодки. Данный пример показывает, насколько важна последовательность действий в обучении с подкреплением. Он также демонстрирует, как мгновенное вознаграждение может навредить в долгосрочной перспективе. Кроме того, в сценарии, где отсутствует лодка, мы будем перемещаться быстрее, плывя по реке, но в итоге окажемся в мокрой одежде, что создаст проблему, когда станет холодно. Если же пойти пешком, то хоть мы и будем перемещаться медленнее, зато останемся в сухой одежде. Это говорит о том, что одно и то же действие может работать в одном сценарии, но не подходить для других. Чтобы найти общий подход, требуется изучить множество симуляций.

Рис. 10.4. Пример возможных действий с долгосрочными последствиями

Применение обучения с подкреплением

Подводя итог, можно сказать, что обучение с подкреплением занимается решением задач, в которых известна цель, но неизвестны действия для ее достижения. Эти задачи подразумевают контроль поведения агента в среде. Отдельные его действия награждаются больше, чем другие, но главное — суммарная награда за все действия.

Обучение с подкреплением оказывается наиболее эффективным в задачах, где к цели ведет сумма отдельных действий. Например, это стратегическое планирование, автоматизация промышленных процессов или роботостроение. В этих предметных областях для достижения конечной цели отдельные действия могут быть не самыми оптимальными. Представьте стратегическую игру, скажем шахматы. Некоторые ходы применительно к ситуации на доске могут выглядеть неудачными. Но они помогают выстроить стратегию дальнейших действий, которые приведут к победе.

Обучение с подкреплением отлично подходит в тех областях, где для достижения хорошего результата важно выстроить цепочку событий.

В качестве примера для проработки этапов алгоритма обучения с подкреплением мы, как и в главе 9, обратимся к задаче о ДТП, но теперь рассмотрим парковку, на которой на основе визуальных данных беспилотный автомобиль должен будет найти дорогу к владельцу. Предположим, что у нас есть карта парковки, на которой отмечены сам беспилотный автомобиль, другие автомобили и пешеходы. Наш автомобиль может ехать на север, юг, восток и запад. Другие машины и пешеходы в этом примере будут неподвижны.

Цель нашего автомобиля — найти дорогу к владельцу, сталкиваясь с как можно меньшим количеством машин и пешеходов, а в идеале не сталкиваясь вообще. Столкновение с машинами ведет к их повреждению, но в случае с пешеходом последствия оказываются гораздо серьезнее. В этой задаче необходимо минимизировать количество столкновений, но если будет возникать выбор между наездом на пешехода или другую машину, то выбирать нужно наезд на машину. Схема сценария изображена на рис. 10.5.

На примере этой задачи мы рассмотрим применение обучения с подкреплением для определения и запоминания правильных действий в динамической среде.

Жизненный цикл обучения с подкреплением

Как и любые другие алгоритмы МО, эту модель перед использованием необходимо обучить. Этап обучения основан на изучении среды и получении обратной связи, исходя из конкретных действий, выполненных в конкретных обстоятельствах или состояниях. В основе жизненного цикла данной модели лежит *марковский процесс принятия решений*, предоставляющий математическую структуру для моделирования решений (рис. 10.6). Путем квантификации принятых решений и их результатов можно обучить модель понимать, какие из решений наиболее эффективны в достижении цели.

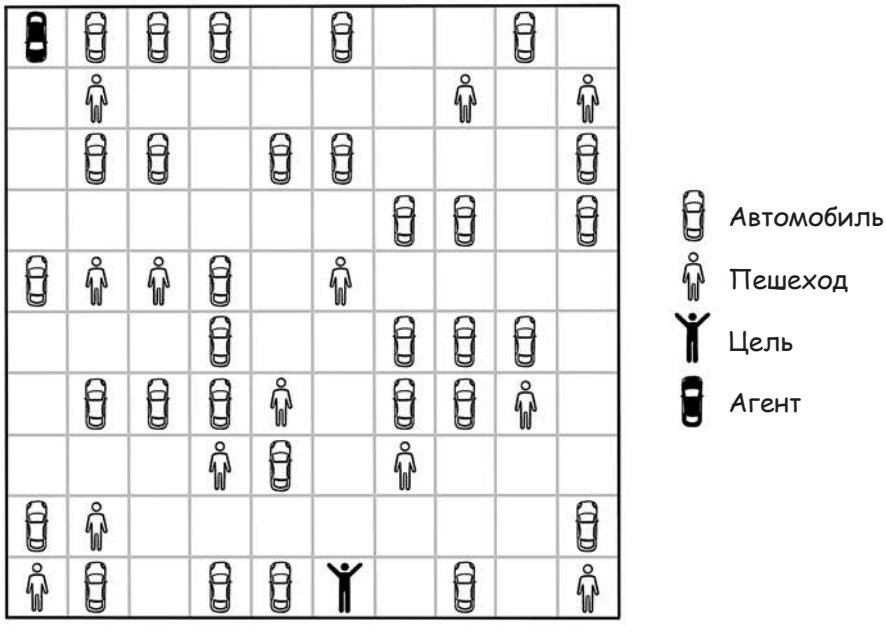


Рис. 10.5. Беспилотный автомобиль в задаче с парковкой

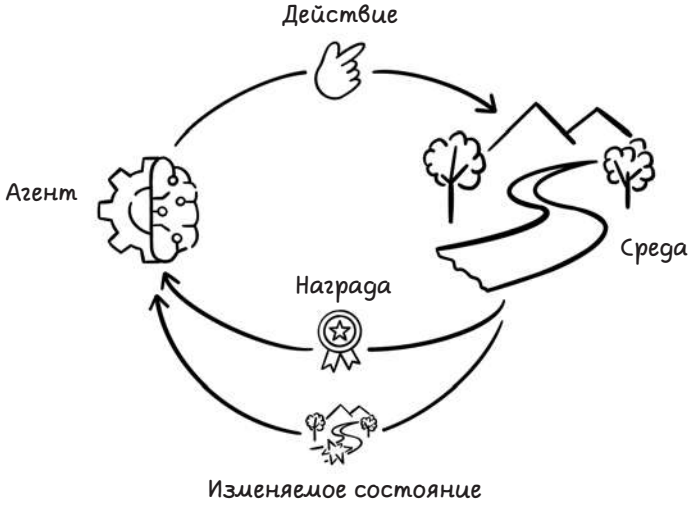


Рис. 10.6. Марковский процесс принятия решения для обучения с подкреплением

Прежде чем приступить к обучению модели, нужно настроить среду, симулирующую область задачи. В нашем случае эта среда включает беспилотный автомобиль, старающийся проехать по парковке к своему владельцу без столкновения с препятствиями. Данную задачу нужно смоделировать как симуляцию, чтобы действия в среде можно было оценить с позиции эффективности достижения цели. Эта симулированная среда отличается от модели, которая будет обучаться предпринимать необходимые действия.

Симуляция и данные: оживление среды

На рис. 10.7 изображен сценарий парковки с множеством машин и пешеходов. Начальные позиции беспилотного автомобиля (агента) и его владельца представлены черными фигурами.

Агент может совершать в среде несколько действий, а именно ехать на север, юг, восток и запад. Выбор действия приводит к его перемещению в соответствующем направлении. По диагонали двигаться запрещено.

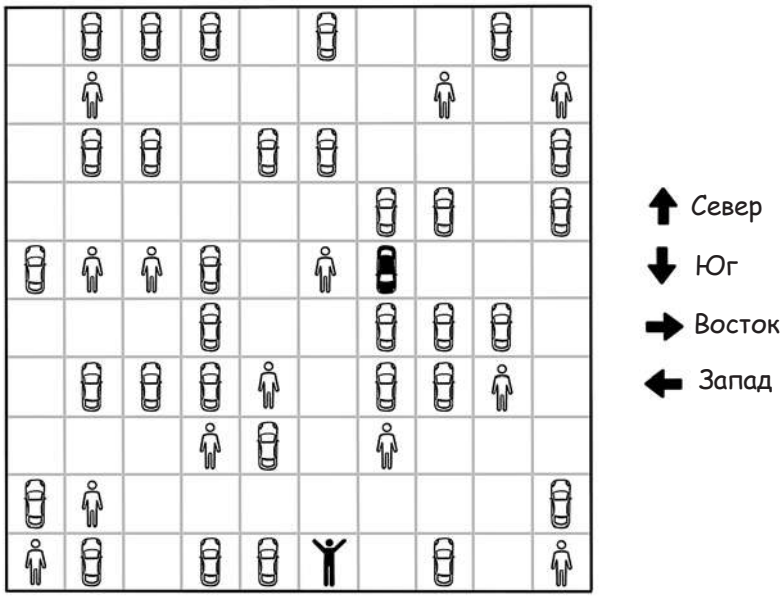


Рис. 10.7. Действия агента в среде парковки

За совершение в среде определенных действий назначаются награды или штрафы. На рис. 10.8 показаны размеры наград и штрафов, присваиваемых агенту на основе изменений, которые его действия вызвали в среде. Столкновение с другим автомобилем — это плохо; столкновение с пешеходом — ужасно. Перемещение в свободную зону — это хорошо; нахождение владельца — еще лучше. Задача штрафов состоит в лишении агента стимула к столкновению с другими

машинами и пешеходами, а задача наград — в поощрении передвижения по направлению к владельцу. Обратите внимание, что может присутствовать награда/штраф за перемещение за границы территории, но в целях упрощения мы эту возможность исключим.

Рис. 10.8. Награды по результатам событий в среде, вызванных действиями агента

> ***Примечание***
>
> Интересно отметить, что в результате такой схемы агент с целью сбора наград может начать бесконечно перемещаться между двумя свободными зонами. В данном примере мы эту вероятность исключим, но она говорит о важности применения правильной схемы поощрений.

Симулятор должен моделировать среду, действия агента и награды, получаемые за каждое действие. Алгоритм обучения с подкреплением будет использовать симулятор, чтобы на практике обучаться совершению действий и оценке их результатов. Как минимум симулятор должен включать следующую функциональность и информацию:

- *Инициализация среды*. Эта функция включает сброс среды, в том числе агента, к начальному состоянию.
- *Получение текущего состояния среды*. Эта функция должна сообщать текущее состояние среды, которое после каждого действия будет изменяться.
- *Применение действия к среде*. Эта функция отвечает за выполнение агентом действий, влияющих на среду и приводящих к получению награды или штрафа.

- *Вычисление награды за действие*. Эта функция описывает применение действия к среде. В ней будут вычисляться оказанный на среду эффект и соответствующая награда.
- *Определение, была ли достигнута цель*. Эта функция определяет, достиг ли агент цели. Цель также иногда может выражаться как `is complete`. В среде, где цели достигнуть невозможно, симулятор должен сигнализировать о завершении, когда сочтет это необходимым.

На рис. 10.9 и 10.10 показаны возможные пути на парковке. На рис. 10.9 агент едет на юг, пока не достигает границы участка; далее он едет на восток вплоть до цели. Несмотря на то что цель достигнута, в сценарии произошло пять столкновений с машинами и одно с пешеходом — далеко не идеальный результат. На рис. 10.10 показан другой сценарий, где агент едет по более специфичному пути к цели, полностью избегая столкновений, — отличный результат. Важно заметить, что с учетом установленных в задаче наград агент не обязательно будет находить кратчайший путь. Поскольку мы более существенно стимулируем избежание препятствий, он может находить любой маршрут, где они отсутствуют.

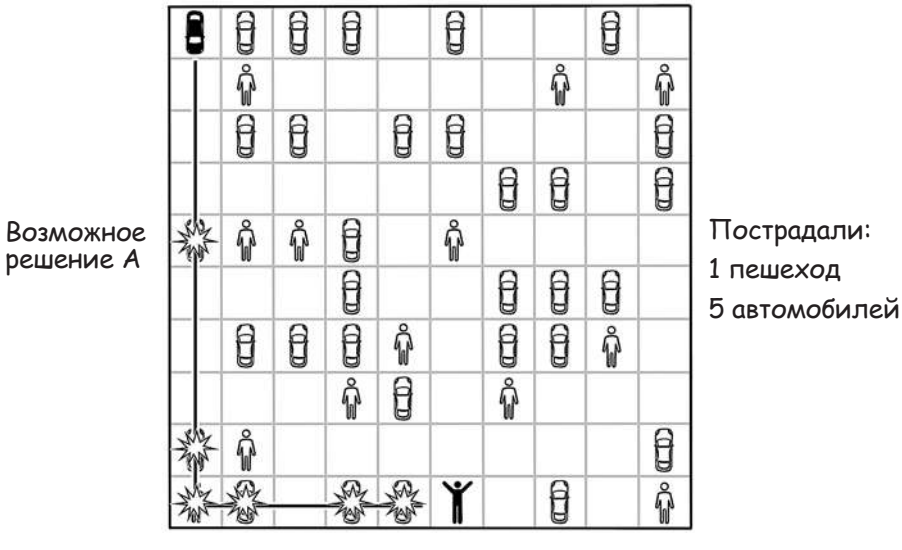


Рис. 10.9. Плохое решение задачи с парковкой

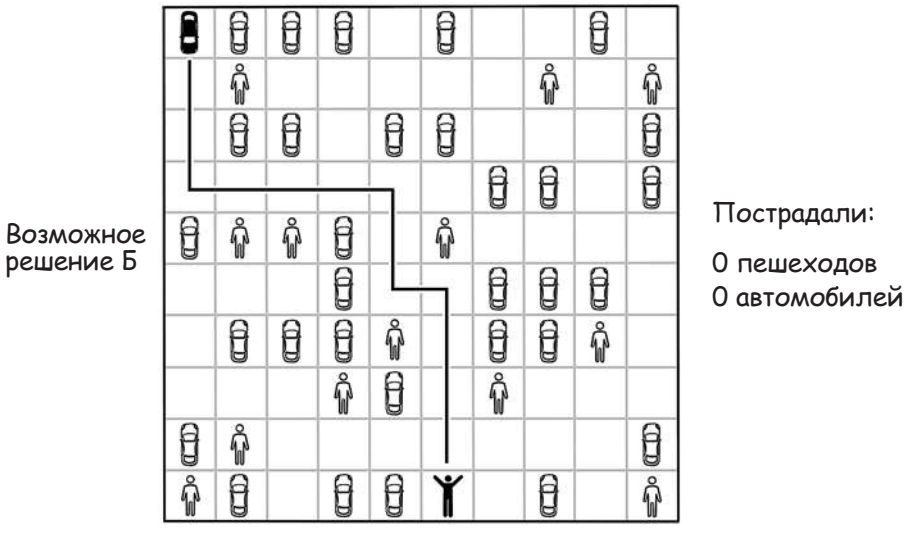


Рис. 10.10. Хорошее решение задачи с парковкой

Пока что симулятор не предполагает автоматического генерирования действий. Сейчас он подобен игре, в которой ход совершает игрок, а не ИИ. Далее речь пойдет об обучении автономного агента.

Псевдокод

Псевдокод для этого симулятора включает описанные выше функции. Класс `Simulator` будет инициализироваться на основе информации, описывающей начальное состояние среды.

Функция `move_agent` отвечает за перемещение агента на север, юг, восток или запад. Она определяет, происходит ли движение в рамках границ, меняет координаты агента, проверяет, было ли столкновение, и возвращает значение награды исходя из результата:

```
Simulator(road, road_size_x, road_size_y,
          agent_start_x, agent_start_y, goal_x, goal_y):

  move_agent(action):
    if action equals COMMAND_NORTH:
      let next_x equal agent_x - 1
      let next_y equal agent_y
    else if action equals COMMAND_SOUTH:
      let next_x equal agent_x + 1
      let next_y equal agent_y
    else if action equals COMMAND_EAST:
      let next_x equal agent_x
      let next_y equal agent_y + 1
    else if action equals COMMAND_WEST:
      let next_x equal agent_x
      let next_y equal agent_y - 1
    if is_within_bounds(next_x, next_y) equals True:
      let reward_update equal cost_movement(next_x, next_y)
      let agent_x equal next_x
      let agent_y equal next_y
    else:
      let reward_update equal ROAD_OUT_OF_BOUNDS_REWARD
    return reward_update
```

Вот описания функций псевдокода:

- `cost_movement` определяет объект в целевой точке координат, куда перемещается агент, и возвращает соответствующее значение награды.
- `is_within_bounds` — вспомогательная функция, проверяющая, находятся ли целевые координаты в рамках границ участка.
- `is_goal_achieved` определяет, была ли достигнута цель. Если да — симуляция завершается.
- `get_state` использует местоположение агента для определения числа, соответствующего текущему состоянию среды. Каждое состояние должно быть уникально. В других пространствах задач оно может быть представлено самим фактическим состоянием.

```
cost_movement(next_x, next_y):
  if road[next_x][next_y] equals ROAD_OBSTACLE_PERSON:
    return ROAD_OBSTACLE_PERSON_REWARD
  else if road[next_x][next_y] equals ROAD_OBSTACLE_CAR:
    return ROAD_OBSTACLE_CAR_REWARD
  else if road[next_x][next_y] equals ROAD_GOAL:
    return ROAD_GOAL_REWARD
  else:
    return ROAD_EMPTY_REWARD

is_within_bounds(next_x, next_y):
  if road_size_x > next_x >= 0 and road_size_y > next_y >= 0:
    return True
  return False

is_goal_achieved():
  if agent_x equals goal_x and agent_y equals goal_y:
    return True
  return False

get_state():
  return (road_size_x * agent_x) + agent_y
```

Обучение на симуляции с помощью Q-learning

Q-обучение (Q-learning) — это подход обучения с подкреплением, который использует состояния и действия в среде для моделирования таблицы, содержащей информацию о предпочтительных действиях на основе конкретных состояний. Представьте Q-обучение как словарь, в котором ключ — это состояние среды, а значение — это наилучшее действие для этого состояния.

Q-обучение предполагает использование таблицы наград, называемой *Q-таблицей*. Эта таблица состоит из столбцов, содержащих все возможные действия, и строк, содержащих возможные состояния среды. Задача Q-таблицы — описать, какие действия являются наиболее предпочтительными для агента на пути к цели. Значения, представляющие эти предпочтительные действия, должны быть получены путем анализа симуляций возможных вариантов действий и происходящих в среде изменений. Стоит заметить, что для агента существует шанс выбрать случайное действие или действие из Q-таблицы, как уже показывалось на рис. 10.13. *Q* выражает функцию, которая предоставляет награду, иначе говоря, качество действия в среде.

На рис. 10.11 показана обученная Q-таблица и два возможных состояния, которые можно представить через значения действий для каждого из них. Эти состояния соответствуют решаемой задаче; другой ее вариант мог бы, к примеру, допускать движение агента по диагонали. Обратите внимание, что количество состояний отличается в зависимости от среды и что новые состояния можно добавлять по ходу их обнаружения. В состоянии 1 агент находится в левом верхнем углу, а в состоянии 2 — в позиции ниже его предыдущего состояния. Q-таблица кодирует лучшие возможные действия с учетом каждого соответствующего состояния. Действие с наибольшим числом является наиболее предпочтительным. Значения Q-таблицы на этом рисунке были найдены ранее путем обучения. Сам процесс их вычисления рассмотрим чуть позже.

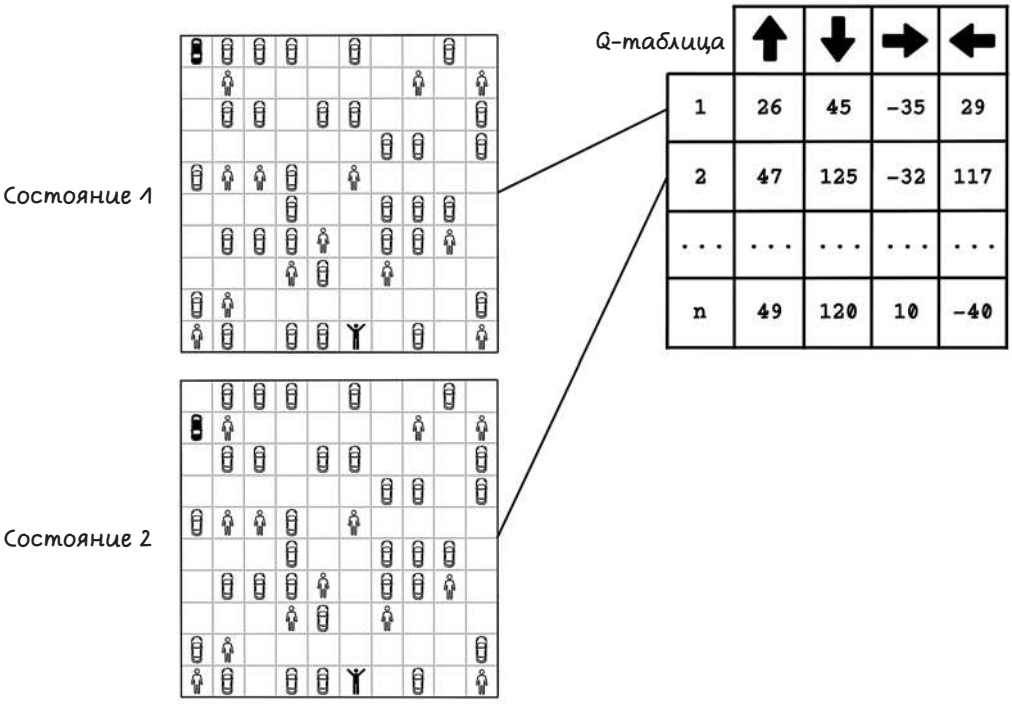


Рис. 10.11. Пример Q-таблицы, содержащей состояния

Проблема представления состояния с помощью всей карты заключается в том, что расположение других машин и людей специфично для конкретной задачи, в связи с чем Q-таблица изучает лучшие выборы только для данной карты.

Более удачный способ выразить состояние в этом примере — рассматривать объекты, находящиеся в непосредственной близости от агента. Это позволит Q-таблице адаптироваться к другим конфигурациям парковки, потому что состояние будет уже не столь специфичным. Кажется довольно просто, но в соседнем блоке может находиться другая машина или пешеход, либо же блок может оказаться пустым или выходить за границы поля. В общей сложности получается четыре возможных состояния для каждого блока и 65 536 допустимых состояний в общем. При таком большом разнообразии потребуется обучать агента во множестве конфигураций парковки множество раз, чтобы он запомнил, какие краткосрочные действия выбирать (рис. 10.12).

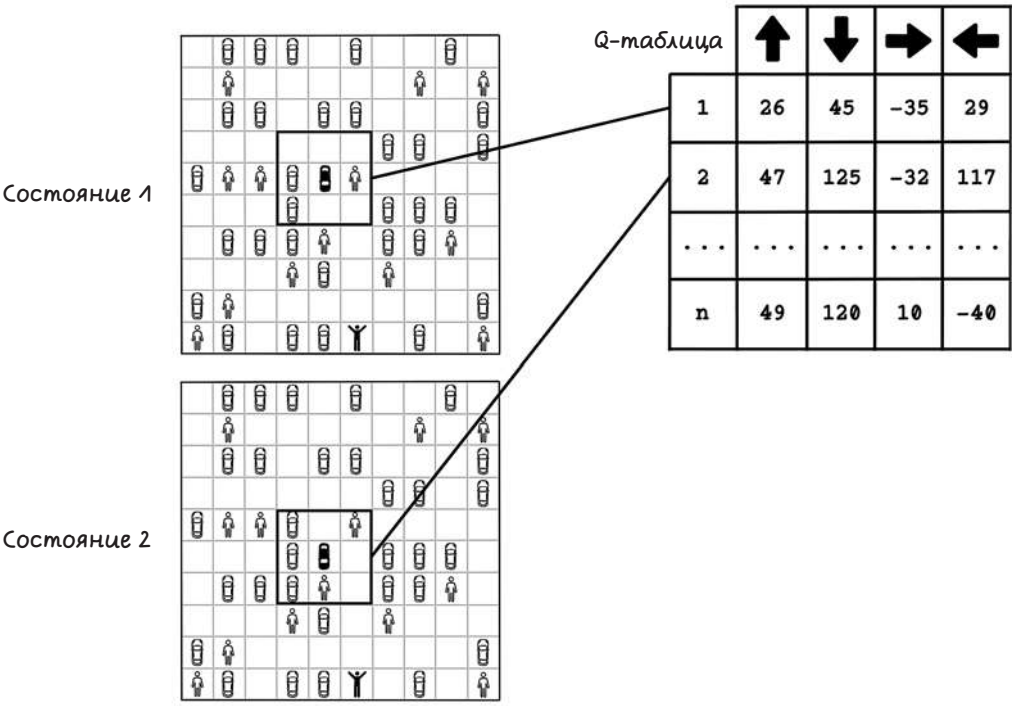


Рис. 10.12. Пример оптимальной Q-таблицы, содержащей состояния

К таблице наград мы еще будем возвращаться по мере знакомства с процессом обучения модели с помощью Q-обучения. Она будет представлять модель для действий, совершаемых агентом в среде.

Далее рассмотрим жизненный цикл алгоритма Q-обучения. Речь пойдет о двух фазах: инициализации и процессов, происходящих в течение итераций обучения алгоритма (рис. 10.13).

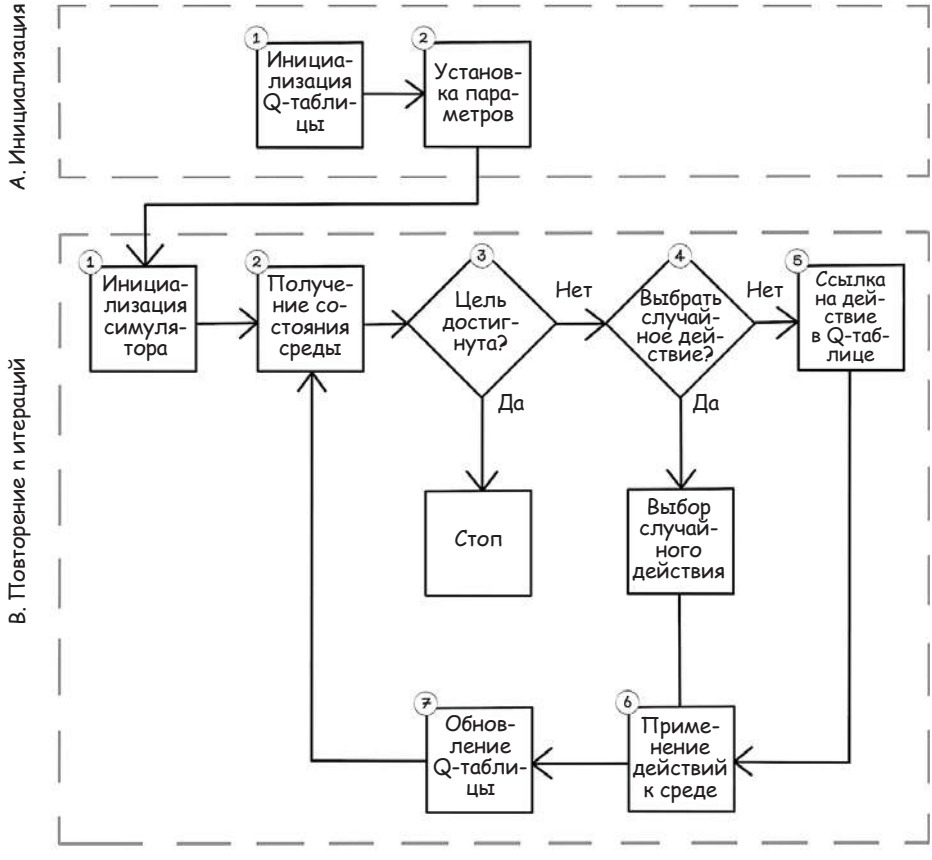


Рис. 10.13. Жизненный цикл алгоритма обучения с подкреплением с использованием Q-обучения

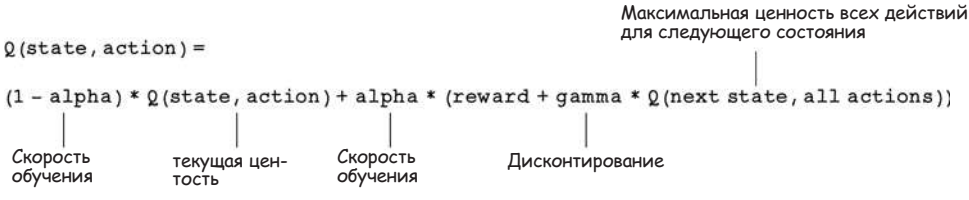
- *Инициализация*. Шаг инициализации включает в себя настройку соответствующих параметров и начальных значений для Q-таблицы:
  1. *Инициализация Q-таблицы*. Инициализация Q-таблицы, в которой каждый столбец — это действие, а каждая строка — возможное состояние. Обратите внимание, что состояния могут добавляться в таблицу по мере возникновения, так как изначально знать их все очень сложно. Начальные значения действий для каждого состояния инициализируются с 0.
  2. *Установка параметров*. Этот шаг включает в себя настройку различных гиперпараметров алгоритма Q-обучения, в том числе:
     - *Шанс выбора случайного действия*. Это пороговое значение, определяющее выбор случайного действия вместо действия из Q-таблицы.

- *Скорость обучения*. Здесь этот параметр работает так же, как и при обучении с учителем, определяя, насколько быстро алгоритм обучается на наградах в различных состояниях. Высокая скорость ведет к тому, что значения в Q-таблице изменяются хаотично, медленная же обусловливает их постепенное изменение, но потенциально требует большего числа итераций для нахождения оптимальных значений.
- *Коэффициент дисконтирования*. Описывает, насколько ценна потенциальная будущая награда, что позволяет выбрать мгновенное вознаграждение либо долгосрочную награду. При низком коэффициенте предпочтение отдается мгновенному вознаграждению. Более же высокий коэффициент склоняет к выбору долгосрочных наград.

- *Повторение n итераций*. Для нахождения лучших действий в одних и тех же состояниях выполняются описываемые ниже шаги, подразумевающие многократное вычисление этих состояний. В течение всех итераций будет обновляться одна и та же таблица. Поскольку для агента важна последовательность действий, награда за действие в любом состоянии может меняться в зависимости от предыдущих действий. По этой причине необходимо производить несколько итераций. Каждую из них можно рассматривать как попытку достижения цели:
  1. *Инициализация симулятора*. Этот шаг включает в себя сброс среды к начальному состоянию, когда агент в нейтральном состоянии.
  2. *Получение состояния среды*. Эта функция должна предоставлять текущее состояние среды, которое будет изменяться после выполнения каждого действия.
  3. *Цель достигнута?* Определение, была ли достигнута цель (либо симулятор считает поиск завершенным). В нашем примере цель — достижение автомобилем владельца. Если она достигнута, алгоритм завершается.
  4. *Выбор случайного действия*. Определение, нужно ли выбрать случайное действие. Если да, то автомобиль случайным образом определяет, в каком направлении перемещаться: на север, юг, восток или запад. Случайные действия полезны для исследования возможностей в среде вместо изучения их узкого подмножества.
  5. *Ссылка на действие в Q-таблице*. Если решение о выборе случайного действия принято не было, в Q-таблицу переносится текущее состояние среды и на основе ее значений выбирается соответствующее действие. Рассмотрим Q-таблицу подробнее чуть позже.
  6. *Применение действий к среде*. Этот шаг включает применение действия к среде, будь то случайное действие или выбранное из Q-таблицы. Это действие будет иметь последствия в среде и определит соответствующую награду.

7. *Обновление Q-таблицы*. Ниже описывается обновление Q-таблицы и его этапы.

Ключевой аспект Q-обучения — это уравнение, используемое для обновления значений Q-таблицы. Это уравнение основано на *уравнении Беллмана*, которое определяет ценность решения, принятого в конкретный момент времени с учетом полученной за него награды или штрафа. Уравнение Q-обучения представляет собой адаптацию уравнения Беллмана. В нем наиболее важными свойствами для обновления значений Q-таблицы являются текущее состояние, действие, состояние, получаемое после этого действия, и итоговая награда. Скорость обучения работает так же, как при обучении с учителем, и представляет величину, на которую обновляется значение Q-таблицы. Дисконтирование используется для указания значимости возможных будущих наград, что позволяет соблюдать баланс между выбором мгновенного вознаграждения или долгосрочных наград:



Поскольку значения Q-таблицы инициализируются как 0, изначально она выглядит как на рис. 10.14.

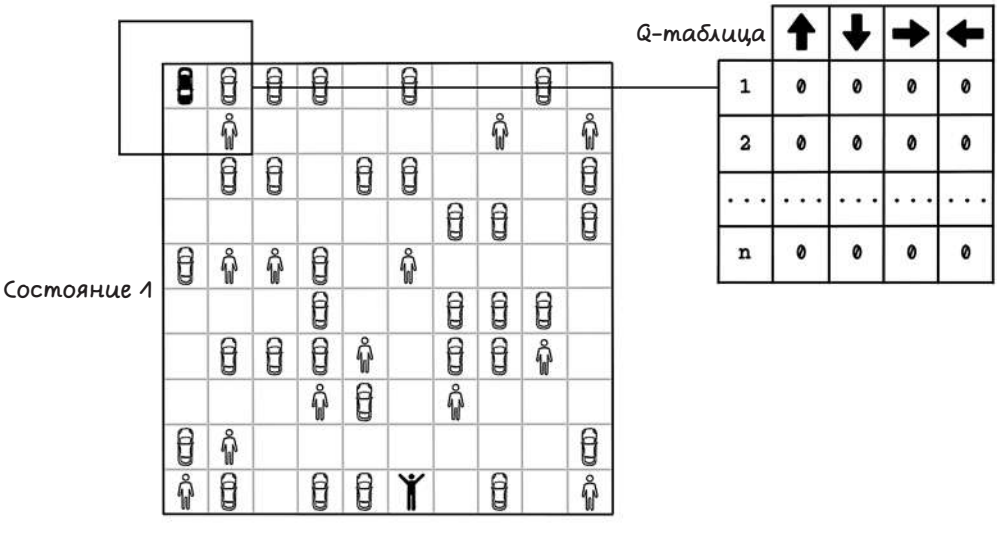


Рис. 10.14. Пример инициализированной Q-таблицы

Далее мы изучим, как обновлять Q-таблицу с помощью уравнения Q-обучения на основе разных действий с разными значениями награды. Эти значения будут использованы для скорости обучения (alpha) и дисконтирования (gamma):

- Скорость обучения (alpha): 0,1
- Дисконтирование (gamma): 0,6

На рис. 10.15 показано, как уравнение Q-обучения используется для обновления Q-таблицы, если агент выбирает из начального состояния в первой итерации действие «восток». Помним, что начальная Q-таблица содержит нули. Скорость обучения (alpha), дисконтирование (gamma), значение текущего действия, награда и следующее лучшее состояние подставляются в это уравнение, чтобы определить новое значение для совершенного действия. В данном случае действием выступает движение на восток, что приводит к столкновению с другой машиной и штрафу –100. После вычисления нового значения значение для движения на восток в состоянии 1 равно –10.

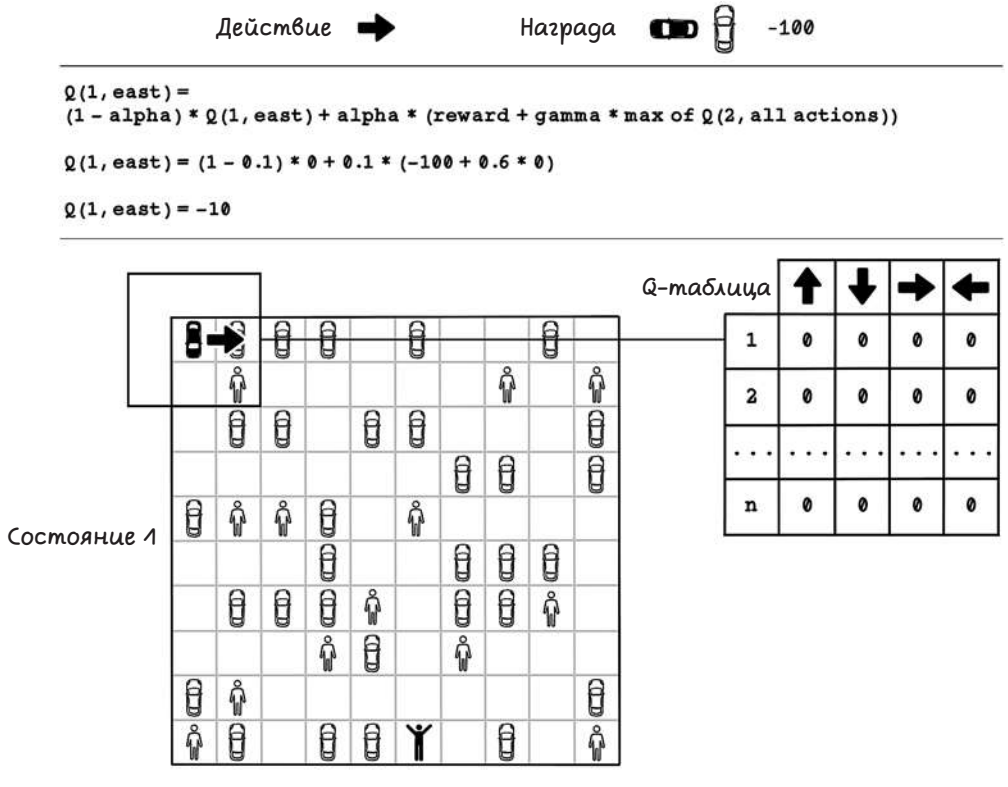


Рис. 10.15. Пример вычисления обновления Q-таблицы для состояния 1

Далее вычисляется следующее состояние в среде, возникающее после очередного действия — движения на юг, которое приводит к столкновению с пешеходом и штрафу –1000. После вычисления нового значения значение для движения на юг в состоянии 2 будет равно –100 (рис. 10.16).

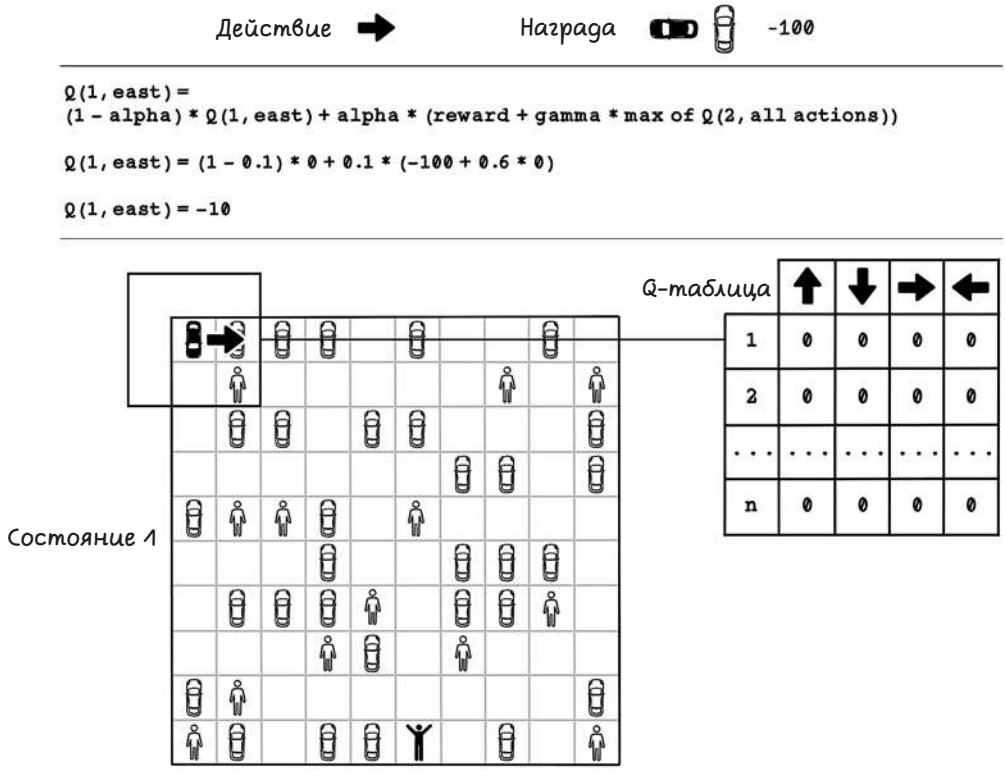


Рис. 10.16. Пример вычисления обновления Q-таблицы для состояния 2

На рис. 10.17 показано, как вычисленные значения Q-таблицы отличаются от начальных, так как мы работали с таблицей, инициализированной с 0. На рисунке представлен вид уравнения Q-обучения после нескольких итераций. Данную симуляцию можно выполнить несколько раз, чтобы обучить алгоритм на нескольких попытках, так что эта итерация завершает собой ряд других, в которых значения таблицы обновлялись. Действие движения на восток приводит к столкновению с другой машиной и штрафу –100. После вычисления нового значения значение для движения на восток в состоянии 1 равно –34.

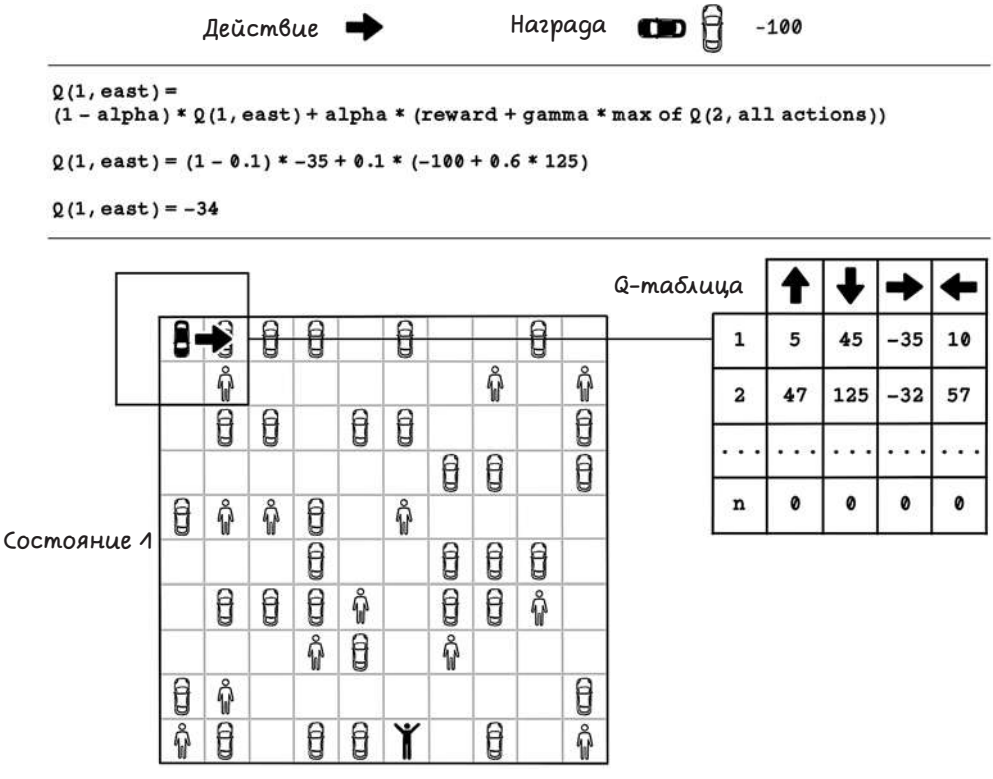
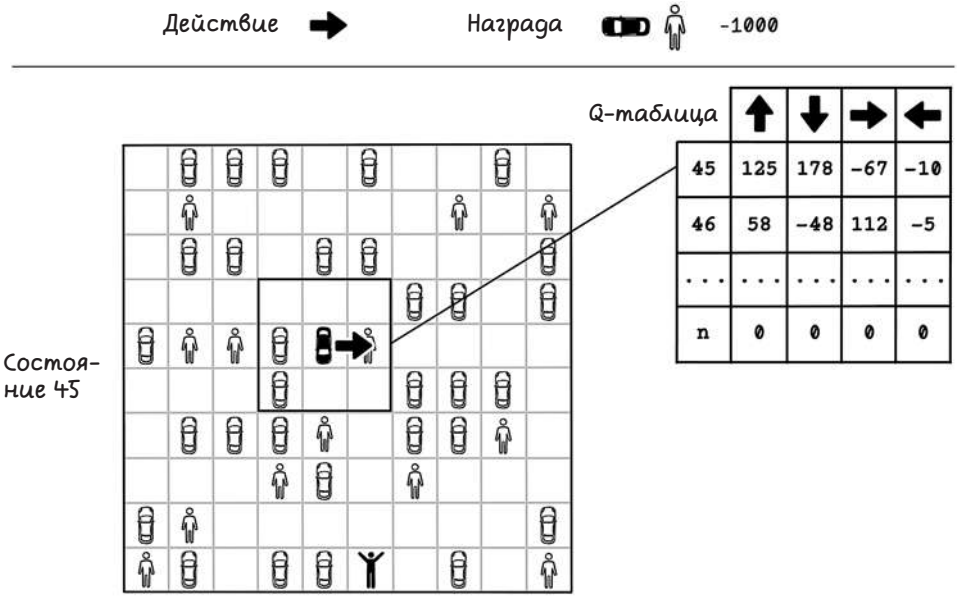


Рис. 10.17. Пример вычисления обновления Q-таблицы для состояния 1 после нескольких итераций

УПРАЖНЕНИЕ: ВЫЧИСЛИТЕ ИЗМЕНЕНИЕ ЗНАЧЕНИЙ ДЛЯ Q-ТАБЛИЦЫ

Вычислите новое значение для выполненного действия в следующем сценарии, используя уравнение обновления Q-обучения. Предположим, что последним действием было движение на восток со значением –67:



ОТВЕТ: НОВЫЕ ЗНАЧЕНИЯ ДЛЯ Q-ТАБЛИЦЫ

Подставим значения гиперпараметров и состояний в уравнение Q-обучения и получим новое значение для Q(1, восток):

- Скорость обучения (alpha): 0.1
- Дисконтирование (gamma): 0.6
- Q (1, восток): –67
- Max of Q (2, все действия): 112

```
Q(1, east) =
(1 - alpha) * Q(1, east) + alpha * (reward + gamma * max of Q(2, all actions))

Q(1, east) = (1 - 0.1) * -67 + 0.1 * (-100 + 0.6 * 112)

Q(1, east) = -64
```

Псевдокод

Этот псевдокод описывает функцию, обучающую Q-таблицу с помощью Q-обучения. Ее можно разбить на более мелкие функции, но мы привели ее в таком виде для наглядности и чтобы отразить все шаги, описанные по ходу главы.

Q-таблица инициализируется с 0. Далее выполняется логика обучения в течение нескольких итераций. Напомню, что итерация — это попытка достижения цели. Пока цель не достигнута, выполняются следующие действия:

1. Решить, выбирается ли случайное действие для изучения возможностей среды. Если нет, то из Q-таблицы берется действие с наибольшим значением для текущего состояния.
2. Применить выбранное действие к симулятору.
3. Собрать информацию из симулятора, включая награду, состояние после выполнения действия и определение, была ли достигнута цель.
4. Обновить Q-таблицу на основе собранной информации и гиперпараметров. Обратите внимание, что в этом коде гиперпараметры передаются в качестве аргументов функции.
5. Установить текущее состояние как результирующее после только что выполненного действия.

Эти шаги будут повторяться до достижения цели. Конечным результатом после всех итераций будет обученная Q-таблица, которую можно использовать для тестирования в других средах. Сам процесс тестирования будет рассмотрен в следующем разделе.

```
train_with_q_learning(observation_space, action_space,
                      number_of_iterations, learning_rate,
                      discount, chance_of_random_move):
  let q_table equal a matrix of zeros [observation_space, action_space]
  for i in range(number_of_iterations):
    let simulator equal Simulator(DEFAULT_R0AD, DEFAULT_R0AD_SIZE_X,
                                  DEFAULT_R0AD_SIZE_Y, DEFAULT_START_X,
                                  DEFAULT_START_Y, DEFAULT_G0AL_X,
                                  DEFAULT_G0AL_Y)
    let state equal simulator.get_state()
    let done equal False
    while not done:
      if random.uniform(0, 1) > chance_of_random_move:
        let action equal get_random_move()
      else:
        let action max(q_table[state])

      let reward equal simulator.move_agent(action)
      let next_state equal simulator.get_state()
      let done equal simulator.is_goal_achieved()

      let current_value equal q_table[state, action]
      let next_state_max_value equal max(q_table[next_state])
```

```
        let new_value equal (1 - learning_rate) * current_value + learning_rate *
                            (reward + discount * next_state_max_value)

        let q_table[state, action] equal new_value
        let state equal next_state

    return q_table
```

Тестирование симуляции и Q-таблицы

В случае применения Q-обучения Q-таблица является моделью, заключающей в себе изученные данные. При работе с новой средой, где отражены новые состояния, алгоритм ссылается на соответствующее состояние из Q-таблицы и выбирает действие с максимальным значением. Так как Q-таблица уже обучена, данный процесс состоит из повторяющегося получения текущего состояния среды и обращения к соответствующему состоянию в Q-таблице для выбора действия, пока не будет достигнута цель (рис. 10.18).

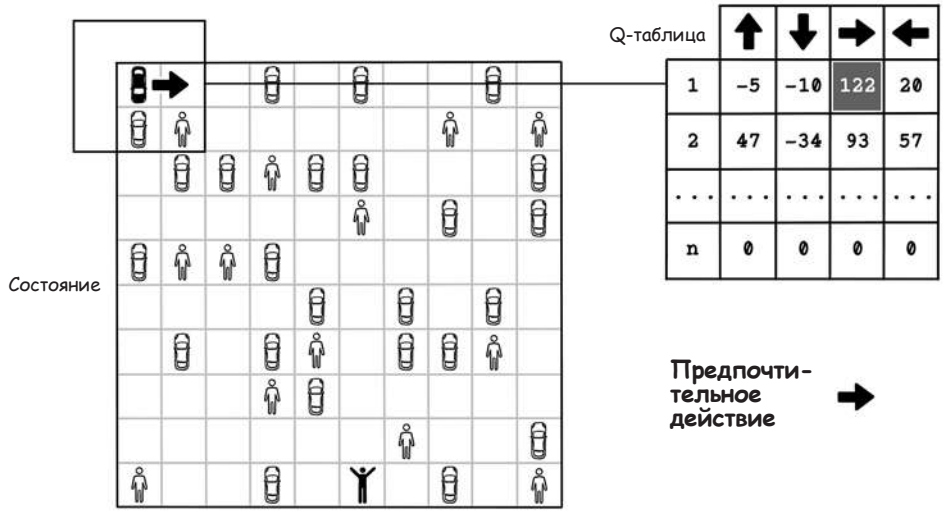


Рис. 10.18. Обращение к Q-таблице для выбора действия

Так как состояние в Q-таблице учитывает, находятся ли непосредственно по соседству с агентом другие объекты, таблица обучается хорошим и плохим действиям для краткосрочных наград. Поэтому ее можно использовать в другой конфигурации парковки, например той, что показана на рис. 10.18. Недостаток в том, что агент предпочитает краткосрочные награды долгосрочным, потому

что при совершении очередного действия не знает о содержимом остальной части карты.

В процессе освоения обучения с подкреплением вам наверняка также встретится термин *эпизоды*. Эпизод включает все состояния между начальным и тем, в котором была достигнута цель. Если же цель не достигается, эпизод называется *бесконечным*.

Оценка эффективности обучения

Алгоритмы обучения с подкреплением сложно оценивать в общем. При рассмотрении конкретной среды и цели могут присутствовать различные штрафы и награды, и какие-то из них будут больше влиять на контекст задачи, чем другие. В примере с парковкой мы добавляем крупный штраф за наезд на пешехода. Но в другой ситуации агентом может быть человек, который стремится определить, какие мышцы задействовать, чтобы пройти пешком как можно большее расстояние. В этом сценарии штрафом может стать падение или что-то более конкретное, например слишком широкий шаг. Для точной оценки эффективности обучения необходим контекст задачи.

Один из универсальных способов оценки — это подсчет количества штрафов за определенное количество попыток. Штрафы могут быть за события, которых требуется избегать и которые происходят в среде как результат действия.

Еще один способ измерения эффективности обучения с подкреплением — это *средняя награда за действие*. Назначая максимальную награду за совершаемое действие, мы стремимся избежать плохих вариантов независимо от того, ведут они к достижению цели или нет. Этот показатель можно вычислить, разделив суммарную награду на общее количество действий.

Обучение с моделью и без

Для понимания обучения с подкреплением вам следует знать два его подхода: *с использованием модели* и *без использования модели*. Речь идет не о тех моделях машинного обучения, о которых говорилось ранее в книге. В данном случае модель можно рассматривать как абстрактное представление агентом среды, в которой он оперирует.

К примеру, мы можем иметь представление о модели окружающей местности, включающей расположение основных объектов, понимание направлений и общую схему дорог. Эта модель сформировалась после изучения определенных маршрутов, но мы можем симулировать в своем сознании разные сценарии и не обязательно пробовать каждый возможный вариант. На основе этой модели можно, например, определить, как лучше добраться до работы. Такой подход считается основанным на модели. Обучение же без модели аналогично подходу Q-обучения. В нем изучение множества вариантов взаимодействия со средой происходит методом проб и ошибок, что позволяет определить оптимальные действия в различных сценариях.

Рисунок 10.19 иллюстрирует эти два подхода на примере навигации по дорогам. Для реализации обучения с подкреплением на основе модели можно использовать разные алгоритмы.

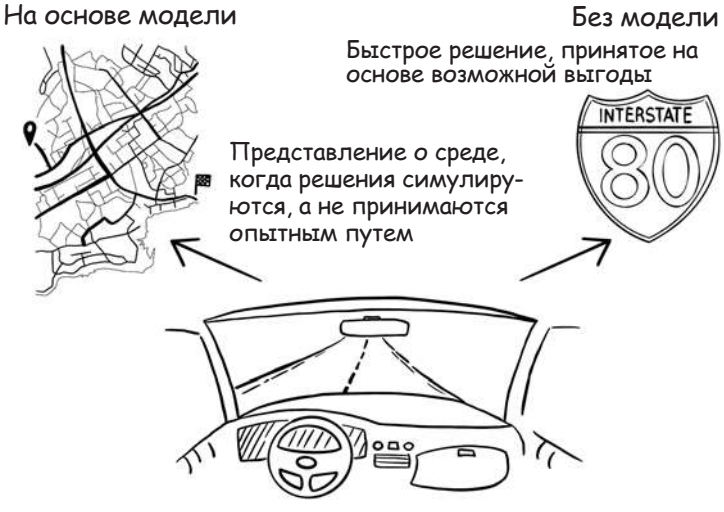


Рис. 10.19. Примеры обучения с подкреплением на основе модели и без нее

Глубокое обучение в обучении с подкреплением

Q-обучение — это один из подходов в обучении с подкреплением. Понимая, как оно работает, можно применять похожую логику и к другим алгоритмам обучения с подкреплением. В зависимости от решаемой задачи можно выбирать подходящую технику. Довольно популярно *глубокое обучение*, эффективно применяемое в роботостроении, компьютерных играх, а также для обработки изображений и видео.

В глубоком обучении с подкреплением для обработки состояний среды и совершения действий могут применяться искусственные нейронные сети (ANN). Действия изучаются путем корректировки весов в ANN на основе получаемой награды и изменений в среде. В обучении с подкреплением для решения различных прикладных задач также могут использоваться возможности сверточных нейронных сетей (CNN) и других специфичных архитектур ANN.

На рис. 10.20 отражено высокоуровневое представление того, как можно использовать ANN для решения задачи с парковкой. Входами в нейронную сеть служат состояния. Выходами — вероятности выбора лучшего действия для агента. Награда же и оказываемый на среду эффект могут с помощью обратного распространения возвращаться в сеть для подстройки весов.

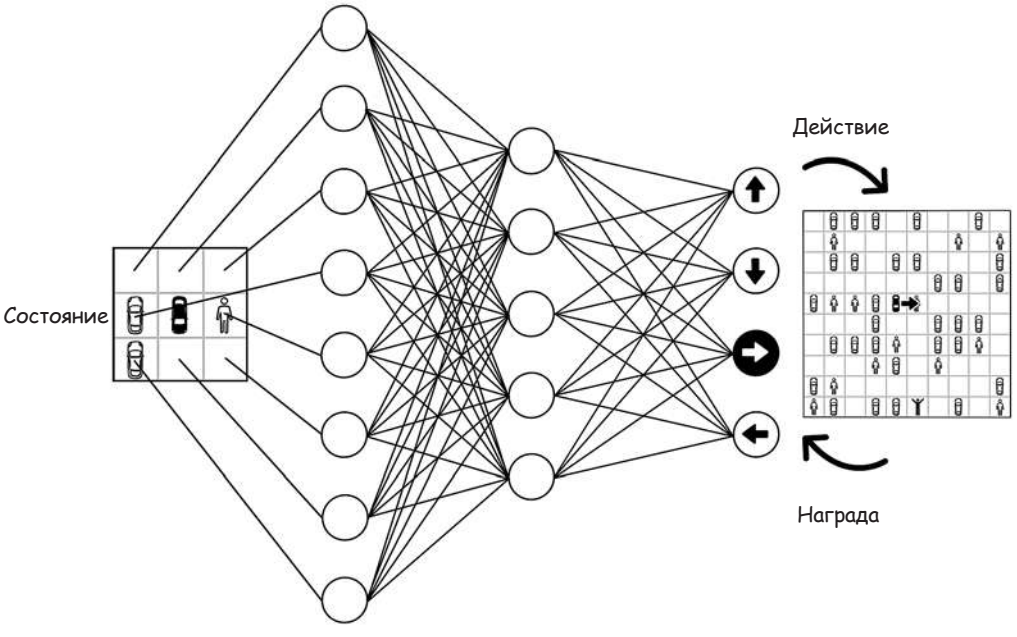


Рис. 10.20. Пример использования ANN для задачи с парковкой

В следующем разделе рассматриваются некоторые варианты реального применения обучения с подкреплением.

Применение обучения с подкреплением

Обучение с подкреплением применяется во многих ситуациях, когда данные для обучения отсутствуют либо их недостаточно. Обучение происходит через взаимодействие со средой, в которой присутствует эвристика для оценки высокой эффективности. Эту технику можно применять практически во всех сферах. Разберем некоторые из них.

Роботостроение

Роботостроение подразумевает создание машин, которые взаимодействуют с реальными средами, выполняя поставленные задачи. Некоторые роботы используются для сложной навигации по пересеченной местности с разнообразными типами покрытия дорог, препятствиями и уклонами. Другие применяются в качестве лабораторных помощников, получая от ученых инструкции, передавая нужные инструменты или управляя оборудованием. Когда смоделировать каждый возможный результат каждого действия в огромной динамической среде невозможно, на выручку приходит обучение с подкреплением. Устанавливая в среде основную задачу и вводя награды и штрафы в качестве эвристик, можно

использовать обучение с подкреплением для обучения роботов в динамических средах. К примеру, робот-навигатор может изучить, каким колесам передавать крутящий момент и как подстраивать подвеску для успешного преодоления сложных участков. Эта цель достигается после множества попыток.

Эти сценарии можно симулировать виртуально, если есть возможность смоделировать на компьютере основные особенности среды. В некоторых проектах обучение беспилотных автомобилей сначала происходило на основе компьютерных игр, после чего машины отправляли доучиваться на реальные дороги. Цель обучения роботов с помощью обучения с подкреплением состоит в создании более общих моделей, которые смогут адаптироваться к новым и разнообразным средам, изучая, подобно людям, обобщенные взаимодействия.

Рекомендательные системы

Рекомендательные системы используются во многих цифровых продуктах. Платформы потокового видео применяют их для изучения пользовательских предпочтений видео и выдачи рекомендаций по интересам. Этот же подход используется в платформах потокового аудио и онлайн-магазинах. Модели обучаются на поведении зрителя, когда он принимает решение о просмотре рекомендованных видео. Суть в том, что если рекомендованное видео было выбрано и полностью просмотрено, то модель получает высокую награду, так как она предположила, что это видео окажется хорошей рекомендацией. И наоборот, если пользователь не выберет видео или просмотрит лишь небольшой фрагмент, то разумно предположить, что оно ему не понравилось. В результате модель получит либо малую награду, либо штраф.

Биржевая торговля

К финансовым инструментам трейдинга относятся биржевые компании, криптовалюты и другие пакетные инвестиционные продукты. Задача трейдинга весьма сложная. Аналитики отслеживают закономерности в изменениях цен и мировые новости, после чего на основе анализа дают рекомендации придержать инвестиции, продать их часть либо купить еще. С помощью обучения с подкреплением можно научить модели принимать эти решения, награждая их за получение дохода и штрафуя за утрату средств. Разработка такой модели для успешных торгов требует большого количества проб и ошибок, что подразумевает в ходе обучения потерю агентом значительных сумм денег. К счастью, большинство исторических публичных финансовых данных доступно бесплатно, а некоторые платформы инвестирования даже предлагают песочницы для экспериментирования.

Несмотря на то что модель обучения с подкреплением может генерировать хорошую отдачу от инвестиций, подумайте: как будет выглядеть рынок, если все инвесторы станут абсолютно рациональны, перейдя на автоматизированные процессы и избавившись от человеческого фактора?

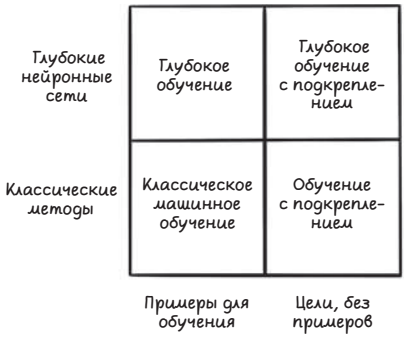
Видеоигры

Популярные игры-стратегии многие годы развивают интеллектуальные способности игроков. Эти игры, как правило, включают управление множеством видов ресурсов при одновременном планировании краткосрочных и долгосрочных тактических действий для победы над противником. Очень часто по ним проводятся чемпионаты, на которых малейшие ошибки оборачиваются для лучших участников сокрушительными поражениями. Обучение с подкреплением применялось в таких играх на профессиональном уровне. В их реализациях, как правило, участвует агент, который так же, как человек, наблюдает за экраном, изучает закономерности и совершает действия. Награды и штрафы ассоциируются непосредственно с самой игрой. Спустя множество итераций игры в различных сценариях с разными оппонентами агент вырабатывает наиболее эффективные тактики для достижения долгосрочной цели в виде окончательной победы. Цель исследования в этой области связана с поиском более общих моделей, которые смогут собирать контекст из абстрактных состояний и сред, понимая схемы, которые невозможно отобразить в логическом представлении. К примеру, в детстве нам не нужно было обжигаться о множество предметов, чтобы понять, что горячее опасно. Мы постепенно вырабатывали интуицию и проверяли ее по мере взросления. Эти проверки укрепляли наше представление о горячих объектах и их потенциальном вреде или пользе.

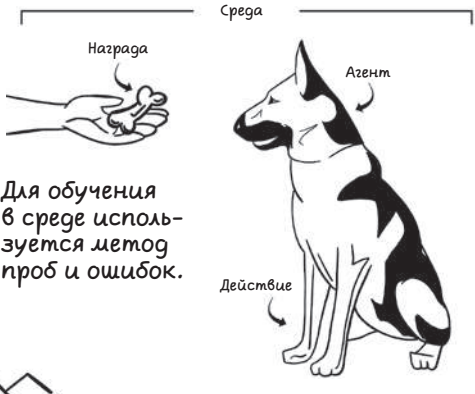
Ну и наконец, исследуя и разрабатывая ИИ, человек стремится сделать компьютеры способными обучаться решать задачи теми же способами, в которых преуспел он сам: объединять абстрактные идеи и принципы для нахождения успешных решений.

Краткий обзор главы «Обучение с подкреплением»

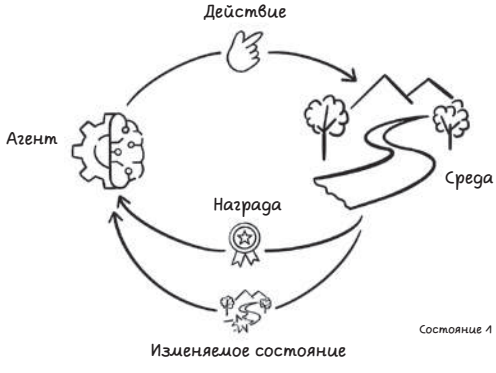
Обучение с подкреплением используется, когда известна цель, но нет примеров для изучения.



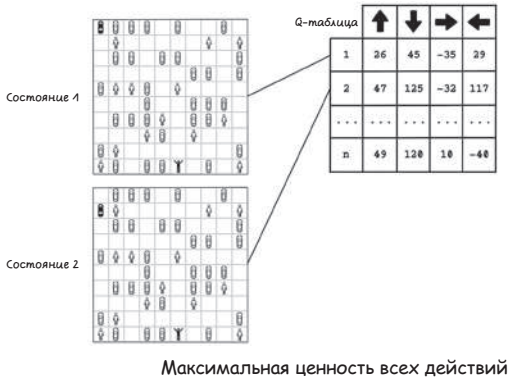
В обучении с подкреплением используются как классические методы, так и методы глубоких нейросетей.



Для обучения в среде используется метод проб и ошибок.



Q-таблица включает столбцы-действия и строки-состояния.



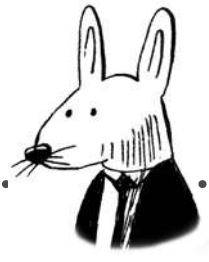
Q-обучение использует Q-таблицу и функцию обучения, чтобы обучаться на совершенных действиях.

Максимальная ценность всех действий для следующего состояния

```
Q(state, action) =
(1 - alpha) * Q(state, action) + alpha * (reward + gamma * Q(next state, all actions))
```

Скорость обучения — Текущая ценность — Скорость обучения — Дисконтирование

Применение алгоритмов **искусственного интеллекта**



Поиск в глубину

Когда известно, что решения в дереве находятся глубоко в пространстве поиска и можно произвести вычисление каждой ветки без особых затрат.

Поиск в ширину

Когда известно, что решения в дереве находятся неглубоко в пространстве поиска и можно произвести вычисление каждой ветки без особых затрат.

Поиск A*

Для поиска решения в дереве, когда для направления поиска и оптимизации вычислений можно создать эвристику.

Минимаксный поиск

В решении состязательных задач, где в поиске оптимальных решений соревнуются два агента.

Генетический алгоритм

Когда вероятные решения можно закодировать в виде хромосом, а для точной их оценки создать функцию приспособленности.

Муравьиный алгоритм

В задачах, где решение состоит из последовательности действий или выборов и допускается «приемлемое» решение.

Оптимизация роем частиц

При поиске в многомерных и обширных пространствах решений, когда не требуется абсолютно лучший вариант.

Линейная регрессия

Для создания прогнозов на основе корреляций двух и более признаков датасета.

Дерево решений

Для категоризации образцов данных из датасета на основе их признаков, когда эти признаки непосредственно связаны с категориями образцов.

Искусственная нейронная сеть

Для формирования прогнозов на основе набора неструктурированных данных, внутренние корреляции которых неочевидны.

Q-обучение

При решении задач, где агент действует в среде и обучение должно происходить опытным путем, а не на исторических данных.

Ришал Харбанс

Грокаем алгоритмы искусственного интеллекта

Перевел с английского Д. Брайт

Руководитель дивизиона	*Ю. Сергиенко*
Ведущий редактор	*Е. Строганова*
Научный редактор	*В. Кадочников*
Литературный редактор	*М. Трусковская*
Корректоры	*С. Беляева, М. Лауконен*
Верстка	*Л. Соловьева*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 01.2023.

Наименование: книжная продукция.

Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014,
58.11.12 — Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск,
ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 02.11.22. Формат 70×100/16. Бумага офсетная.
Усл. п. л. 29,670. Тираж 1200. Заказ 0000.