4.5 Множества правил

Множество является коллекцией объектов, возможно пустой, в которой все объекты различны. Множества могут использоваться для самых различных целей, среди которых значимое место занимают множества *правил* (*rules*), которые могут представлять компьютерные программы разного рода (например, для управления роботом в модельной среде), или же определять косвенное кодирование, задающее правила роста для структуры графа, начиная с некоторой простой стартовой структуры.

Правила во множестве правил обычно представлены в виде *если* $\rightarrow$*то*. Часть с *если* обычно называется **телом** (***body***) правила, а часть с *то* — **головой** (***head***) правила. Часто используются две разновидности множеств правил: **состояние-действие** (***state-action***) и **продукционные** (***production***). Правила состояние-действие используют для выполнения действия (часть *то*) при наступлении в мире определенного события или ситуации (часть *если*). Например, состояние сенсоров робота может привести к работе правила, по которому робот повернет налево. Отличие продукционных правил заключается в том, что части *то* некоторых правил активируют части *если* других правил. Например, выполнение правила $a \rightarrow b$ может привести к выполнению правила $b \rightarrow c$. Продукционные правила часто используют в косвенных кодировках для задания параметров роста структуры графа. Взаимодействие между правилами в продукционных множествах означает, что эти правила являются большим, чем нечто напоминающее структуру ориентированного графа (в зависимости от используемой кодировки).

Первый вопрос: какую структуру данных можно использовать для хранения множества объектов? Можно взять вектор переменной длины, как список. Или можно использовать хеш-таблицу, в которой ключами являются элементы множества, а значениями — нечто произвольное. По моему опыту в большинстве случаев для множеств используются списки.

Простейшим ограничением для множества является свойство уникальности: часто необходимо, чтобы создание множеств, их скрещивание и мутация не приводили к появлению одинаковых правил. Если это не отслеживается в операторах мутации и кроссинговера, то необходимо производить проверку на наличие особей-дубликатов после соответствующих операций. Вот простая процедура:

Алгоритм 61 Удаление особей-дубликатов

1: $\overrightarrow{v} \leftarrow$набор элементов, преобразованный в вектор $\langle v_1, v_2, \ldots, v_l \rangle$

2: $h \leftarrow \{\ \}$ $\{h$ представлено хеш-таблицей, так быстрее.$\}$
3: $l' \leftarrow l$
4: **for** i от l до 1 **do**
5: **if** $v_i \in h$ **then**
6: Swap v_i и $v_{l'}$
7: $l' \leftarrow l' - 1$
8: **else**
9: $h \leftarrow h \cup \{v_i\}$
10: **end if**
11: **end for**
12: $v \leftarrow$пустой вектор $\langle v'_1, v'_2, \ldots, v'_l \rangle$
13: **for** i от 1 до l' **do**
14: $v'_i \leftarrow v_i$
15: **end for**
16: **return** $\overrightarrow{v'}$ преобразованный обратно в набор

Заметьте, что алгоритм меняет порядок в исходном списке $\overrightarrow{v}$. Реализовать h в виде хеш-таблицы легко: для добавления элемента в h необходимо просто добавить его в качестве ключа в эту таблицу (значением может быть что угодно, например, сам элемент). Чтобы проверить условие $v_l \in h$ необходимо просто убедиться, что ключ v_l уже присутствует в таблице. Проще пареной репы.

[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_4_5.pdf`

4.5.1 Правила состояние-действие

Правила состояние-действие обычно используются для формирования небольших программ, называемых **правилами действия** (***policies***), предписывающих агенту (робот, игрок и т.д.) что делать в любой ситуации, в которой этот агент может оказаться. Вы в пригороде? Садитесь на поезд. Вы в поезде? Едьте до причала. И так далее. Правила состояние-действие записывают по-разному, но общепринятой является форма $a \wedge b \wedge \cdots \wedge y \rightarrow z$, где $a, b, \ldots, y$ — *описания состояния* (*state descriptions*), а z — *действие* (*action*) или *класс* (*class*). Описание состояния представляет некоторый признак мира, который может быть истинным или ложным. Действие — это то, что необходимо сделать, если признак истинен. Например, робот может иметь правила напободие следующего:

$$\text{Показание левого сонара} > 3,2 \wedge \text{Показание правого сонара} \leq 5,0 \rightarrow \text{Повернуть налево на } 50^\circ$$

Множество правил можно проверить, если поместить моделируемого робота в среду и управлять им на основе этих правил. Каждый раз при получении сенсорной информации, робот определяет правила, тела которых истинны для данных показаний сенсоров. Набор таких правил называет **множеством подходящих правил** (***match set***). После этого робот решает, что делать на основании действий в голове отобранных правил (таких как «повернуть налево на 50°»).

Можно считать, что тело правила описывает некоторую область в **пространстве состояний** (***state space***) робота, а голова — что делать в этих областях. В случае вышеприведенного правила, его тело накрывает область, значения в которой больше 3,2 по одному измерению и $\leq$ 5,0 по другому измерению, не учитывая другие измерения.

Здесь есть два интересных момента. Первый, что если к текущему состоянию не подходит ни одно правило? Эта ситуация известна как **недоопределенность** (***under-specification***) пространства состояний: в пространстве есть дыры, не накрываемые ни одним правилом. В таких случаях часто используют **правило по умолчанию** (***default rule***), которое срабатывает тогда, когда не подходит ни одно имеющееся правило. Более интересной является проблема, когда текущему состоянию соответствует более одного правила, причем эти правила содержат в головной части противоречивые действия (к примеру, одно говорит «поворачивай налево», а другое — «поворачивай направо»). Такая ситуация называется **переопределенностью** (***over-specification***) пространства состояний. Для ее разрешения необходимо применить некоторую **арбитражную схему** (***arbitration scheme***). В большинстве случаев, если имеется много правил, то проводится **голосование** (***vote***). Другим способом является выбор случайного правила. И, да, пространство состояний может быть одновременно недо- и переопределенным.

Использование множества правил часто вносит коррективы в процесс определения приспособленности. В частности, по мере передвижения агента можно присваивать приспособленность не только ему, но также и правилам из *его* множества правил. В крайнем случае можно разделить правила на те, которые были активированы в процессе запуска особи, и на те, которые ни разу не сработали (и поэтому не имеют отношения к тому отличному или отвратительному результату, которые был получен). Можно вознаграждать или штрафовать только те правила, которые работали. Либо если после поворота Налево робот получил удар электрическим током, то можно оштрафовать последовательность правил, активация которых привела к этому удару, но не штрафовать последующие правила. При этом можно быть более избирательным при мутации или удалении (при кроссинговере) правил с наибольшим штрафом.

Метаэвристики, разработанные для оптимизации правил действия с использованием правил состояние-действие: **Мичиганский подход к системам обучающихся классификаторов** (***Michigan-Approach Learning Classifier Systems***) и **Системы правил по Питт-подходу** (***Pitt-Approach Rule Systems***), — обсуждаются в **Главе 10**.

4.5.2 Продукционные правила

Продукционные правила похожи на правила состояние-действие, но отличаются тем, что «действия»одного правила могут привести к истинности состояний других правил. Продукционные правила являются своего рода перевертышами, их часто можно представить так: $a \rightarrow b \wedge c \wedge \cdots \wedge z$, ввиду того, что продукционные правила активируются (как правило) одним событием (которое обычно включается другим правилом), а это приводит к последующей активации множества других правил. Во многих случаях продукционные правила применяют для представления *модульных* (*modular*) косвенных кодировок, описывающих большие сложные решения с большим количеством повторяющихся элементов, и при этом требующих компактного пространства правил, в котором возможен

Рис. .32: Схемы растений созданные с помощью системы Линденмайера

эффективный поиск. Это, естественно, подразумевает, что хорошие решения будут *иметь* повторяющиеся элементы, что в свою очередь существенно зависит от рассматриваемой задачи.

Обычно символы, присутствующие в голове (правой части) продукционных правил, бывают двух видов: **нетерминальные** (***nonterminal***) символы, которые также могут появляться в теле правила, и **терминальные** (***terminal***) символы, как правило отсутствующие в теле правила. Терминальные символы не могут быть изменены. Заметим, что для большинства продукционных систем набор правил зафиксирован, по одному на каждый терминальный символ.

Одним из первых примеров применения эволюционных вычислений для продукционных правил был является исследование Хироаки Китано (*Hiroaki Kitano*) для поиска оптимальной структуры графа для рекуррентных нейронных сетей[1]. Предположим, что необходимо найти структуру ориентированного графа с 8 непомеченными вершинами. Набор правил может выглядеть следующим образом (числа являются терминальными символами):

$$a \to \begin{bmatrix} b & c \\ c & d \end{bmatrix} \qquad b \to \begin{bmatrix} 1 & 0 \\ d & c \end{bmatrix} \qquad c \to \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \qquad d \to \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \qquad 0 \to \begin{bmatrix} 0 & 0 \\ 0 & c \end{bmatrix} \qquad 1 \to \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Это косвенная кодировка для структуры графа, хотите верьте, хотите — нет. Начинаем с матрицы $[a]$ размером 1×1. Затем применяем правило для a, которое расширяет матрицу $[a]$ в $\begin{bmatrix} b & c \\ c & d \end{bmatrix}$. Далее применяем правила для каждого элемента в *полученной* матрице, расширяя их до элементов 2×2, что приводит к матрице $\begin{bmatrix} 1 & 0 & 1 & 1 \\ d & c & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$. Потом расширяем ее до $\begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$.

После этого нетерминальные символы закончились. (Поскольку для терминальных символов также были введены «правила расширения» , то можно было расширять начальную матрицу до тех пор, не закончатся нетерминальные символы, либо на протяжении фиксированного количества шагов.) Таким образом получена матрица смежности для графа, в которой 1 в ячейке $\langle i, j \rangle$ обозначает, что «есть ребро от вершины i к вершине j», а 0 обозначает, что «ребра нет». Рисовать этот ужасный граф я не буду!

Более свежий пример применения продукционных правил для косвенной кодировки — это поиск оптимальной **системы Линденмайера** (***Lindenmayer Systems***, **L-Systems**). Она представлена множеством продукционных правил, генерирующих символьную строку. Эта строка затем интерпретирует как небольшая компьютерная программа для создания некоторого конечного объекта, такого как растение или дерево, фрактал, шаблон, либо некая машина. L-системы были популяризованы биологом Аристидом Линденмайером (*Aristid Lindenmayer*), разработавшим их для описания схемы роста растений[2].

[1] Эта статья является одной из пионерских для косвенных кодировок. Hiroaki Kitano, 1990, Designing neural networks using a genetic algorithm with a graph generation system, Complex Systems, 4, 461–476.

[2] Пржемыслав Прусинкевич (*Przemyslaw Prusinkiewicz*) и Аристид Линденмайер написали замечательную книгу об L-системах: Przemyslaw Prusinkiewicz and Aristid Lindenmayer, 1990, The Algorithmic Beauty of Plants, Springer-Verlag. Она уже не издается, однако ее текст в открытом доступе по адресу: `http://algorithmicbotany.org/papers/#abop`

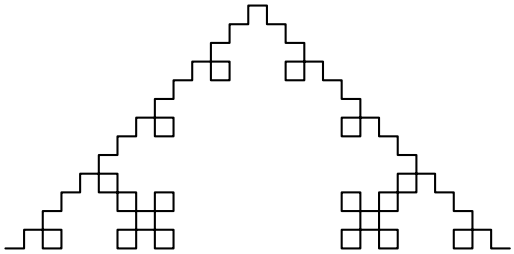

Рис. .33: Квадратичная кривая Коха

Простым примером L-системы является система для генерации фрактальной **кривой Коха** (***Koch Curve***). Система правил включает единственное правило $F \to F + F - F - F + F$. Оно работает следующим образом: начинаем с одного символа F. Применяя данное правило расширяем этот символ до $F + F - F - F + F$. Расширяя каждый символ F получаем:

$$F + F - F - F + F + F + F - F - F + F - F + F - F - F + F - F + F - F - F + F + F + F - F - F + F$$

Снова применяем расширение:

$$\begin{gathered}
F + F - F - F + F + F + F - F - F + F - F + F - F - F + F - F + F - F - F + F + F + F - F - F + F + \\
F + F - F - F + F + F + F - F - F + F - F + F - F - F + F - F + F - F - F + F + F + F - F - F + F - \\
F + F - F - F + F + F + F - F - F + F - F + F - F - F + F - F + F - F - F + F + F + F - F - F + F - \\
F + F - F - F + F + F + F - F - F + F - F + F - F - F + F - F + F - F - F + F + F + F - F - F + F + \\
F + F - F - F + F + F + F - F - F + F - F + F - F - F + F - F + F - F - F + F + F + F - F - F + F
\end{gathered}$$

Символы $+$ и $-$ терминальные. Что с этой строкой делать? Если читать F как «нарисовать линию вперед», а $+$ и $-$ как «повернуть налево» и «повернуть направо» соответственно, то получится кривая Коха, как на рис. 33. Последующие расширения создают еще более сложные кривые.

В реальности эти правила могут быть весьма сложными. На рис. 34 показан пример L-системы, используемой биологами для описания схемы ветвления красной водоросли *Bostrychia radicans*[3].

$$\begin{array}{lcl}
O & \to & FGD \\
D & \to & G[+++FGFGRG][-GF]GFGA \\
A & \to & FGFGFGFG[+++FGR][-GF]GFGB \\
B & \to & FGFGFGFG[+++FGR][-GF]GFGC \\
C & \to & FGFGFGFG[+++FGR][-GF]GFGK \\
R & \to & FG[+FGFGU]GFGFGE \\
E & \to & [-FGFGX]GFGFGH \\
H & \to & [+FGFGW]GFGFGZFG \\
K & \to & FGFGFG[+++FGR][-FGA]GFGL \\
L & \to & FGFGFG[+++FGR][-GF]GFGP \\
P & \to & FGFGFG[+++FGR][-GF]GFGQ \\
Q & \to & FGFGFGT \\
T & \to & FGFGFG[+++FGR][+FGA]GFGA \\
U & \to & [+FGFGF]GFG \\
X & \to & [-FGFGF]GFG \\
W & \to & [+FGFGF]GFG \\
Z & \to & [-FGFGF]GFG
\end{array}$$

Рис. 34. Еще одна L-система

Интересный способ использования L-систем в эволюционных вычислениях для поиска оригинального дизайна столов и стульев предложил Грег Хорнби (*Greg Hornby*), который также применил

[3] Из статьи Ligia Collado-Vides, Guillermo Gomez-Alcaraz, Gerardo Rivas-Lechuga, and Vinicio Gomez-Gutierrez, 1997, Simulation of the clonal growth of Bostrychia radicans (Ceramiales-Rhodophyta) using Lindenmayer systems, Biosystems, 42(1), 19–27.

L-системы совместно с кодировкой ребер для поиска форм тел животных и графовых структур, напоминающих конечные автоматы[4]. Множество правил L-системы было преобразовано в строку, которая затем была интерпретирована как последовательность инструкций для кодировки ребер (Удвоение, Разделение и т.д.) для формирования графа.

4.5.3 Инициализация

Построение множества правил в основном зависит от количества требуемых элементов, также как и структура ориентированного графа, и их создания. Для начала определим мощность n множества правил, используя некоторое распределение (вполне возможно, отлично подойдет геометрическое распределение, алгоритм 46), а затем создадим множество правил из n случайно сгенерированных элементов.

При создании продукционных правил необходимо учитывать некоторые дополнительные ограничения. В частности, различные **символы**, которые присутствуют в головной части правил, должны соответствовать телу этих правил. В противном случае не будет логической связи между событием, которое запускается правилом, со следующим активированным правилом. По аналогии, возможно, нежелательно, чтобы у двух правил совпадали тела, т.е. возникала ситуация наподобие: $a \rightarrow b, c$ и $a \rightarrow d, e, f$. Какое из этих правил должно сработать? Арбитражная схема не очень хорошо подходит для продукционных правил, в отличие от правил состояние-действие, если только продукционные правила не срабатывают с некоторой вероятностью.

В некоторых системах продукционных правил количество правил ограничено размером нетерминального множества. В других системах правил количество символов может изменяться, и в этом случае необходимо убедиться, что все символы в головной части правил встречаются в телах каких-либо других правил. Очевидно, что правила, символы из тел которых не встречаются в головной части ни одного правила, являются своего рода сиротами (это может случиться и для фиксированного множества нетерминальных символов). Кроме этого можно дополнительно разрешать или запрещать рекурсию, т.е. сможет ли правило A включить правило B, которое затем снова включит A? К примеру, представим, что буквы — это расширяемые символы, а цифры — терминальные. Тогда вот множество правил, которое содержит потенциальные проблемы:

a	$\rightarrow$	b, c	*Нет правила для c! Что делать, если произойдет событие c?*
b	$\rightarrow$	$d, 0$	
a	$\rightarrow$	d, b	*Хм, мы точно хотим дублировать тела правил?*
d	$\rightarrow$	b	*А в данном множестве правил разрешена рекурсия?*
e	$\rightarrow$	1	*Это правило никогда не включится! Оно вообще не нужно!*

Во время инициализации такие ситуации нужно обрабатывать. Можно сначала сгенерировать правила, а затем попытаться их «починить». Либо можно создать n нетерминальных символов и создать для них правила. Далее приведен возможный алгоритм. Он не является универсальным, но позволяет выбирать требуется ли создание рекурсивных правил, а также разрешать или нет появление изолированных (*disconnected*) правил (т.е. таких правил, которые никогда не будут активированы). Алгоритм, скорее, призван показать общую идею, но если вы собираетесь его использовать, то, вполне вероятно, его придется серьезно переделать под вашу задачу.

4.5.4 Мутация

Мутация множеств часто похожа на мутацию списков. Т.е. как правило ставятся две подзадачи: изменение мощности множества правил (если это разрешено) и изменение правил множества. И здесь можно дать такой же совет, как и для мутации списков. Например, один из способов изменить мощность множества заключается в генерации небольшого числа из геометрического распределения, чтобы добавить или удалить такое число правил из множества (удаляемые правила можно

[4] Грег сделал, на мой взгляд, *самую* лучшую презентацию на GECCO. Он делал обычную презентацию по использованию L-систем для эволюции шагающих созданий, а в конце вытащил из холщового мешка груду игрушечных запчастей и сервоприводов. Затем нажал на кнопку, и груда ожила и начала шагать по столу. Это была игрушечная версия лучшей особи по результатам запуска. Для поиска более подробной информации о работе Грега хорошим началом будет его диссертация: Gregory Hornby, 2003, Generative Representations for Evolutionary Design Automation, Ph.D. thesis, Brandeis University

Алгоритм 62 Простой способ генерации множества продукционных правил

$\overrightarrow{t}$ ←данное множество терминальных символов (которые не расширяются)
p ←вероятность выбора терминального символа
r ←флаг, обозначающий допускается ли рекурсия (истина) или нет (ложь)
d ←флаг, обозначающий допускается ли появление изолированных правил (истина) или нет (ложь)

n ←случайное целое > 0, выбранное по некоторому распределению
$\overrightarrow{v}$ ←вектор уникальных символов $\langle v_1, \ldots, v_n \rangle$ {Символ v_1 будет «стартовым».}
$\overrightarrow{\textit{Правила}}$ ← вектор пустых правил $\langle \textit{правило}_1, \ldots, \textit{правило}_n \rangle$ {Создаваемые правила.}
for i от 1 до n **do**
  l ←случайное число ≥ 1, выбранное по некоторому распределению
  $\overrightarrow{h}$ ←вектор символов $\langle h_1, \ldots, h_l \rangle$
  for j от 1 до l **do**
    if ($r =$ ложь И $i = n$) ИЛИ $p <$ случайное число, равномерно распределенное в [0.0; 1.0] **then**
      h_j ←случайно выбранный терминальный символ из $\overrightarrow{t}$ и отсутствующий в $\overrightarrow{h}$
    else if $r =$ ложь **then**
      h_j ←случайно выбранный нетерминальный символ из $v_{i+1}, \ldots, v_n$ и отсутствующий в $\overrightarrow{h}$
    else
      h_j ←случайно выбранный нетерминальный символ из $\overrightarrow{v}$ и отсутствующий в $\overrightarrow{h}$
    end if
  end for
  $\textit{правило}_i$ ← правило вида $v_i \to h_1 \wedge h_2 \wedge \cdots \wedge h_l$
end for
if $d =$ ложь **then**
  for i от 2 до n **do**
    if v_i не встречается в головной части среди правил $\langle \textit{правило}_1, \ldots, \textit{правило}_{i-1} \rangle$ **then**
      l ←случайное число, равномерно распределенное в интервале $[0; i-1]$
      Изменить $\textit{правило}_l$ от вида $v_l \to h_1 \wedge \ldots$ к виду $v_l \to v_i \wedge h_1 \wedge \ldots$
    end if
  end for
end if
return $\overrightarrow{\textit{Правила}}$

выбирать случайно). Мутацию правил множества можно осуществлять аналогично битовой мутации: каждое правило с некоторой вероятностью мутирует независимо от других.

Для продукционных правил обычно есть дополнительные ограничения. Если мутации подвергается головная часть правила, то необходимо, чтобы новые символы не приводили к появлению правил-«сирот»(в случае, если это не разрешается). Аналогично необходимо очень осторожно подходить к мутации *тела* (первый символ) продукционного правила — вероятно появление правила-«сироты», либо нескольких правил с совпадающим телом.

4.5.5 Рекомбинация

Если количество правил ограничено, например, по одному для каждого продукционного символа, то рекомбинацию можно произвести просто, с использованием однородного кроссинговера, который будет осуществлять обмен подмножествами правил. Если количество правил различно, то можно формировать подмножества правил от каждой особи и проводить обмен (для выбора подмножества см. Алгоритм 49).

В любом случае, если для правил есть ограничения (такие как у продукционных правил), то нужно соблюдать осторожность и с кроссинговером: что если получится правило-«сирота»? (Или это не важно?) Что если исчезнет правило, которое активируется другими правилами? Нужно ли чтобы какое-либо правило было активировано вместо него? Одна из самых больших загвоздок при скрещивании множества продукционных правил произвольной мощности заключается в объединении символов: в одном из множеств могут встречаться символы, отсутствующие в другом. Поэтому такие множества достаточно сильно «разобщены». Как их объединить? Можно пройтись по множествам правил и определить, какие правила из одного множества включают правила из скрещиваемого множества и каким образом. Результат может получиться очень сложным, и здесь нельзя дать хорошего совета или рекомендации: как и для графов, все держится на эвристиках (*ad-hoc*).