10.2 Разреженная стохастическая оптимизация правил поведения

Как уже было сказано, основной проблемой обучения с подкреплением является то, что для каждого возможного состояние создается уникальное правило. Q-обучение в этом плане еще хуже, так как оно строит таблицу для каждой комбинации состояние/действие. При наличии большого числа состояний (и большого числа действий) количество записей в таблице будет велико. Существуют различные способы преодоления этой проблемы, включающие упрощение пространства состояний, либо попытку использования методов обучения, таких как нейронные сети, чтобы определить, какие из состояний требуют одних и тех же действий. Современные популярные методы, известные под общим названием **поиск правил поведения** (***policy search***), включают алгоритмы Reinforce[1] Рональда Вильямса (*Ronald Williams*) и Pegasus [2] Эндрю Нг (*Andrew Ng*) и Майкла Джордана (*Michael Jordan*).

Метаэвристики можно использовать, чтобы узнать разреженное представление простраства правил. Идея заключается в том, чтобы выучить такой набор правил, каждое из которых задает действие не для одного состояния, а для *набора* состояний, имеющих общие характеристики. Вместо того, чтобы определять правило для каждого состояния, ищется небольшой набор правил, которые в совокупности описывают все пространство в своего рода обобщающей манере.

Представим, что состояния описывают точку в N-мерном пространстве. Например для робота-футболиста можно использовать четыре измерения: размер мяча, положение мяча (включая «отсутствует »), размер цели и положение цели (включая «отсутствует »). В примере с тараканом в качестве измерений можно использовать следующие два: значения x и y координат положения таракана на координатной сетке. Для примера игры в «Крестики-нолики»размерностей будет девять: по одному для каждой клетки доски. В общем случае N-мерного пространства правило может описывать прямоугольную область в этом пространстве вместо некоторого точно определенного участка. Вот, например, возможное правило для робота-таракана:

$$x \geq 4 \text{ и } x \leq 5 \text{ и } y \geq 1 \text{ и } y \leq 9 \rightarrow \text{ идти вверх}$$

Подобное правило называется **классифицирующим** (***classification rule***), поскольку оно классифицирует (помечает) прямоугольную область от $\langle 4, 1\rangle$ до $\langle 5, 9\rangle$ действием «идти вверх». Говорят, что правило покрывает прямоугольную область. Конечной целью является поиск такого набора правил, который покрывает все пространство состояний и корректно классифицирует состояния в покрываемых областях действиями из оптимального набора правил. Например на рис. 65 представлен небольшой набор правил, которые совместно определяют точно такой же набор правил, что и представленный на рис. 62.

Если правила накладываются друг на друга (если проблема является **переопределенной** (***over-specified***)), то требуется **арбитражная схема** (***arbitration scheme***). Если бы я определял набор правил для этой схемы, то опирался бы на понятие **специфичности** (***specificity***): правила, покрывающие малые области, предпочтительнее правил, покрывающих более крупные области. Эта ситуация показана на рис. 65. Тем не менее, рассматриваемые далее методы используют другие подходы к арбитражу.

Существует два простых способа использования метаэвристик для поиска описываемых правил поведения:

- Потенциальное решение (или особь) представляет полный набор правил. Эволюция таких наборов правил известна как Системы правил по **Питт-подходу** (***Pitt Approach*** *Rule Systems*).
- Особь представляет единственное правило, а вся популяция описывает полный набор правил. Эволюция отдельных правил и их коллективное использование называют **Мичиганским подходом** (***Michigan Approach***) к обучающимся системам классификаторов или просто **Системы обучающихся классификаторов** [3] (***Learning Classifier Systems***, ***LCS***).

[0]Перевод раздела из книги Luke S. Essentials of Metaheuristics. A Set of Undergraduate Lecture Notes. Zeroth Edition. Online Version 0.6. October, 2009 (`http://cs.gmu.edu/~sean/book/metaheuristics/`). Перевел – Юрий Цой, 2010 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_2.pdf`

[1]Ronald J. Williams, 1992, Simple statistical gradient-following algorithms for connectionist reinforcement learning, in *Machine Learning*, pages229–256.

[2]Нетривиальная статья. Andrew Ng and Michael Jordan, 2000, PEGASUS: A policy search method for large MDPs and POMDPs, in *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 406–415.

[3]Не нужно путать их с алгоритмами классификации в машинном обучении, подобные упоминаемым в разделе 9.1.

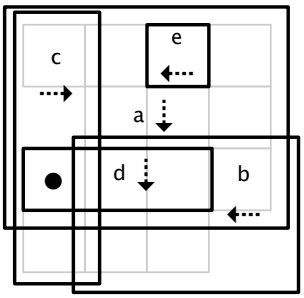

Рис. .65: Разреженная версия оптимальных правил поведения в мире робота-таракана, использующая пять правил ($a \ldots e$). Сравните с рис. 62. Состояние, помеченное символом • покрывается сразу тремя различными правилами (a, c и d), из которых d является наиболее специфичным.

10.2.1 Представление правила

В Q-обучении правила состояние-действие имеет следующий вид:

Если я в состоянии ... $\rightarrow$ *То выполнить ...*

Первая часть называется **телом правила** (***rule body***) и определяет состояния мира, которые это правило активируют. Вторая часть называется **головой тела** (***rule head***), она определяет действие, выполняемое при активации правила. Для того, чтобы правило описывало сразу несколько состояний, можно обобщить его одним из двух способов. Во первых, тело правила может требовать точного соответствий состояний:

Если я в состоянии, которое обладает в точности следующими свойствами ... $\rightarrow$ *То выполнить ...*

Либо можно использовать правила, в которых допускаются неточности при проверке состояний

Если я в состоянии, которое, пусть и с некоторой погрешностью, напоминает ... $\rightarrow$ *То выполнить ...*

В первом случае существует проблема **недоопределения** (***under-specification***): необходимо убедиться, что для каждого возможного состояния существует *некоторое* правило. Для этого можно использовать нечто вроде **правила по умолчанию** (***default rule***), которое выполняется, если соответствующее состоянию правило не найдено. Либо можно генерировать правила «на лету»и добавлять их к набору уже имеющихся, чтобы оно в последствии могло бы быть использовано для данного состояния.

Во втором случае волноваться о недоопределенности нет необходимости, поскольку каждое правило в *некоторой* степени соответствует каждому состоянию, но нужно определить понятие *степени* соответствия правил. Это понятие называют **показатель соответствия** (***match score***). Правило с наибольшим показателем соответствия выбирается с наибольшей вероятностью.

Как бы то ни было, необходимо учитывать переопределенность, которая требует использование арбитражной схемы. Вместо специфичности, известные методы используют комбинацию следующих понятий:

- Полезность правила, которая естественным образом задается как его Q-значение, определяемое агентом в результате попыток применения этого правила в различных ситуациях. Напомним, что полезность — это показатель того, как часто применение правила приводило к большим вознаграждениям. Соответственно правила с большей полезностью являются предпочтительнее менее полезных правил.

Эти алгоритмы используются для классификации всего пространства на основании имеющихся примеров этого пространства, которые были помечены заранее (поднаправление исследований, называемое **обучение с учителем** (***supervised learning***)). В отличие от этих алгоритмов метаэвристики, описываемые в этой главе, определяют классификацию для областей, используют исключительно информацию отклика, полученную в процессе блуждания по пространству.

- **Дисперсия** (***variance***) полезности. Если применение правила постоянно приводит к высоким вознаграждениям, то оно является предпочтительнее более «размытых» правил, которые хороши только время от времени.

- **Ошибка** (***error***) полезности правила. Вычисляется как разность между полезностью правила и полезностью правил, к которым оно приводит.

- Индекс соответствия правила. Правила, которые лучше подходят к текущей ситуации, предпочтительнее правил, тела которых не очень хорошо ей соответствуют.

Многие способы представления правил зависят от тела правила, которое может принимать различные формы, поэтому имеет смысл рассмотреть их:

Вещественные или целочисленные метрические пространства. Такие пространства состояний являются общепринятыми для систем правил по Питт-подходу, хотя их все больше исследуют и в методах Мичиганского подхода. Эти пространства могут быть описаны многими способами, но чаще всего используют прямоугольные ограничения (*boxes*). Вот некоторые из этих способов:

- **Прямоугольные ограничения** (***boxes***). Этот способ мы уже встречали:

  Пример: $x \geq 20$ и $x \leq 30$ и $y \geq 1$ и $y \leq 9 \rightarrow$ идти вверх

  Показатель соответствия: Если точка находится внутри ограничений, то соответствие идеальное (равно $1,0$?); в противном случае показатель равен проценту измерений, в которых ограничения все еще выполняются. Например, $\langle 40, 5 \rangle$ попадает в ограничения для y, и не попадает для x, что приводит к показателю соответствия $0,5$. Другой способ: показатель соответствия уменьшается с удалением от границы ограничений.

- **Тороидальные ограничения** (***Toroidal Boxes***). Если пространство состояний ограничено, то ограничения могли бы уходить за одну сторону, чтобы продолжиться противоположной. Представим, что пространство тороидально по координате x и ограничено по ней от 0 до 360. Данное ниже правило будет работать при $60 \leq x \leq 360$ или $0 \leq x \leq 20$ (в предположении, что y удовлетворяет ограничениям). И это не такая и сумасшедшая идея, к примеру, она полезна, если x описывает угол.

  Пример: $x \geq 60$ и $x \leq 20$ и $y \geq 1$ и $y \leq 9 \rightarrow$ идти вверх

  Показатель соответствия: Такой же, как и для обычных ограничений.

- **Гиперсферы или гиперэллипсоиды** (***Hyperspheres, Hyperellipsoids***). Правило может быть определено точкой (центром сферы) и радиусом. Либо правило может быть описано точкой и данными, описывающими повернутый многомерный эллипсоид (возможно, матрицей ковариаций, наподобие тех, которые используются для описания многомерных гауссовских поверхностей). Вот пример простой гиперсферы:

  Пример: Если состояние $\langle x, y, z \rangle$ принадлежит сфере с центром $\langle 4, 7, 2 \rangle$ и радиусом 9,3 $\rightarrow$ идти вверх

  Показатель соответствия: Вычисляется аналогично случаю обычных ограничений.

- **Образцы** (***Exemplars***). Определенные точки в пространстве, которые служат примерами для своих окрестностей. Набор правил с образцами разбивает пространство с помощью **многоугольников Вороного** [4] (***Voronoi tessellation***): областей пространства, ограниченных в зависимости от ближайшего образца. Такие правила полностью полагаются на показатели соответствия, поэтому некоторые методы (часто методы Мичиганского подхода) не могут их использовать. Образцы можно представлять как гиперсферы бесконечно малого радиуса.

  Пример: Если состояние ближе всего к $\langle 4, 7, 2 \rangle \rightarrow$ идти вверх

  Показатель соответствия: Чем дальше от образца, тем меньше показатель соответствия.

[4] В честь русского математика, Георгия Феодосиевича Вороного (1868-1908). Многоугольники Вороного (иногда называемые «диаграммами Вороного») широко используются во многих областях вычислительной геометрии, начиная от компьютерной графики и заканчивая беспроводными сетями и робототехникой.

- **Гиперплоскости** (***Hyperplanes***) [5]. Правило разрезает пространство плоскостью, деля его на область, в которой правило работает, и на область, в которой оно не применимо. Аналогично предыдущему способу применение гиперплоскостей в некоторых методах Мичиганского подхода может быть проблематично.

Пример:	Если $2,3x + 9,2y - 7,3z > 4,2 \rightarrow$ идти вверх
Показатель соответствия:	Если точка находится с «положительной»стороны гиперплоскости, то соответствие идеально (или же показатель улучшается по мере отдаления от гиперплоскости). Если точка с обратной стороны, то ее показатель соответствия невелик, однако он увеличивается при приближении к гиперплоскости.

Неметрические целочисленные пространства. Как мы уже видели ранее в Главе 4 (Способы представления), целочисленные пространства могут представлять **метрические пространства** (***metric spaces***), либо просто определять **неупорядоченные наборы объектов** (***unordered sets of objects***) (0 = «красный», 1 = «синий» и т.д.). Целочисленные пространства для тел правил устроены таким же образом. Правило с неупорядоченным целочисленным параметром может выглядеть вот так:

$$\text{Если } x = \text{ «красный» и } y = \text{ «мягкий» и } z = \text{ «пустой»} \rightarrow \text{ идти вверх}$$

В данном случае правило, как и образцы, описывает точное положение в (неупорядоченном) пространстве. Показатель соответствия может быть определен как количество переменных, значение которых точно совпадает с описанием данного состояния.

В неупорядоченных наборах правил могут содержаться дизъюнкции:

$$\begin{aligned}\text{Если } x = \text{ «красный» и } y = \text{ «мягкий» и}\\ z = \text{ «пустой» или «сплошной»} \rightarrow \text{ идти вверх}\end{aligned}$$

Дизъюнкция рассматривается как единое условие, которое истинно, если выполняется хотя бы одна его часть.

Булевские пространства. Системы классификаторов по Мичиганскому подходу традиционно применяют один вид правил: использующие булевские условия, хотя в последнее время этот подход был обобщен и для правил другого вида.

В силу своей простоты, булевские правила используют своего рода шаблон из комбинаций условий «да», «нет»и «не имеет значения». Допустим, каждое состояние в нашем пространстве состояний описывается тремя переменными x, y, и z. Таким образом, всего имеется 8 состояний. Булевское правило для трех измерений может выглядеть следующим образом:

$$x = 1 \text{ и } y = 0 \text{ (и } z \text{ не имеет значения)} \rightarrow \text{ идти вверх}$$

В нотации систем классификаторов Мичиганского подхода такое правило обычно записывают так:

$$10\# \rightarrow \text{ идти вверх}$$

Отметим, что символ # означает «значение данной переменной не важно». Чем больше измерений «не имеют значения», тем менее специфичным является правило. Показатели соответствия снова могут определяться как количество («значимых») переменных, полностью соответствующих рассматриваемому правилу.

Может ли тело правила быть деревом или графом? Или более сложной функцией? Кто знает?

10.3 Системы правил по Питт-подходу

Питт-подход[6] (***Pitt Approach***) использует эволюционный алгоритм для поиска набора правил, которые наилучшим образом описывают оптимальные правила поведения. Потенциальное решение

[5] Существует разумный способ преобразования гиперплоскостей в более сложные подобласти пространства, называемый **выделение ядер** (***kernelization***). Данный метод был популяризован в машинном обучении **машинами опорных векторов** (***Support Vector Machines, SVM***). Однако, применительно к системам правил, у меня с выделением ядер особо примечательных результатов не получилось.

[6] Кэн де Йонг со студентами разработали Питт подход в Университете Питтсбурга. Отсюда и название.

просто представляет собой этот набор правил. В Разделе 4.5 представлены популярные понятия для наборов правил, используемые в системах правил по Питт-подходу, и предолжены способы их инициализации, рекомбинации и мутации. В данном разделе будем рассматривать очень известный алгоритм Питт-подхода **Samuel**[7].

Samuel был разработан Джоном Грефенстеттом, Конни Рэмси и Аланом Шульцем в Военно-морской исследовательской лаборатории[8]. Идея заключается в применении Питт-подхода для оптимизации наборов правил, представленных как одно потенциальное решение в стохастической оптимизации, и в использовании идей обучения с подкреплением для улучшения правил внутри решения. В Samuel традиционно применяется генетический алгоритм, однако он может быть заменен практически любым другим методом оптимизации. Главное волшебство заключается в вычислении функции приспособленности, в которой дополнительно к приспособленности всего набора правил вычислется полезность каждого правила, и в операторах размножения. Итерации Samuel включают четыре простых шага:

1. Каждая особь тестируется n раз, и результаты используются для обновления полезностей ее правил.
2. На основании обновленной информации о полезности, правило каждой особи улучшается с использованием специальной процедуры мутации правил.
3. Каждая особь снова дополнительно тестируется m раз, и полученные результаты используются для обновления приспособленности особи (набора правил) как целого.
4. После того, как все особи прошли через три первых шага, производится традиционные для эволюционного алгоритма этапы размножения и селекции на основе приспособленности.

Вычисление приспособленности и полезности. Два шага вычислений (под номерами 1 и 3) практически идентичны, за исключением обновляемых данных, поэтому будем рассматривать их совместно, и, фактически, алгоритм 124 используется для описания обоих шагов.

В обеих процедурах вычисления приспособленности агент помещается в мир и управляется в соответствии с тестируемым набором правил. По мере блуждания агента необходимо решить, какое действие нужно совершить на каждом шаге. Для этого сначала определяется **подходящий набор правил** (***matching set***), содержащий правила, наиболее всего соответствующие текущему состоянию, т.е. имеющие наибольший показатель соответствия. Затем из них выбираются только правила с наибольшим вознаграждением. После этого в Samuel из оставшегося набора выбирается правило для действия на основании некоторого подобия процедуры селекции по значению вознаграждения. К примеру, можно просто выбрать правило с наибольшим вознаграждением, либо определить нужное правило с вероятностью пропорциональной вознаграждению (как в пропорциональной селекции, Алгоритм 30). Такой двухуровневый механизм (отсечение, затем селекция по вознаграждению) создан для предотвращения выбора из большого числа бесполезных правил в случае, если будет мало эффективных правил.

Первая процедура вычисления приспособленности обновляет полезности правил. Вспомним, что Q-обучение предполагает, что агент получает вознаграждение в течение всей своей жизни. В Samuel же, напротив, вознаграждение обычно определяется в конце жизни агента, что приводит к другим стратегиям распределения вознаграждений. В Q-обучении полученное вознаграждение сохраняется в Q-значении для соответствующей пары состояние-действие. В дальнейшем, когда другая комбинация состояния и действия приводит к рассматриваемому состоянию, его Q-значение частично распространяется на предшествующие комбинации. Позднее мы еще встретим подобное предположение в методах Мичиганского подхода, в Разделе 10.4. Однако в Samuel вознаграждение непосредственно и сразу распределяется по всем правилам состояние-действие, приведшим к нему. Такие

[7] Samuel является аббревиатурой от Strategy Acquisition Method Using Empirical Learning (*метод поиска стратегии, использующий эмпирическое обучение*). Да, она достаточно надумана. На самом деле Грефенстетт (*Grefenstett*), Рэмси (*Ramsey*) и Шульц (*Shultz*) искали способ, чтобы назвать алгоритм в честь Артура Самюэля (*Arthur Samuel*), известного пионера машинного обучения, которые (полагаю по совпадению) умер в год появления базовой статьи по алгоритму Samuel. Работая в IBM в 1950-х годах, Артур Самюэль разработал программу, которая сама училась играть в шашки и стала существенным шагом в истории искусственного интеллекта. Гм, у меня уже получилось много сносок про шашки...

Samuel был впервые описан в статье John Grefenstette, Connie Ramsey, and Alan Schultz, 1990, Learning sequential decision rules using simulation models and competition, *Machine Learning*, 5(4), 355–381. Хотя в CiteSeer^x доступна онлайн приблизительная текущая версия руководства, по адресу `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.9876`

[8] Там же изобрели радар!

правила называют **активными** (***active***). Более точно: если правило содержало действие, которое было использовано в некоторый момент времени в прошлом до получения вознаграждения r, то когда получено r, полезность правила обновляется в соответствии с выражением:

$$\text{Полезность}(R_i) \leftarrow (1-\alpha)\text{Полезность}(R_i) + \alpha r.$$

В Samuel также сохраняется аппроксимация **дисперсии** полезности для каждого правила, поскольку необходимо получить правила, которые не только приводят к высоким вознаграждениям, но и делают это *с большим постоянством*. Каждый раз при обновлении полезности дисперсия полезности также обновляется:

$$\begin{aligned}\text{ДисперсияПолезности}(R_i) \leftarrow (1-\alpha)&\text{ДисперсияПолезности}(R_i)\\ &+ \alpha(\text{Полезность}(R_i) - r)^2.\end{aligned}$$

Алгоритм Samuel использует эту информацию для определения своеобразного «качества» правила, которое называется **силой** [9] (***strength***) правила, и представляет комбинацию полезности и дисперсии полезности. Сила влияет на вероятность последующей мутации правила.

$$\text{Сила}(R_i) \leftarrow \text{Полезность}(R_i) + \gamma\text{ДисперсияПолезности}(R_i).$$

Обычно значение γ устанавливается небольшим, меньше 1, т.к. полезность важнее, чем дисперсия полезности.

Равномерное распределение вознаграждения по всем правилам выглядит странно. Я бы сделал так, чтобы поздние правила получили большее вознаграждение, чем ранние. Любопытно, что в Samuel используется информация о том, как давно активизировалось данное правило, хотя это необходимо только чтобы определить, какие правила необходимо удалить. Это значение называется **уровнем активности** (***activity level***) правила. В начале уровень активности правил равен 1/2 и обновляется каждый раз, когда агент совершает действие. Правила, содержащие данное действие, обновляются следующим образом:

$$\text{Активность}(R_i) \leftarrow (1-\beta)\text{Активность}(R_i) + \beta.$$

При $0 \leq \beta \leq 1$ это приводит к смещению активности правила к 1, если выбирается действие из этого правила. Для правил, не содержащих данного действия уровень активности понижается:

$$\text{Активность}(R_i) \leftarrow \delta\text{Активность}(R_i).$$

для $0 \leq \delta \leq 1$, что в результате приводит к снижению уровня активности в сторону нуля.

Вторая процедура оценки в Samuel необходима для вычисления приспособленности всей особи (набора правил). Она просто определяется как суммарное вознаграждение, полученное особью в ходе тестирования. Следующий алгоритм описывает обе процедуры оценки. Конкретный выбор процедуры (для полезности или для приспособленности) определяется значением переменной $dofitness$.

Мутация. В Samuel используется два шага мутации, каждый из которых следует за оцениванием. После первой процедуры оценки (которая определяет силу правила), изменяются *правила* особей. Предполагается, что это приведет к улучшению особи для второго оценивания (цель которого — вычисление приспособленности особи). После второй процедуры оценивания, производится обычное размножение с более традиционными операторами, оказывающими большее влияние на хромосому.

Рассмотрим сначала первый шаг мутации: улучшение правил. Samuel выполняет любой из следующих операторов мутации над особью в попытке улучшить ее для второго оценивания:

- **Удаление правила (*Rule Deletion*)**. Если правило является достаточно старым (новые правила никогда не удаляются), имеет достаточно низкое значение активности (не было задействовано в недалеком прошлом), либо достаточно малую силу, либо если это правило «*поглощено*»(*subsumed*) другим правилом, обладающим большей силой, то такое правило становится кандидатом на удаление. Можно также удалить несколько правил случайно. Вам решать, какие устанавливать пороги и сколько производить операций удаления. Говорят, что правило A поглощено правилом B, если B соответствует всем состояния A, учитывает кроме этого ряд других состояний, *и* эти правила имеют одинаковые действия в головной части.

[9] Не путать с Парето «силой»(Раздел 7.3).

Алгоритм 124 Вычисление приспосоленности в SAMUEL

S ←оцениваемая особь
α ←скорость обучения и затухания
β ←скорость повышения уровня активности
γ ←доля дисперсии
δ ←скорость затухания уровня активности
$dofitness$ ←определяет, вычисляется ли приспособленность
n ←количество испытаний при тестировании агента

$f \leftarrow 0$
$R \leftarrow \{R_1, \ldots, R_l\}$ правила из набора правил особи S
for n раз **do**
  $s \leftarrow$ начальное состояние агента
  $Z \leftarrow \{\}$ {Активный набор правил}
  for каждое правило $R_i \in R$ **do**
    Активность$(R_i) \leftarrow 0,5$
    repeat
      for каждое правило $R_i \in R$ **do**
        ComputeMatchScore(R_i, s)
      end for
      $N \leftarrow$ все действия в головной части любого правила в R
      $Z \leftarrow \{\}$ {Подходящий набор}
      for каждое действие $N_j \in N$ **do**
        $R' \subseteq R \leftarrow$ все правила в R, содержащие в головной части действие N_j
        $M \leftarrow M \cup \{$ правило $R'_j \in R'$ с наибольшим показателем соответствия $\}$
      end for
      $R_a \leftarrow SelectWithReplacement(M)$ {Выбор из наиболее подходящих правил}
      $A \subseteq R \leftarrow$ все правила, у которых голова (действия), такие же, как и голова в R_a {Набор действий}
      for каждое правило $A_i \in A$ **do**
        Активность$(R_i) \leftarrow (1-\beta)$Активность$(R_i) + \beta$
        if $A_i \notin Z$ **then**
          $Z \leftarrow Z \cup \{A_i\}$
        end if
      end for
      for каждое правило $R_i \in R - A$ **do**
        Активность$(R_i) \leftarrow \delta$Активность(R_i)
      end for
      Выполнить действие R_a, приводящее к новому состоянию s {Без вознаграждения}
    until Жизнь агента не закончена
  end for
  $r \leftarrow$ кумулятивное вознаграждение (оценка) агента {Ага, вот и вознаграждение. В самом конце.}
  if $dofitness$ ложно **then**
    for каждое правило $Z_i \in Z$ **do**
      Полезность$(Z_i) \leftarrow (1-\alpha)$Полезность$(Z_i) + \alpha r$
      ДисперсияПолезности$(R_i) \leftarrow (1-\alpha)$ДисперсияПолезности$(R_i) + \alpha($Полезность$(R_i) - r)^2$
      Сила$(R_i) \leftarrow$ Полезность$(R_i) + \gamma$ДисперсияПолезности(R_i)
    end for
  else
    $f \leftarrow f + r$
  end if
end for
if $dofitness$ ложно **then**
  Приспособленность $S \leftarrow f$
end if

- **Уточнение правила (*Rule Specialization*)**. Если правило — не очень сильное и соответствует большому числу состояний, то оно является кандидатом на *уточнение*, поскольку оно может быть слишком расплывчатым *из-за* накрытия большой области. В набор правил добавляется новое правило, поглощаемое старым (и потому более точное), и которое имеет в головной части такое же действие. При этом оригинальное правило сохраняется. К примеру правило

  $$x \geq 4 \text{ и } x \leq 5 \text{ и } y \geq 1 \text{ и } y \leq 9 \rightarrow \text{ идти вверх}$$

  может быть уточнено так:

  $$x = 5 \text{ и } y \geq 6 \text{ и } y \leq 9 \rightarrow \text{ идти вверх}$$

- **Обобщение правила (*Rule Generalization*)**. Противоположно уточнению правила. Если правило очень сильное и накрывает *малое* число состояний, то оно становится кандидатом на обобщение, поскольку может подходить и для большего набора состояний. В набор правил добавляется новое, поглощающее старое (и потому являющееся более общим) и имеющее в головной части такое же действие. При этом оригинальное правило сохраняется.

- **Покрытие правил (*Rule Covering*)**. Покрытие похоже на обобщение, но опирается на информацию, полученную в процессе оценивания. Предположим, что во время оценивания обнаружилось, что некоторое правило часто используется, но при этом весьма часто не *полностью* соответствует текущему состоянию. К примеру, вернемся к уже знакомому правилу

  $$x \geq 4 \text{ и } x \leq 5 \text{ и } y \geq 1 \text{ и } y \leq 9 \rightarrow \text{ идти вверх}$$

  Представим, что это правило часто выбиралось при $y = 4, x = 6$. Очевидно, что $x = 6$ не удовлетворяет границам применимости правила, хотя $y = 4$ вполне подходит, и правило является достаточно сильным, чтобы быть выбранным, даже несмотря на неполное соответствие. При операции покрытии, это правило может быть выбрано для создания нового, более «адекватного», например:

  $$x \geq 4 \text{ и } x \leq 6 \text{ и } y \geq 1 \text{ и } y \leq 9 \rightarrow \text{ идти вверх}$$

  Оригинальное правило сохраняется.

- **Слияние правил (*Rule Merging*)**. Если два правила достаточно сильны, их действия в головной части совпадают, и достаточно существенно пересекаются, в смысле учитываемых состояний, то они являются кандидатами на то, чтобы быть слитыми в одно правило, представляющее их объединение. Оригинальные правила остаются.

Отметим, что все эти механизмы мутации — **направленные** (***directed***), т.е. они явно эксплуатирующие, нацеленные на улучшение работы правил. По этой причине Джон Грефенстетт (*John Grefenstette*) называет этот шаг мутации **Ламарковским** (см. Раздел 4.3.4) — он улучшает особей в процессе оценивания.

Оставшиеся операторы мутации производятся во время размножения, также как и в любом другом эволюционном алгоритме, и имеют более исследовательскую природу:

- **Старая добрая мутация (*Plain Old Mutation*)**. Производит случайные изменения правил. Оригинальные правила не сохраняются. Это более исследовательская мутация.

- **Незаметная мутация**[10] (***Creep Mutation***). Производит очень малые локальные случайные изменения для нескольких правил. Основная цель — слегка подтолкнуть для локального поиска.

Рекомбинация. В Разделе 4.5.5 упоминаются различные подходы к скрещиванию правил. В SAMUEL рассматриваются другие варианты:

- Разновидность **однородного кроссинговера** (***Uniform Crossover***). Две особи случайно обмениваются n правилами.

[10] Мой выбор за самое незаметное имя для мутации.

- **Кластеризованный кроссинговер** (***Clustered Crossover***). В ходе оценивания особей собирается некоторая статистика, в частности, необходимо знать, какие последовательности правил привели к вознаграждению. Далее можно определить пары правил, которые часто давали вознаграждение, в случае появления *обоих* в последовательности. Затем производится однородный кроссинговер, но в конце процессе необходимо убедиться, что эти пары не оказали разбитыми. Если одно правило перешло к особи A, а второе к особи B, то мы перемещаем одно из этих правил к другой особи (заменив его некоторым другим правилом). Идея заключается в выявлении *очень сильной связанности* (*very strong linkage*) отдельных правил в наборе правил, и скрещивании с обменом *групп* правил, которые хорошо работают вместе.

Отметим, что оба описанных оператора скрещивания не изменяют размера набора правил. Не приводят к этому и операторы мутации в процессе размножения. В Samuel изменять размеры правил разрешается только эксплуатационным «Ламарковским» операторам мутации, работающим после первой процедуры оценивания.

Селекция. Допускается использование любой процедуры селекции, основанной на значениях приспособленности. Хотя в Samuel традиционно применяется своеобразная комбинация селекции усечением и стохастического универсального отбора (*Stochastic Universal Sampling*). В частности для всей популяции вычисляются средняя приспособленность и дисперсия приспособленности, а затем обновляется **базовая приспособленность** (***baseline fitness***):

$$\text{базовая} \leftarrow (1-v)\text{базовая} + v(\text{средняя приспособленность} - \psi\text{дисперсия приспособленности})$$

где $0 \leq v \leq 1$, ψ — параметр, определяющий насколько важна дисперсия. Зная базовую приспособленность, можно определить особей, которые хотя бы *рассматриваются* для селекции и приспособленность которых выше базовой. Для отбора среди таких особей применяется стандартная процедура селекции (в Samuel используется *Stochastic Universal Sampling*).

На самом деле, представляет интерес вопрос: будет ли успешным применение обычной селекции усечением?

Инициализация. Существуют различные способы инициализации набора правил. В Samuel обычно используются три:

- Создание набора случайных правил.
- Начальные правила особи генерируются с учетом полезных, на ваш взгляд, правил.
- **Адаптивная инициализация** (***adaptive initialization***). В начале у каждой особи имеется набор полностью обобщенных правил, по одному для каждого действия:

  Во всех случаях $\rightarrow$ идти вверх Во всех случаях $\rightarrow$ идти влево ...

  Чтобы определить силу каждого правила на некоторое время запускают алгоритм, а затем применяют достаточное число операторов **уточнения правил** (***Rule Specialization***), описанных выше, чтобы сделать эти общие правила более точными. Идея заключается в том, чтобы предоставить Samuel возможность самостоятельно найти хорошие начальные операторы на основании небольшого опыта.

Самоадаптивные операторы. В Samuel испльзуется еще один прием для подстройки вероятностей применения операторов мутации (в частности «Ламарковских»). Каждая особь хранит свои вероятности операторов. Обозначим через $P(O_i, I_j)$ вероятность, что оператор O_i будет применен к особи I_j. Эту информацию особь I_j хранит самостоятельно, а потомки наследуют такой же набор вероятностей, который был у родительских особей. На каждом шаге все вероятностей операторов для всех особей уменьшаются следующим образом:

$$P(O_i, I_j) \leftarrow (1-\tau)P(O_i, I_j)$$

где $0 \leq \tau \leq 1$. Это ведет к уменьшению вероятностей до 0. Однако при скрещивании или мутации особей с использованием некоторого оператора, его вероятность у создаваемых особей увеличивается. Например, возможен такой вариант:

$$P(O_i, I_j) \leftarrow (1-\tau)P(O_i, I_j) + \tau$$

Что ведет к увеличению вероятности до 1.

Это пример **само-адаптивных операторов** (***self-adaptive operators***), когда особь содержит свои значения вероятностей для кроссинговера и мутации. Само-адаптивные операторы используются уже довольно давно, начиная с первых работ по эволюционным стратегиям. Но по моему личному опыту, их не так-то легко заставить работать так, как нужно[11]. Я бы не стал их использовать.

10.4 Мичиганский подход к системам обучающихся классификаторов

После того, как приблизительно в 1973 г. Джон Холланд[12] (*John Holland*) разработал генетический алгоритм, он обратил свое внимание на смежную область: как использовать эволюционный процесс для формирования набора правил, определяющих, что делать в каждой ситуации, в которой может оказаться агент. При этом, на мой взгляд, Холланд ставил задачу более широко — как поиск решения для общей задачи машинного обучения, а не только для действий агента, что в результате повлияло на название: системы обучающихся классификаторов, СОК (***Learning Classifier Systems, LCS***). Вместо подхода, в котором особь представляет полное решени (набор правил), Холланд предложил использовать *популяцию* правил, которые борются за выживание на основании того, насколько они эффективны для классификатора, как целого. Таким образом, как и в оптимизации с использованием муравьиной колонии, системы обучающихся классификаторов напоминают по духу однопопуляционный коэволюционный подход.

Изначально, оригинальная идея Холланда была довольно вычурной. Позднее Стюарт Уилсон (*Stewart Wilson*) разработал более рациональную версию алгоритма, названную **классифицирующая система нулевого уровня**[13] (***Zeroth Level Classifier System, ZCS***). ZCS является устойчивым эволюционным алгоритмом, и итерации эволюционного цикла производятся только время от времени. Большая часть времени затрачивается на обновление приспособленности для всех особей поколения, коллективно взаимодействующих наподобие правил в обучении с подкреплением. Спустя некоторое время из текущей популяции создаются новые правила, которые добавляются в нее, замещая правила с низкой приспособленностью.

В ZCS используется популяция разреженных правил в виде *если* $\rightarrow$ *то*. Каждому правилу поставлена в соответствие приспособленность, отражающая его полезность. Для оценки правил агент переводится в начальное состояние, а затем совершает действия, выбранные из популяции. Выбор правила осуществляется сначала путем отбора правил, соответствующих текущему состоянию агента, которые формируют **набор подходящих правил** (***match set***) M. Если отобрано более одного правила, то арбитражная схема ZCS выбирает правило из набора, используя метод селекции, основанный на приспособленности (обычно применяется пропорциональная селекция).

Одно из отличий ZCS от Samuel заключается в требовании полного соответствия правил состоянию, вместо частичного, поэтому показатели соответствия не используются. Если набор подходящих правил пуст, т.е. не содержит ни одного правила, соответствующего данному состоянию, ZCS создает случайное правило, покрывающее состояние (и, возможно, некоторые другие) и содержащее случайное действие. Приспособленность правила устанавливается равной средней приспособленности популяции. После этого ZCS отмечает существующее правило, как доступное для удаления, и замещает его новым правилом. Обычно правила помечаются для удаления посредством селекции по приспособленности, которая чаще выбирает малоприспособленные правила.

Определив правило-победитель, в ZCS создается подмножество набора подходящих правил, называемое **набором действия** (***action set***) A, в который входят правила, имеющие в головной части то же самое действие, что и победитель. Это действие выполняется, агент получает вознаграждение r и переходит в новое состояние s', после чего ZCS формирует новый набор подходящих правил M'

[11] Моя диссертация изначально должна была быть посвящена самоадаптивным операторам. Но я просто, скажем так, переключился на другую тему.

[12] Джон Холланд работает в Мичиганском университете, что отразилось на названии метода. Одна из ранних работ Холланда по рассматриваемой теме: John Holland, 1975, *Adaptation in Natural and Artificial Systems*, University of Michigan Press. Однако понятие систем обучающихся классификаторов не было строго формализовано до публикации статьи John Holland, 1980, Adaptive algorithms for discovering and using general patterns in growing knowledge bases, *International Journal of Policy Analysis and Information Systems*, 4(3), 245–268.

[13] Описана в Stewart Wilson, 1994, ZCS: A zeroth level classifier system, *Evolutionary Computation*, 2(1), 1–18.

и набор действия A'. Затем для каждого действия $A_i \in A$ обновляется приспособленность:

$$\text{Приспособленность}(A_i) \leftarrow (1-\alpha)\text{Приспособленность}(A_i) \tag{.4}$$

$$+\alpha \frac{1}{\|A\|}(r + \gamma \sum_{A'_j \in A'} \text{Приспособленность}(A'_j)) \tag{.5}$$

Ничего не напоминает? Подсказка: зададим функцию G, включающую объединенную приспособленность (полезность) всех правил в текущем наборе действия A. Т.е. $G(A) = \sum_i \text{Приспособл.}(A_i)$. Уравнение (4) для функции G будет выглядеть так:

$$G(A) \leftarrow (1-\alpha)G(A) + \alpha \frac{1}{\|A\|}(r + \gamma G(A')).$$

Сравните с уравнением (2). В отличие от Samuel, обновление полезности в ZCS (правило приспособленности в ZCS), производится как в Q-обучении. При этом ZCS штрафует правила, которые не выбирают (т.е. правила из $M - A$). Пусть $B = M - A$, тогда приспособленность каждого правила $B_i \in B$ уменьшается согласно:

$$\text{Приспособленность}(B_i) \leftarrow \beta\text{Приспособленность}(B_i).$$

В результате эффект напоминает улетучивание (*evaporation*) в алгоритме муравьиной колонии. β принимает значения от 0 до 1 и не должно быть слишком большим. Таким образом, алгоритм обновления приспособленности в наборе подходящих правил будет следующим:

Алгоритм 125 Обновление приспособленности в классифицирующей системе нулевого уровня

1: M ←предыдущий набор подходящих действий
2: M' ←следующий набор подходящих действий {Не используется. Присутствует для единообразия с алгоритмом 131.}
3: A ←предыдущий набор действия
4: A' ←следующий набор действия
5: r ←вознаграждение за предыдущее действие
6: α ←скорость обучения
7: β ←константа улетучивания
8: γ ←константа для отсечения

9: **for** каждое $A_i \in A$ **do**
10: Приспособл.$(A_i) \leftarrow (1-\alpha)$Приспособл.$(A_i) + \alpha \frac{1}{\|A\|}(r + \gamma \sum_{A'_j \in A'}$ Приспособл.$(A'_j))$
11: **end for**
12: $B \leftarrow M - A$
13: **for** каждое $B_i \in B$ **do**
14: Приспособл.$(B_i) \leftarrow \beta$Приспособл.(B_i)
15: **end for**

Поскольку ZCS использует приспособленность как полезность, то когда в ZCS в результате устойчивого (*steady-state*) размножения создаются потомки, необходимо определить для них начальную приспособленность, иначе они никогда не будут выбраны в набор подходящих правил. Для этого от каждого родителя отнимается половина приспособленности и добавляется к каждому потомку (для того, чтобы суммарная приспособленность всех особей в популяции оставалась примерно одной и той же).

Теперь мы можем рассмотреть, как устроен в ZCS цикл верхнего уровня. Он состоит из двух частей:

1. Обновляем полезности (приспособленности) правил, тестируя их с использованием агента: в цикле создаем набор подходящих правил, выбираем из него действие, определяем набор действия, совершаем это действие и получаем вознаграждение, а затем обновляем значения приспособленности для правил в наборе подходящих правил. Значения приспособленностей обновляются согласно алгоритму 125.

2. Проделав вышеописанную операцию n раз, выполняем несколько шагов устойчивого размножения, в результате чего появляется несколько новых правил, которые добавляются в популяцию. Приспособленность созданных потомков инициализируется с использованием алгоритма 126.

Алгоритм 126 Перераспределение приспособленности в классифицирующей системе нулевого уровня

1: P_a, P_b ←родители
2: C_a, C_b ←потомки
3: $crossedover$ ←потомки получены в результате скрещивания?

4: **if** $crossedover = true$ **then**
5: Приспособл.(C_a), Приспособл.$(C_b) \leftarrow \frac{1}{4}$(Приспособл.$(P_a)$ + Приспособл.(P_b))
6: **else**
7: Приспособленность$(C_a) \leftarrow \frac{1}{2}$Приспособленность$(P_a)$
8: Приспособленность$(C_b) \leftarrow \frac{1}{2}$Приспособленность$(P_b)$
9: **end if**
10: Приспособленность$(P_a) \leftarrow \frac{1}{2}$Приспособленность$(P_a)$
11: Приспособленность$(P_b) \leftarrow \frac{1}{2}$Приспособленность$(P_b)$

Параметр n задает количество обновлений приспособленности, совершаемых перед каждой итерацией устойчивой эволюции. Если n слишком мало, то алгоритм осуществляет эволюцию на основании приблизительной информации, и потому становится нестабильным. Если n слишком велико, то алгоритм тратит лишнее время на вычисление точного значения приспособленности вместо того, чтобы продолжать дальнейший поиск решения. Обычно n необходимо выбирать достаточно большим.

Применение кроссинговера в ZCS — опционально. Конечно же, он используется во многих алгоритмах, но в ZCS его роль особенно важна, потому что кроссинговер часто обладает большой разрушающей способностью. Параметр p определяет частоту применения кроссинговера для создания потомков (как правило, не очень большую). При этом если кроссинговер выполняется, то необходимо перераспределить среднюю приспособленность по потомкам.

ZCS является первой встреченной нами метаэвристикой, которая не возвращает «наилучший найденный результат», вместо этого результатом является *вся популяция*, представляющая решение поставленной задачи.

Алгоритм XCS. Создав ZCS, Стюарт Уилсон разработал следующую версию алгоритма, названную **XCS**[14]. С момента создания XCS претерпел множество изменений, включая расширения Пьера Люки Ланци (*Pier Luca Lanzi*) и Мартина Бутца (*Martin Butz*). ZCS отличается от XCS по четырем основным пунктам:

- Выбор действия.
- Функция UpdateFitnesses.
- Функция SelectWithReplacement на эволюционном этапе алгоритма.
- Функция RedistributeFitnesses.

Существенная разница между алгоритмами заключается и в том, что в XCS не используется в качестве приспособленности Q-значение. Вместо этого для каждой особи дополнительно к значению приспособленности определяются отдельные величины **полезности** (***utility***) и **ошибки полезности**[15] (***utility error***). Мера полезности имеет примерно такой же смысл, как и Q-значение. Мера ошибки полезности приблизительно равна средней разности между текущей полезностью правила и текущей полезностью правил на *следующем* временном этапе. *Приблизительно*, потому что

[14] По всей видимости, XCS никак не расшифровывается! Наиболее ранняя версия алгоритма появилась в Stewart Wilson, 1995, Classifier fitness based on accuracy, *Evolutionary Computation*, 3(2), 149–175.

XCS — весьма сложен. Если взять алгоритмы, в которых я мог бы сделать несколько ошибок (а такие несомненно имеются!), он обязательно был бы среди них. Более подробное описание алгоритма приведено в Martin Butz and Stewart Wilson, 2001, Analgorithmic description of XCS, in Advances in Learning Classifier Systems, volume 1996/2001, pages 267–274, Springer. Значительная доля кода в данном пособии к лекциям создана с использованием информации из этой статьи. Заметьте, что в моей версии присутствуют некоторые упрощающие синтаксические изменения (к примеру, нет «прогнозного массива»(*prediction array*)), но работать все должно так же (постучим по дереву).

[15] В XCS они называются соответственно **прогнозом** (***prediction***) и **ошибкой прогноза** (***prediction error***) правила.

Алгоритм 127 Классифицирующая система нулевого уровня (ZCS)

```
popsize ←требуемый размер популяции
f ←значение приспособленности для особей начальной популяции {Может быть любым. Например, 1.}
n ←количество запусков агента в одной итерации эволюции {Пусть будет большим.}
p ←вероятность выполнения кроссинговера {Пусть будет небольшой.}

P ← {} {Здесь P как и обычно обозначает популяцию, а не распределение P(s'|s,a) из Q-обучения.}
for n раз do
  P ← P ∪ {новая случайная особь с приспособленностью f}
end for
repeat
  for n раз do
    s ←начальное состояние агента
    r ←0
    M ←{ }
    A ←{ }
    repeat
      M' ⊆ P ←набор подходящих правил для состояния s {Т.е. все P_i ∈ P, покрывающие s}
      if M' пусто then
        M' ← M'∪ {новая случайная особь, покрывающая s, с приспособленностью равной средней приспособленности в популяции}
        Особь P_c ←SelectForDeath(P)
        P ←(P − {P_c}) ∪ M'
      end if
      a ← лучшее действие из M' {Действие победителя по SelectWithReplacement(M')}
      A' ⊆ M' ←набор действия для действия {Т.е. все M'_j ∈ M, действие которых есть a}
      UpdateFitness с использованием M, M', A, A' и r
      Агент выполняет действие a, получает новое вознаграждение r и переходит в новое состояние s
      M ← M'
      A ← A'
    until не закончилась жизнь агента
    UpdateFitness с использованием M, M', A, {} и r {Последняя итерация. Заметим, что M = M' и A' = {}}
  end for
  Родитель P_a ← SelectWithReplacement(P) {А теперь начинаем размножение}
  Родитель P_b ← SelectWithReplacement(P)
  Потомок C_a ← Copy(P_a)
  Потомок C_b ← Copy(P_b)
  if p ≥ случайное число равномерно распределенное от 0,0 до 1,0 then
    C_a, C_b ← Crossover(C_a, C_b)
    RedistributeFitnesses(P_a, P_b, C_a, C_b, true)
  else
    RedistributeFitnesses(P_a, P_b, C_a, C_b, false)
  end if
  C_a ← Mutate(C_a)
  C_b ← Mutate(C_b)
  Особь P_d ← SelectForDeath(P)
  Особь P_e ← SelectForDeath(P) {P_d не должно быть равно P_e}
  P ← (P − {P_d, P_e}) ∪ {C_a, C_b}
until не закончилось время
return P
```

ошибка полезности (также как и полезность и приспособленность) использует фокус с $1-\alpha$ чтобы учитывать новые результаты, так чтобы недавние ошибки полезности имели больший вес, чем более старые. В идеальном случае с ростом эффективности правил их полезность становится более точной и ошибка полезности должна снижаться. Приспособленность вычисляется на основе нормализованной ошибки полезности, учитывая ошибки других членов в наборе действия: чем меньше ошибка, тем лучше приспособленность.

Выбор действия. Для выбора действия из набора подходящих действий M в XCS сначала определяется «наилучшее»действие из M. Для этого сначала объединяются правила из M, приводящие к одному действию, а затем их полезности складываются с весами, определяемыми приспособленностями правил (более приспособленные правила вносят больший вклад в полезность действия).

Алгоритм 128 Взвешенная по приспособленности полезность действия в XCS

1: M ←набор подходящих действий
2: N_i ←действие

3: $R \subseteq M$ ←все правила из M, содержащие в головной части N_i
4: **if** $\sum_{r \in R} \text{Приспособленность}(r) \neq 0$ **then**
5: **return** $\frac{\sum_{r \in R} (\text{Полезность}(r) \times \text{Приспособленность}(r))}{\sum_{r \in R} \text{Приспособленность}(r)}$
6: **else**
7: **return** 0
8: **end if**

Теперь можем определить, какое из действий является «наилучшим»:

Алгоритм 129 Определение лучшего действия в XCS

1: M ←набор подходящих действий
2: N ←все действия, присутвующие в головных частях правил в M

3: $Best \leftarrow \square$
4: $bestc$ ←0
5: **for** каждое действие $N_i \in N$ **do**
6: c ←Взвешенная по приспособленности полезность действия N_i в XCS
7: **if** $Best = \square$ или $c > bestc$ **then**
8: $Best$ ←N_i
9: $bestc$ ←c
10: **end if**
11: **end for**
12: **return** $Best$

Далее можем выбрать либо случайное (с вероятностью ϵ), либо наше «наилучшее»действие. Такой подход должен быть уже знакомым: это все тот же ϵ-жадный выбор действия, такой же, как и в Q-обучении. Его использование для XCS впервые предложил Пьер Люка Ланци[16].

Обновление приспособленности. При тестировании необходимо обновлять не только приспособленность, но все три элемента: полезность, ошибку полезности и приспособленность. Полезность обновляется в стиле Q-значения:

$$\text{Полезность}(A_i) \leftarrow (1-\alpha) \times \text{Полезность}(A_i) + \alpha(r + \gamma b)$$

Что обозначает b? Это взвешенная по приспособленности полезность (алгоритм 128) лучшего действия (алгоритм 129) *на следующей итерации.* Т.е. необходимо отложить обновление приспособленности для данной итерации, до тех пор, пока не наступит следующая. И, как и прежде, сравните формулу с уравнением (2).

Ошибка полезности обновляется аналогично, но с использованием новой ошибки, которая подсчитывается как разность полезности и лучшей полезности для *следующего* набора действия:

$$\text{ОшибкаПолезности}(A_i) \leftarrow (1-\alpha) \times \text{ОшибкаПолезности}(A_i) + \alpha \| b - \text{Полезность}(A_i) \|$$

[16] Pier Luca Lanzi, 1999, Ananalysis of generalization in the XCS classifier system, *Evolutionary Computation*, 7(2), 125–149

Алгоритм 130 Выбор действия в XCS

1: M ←набор подходящих действий
2: ϵ ←вероятность исследования $\{0 \geq \epsilon \geq 1\}$

3: N ←все действия, присутвующие в головных частях правил в M
4: $bestc$ ←0
5: **if** $\epsilon \geq$ случайное число равномерно распределенное в диапазон от 0,0 до 1,0, включая границы **then**
6: **return** случайно выбранный по равномерному закону элемент из M
7: **else**
8: **return** действие, полученное в результате работы алгоритма Определение лучшего действия в XCS для данных M и N
9: **end if**

Для вычисления приспособленности необходимо сначала преобразовать ошибку в «точность»a_i. Если ошибка больше некоторого малого e, то точность полагается идеальной, и равняется 1. В противном случае точность a_i определяется как $\delta(\frac{e}{\text{ОшибкаПолезности}(A_i)})^{\beta}$. В заключение точность включается в формулу для обновления приспособленности:

$$\text{Приспособленность}(A_i) \leftarrow (1-\alpha) \times \text{Приспособленность}(A_i) + \alpha \frac{a_i}{\sum_{A_j \in A} a_j}$$

Улетучивание не используется. Вот алгоритм обновления целиком:

Перераспределение приспособленности. Помимо приспособленности в XCS необходимо перераспределять полезность и ошибку полезности. В отличие от алгоритма ZCS, где производится перераспределение приспособленности, полученной от родителей, в XCS просто обрезается значение приспособленности потомков. В частности:

Выполнение SelectWithReplacement. SelectWithReplacement выполняется не над всей популяцией, как в ZCS, а только над набором действия. Т.е. строки 31 и 32 из алгоритма 127 будут выглядеть так:

Родитель $P_a \leftarrow \text{SelectWithReplacement}(A)$
Родитель $P_b \leftarrow \text{SelectWithReplacement}(A)$

Другие приемы. В добавление к этому базовому алгоритму в XCS используются и другие приемы. Во-первых, применяется понятие **микроклассификатор** (***microclassifiers***). XCS рассматривает особь не как одно правило, а как множество одинаковых правил. Для этого вводится дополнительная переменная **count**, которая обозначает сколько «копий»правила включает в себя особь. Эта переменная используется во время обновления приспособленности (алгоритм 131) в последней строке, задействуя «вложенные»правила:

$$\text{Приспособленность}(A_i) \leftarrow (1-\alpha)\text{Приспособленность}(A_i) + \alpha \frac{a_i \times Count(A_i)}{\sum_{A_j \in A} a_j \times Count(A_j)}$$

Переменная count также учитывается при создании и удалении правил. Во время создания нового правила, сначала проверяется, совпадает ли оно с уже существующим. Если да, то count для существующего правила увеличивается на 1, а новое правило в популяцию не добавляется. При удалении правила, если count больше 1, то значение переменной уменьшается, а правило остается в популяции. Правило удаляется, только если счетчик равен 1. Заметьте, что это может привести к небольшому измнению размера популяции. По сути данный прием является разновидностью нишинга, поскольку предотвращает доминирование правила, при его большой распространенности в популяции. Однако он может привести и к тому, что от распространенного правила тяжело избавиться.

В силу произвольности начальных значений приспособленности и полезности для новых правил в XCS, приводящей к их отставанию от существующих, имеется механизм, позволяющий повысить быстро повысить полезность и ошибку полезности новых правил. Для этого для каждого правила используется счетчик **опыта** (***experience***), который увеличивается каждый раз, когда правило

Алгоритм 131 Обновление приспособленности в XCS

M ←предыдущий набор подходящих действий {Отметим, что на последней итерации верхнего цикла ZCS $M = M'$}
M' ←следующий набор подходящих действий
A ←предыдущий набор действия
A' ←следующий набор действия {Не используется. Присутствует для единообразия с алгоритмом 125.}
r ←вознаграждение, полученное для предыдущего действия
e ←наибольшая ошибка полезности, при которой все еще сохраняется максимальная приспособленность
α ←скорость обучения {$0 < \alpha < 1$. Пусть будет небольшой.}
β ←параметр подстройки приспособленности {$\beta > 1$}
γ ←константа для отсечения {$0 < \alpha < 1$. Хорошо подходит 0,5.}
δ ←параметр подстройки приспособленности {Предположительно $0 \geq \delta \geq 1$. На мой взгляд, лучше взять 1.}

n ←действие, полученное в результате работы алгоритма Определение лучшего действия в XCS для данного M
b ←результат алгоритма Взвешенная по приспособленности полезность действия n в XCS
$\overrightarrow{a} \leftarrow \langle a_1, \ldots, a_{\|A\|} \rangle$ вектор значений точности для каждого правила в A
for каждое правило $A_i \in A$ **do**
  $\text{Полезность}(A_i) \leftarrow (1-\alpha) \times \text{Полезность}(A_i) + \alpha(r + \gamma b)$
  $\text{ОшибкаПолезности}(A_i) \leftarrow (1-\alpha) \times \text{ОшибкаПолезности}(A_i) + \alpha\|b - \text{Полезность}(A_i)\|$
  if $\text{ОшибкаПолезности}(A_i) > e$ **then**
    $a_i \leftarrow \delta(\frac{e}{\text{ОшибкаПолезности}(A_i)})^{\beta}$ {Преобразование ошибки в «точность»(большая ошибка—низкая точность)}
  else
    $a_i \leftarrow 1$ {Понятия не имею, почему не $a_i \leftarrow \delta$}
  end if
end for
for каждое правило $A_i \in A$ **do**
  $\text{Приспособл.}(A_i) \leftarrow (1-\alpha) \times \text{Приспособл.}(A_i) + \alpha \frac{a_i}{\sum_{A_j \in A} a_j}$ {Нормализация точностей}
end for

Алгоритм 132 Перераспределение приспособленности в XCS

P_a, P_b ←родители
C_a, C_b ←потомки
v ←коэффициент обрезания приспособленности
crossedover ←показывает, получены ли потомки в результате скрещивания?

if *crossedover* =true **then**
  $\text{Приспособл.}(C_a), \text{Приспособл.}(C_b) \leftarrow v\frac{1}{4}(\text{Приспособл.}(P_a) + \text{Приспособл.}(P_b))$
  $\text{Полезность}(C_a), \text{Полезность}(C_b) \leftarrow v\frac{1}{4}(\text{Полезность}(P_a) + \text{Полезность}(P_b))$
  $\text{ОшибкаПолезн.}(C_a), \text{ОшибкаПолезн.}(C_b) \leftarrow v\frac{1}{4}(\text{ОшибкаПолезн.}(P_a) + \text{ОшибкаПолезн.}(P_b))$
else
  $\text{Приспособленность}(C_a) \leftarrow v\frac{1}{2}\text{Приспособленность}(P_a)$
  $\text{Приспособленность}(C_b) \leftarrow v\frac{1}{2}\text{Приспособленность}(P_b)$
  $\text{Полезность}(C_a) \leftarrow v\frac{1}{2}\text{Полезность}(P_a)$
  $\text{Полезность}(C_b) \leftarrow v\frac{1}{2}\text{Полезность}(P_b)$
  $\text{ОшибкаПолезности}(C_a) \leftarrow v\frac{1}{2}\text{ОшибкаПолезности}(P_a)$
  $\text{ОшибкаПолезности}(C_b) \leftarrow v\frac{1}{2}\text{ОшибкаПолезности}(P_b)$
end if

попадает в набор действия. Скорость обучения постепенно понижается до тех, пока счетчик опыта не превысит $1/\alpha$, после чего скорость обучения остается фиксированной и равна α.

Учитывая вышесказанное можем расширить алгоритм обновления приспособленности в XCS (алгоритм 131), чтобы включить рассмотренные приемы:

Алгоритм 133 Обновление приспособленности в XCS (расширенное)

1: M ←предыдущий набор подходящих действий {Отметим, что на последней итерации верхнего цикла ZCS $M = M'$}
2: M' ←следующий набор подходящих действий
3: A ←предыдущий набор действия
4: A' ←следующий набор действия {Не используется. Присутствует для единообразия с алгоритмом 125.}
5: r ←вознаграждение, полученное для предыдущего действия
6: e ←наибольшая ошибка полезности, при которой все еще сохраняется максимальная приспособленность
7: α ←скорость обучения {$0 < \alpha < 1$. Пусть будет небольшой.}
8: β ←параметр подстройки приспособленности {$\beta > 1$}
9: γ ←константа для отсечения {$0 < \alpha < 1$. Хорошо подходит 0,5.}
10: δ ←параметр подстройки приспособленности {Предположительно $0 \geq \delta \geq 1$. На мой взгляд, лучше взять 1.}

11: n ←действие, полученное в результате работы алгоритма Определение лучшего действия в XCS для данного M
12: b ←результат алгоритма Взвешенная по приспособленности полезность действия n в XCS
13: $\overrightarrow{a} \leftarrow \langle a_1, \ldots, a_{\|A\|} \rangle$ вектор значений точности для каждого правила в A
14: **for** каждое правило $A_i \in A$ **do**
15: $\quad$ Опыт$(A_i) \leftarrow$ Опыт$(A_i) + 1$
16: $\quad \alpha' \leftarrow \max(\frac{1}{\text{Опыт}(A_i)}, \alpha)$
17: $\quad$ Полезность$(A_i) \leftarrow (1 - \alpha') \times$ Полезность$(A_i) + \alpha'(r + \gamma b)$
18: $\quad$ ОшибкаПолезности$(A_i) \leftarrow (1 - \alpha') \times$ ОшибкаПолезности$(A_i) + \alpha' \|b -$ Полезность$(A_i)\|$
19: $\quad$ **if** ОшибкаПолезности$(A_i) > e$ **then**
20: $\quad\quad a_i \leftarrow \delta(\frac{e}{\text{ОшибкаПолезности}(A_i)})^{\beta}$ {Преобразование ошибки в «точность» (большая ошибка — низкая точность)}
21: $\quad$ **else**
22: $\quad\quad a_i \leftarrow 1$ {Понятия не имею, почему не $a_i \leftarrow \delta$}
23: $\quad$ **end if**
24: **end for**
25: **for** каждое правило $A_i \in A$ **do**
26: $\quad$ Приспособл.$(A_i) \leftarrow (1-\alpha) \times$ Приспособл.$(A_i) + \alpha \frac{a_i \times Count(A_i)}{\sum_{A_j \in A} a_j \times Count(A_i)}$ {Нормализация точностей}
27: **end for**

Наибольшие изменения в строках 15, 16 и 26.

И, наконец, в XCS имеются опциональные процедуры **категоризации** (***subsumption***): ищется *категоризованное* правило, покрываемые состояния которого полностью покрываются некоторым другим достаточно приспособленным и старым правилом. Основная цель, как и раньше, — усиление разнообразия и избавление от избыточности. Категоризацию можно проводить в двух местах. Первое — создание нового правила, при этом XCS может и не включить данное правило, если оно категоризовано другим правилом, а увеличить переменную count категоризующего правила на единицу. Второе — после формирования набора действия A. XCS может проверить A на наличие категоризующих правил и удалить из популяции категоризованные.

10.5 Это генетическое программирование?

В определенном и очень важном смысле правила поведения являются **программами**, управляющими агентами. Эти программы состоят из правил вида *если*→*то*, где сторона *если* задается текущим состоянием мира. Даже если мы избавимся от управляющих структур, то что останется будет все равно гораздо сложнее, чем львиная доля «программ», получаемых с использованием

генетического программирования с деревьями и машинными кодами (см. Разделы 4.3 и 4.4). Но достаточно ли этого факта, чтобы говорить о «программировании»?

Во многих средах необходимо больше информации, чем просто описание состояния мира, чтобы решить, что делать. Требуется **память**, в которой будет храниться выборочная информация из **истории** уже происшедших событий. Такая память часто называется **внутренним состоянием** (***internal state***) агента (противопоставляя состоянию мира, или **внешнему состоянию** (***external state***)).

Рассмотрим рис. 66. Робот начинает движение в комнате A и ему требуется выйти через дверь внизу. Необходимо разработать правила, согласно которым робот прошел бы в комнату C, переключить тумблер, открывающий дверь, вернулся в комнату A и вышел через дверь. Правила поведения могли бы быть такими:

В комнате A, дверь закрыта → идти в комнату B
В комнате B → идти в комнату C
В комнате C, тумблер выключен → включить тумблер
В комнате C, тумблер включен → идти в комнату B
В комнате B → **гм ...**
В комнате A, дверь открыта → выйти через дверь

Загвоздка в том, что для комнаты B уже есть правило! Идти в комнату C. Поэтому для комнаты B нужно *два* правила: если я направлялся, чтобы включить тумблер, то идти в комнату C, а если я шел к двери, то идти в комнату A. Сложность в том, что в комнате B ничего не происходит, и нет информации о внешнем состоянии, которая помогла бы различить условия правил. Две ситуации в комнате B **синонимичны** (***aliased***): они требуют различных действий, однако внешнее состояние у них совпадает.

Нужна некоторая память, которая хранила бы, в частности, включили мы тумблер или нет. Пусть у агента появился один бит памяти, в котором в начальный момент времени записан 0. Тогда можем сформировать следующие правила поведения:

В комнате A, дверь закрыта → идти в комнату B
В комнате B и бит памяти содержит 0 → идти в комнату C
В комнате C, тумблер выключен → включить тумблер, установить бит памяти в 1
В комнате C, тумблер включен → идти в комнату B
В комнате B и бит памяти содержит 1 → идти в комнату A
В комнате A, дверь открыта → выйти через дверь

Проблема решена! Главная особенность в том, что добавив один единственный бит памяти, мы **вдвое увеличиваем потенциальное пространство состояний**. Таким образом, один бит — это очень даже неплохо, а несколько битов позволяют существенно повысить сложность мира. Подобные методы в настоящее время вполне можно считать передовыми. Лично я считаю, что методы оптимизации правил поведения наиболее успешные представители современного «генетического программирования». Тем не менее, настоящее автоматическое программирование еще очень далеко. Ваша работа в безопасности.