# 10. Оптимизация правил поведения

Значительная часть этого раздела посвящена методам обучения или оптимизации **правил поведения** (***policies***) [1], представляющих очень небольшие программы, включающие правила, которые описывают что делать агенту в каждой из возможных ситуаций. Чтобы выучить правило поведения, агент должен делать то, что делает обычно и, получая за некоторые действия **вознаграждение** (***reward***) (или **подкрепление**, ***reinforcement***), он будет поощрен или наказан за свои действия. Вознаграждение распространяется на предыдущие действия агента и со временем научит агента, какие действия приводят к положительным вознаграждениям и помогают избежать отрицательных.

В сообществе, занимающимся машинным обучением, неметаэвристические методы для обучения правилам поведения основываются на направлении исследований, называемом **обучение с подкреплением** (***reinforcement learning***). Такие методы пытаются найти правила для каждого возможного состояния окружающего мира. В отличие от них, эволюционные методы, применяемые для решения этой задачи, такие как **Мичиганский подход к системам обучающихся классификаторов** (***Michigan-Approach Learning Classifier Systems, LCS***) или **Система правил по Питт-подходу** (***Pitt-Approach Rule Systems***), способны находить значительно более компактные и разреженные описания всего пространства возможных состояний. Рассмотрение упомянутых методов мы начнем с обучения с подкреплением, поскольку оно и исторически, и теоретически тесно связанно с эволюционными методами. В частности кратко, на несколько страниц, будет дано описание не метаэвристического метода обучения с подкреплением, который называется ***Q*-обучение** (***Q-Learning***). Затем перейдем к эволюционным методам.

Не буду вас обманывать. Данная тема является непростой для понимания. Поэтому, если что, то я вас предупреждал [2].

## 10.1 Обучение с подкреплением: Плотная оптимизация правил поведения

Мы начнем рассмотрение с методов для обучения плотным правилам поведения (*dense policies*)), известных под общим названием **обучение с подкрплением** (***reinforcement learning***). Это делается отчасти чтобы подготовить почву для метаэвристических подходов, а частично из-за того, что ряд используемых здесь понятий понадобится и в дальнейшем.

*Обучение с подкреплением* — это странный термин. В общем случае он обозначает любой метод обучения или адаптации, основанный на получении оценки качества (вознаграждение или наказание — *подкрепление* (*reinforcement*)). Поэтому любой рассмотренный ранее метод может отнесен к обучению с подкреплением. Но, к несчастью, данный очень общий термин активно использовался в достаточно специализированной области исследований, в которой рассматривается обучение небольших программ, состоящих из правил вида *если* $\rightarrow$*то*. Напомним, что в Разделе 4.5 такие правила назывались **правила состояние-действие** (***state-action rules***). Обучение с подкреплением направлено на поиск оптимального набора состояние-действие для данного окружения на основании только лишь подкрепления, получаемого в результате испытаний различных наборов правил.

---

[0] Перевод раздела из книги Luke S. Essentials of Metaheuristics. A Set of Undergraduate Lecture Notes. Zeroth Edition. Online Version 0.5. October, 2009 (`http://cs.gmu.edu/~sean/book/metaheuristics/`). Перевел – Юрий Цой, 2009 г. Любые замечания, касающиеся перевода, просьба присылать по адресу yurytsoy@gmail.com
Данный текст доступен по адресу: `http://qai.narod.ru/GA/meta-heuristics_10_1.pdf`

[1] В отличие от большинства уже рассмотренных тем, данная представляет собой весьма специфичную задачу, а не широкое направление исследований. Однако здесь она приводится благодаря тому, что ее решение привело к появлению необычных и важных специализированных метаэвристик. Кроме того, данная задача имеет достаточно большую важность, и поэтому мы уделяем ей внимание.

[2] Если есть желание глубже познакомиться с $Q$-обучением и родственными методами, то классическим произведением по обучению с подкреплением является Richard Sutton and Andrew Barto, 1998, Reinforcement Learning: an Introduction, MITPress. Эта замечательная книга доступна онлайн по адресу `http://www.cs.ualberta.ca/~sutton/book/the-book.html`

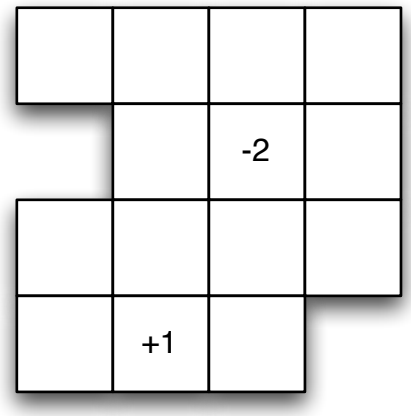


Рис. .1: **Рис. 61**. Мир с вознаграждениями для робота-таракана (пустые клетки соответствуют нулевому вознаграждению)

О каких окружениях идет речь? Рассмотрим пример: мир робота-таракана поделен на квадраты, определяемые их GPS координатами. Когда робот пытается переместиться с одного квадрата на другой (т.е. идет на север, юг, восток или запад), то иногда у него это получается, но с некоторой вероятностью его сдувает ветром на случайный соседний квадрат. Некоторые квадраты блокируют путь робота в определенном направлении (возможно, там находится стена). В некоторых квадратах есть разные вкусные штуки. А в других робота бьет током. Робот не знает заранее, где находится еда, а где высокое напряжение, он просто пытается определить для каждого квадрата своего мира, в каком направлении двигаться, чтобы съесть как можно больше еды и при этом получить как можно меньше ударов тока за все время своей жизни.

Справа показан возможный мир робота-таракана, в котором если робот попадает в один квадрат, то он получает вкусную награду (+1), а в другом — электрический разряд (-2).

В данном примере робот-таракан это наш **агент** (***agent***). Квадраты представляют **внешние состояния** (***external states***) (или просто **состояния**, ***states***), в которых может оказаться агент. Направления, в которых таракан пытается двигаться — это действия, доступные агенту. В различных состояниях наборы действий могут отличаться (в нашем случае из-за присутствия стен). Вкусные штуки дают **положительное подкрепление** (***positive reinforcement***) или **положительное вознаграждение** (***positive reward***), а удары электричеством соответственно **отрицательное подкрепление** (***negative reinforcement***), или **наказание** (***punishment***), или **отрицательное вознаграждение** (***negative reward***) (если так можно сказать). Попытку агента максимизировать положительное подкрепление в течение жизни также известно как максимизация **полезности**[3] (***utility***) агента (или **ценности**, ***value***). Вероятность сдувания агента в новое состояние в зависимости от текущего и выбор действия называют **моделью переходов** (***transition model***). Агент, как правило, не знает о модели переходов, но в действительности она существует.

Причина, по которой в данном контесте правило *если* →*то* называется *правилом состояние-действие* в том, что *если* обозначает возможное внешнее состояние, а *то* — какое действие следует совершить в этом состоянии. Агент пытается сформировать такой набор правил, по одному для каждого состояния, который описывает все возможные действия, необходимые для выживания в данном мире. Это множество правил известно также как **правила поведения** (***policy***) и традиционно [4] обозначается функцией $\pi(s)$, которая возвращает действие $a$, требуемое для данной ситуации $s$. На рис. 62 показаны возможные оптимальные правила для мира таракана.

Рассмотрим другой пример. Допустим, мы хотим научиться играть в крестики-нолики (играем за X) против случайного противника имея только информацию о победах и поражениях. Каждая возможная позиция на доске перед ходом X может рассматриваться как *состояние*. Для каждого состояния имеется определенное количество возможных ходов X, составляющих набор доступных *действий*. После случайного хода оппонента имеем дело с новым состоянием, и вероятность того, что сделанный ход в данном состоянию приводит к новому состоянию определяет *модель переходов*. Некоторые действия в определенных состояниях приводят к награде или вознаграждению,

[3] Не путать с *полезностью* из Раздела 8.4

[4] Да, использовать $\pi$ в качестве имени функции — глупо

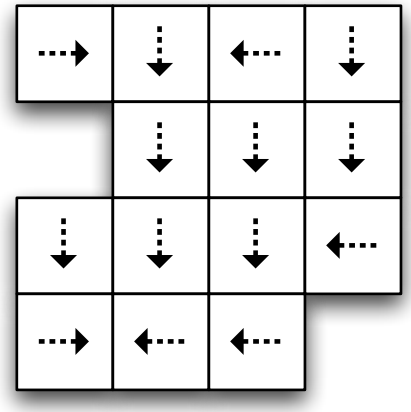

Рис. .2: **Рис. 62**. Оптимальные правила действий для мира робота-таракана

поскольку немедленно влекут победу или поражение. Это и есть *подкрепление*.

К примеру, если X делает ход в ячейке + в состоянии

| | | |
|---|---|---|
| X | X | + |
| 0 | – | X |
| 0 | Δ | 0 |

, то X получает положительное подкрепление, т.к. X выигрывает партию. Если X сделает ход в ячейке –, то возможно сразу же проиграет и получит отрицательное подкрепление, при условии, что противник не будет делать глупостей [5] (нужно помнить, что следующее состояние наступает только *после* хода оппонента). Если же X сходит в ячейку Δ, то не получит следующим ходом подкрепления, т.к. игра должна будет (еще немного) продолжиться. Отсутствие подкрепления также является подкреплением, только с нулевым результатом. В конечном итоге, нам необходимо научиться таким правилам поведения, которые укажут действия для любой конфигурации на доске.

Вот третий пример, заимствованный в работе Минору Асады [6] о роботе-футболисте. Робот пытается научиться бить по мячу так, чтобы попасть в цель. Робот оснащен камерой и он получает информацию о том что видит в следующей форме: является ли мяч *не видимым*, или же он *слева*, *справа* или *по центру* относительно поля зрения робота. Если мяч видно, то известно, что он *маленький* (далеко), *средний* или *большой* (близко). Аналогично цель либо невидима, либо слева, справа или по центру, а если видима, то либо маленькая, средняя или большая. По этим данным можно составить десять различных ситуаций для мяча (невидимый, слева маленький, слева средний, слева большой, по центру маленький, по центру средний, по центру большой, справа маленький, справа средний, справа большой) и аналогичные ситуации для цели. Состояние представляет пару ситуаций для мяча и цели, поэтому всего имеем 100 состояний. Робот может двигаться вперед, налево, направо, назад, назад налево и назад направо. Итого для каждого состояния есть 6 вариантов действий. Робот получает положительное подкрепление, если мяч попадает в цель, в противном случае подкрепление равно нулю.

Приложения не ограничиваются только лишь роботами и играми: обучение с подкреплением широко используется во всех областях от принятия производственных решений до азартных игр, от управления двигателем автомобиля для повышения эффективности путем изменения объема впрыска топлива до моделирования противоборства стран и коммерческой конкуренции. Оно действительно много где используется.

Все описанные примеры имеют схожие свойства. Во-первых, имеется конечное число состояний. Во-вторых, для каждого состояния характерно фиксированное число действий, хотя количество и эффект действий могут отличаться для разных состояний. В-третьих, мы руководствуемся предположением, что применение действия в одном состоянии приводит с **определенной вероятностью** к **переходу** к другому состоянию. Это может выглядеть нелепо, но для решения проблемы такая мера является необходимой. В-четвертых, мы считаем, что совершая определенные действия в определенном состоянии, мы получаем вознаграждение, причем эти вознаграждения либо **детерминированы**, либо с **заданной вероятностью** в зависимости от состояния и действия. Это также достаточно нелепое предположение, но оно помогает упростить постановку задачи. И, наконец, еще

[5] Что естественно, т.к., учитывая ситуацию, наш случайный оппонент далеко не образец благоразумия

[6] Помимо всего прочего, Минору Асада является со-основателем чемпионата роботов по футболу RoboCup.

одно нелепое предположение: вероятности переходов *полностью* зависят от текущего состояния и действия — ранние действия и состояния не влияют на распределение вероятности, если не считать, что они влияют на то, как мы оказались в текущем состоянии. Другими словами, чтобы определить наилучшее действие для текущего состояния, нам **не требуется память** о предыдущих действиях. Нам просто нужны правила *если* $\rightarrow$*то*, которые предписывают, что делать в *текущей* ситуации. Последнее преположении общеизвестно как предположение **Марковской** [7] **среды**. Очень мало количество реальных ситуаций являются марковскими, однако это предположение существенно облегчает постановку и решение задачи, поэтому мы будем его использовать везде, где возможно и если это не приведет к полному безумию.

### 10.1.1 $Q$-обучение

$Q$-обучение является популярным алгоритмом обучения с подкреплением, знать который очень полезно, прежде чем переходить к эволюционным моделям. В $Q$-обучении агент придерживается определенных правил поведения $\pi(s)$ (наилучших среди использованных агентом ранее) и с их помощью перемещается в среде. Выучив, что некоторые действия являются не очень хорошими, агент обновляет и изменяет свой набор правил. Конечной целью является поиск оптимальных (наиболее разумных) правил поведения, т.е. таких правил, которые приносят наибольшее вознаграждение за время жизни агента. Оптимальные правила обозначаются $\pi^*(s)$.

Агент не хранит правила поведения, а запоминает более общую структуру — $Q$-таблицу. $Q$-таблица — это функция $Q(s,a)$ для каждого состояния $s$ и каждого возможного действия $a$ в этом состоянии. Эта таблица показывает, насколько хорошо находиться в состоянии $s$, а затем совершить действие $a$, и после этого *следовать оптимальным правилам поведения*. Таким образом, $Q$-значение соответствует **полезности** (***utility***) действия в состоянии $s$ для идеального агента (в отличие от нашего первоначального выбора $a$). В самом начале $Q$-таблица агента содержит информацию с большим количеством неточностей, и агент пытается обновить их, пока они не приблизятся к *оптимальной $Q$-таблице*, обозначаемой $Q^*(s,a)$, в которой вся информация — абсолютно точна. Для данного состояния $s$ считается, что соответствующее ему наилучшее действие $a$ (т.е. $\pi^*(s)$), имеет наибольшее значение $Q^*$ по сравнению с другими действиями. Поэтому мы можем определить $\pi^*(s) = \mathrm{argmax}_a Q^*(s,a)$, которое обозначает «действие $a$, при котором $Q^*(s,a)$ максимально».

Мир является марковским: когда агент, будучи в состоянии $s$, совершает действие $a$, то он переходит в новое состояние $s'$ с определенной вероятностью перехода $P(s'|s,a)$. В качестве результата агент получает вознаграждение $R(s,a)$. На рис. 64 показана модель состояние-действие для $Q$-обучения: агент совершает некоторое действие $a$, которое приводит его к одному из возможных состояний $s'$, и далее мы предполагаем (возможно, ошибочно) что в дальнейшем агент выбирает оптимальные действия из $\pi^*$.

В совершенном мире, когда мы заранее знаем $P(s'|s,a)$ существует волшебное уравнение для вычисления $Q^*(s,a)$:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a') \tag{1}$$

Это уравнение означает, что значение $Q^*$ для действия $a$ в состоянии $s$ равно средней сумме всех будущих вознаграждений, полученных на следующих шагах. Оно равно непосредственно получаемому вознаграждению плюс сумма по всем возможным состояниям $s'$, в которых мы можем оказаться, произведения вероятности перейти в некоторое состояние и значения $Q^*$ наилучшего действия $a'$ в этом состоянии. Определение рекурсивно[8].

[7] Андрей Андреевич Марков (1856-1922) — русский математик, наиболее известный за вклад в марковские цепи, которые представляют список состояний $Ss_1, s_2, \ldots$, в которых может оказаться агент после выполенния различных действий в марковской среде. Данное направление исследований представляет большой раздел теории вероятностей и является большой частью теории **стохастических процессов**, не путать со стохастической оптимизацией

[8] Уравнение для $Q^*$ выведено из знаменитого более простого уравнения **Ричарда Беллмана**, называемого **уравнением Беллмана**. В уравнении не присутствуют действия в явном виде, вместо этого предполагается что агент выполняет некоторую (возможно, субоптимальную) фиксированную последовательность действий $\pi$. Уравнение Беллмана выглядит следующим образом:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s,a) U(s')$$

$U(s)$ является эквивалентом $Q^*(s,a)$, но предполагает, что совершаемое действие $a$ всегда является $\pi(s)$. Кстати, $U$ сокращение от *Utility* (полезность), точно также как $R$ — сокращение от *Reward* (вознаграждение) или *Reinforcement* (подкрепление). Функцию вероятности как правило не обозначают как $P(s'|s,a)$, я написал так в целях соответствия правилам теории вероятностей, хотя обычно пишут $T(s',a,s)$. То есть $T$ — сокращение от

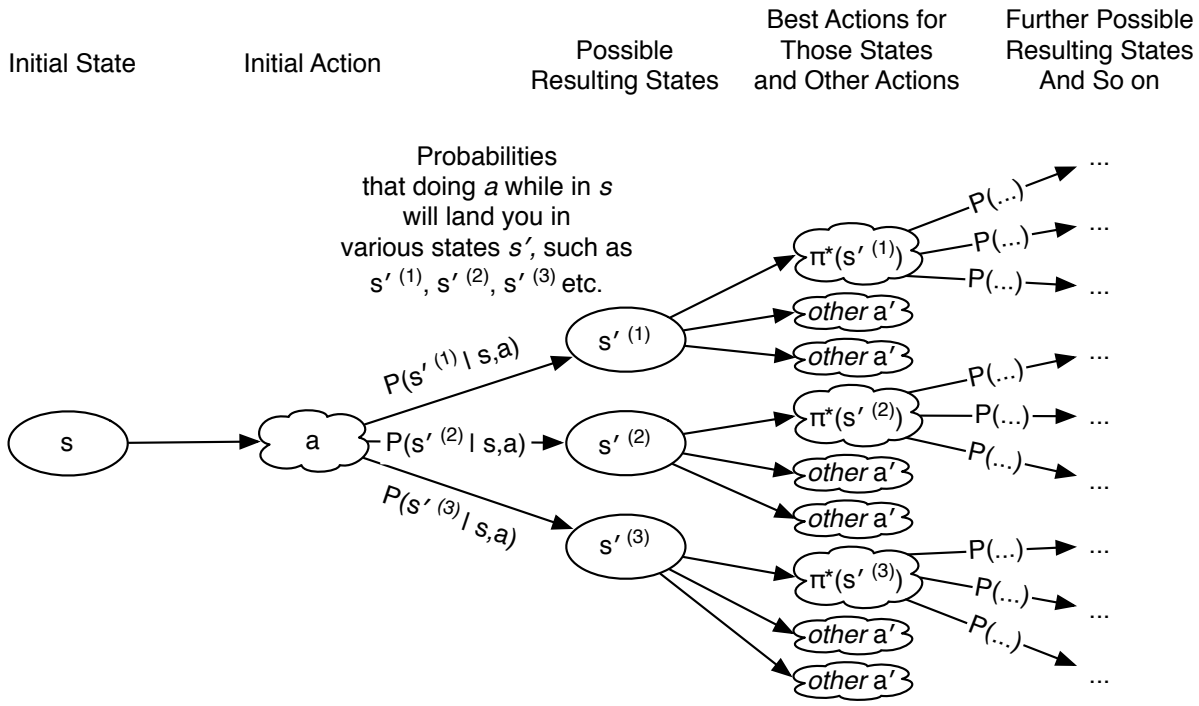


Рис. .3: **Рис. 63**. Модель состояние-действие для $Q$-обучения. Мы находимся в некотором состоянии $s$ и хотим совершить действие $a$. Совершив это действие, мы с определенной вероятностью $P(s'|s,a)$ окажемся в состоянии $s'$ (в данном примере таких состояний может быть три: $s'^{(1)}, s'^{(2)}, s'^{(3)}$). Предполагаем, что с этого момента мы выбираем *наиболее разумные действия* $\pi^*(s')$ для любого состояния $s'$, далее переходим в новые состояния, выбираем самое разумное действие и т.д. Отметим, что в данной модели самое первое действие $a$, которое мы совершаем, может не быть самым разумным.

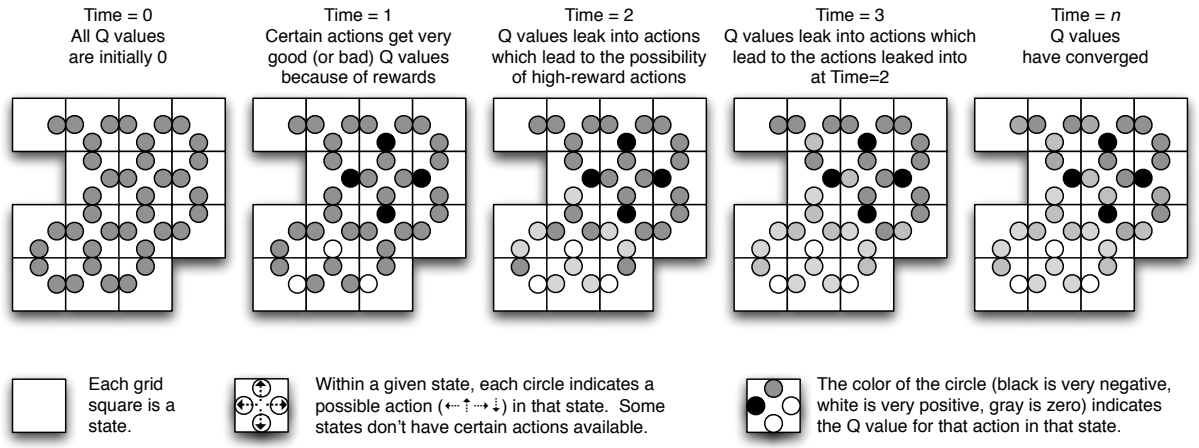


Рис. .4: **Рис. 64** Иллюстративный пример $Q$-обучения с моделью в мире робота-таракана. К моменту времени $= n$ наилучшее (самое светлое) действие в каждом состоянии соответствует оптимальному правилу действия, показанному на рис. 62

Все это сразу понять непросто, поэтому попробуем рассмотреть следующий пример. В различные моменты времени $t$ агент оказывается в различных состояниях $s_t$ и совершает различные действия $a_t$. При этом агент также получает и различные вознаграждения (которые могут оказаться равными 0). Таким образом, суммарное вознаграждение агента есть:

$$R(s_0, a_0) + R(s_1, a_1) + R(s_2, a_2) + R(s_3, a_3) + \dots$$

Предположим, что все действия агента — оптимальные, а также, для простоты, что вероятности отсутствуют, т.е. совершая действие в данном состоянии, агент всегда перейдет в строго определенное новое состояние. Тогда значение $Q^*(s_2, a_2)$ в момент времени 2 равно сумме всех последующих вознаграждений $(R(s_2, a_2) + R(s_3, a_3) + \dots)$. Аналогично, значение $Q^*(s_1, a_1)$ в момент времени 1 равно $(R(s_1, a_1) + R(s_2, a_2) + R(s_3, a_3) + \dots)$. Таким образом, $Q^*(s_1, a_1) = R(s_1, a_1) + Q^*(s_2, a_2)$. Точно также $Q^*(s_0, a_0) = R(s_0, a_0) + Q^*(s_1, a_1)$. Заметили сходство с уравнением (1)? В уравнении присутствует дополнительное слагаемое
$\sum_{s'} P(s'|s, a) \max_{a'} Q^*(s', a')$ вместо просто $Q^*(s', a')$, что объясняется необходимостью учитывать вероятности перехода $P$. Это слагаемое равно взвешенному среднему значению $Q^*$, полученному на последующих шагах.

Зачем нужно $\gamma$? Это ограничивающая константа в диапазоне от 0 до 1. Она используется, чтобы сделать еще не полученные будущие вознаграждения менее значимыми, чем текущее. Кроме этого, без нее значение $Q^*$ может стать бесконечным (что нежелательно).

Зная распределение $P(s'|s, a)$ и функцию $R(s, a)$, можно использовать это волшебное уравнение, чтобы вычислить $Q^*$. Например, так:

**Алгоритм 1 122** $Q$-обучение с моделью

$R(S, A) \leftarrow$функция вознаграждения за выполнение $a$ в состоянии $s$ для всех $s \in S$ и $a \in A$
$P(S'|S, A) \leftarrow$распределение вероятности, при котором выполнение $a$ в состоянии $s$ приводит к новому состоянию $s'$ для всех $s, s' \in S$ и $a \in A$
$\gamma \leftarrow$ограничивающая константа {Хорошо подходит $0 < \gamma < 1.5$}
$Q^*(S, A) \leftarrow$таблица значений полезности для всех $s \in S$ и $a \in A$, изначально заполняется нулями

**repeat**
  $Q'(S, A) \leftarrow Q^*(S, A)$ {Копирование всей таблицы}
  **for** каждое состояние $s$ **do**
    **for** каждое действие $a$, выполнимое в $s$ **do**
      $Q^*(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q^*(s', a')$
    **end for**
  **end for**
**until** $Q^*(S, A)$ не изменяется
**return** $Q^*(S, A)$

Таким образом, начиная с некоторых бессмысленных значений $Q^*$ и предполагая, что они корректны, мы постепенно складываем вознаграждения, пока значения $Q^*$ не перестают изменяться. Такой подход называется **бутстреппингом** (***bootstrapping***) и может представляться весьма безумным. Тем не менее, он отлично работает, благодаря особенности $Q$-обучения, обусловленной марковости среды: в мире $Q$-обучения **нет локального максимума**. Есть просто один большой глобальный оптимум и подобный подход к решению превращается в запутанный вариант локального поиска.

**$Q$-обучение в качестве обучения с подкреплением.** Только что рассмотренный алгоритм является примером того, что в инженерных кругах и исследовании операций известно как **динамическое программирование** (***dynamic programming***). Не следует путать это с аналогичным термином компьютерных наук [9] , в которых динамическое программирование представляет подход, позволяющий решать задачи определенного вида быстрее за счет разбиения их на пересекающиеся подзадачи. В инженерии динамическое программирование как правило подразумевает поиск правил поведения

*Transition Probability* (вероятность перехода). Хм, интересно, можно ли использовать $Q$ сокращение для *Q-tility* ($Q$-лезность)...

[9] Вообще-то в отношении этого вопроса между двумя упомянутыми направлениями исторически сложились некоторые отношения, но это длинная история. Достаточно сказать, что инженерный подход совсем немного опередил компьютерные науки.

агентов в марковских средах, при условии, что вероятности переходов $P$ и функция вознаграждения $R$ известны заранее.

С точки зрения искусственного интеллекта, если даны $P$ и $R$, то рассматриваемый алгоритм не представляет большого интереса. В *действительности* же намного более значимым является алгоритм, который устанавливает $Q^*$ без использования $P$ и $R$, а путем блуждания по среде и установления $P$ и $R$ на основании своего опыта. Такие алгоритмы часто называют **безмодельными** (***model-free***) алгоритмами, и обучение с подкреплением отличается от динамического программирования тем, что уделяет внимание именно безмодельным алгоритмам.

Мы можем заполнить $Q^*$ не зная $P$ или $R$, просто блуждая по среде и узнавая о ней новые интересные факты. С $R$ все легко: значения вознаграждений будут накапливаться по мере их поступления. А вот $P$ объяснить сложнее. Нам нужно заменить слагаемое $\sum_{s'} P(s'|s,a)$ в уравнении (1). Это слагаемое увеличивается с ростом частоты появления различных $Q^*(s',a')$. Его можно определить, если достаточно долго блуждать по среде и накапливать статистику переходов в $s'$. Со временем полученное распределение приблизится к $P(s'|s,a)$.

Итак, мы строим аппроксимацию $Q^*$ на основании откликов от взаимодействия с миром, обозначаемых $Q$. В начальный момент времени элементы таблицы равны нулю. По мере блуждания, в различных состояниях совершаются разные действия, которые приводят к новым состояниям и вознаграждениям. Допустим, что мы находимся в состоянии $s$ и решили совершить действие $a$. При этом мы перешли в состояние $s'$ и получили вознаграждение $r$, что позволяет обновить таблицу $Q$ следующим образом:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a')) \tag{2}$$

Заметьте, что мы отбрасываем часть уже известной информации, используя трюк с множителем $1-\alpha$. Подобное уже встречалось нам в Оптимизации муравьиной колонией (Раздел 8.3) и в Алгоритмах оценки распределений (Раздел 9.2.1). При этом мы добавляем часть новой информации, которую получили только недавно. Эта информация состоит складывается из уже известных слагаемых: вознаграждения $r$ и наибольшего значения $Q$ в следующем состоянии $s'$. Отметим сходство с уравнением (1). Обновленный алгоритм выглядит следующим образом:

**Алгоритм 2 123** Безмодельное $Q$-обучение

$\alpha \leftarrow$скорость обучения $\{0 < \alpha < 1$. Пусть будет небольшой.$\}$
$\gamma \leftarrow$ограничивающая константа $\{$Хорошо подходит $0 < \gamma < 1.5\}$
$Q(S,A) \leftarrow$таблица значений полезности для всех $s \in S$ и $a \in A$, изначально заполняется нулями

**repeat**
  Запуск агента в начальном состоянии $s \leftarrow s_0$ $\{$Будет лучше, если $s_0$ каждый раз будет разным.$\}$
  **repeat**
    Наблюдаем, как агент совершает действие $a$, переходит в новое состояния $s'$ и получает вознаграждение $r$
    $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$
    $s \leftarrow s'$
  **until** Пока агент жив
**until** $Q(S,A)$ не изменяется или пока не закончилось отведенное время
**return** $Q(S,A)$ $\{$В качестве аппроксимации $Q^*(S,A)\}$

Как агент определяет, какое действие выполнить? В случае абсолютно случайного выбора действия алгоритм медленно сойдется к оптимиму. Можно также выбирать наилучшее возможное действие для состояния $s$, т.е. использовать $\pi^*(s)$, также известное как $\text{argmax}_a Q^*(s,a)$. А, точно, мы же не знаем $Q^*$. В этом случае можем подменить ее, выбирая наилучшее из уже использованных действий, неполной $Q$-таблицей, т.е. $\text{argmax}_a Q(s,a)$.

Все это выглядит довольно неплохо. Однако появляется следующая проблема. Вернемся к нашему примеру с тараканом. Он бродил, бродил и нашел маленькую конфетку. Ням! Исследовав затем окрестность и не найдя ничего лучше этой конфеты, получим, что для каждого состояния в локальной окрестности $Q$-таблица таракана указывает вернуться к конфете. Было бы отлично, если бы эта конфета была единственно интересной вещью, но если бы таракан побродил еще *совсем немного*, то нашел бы гигантскую кучу сахара! К несчастью, он никогда ее не найдет и будет довольствоваться маленькой конфетой. Узнаете проблему? Это снова **исследование против экс-**

**плуатации** (***Exploration versus Exploitation***). Если мы будем использовать только наилучшее действие из числа уже испытанных, то $Q$-обучение будет на 100% эксплуатационным. Проблема заключается в том, что в безмодельной версии алгоритма, в отличие от версии для динамического программирования, **есть локальные экстремумы**. И мы оказались в одном из них. Решение приходит из стохастической оптимизации: нужно усилить исследование. Этого можно добиться добавив *некоторый* элемент случайности в выбор действия. Иногда будем совершать наилучшее известное действие, а иногда будем сходить с ума. Данный подход называется **$\epsilon$-жадный выбор действия** (***$\epsilon$-greedy action selection***), и он гарантирует выход из локального экстремума, хотя если доля случайности будет очень малой, нам возможно придется ждать очень долго. Либо можно использовать подход, напоминающий имитацию отжига, и сначала совершать только сумасшедшие действия, а потом понемногу только наилучшие известные.

Наконец, хорошо бы иметь константную $\alpha$ на протяжении всей работы алгоритма. Но можно получить и лучшие результаты, если уменьшать $\alpha$ для тех значений $Q(s, a)$, которые уже много раз обновлялись.

**Обобщение.** Хотите — верьте, хотите — нет, но была причина, по которой мы все вышеописанное разбирали. Обучение с подкреплением могло бы быть отличным подходом, если бы не присущая ему проблема: оно не обобщает. Обычно алгоритм обучения должен уметь производить утверждения общего характера для всей среды на основе всего лишь нескольких примеров ее работы. В этом заключается основной смысл алгоритма обучения. Если придется рассматривать каждую точку в пространстве, зачем тогда использовать алгоритм обучения? Мы и так будем знать всю Вселенную.

Обучение с подкреплением выучивает отдельное действие для каждой точки во всем пространстве (для каждого состояния). И даже еще хуже: $Q$-обучение определяет полезность для *каждой возможной комбинации состояния и действия.* Вспомните пример с роботом футболистом. Там было 100 состояний и 6 действий. Получается база данных из 600 записей! И это *маленькая среда.* Обучение с подкреплением не очень хорошо масштабируется.

Большинство подходов к решению этой проблемы являются вариантами дискретизации пространства и уменьшения его размера и сложности. Можно также использовать *второй* алгоритм обучения в процедуре обучения с подкреплением, обычно применяется нейронная сеть, чтобы выучить простой набор правил поведения для различных состояний, описываюших всю среду.

Еще одним подходом является применение метаэвристик для выучивания набора простых правил, описывающих среду в общем. Такие системы как правило используют эволюционные алгоритмы, чтобы разбить пространство состояний на области, внутри каждой из которых требуются одинаковые действия. Тогда каждое правило принимает вид *описание области* $\rightarrow$*действие.* Вместо того, чтобы иметь одно правило для каждого состояния, в данном случае будет одно правило для целой области, и мы можем попытаться использовать как можно меньше областей, чтобы корректно представить все пространство поиска целиком. Далее мы рассмотрим некоторые методы. Но сначала . . .

**Последнее отклонение.** Этот раздел можно пропустить. Основной его целью является показать, откуда берется волшебное уравнение

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q^*(s', a')$$

Для этого мы очень подробно рассмотрим само происхождение $Q^*(s, a)$. Для начала определим, что для каждого состояния $s$ и действия $a$, $Q^*$ показывает, насколько хорошо *начать* в этом состоянии $s$, затем выполнить действие $a$, а затем выполнять только наиболее разумные действия (т.е. будем использовать $\pi^*(s)$ для всех последующих $a$). Можно считать, что $Q^*$ есть ожидаемое значение суммарного вознаграждения, которое мы получим, начав в $s$, выполнив $a$ и будучи разумним, среди всех возможных последовательностей состояний и действий. Вот как это можно записать:

$$Q^*(s, a) = E[\sum_{t=0}^{\infty} R(s_t, a_t)|s_0 = s, a_0 = a, a_{t\geq 1} = \pi^*(s_t)] \tag{3}$$

Здесь есть проблема. Представим, что есть два действия, А и Б, и если всегда, вне зависимости от состояния, совершать действие А, то будем получать вознаграждение, равное 1. А если всегда выполнять Б, то получим вознаграждение, равное 2. Если жизнь нашего агента бесконечна, то и обе суммы будут равны бесконечности, хотя очевидно, что Б — предпочтительнее. Этого можно избежать, если ограничивать вознаграждение за будущие действия так, чтобы оно не оказывало

решающего влияния. Для этого добавим множитель $0 < \gamma < 1$ в степени $t$, чтобы уменьшить влияние будущих вознаграждений. Это приведет к тому, что суммы всегда будут конечны и теперь сумма для Б будет больше таковой для А.

$$Q^*(s,a) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a, a_{t\geq 1} = \pi^*(s_t)]$$

Теперь вынесем из общей суммы начальные состояние и действие $s$ и $a$, которые будут обозначены как $s_0$ и $a_0$. Вместе с ними будет вынесен также и множитель $\gamma$, равный $\gamma^0$.

$$Q^*(s,a) = E[\gamma^0 R(s_0, a_0) + \sum_{t=1}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a, a_{t\geq 1} = \pi^*(s_t)]$$

Далее нашей задачей является преобразование ожидаемого значения таким образом, чтобы оно *снова походило на самого себя в уравнении (3)*. Давайте приступим. Очевидно, что $\gamma^0 = 1$, поэтому можем легко от нее избавиться. Затем, в выражении для ожидаемого значения слагаемое $R(s_0, a_0)$ ни от чего не зависит, поэтому его можно вынести за скобки и переименовать параметры $s_0$ и $a_0$ обратно в $s$ и $a$.

$$Q^*(s,a) = R(s,a) + E[\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a, a_{t\geq 1} = \pi^*(s_t)]$$

После этого идет самая сложная часть вывода. нам нужно избавиться от $s_0$ и $a_0$, которые все еще присутствуют в выражении для ожидаемого вознаграждения. Для этого создадим новое состояние $s'$, которым будет $s_1$. Вспомним из рис. 63, что на самом деле существует множество возможных состояний $s'(1), s'(2), \ldots$, для каждого из которых задана вероятность перехода $P(s'(1)|s,a)$, $P(s'(2)|s,a), \ldots$, что состояние $s'$ — это такое состояние, в котором мы окажемся выполнив действие $a$ в состоянии $s$. Поэтому если мы удалим $s_0$ из выражения для ожидаемого вознаграждения, то ничего в этом выражении не будет о нем напоминать, и нам нужно будет просто обозначить в явном виде, что предыдущее выражение для ожидаемого вознаграждения «распалось»на множественные ожидаемые вознаграждения, по одному для каждого $s'$, и мы их складываем, предварительно умножив на соответствующие вероятности перехода. Получим:

$$Q^*(s,a) = R(s,a) + \sum_{s'} P(s'|s,a) E[\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t) | s_1 = s', a_{t\geq 1} = \pi^*(s_t)]$$

Теперь мы снова можем изменить во внутренней сумме нижний предел на $t = 0$, поскольку в выражении для ожидаемого вознаграждения ничего не зависит от момента времени 0. Поэтому просто переопределим $t = 1$ на $t = 0$. Вместе с этим каждое слагаемое будет умноженным на дополнительное $\gamma$, так что нужно будет также и $\gamma$:

$$Q^*(s,a) = R(s,a) + \sum_{s'} P(s'|s,a) E[\gamma \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s', a_{t\geq 0} = \pi^*(s_t)]$$

Так как $\gamma$ ни от чего не зависит, то вынесем его за внешнюю сумму:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s', a_{t\geq 0} = \pi^*(s_t)]$$

Отметим, что сейчас в выражении для ожидаемого вознаграждения есть $s_0$, но нет $a_0$. Мы исправим это, «раздробив»$a_{t\geq 0}$. Вместо определения $a_0$ как $\pi^*(s_0)$, введем новый символ $a'$, представляющий действие, совершаемое в состоянии $s'$, т.е. $a' = a_0$. Это позволит вынести $a'$ за пределы выражения для ожидаемого вознаграждения. Но, опять-таки, мы должны помнить, что $a'$ является наиболее разумным действием, которое только можно выполнить в состоянии $s'$. Для учета этого фактора введем оператор $max$, который выбирает $a'$, доставляющее наибольшее возможное ожидаемое вознаграждение (т.е. самое разумное действие, которым очевидно является $\pi^*(s_0)$).

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s', a_0 = a', a_{t\geq 1} = \pi^*(s_t)]$$

И, наконец, основное достижение всех этих манипуляций. Заметим, что ожидаемое вознаграждение (все, что следует после $max$) выглядит очень похоже на уравнение (3). Единственная разница в том, что вместо $s$ используется $s'$, а вместо $a$ — $a'$. Что позволяет записать:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a')$$

Вуаля! Словно по волшебству возникает рекурсивное определение!