К. НЕЙЛОР

КАК ПОСТРОИТЬ СВОЮ ЭКСПЕРТНУЮ СИСТЕМУ

Перевод с английского
канд. техн. наук
Н. Н. СЛЕПОВА

МОСКВА
ЭНЕРГОАТОМИЗДАТ
1991

ББК 32.973-01
Н 45
УДК 681.518.54

CHRIS NAYLOR
BUILD YOUR OWN EXPERT SYSTEM
John Wiley & Sons Ltd., Chichester, 1987

Нейлор К.
Н 45 Как построить свою экспертную систему: Пер. с англ. — М.: Энергоатомиздат, 1991. — 286 с.: ил.
ISBN 5-283-02502-0

Книга написана как введение в теорию экспертных систем. В простой и доходчивой форме показано, как построить нужиую пользователю экспертную систему и соответствующую базу знаний, имея под руками персональную ЭВМ и используя язык программирования БЕЙСИК. Изложение ведется на конкретных примерах и доведено до рабочих программ. Приведены примеры функционирующих экспертных систем.

Для программистов-любителей и широкого круга читателей

Н $\frac{2402010000\text{-}311}{051(01)\text{-}91}$ 202-90 **ББК 32.973-01**

ISBN 5-283-02502-0 (рус.)
ISBN 1-84058-071-5 (англ.)
ISBN 0-470-20946-1 (англ.)

ПРЕДИСЛОВИЕ ПЕРЕВОДЧИКА

Предлагаем читателям перевод книги Криса Нейлора «Как построить свою экспертную систему». Первое издание этой книги (в оригинале) появилось в 1987 г. и быстро разошлось. Книга была переиздана в 1988 г. С этого издания и был сделан перевод.

В книге описаны основы проектирования экспертных систем — предмета достаточно нового и модного в настоящее время в связи с бурным прогрессом в области персональных компьютеров, открывших широкие практические возможности создания баз данных и баз знаний — основы для создания экспертных систем.

Экспертные системы как предмет создания и эксплуатации были до недавнего времени закрыты для широкого круга пользователей компьютеров. Такие системы были ориентированы на мощные ЭВМ и использовали для своего создания специальный язык программирования ЛИСП (язык обработки списков), что создавало дополнительные трудности в их освоении.

Прочтя книгу Криса Нейлора, вы поймете, что таких трудностей больше не существует. В простой и доходчивой манере автор вводит вас в круг проблем экспертных систем и освещает все этапы их создания, иллюстрируя примерами, реализуемыми на любом совместимым с IBM PC компьютере, так как в качестве языка программной реализации использован доступный всем БЕЙСИК.

Книга написана живым образным языком, изобилует в оригинале истинно английским юмором, что делает изложение похожим на доверительный рассказ об истории развития экспертных систем «в картинках», хотя надо отметить, что автор не избежал естественных при этом многословия и повторов.

Мы будем рады всем, кто прочтет эту книгу (это будет полдела) и воспользуется советами автора книги, рискнув построить свою экспертную систему (вот тут-то и начнется настоящее дело).

Чтобы помочь рядовым пользователям компьютеров, все программы были просмотрены на отсутствие синтаксических ошибок, были устранены опечатки в оригинальных программах и внесены незначительные изменения, улучшающие в некоторых программах форму распечатки результатов (например, форма диалога в гл. 7) и позволяющие получить совпадающие результаты для тех примеров, которые были приведены автором. Вместе с тем ни переводчик, ни редакция в целом не могут гарантировать безошибочной работы приведенных автором программ.

Слепов Н. Н.

ПРЕДИСЛОВИЕ

Мой потенциальный читатель стоит сейчас в книжном магазине с книгой в руках, которую он, возможно, купит. Бросив взгляд на обложку книги, он читает предисловие, чтобы понять нужно ли ее покупать.

Предисловие, где сказано: «Теория...», — вызывает обычно негативную реакцию. Слишком много посвящено теории.

Реальным доводом в оправдание покупки может служить то, что книги, посвященные компьютерам, относительно дешевы. Кроме того, наша книга содержит рабочие примеры на языке БЕЙСИК для IBM PC и совместимых с ней ПЭВМ, следовательно, покупатель будет иметь ряд программ за те же деньги, т. е. бесплатно. Можно также сказать, что использованный в программах язык БЕЙСИК настолько прост, что не представляет труда переделать написанные на нем программы для использования их на какой-то другой машине, если вам не довелось купить IBM PC.

Книга расскажет вам об экспертных системах не так много, но, честно говоря, раскрыть в точности то, что она действительно могла бы сообщить о них, было бы скорее похоже на совет вообще отказаться от ее приобретения: ведь вы могли просто стоять в книжном магазине и читать предисловие, упорно не желая расставаться со своими деньгами.

Выясняется, однако, что она позволит вам построить либо экспертную систему для медицинской диагностики (это возбуждает аппетит, как хорошая закуска), либо экспертную систему, которая определит, почему ваша машина не хотела заводиться сегодня утром. Либо, наконец, она позволит вам построить обучаемую (на примерах) экспертную систему, которая поможет провести экспертизу по широкому кругу проблем в различных областях.

Она научит вас достаточно широко использовать статистические методы и правила логического вывода, так что я думаю, вам стоит раскошелиться и купить один экземпляр.

Я знаю, что сейчас у вас туго с деньгами и вы, возможно, могли бы потратить их с большей пользой на что-нибудь еще, на развлечение, например, что принесло бы вам больше удовольствия, но не забывайте и об авторе, он тоже должен на что-то жить — это ясно. Если подсчитать затраты только на ленту для пишущей машинки, то они выглядят довольно внушительно.

Итак, я чувствую, что после всего сказанного вы уже собрались раскошелиться и купить книгу и подумываете, как теперь начать ее читать. Все, что вам осталось, это начать сначала и изучать ее до тех пор, пока не дойдете до технического приложения. Потом подумайте о том, что вам хотелось бы сделать самим: например, компьютерную обучающую систему. Но может быть вы захотите овладеть основами диагностики? После этого начинайте отыскивать нужные главы, понятия или формулы и, наконец, свяжите все воедино.

С другой стороны, вы могли бы, держа компьютер включенным, в процессе чтения вводить примеры, приведенные в книге, и разбираться, как они работают, что во многом способствовало бы лучшему усвоению материала. Правда, если вы последуете этому совету, то для его осуществления вам, возможно, потребуется вся оставшаяся жизнь.

Важно отметить, что эта книга составлена не так, как обычный учебник. Ее главы нельзя рассматривать изолированно друг от друга. Так или иначе, но вы должны изучить весь материал от начала и до конца, чтобы впитать идеи, заложенные в книге, в нужной последовательности. Некоторые читатели уже отметили, что она читается скорее как роман, а не как учебник. Это, конечно, прекрасный отзыв.

Перед публикацией книгу прочитали Грехем Бич, Фил Брэдли, Билл Хадспит и Фил Манчестер. Все они приняли участие в обсуждении книги и своими комментариями способствовали некоторому улучшению окончательного варианта рукописи.

Если кто-то, читая книгу, найдет возможным сделать замечания, которые могли бы привести к полезным изменениям в книге, напишите пару строк об этом издателю книги.

Крис Нейлор

ОБЩИЕ ЗАМЕЧАНИЯ О ПРОГРАММАХ

Все программы, приведенные в этой книге, написаны на языке БЕЙСИК и могут быть выполнены на ПЭВМ типа IBM PC или совместимых с ними персональных ЭВМ Программы прошли тестирование с использованием интерпретаторов BASICA и GWBASIG (IBM PC) и Locomotive BASIC 2 (AMSTPAD PC 1512).

Все программы, однако, были написаны с использованием довольно простого подмножества команд БЕЙСИКа, что дает возможность любому программисту средней квалификации перевести их на любой другой диалект этого языка, имеющийся на компьютере, находящемся в распоряжении пользователя.

Все примеры распечаток результатов выполнения программ в книге получены с помощью интерпретатора BASICA IBM PC. Читатели, использующие другие компьютеры или интерпретаторы БЕЙСИКа, получат, вполне вероятно, похожие результаты «прогонов» программ, хотя не исключены и некоторые небольшие отличия. Это обычно происходит из-за различия методов округления числовых значений на разных машинах и методов генерации случайных чисел, а также ввиду неодинаковых методов реализации вычисления встроенных функций. Однако эти различия не оказывают существенного практического влияния на результаты вычислений.

Все приведенные программы были напечатаны прописными буквами, за исключением символьных констант, где часто использовалось сочетание строчных и прописных букв. В зависимости от интерпретатора БЕЙСИКа, который здесь используется, вы можете обнаружить, что образ программы на экране не всегда соответствует используемому в программе сочетанию прописных и строчных букв. Например, пользователи Locomotive BASIC 2 на AMSTPAD PC 1512 заметят, что только ключевые слова языка BASIC 2 отображаются на экране прописными буквами, тогда как имена переменных отображаются строчными буквами. Это вполне нормально и не является ошибкой в программе.

Там, где длина строки в программе оказывается больше длины строки обычной печатной страницы, осуществляется перенос информации на другую строку, что гораздо удобнее для пользователя. В результате этого строки на экране могут оказаться «разбитыми» иначе, чем это показано в книге. Например, пользователи BASICA на IBM PC заметят, что любая строка длиннее 80 символов отображается на экране так, как если бы она была продолжена без всяких отступов. Это сделано специально и здесь нет никакой ошибки.

Итак, вводя программу в компьютер, строго придерживайтесь сути программы и не старайтесь копировать расположение элементов отдельных строк, показанное в распечатках программ в книге[1].

---

[1] В оригинале некоторые строки превышают допустимую интерпретаторами БЕЙСИКа длину — 255 символов. Чтобы это не приводило к ошибкам при копировании программы пользователями, которые не любят читать «Общие указания», такие строки при переводе отредактированы. При переводе программ были устранены замеченные опечатки, что говорит о том, что программы набирались, а не распечатывались на принтере с компьютера, и гарантировать их работоспособность, видимо, нельзя. — *Прим. пер.*

Глава 1. ЧТО ТАКОЕ ЭКСПЕРТНЫЕ СИСТЕМЫ!

Под экспертной системой понимается система, объединяющая возможности компьютера со знаниями и опытом эксперта в такой форме, что система может предложить РАЗУМНЫЙ СОВЕТ или осуществить РАЗУМНОЕ РЕШЕНИЕ поставленной задачи. Дополнительно желаемой характеристикой такой системы, которая многими рассматривается как основная, является способность системы пояснять, по требованию, ХОД СВОИХ РАССУЖДЕНИЙ в понятной для спрашивающего форме. Метод достижения таких характеристик, основанный на наборе формальных решающих правил, называется программированием.

Такое формальное определение экспертных систем одобрено комитетом группы специалистов по экспертным системам Британского компьютерного общества.

Много лет назад, когда Земля была еще молодой, а Солнце широко улыбалось, вставая каждое утро, не было ничего похожего на экспертные системы. И вдруг оказалось, что все вокруг только и говорят об этом. Почему это произошло?

Ответ на поставленный вопрос, видимо, заключается в том, что в один прекрасный день ученые вдруг обнаружили, что правительство выделяет все больше и больше денег на работы, связанные с применением ЭВМ. Затем оказалось, что имеется, хотя и ограниченный, шанс получить доступ к этому жирному финансовому пирогу, пока еще не съеден добрый кусок дорогой и заманчивой исследовательской активности, которая поощрястся в рамках программ применения ЭВМ. Понимая это, ученые изобрели экспертные системы, которые в силу того, что никто в правительственных кругах не знал, что это такое, привлекли их внимание. Они добились выделения субсидий, хотя бы для того, чтобы понять, что же это такое.

Другие ученые, не будучи столь меркантильными — племя, которое теперь вымирает, — и не интересуясь правительственными субсидиями как таковыми, посчитали, что уровень правительственных субсидий явно недостаточен для того, чтобы быть на уровне в этой области. Эти альтруисты без обиняков утверждают, что они хотят сделать компьютеры более доступными для широкого круга людей, а кроме того «заставить» компьютеры думать наподобие людей и вообще желают заменить людей компьютерами. По их мнению компьютеры должны быть для пользователя удобны и приятны в общении. На конечной стадии такого умозрительного анализа они, вероятно, захотят, чтобы компьютеры взяли на себя работу по распределению правительственных субсидий.

Все это прекрасно, но реальная задача заключается в получении достаточной информации об экспертных системах для того, чтобы пользователь сумел привлечь государственные субсидии, внося на равных и свой вклад. Проблеме субсидий не будет уделено в этой книге существенного внимания, но автор надеется, что книга поможет вам понять сущность экспертных систем настолько, чтобы вы могли по крайней мере построить свою собственную систему независимо от того, будет она субсидирована правительством или нет.

В первой части книги намеренно не рассматриваются вопросы построения экспертной системы — компьютерной обучающей системы, которая может аккумулировать накопленный опыт в широком спектре областей знаний. Его приобретают на основе экспертизы.

В дальнейшем, предполагая наличие знаний в некоторой определенной предметной области, мы построим экспертную систему, способную использовать эти знания в манере, схожей с экспертом-человеком.

Необходимо сделать еще одно замечание. В этой книге с целью большего понимания все примеры приведены на языке БЕЙСИК. Однако в ней нет описания команд или приемов программирования на БЕЙСИКе.

Предполагается, что пользователь должен научиться этому самостоятельно.

Все примеры программ изложены на тех диалектах БЕЙСИКа, которые позволяют прогнать программы на IBM PC или совместимых с ним машинах, включая Amstrad PC 1512. Программы проходили проверку на интерпретаторах BASICA, GWBASIC и Locomotive BASIC 2. Однако в программах не использовались дополнительные возможности языка БЕЙСИК, представляемые его конкретными версиями, поэтому указанные программы довольно просто модифицировать для использования на других компьютерах.

В последней главе книги подытожен материал, изложенный в предыдущих главах, так что вы сможете оценить то, что уже смогли прочесть раньше.

1.1. ДЛЯ ЧЕГО ВАМ НУЖНА ЭКСПЕРТНАЯ СИСТЕМА?

Большинство существующих экспертных систем обладают двумя главными недостатками: во-первых, лично вы часто не понимаете, как они работают, а во-вторых, лично у вас их нет. В ряде случаев отмеченные недостатки можно рассматривать как достаточно серьезные. В этой ситуации вы оказываетесь в глупом положении, избегая смотреть людям в глаза или обсуждать это с кем-либо. Вы «погружаетесь» в руководство по КОБОЛу, прислушиваетесь к разговорам других о «базах знаний», «искусственном интеллекте», «отображении реального мира» и т. д. Вы боитесь прийти неподготовленным и спросить обо всем этом, чтобы не попасть в глупое положение. Вы чувствуете себя изгоем и презираете самого себя. Все это минут через пять может показаться утомительным и скучным. Тем более что вы чувствуете интуитивно, что сам по себе указанный предмет не может быть настолько сложным, как это показалось вначале. Об этом свидетельствует и тот факт, что люди, которые освоили эти системы, вряд ли в чем-то умнее вас.

Единственное, что вам нужно, чтобы все было так, как полагается, это иметь свою экспертную систему. В отличие от искусно сделанной, но малоизвестной конструкции она позволит вам доверительно относиться к каждой мелочи и разглагольствовать с важным видом о предмете экспертных систем. Вместо того чтобы прятать голову и скрывать свои мысли, вы сможете бросить насмешливый взгляд в сторону тех, кто действительно не понимает, как работает их экспертная система. А в кругу тех, кто вообще не имеет никакой экспертной системы (кто даже не начал понимать, как они вообще организуют свои дела), вы сможете удивленно поднять насмешливо сдвинутые брови. Для вас (высокого, уверенного, находящегося в центре внимания и свободно держащегося перед толпой почитателей) собираемся мы строить экспертную систему.

При работе с этой книгой не требуется никаких специальных знаний, хотя было бы не плохо, если бы вы имели доступ к компьютеру, в противном случае многое из того, что будет излагаться, покажется слишком академичным.

1.2. ДЛЯ ЧЕГО ЛЮДИ ХОТЯТ ИМЕТЬ ЭКСПЕРТНУЮ СИСТЕМУ!

Существуют два основных варианта использования экспертных систем, соответствующие социологическим концепциям явной и скрытой функций. Явная функция экспертной системы должна обеспечить с помощью комьютера компетентность (специальные знания) человека-эксперта. Например, такие системы могут диагностировать болезнь, воссоздавать химическую структуру, разведывать места добычи полезных ископаемых или решать другие подобные задачи. Они достаточно удобны в работе, а кроме того, если можно так сказать, имеют возможность объяснить свои действия и мнения так, как это мог бы сделать человек-эксперт. И, наконец, подобно человеку они способны даже научить кого-то, как проводить экспертизу.

Другая функция экспертной системы — скрытая — должна препятствовать получению скрытой от непосвященного информации о том, как построена система. Обычно для экспертных систем используют большие ЭВМ, такие, каких у вас нет, и экзотические языки, например ЛИСП и ПРОЛОГ, о которых вы даже не знаете. Все это направлено на то, чтобы реализовать преимущества программного обеспечения, имеющегося на рынке для применения в экспертных системах. Ведь если специальные типы запросов могут быть заложены в описании одной из явных функций таких экспертных систем, то эти запросы наверняка не могут быть удовлетворены кем-то, имеющим в своем распоряжении микро-ЭВМ и интерпретатор с языка БЕЙСИК. Ясно, что от этого зависит и стоимость экспертной системы.

Чтобы устранить эти препятствия, необходимо иметь руководство, ориентированное на применение микро-ЭВМ и описывающее, как построить экспертную систему. Причем это должно быть не исчерпывающее руководство, сообщающее о последних достижениях в этой области, а руководство, достаточное для того, чтобы снять покров мистики с самого названия «экспертные системы» и дать рядовому пользователю возможность начать самостоятельно создавать такие системы.

До настоящего времени действительной проблемой в понимании экспертных систем было отсутствие консультационных пунктов, куда можно было бы прийти и получить начальные сведения о предмете. Пока вы не подготовили себя к тому, чтобы иметь определенный взгляд на указанные вопросы, все возможные на данный момент руководства по указанному предмету только подчеркивают трудности какого-либо его аспекта. Последнее обстоятельство заставляет пользователя все бросить и оставить на откуп экспертам.

Это напоминает попытку забраться на гору (или даже на холм), на склонах которой совершенно не за что зацепиться. Если бы вы только смогли начать карабкаться, то почувствовали бы большой оптимизм в оценке того, что можно сделать в этой ситуации, и смогли бы самостоятельно проделать несколько последовательных шагов в этом направлении, не пользуясь какой-либо книгой. И наконец осуществить то, что и делает человек-эксперт с помощью экспертной системы, — разработать что-то для себя.

Искусство разработчика заключается в том, чтобы научиться думать в том же ключе, как и современные лидеры в этой области. Нужно уяснить, что существуют какие-то конкретные и доступные вещи, какие-то спасительные зацепки, за которые можно уцепиться. Важно понять, что здесь есть интересные задачи и что, если подумать, они прекрасно решаются, что решения, уж если они получены, полностью поддаются объяснению на понятном всем языке и что предмет экспертных систем не является на деле чем-то мистическим, а как и многие предметы, основанные на использовании компьютеров, есть нечто практическое, приземленное, как плотницкое искусство.

1.3. ЧТО ТАКОЕ ЭКСПЕРТНАЯ СИСТЕМА?

Прежде чем вы начнете взбираться на свои вершины и поспешите сесть за свой рабочий стол, было бы хорошо на минуту остановиться и спросить: «Что же в сущности такое экспертные системы?» Если вы подумаете, то поймете, что ответ на этот вопрос может оказать глубокое воздействие на ваше понимание этого законченного объекта. Фактически же мы увидим, что ответ существенного влияния на сам объект не имеет, но вы, конечно, думали иначе.

Все это началось много лет назад и обернулось пустой потерей времени, ЭВМ тогда были по производительности и возможностям такими же, как сейчас карманные калькуляторы, а слово «транзистор» произносили с благоговейным трепетом. Некоторое время спустя, будучи повержены без слов силой и сложностью монстров, с которыми они работали, специалисты по ЭВМ нашли слова, чтобы высказать свои сокровенные желания. «Было бы прекрасно, — обычно говорили они друг другу, — если бы мы получили возможность делать нечто большее, чем расчет зарплаты». «Да, — обычно соглашались говорящие, — мы получили в распоряжение сотни слов в памяти и ЭВМ решает теперь задачи быстрее, чем главный бухгалтер, но все-таки наше общение с ней похоже на общение с идиотом». «Допустим, что это действительно идиотизм, но наша ЭВМ выглядит все же лучше идиота». И они продолжали бы попивать допоздна свое пиво и обдумывать вновь и вновь схемы, с помощью которых можно было бы высвободить всю мощь их франкенштейновского монстра [1].

Все это так, но они все еще здесь, хотя время идет, все так же попивают пиво и строят козни против главного бухгалтера, а компьютер все так же исправно печатает платежные ведомости. Прогресс не был столь драматически быстрым в направлении обучения компьютера делать что-то неординарное, но побудительные мотивы к этому остались прежними.

[1] Роман Мэри Шелли (герой романа — робот, созданный Франкенштейном). — *Прим. пер.*

Что касается таких, как Франкенштейн, то они мечтали вдохнуть немного жизни в свое детище. Они не хотели иметь только машину, способную лишь выполнять арифметические действия. Они хотели иметь по-настоящему думающую машину. Так на практике начало развиваться новое направление — искусственный интеллект, и экспертные системы были частью его.

Вы, очевидно, отметили, что и бухгалтер, например, воплощал не только знаменитую суммирующую машину (хотя и ходят слухи), он скорее был кем-то вроде эксперта в своей области. Он мог бы написать книги, демонстрируя знания и возможности далеко за пределами, доступными компьютеру. И это не были бы только счета, которые не в состоянии разработать компьютер. В любом деле существовали эксперты, обладающие знаниями и профессиональным опытом, которые делали их незаменимыми и в полном смысле слова дорогими. Каждый раз, когда группа специалистов по ЭВМ беседовала с одним из таких экспертов, его могли слушать часами.

Прошло время, и настал такой день, когда эксперта смогли заменить компьютером, т. е. «выключить» его на время и даже забыть заплатить.

Мечты тем временем неслись вперед. Неслись так, что, увлекая за собой, заставляли людей думать, что должны быть пути реализации их мечты. Вся проблема заключалась в том, как это сделать.

Легко видеть, что если бы кто-то решил эту проблему, то он мог бы назвать свое детище экспертной системой. Появилась бы возможность провести экспертизу, которую до этого осуществлял человек-эксперт, «отключив» последнего на ночь. Но проблему, состоящую в том, чтобы показать, как это должно быть сделано, трудно было решить — едва ли она уже разрешена.

Спросите кого-нибудь, как работает эксперт? Вы увидите, что кроме комментариев типа «медленно» никто ничего вразумительного не скажет. Или, по крайней мере, говорящие это не знают с определенной степенью уверенности, что вы можете написать компьютерную программу, которая будет делать то же, что и эксперт. Любые объяснения только одурманят вас такими словами, как «суждение» и «опыт», которые отсутствуют в словарях формальных языков. Основные идеи здесь, однако, довольно просты.

Люди, как говорят, являются думающими машинами общего назначения. Дай им любую задачу и немного опыта и они используют свои суждения для того, чтобы придумать удовлетворительное решение данной задачи. Все эти люди могут быть представлены неким скоплением клеток мозга, как-то связанных друг с другом, тогда как все компьютеры — это набор ячеек памяти, также определенным образом связанных друг с другом. Следовательно, достаточно написать программу, которая будет решать задачи (в самом общем случае), и вы получите систему, заменяющую людей. Добавьте сюда преимущества таких систем: отсутствие поломок и «забывчивости», безошибочную работу, отсутствие затрат на оплату труда и т. д. Проблема, однако, заключается в том, что подобный «решатель проблем общего назначения» — вещь во многом иллюзорная.

Меньше иллюзий возникает в случае «решателя специальных задач». Допустим, например, что вам требуется что-то вычислить. В этом случае вы легко можете написать программу для осуществления этих вычислений Введите в машину соответствующее арифметическое выражение, и компьютер для вас все сделает сам. Кажется вполне правдоподобным, что компьютер при оценке арифметического выражения выполняет те же операции, что и математик. Он не просто складывает несколько чисел друг с другом, а взяв строку символов и манипулируя ими, преобразует последние в форму, удобную для восприятия. В итоге любые арифметические действия оказываются тривиальными.

Если вам кажется, что приведенные выше рассуждения не относятся к рассматриваемой теме, предлагаем решить задачу, которую ваш компьютер не смог бы в данный момент выполнить. Рассмотрим задачу интегрирования математического выражения. Для функций определенного типа интегрирование может вызвать известные трудности, следовательно, справедливо предположить, что эта задача является подходящим упражнением для экспертных систем. Допустим, однако, что вы создали такую систему. Как бы она в действительности выглядела? Вероятно, система напоминала бы ваш компьютер с интерпретатором языка БЕЙСИК. Правда, в отличие от ЭВМ такая система имела бы в языковой структуре больше выражений (операторов). В самом деле, если вы посмотрите на некоторые калькуляторы, то увидите клавиши «интегрирование», которые позволяют проводить численное интегрирование некоторых функций. Итак, нет ничего особенно привлекательного в такой постановке задачи.

Проиллюстрируем свою мысль на следующем примере. Если вам известно, как решить какую-то задачу, и вы в состоянии написать соответствующую программу, то задача перестает казаться привлекательной. Некоторое время назад, если бы у вас не было специальной математической подготовки, то вы должны были бы прибегнуть к чьей-либо помощи, чтобы выполнить интегрирование. Другими словами, вам пришлось бы послать за экспертом. Имея в своем распоряжении калькуляторы, снабженные соответствующими программами, отпадает необходимость делать это, потому что данная задача фактически перестала требовать для своего решения вмешательства эксперта. Следовательно, мы едва ли назвали бы клавишу «интегрирование» на карманном калькуляторе моделью экспертной системы.

Рассмотрим более прозаический пример: работу экспертов по составлению платежных ведомостей. Фактически наиболее «живые» умы в Британской империи можно было бы найти в Армейском платежном ведомстве (Army Pay Corps). Будучи специалистами в отыскивании лазеек, особых случаев и исключений, неизвестных доселе пособий, эти эксперты (кто-то сказал бы «сверхчеловеки») безраздельно властвуют годами. И вот простая программа начисления заработной платы сделала чиновников бухгалтерий вымирающим, как динозавры, видом. Можно легко заметить, что эта программа является примером экспертной системы — она воплощает всю сумму человеческих знаний о таком предмете, как расчет заработной платы. Но сейчас уже об этом никто так не думает. Просто это еще одна проблема, решение которой кажется тривиальным после того, как оно было однажды найдено.

В аналогичной ситуации находятся экспертные системы сегодня. Успехи, полученные на предыдущих шагах, перестают быть таковыми, едва только они достигаются, по очень простой причине. Мы, как и специалисты по ЭВМ в давние времена, сидим и выжидаем, желая получить от компьютера что-то особенно интересное. Такое, что он еще не делал. Но как только мы это что-то получаем, снова жаждем чего-то нового и так до бесконечности.

Любой успех на этом пути приводит к еще одной программе — не больше, не меньше. Но все это не кажется тем одним, окончательным, большим успехом, который приносит потребителю самую совершенную программу. Ту самую, которая окончательно вдохнет жизнь в нашего монстра.

Однако если существует такая экспертная система, то положение может усугубиться тем, что появится новое поколение экспертов по функционированию таких совершенных систем. Они будут знать о системах больше, чем кто-либо другой, и им, конечно, хорошо платят за это. Если вы не верите в это сейчас, то спросите себя: «Как случилось, что с появлением системных программистов они заняли в этом деле ведущее место?».

Предположим, например, что вы разработали экспертную систему, которая в состоянии поставить диагноз заболевания. Это, конечно, уже сделано. Что, вы думаете, произошло бы дальше? В академических журналах появились бы научные статьи, описывающие эту систему и рассказывающие, что она могла бы делать. В других статьях указывалось бы, чего она не может делать, и рассказывалось бы, как построить более совершенную систему. Это то же пройденный этап. Компьютерная программа была написана, затем она подверглась критике и была улучшена экспертами, с помощью которых была затем написана более удачная программа. И так далее...

Неизбежно то, что любая экспертная система может рассматриваться как временный паллиатив, т. е. нечто, способное на некоторое время предотвратить боль потерь. Вся хитрость заключается в том, чтобы найти хороший паллиатив. Что-то такое, что заставит вас думать: «Система в самом деле умная!», даже если вы не всегда будете уверены, что она таковой является. Компьютерная программа, реализующая в конце концов то, что вы еще сами не до конца представляете, может быть написана самим компьютером. Программа, которая выполняет операции, задуманные вами, фактически нуждается в обслуживании со стороны человека-эксперта. Но она несмотря ни на что все же остается компьютерной программой.

1.4. ЧТО ВЫ ХОТЕЛИ БЫ ПОЛУЧИТЬ ОТ ЭКСПЕРТНОЙ СИСТЕМЫ!

Определяя экспертную систему не больше, не меньше как компьютерную программу, мы могли бы успокоиться, удовлетворенные сознанием того, что мы знаем все об экспертной системе и она у нас уже есть. Такой путь самый простой. Более сложный путь — выработать функциональный подход и спросить, что вы хотели бы получить от экспертной системы?

Это довольно опасный вопрос, так как ответы на него могут заставить вас провести какую-то предварительную работу, которая обычно не из приятных. Но эти компьютеры предназначены для вас. Как правило, люди определяют достоинства компьютерных программ по конечным результатам. Следовательно, если экспертные системы являются компьютерными программами, то что они делают?

В этом случае ответственность временно перекладывается на вас. Потому что вопрос не ставится так: «Что пользователь хотел бы получить от экспертной системы?». Он стоит так: «Что вы хотели бы получить от экспертной системы?» В конце концов ведь это вы собираетесь ее строить. Вы могли бы сказать в этом деле что-то свое, если получится.

Этот вопрос, конечно, легче. Усевшись в кресло, вы включили свой компьютер, закрыли глаза и размечтались. Комната плывет перед вашими глазами, приятная теплота разливается по телу. Спокойная загадочная улыбка играет на ваших губах, когда вы из праздного любопытства ввели в машину вопрос: «Как я могу стать миллионером?» — и несколько простых, хорошо подобранных фраз появились на экране в качестве ответа. «Конечно! — воскликните вы. — Да, безусловно так. Это наверняка сделает меня миллионером. Если только я построю свою экспертную систему».

Вы запишите себе этот ответ и продолжите диалог с машиной по еще одному спорному вопросу о том, как остановить процесс поседения волос, и последовавший за ним диалог о том, как остановить рост одуванчиков на вашем газоне. Все это похоже на мечту, чем оно, собственно, и является. Жаль, конечно, что на самом деле это так. Но ведь и жизнь так похожа на мечту, и никакая, пусть даже большая, работа по программированию не сможет фактически помочь этому.

Проблема заключается в том, что некоторые вещи невозможно запрограммировать. Экспертная система это или не экспертная система, но если вы собираетесь запрограммировать невозможные (непрограммируемые) вещи, то столкнетесь с определенными трудностями при их практической реализации.

Возвращаясь от мечты к реальной жизни, где вы сидите перед компьютером, не нужно слишком сильно разочаровываться. Говоря о том, что некоторые вещи невозможно запрограммировать, мы не имеем в виду, что ничего нельзя запрограммировать. Но почему, можете спросить вы, мы говорим о программировании, когда все, что мы хотим, — это получить свою собственную экспертную систему? Все это лишь для того, чтобы быть уверенным, что «экспертная система» не позволит сделать ошибки во фразе «всеобщая панацея». Рассмотрим аналогичный случай, известный как база данных.

Базы данных являются очень сложными программными продуктами. Для их создания необходимо проделать значительную работу и хорошо понимать изучаемый предмет. Это может привести к параличу интеллектуального плана среди тех, кто столкнулся с базами данных впервые. Ясно, что если у вас имеются несколько файлов, то у вас появляется и база данных (или нечто подобное). В чем же здесь проблема? Если говорить честно, то проблемы нет. Вы лишь облегчите себе «проникновение» в предмет, достаточно логичный, и вскоре освоитесь с ним.

То же самое и с экспертными системами. В них действительно нет ничего сложного. Особенно привлекает возможность наблюдать за процессом их создания. В этой связи полезно помнить идею о том, что экспертная система является системой, использующей суждения. Как только вы записали команду «сравнить одну величину с другой и затем осуществить определенное действие, зависящее от результата сравнения», вы имеете суждение или оценивание. Только при проектировании экспертных систем суждениям принадлежит ведущая роль. Суждение, а не вычисление типично для экспертных систем. Но так как суждение обычно является результатом вычислений, то разница между ними является скорее концептуальной, нежели фактической.

Итак, думая о том, что могла бы экспертная система сделать для вас, выясните, можно ли это свести к последовательности суждений. Если да, то вы имеете хорошие шансы построить такую систему.

Перечислим в качестве примера некоторые области ее применения: диагностика общих болезней; поиск неисправностей в простых цепях; диагностика болезней растений; анализ электрокардиограмм; классификация животных, птиц или растений по видам.

Но у вас, конечно, могут быть свои собственные идеи.

1.5. ЭКСПЕРТНЫЕ СИСТЕМЫ: НЕКОТОРЫЕ НЕВЕРНЫЕ ПРЕДСТАВЛЕНИЯ

По поводу экспертных систем высказывается довольно много разных суждений, но не все они верны́. Общая ошибка, как это ни странно, заключается не в переоценке того, что могут делать экспертные системы в принципе, а скорее в недооценке их возможностей. Вы, наверное, слышали, что «экспертная система может делать только...», и дальше перечисляют ее функции, а в конце — вывод о том, что ничего другого она делать не в состоянии. Это странно, потому что если вы сумели продумать, как сделать какую-то вещь, то вы сможете спроектировать экспертную систему, реализующую то же самое.

Чья-то экспертная система выполняет то-то и то-то. Но ваша! Это другое дело. Ваша система не подвержена таким ограничениям. Ваша система будет больше и лучше, чем любая другая.

Рассмотрим такой пример. Ваша экспертная система может быть экспертом только в определенном вопросе. Явная неправда. Все, что вы собирались сделать, — это создать экспертную систему по двум вопросам, следовательно, вы уже доказали, что критика неверна.

1. **Ваша экспертная система может делать только то, что (и лучше) мог бы делать человек-эксперт.** Это также неверно. Допустим, что вы решили построить систему, для которой вообще не существует экспертов. Если она работает, то вы тем самым доказали, что вышеупомянутое утверждение ложно. Более типичной является картина, когда обычная экспертиза в конкретной области менее полна́. В таком случае ваша собственная экспертная система должна делать все немного лучше, чем человек-эксперт.

2. **Ваша экспертная система никогда не заменит че-**

ловека-эксперта. Конечно заменит. Иначе нет необходимости ее строить.

Человеческая натура такова, что всегда найдется кто-нибудь не согласный с тем описанием экспертной системы, какое дано в этой книге. Возможно, и вы, читатель, не согласны со мной. Тогда разрешите сделать следующие замечания.

Специальное свойство экспертных систем — свойство «адаптивности» — означает, что поведение реализующей ее программы меняется (обычно в лучшую сторону) в течение некоторого времени. Это условие выполняется, если исходная информация (знания) будет заложена в систему правил, которые в дальнейшем могут быть легко изменены пользователем, или путем построения новых знаний в результате анализа входной информации при небольшом вмешательстве пользователя. И в том, и в другом смысле системы, описанные в этой книге, являются адаптивными.

Цель большинства современных экспертных систем — осуществлять общую экспертизу в широкой области путем «очерчивания» различия между знаниями, которые они используют, и механизмами, манипулирующими этими знаниями. В ходе изменения природы специфического знания такая система становится затем способной выполнять свои манипуляции, чтобы стать экспертом в новой области. Например, экспертная система, построенная для осуществления диагностики заболеваний, при замене медицинских знаний на знания об инженерных сооружениях становится экспертом по инженерным сооружениям. Программы, описанные в этой книге, организованы именно так.

Часто говорят, что специальные свойства экспертных систем основаны на правилах логического вывода. В отличие от обычных компьютерных программ, реализуемых из последовательно выполняемых операторов, экспертные системы состоят из набора правил, которые не выполняются последовательно, а «срабатывают» только так и тогда, когда выполняются соответствующие условия.

В буквальном смысле такое описание довольно обманчиво, потому что за исключением некоторых машин все компьютеры обрабатывают свои программы аналогичным образом. Концептуально, однако, целесообразно рассматривать один вариант организации работы машины — но только в том смысле, что экспертные системы часто ведут себя так, как если бы каждая строка программы начиналась со слова IF (ЕСЛИ)... И это, косвенно может быть, объясняет, как записаны программы в данной книге.

Глава 2. СТАТИСТИЧЕСКИЙ ПОДХОД

2.1. ФОРМИРОВАНИЕ МАТРИЦЫ

Вы можете воскликнуть: «Не достаточно ли предварительных сведений? Где же, наконец, обещанные экспертные системы?» Как же на самом деле построить экспертные системы? Думаю, сделать это не сложно.

Прежде всего определим двухмерный массив, считая его образом прямоугольной матрицы. Обдумаем все вопросы, которые вы хотели бы задать вашей экспертной системе. Поместите в столбцы матрицы все возможные ответы, затем продумайте все сопутствующие фрагменты исходной информации, которые, возможно, потребуются эксперту для получения ответов на поставленные вопросы, и отметьте строки матрицы с указанными фрагментами. Поясним сказанное на конкретном примере.

Допустим, что вы хотели бы, чтобы ваша экспертная система сообщила, будет ли завтра дождь. И компьютер, после некоторой паузы на обдумывание, ответил бы ДА или НЕТ. Чтобы не оконфузиться, он не ответит ДА и НЕТ.

Итак, есть два возможных ответа, поэтому явно недостаточно иметь матрицу с двумя столбцами. На этом этапе мы не знаем, сколько фрагментов информации нам нужно, чтобы ответить на данный вопрос, но можно предположить, что 10 строк матрицы будет достаточно. Следовательно, в БЕЙСИКе это можно описать массивом E(10,2). Символ E используется из мнемонических соображений для идентификатора, связанного со словом эксперт (expert).

Проанализируем табл. 2.1, где слева сверху вниз перечислена необходимая информация, используемая в системе для выработки прогноза погоды. Номера 1—10

Таблица 2.1. Экспертная матрица

Наблюдения	*Завтра дождь*	*Завтра нет дождя*
1	a	b
2	c	d
.	.	.
.	.	.
.	.	.
10	s	t

соответствуют вопросам: сегодня дождь?, сегодня холодно? и т. д. Буквы a, b, ..., s, t в массиве означают возможные варианты правильных ответов, которые мы пока не знаем. Они могут впоследствии быть вписаны вместе с фрагментами информации, связанными с погодой. Задача экспертной системы — решить, какую из двух колонок: *Завтра дождь* или *Завтра нет дождя* — следует выбрать для ответа на указанные (1—10) вопросы.

Например, наблюдение 1 справедливо, если существует утвердительный ответ (ДА) на вопрос: «Сегодня холодно?»

Из многочисленных наблюдений экспертами холодных дней в 60 % случаев на следующий день снова будет дождь. Следовательно, a будет равно 60, или 0,6, если вы предпочитаете дроби. Так же могут быть получены значения b, c и т. д., т. е. появляется уверенность в том, что можно получить определенный вид ответа, основанный на сделанных нами наблюдениях.

В результате мы установили базу знаний и область запросов — понятия достаточно важные, чтобы вспомнить о них не раз при удобном случае.

Рассмотрим *область запросов*. Ее название говорит само за себя. Это основной предмет экспертных систем. Если что-то попало в область запросов, то вы можете спросить систему об этом, если нет — то не можете. В нашем случае областью запросов является погода (или дождь, если вы хотите ограничить ее). Нет абсолютно никакого смысла спрашивать вашу экспертную систему, как вы можете стать миллионером, так как этот вопрос лежит за пределами области запросов. Может быть, конечно, и другая причина, по которой вы не

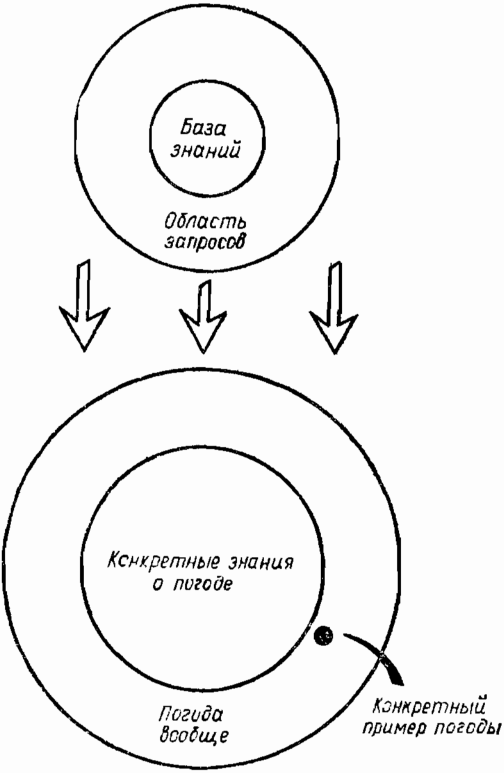


Рис. 2.1. База знаний и область запросов

можете задать системе такой вопрос, но всему свое время.

Рассмотрим теперь *базу знаний*. Ей соответствует массив E (10,2). Он содержит все знания эксперта[1] по данному предмету, и в нашем случае (и это существенно) база знаний пуста. В ней нет никаких сведений о погоде. Поэтому не имеет смысла даже спрашивать такого эксперта о погоде. Обратимся теперь к рассмотрению рис. 2.1.

---

[1] Имеется в виду машина-эксперт. Здесь и далее экспертную систему будем называть экспертом, если это не приводит к путанице. — *Прим. пер.*

Область запросов — это та область, в которой экспертная система должна быть экспертом. Наш эксперт должен предсказывать погоду, следовательно, область запросов содержит запросы о погоде. Чтобы быть экспертом в этой области, экспертная система должна иметь базу знаний, содержащую информацию, которую мы заложили в нее относительно прогноза погоды. В идеальном случае база знаний целиком определяет область запросов, т. е. ей «следовало бы знать» все относительно рассматриваемого объекта (погоды). На практике маловероятно знать все — она будет знать лишь некоторую часть информации. Другими словами, база знаний фактически меньше, чем область запросов, и существует внутри нее.

Когда возникает конкретный вопрос, запрос должен попасть в область запросов. Однако он может не всегда точно попасть в базу знаний (см. рис. 2.1). Место, в которое поместили конкретный пример, зависит от степени охвата базой знаний данного конкретного примера внутри экспертной системы.

Рассмотрим человека-эксперта, к которому вы обратились с вопросом о том, возможен ли завтра дождь, надеясь получить от него ответ. Большинство людей обычно загадывает что-то, задавая такой вопрос. На сегодня у них преимущество перед компьютером: они знают, какая погода стоит сейчас, или могут выглянуть в окно, чтобы убедиться в этом наверняка, и предположить прогноз на завтра. Нам хотелось бы, чтобы экспертная система таким же образом формировала и свое заключение.

Будем различать два типа информации, связанной с данной задачей: *постоянную информацию* и *переменную информацию*. Постоянная информация должна поступать в базу знаний. Она содержит неизменные данные, которые в нашем случае будут характеризовать погоду в целом. Переменная информация не должна поступать в базу знаний. Ее специфика связана с задачей, поставленной в данный момент. В нашем случае это информация, позволяющая эксперту сказать: будет ли завтра дождь или нет. Причем такая информация предоставляется в том случае, если мы задаем этот вопрос именно сегодня, а не просто интересуемся погодой вообще.

В некотором смысле различие в этих двух типах ин-

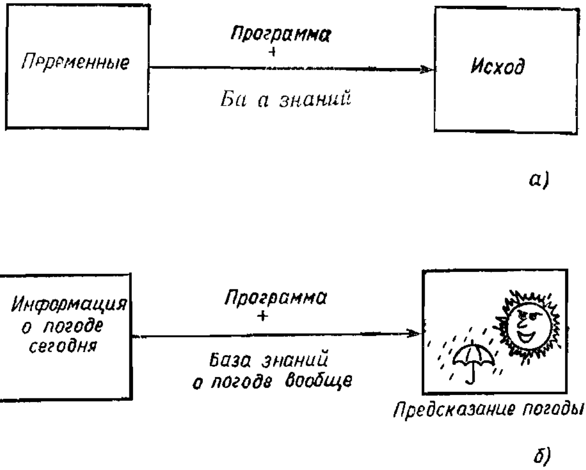


Рис 2 2 Алгоритм предсказания погоды:

а — в общем виде; *б* — применительно к конкретному прогнозу

формации похоже на различие между программами и данными. Постояниая информация — это программная часть, а переменная информация — данные для конкретной задачи. Но чтобы быть уверенным, что объект не слишком четко очерчен, полезно «размыть» его границы, указав, например, что мы можем по желанию изменять постоянную информацию и довольно часто (т. е. фактически эта информация уже не является постоянной), поэтому она оказывается по существу переменной. Большинство экспертов меняют со временем методы своей работы, и было бы неверно исключить для экспертной системы возможность обучения по мере накопления опыта. Фактически это свойство следует рассматривать как один из атрибутов экспертной системы.

Вернемся, однако, к эксперту, предсказывающему погоду (рис. 2 2). Спросим его: «Будет ли завтра дождь?» Допустим, что он не может сказать ничего определенного. Что дальше? Нас это не устраивает. Для начала сообщим эксперту какие-то общие соображения об изменении погоды. Например, если сегодня дождь, то наиболее вероятно, что завтра тоже будет дождь, поскольку он повторяется периодически. Предположим, например, что сейчас как раз дождливый период, тогда наше предположение вполне может оказаться верным. Фактически то, что мы делаем, является внедрением элементов обычной экспертизы в экспертную систему.

Допустим, если сегодня дождь, то существует 60%-ная вероятность того, что завтра тоже будет дождь. Поэтому по законам математики мы должны заключить, что завтра будет сухая погода, с вероятностью 40%. Предположим далее, что если сегодня сухо, то существует 55%-ная вероятность, что завтра тоже будет сухо. Используя аналогичные рассуждения, получаем 45%-ную вероятность дождя на завтра. Перепишем теперь исходную схему заполнения нашего массива с учетом этих знаний:

	Завтра дождь	*Завтра нет дождя*
Сыро	60	40
Сухо	45	55

Теперь, видимо, ясно, что нам нужно делать. Экспертная система должна напечатать утверждение *Завтра дождь* или *Завтра нет дождя*. Чтобы решить, что печатать, она спрашивает: «Сегодня сыро?» Если ответ утвердительный, то она печатает *Завтра дождь* так как это является наиболее вероятным исходом. Если сегодня сухо, то таким исходом может быть одно из двух событий в зависимости от того, как вы составите программу. Ответ, например, *Завтра нет дождя* мог бы быть основан на том, что если сейчас нет дождя, то должно быть сухо и, значит, наиболее вероятно, что будет сухо и завтра. Или система может спросить: «Сегодня сухо?» — и, получив утвердительный ответ, сделать из этого соответствующий вывод.

Теперь все выглядит достаточно конкретно, и вопрос заключается в том, насколько это необходимо и полезно. В конце концов вы всегда можете прослушать прогноз погоды по радио или же не слушать его, а только ждать и смотреть, что за погода установится на самом деле. И еще один момент. У вас, наверное, нет большого доверия к тем числам, которые мы поместили в матрицу (т. е. в базу знаний).

Рассмотрим некоторые возможные исходы. Возьмем, например, те числа, о которых мы рассуждали. Они, конечно, выглядят надуманными. Допустим, однако, что они верны, т. е. были точно определены, что тогда?! Видимо, в зависимости от обстоятельств нужен различный подход. Одно число может быть равно 100 %, тогда другое должно быть 0 %. В этом случае эксперт мог бы принять какое-то решение, а не просто угадать ответ. Предположим далее, что вы не знаете, что дождю обычно предшествует дождь или что чередуются мокрый и сухой периоды. Если бы наша экспертная система могла сделать такое заключение, то она знала бы больше нас и предсказывала бы погоду гораздо точнее.

Вернемся теперь снова к рассмотрению вопроса о том, будет ли завтра дождь. Вопрос, вероятно, тривиален для экспертной системы, которой мы хотим удивить мир. Но эта экспертная система имела бы разные названия при различных переменных, выводах и областях применения.

Мы могли бы спросить: «Победит ли мистер *X* в этом заезде?» Или: «Моя головная боль — это результат похмелья?» При условии, что мы можем придумать, как определять некоторые переменные, выводы и их связи между собой, никто нам не мешает взяться за решение большего набора различных задач с помощью нашей экспертной системы. Вся хитрость в том, чтобы иметь какой-то метод, подобно рассмотренному выше, для создания системы, организованной «вокруг» данной задачи, чтобы можно было начать работу.

2.2. ВЕРОЯТНОСТИ

В предыдущем примере мы думали о событии, вероятность появления которого, скажем 55, или 45, или же 100 %. Нет особых причин, указывающих, что так следует поступать и дальше. Однако использование вероятностного подхода имеет некоторые преимущества.

Вероятности являются удивительно точными характеристиками. Они задаются числами в диапазоне от 0 до 1. Если событие наверняка *должно* произойти, то вероятность его появления равна 1. Если событие не имеет шанса произойти когда-либо, то вероятность его появления равна 0. Если же событие равновероятно, т. е. может как произойти, так и нет, то вероятность его появления равна 0,5. Все остальные возможные случаи находятся в указанном диапазоне.

Рассмотрим два крайних случая, когда вероятности равны соответственно 0 и 1. Существует различие между случаем, когда вероятность равна в точности 1, и случаем, когда она равна приблизительно 1, хотя эта разница настолько мала, что мы могли бы писать просто 1. Если вероятность появления какого-то события равна точно 1, то это событие *должно* произойти. Оно просто не может не случиться. Здесь имеет место причинно-следственная связь. Допустим, что в вашей руке имеется стакан и вы вдруг выпускаете его из руки, когда под ним нет никакой опоры. В этом случае он упадет на землю. Произошло определенное событие, имеющее вероятность, равную 1. Это также причинно-следственная связь, ввиду того что падение стакана вызвано тем, что вы разжали руку.

Другой случай. Когда вы покидаете бар, в котором «пропустили» стаканчик, вы видите, что кто-то из посетителей тоже уходит. Короче говоря, все вы покидаете указанное заведение в одно и то же время с вероятностью 1 — это также определенное событие. Но оно не имеет причинно-следственных связей. Нельзя сказать, что факт вашего ухода (персонально) заставил кого-то еще решить, что больше нет никакого резона (удовольствия) оставаться в баре. Ясно только, что бармен решил закрыть свое заведение на ночь и выгоняет всех вас. Именно это и послужило причиной указанного события.

Итак, если два или больше событий происходят совместно с вероятностью 1, то связь между ними не обязательно является причинной. События могут иметь место чисто случайно, но они всегда происходят вместе.

Вернемся снова в бар и снова уроним наш стакан.

Мы сказали, что он падает на пол, но не знаем, разбивается ли при этом. Вероятно, разбивается, допустим, с вероятностью 0,999999. Чтобы не выписывать все эти девятки, мы могли бы сказать, что он разбивается с вероятностью, близкой к 1. Но это вовсе не то же самое, что с вероятностью 1. Если бы вероятность была равна 1, то стакан действительно разбился бы, и двух мнений здесь быть не может. Но до тех пор, пока магическая цифра 1 не достигается точно, существует некоторая, хотя и малая, вероятность, что указанное событие не произойдет. Эта весьма малая разница может превратиться в неразбитый стакан.

Все вышеизложенное справедливо и для вероятности, равной 0. Событие, вероятность наступления которого 0, исключено. Но событие, вероятность которого близка к нулю, не исключено.

Теперь становится совершенно ясно, что нет никаких препятствий для использования процентов. Стопроцентная вероятность того или иного события означает, что вы вполне уверены в том, что оно произойдет. Если нет четкой уверенности, то справедлива запись — вероятность 99,99 % и т. д. Вы можете использовать любую шкалу оценок, которая вам больше нравится, например от —5 до +5. Чтобы выработать определенную точку зрения, нужно пояснить, что можно говорить о вероятностях с высокой степенью точности как о математических величинах. Если требуется описать таким образом некоторое событие, то необходимо указать соответствующее значение вероятности. При выборе значений таких вероятностей используется диапазон от 0 до 1; выбор его условен, но важно, что мы можем использовать точные числовые значения.

Рассмотрим наше утверждение: «Если сегодня идет дождь, то существует 60 %-ная вероятность того, что завтра тоже будет дождь». Мы, конечно, имеем в виду, что вероятность наступления этого события — 0,6. С этого момента мы можем сделать вывод, что если бы мы наблюдали бесконечно большое число дождливых дней, то в 0,6 случаях за ними следовали бы также дождливые дни, причем без всяких оговорок типа но, если или может быть...

Наша цифра, безусловно, не является абсолютно точной, но это не делает наше утверждение, в котором она использовалась, бессмысленным. Его смысл определен достаточно однозначно, неточна только цифра, характеризующая значение вероятности. Если бы мы могли дать точную цифру, то наше утверждение было не только осмысленным, но и верным. Именно в этом все дело.

Представьте себя обладателем собственной экспертной системы. «Она решила, — объявили вы, — что существует 0,75 %-ная вероятность того, что в любой день следующей недели может начаться третья мировая война». «О! — воскликнет скучающая аудитория, — значит она в этом не уверена?» И все начнут говорить о чем-то более определенном, например о погоде. Можно было бы прийти в ярость от сознания того, что вы имеете экспертную систему, способную выработать подобное предположение, которое, оказывается, так легко опровергнуть. Есть от чего при таком повороте событий подскочить давлению крови. Слушайте, что вы сказали бы по этому поводу.

Вы используете вероятностную меру, поэтому если вероятность наступления третьей мировой войны составляет 0,75 (причем в один из дней недели), то вероятность того, что она не наступит ни в один из дней недели, равна 0,25, т. е. 1—0,75. Вероятность же того, что война не начнется ни в один из 7 дней недели (т. е. неделю мы будем жить в мире), равна $0{,}25^7=0{,}000061035$.

Вероятность же того, что следующая неделя не будет мирной, соответственно $1-0{,}25^7=0{,}999938965$, т. е. третья мировая война может стать реальностью на следующей неделе. Все это предполагает, конечно, что когда третья мировая война разразится, то факт ее возникновения (событие) окажется в действительности совершенно независимым от тех событий, что могли бы произойти накануне, а так как компьютер сделал свой прогноз на этой неделе, то это не кажется таким уж неразумным предположением. Итак, карты, разложенные в статистически независимой манере, приводят обычно к ошибочным выводам.

Вы не указали, в какой день недели все это собирается произойти, но, как видно, у нас около 6 шансов на 100 000 за то, что война не начнется вообще. Таким образом, все, что нам осталось, — пойти и купить оловянную каску в ответ на ту напряженность, которую создала информация, только что полученная нами.

Одну важную вещь следует все же помнить: не смотрите свысока на вероятности. Используя их, вы можете вычислить ряд «точных» вещей и сделать несколько «точных» утверждений, которые могут оказаться верными. И еще одно практическое замечание; если вы собираетесь построить свою экспертную систему, то не следует обольщаться тем, что вы многое получите от нее, если будете думать, что все, связанное с ней с вероятностью— лишь неясный шанс, а не реальный вклад, рассматриваемый серьезно.

2.3. ДОПОЛНИТЕЛЬНЫЕ СВЕДЕНИЯ О ВЕРОЯТНОСТЯХ

Теперь вам могло бы показаться, что вероятности для вас — открытая книга. С другой стороны, было бы полезно напомнить, что происходит, если вы имеете дело с несколькими вероятностями.

Введем некоторые обозначения: $P(A)$, $P(A\&B)$, $P(A:B)$ и определим их следующим образом: $P(A)$ — вероятность наступления события A; $P(A\&B)$ — вероятность одновременного наступления событий A и B; $P(A:B)$ — вероятность наступления события A, вычисленная при условии, что событие B уже наступило.

В дальнейшем будем называть просто $P(A)$ вероятность A; $P(A\&B)$ — совместная вероятность A и B; $P(A:B)$ — условная вероятность A при заданном B.

Использование этих вероятностей не означает попытку определить одно и то же различными способами. Если вернуться к нашему примеру с погодой, то матрица прогноза погоды (в ней используются вероятности, а не проценты) будет выглядеть следующим образом:

	Завтра будет дождь	*Завтра не будет дождя*
Сыро	0,6	0,4
Сухо	0,45	0,55

Это — *условные* вероятности. Они являются вероятностями соответствующей погоды завтра при *заданной* погоде сегодня. Например, вероятность дождя завтра при условии, что он идет сегодня, равна 0,6. Это не то же самое, что вероятность дождя завтра или совместные вероятности дождя завтра и дождя сегодня. На первый взгляд — это похожие вещи. На самом деле ничего подобного, просто делается попытка классификации вещей, в противном случае весь разговор о вероятностях как о точных вещах будет только потерей времени.

Во-первых, что такое вероятность дождя завтра? Заметьте, что мы ничего не сказали о погоде сегодня. Этот вопрос поставлен отдельно, вне связи его с погодой накануне. Он поставлен как бы вообще: будет ли дождь завтра? Информация, которую мы имеем, такова, что если в какой-то день было сыро, то вероятность дождя завтра составляет 0,6.

Во-вторых, если было сухо, то вероятность дождя завтра равна 0,45. Что мы должны сделать для того, чтобы получить общую вероятность дождя завтра? Сложить их вместе (1,05)? Вычислить их среднее значение (0,525)?

Обозначим $P(Д)$ в качестве вероятности дождя завтра и используем символы $С$ и $М$ для обозначения прогнозов *Сухо* и *Сыро* соответственно. Мы хотим найти

$P(Д)$, а в нашей таблице приводятся значения $P(Д:М)$ и $P(Д:С)$ в первой колонке. Здесь уместно дать следующую формулу:

$$P(A \& B) = P(A:B)\,P(B),$$

которая утверждает, что вероятность одновременного осуществления событий A и B равна условной вероятности A при заданном B, умноженной на вероятность B. Подумав, вы найдете эту формулу разумной.

Мы хотим знать о совместном наступлении событий A и B. Если событие B действительно произошло, то существует $P(A:B)$ — вероятность того, что произойдет событие A. Событие B происходит с вероятностью $P(B)$, поэтому мы должны допустить, что эта величина также должна фигурировать в формуле совместной вероятности появления событий A и B. Следовательно, мы умножаем $P(A:B)$ на $P(B)$, которая меньше 1, что уменьшает общую вероятность ответа с учетом того факта, что B наступает не всегда.

Итак, вероятность дождя завтра и сегодня, рассматриваемых как совместные события,

$$P(Д \& М) = P(Д:М)\,P(М),$$

т. е. вероятность того, что дождь будет и сегодня, и завтра, равна вероятности дождя завтра при условии, что сегодня сыро, умноженной на вероятность дождя сегодня, или $P(Д\&М) = 0{,}6\,P(М)$. А если сегодня сухо, то $P(Д\&С) = 0{,}45P(С)$, что соответствует выражению

$$P(Д \& С) = P(Д:С)\,P(С).$$

Но мы ведь хотели получить вероятность дождя завтра $P(Д)$. Эта величина равна, очевидно, сумме совместной вероятности дождя сегодня и завтра и совместной вероятности того, что завтра будет дождь, а сегодня сухая погода:

$$P(Д) = P(Д \& М) + P(Д \& С) = P(Д:М)\,P(М) + P(Д:С)\,P(С) = 0{,}6P(М) + 0{,}45P(С).$$

Здесь уместно напомнить, что сегодня день был или сырым, или сухим, т. е. $P(М)+P(С)=1$, как это и должно было быть при двух исходах.

Окончательно получаем

$$P(Д) = 0{,}6P(М) + 0{,}45P(С) = 0{,}6P(М) + 0{,}45(1 - P(М)) = 0{,}6P(М) + 0{,}45 - 0{,}45P(М) = 0{,}15P(М) + 0{,}45.$$

Итак, для того чтобы ответить на вопрос: «Какова вероятность, что завтра будет дождь?», мы должны знать вероятность того, что сегодня будет сыро. Это, конечно, вопрос, на который мы можем легко ответить, так как считаем, что вероятность дождя завтра — это то же самое, что и вероятность дождя в любой другой день, т .е. то же самое, что и вероятность дождя сегодня. Следовательно, $P(Д)=P(М)$, тогда

$$P(Д) = 0,15P(Д) + 0,45 \text{ или } 0,85P(Д) = 0,45, \text{ т. е.}$$

$$P(Д) = 0,45/0,85 \approx 0,53.$$

Наиболее вероятно завтра будет дождь, как это было и сегодня и, вероятно, будет до конца света. Это наше предсказание погоды для вас.

Заметим, что мы не смогли бы получить этого результата, если бы не приняли $P(М)=P(Д)$. Если же $P(М)$ характеризовало бы туманный день, например, то оно не было бы равно $P(Д)$ на завтра. Тогда мы должны были бы иметь другие источники информации для оценки значения $P(М)$.

Важно отметить, что данная вероятность дождя завтра не указана в таблице предполагаемых прогнозов погоды. Там приведены условные вероятности, а не вероятности отдельно взятых событий, а это не одно и то же.

В процессе вычислений мы также получили некоторые совместные вероятности $P(Д\&М)$ и $P(Д\&С)$, которые оказались равны $0,6\,P(М)$ и $0,45\,P(С)$ соответственно. Теперь мы знаем, что $P(Д)=P(М)$ и $P(С)= =1-P(М)=1-P(Д)$, т. е. $P(Д\&М)=0,6P(Д)\approx 0,3176$, а $P(Д\&С)=0,45(1-P(Д))\approx 0,2117$.

Эти вероятности намного меньше $P(Д)$, так как мы интересуемся, чему равна вероятность дождя завтра и сегодня (т. е. двух дождливых дней последовательно) и какова вероятность дождя завтра и сухой погоды сегодня (т. е. двух последовательных дней с разной погодой). Совместная вероятность формирует более точный запрос на специфическую ситуацию. В нем меньше предположения и, следовательно, шансов получить ответ, который нас интересует.

Прежде чем вернуться к обсуждению экспертных систем, познакомимся со следующей информацией.

16 июля 1917 г. в Кенсингтоне за 2 ч выпало не менее 118 мм осадков. 18 августа 1924 г. в Бриджуотере в тече-

ние 5 ч выпало 203 мм осадков. 28 июля 1917 г. в Брутоне (Сомерсет) в течение одиого дня выпало 243 мм осадков. В другой части Британской империи, в Гибралтаре, в 1836 г. 25 октября в течение одного дня выпало 765 мм осадков. В Багио (Филиппины) в 1911 г. за четыре дня с 14 по 17 июля выпало соответственно 889, 737, 432, 203 мм осадков. Вышеприведенные примеры показывают то, что мы еще не закладывали в экспертную систему. Речь идет об информации о том, что порой вообще не бывает дождей, а иногда дождь льет как из ведра.

2.4. БОЛЬШЕ ПЕРЕМЕННЫХ

Снова рассмотрим типичный день. Холодно, сыро, ветрено и густой туман. Вы подняли воротник своей поношенной куртки, чтобы укрыться от всего этого, и закурили сырую сигарету, дабы согреться от ее тепла. Капля дождя большего, чем обычно, размера погасила вашу сигарету. Вы невольно спрашиваете самого себя: «Завтра будет такая же погода?». Возможно, да. Кто-то мог, видимо, сказать вам, что здесь Великобритания — страна, в которой может быть все, что угодно, до тех пор, пока дождь льется вам за шиворот. Но хотя бы для того, чтобы дать волю дикой фантазии, вы должны предположить, что и сухие дни также случаются. Итак, вопрос все тот же: «Будет ли завтра дождь?»

Включите экспертную систему, чтобы узнать это. Однако прежде чем включить ее, убедитесь, что под крышкой системы нет воды. Помните матрицу системы E(10,2)? Хорошо, заполните ее так, как показано ниже.

	Завтра будет дождь	*Завтра не будет дождя*
Холодно . . .	*P(Дождь : Холодно)*	*P(Сухо : Холодно)*
Сыро	*P(Дождь : Сыро)*	*P(Сухо : Сыро)*
Ветрено . . .	*P(Дождь : Ветрено)*	*P(Сухо : Ветрено)*
Туман	*P(Дождь : Туман)*	*P(Сухо : Туман)*

Допустим, что мы можем легко найти условные вероятности, помещенные в матрице. Если, скажем, сегодня холодно и не сыро, не ветрено и не туманно, то решение задачи вам покажется смешным. Так как все, что от вас требуется, — это посмотреть на указанную таблицу, определить, что больше — условная вероятность, что будет дождь при заданном *Холодно,* или условная веро-

Рис. 2.3. Случай с двумя переменными:

Сегодня дождь и *Завтра дождь*. Заштрихованная область показывает, что они коррелированы друг с другом. Если мы знаем, был или не был *Сегодня дождь*, то можем оценить вероятность *Дождя завтра* по площади этой заштрихованной области

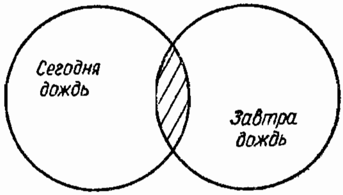


Рис. 2.4. Случай, аналогичный показанному на рис. 2.3, но с переменной *Сегодня холодно*

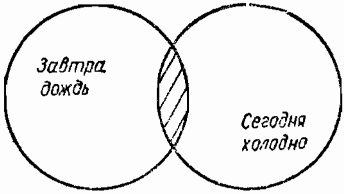
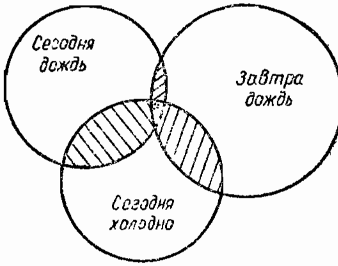


Рис. 2.5. Случай с тремя переменными. Если мы одновременно имеем информацию как о том, что *Сегодня дождь*, так и о том, что *Сегодня холодно*, то проблема усложняется, поскольку площадь заштрихованной области, содержащаяся в переменной *Завтра дождь*, зависит от степени корреляции этих двух переменных



ятность, что не будет дождя при заданном *Холодно*, и выбрать наиболее вероятный исход, который соответствует наибольшему значению условной вероятности.

Несмотря на это возникает вопрос: что же делать, если переменных больше, чем одна (рис. 2.3—2.5)? В нашем случае есть четыре переменные. Как будет работать ваша экспертная система при прогнозе сырой или сухой погоды завтра? Может показаться, что это тривиальный вопрос. Если, допустим, в действительности холодно, сыро, ветрено и туманно, то это звучит так же, как если бы какой-то идиот мог предсказать дождь с высокой степенью вероятности. И в этом следует разобраться.

Даже если какой-то идиот сделал для себя некоторое предположение на основе этой информации, то ваша экспертная система должна быть в состоянии сделать хоть какое-то предположение.

Давайте вначале рассмотрим переменные, которые у нас есть. *Холодно* — это признак плохой погоды, но значит ли это, что идет дождь? На деле чаще всего — нет. Если погода холодная, то атмосфера не может содержать так много водяных паров, как это бывает в теплую погоду, поэтому если и был дождь, то воды в облаках не так много, как кажется. Чтобы поверить в справедливость этих слов, вы можете взглянуть на цифры в месячных сводках осадков, выпавших в Великобритании. В зимние месяцы дождь идет не так часто, как летом, что может показаться несколько странным для вас, если ваша крепкая память хранит дни, когда вы насквозь промокали зимой. Вполне возможно, что вы не запоминаете теплый дождь так хорошо, как холодный.

Сыро. Допустим, наступает дождливый период, поэтому дождливый день вполне мог предшествовать дождливому дню. До тех пор, конечно, пока он не перестанет лить завтра (это последнее предположение известно у нас как оптимизм).

Ветрено — это часто предвещает дождь. Теплый влажный воздух в холодной области может обрушить на нашу голову холодный дождь. Такое движение воздуха — еще один пример, объясняющий ветреную погоду.

Туманно. Если стоит туман, то ветра, как правило, не бывает, не правда ли? Туман возникает обычно из неподвижного переувлажненного воздуха. Он может или превратиться, или не превратиться в дождь.

Итак, что мы имеем после рассмотрения наших четырех переменных? Пока ничего определенного. Некоторые (но не все) переменные указывают на дождь. Конечно, было бы очень удобно для нас, если бы все они указывали на дождь с вероятностью, большей 0,5, поскольку в этом случае мы не сомневались бы, что наша экспертная система предскажет дождь.

Но представим себе, что вместо переменной *Туманно* мы использовали переменную *Солнечно.* Предположим далее на минуту, что погода характеризуется переменными *Холодно, Сыро, Ветрено* и *Солнечно.* Метеоролог назвал бы такую погоду «неустановившейся», но для наших целей это лишь досадное обстоятельство, потому что переменная *Солнечно,* как правило, указывает на

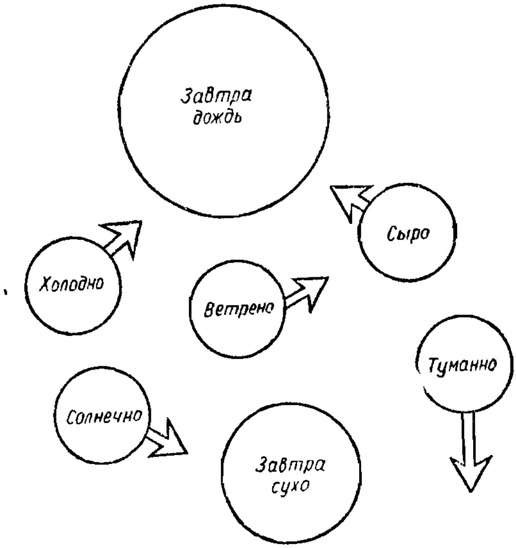


Рис 2.6. Указатели погоды

сухую погоду завтра, а переменные *Холодно, Сыро* и *Ветрено* предсказывают обычно дождь.

В общем случае мы имеем множество переменных и у нас все будет хорошо до тех пор, пока все они будут указывать в «одном направлении». Но неизбежно возникнут проблемы, если, как показано на рис. 2.6, ситуация изменится. Что же делать со всем этим?

Вернемся к нашим условным вероятностям. Мы начали с *P (Дождь : Дождь сегодня)* — вероятности того, что завтра пойдет дождь с учетом дождя сегодня. Однако то, что мы получили, можно представить как *P (Дождь : событие X)*, где под указанным событием *X* имеются в виду следующие варианты: *Холодно, Сыро, Ветрено* и *Туманно.*

Мы хотим знать вероятность *P (Дождь : Холодно&&Сыро&Ветрено&Туманно)* и особенно — будет она больше или меньше, чем вероятность *P (Нет Дождя : : Холодно&Сыро&Ветрено&Туманно)*. Возвращаемся к одной из формул, которые уже рассматривали *P (Дождь* : событие *X*) = *P*(*Дождь&*событие *X*)/*P* (событие *X*) или

P(*Дождь : Холодно & Сыро & Ветрено & Туманно*) =

$$= \frac{P\text{ (Дождь завтра \& Холодно сегодня \& Сыро сегодня \&}}{P\text{ (Холодно \& Сыро \& Ветрено \&}} \rightarrow$$

$$\rightarrow \frac{\text{Ветрено сегодня \& Туманно сегодня)}}{\text{Туманно)}}.$$

Возможно, теперь вам станет понятнее, почему мы потратили столько времени для описания различных видов вероятностей. Это было сделано, чтобы подчеркнуть, что существуют различные виды информации о возможных конкретных обстоятельствах и задаваемых вопросах, которой мы можем располагать.

Допустим, мы начали с такого, например, утверждения об осуществлении события, как *P (Дождь : Дождь)*. Утверждение по форме отличается от вопроса, который мы только что задавали. Вы можете в общем случае рассмотреть несколько утверждений относительно условных вероятностей и преобразовать их в одно большое утверждение относительно одного события, связанного фактом появления со многими другими событиями.

Рассмотрим пример. Предположим, что погода пасмурная и туманная. Вероятность *Дождя завтра* при условии, что сегодня пасмурно, равна 0,75; вероятность *Дождя завтра* при условии, что сегодня туманно, также 0,75. Поскольку по условию задачи сегодня пасмурно и туманно, то, складывая обе вероятности, мы получаем вероятность *Дождя завтра,* равную 1,5, что, очевидно, неверно, так как она не может превышать 1.

Допустим, что мы рассматриваем вероятность $P(A\&B)$ совместного осуществления двух событий. Первое событие — *Дождь завтра* при условии, что сегодня пасмурно, а второе событие — *Дождь завтра* при условии, что сегодня туманно. Примем также, что события эти не зависят друг от друга, тогда $P(A\&B) = P(A)P(B)$, что соответствует в нашем случае $0{,}75 \cdot 0{,}75 = 0{,}5625$. Полученный результат нас не удовлетворяет. Действительно, оба условия указывали на *Дождь завтра,* а фактическая вероятность, вычисленная с учетом обоих факторов, оказалась меньше, чем с каждым из них в отдельности, что противоречит логике. Очевидно, что *Пасмурно* и *Туманно* — признаки, указывающие на один и тот же характер погоды. Оба они вызваны частицами воды в воздухе, разница же состоит в том, что в случае тумана частицы достигают бóльших размеров, чем при пасмурной погоде.

Итак, *Туманно* и *Пасмурно* — это лишь различные слова, характеризующие одну и ту же погоду, поэтому учет обоих признаков не дал нам никакой дополнительной информации о погоде. Следовательно, вероятность *Дождя завтра*, равная 0,75, ничем не должна отличаться от предсказанной при наличии одного из признаков. Не нужно долго размышлять, чтобы увидеть, что в нашей исходной таблице нет ничего, что указывало бы на это. Если бы у нас в левой части таблицы находились признаки *Пасмурно* и *Туманно*, то наряду с условными вероятностями *Дождя завтра* мы имели бы фактически один и тот же признак, указанный дважды, но в экспертной системе отсутствовали бы указания, что это одно и то же.

Отмеченная проблема до некоторой степени характерна для многих переменных. То, что мы обнаружили (и не учли), является фактом корреляции между этими переменными. Если бы последние не зависели друг от друга, то нам было бы легче, однако в общем случае это не так. Рассмотрим признаки *Сыро* и *Ветрено*. В этом случае они не означают, конечно, одно и то же. Но часто бывает так, что если *Сыро*, то и *Ветрено*, т. е. эти признаки не являются совершенно независимыми друг от друга. Если бы мы рассматривали указатель *Солнечно*, то он не был бы зависим от указателя *Туманно*, так как одно исключает другое.

К счастью, проблема разрешима. Что же для этого нужно?

1. Рассмотреть все переменные, которые вы имеете, например n.

2. Продумать все возможные комбинации указанных переменных. Их общее число есть сумма всех сочетаний «из n по x», где x изменяется от 0 до n, т. е. $n!/((n-0)!0!)+n!/((n-1)!1!)+n!/((n-2)2!)+\ldots+$ и т. д. Например, в случае четырех переменных, влияющих на погоду, мы имеем $4!/((4-0)!0!)+4!((4-1)!1!)+4!/((4--2)!2!)+4!/((4-3)!3!)+4!/((4-4)!4!) = 1+4+6+6++1=16$.

Существует 16 возможных комбинаций с четырьмя переменными типа Да/Нет. Здесь восклицательный знак означает факториал, он выглядит как крик о вычислительных страданиях.

3. Составить новый список переменных. В нем надо указать не только сами переменные, но и их возможные комбинации.

4. Вписать в таблицу вероятности для всех переменных и их комбинаций при условии осуществления тех или иных комбинаций переменных.

5. Когда вы хотите использовать свою систему, нужно только посмотреть, какая комбинация переменных у вас есть, найти эту комбинацию в новой расширенной таблице и считать соответствующие ей значения условных вероятностей для каждого из возможных исходов.

Выслушав все это, вы с удивлением поднимите брови, скривите губы и воскликните: «Вы должно быть шутите!» (Здесь! — восклицательный знак, а не факториал).

Прошу извинить меня, но это то, что вы должны были сделать. Все, конечно, требует времени, но...

Задача решается значительно проще, если у вас не так много переменных. Но вы ошибаетесь, считая, что можно просто что-то загрузить в компьютер, благо у него большая память.

2.5. ТЕОРЕМА БАЙЕСА

Если, однако, вы решили сделать так, как мы вам советовали, то могли бы ошибиться в дальнейшем, пытаясь применить теорему Байеса в процессе вычислений. Преподобный Байес (об имени вы могли догадаться из названия теоремы) посвятил свою жизнь изучению извечных проблем, как, например, проблемы статистики. Его теорема утверждает, что

$$P(Д:X) = P(X:Д)\,P(Д)/(P(X:Д)\,P(Д) + \\ + P(X:\text{не } Д)\;P(\text{не } Д)).$$

Другими словами, пусть *Д* — событие, означающее *Дождь завтра*, а *X* — любая конкретная комбинация событий, описывающих погоду сегодня (например, *Сыро, Ветрено, Холодно, Противно*). Тогда вероятность дождя завтра при условии, что погода сегодня характеризуется комбинацией *X*, равна вероятности наступления события *X* сегодня при условии, что завтра будет дождь, умноженной на вероятность дождя сегодня, деленной на полную вероятность наступления события *X*. Докажем это утверждение.

Известно, что

$$P(Д:X) = P(Д:X)/P(X),$$

поэтому $P(X:Д) = P(X\&Д)/P(Д)$.

А поскольку $P(X\&Д) = P(X:Д)P(Д) = P(Д:X)P(X)$, то окончательно получаем

$$P(Д:X) = P(X:Д)P(Д)/P(X) = P(X:Д)P(Д)/(P(X:Д)P(Д) + P(X:\text{не }Д)P(\text{не }Д)).$$

Теперь, если вам известна вероятность $P(Д:X)$, воспользуемся этой величиной. Однако может оказаться, что легче определить $P(X:Д)$ — это зависит до некоторой степени от характера вопроса. Для ответа на один вопрос вам придется найти вероятность дождя завтра при наступлении события X сегодня. Для ответа на другой вопрос вам придется вычислить вероятность наступления события X сегодня при условии, что завтра пойдет дождь (или более правдоподобно — вероятность наступления события X вчера при условии, что сегодня идет дождь). Вам также (хотя бы для того, чтобы почувствовать, что вы собираетесь углубиться в проблему) нужно задать вероятность $P(X:\text{не }Д)$, т. е. вероятность наступления события X при условии отсутствия осадков сегодня. Вы, возможно, решите, что этот метод даже хуже, чем любой другой, рассмотренный ранее.

Однако это не так. Конечно, будет совсем плохо, если все события окажутся коррелированными друг с другом. Тогда вам придется задать вероятности для каждой возможной комбинации. Если же все они окажутся независимыми, можно вычислить вероятность $P(Д:X)$. Допустим, например, что под событием X подразумеваются два события Y и Z. В этом случае $P(X:Д) = P(Y\&Z:Д) = P(Y:Д)P(Z:Д)$. Но $P(Д:X) = P(Д:Y\&Z)$ не равно $P(Д:Y)P(Д:Z)$. Если вы не верите этому, то достаточно выполнить вычисления двумя различными способами и сравнить полученные результаты.

Ниже (во второй части книги) приведена программа, реализующая описанный метод построения экспертной системы в предположении, что все события независимы друг от друга. Если это не так, то программа работает плохо. Предположение о независимости событий делается слишком часто. Конечно, это упрощает задачу создания системы, облегчает нам жизнь, снимает проблему задания большого числа коррелированных комбинаций, которые следует рассмотреть. Ясно, однако, что такое предположение во многих случаях вряд ли бывает оправдано. С другой стороны, если среди переменных оказывается много взаимосвязанных вещей, то задача фактически становится «необъятной».

Если вы когда-нибудь доберетесь до конца книги, то вам, видимо, следует вернуться к этой главе и подумать хорошенько, как реализовать систему с помощью данного конкретного метода. Но может случиться и так, что, дойдя до конца книги, вы продадите ее и пойдете в бар пить пиво. В этом случае вероятнее всего вы не будете сильно обеспокоены тем, будет или не будет дождь завтра.

Глава 3. ПОСТРОЕНИЕ СИСТЕМ БЕЗ ИСПОЛЬЗОВАНИЯ ВЕРОЯТНОСТЕЙ

К этому моменту странное беспокойство, видимо, заполнило ваше существо. Вы думали, что построение экспертных систем должно быть проще. Фактически я обещал это в той или иной мере. У вас в конце концов есть компьютер — почему бы не поручить ему делать работу за вас? Все, что вы спрашивали, касалось вероятностей. Почему, например, вы должны давать эксперту вероятности тех или иных исходов, для того чтобы получить готовое решение?

Честно говоря, если вы проделали всю эту работу, то вам не нужен никакой компьютер. Вы можете сэкономить немного времени и сделать это самостоятельно. Если вы знаете вероятности, соответствующие каждому набору событий (признаков), то вам не нужен никакой механический эксперт. Истинная причина, почему вам необходима экспертная система, заключается в том, чтобы она выполняла за вас всю работу, которую вы сами сделать не можете, и сообщала вам новую информацию.

Все это, конечно, комментарий, эмоции. Однако ваша собственная экспертная система отличается от большинства существующих систем. В основном экспертные системы в настоящее время основываются на довольно интенсивных исследованиях и определенных знаниях экспертов, чтобы выяснить шансы на получение того или иного исхода в зависимости от наступления каждого возможного события. Человек-эксперт знает эти шансы. Люди, которые используют его знания, строя на этой основе экспертные системы, также знают эти шансы. Они закладывают их в экспертную систему, чтобы последняя после соответствующего программирования тоже с ними «познакомилась». Затем разработчики могут передать экспертную систему кому-то еще, кто не знает всего вышеизложенного, и этот человек будет потрясен.

Некоторые экспертные системы работают по-другому. В них компьютер использует знания эксперта, но в любом случае предполагается присутствие человека в качестве действующего лица в процессе выработки решения. Как же построить вашу собственную экспертную систему? Если вам все известно о каком-то вопросе, то нет необходимости строить экспертную систему, чтобы сообщить ей ваши знания. Необходимость в ее построении возникает, когда вы не знаете всего, что требуется. Если вы чего-то точно не знаете, то просто не сможете это запрограммировать. Если же вы пошли на поводу у обстоятельств, пытаясь выяснить все относительно изучаемого предмета, в экспертной оценке которого вы нуждаетесь, то это неизбежно повлечет за собой новые беды — потребуется описать все в программе, вместо того чтобы просто зафиксировать нужное у себя в голове. Короче говоря, ваши требования к экспертной системе существенно отличаются от требований тех, кто построил в настоящее время такие системы.

Вам нужно нечто такое, что поможет построить ее быстро и просто, даст хорошие результаты и не заставит вас углубиться на много лет в изучение предмета экспертизы.

Итак, что же вам делать? С самого начала вы стремились, чтобы ваша экспертная система была лучше всех остальных или по крайней мере отличалась от них.

3.1. КАК ЗАСТАВИТЬ КОМПЬЮТЕР ДЕЛАТЬ ЧЕРНОВУЮ РАБОТУ

Система теперь меняется. Больше вы не будете вводить в машину необходимые вероятности, хотя знаете о них достаточно, чтобы оценить, что с ними можно сделать. Вы дадите ей столько информации, чтобы она смогла переработать ее в нужный вам ответ. В нашем случае мы собираемся получить не только экспертную, но также и обучающую систему, так как хотелось бы, чтобы машина кого-то еще и обучала. Процесс обучения называется машинным обучением (термин полезно запомнить, чтобы при случае поразить им окружающих).

Чтобы вы чувствовали, что дела идут, уместно напомнить о программе, которая была заложена в компьютер.

Что же вы при этом делаете? Вы устанавливаете экспертную систему так, чтобы она провела с вами урок, во время которого вы учите ее принимать решения на основе имеющегося «жизненного» опыта. После такой тренировки вы освобождаете систему от своей опеки и даете ей возможность сделать самостоятельное решение, используя ее собственные экспертные возможности. Пока вам не пришлось анализировать программу, нет необходимости выяснять, как система получила данное решение.

3.2. ОБУЧАЮЩАЯ СИСТЕМА

Программа, которую мы будем использовать, приведена на рис. 3.1, а ее структурная схема — на рис. 3.2. Если вы введете программу в компьютер и запустите ее, то на экране появятся вопросы о количестве задействованных переменных. Затем машина задаст в программе три массива с помощью оператора DIM. Один из них — массив RULES — должен содержать правила вывода суждений (которые разрабатывает ЭВМ), другой—массив VALUE — значения переменных, представленных в данном конкретном случае, а массив VAR$ служит для хранения имен переменных.

Программа по мере продолжения диалога запрашивает имена этих переменных. Она также запрашивает вас о двух других именах — OUTCOME1$ и OUTCOME2$ — двух возможных исходных. Затем в цикле программы выясняются детали конкретного примера (в этом случае требуются ответы типа Да/Нет). Собрав все необходимые сведения, делают предположение относительно возможного исхода и формируется некое предложение. Когда вы согласны с мнением системы, вводите *Y*, если Да, или *N*, если Нет. Затем перехо-

```
10 CLS
20 INPUT "ВВЕДИТЕ ЧИСЛО ПЕРЕМЕННЫХ"; VAR
30 DIM VALUE(VAR), RULES(VAR), VAR$(VAR)
40 FOR I=1 TO VAR
50 VALUE(I)=0
60 RULES(I)=0
70 NEXT: PRINT
80 PRINT "НАЗОВИТЕ ЭТИ ПЕРЕМЕННЫЕ"
90 FOR I=1 TO VAR
100 INPUT "ИМЯ ПЕРЕМЕННОЙ: "; VAR$(I)
110 NEXT: PRINT
120 PRINT "НАЗОВИТЕ ВОЗМОЖНЫЕ ИСХОДЫ:"
130 INPUT "ПЕРВЫЙ ИСХОД: "; OUTCOME1$
140 INPUT "ВТОРОЙ ИСХОД: "; OUTCOME2$
150 PRINT
160 FOR I=1 TO VAR
170 VALUE(I)=0
180 PRINT "ПЕРЕМЕННАЯ: "; VAR$(I)
190 INPUT "ЭТА ПЕРЕМЕННАЯ ЕСТЬ (y/n) ";A$
200 IF A$="Y" OR A$="y" THEN VALUE(I)=1
210 NEXT
220 DECISION=0
230 FOR I=1 TO VAR
240 DECISION=DECISION+VALUE(I)*RULES(I)
250 NEXT
260 PRINT "ВОЗМОЖНЫЙ ИСХОД – "; : IF DECISION>0 THEN
    PRINT OUTCOME1$ ELSE PRINT OUTCOME2$
270 INPUT "ЭТО ВЕРНО (y/n) "; A$
280 IF A$= "Y" OR A$= "y" THEN 150
290 IF DECISION>0 THEN
    FOR I=1 TO VAR: RULES(I)=RULES(I)–VALUE(I): NEXT
    ELSE FOR I=1 TO VAR: RULES(I)=RULES(I)+VALUE(I): NEXT
300 GOTO 150
```

Рис. 3.1. Простая обучающая программа

дите к другому примеру. Если же вы не согласны, то система изменяет свое правило вывода суждений и переходит к другому случаю.

Следует отметить, что постепенно машина улучшает свои предсказания относительно возможного исхода, хо-

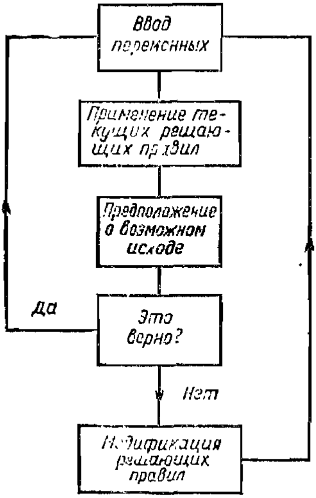


Рис. 3.2 Структурная схема процесса набора решающих правил

тя насколько лучше — зависит от того, какую информацию вы в нее ввели вначале. Если, например, вы все еще пытаетесь предсказать погоду и возможными исходами является *Дождь/Нет Дождя*, то трудно сказать, была ли машина права, до тех пор пока вы не подождете до завтра, чтобы все это выяснить. Тогда окажется, что и один запрос в день — это не слишком часто, поскольку такова скорость обучения.

Существенно более продуктивной схемой обучения является использование систематизированного списка сводок погоды, так как в этом случае вы будете фактически знать реальный исход.

С другой стороны, лучше не иметь таких проблем. Вам следует изобрести нечто более строгое, например систему классификации объектов. Допустим, вы исследуете некоторый объект — он должен принадлежать к одному из двух классов. Продумайте, какие переменные будут его характеризовать, и введите их. Программа должна быть в состоянии определить, к какому классу принадлежит объект, исследуемый вами.

Вы можете, например, захотеть, чтобы экспертная система отгадала, что вы задумали: *Птицу* или *Самолет*. Список переменных в этом случае может быть таким: *Крылья, Хвост, Клюв, Двигатель, Оперение, Шасси* и т. д. Очевидно, если вы определили, что предполагаемый объект имеет *Шасси*, то каждый скажет, что это несомненно *Самолет*, а не *Птица*, так как если вы определяете объект, то невозможно избежать ответа на вопрос о *Шасси*. Следовательно, ответ вполне определен и задача состоит только в том, чтобы выяснить, действительно ли экспертная система отвечает с такой же определенностью, как и мы.

Если выберем в качестве объектов *Птицу* и *Самолет* и запустим программу, то вначале система будет предсказывать, что мы загадали *Самолет,* а не *Птицу.* Мы сообщим ей, что она ошиблась. В следующий раз она угадает уже правильно. Затем мы загадаем *Самолет* и посмотрим, может ли система дать правильный ответ. Если нет, то мы сообщим ей правильный ответ и фактически она достигнет такого состояния, что не даст больше ни одного неправильного ответа. Таким образом, система станет действительно экспертом в распознавании двух объектов: *Птицы* и *Самолета.*

Это прекрасно, мы начинаем думать, что наконец-то получили что-то существенное. Но перед тем как отпраздновать это событие, было бы полезно выяснить, что же все-таки произошло и почему произошло именно так, а не иначе. Может ли указанная программа быть универсальным средством решения задач, которые ставятся перед экспертными системами? Прежде всего опишем переменные, которые считаем равными 1, как если бы они были истинными.

	Да?
Крылья	1
Хвост	1
Клюв	1
Двигатель	0
Оперение	1
Шасси	0

Другими словами, массив переменных VALUE сформирован из 0 и 1 в соответствии с тем, имеет или нет данный объект указанные признаки. Если рассматриваемый объект *Птица,* то массив VALUE формируется в виде (1,1,1,0,1,0), если *Самолет,* то (1,1,0,1,0,1).

Теперь посмотрим массив RULES, который содержит правила, вырабатываемые экспертной системой для вывода суждений относительно исхода двух возможностей. В момент, когда система перестает делать ошибки, массив RULES принимает вид (0,0,1,—1,1,—1).

Сформируем теперь переменную DECISION путем перемножения массивов VALUE и RULES так, чтобы

$$\text{DECISION} = \text{DECISION} + \text{VALUE(I)}\cdot\text{RULES(I)}$$

для всех значений $I = 1,\dots,6$.

Следовательно, если теперь мы возьмем массив VALUE для *Птицы*, то получим для переменной

DECISION=0+0+1+0+1+0=2. Если же определим массив VALUE для *Самолета,* то переменная DECISION = 0+0+0—1+0—1=—2. Эксперт может сказать, что объект — *Птица,* если переменная DECISION положительна, и *Самолет,* если она отрицательна. Закончив анализ, делаем вывод, что экспертная система обучена корректно (на рассматриваемом наборе переменных) и любые ошибки исключены.

Возвращаясь вновь к изучению массива правил вывода, заметим, что переменным *Крылья* и *Хвост* соответствуют 1. Мы этого не учли при выработке решения. Для этих переменных в векторе решения были 0, что не могло изменить значение вектора DECISION. Это вполне объяснимо, так как и *Крылья,* и *Хвост* есть и у *Птицы,* и у *Самолета.* Эти знания, следовательно, не говорят нам ничего решающего. С другой стороны, значения переменных *Клюв* и *Крылья* равны +1, а *Двигателя* и *Шасси* — —1.

Итак, кажется, что поведение системы выглядит достаточно логичным, по крайней мере, для данного примера. На этом этапе картина существенно отличается от той, о которой мы говорили в разделе о вероятностях. Так ли это? Проверим. Мы можем выписать таблицу вероятностей снова и заполнить ее условными вероятностями, как и раньше.

Птица		*Самолет*	
P(Птица : Крылья)	0,5	*P(Самолет : Крылья)*	0,5
P(Птица : Хвост)	0,5	*P(Самолет : Хвост)*	0,5
P(Птица : Клюв)	1,0	*P(Самолет : Клюв)*	0,0
P(Птица : Двигатель)	0,0	*P(Самолет : Двигатель)*	1,0
P(Птица : Оперение)	1,0	*P(Самолет : Оперение)*	0,0
P(Птица : Шасси)	0,0	*P(Самолет : Шасси)*	1,0

Если рассматривать *Крылья* и *Хвост,* то таблица не позволяет нам однозначно выбрать объект: *Птицу* или *Самолет,* так как они имеют одинаковые шансы. Четыре другие переменные указывают нам либо на *Птицу,* либо на *Самолет* (но не на оба объекта сразу), поскольку соответствующие им вероятности равны или 1, или 0. Показательной в этом смысле является диаграмма на рис. 3.3.

В этом примере если у вашего объекта есть *Клюв,* то он должен иметь *Оперение* и не иметь *Двигателя* и *Шасси.* Следовательно, это *Птица* с вероятностью, равной 1,

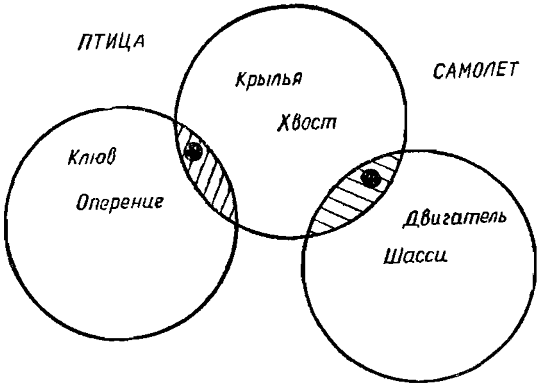


Рис. 3.3. *Птица* имеет *Крылья, Хвост, Клюв* и *Оперение*. *Самолет* имеет *Крылья, Хвост, Двигатель, Шасси*

Если же объект имеет *Двигатель*, то значит и *Шасси*, поэтому он не может иметь *Клюв* и *Оперение*. Это должна быть не *Птица*, а *Самолет* с вероятностью 1.

Рассмотренный пример дает представление о возможностях нашей экспертной системы — ее правило вывода разработано таким образом, что ошибка исключается. То, что является для этой системы исходом (событием), совершается с вероятностью 1.

3.3. ДРУГИЕ ТИПЫ ДАННЫХ

В предыдущем параграфе мы сконцентрировали свое внимание на простом дихотомическом варианте (Да/Нет) работы экспертной системы. Нам хотелось лишь выяснить, присутствует или отсутствует заданное свойство, отсюда характер счета: 1 (Да) или 0 (Нет). Заметим, однако, что применялись 1 и 0, а не Да и Нет. Если использовались цифры, то могли ли они быть отличными от 1 и 0? Можно дать утвердительный ответ. Работа системы при этом не изменится.

Снова рассмотрим прогноз погоды. Обычно мы рассматривали такие характеристики, как *Холодно, Сыро, Ветрено* или *Туманно*. Теперь, если обратить внимание на реальный прогноз погоды, то увидим в нем сообщение о температуре (максимальной и минимальной), измеренной в градусах, количестве выпавших осадков в миллиметрах, средней скорости ветра в километрах в час, видимости (на дорогах) в километрах (если это Великобритания, то скорее всего в миллиметрах). Все эти характеристики дают нам значительно больше информации о погоде, чем просто указатели типа Да/Нет. Мы инстинктивно чувствуем, что потеряем что-то, отбросив эти потенциально полезные данные.

Теперь мы не будем пренебрегать ими. Вместо того чтобы спрашивать эксперта, произошло ли какое-либо событие (описанное той или иной переменной), мы напечатаем имя этой переменной и сопоставим соответствующее ей численное значение. Другими словами, мы можем задать температуру в градусах, уровень выпавших осадков в миллиметрах, видимость на дорогах в километрах и т. д.

И если мы хотим включить в рассмотрение простые (на которые можно ответить Да или Нет) события, например *Грозу* (будет или нет), то это можно делать всегда, полагая, например, что 1 соответствует наличию данной характеристики или особенности, а 0 — ее отсутствию. Указанная программа будет работать, как и раньше, но, возможно, даст лучшие результаты, так как мы вводим более исчерпывающую информацию.

Чтобы прояснить, что происходит с указанными примерами, использующими дискретные, альтернативные (Да/Нет) данные, напомним только о возможности применять и непрерывные переменные. Причем в ряде случаев результаты могли бы быть в этом случае гораздо лучше. Тем не менее мы не можем сделать этого, оставаясь в рамках, диктуемых указанными альтернативными исходами. На самом деле мы имеем право выбирать только между различными категориями исходов. Система не способна сформировать ответ, который содержал бы точное указание, сколько осадков выпадет завтра, она способна только сообщить, будут или нет, по ее мнению, завтра осадки.

Сказанное вовсе не означает, что мы не в состоянии грубо описать уровень осадков. Можно, например, разделить уровень выпадения осадков на пять категорий (табл. 3.1).

В этом случае экспертная система, имеющая теперь пять возможных исходов, могла бы сказать нечто более правдоподобное о прогнозе погоды. Теперь ясно, как построить систему с таким числом исходов. Это не озна-

Таблица 3.1. **Категории осадков**

Характеристика осадков, мм	Нет дождя	Меньше 12,5	От 12,5 до 25	От 25 до 37,5	Выше 37,5
Возможные исходы	*Сухо*	*Сыро*	*Мокро*	*Совсем мокро*	*Ужасно мокро*

чает, однако, что систему нельзя построить так, чтобы она давала точные численные ответы. Например, при желании мы могли бы даже постараться разработать уравнение множественной регрессии, описывающее выпадение осадков, основанное на большом количестве входных переменных. С его помощью удалось бы получить точную оценку (она, правда, не обязательно является точным предсказанием) уровня выпадения осадков на завтра. Проблема, однако, состоит в том, что хотя такая система может быть использована для предсказания выпадения осадков, ее нельзя достаточно легко переориентировать для работы в другой предметной области. Как, например, она могла бы распознавать, является ли исследуемый объект *Птицей, Самолетом* или, к слову, *Планером?*

Наша цель — создать собственную экспертную систему, способную быть экспертом в ряде различных предметных областей, нечто вроде устройства общего назначения, которое можно применять по собственному усмотрению. В том-то и дело, что нет особой необходимости добиваться возможности предсказывать погоду, если вы больше заинтересованы в диагностике состояния своего здоровья, чтобы, например, успешно справиться с ролью нейрохирурга или с чем-то другим, подобным этому.

3.4. ПРАВИЛА ВЫВОДА СУЖДЕНИЙ

Пусть вы сейчас установите обучающую систему и найдете, что она более или менее работает. Вы можете сказать, что все хорошо. Но как на самом деле работает система? Что она делает и может ли она делать это лучше?

Рассмотрим идею организации *пространства описаний*, которое не только полезно само по себе, но и очень выразительно с точки зрения возможности со знанием дела говорить о других типах пространств, отличных от порядком надоевшего трехмерного пространства, вечно маячащего перед вашими глазами. Для того чтобы облегчить знакомство, предположим, что вы сидите за своим столом, который для удобства имеет плоскую (двухмерную) поверхность. В вашем распоряжении наверняка имеются два типа предметов: карандаши и скрепки для бумаг — их мы будем использовать для того, чтобы проиллюстрировать некоторые моменты.

Допустим, вы хотите взять несколько карандашей и скрепок и разделить все на две кучки. Одна кучка будет состоять только из карандашей, другая — только из скрепок. Ничего сложного. Даже самый неспособный среди нас может выполнить такую задачу: достаточно брать по очереди из общей кучи предмет, определять, что это — карандаш или скрепка — и раскладывать в ту или иную кучку в зависимости от результата распознавания. Это так же легко, как и классифицировать *Птиц* и *Самолеты*, просто взглянув на сами объекты классификации. Дело в том, что задача классификации таких объектов тривиальна, если описание объекта четко совпадает с тем критерием, с помощью которого мы хотим провести классификацию.

Предположим, что по какой-то причине вы, посмотрев на объект, не можете сразу сказать, что это — *Птица* или *Самолет*, *Карандаш* или *Скрепка*. Тогда поставленная классификационная задача усложняется и вы должны подумать о других методах выработки суждения. Ситуация очень похожа на ту, в которой находится компьютер. Здесь нет большой разницы, даже если она для вас очевидна. Процесс решения задачи должен быть более сложным.

Снова возьмем карандаш и скрепки и распределим их по поверхности стола, поместив карандаш слева, а скрепки справа. Как теперь, игнорируя факт, что вы можете различить их просто визуально, вы могли бы решить, где какой предмет находится? Оказывается, это очень просто, достаточно провести линию посредине стола так, чтобы все карандаши оказались на его левой половине, а все скрепки на правой. Теперь вы в состоянии классифицировать указанные два класса объектов просто путем измерения расстояния от объекта до края стола. Именно так компьютер и делает. Кроме того, он работает более усердно, чем вы, потому что объекты не

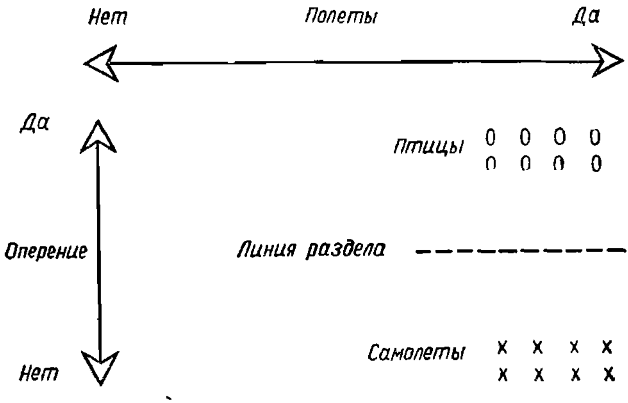


Рис. 3.4. Пространство описаний, состоящее из двух измерений: *Полеты* и *Оперение. Самолеты* могут летать, но у них нет *Оперения; Птицы* могут летать и имеют *Оперение*

просто «аккуратно» распределены в пространстве. Они распределены в особом пространстве — *пространстве описаний.*

Вернемся вновь к примеру с *Птицами* и *Самолетами* и предположим, что возможны только два типа объектов, которые описываются двумя видами информационных признаков. Летает ли объект? Имеет ли объект *Оперение?*

Освободите теперь ваш стол и напишите вдоль него признак *Полеты,* а поперек — признак *Оперение,* как показано на рис. 3.4. Если объект может летать, то поместите его справа на столе, если нет — слева. Если объект имеет *Оперение,* то поместите его у противоположного края стола, если нет — у ближнего края стола. Очевидно, в случае *Птицы* вы поместили бы объект справа у противоположного края стола. Если бы это был *Самолет,* то вы расположили бы его справа перед собой. В этой ситуации для выяснения, какой объект перед вами—*Птица* или *Самолет,* вы просто попытались бы установить, на какой четверти стола находится этот объект. Причем вы отвечали бы всегда наверняка.

Итак, вы могли бы выработать суждение просто путем проведения линии между двумя скоплениями объектов и последующего просмотра, где конкретный объект находится. И это не обязательно должны быть суждения типа Да/Нет, как в случае *Оперение/Нет оперения.* Вы могли бы при желании рассмотреть возможность предсказания погоды, проведя, например, вдоль стола шкалу с указанием на ней уровня осадков.

Отбросив предложенный образ стола, можно манипулировать любыми числами и любыми объектами, размещая их в *пространстве описания.* Это теперь не просто плоская поверхность стола или трехмерное пространство, а совершенно новое пространство, которое определяется теми переменными, которые вы используете, — каждая переменная соответствует одной оси. И значения этих переменных, соответствующие конкретному объекту, определяют его положение в *пространстве описания.* К сожалению, мы не сможем уже нарисовать эту картину для вас!

Чтобы проиллюстрировать описанную идею в трехмерном пространстве, предположим, что вы гуляете в саду в один прекрасный летний вечер. Вы увидели два облачка мошек, роящихся в лучах заходящего солнца. Допустим, что одно облачко состоит из *Больших мошек,* другое — из *Малых мошек,* а вы хотите выделить второе облачко. Один из возможных вариантов — это взять большой лист картона и поместить его между двумя облачками мошек. Это, конечно, можно сделать, хотя нужно иметь в виду, что единственной причиной привести такой пример было желание показать, что вы можете иметь две группы объектов, характеризующиеся своим положением (пространственной координатой каждой мошки), которые в состоянии теоретически разделить, расположив некоторую разделяющую поверхность между ними.

Фактически как только вы перешли в *пространство описания,* объекты могут определяться в терминах любого числа переменных. Если у вас одна размерность соответствует одной переменной, то множество объектов, определенных десятью переменными, существует в десятимерном *пространстве описания.* Идея расположения поверхности между ними по-прежнему справедлива. Вы можете иметь десятимерную поверхность для десяти переменных, и каждый объект классифицируется в соответствии с тем, на какой стороне поверхности он находится. Уравнение поверхности в n-мерном пространстве имеет вид

$$y = b_1 x_1 + b_2 x_2 + \ldots + b_n x_n,$$

где x_i — расстояние, измеренное по n осям; b_i — соответствующий коэффициент.

Следовательно, поверхность стола в двухмерной системе координат описывается уравнением $y = b_1 x_1 + b_2 x_2$.

После определения координат x_i для нового объекта можно вычислить значение y и ответить, на какой стороне поверхности он расположен. Именно это и делает обучающая программа. Она берет все x_i, характеризующие объекты, а затем вычисляет какие-то значения коэффициентов, используя указанный набор объектов в качестве примеров, и проверяет, все ли «работает» правильно. В результате получается значение y, однозначно определяющее, к какой группе принадлежит тот или иной объект.

Метод, который мы только что описали, — это метод проб и ошибок. Вы имеете программу с объектом и набором значений характеризующих его переменных. Первоначально программа формирует поверхность, которая расположена таким образом, чтобы можно было увидеть, на какой стороне ее находится объект, для принятия соответствующего решения. Возможно, ее положение таково, что принятое решение правильно. Но если оно ошибочно, то это толкает нас на то, чтобы изменить положение поверхности в *пространстве описания,* воспользовавшись вновь полученными значениями. Следующий объект позволяет провести аналогичный процесс корректировки положения поверхности. Это происходит многократно до тех пор, пока мы не будем получать каждый раз правильное решение, если, конечно, оно вообще может быть достигнуто.

Прочитав такое объяснение, вы, наверное, немало удивитесь, если найдется «наилучший» способ использования данного конкретного метода. Например, было бы целесообразно корректировать положение поверхности не только тогда, когда получено неверное решение, но и в случае, если оно правильное. Все зависит от характера решаемой задачи, но эта зависимость не такая простая.

Если у вас есть данные, которые вы хотите ввести в вашу экспертную систему, то нужно иметь в виду следующее:

когда число реально существующих примеров конечно, то наилучший результат можно получить, предъявив эксперту для обучения все возможные примеры;

может потребоваться определенное время, чтобы привести эту поверхность (правило выработки суждения) в правильное положение — однократное предъявление каждого примера может и не дать должных результатов. Следовательно, вам придется предъявлять каждый пример несколько раз.

Трудно точно сказать, сколько раз придется повторить такое обучение, чтобы все было правильно, поэтому ваш подход должен быть прагматическим (что означает: попробуй, а потом посмотри). Дайте такому эксперту набор примеров, пусть он их классифицирует, и подсчитайте количество допущенных им ошибок. Затем запустите процесс еще раз и снова подсчитайте число ошибок. Проведите ряд таких экспериментов до тех пор, пока либо не будет больше ни одной ошибки, либо статистика ошибок перестанет улучшаться.

Если вы хотите заставить систему работать на «новых» примерах, то постарайтесь сделать так, чтобы обучающие примеры были по возможности похожи на те, которые будут предъявлены для экспертизы. Это кажется очевидным, но следует иметь в виду, что если обучение проводилось на одном наборе данных, это не значит, что система будет использовать тот же набор правил на другом, отличном от первого наборе данных.

Есть еще один момент, который, возможно, удивит вас — коль экспертные системы должны заменить человека-эксперта, используя при этом похожую экспертизу, то должна ли система сама вырабатывать для себя набор правил? Получается, что она может обойтись без помощи человека вообще? Ответ мог бы быть, возможно, утвердительным, если учесть, что вы передали системе (сами того не сознавая) много собственных знаний. Однако это только начальные знания о том или ином объекте — ведь лишь благодаря вам появилась возможность судить о разнице между *Птицей* и *Самолетом*, что позволило программе заполнить ее.

3.5. ПОСТРОЕНИЕ ПРАВИЛ

Коль скоро вы поняли, что экспертная система находит уравнение поверхности в n-мерном *пространстве описания*, причем так, что эта поверхность достаточно хорошо разделяет две группы объектов, вы можете подумать, что система способна делать что-то еще более интересное.

Кроме того, обучающие системы находят разделяющую поверхность достаточно грубым и рутинным способом и часто нуждаются в длительном обучении. Почему бы тогда не вычислить действительно хороший набор значений коэффициентов для экспертной системы? Этот способ не требует обучения, и в случае, когда достаточно трудно классифицировать объекты, вы были бы уверены, что использоваиа наилучшая разделяющая поверхность.

Не желая обескуражить энтузиастов, заметим, что задача разделения объектов является достаточно сложной. Некоторые люди между прочим ссылаются на нее как на задачу разделения, другие — как на задачу классификации, и то и другое на самом деле одинаково. Как правило, в книгах по многомерному анализу вы всегда найдете одну или две главы, посвященные этому вопросу.

Рассмотрим снова задачу разделения двух облачков из мошек. Сделать это просто, если облачка довольно далеки друг от друга, если же нет, то, видимо, не только обучающаяся система, но и любая другая будет делать ошибки, потому что не может быть позиции, в которой разделяющая поверхность могла бы отделить эти две группы объектов.

Можно было бы рассчитать точный центр каждой группы, провести линию между этими центрами и классифицировать каждую мошку в группе в соответствии с тем, по какую сторону от центра данной линии она находится. На практике такой метод является в большинстве случаев основным при разделении объектов. Лучше использовать именно этот метод, чем полагаться на обучающую систему, потому что последняя будет при выработке решения ориентироваться на удаленные объекты, использованные для определения положения разделяющей поверхности.

Система изменяет положение этой поверхности, только если делает ошибки классификации. В конце процесса обучения она допускает ошибки только тогда, когда поверхность расположена вблизи правильной позиции. Поэтому если вновь вернуться к задаче распознавания облачков мошек, то разделяющая поверхность будет колебаться около некоторого положения по прихоти не-большого числа мошек, расположенных на краю облачка. Причем учет их позиции в выборке не делает ее более представительной с точки зрения позиции облака в целом. То же можно сказать и об объектах в *пространстве описания.* Удаленные объекты, не будучи представительными, могут привести к некоторым нежелательным результатам.

Ситуация, с которой мы столкнулись, не однозначна. Поэтому при классификации объектов в n-мерном *пространстве описания* можно было бы в нижеприведенной программе представить размерность N с помощью переменной VAR, причем каждая размерность характеризуется одной переменной. Необходимо зарезервировать память под VAR переменных и затем описать примеры. Допустим, что мы имеем N1 примеров первого рода и N2 примеров второго рода; тогда соответствующий фрагмент программы выглядит следующим образом:

```
10 REM N1 примеров первого рода и N2 примеров второго рода
20 INPUT "N1=";N1
30 INPUT "N2=";N2
40 INPUT "Введите число переменных";VAR
50 DIM MEAN1(VAR),MEAN2(VAR)
```

Затем вычисляется среднее значение (или MEAN) для каждой переменной (их число задается параметром VAR) первого рода, которое запоминается в массиве MEAN1:

```
60 REM Вычисление среднего для N1 значений:
70 FOR I=1 TO VAR
80 MEAN1(I)=0
90 NEXT
100 FOR I=1 TO N1
110 FOR J=1 TO VAR
120 INPUT "Переменная = ";VALUE
130 MEAN1(J)=MEAN1(J)+VALUE/N1
140 NEXT: NEXT
```

Здесь VALUE является значением переменной J для примера I. То же самое можно сделать и для вычисления среднего значения MEAN2 для N2 примеров второго рода. В результате мы будем иметь средние значения для двух родов в массивах MEAN1 и MEAN2:

```
150 REM Вычисление среднего для N2 значений:
160 FOR I=1 TO VAR
170 MEAN2(I)=0
180 NEXT
190 FOR I=1 TO N2
200 FOR J=1 TO VAR
210 INPUT "Переменная = ";VALUE
220 MEAN2(J)=MEAN2(J)+VALUE/N2
230 NEXT: NEXT
```

Теперь постарайтесь классифицировать другой объект с помощью этих средних значений. Допустим, значения переменных для данного объекта хранятся в массиве VALUE. Не забудьте описать этот массив в виде VALUE (VAR).

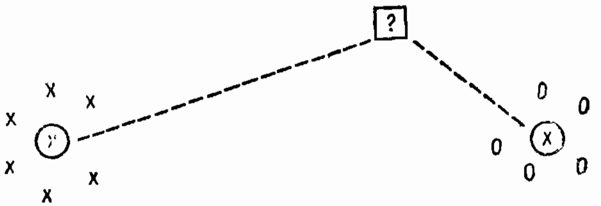


Рис. 3.5. Измерение расстояний между ближайшими средними значениями

Расстояние до нового объекта от этих средних значений (рис. 3.5) вычисляется теперь через DISTANCE 1 и DISTANCE 2:

```
240 REM Вычисление расстояний до VALUE(J) от MEAN1 и MEAN2:
250 DIM VALUE(VAR)
260 DISTANCE1=0: DISTANCE2=0
270 FOR J=1 TO VAR
280 INPUT VALUE(J)
290 DISTANCE1=DISTANCE1+VALUE(J)*MEAN1(J)
300 DISTANCE2=DISTANCE2+VALUE(J)*MEAN2(J)
310 NEXT
```

Дополнительно мы могли бы определить разности между средними значениями и хранить их в новом массиве MID — массиве средних точек:

```
320 REM Альтернативный метод
330 DIM MID(VAR)
340 FOR J=1 TO VAR
350 MID(J)=0
360 NEXT
370 FOR J=1 TO VAR
380 MID(J)=MEAN1(J)-MEAN2(J)
390 NEXT
```

Теперь можно вычислить значение DECISION на основании массивов MID (VAR) и VALUE, а также классифицировать данный объект в зависимости от того, больше или меньше нуля значение DECISION:

```
400 DECISION=0
410 FOR I=1 TO VAR
420 DECISION=DECISION+VALUE(I)*MID(I)
430 NEXT
```

Очевидно, что с точки зрения реализации программы это похоже на метод, используемый в обучающих системах. На практике вполне возможно модифицировать обучающую систему так, чтобы преобразовать ее в рассмотренный метод «ближайшего среднего».

Вместо того, чтобы корректировать правила выработки решений только тогда, когда допущена ошибка, задайте системе какой-нибудь пример и сообщите, к какому классу он принадлежит. Затем измените указанные правила так, чтобы они постоянно сохраняли последнее значение разности между указанными двумя наборами средних значений.

Скажем, если до сих пор было N1 примеров первого класса, а затем вы задали еще один пример того же класса, то

```
440 REM Альтернативный метод
450 FOR J=1 TO VAR
460 MEAN1(J)=MEAN1(J)*N1+VALUE(J)/(N1+1)
470 NEXT
```

В массиве VALUE содержатся значения для нового объекта. Причем удается постоянно обновлять значения массива MEAN1, оставляя в нем последние оценки среднего значения для данной группы. Следовательно, с учетом MEAN1 и MEAN2 можно иметь последние значения для данного правила принятия решения. Например,

```
480 REM Формирование правила принятия решения
490 FOR J=1 TO VAR
500 RULES(J)=MEAN1(J)-MEAN2(J)
510 NEXT
```

После этого осуществляются последовательные решения в зависимости от того, больше или меньше нуля значение DECISION:

```
520 REM Используем это правило для формирования DECISION
530 DECISION=0
540 FOR J=1 TO VAR
550 DECISION=DECISION+VALUE(J)*RULES(J)
560 NEXT
```

Существует много других способов формирования решающих правил, которые можно было бы использовать, но все они зависят от получения подробных сведений о переменных. В то же время если вы хотите узнать, какой из двух методов предпочесть — обучающий алгоритм или подход, основанный на «ближайшем среднем», попробуйте применить тот и другой, а потом решите. Не существует в общем случае однозначного ответа, так как очень многое зависит от самой природы переменных, которые мы выбрали для ввода в систему.

3.6. АПРИОРНЫЕ ВЕРОЯТНОСТИ

Существует, однако, единственная возможность усовершенствовать подход, основанный на ближайшем среднем, который мы постараемся осуществить. Ее можно реализовать на практике, если вы знаете наверняка, что один исход более вероятен, чем другой, независимо от значений переменных. Речь идет о так называемой априорной вероятности каждого исхода, и если Исход 1 встречается в 3 раза чаще, чем Исход 2, то P(Исход 1) = =0,75, а P(Исход 2) = 0,25. В этом случае вы могли бы принять решение, не основанное на ближайшем среднем.

Например, чтобы принять решение, основанное на ближайшем среднем, для массива RULES, содержащего разность между двумя средними, вы просто анализируете значение переменной DECISION и выясняете, оно больше или меньше нуля. Но может случиться так, что значение DECISION окажется близким к нулю для какого-то случая, а вам известно, что, например, Класс 1 значительно более вероятен, чем Класс 2. Тогда вы принимаете решение в пользу выбора Класса 1, даже если вычисленное значение переменной DECISION говорит о необходимости выбора противоположного решения. С точки зрения программиста это равносильно проверке того, больше ли значение DECISION некоторого значения C (обычно $C=0$, хотя будет лучше, если оно будет немного отличаться от 0).

Главная трудность, очевидно, состоит в определении достаточно удачного значения C, отличного от нуля. Если вы — квалифицированный математик и хорошо представляете данные, с которыми работаете, то можно определить некоторое значение C, которое позволит вам сделать вполне определенные суждения относительно поведения системы. В противном случае вам придется вернуться к эксперименту (что, правда, не так уж плохо), и единственное допущение, которое следует сделать, заключается в том, что если отсутствуют типичные примеры для тестирования системы, то вы никогда не будете знать, как она должна работать на практике, поэтому лучше избегать любых ее усложнений.

3.7. РАСШИРЕНИЕ ВОЗМОЖНОСТЕЙ ВЫБОРА

Итак, представьте, что вы взглянули на небо. Люди вокруг вас выкрикивают: «Это птица?, это самолет?». Вы добираетесь до своей экспертной системы, вводите несколько известных вам переменных и через несколько секунд что-то авторитетно произносите. «Неправильно!— восклицают люди вокруг со злорадством в голосе. — Это планер!».

Вы чувствуете себя так же плохо, как и при неверном предсказании погоды. Вернемся к вашей тщательно продуманной экспертной системе. Она предсказывает дождь или, может быть, сухую погоду на завтра. И в том и в другом случае, двигаясь завтра наощупь, кругами в непроницаемом тумане, вы на грани упадка. Туман, как вы знаете, напоминает сырость, но не такую сырость, какая бывает в дождь. Он также чем-то напоминает сухую погоду. «Что же туман в действительности напоминает? — удивляетесь вы, — только туман, и ничего больше». И где-то вы считаете себя обманутым в том, что ваша экспертная система должна была ошибиться, чтобы допустить другой исход.

В общем случае вы хотели бы, чтобы экспертная система реализовывала любое число различных исходов. У вас, возможно, останется чувство необходимости заранее определить эти исходы, но вы все же хотите, чтобы их было больше двух.

Существуют, конечно, несколько способов построения экспертных систем, поэтому не исключены различные исходы. Но в данный момент мы проанализируем только одно возможное расширение системы, которое до сих пор не рассматривали.

Напомним правило для выработки решения. На основе этого правила наша экспертная система принимала решение типа Да/Нет. Теперь же мы обеспечим системе некоторый общий набор правил, чтобы она могла сформировать общий набор возможных исходов.

Нарисуем двухмерный массив, содержащий эти правила. Он напоминает прямоугольную матрицу, в которую помещены наши вероятности. Пусть массив RULES (VAR, OUTCOMES) содержит правила для переменных VAR с исходами OUTCOMES.

Вновь воспользуемся примером с объектами *Птицы/Самолеты:*

	Птица	*Самолет*
Крылья	0	0
Хвост	0	0
Клюв	0	0
Двигатель	0	0
Оперение	0	0
Шасси	0	0

В данный момент «забудем» о *Планере* и с шестью переменными будем иметь RULES (6,2), т. е. OUTCOMES = 2.

Предлагается следующий порядок работы. Каждый столбец массива RULES содержит правило, которое используется, чтобы указать, насколько «твердо» экспертная система уверена в исходе, соответствующем этому столбцу. Степень уверенности измеряется путем вычисления значения DECISION, как и раньше, с помощью массива VALUE, который содержит значения переменных, рассматриваемых в данный момент. Например, для исхода J (J=1 или 2) DECISION=DECISION+ +RULES(I, J) *VALUE(I) для I=1, ..., 6.

Вы заметили, что если начать с RULES(I, J)=0, то получим всегда DECISION=0. Другими словами, оба вероятных исхода в этом случае равны. Теперь в зависимости от того, как вы хотите заполнить массив, вы либо сообщаете экспертной системе, к какой группе принадлежат элементы массива VALUE, либо организуете «угадывание» их, и если угадали неправильно, то поступаете следующим образом. Когда элемент массива VALUE принадлежит исходу *Птица*, складываете значение VALUE со значением элемента в колонке *Птица* и вычитаете его из колонки *Самолет*. Когда же элемент массива VALUE принадлежит к колонке *Самолет*, складываете значение VALUE (I) со значением в колонке *Самолет* и вычитаете его из колонки *Птица*. После добавления элементов к колонке *Птица* получаем:

	VALUE (I)	VALUE (I)	RULES (I,1)	RULES (I,2)
	Птица	*Самолет*	*Птица*	*Самолет*
Крылья . .	1	1	1	−1
Хвост . . .	1	1	1	−1
Клюв . . .	1	0	1	−1
Двигатель .	0	1	0	0
Крылья . .	1	0	1	−1
Шасси . .	0	1	0	0

До сих пор мы представляли значения массива VALUE для колонки *Птица* в массиве RULES. Это не позволяет сформировать решение, так как все значения DECISION одинаковые. Полученный результат очевиден, поскольку мы суммировали значение VALUE(I) со значением в колонке *Птица* и вычитали его из значения в колонке *Самолет*. Теперь если возьмем колонку *Птица*, то система может принять более точное решение. Переменной DECISION присваивается значение +4 с учетом столбца RULES(I, 1) либо −4 с учетом RULES(I, 2). Следовательно, все, что должен делать эксперт, — это выбрать ту колонку правил, в которой находится максимальное значение переменной DECISION.

Однако если мы теперь дадим эксперту значения столбца VALUE(I) для *Самолета*, то он снова выберет *Птицу* как наиболее вероятный исход, поскольку при использовании RULES(I, 1) DECISION=+2, а при использовании RULES(I, 2) DECISION=—2. Итак, наша замечательная экспертная система не может в этом случае даже отличить *Птицу* от *Самолета*.

Сложим теперь значения массива VALUE для *Самолета* с уже имеющимися в одноименной колонке данными и вычтем их же из колонки *Птица*:

	RULES (I, 1)	RULES (I, 2)
	Птица	*Самолет*
Крылья	0	0
Хвост	0	0
Клюв	1	—1
Двигатель	—1	1
Оперение	1	—1
Шасси	—1	1

Если представим значения VALUE(I) для *Самолета* вместе с полученными значениями, то для колонки *Самолет* переменная DECISION будет равна —2, а для колонки *Птица* +2. Следовательно, эксперт выберет правильный исход: *Самолет*. И если вы проверите значения VALUE(I) для колонки *Птица*, то получите для колонки *Птица* DECISION=+2, а для колонки *Самолет* DECISION=—2. Следовательно, наша экспертная система работает так же хорошо, как она работала и до этого.

Фактически если вы вернетесь к уже изученному материалу, то обнаружите, что система функционирует аналогично случаю одномерного массива правил. Поэтому вы, возможно, решите, что затраченные усилия были напрасны. Действительно, если задача заключается лишь в распознавании *Птицы* или *Самолета*, то можно сказать, что игра не стоит свеч.

Возвратимся, однако, к примеру с *Планером* и добавим к имеющимся еще одну колонку RULES(I, J), чтобы учесть его. Получить решение можно так же, как и в случае двух исходов. Вычислим значения переменной DECISION для каждой колонки и выберем ту, в которой находится ее максимальное значение. В случае ошибки или сбоя сообщим системе, какую колонку выбрать.

В этом случае она добавит значения VALUE(I) к нужной колонке, чтобы скорректировать показания последней и вычтет их из любых других колонок, где значение DECISION больше потенциально выбираемого значения или равно ему.

Это позволяет повысить те значения, получаемые от применения правил выбора, которые дают правильные решения, и понизить другие значения, чтобы устранить их влияние на принятие конкретного решения.

Допустим, что мы рассматриваем *Планер*. Он, конечно, похож на *Самолет*, но не имеет *Двигателя* или *Шасси*. Сначала системе задается *Птица* (RULES(I, J)=0), поэтому *Планер* может быть выбран наряду с двумя другими вариантами. Следовательно, значение в колонке для *Планера*, как и для *Самолета*, будет уменьшено. При второй попытке задается *Самолет*, а система отгадала *Птицу*. Это значит, что мы добавили значение VALUE(I) для *Самолета* ко второй колонке и вычли это значение из первой колонки. Теперь у нас есть третья колонка — *Планер*, которая до этого была такой же, как и колонка *Самолет*, следовательно, мы должны вычесть значение VALUE(I) из колонки *Планер*. В результате получаем:

	RULES (I, 1)	RULES (I, 2)	RULES (I, 3)
	Птица	*Самолет*	*Планер*
Крылья	0	0	—2
Хвост	0	0	—2
Клюв	1	—1	—1
Двигатель	—1	1	—1
Оперение	1	—1	—1
Шасси	—1	1	—1

Теперь очевидно, что в таком виде *Планер* как один из возможных исходов никогда не будет выбран, так как все значения в его колонке отрицательны и всегда найдется другое правило, которое будет иметь большее значение для DECISION. Рассмотрим, однако, в качестве объекта именно *Планер:*

	VALUE (I)	RULES (I, 3)
	Планер	*Планер*
Крылья	1	—1
Хвост	1	—1
Клюв	0	—1
Двигатель	0	—1
Оперение	0	—1
Шасси	0	—1

Нет необходимости показывать колонки *Птица* и *Самолет*, потому что каждый раз, когда наступает очередь *Планера*, а система не может угадать, что это за объект, одинаковые значения вычитаются как из колонки *Птица*, так и из колонки *Самолет* (вычитается единица в строках *Крылья* и *Хвост*). Это в любом случае не влияет на способность системы угадывать исходы *Птицы* или *Самолета*. Однако теперь, добавляя +1 в строках *Крылья* и *Хвост* колонки *Планер*, мы увеличиваем соответствующие значения до тех пор, пока значения массива RULES (I, 3) обеспечивают максимальный параметр переменной DECISION. К этому моменту массив RULES выглядит следующим образом:

	RULES (I 1) *Птица*	RULES (I, 2) *Самолет*	RULES (I, 3) *Планер*
Крылья	—1	—1	0
Хвост	—1	—1	0
Клюв	1	—1	—1
Двигатель	—1	1	—1
Оперение, ...	1	—1	—1
Шасси	—1	1	—1

Вы можете проверить, что теперь возможен корректный выбор из трех альтернатив. Система обучена и не делает больше ошибок. Более того, пусть у вас миллион различных исходов и миллион переменных, все равно система будет в процессе обучения манипулировать числами до тех пор, пока не найдет нечто похожее на корректный набор правил выбора возможного исхода из конкретного набора переменных.

3.8 МОЖЕТ ЛИ СИСТЕМА ДЕЛАТЬ ОШИБКИ!

Пока все хорошо. Но каковы шансы, что произойдет ошибка или же обучение в любом случае неадекватно получению корректных исходов? Действительно, такие шансы очень трудно оценить в терминах точных значений вероятностей, поскольку все зависит от типа задачи, поставленной перед экспертом. Сам метод определяется концепцией *линейной сепарабельности*, и это еще одно сочетание терминов, которое следует запомнить.

«Конечно, — произнесете вы, наклонившись у стойки бара, — экспертная система работает превосходно, по-

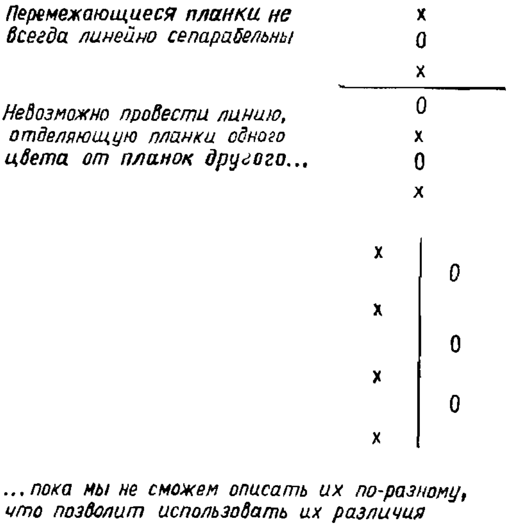


Рис. 3.6. Задача о садовой ограде с планками перемежающегося цвета

скольку строится соответствующая база знаний». В этот момент вы постараетесь сделать глубокомысленную затяжку, пользуясь трубкой для большего эффекта, и добавите: «До тех пор, конечно, пока она работает с линейно-сепарабельными задачами». Неодобрительно усмехнувшись, вы воскликнете: «Но покажите мне задачу, которая не была бы таковой!»

Вернемся к нашим облакам мошек. Пока вы размышляете над тем, как провести линию или поместить разделяющую поверхность между двумя группами мошек, ваша экспертная система будет обучаться различать их. Если же вы не можете этого сделать, то и система не всесильна, так ведь?

Рассмотрим еще один пример — решетчатую ограду, выкрашенную в красный и голубой цвета (рис. 3.6). Допустим, у вас есть сад, огороженный изгородью, планки которой покрашены в перемежающиеся цвета: красный и синий. Можно ли планки этой решетки считать линейно сепарабельными? В состоянии ли вы разделить красные и синие планки изгороди, поместив разделяющую поверхность между ними?

На практике все зависит от того, как вы будете решать эту задачу. Если вы измеряете (или описываете) планку в терминах расстояния вдоль изгороди сада, то ответ отрицательный. Допустим, вы решили описывать их именно таким образом. Тогда вы заметите, что на расстоянии 3 м от начала ограды расположена красная планка, а на расстоянии 3,5 м — синяя. В этом случае можно поместить разделяющую поверхность между ними. Очевидно, что эта поверхность разграничивает только эти две планки, но не разделяет другие красные и синие планки. При таком подходе ограда, описанная выше, не является линейно-сепарабельной.

Если, однако, вы прибьете планки красного цвета с одной (внутренней) стороны перекладины, а синего цвета — с другой (внешней), перемежая их вдоль изгороди, то удастся довольно легко разделить их по цвету, поместив разделяющую поверхность вдоль изгороди по центру перекладин. Ограда останется приблизительно той же самой, но планки для фиксации места их расположения вдоль изгороди сада можно просто перенумеровать все подряд. Поскольку цвета планок по условию задачи чередуются, все четные планки будут синего цвета, а все нечетные — красного. Если таким образом описать планки ограды, то они будут линейно-сепарабельны и экспертная система сможет обучиться отличать, какая из планок красная, а какая синяя.

Вся сложность в том, что положение планок ограды практически не изменилось, вы просто описали их по-другому. В этом суть дела. Следует так выбирать переменные, чтобы их использование позволяло экспертной системе распознавать различные исходы. Это, конечно, не очень удобная точка зрения, если говорить честно, поскольку так или иначе необходимо решить поставленную задачу. Если вы, например, желаете, чтобы экспертная система предсказывала, пойдет или не пойдет завтра дождь, то вы не дадите ей для обработки результаты футбольного матча. Ясно, что предоставление знаний о результатах футбольного матча не поможет системе предсказать погоду, следовательно, и решаемая проблема не станет линейно-сепарабельной. (Кто-то, возможно допускает, что результаты футбольных матчей иногда говорят о погоде, что дает возможность извлечь какую-то информацию, полезную для ее предсказания, но едва ли можно рассчитывать на четкий ответ в этом случае.)

```
10 CLS
20 INPUT "ВВЕДИТЕ ЧИСЛО ПЕРЕМЕННЫХ"; VAR
30 DIM VALUE(VAR), VAR$(VAR)
40 PRINT: PRINT "НАЗОВИТЕ ИХ ИМЕНА"
50 FOR I=1 TO VAR
60 PRINT "ПЕРЕМЕННАЯ"; I;" –";: INPUT VAR$(I)
70 NEXT: PRINT
80 INPUT "ВВЕДИТЕ ЧИСЛО ВОЗМОЖНЫХ ИСХОДОВ";
   OUTCOMES
90 DIM OUTCOMES$(OUTCOMES), RULES(VAR,OUTCOMES),
   DECISION(OUTCOMES),SCORE(OUTCOMES)
100 PRINT: PRINT "НАЗОВИТЕ ЭТИ ИСХОДЫ"
110 FOR I=1 TO OUTCOMES
120 SCORE(I)=0
130 PRINT "ИСХОД"; I;" –"; INPUT OUTCOMES$(I)
140 NEXT
150 CLS
160 PRINT: PRINT "НАЧАЛО СЕАНСА ОБУЧЕНИЯ:"
170 PRINT "ВЫ ДОЛЖНЫ ВВОДИТЬ ЗНАЧЕНИЯ ПЕРЕМЕННЫХ."
180 PRINT "Я БУДУ УГАДЫВАТЬ ВОЗМОЖНЫЙ ИСХОД."
190 PRINT "ВЫ ЖЕ ДОЛЖНЫ БУДЕТЕ СКАЗАТЬ: ПРАВ Я ИЛИ НЕТ."
200 DECISION= –10000
210 FOR I=1 TO OUTCOMES
220 DECISION(I)=0
230 NEXT: PRINT
240 FOR I=1 TO VAR
250 PRINT "ПЕРЕМЕННАЯ"; I;"(";VAR$(I);") –";: INPUT VALUE(I)
260 NEXT
270 FOR I=1 TO VAR
280 FOR J=1 TO OUTCOMES
290 DECISION(J)=DECISION(J)+VALUE(I)* RULES(I,J)
300 NEXT
310 NEXT
320 FOR I=1 TO OUTCOMES
330 IF DECISION(I)>DECISION THEN DECISION=DECISION(I): BEST=I
340 NEXT
350 PRINT "ЭТО, ВЕРОЯТНО, ИСХОД"; BEST;" ("; OUTCOMES$
    (BEST); ")<y/n> ";: INPUT A$
360 IF A$="Y" OR A$="y" THEN SCORE(BEST)=1: SCORE=0:
    FOR I=1 TO OUTCOMES: SCORE=SCORE+SCORE(I): NEXT:
    IF SCORE=OUTCOMES THEN PRINT "Я ДОСТИГ СОВЕРШЕНСТ-
    ВА!": END ELSE GOTO 150
370 FOR I=1 TO OUTCOMES
380 PRINT I;" "; OUTCOMES$(I)
```

```
390 NEXT
400 IMPUT "КАКИМ ДОЛЖЕН БЫТЬ ИСХОД"; CORRECT
410 FOR I=1 TO OUTCOMES
420 IF DECISION(I))=DECISION AND I<>CORRECT THEN FOR J=1
    TO VAR: RULES(J, I)=RULES(J, I)-VALUE(J): NEXT
430 NEXT
440 FOR J=1 TO VAR
450 RULES(J,CORRECT)=RULES(J,CORRECT)+VALUE(J)
460 NEXT
470 PRINT "Я ПОЛУЧИЛ"; SCORE;" ПЕРЕД ТЕМ, КАК СДЕЛАЛ
    ОШИБКУ!"
480 PRINT
490 PRINT "ДЛЯ ПРОДОЛЖЕНИЯ НАЖМИ ЛЮБУЮ КЛАВИШУ"
500 X$="": WHILE X$="": X$=INKEY$: WEND
510 FOR I=1 TO OUTCOMES
520 SCORE(I)=0
530 NEXT
540 GOTO 150
```

Рис. 3.7. Программа решения линейно-сепарабельной задачи

Кроме собственно проблемы распознавания (т.е. случаев линейной сепарабельности) важным также является то, как долго экспертная система будет обучаться достаточному набору правил. Наилучший способ — это задавать системе все возможные примеры до тех пор, пока она перестанет делать ошибки, если число возможных исходов достаточно мало́. В противном случае требуется ввести в систему по крайней мере по паре примеров для каждого возможного исхода и внимательно наблюдать за ее поведением, пока оно не установится.

Программа, представленная на рис. 3.7, функционирует на основе метода, который мы только что описали, однако она, кроме того, осуществляет еще и проверку полученного исхода: выясняет, не произошло ли ошибки.

Сначала следует определить, сколько переменных будет использоваться в процессе выработки решения и сколько при этом может быть исходов.

С помощью значений VAR и OUTCOMES в программе описаны следующие массивы: VALUE (VAR) — для хранения значений текущих переменных; VAR$ (VAR) — для хранения имен переменных; OUTCOMES$ (OUTCOMES) — для хранения имен исходов; RULES

(VAR, OUTCOMES) — для хранения правил переменной VAR и возможных исходов OUTCOMES; DECISION (OUTCOMES) — для хранения значений, вычисленных для данного VALUE (I) с учетом правил RULES (I, J), возможных исходов OUTCOMES. Имена этих переменных и исходов также вводятся в программу.

Со строки 150 начинается процесс обучения. Значения для VALUE (I) вводятся в программу, и с использованием RULES(I, J) вычисляются значения DECISION (J) для всех исходов OUTCOMES. Массив DECISION сортируется для нахождения максимального значения, и делается предположение, что максимальное DECISION(I) = DECISION (BEST) указывает на правильный исход OUTCOMES$ (BEST).

Если это правильный исход, но поведение системы еще недостаточно совершенно, то программа возвращается к строке 150 для продолжения процесса обучения на другом примере. Если же ответ неверен, то все возможные исходы отображаются на дисплее и необходимо определить, какой из них правильный. Этому исходу присваивается значение OUTCOMES$ (CORRECT).

Затем система скорректирует параметр RULES (I, J), соответственно вычитая VALUE (J) из каждого правила, которое больше или равно DECISION (BEST), и, наконец, добавляя значения VALUE (J) к тому правилу, которое правильно, т. е. к RULES (J, CORRECT).

Откорректировав массив RULES, программа затем вновь возвращается к строке 150 для обработки другого примера.

В процессе обучения распознавание системы должно улучшаться, а в некоторых случаях, таких как пример *Птица—Самолет—Планер*, быть корректным во всех последующих случаях. В программе имеется встроенный счетчик, используя который, она может судить о том, что ее ответы стали достоверными.

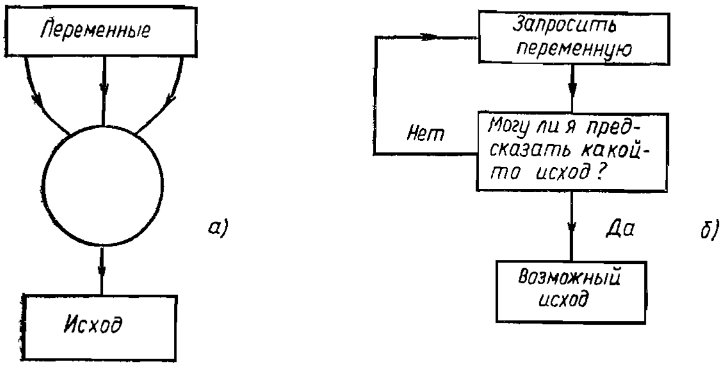
Глава 4. УЛУЧШЕНИЕ ВАШЕЙ ЭКСПЕРТНОЙ СИСТЕМЫ

4.1. ПАРАЛЛЕЛЬНЫЕ И ПОСЛЕДОВАТЕЛЬНЫЕ РЕШЕНИЯ

Мы достигли теперь такой степени совершенства, что разработали свою экспертную систему. После определенной процедуры обучения она должна вырабатывать для нас решения в той предметной области, которую мы сами выбрали, причем до тех пор, пока мы не начали проверять содержимое массива RULES нам нет никакой необходимости знать, как она это делает. Этого вполне достаточно для большинства людей, чтобы чувствовать себя вполне удовлетворенным. Кроме, конечно, неизбежных скептиков. «Если это была бы настоящая экспертная система, — заныли бы они, — она работала бы не так. Она работала бы по-другому». И указали бы на то, что каждый раз для получения экспертного решения им нужно ввести ответы на целый набор вопросов сразу, прежде чем экспертная система соблаговолит сделать элементарную работу. «Реальная система, — будут спорить они, — задаст вам всего один, ну, может быть, два вопроса, а затем, в зависимости от ответа, либо задаст еще несколько вопросов, либо выскажет свое мнение». Она не станет спрашивать вас сразу обо всем.

Имеет смысл подчеркнуть, что во многих системах используются последовательные, а не параллельные процедуры принятия решения. До сих пор мы имели дело с параллельными процедурами, где сначала запрашивается вся информация об объекте, а затем вырабатывается решение.

Последовательная процедура всегда принимает решение (рис. 4.1) на основании последних сведений, которые она получила. Например, в нашем примере с прогнозом погоды последовательная процедура могла бы решить, что гораздо важнее при выработке мнения относительно завтрашней погоды знать: был ли дождь сегодня? Она задала бы этот вопрос и либо стала бы искать дополнительно больше сведений о погоде, либо выдала бы свое суждение о ней. Одна из трудностей в примере с предсказанием погоды состоит в том, что мы не знаем наверняка, на основании каких причин можно предположить, что завтра пойдет дождь. Совсем иначе обстоят дела в при-



Рит. 4.1. Параллельное *(а)* и последовательное *(б)* решения

мере *Птица—Самолет*: нам известна по определению разница между *Птицами* и *Самолетами*. Рассмотрим работу последовательной процедуры для этой задачи.

Вам необходимо определить, что это за объект — *Птица* или *Самолет*.

«Есть ли, — посасывая трубку, спрашивает эксперт, — у него перья?» «Действительно, есть», — допускаете вы. И сразу же великий мыслитель делает заключение, что это *Птица*. Как видите, совершенно нет необходимости рассматривать все шесть переменных.

Допустим, что вы задумались над объектом, но не определили, что он может летать. На рис. 4.2 показаны параллельный и последовательный подходы при выработке решения по этому вопросу.

Вас спросят: «Имеет ли объект *Крылья*?» Вы предположите, что Да, и мгновенно окажется, что это противоречит предыдущему ответу, касающемуся *Оперения*. Итак, оказался вполне оправданным последовательный подход.

В общем случае последовательный подход к решению задачи позволяет задавать меньше вопросов об объекте, чем параллельный. Поэтому, по крайней мере, сначала он кажется более привлекательным. Однако несмотря на то что иногда очевидна необходимость применения последовательного подхода, это не совсем так в случае предсказания погоды. Ибо даже если вы действительно хороший эксперт и знаете все относительно этого прогноза,

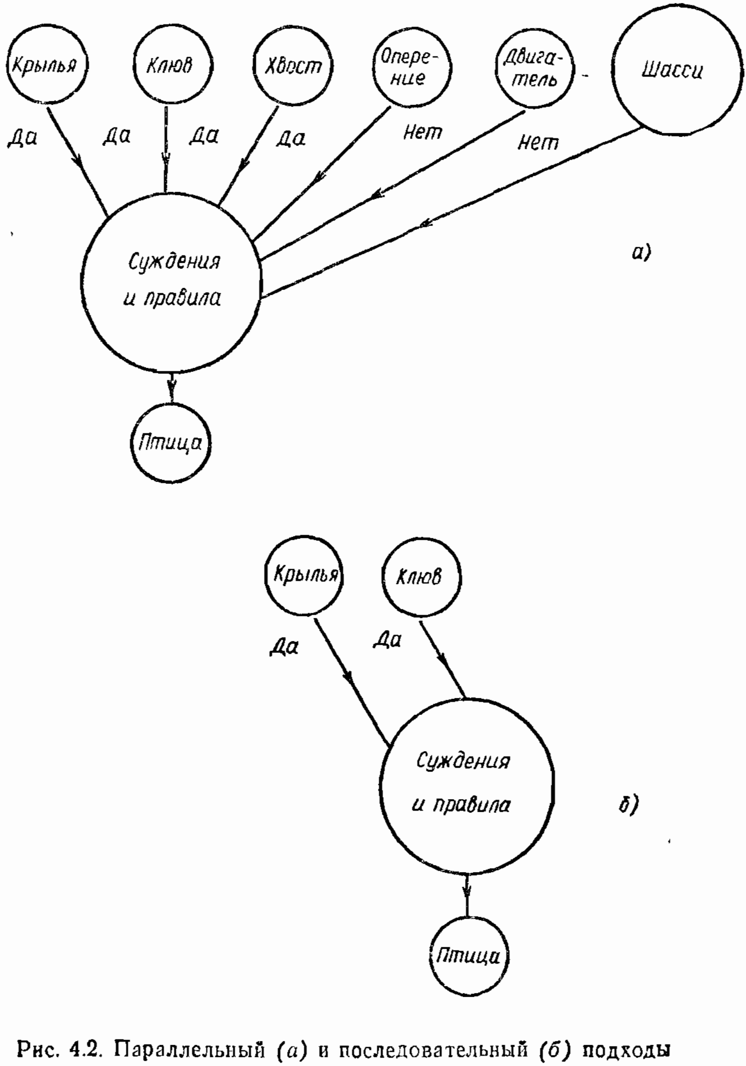


Рис. 4.2. Параллельный *(а)* и последовательный *(б)* подходы

вы все же никогда не будете абсолютно уверены в том, пойдет завтра дождь или нет. Поэтому опытный эксперт должен собрать все возможные сведения о погоде прежде, чем сделать какое-либо заключение. Именно эти операции и осуществляет параллельная процедура.

Поскольку абсолютно никто не может быть уверен в общем случае в том или ином исходе, эксперты собирают всю информацию, которая им доступна, а затем на ее основе делают наилучшее предсказание. Последовательная процедура напоминает последовательность «скачков» к правильным выводам, наша же экспертная система не делает этого.

Если вы поразмыслите над вышеизложенным какое-то время, то поймете, что параллельная процедура не уступает по своим достоинствам последовательной процедуре. При параллельной процедуре используется вся информация для выработки решения и, следовательно, система приходит к наиболее вероятным выводам. С помощью последовательной процедуры можно путем пошагового прохождения через всю возможную информацию прийти к тем же самым выводам, но в процессе поиска решения последнее вырабатывается поэтапно, причем так же эффективно, как и при параллельной процедуре.

Не всегда между двумя этими процедурами существует большая разница. Предположим, что наша экспертная система запрашивает информацию о переменных, стирая информацию о них с экрана после каждого запроса. На первый взгляд, это последовательная процедура, однако если система не делает никаких вычислений в промежутке между запросами, то она фактически является параллельной. Мы могли бы ввести требуемую информацию, если после ввода некоторого основного блока информации экспертная система мгновенно принимала бы решение и отбрасывала остальную, не введенную информацию. Но вопрос в том, как написать такую программу? Конечно, если мы знаем предметную область запросов, которые наша система собирается обрабатывать, то можем написать для этого специальную программу. Однако в нашем случае хотелось бы иметь экспертную систему общего назначения. Если известны точные вероятности для каждого исхода, то так же не сложно написать нужную программу.

Например, поскольку эксперт собирает информацию последовательно, он мог бы просмотреть все возможные исходы и найти наиболее вероятный из них. Затем он был бы в состоянии проверять все возможные исходы, чтобы выяснить: могут ли они после введения текущей информации стать более вероятными, чем тот, который в данный момент наиболее вероятен? Если такого не происходит, то эксперт принимает решение в данный момент и не просматривает другие варианты дальше. В противном случае необходима дополнительная информация.

Рассмотрим пример *Птица—Самолет* и введем истинные вероятности различных исходов, вместо того чтобы полагаться на обучение экспертной системы. В этом случае мы могли бы сообщить эксперту, что если объект имеет *Оперение*, то это *Птица* с вероятностью 1 и *Самолет* с вероятностью 0. Другими словами, как только предъявлен признак *Оперение*, эксперт должен сделать вывод о том, что *Птица* является единственно возможным исходом, и прекратить задавать любые вопросы. Но мы поступим иначе.

Из примеров, на которых мы обучали эксперта, он усвоил, что единственным исходом для *Птицы* было наличие *Оперения*. Но ведь ему известна также и дополнительная информация. Скажем, исходу *Птица* соответствует также переменная *Клюв*. Значит ли это, что если объект имеет как *Оперение*, так и *Клюв*, то это *Птица*? Или этот объект все же *Птица*, если он имеет хотя бы один из этих признаков? И еще: значит ли, что в примерах, рассмотренных до сих пор, случайно оказалось, что *Птицы* имели *Клювы* и *Оперения* и не исключено также появление некой *Птицы*, которая ничего этого не имеет? Другими словами, является ли отмеченный факт абсолютно достоверным с вероятностью 1 или описанное событие имеет близкую, но отличную от 1 вероятность и может не наступить в каком-либо последующем опыте?

Теперь, имея в виду пример *Птица—Самолет*, вы могли бы сказать, чего стоит ждать, а чего нет. Ну, а относительно примера с *Погодой*? В этом случае ничего не происходит с вероятностью 1. Здесь все достаточно неопределенно, и у вас нет четкой позиции, основанной на определенных фактах, чтобы объяснить впоследствии все, что необходимо экспертной системе. Вы сами не знаете нужных ответов, исключая единственный способ: ожидание следующего дня в надежде увидеть как же все обернулось с погодой в действительности. Как вы это объясните экспертной системе?

В частных случаях последовательная процедура разрабатывается таким образом, чтобы сократить время вычислений и исключить ошибки, формируя при этом такие суждения, которые так же хороши, как и при параллельной процедуре. Но в общем случае параллельная

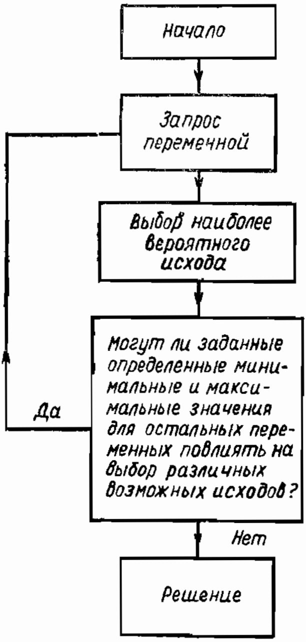


Рис. 4.3. Последовательный алгоритм экспертной системы

процедура позволяет получить более точное решение по сравнению с последовательной процедурой и программы, использующие подобную процедуру, полнее отвечают требованиям, предъявляемым к экспертным системам общего назначения, так как основываются на меньшем числе предположений. Теперь давайте рассмотрим, как можно превратить нашу экспертную систему в систему, основанную на последовательной процедуре, как это показано на рис. 4.3.

Допустим, система отображает на экране имя первой переменной и запрашивает ее значение VALUE. Затем вычисляются значения DECISION для всех исходов OUTCOMES с помощью лишь этой одной переменной. Дальше просматриваются полученные значения DECISION, из них выбирается наибольшее и решается вопрос о том, может или нет оно быть корректным исходом. Здесь используется метод просмотра всех правил в массиве RULES с целью определить, смогут ли какие-то из них увеличить значение переменной DECISION настолько, чтобы оно превысило текущее значение, соответствующее благоприятному исходу.

Однако в процессе вычислений приходится задаваться значениями VALUE (I), которые пока неизвестны, для проведения описанного предсказания. Если задаваемые значения действительно случайны, то это вносит дополнительную неопределенность в ситуацию, и без того уже достаточно неопределенную. Если система собирается провести такое предсказание, то ей должны быть известны минимальные и максимальные значения VALUE (I) для каждой переменной. Если имеется возможность сообщить экспертной системе эти значения, то она может, по крайней мере, отбросить некоторые варианты, а если она сможет отбросить все варианты кроме одного, то она примет окончательное решение. Что же произойдет, если система примет неверное решение?

Когда системе дана корректная информация относительно максимальных и минимальных значений переменных, она способна принять неверное решение, только если массив RULES содержит неверные значения. Это означает также, что массив RULES нуждается в корректировке по всем переменным. Следовательно, программа должна иметь возможность переключаться на параллельную процедуру для получения значений всех переменных и корректировки после этого значений RULES.

В ходе построений персональной экспертой системы делать это не обязательно. Особенно если вам не нравится составлять программы. При желании сделать экспертную систему более интересной или более простой для пользователя описанная процедура, видимо, заслуживает внимания. Будет или нет разница в том, что делает экспертная система, зависит от конкретных значений переменных (минимальных и максимальных), которые вы ей сообщите. Они должны формировать достаточно широкое поле допуска и отображать действительные пределы изменения входных данных — иначе система допустит ошибку. С другой стороны, если поле допуска слишком широко, то это вряд ли поможет эксперту в выборе нужного решения. В этом случае решение задачи очень напоминает параллельную процедуру, но вы придете к нему, затратив слишком много дополнительных усилий!

4.2. ДОБАВИМ НЕМНОГО ЗДРАВОГО СМЫСЛА

Коль скоро вы не можете сообщить экспертной системе дополнительно что-то важное об условиях задачи, которую предстоит решить, параллельная процедура, описанная выше, позволяет получить более точное решение на сегодняшний день. Однако при коррекции минимальных и максимальных значений переменных при определенных обстоятельствах система не в состоянии «отбросить» ряд вопросов в ходе анализа полученной информации.

Давайте рассмотрим этот метод не торопясь, шаг за шагом с помощью языка БЕЙСИК. Допустим, мы описали два массива MINI (VAR) и MAXI (VAR) для хранения минимальных и максимальных значений, где VAR — длина массива. Организовать ввод элементов этих массивов MINI (I) и MAXI (I) в компьютер перед началом цикла обучения можно либо поэлементно, либо в виде отдельного блока. Требуется также описать массивы VARFLAG (VAR) и POSSIBLE (VAR). После этого начнем, пожалуй, цикл обучения.

```
10 REM Значения, необходимые для VAR и OUTCOMES
20 VAR=6: OUTCOMES=3: REM Конкретный пример:
30 DIM VARFLAG(VAR),POSSIBLE(VAR),MINI(VAR),MAXI(VAR),
   DECISION(OUTCOMES),VALUE(VAR),RULES(VAR,OUTCOMES)
40 REM Минимальные и максимальные значения должны быть загружены
   в массивы MINI(VAR) и MAXI(VAR)
50 REM Установка значений элементов массива флажков VARFLAG,
   фиксирующего на данный момент, какие переменные уже были
   использованы в процессе принятия решения
60 FOR I=1 TO VAR
70 VARFLAG(I)=1: REM Значение 1 означает, что данная переменная
   еще не была использована
80 NEXT
90 REM Очистка массива DECISION
100 FOR J=1 TO OUTCOMES
110 DECISION(J)=0
120 NEXT
130 REM Ввод значений для каждого элемента массива VALUE:
140 FOR I=1 TO VAR
150 INPUT VALUE(I)
160 VARFLAG(I)=0: REM Флажок установлен на 0, так как введено
    какое-то значение
170 REM Вычисление и коррекция массива DECISION
180 FOR J=1 TO OUTCOMES
190 X=VALUE(I)*RULES(I,J): DECISION(J)=DECISION(J)+X
200 NEXT
210 REM Нахождение максимального значения DECISION(J)
220 DECISION=0
230 FOR J=1 TO OUTCOMES
240 IF DECISION(J)>DECISION THEN DECISION=DECISION(J): BEST=J
250 NEXT
260 REM Размещение текущих значений массива DECISION в массиве
    POSSIBLE
270 FOR J=1 TO OUTCOMES
280 POSSIBLE(J)=DECISION(J)
290 NEXT
300 REM Вычисление возможных значений, которые массив POSSIBLE
    мог бы иметь, используя неизвестные в данный момент
    переменные
310 FOR J=1 TO OUTCOMES
320 FOR K=1 TO VAR
330 IF VARFLAG(K)=1 THEN IF RULES(K,J)>RULES(K,BEST) THEN
  POSSIBLE(J)=POSSIBLE(J)+(RULES(K,J)-RULES(K,BEST))*MAXI(K) ELSE
  POSSIBLE(J)=POSSIBLE(J)-(RULES(K,BEST)-RULES(K,J))*MINI(K)
340 NEXT: NEXT
```

Программа просматривает все переменные, которые еще не использовались. С учетом известных минимальных и максимальных значений она изменяет параметры POSSIBLE (J) и пытается найти другой исход — BESTPOSS, значение которого могло бы превысить значение, хранящееся в POSSIBLE (BEST). Программа осуществляет эту операцию в предположении, что все остальные переменные могут служить альтернативой этому текущему выбору. Поскольку требуется определить, можно ли изменить текущий выбор, программа работает с разностью RULES (K, J)—RULES (K, BEST), вместо того чтобы работать с переменными RULES (K, J) и RULES (K, BEST) по отдельности.

```
350 MAXPOSS=POSSIBLE(BEST)
360 FOR J=1 TO OUTCOMES
370 IF POSSIBLE(J)>=MAXPOSS THEN MAXPOSS=POSSIBLE(J):
    BESTPOSS=J
380 NEXT
390 IF BESTPOSS=BEST THEN STOP: REM ВСЕ ИСТОЧНИКИ
    УКАЗЫВАЮТ НА ЭТОТ ИСХОД
400 REM ДО СИХ ПОР НЕТ СОГЛАСОВАННОГО МНЕНИЯ,
    ПОЭТОМУ ТРЕБУЕТСЯ ДРУГАЯ ПЕРЕМЕННАЯ
410 NEXT I
```

Если после всего этого все-таки будет принято неверное решение, то просматривается массив VARFLAG, для того чтобы выяснить, какие элементы массива VALUE еще не были введены. Затем вводятся нужные значения. Далее продолжается выполнение программы и изменяется массив RULES.

Определение значений массива путем просмотра всех его элементов — возможно, не самый лучший метод с точки зрения здравого смысла. Например, если все значения массива RULES равны для данной переменной, то присвоение какого-то нового значения этой переменной не представляет никакой дополнительной информации для экспертной системы и его можно не учитывать. В этом случае целесообразно проверить только небольшой фрагмент программы, прежде чем затребовать переменную, если все RULES (I, J) равны для нее. Если это так, то экспертная система могла бы перейти к проверке следующей переменной, не запрашивая значения текущей переменной.

Теперь необходимо выяснить, какая переменная должна быть рассмотрена на следующем шаге. Некоторые переменные оказывают существенно большее воздействие на выбор того или иного исхода по сравнению с другими. До сих пор мы пользовались только методом разбиения переменных на две группы — те, что могут повлиять на выбор данного исхода, и те, что не влияют на него. При дальнейшем написании программ и определении значений переменных, которые, как нам кажется, будут в большей степени влиять на возможный исход, может возникнуть ситуация, когда потребуются все переменные, влияющие на данный исход. Следовательно, единственное преимущество, полученное в этом случае, заключается в изменении порядка запроса значений переменных, а не в уменьшении их общего числа, которое фактически следует рассмотреть.

Один из таких методов состоит в том, чтобы просмотреть в первую очередь те переменные, значения которых имеют наибольший диапазон изменения от максимального до минимального значений (полагаем, что мы в состоянии представить такие значения). При этом предполагается, что раз эти переменные имеют максимальный диапазон изменения, то они являются важнейшими в процессе принятия решения. Очевидно, что в примере *Птица—Самолет* минимальные и максимальные значения (0 и 1), соответствующие отсутствию или наличию тех или иных признаков объекта, вряд ли чем могут помочь в этом смысле.

Однако, рассматривая признаки, определяющие погоду, можно убедиться в наличии у них вариаций минимумов и максимумов. Облачный покров меняется от 0 до 10 (условных единиц) в зависимости от того, ясно или облачно (в пределе — десятки облаков); уровень осадков колеблется от 0 до 2 в зависимости от того, нет осадков или их выпало 50 мм.

Сначала кажется, что облачный покров меняется больше всего, т. е. именно эту переменную нужно проверить в первую очередь. Это справедливо, пока мы не стали уровень осадков измерять в миллиметрах; тогда он будет колебаться от 0 до 50 мм. Теперь уже уровень осадков является той переменной, которую нужно проверять в первую очередь. Данный подход к разработке метода функционирования экспертной системы не очень хорошо «привязан» к самой системе, так как даже если у нас появляется возможность обрабатывать широкий спектр различных типов данных, мы не можем однозначно заключить, как эти данные обработать.

Если вы строите свою систему так, чтобы она почти не зависела от конкретной предметной области, в которой предполагается приобретать экспертный опыт, вероятнее всего, появятся какие-то накладки и трудности. Вы почувствуете себя в роли приспосабливающего некую общую систему к конкретным условиям использования, заставляя ее делать запрос относительно конкретных переменных в самом начале, отбросив другие переменные, основываясь на ваших собственных знаниях. Но раз уж вы сделаете это, то вероятно, увидете, что ваша экспертная система не сможет обучиться другим задачам с такой же готовностью. Она не будет больше экспертной системой общего назначения — даже если «приспособится» к решению тех задач, которые вы в нее заложили. Все зависит от конкретного назначения системы. Когда цель ясна, есть хороший шанс построить более удачную систему по сравнению с той, что вы получили бы, пользуясь каким-то общим методом, подсказанным со стороны.

Описывая указанный метод, мы, наверно, потерпим неудачу, заставляя систему искать вначале «лучшую» переменную. Мы можем сделать это, лишь анализируя степень изменения, которую она может вызвать в данном правиле принятия решения.

Если описать массив RULEVALUE (VAR) для хранения значения этой вариации (численные оценки данного правила) каждой из VAR переменных, то получим

```
10 FOR I=1 TO VAR
20 FOR J=1 TO OUTCOMES
30 RULEVALUE (I)=RULEVALUE (I)+
   +ABS ((MINI (I)—MAXI (I) * RULES(I, J))
40 NEXT : NEXT
```

В этом фрагменте программы мы находим разницу между минимальным и максимальным значениями для каждой переменной и умножаем ее на абсолютное значение правила для данной переменной. Затем организуем в массиве RULEVALUE поиск максимального значения этого правила. Это как раз та переменная, которая, как мы полагаем, наиболее важна, поэтому вводим ее в компьютер в первую очередь. Если система вызывает следующую переменную, то это будет переменная со следующим значением в массиве RULEVALUE и т. д.

Применимость метода без повторения уже сделанных комментариев зависит от точности минимальных и максимальных значений переменных и того, насколько удачно выбраны правила, разработанные до сих пор конкретной системой.

Очевидно, что в начале процесса вычислений RULES (I, J) = 0, поэтому система не может сделать заключение о том, какие переменные важны в решении этой задачи, а какие нет. Поможет ли приобретенный опыт улучшить данную систему, зависит от конкретной постановки задачи.

Существует еще одна возможность, чтобы уточнить, какую переменную запрашивать. Мы знакомы с методом выбора такой переменной с максимальным значением в массиве RULEVALUE. Предположим, что она уже введена в ЭВМ, тогда VARFLAG (I) = 0, т. е. эту переменную можно больше не учитывать. Система контролирует, может ли она получить недвусмысленный исход. Поскольку это исключено, система начинает поиск следующей переменной в массиве RULEVALUE.

При проверке возможных решений в массиве POSSIBLE многие из них можно также не учитывать, поскольку они меньше значения DECISION, уже выбранного системой для этого как наиболее вероятный исход.

Допустим, что мы описали массив OUTFLAG (OUTCOMES) для хранения исходов OUTCOMES и используем его точно так же, как массив VARFLAG (VAR) для переменных VAR. В качестве начальных условий установим его элементы равными 1, а любой из элементов массива OUTFLAG (J) — равным 0, как только обнаружится, что этот исход никогда не будет выбран. Поскольку мы имеем дело с множеством таких исходов, исключим те из них, которые стали ненужными («мертвыми»).

Например:

```
10 FOR I=1 TO VAR
20 FOR J=1 TO OUTCOMES
30 RULEVALUE (I)=RULEVALUE (I)+
   +ABS ((MINI (I)—MAXI (I))* RULES (I, J))*
   *VARFLAG (I)* OUTFLAG (J)
40 NEXT : NEXT
```

Этот фрагмент программы ничего не меняет, если все переменные и исходы остаются активными, так как все элементы VARFLAG (I) и OUTFLAG (J) равны 1. Но если любая переменная уже использована или установлено, что какой-то исход не осуществим, тогда VARFLAG (I) или OUTFLAG (J) равны 0 и значение RULEVALUE (I) изменяется под действием этой переменной или исходов. Мы рассматриваем здесь частную задачу (что делать дальше?), возникшую в результате того, что некоторые переменные не являются больше активными.

Значениями массива RULEVALUE можно воспользоваться и в том виде, в котором они вычисляются. Вы, возможно, захотите использовать квадратичные значения RULES (I, J), а не их модули. Это зависит от того, что вам больше нравится. Допустим, например, вы хотите выбрать одну из двух переменных, которые имеют два исхода. Тогда значения RULES (I, J) выглядят следующим образом:

	Исход 1	*Исход 2*
Переменная 1	2	2
Переменная 2	1	3

Если бы обе переменные имели те же самые минимальные и максимальные значения, то они внесли бы одинаковый вклад в RULEVALUE, так как каждая из этих «добавок» была бы равна 4. В этом примере предпочтительно исследовать в первую очередь *Переменную 2*, потому что она имеет большее влияние на исходы (фактически же это единственная переменная, которая имеет какое-то влияние на исходы в данном примере). Если бы вы использовали квадраты значений RULES (I, J), то получили бы

$$\text{RULEVALUE } (1) = 2^2 + 2^2 = 8;$$

$$\text{RULEVALUE } (2) = 1^2 + 3^2 = 10,$$

т. е. прежде всего необходимо проанализировать *Переменную 2*. Естественно, что *Переменная 1* в любом случае не обрабатывалась бы, поскольку она дает одинаковое значение для каждого исхода.

Предположим теперь, что вы имеете еще и третью переменную, т. е.

	Исход 1	*Исход 2*
Переменная 3	0	4

Так как эта переменная вносит бо́льший, чем *Переменные 1 и 2*, вклад в значение RULEVALUE (3), она будет выбрана первой (затем вторая и первая). Если же оперировать с абсолютными значениями RULES (I, J) в RULEVALUE (I), то не всегда можно обнаружить какую-то разницу между этими тремя переменными, хотя вы, возможно, интуитивно чувствуете, что они по-разному влияют на решение рассматриваемой задачи.

Поскольку отсутствуют ограничения на применение квадратичных зависимостей в предыдущем методе, то нет их и теперь. Возведение в квадрат значений элементов массива RULES (I, J) подчеркивает любые существующие при их формировании различия. Причем чем более высокие ступени мы используем, тем больший акцент получаем. Применение нечетных степеней (в отличие от абсолютных значений) позволяло бы сохранить знаки (положительный или отрицательный) переменных, а манипуляции с другими функциями способствовали бы нестандартному поведению некоторых предлагаемых программ.

Существует, однако, и более сложный способ выявления степени влияния переменных.

Рассматривается сумма квадратов отклонений от среднего значения для каждой переменной. Сначала вычисляется среднее значение оценок, полученных в результате применения правил для каждой переменной, а затем квадратическое отклонение этих оценок от полученных средних значений. Например:

```
10 FOR J=1 TO OUTCOMES
20 MEAN=MEAN+(RULES (I, J)/OUTCOMES)*
   * VARFLAG (I) * OUTFLAG (J)
30 NEXT
40 FOR J=1 TO OUTCOMES
50 RULEVALUE (I)=RULEVALUE (I)+
   +((RULES (I, J)-(MEAN) ^2)* ABS (MINI (I)-
   -MAXI (I))* VARFLAG (I)* OUTFLAG (J)
60 NEXT
```

В этом фрагменте программы определяются значения RULEVALUE (I) как суммы квадратов отклонений относительно RULES (I, J) для переменной I по каждому возможному исходу. Необходимо учитывать также параметры VARFLAG (I) и OUTFLAG (J) при вычислении как средних значений, так и сумм квадратов, если вы хотите учесть заданные переменные и исходы, которые уже больше не рассматриваются. Особенность метода состоит в том, что удается воспользоваться следующими правилами:

	Исход 1	*Исход 2*
Переменная 1	1	2
Переменная 2	1	—2

Вроде бы *Переменная 2* «важнее» ввиду большей разницы, которую она может создать, со значением конкретного исхода. Однако абсолютное значение RULES (I, J) не позволяет обнаружить какую-то разницу между двумя этими переменными, как, впрочем, и возведение в квадрат значений RULES (I, J). Оценки, соответствующие *Переменной 2*, больше изменяются относительно среднего значения. Именно эту вариацию мы и будем измерять для того, чтобы выбрать лучшую переменную при последующем рассмотрении. Среднее значение *Переменной 1* равно 1,5, а для *Переменной 2* — 0,5, поэтому

$$(1-1{,}5)^2+(2-1{,}5)^2=0{,}5$$

$$\text{и}\ (1-(-0{,}5))^2+(-2-(-0{,}5))^2=4{,}5.$$

Следовательно, рассмотренный метод «выберет» *Переменную 2*. Если число переменных, с которыми вы манипулируете, достаточно велико, то вы будете рады любому методу, позволяющему быстрее определить искомую переменную.

4.3. ИСПЫТАНИЕ НАШЕЙ НОВОЙ ЭКСПЕРТНОЙ СИСТЕМЫ

Предположим, что мы модифицировали исходную программу таким образом, чтобы исключить из нее некоторые переменные после анализа их максимальных и минимальных значений (см. § 4.2). Листинг модифицированной программы приведен в конце этого параграфа.

Чтобы убедиться, насколько хорошо (или плохо) работает экспертная система, рассмотрим пример *Птица—Самолет—Планер*. Для простоты разберем только три признака: *Крылья, Клюв* и *Двигатель. Птица* имеет *Крылья* и *Клюв*, но не имеет *Двигателя; Самолет* имеет *Крылья* и *Двигатель*, но не имеет *Клюва*, и, наконец, *Планер* имеет *Крылья*, но не имеет ни *Клюва*, ни *Двигателя.*

Перед началом вычислений массив RULES обнуляется. Мы сообщаем экспертной системе, что наша задача содержит три переменные и три возможных исхода. Каждой переменной и каждому исходу присваивается определенное имя. Затем вводятся в программу минимальные и максимальные значения этих переменных. Ясно, что они равны 0 и 1, что соответствует отсутствию или наличию данного признака в запросе.

	Минимальное значение	Максимальное значение
Крылья	0	1
Клюв	0	1
Двигатель	0	1

После того как мы введем полное описание исходов для любой из трех переменных, получим следующее содержание массива VALUE (I):

	Птица	*Самолет*	*Планер*
Крылья	1	1	1
Клюв	1	0	0
Двигатель	0	1	0

Порядок дальнейших действий осуществляется по усмотрению пользователя, но полагаем, что это будет так...

Когда мы в первый раз запускаем систему, она ничего не знает — массивы RULES и RULEVALUE обнулены. Поэтому делаем вывод, что отсутствуют какие-либо оценки вообще. Система действует «в лоб» и спрашивает вас сразу о том, какой вероятный исход вы задумали. Допустим, вы задумали *Птицу*, тогда следует ввести в систему значения VALUE (I), соответствующие *Птице*. Массив RULES корректируется, так как эксперт мог сделать неверное предположение:

	Птица	*Самолет*	*Планер*
Крылья	1	—1	—1
Клюв	1	—1	—1
Двигатель	0	0	0

Другими словами, значения VALUE (I, J) для *Птицы* были добавлены к оценке, соответствующей исходу *Птица*, и вычтены из значений, соответствующих другим правилам. Эксперт начинает теперь новый цикл обучения. Мы снова загадываем *Птицу*.

На этот раз система, возможно, в состоянии получить правильный ответ, поэтому она сначала спрашивает про значение, соответствующее наличию *Крыльев*. Мы вводим 1. Тогда система решает, что это не могут быть *Самолет* или *Планер* и запрашивает нас, правильный ли она делает вывод, что возможный исход *Птица*. Мы отвечаем: «Да», — в этом случае ничего не надо корректировать.

Теперь нам известно, что *Крыльев* как единственного признака недостаточно для того, чтобы указать на *Птицу*. Но для данного состояния знаний экспертной системы этого достаточно. В текущем состоянии вектор VALUE (I) = $(1, x, x)$, т. е. применение правил для *Птицы* дает бо́льшую оценку, чем для других исходов. Анализируя информацию о минимальных и максимальных значениях, которую имеет система, заметим, что в массиве VALUE (I) отсутствуют значения для последних двух переменных, которые могли бы изменить это решение.

Итак, наш эксперт получил правильный ответ, имея неполную информацию, не потому, что функционировал неправильно, а потому, что набор правил оказался на данный момент несовершенным. Мы могли бы прояснить ситуацию, заявив, что в жизни до сих пор такое могло произойти только с *Птицей*.

Рассмотрим еще один пример. Допустим, мы загадали *Самолет*. Система спрашивает нас о наличии *Крыльев* Мы отвечаем 1. Снова система решает, что это не может быть *Самолет* или *Планер*. Она запрашивает информацию относительного возможного исхода (*Птицы*). Подобный запрос безусловно логичен, так как раньше система об этом уже запрашивала. Мы же отвечаем: «Нет». Тогда система спрашивает нас, какой исход наиболее вероятен. Мы отвечаем: «*Самолет*». Далее система интересуется другими значениями массива VALUE (I), чтобы откорректировать свои правила. Мы даем ей корректные значения для *Клюва* или *Двигателя*, поэтому теперь массив VALUE (I) = (1, 0, 1). Правила корректируются вновь.

Рассмотрим следующий пример. Снова загадаем *Самолет*. На этот раз система интересуется сначала наличием *Клюва*. Мы вводим 0. Затем она спрашивает о *Двигателе*. Мы вводим 1. Система решает, что это не может быть *Птица* или *Планер* и спрашивает, согласны ли мы, что наиболее вероятный исход — *Самолет*? Теперь такой вариант выглядит довольно обещающим. На этот раз система не спросила нас о *Крыльях*, что, конечно, хороший признак, поскольку она ничему бы «не научилась», спросив об этом.

Посмотрим, какой вид имеет массив RULES:

	Птица	*Самолет*	*Планер*
Крылья	0	0	—2
Клюв	1	—1	—1
Двигатель	—1	1	—1

После последней ошибки экспертная система добавила значения переменных, соответствующих *Самолету*, к оценке для *Самолета* и вычла их же из оценки для *Птицы* и *Планера*.

Остается выяснить, почему система спросила сначала о *Клюве*? Она сделала это, так как названная переменная казалась ей наиболее важной среди всех остальных. Каждое правило дает оценки 1 или —1 в строке *Клюв*, поэтому, суммируя произведения максимальных значений переменных и абсолютных значений в массиве RULES, экспертная система оценивает *Клюв* и *Двигатель* тремя, а *Крылья* — двумя баллами в порядке приоритета. *Клюв* в этой последовательности стоит перед *Двигателем*, поэтому система спрашивает сначала о *Клюве*.

С этих позиций, если мы рассматриваем *Птицу* или *Самолет*, экспертная система права. Каждый раз она будет спрашивать о наличии *Клюва* у предполагаемого объекта. Если мы вводим 1, то система будет считать, что это *Птица*, если же нет, то станет запрашивать информацию о *Двигателе*. Если мы вводим 1, то она будет считать, что это *Самолет*.

Рассмотрим еще один пример. На этот раз задаем *Планер*. В ответ на запрос о наличии *Клюва* мы ответим 0. Система спросит нас о *Двигателе* и снова получит в ответ 0. И, наконец, спросив о *Крыльях*, получит в ответ 1. В этот момент она решит, что объект не может быть *Планером*, и предположит, что это какой-то другой *Самолет*, а какой именно, она не знает. Мы ответим системе, что это *Планер*, и она снова скорректирует массив RULES.

Еще один пример. Допустим, мы предположили, что объект — *Птица*. Экспертная система спрашивает о наличии *Крыльев* и *Клюва*, ответ соответственно 1 и 1. Она решает, что это не может быть *Планер*, и спрашивает о *Двигателе.* Мы отвечаем 0, и только тогда система делает правильный выбор.

Теперь загадаем *Планер*. Система интересуется у нас о *Крыльях*, *Клюве* и *Двигателе* Мы отвечаем соответственно 1, 0, 0. Экспертная система полагает, что это на самом деле *Планер*.

Предположим, что объект — *Самолет*. Система спрашивает нас о *Крыльях*, *Клюве* и *Двигателе*. Мы отвечает 1, 0, 1, и она справедливо решает, что это *Самолет.*

Еще один пример. Мы предположили *Птицу*. Система спрашивает нас о *Крыльях*, *Клюве* и *Двигателе*. Мы отвечаем соответственно 1, 1, 0, и она делает верное предположение. После этих сеансов система больше не делает ошибок. Она может точно определить (идентифицировать) *Птицу*, *Самолет* и *Планер*. Это как раз то, что мы от нее хотели.

Набор правил на этом этапе имеет следующий вид:

	Птица	*Самолет*	*Планер*
Крылья	—1	—1	—1
Клюв	1	—1	—1
Двигатель	—1	1	—1

Сделаем несколько замечаний относительно полученного набора правил. Во-первых, система угадывает корректно, по умолчанию, только *Планер*. Этот объект имеет вектор переменных (1, 0, 0), следовательно, любые из трех исходов дают значение вектора решений — 1, поэтому нет особых оснований для выбора *Планера*, а не любого другого исхода. Во-вторых, гарантией правильного выбора системы будет то, что при наличии объекта, имеющего *Крылья* и не имеющего *Клюва*, она ищет максимальное значение вектора решений. Все три исхода имеют одинаковое значение, поэтому система останавливается на *Планере* — последнем исходе, который она просматривает. Отсутствие у объекта *Двигателя* означает фактически то же самое — выбор *Планера* как последнего исхода в списке с указанием на то, что этому решению нет альтернатив.

В любом аналогичном примере многое зависит от корректировки правил и способа решения, полученного на основе тестов, реализующих условия типа «больше чем» или «больше чем или равно». Более строгий тест «больше чем или равно» изменяет правила чаще и вначале делает существенно больше ошибок. Однако в результате его использования окончательный набор правил получается несколько лучше, т. е. удается задавать меньшее количество вопросов. Более простой тест «больше чем» изменяет правила не так часто и «работает», в частности, с учетом естественного порядка данных в списке. Это следует помнить, когда вы пытаетесь понять, что же ваша программа сделала с вашими правилами.

Заметим далее, что экспертная система часто спрашивает о наличии *Крыльев* — а мы знаем, что этот вопрос вообще не оказывает на окончательное решение никакого влияния. Дело в том, что значения массива RULEVALUE суть (3, 3, 3) соответственно для *Крыльев, Клюва, Двигателя*, — все это заставляет нас думать о наличии некоторых других методов вычисления оценок соответствующих правил.

Например, если мы вычислили среднее значение для каждой переменной (—1; —1/3, —1/3), то мы могли бы определить сумму квадратов отклонений от этого среднего значения по каждой переменной (0; 2,67; 2,67). Совершенно ясно, что *Клюв* и *Двигатель* являются наиболее «важными» переменными, тогда как переменная *Крылья* остается по существу пассивной. [Сумма квадратов отклонений переменных от средних значений фактически характеризует дисперсию, если не учитывать, что в последнем случае она делится на n (см. технический обзор, помещенный в конце книги).]

Итак, можно получить различные наборы правил, если обучать экспертную систему, ставя ей задачи в различном порядке. Однако соответствующий или подходящий набор правил должен при этом «работать».

Вспомним наш разговор о линейной сепарабельности. Мы получили экспертную систему, которая проводит разделяющую линию между тремя группами объектов — *Птицами, Самолетами* и *Планерами*. Не имеет существенного значения, где проходит эта линия, поскольку она служит для разделения трех названных групп объектов. Текущий набор правил (правда, он не единственный) позволяет получить такое разграничение. Пример, приведенный ниже, очень показателен в этом смысле, но не стоит обольщаться, что на практике все проблемы, поставленные перед экспертной системой, будут такими же четкими.

Предположим, мы сохранили те же три исхода (*Пти-*

ца, Самолет, Планер), а теперь работаем с шестью переменными: *Крылья, Хвост, Клюв, Оперение, Двигатель* и *Шасси*. Здравый смысл подсказывает нам, что такое количество переменных избыточно, т. е. они не нужны для ответа на основной вопрос. Но сможет ли экспертная система его разрешить? Ответ — нет.

Поскольку мы все еще имеем три исхода, известно, что все дело можно решить, задав два хорошо поставленных вопроса. Однако после того как экспертная система была достаточно хорошо обучена, она спросила о следующих признаках (для каждого из трех исходов):

Птица	*Двигатель*	— ответ 0	
	Шасси	— ответ 0	
	Клюв	— ответ 1	(вывод—*Птица*)
Самолет . . .	*Двигатель*	— ответ 1	
	Шасси	— ответ 1	(вывод—*Самолет*)
Планер . . .	*Двигатель*	— ответ 0	
	Шасси	— ответ 0	
	Клюв	— ответ 0	
	Оперение	— ответ 0	(вывод — *Планер*)

В данном случае это неплохо. Система дала правильные ответы во всех случаях и спросила не о всех признаках. Например, она не спросила нигде о *Крыльях* и *Хвосте*, что, конечно, хорошо, потому что эти переменные не несут новой информации для системы. Однако с нашей точки зрения слишком «неэкономно» спрашивать о *Шасси* после вопроса о *Двигателе*, тем более каждый раз повторять эту связку. Но это происходит потому, что в данном конкретном примере эти вопросы ставятся всегда друг за другом. По аналогии часто спрашивают об *Оперении* сразу после вопроса о *Клюве* — еще одна пара вопросов, которая, как мы знаем, всегда идет вместе.

Особенность здесь заключается в том, что экспертная система не знает, что эти два признака (вопроса) всегда идут один за другим. Спрашивая о *Двигателе*, система «не совсем уверена» в принятии решения и поэтому должна задать другой вопрос. Для нее вопрос о *Шасси* кажется важным, и она спрашивает об этом. Система — не человек, она не может сделать вывод о наличии *Шасси* на основании информации о *Двигателе*. С учетом имеющихся знаний эксперта вопросы о *Двигателе* и *Шасси* могут случайно оказаться следующими один за другим.

В задаче, которую решает экспертная система в данный момент, могут и не встретиться вопросы о *Двигателе* и *Шасси*, что оказывает влияние на решение о возможном исходе. Система просто продолжает задавать вопросы, определять состояния правил вывода минимальных и максимальных значений, присваиваемых каждой переменной. Она не может получить свежую информацию о любой переменной, которая могла бы заставить ее изменить свое мнение относительно вероятного исхода.

Несмотря на то что у объекта есть *Двигатель*, этой информации недостаточно, чтобы система была абсолютно уверена в конкретном исходе. Предположим, у вас имеется некий объект с *Двигателем*, а затем вдруг выяснилось, что у него нет *Шасси*. Измените ли вы свое мнение об этом объекте? Да, измените. Лучше спросить, есть ли *Шасси*.

Откровенно говоря, трудно себе представить, о каком типе объекта мог бы подумать эксперт в таком случае. Вероятно, следовало бы рассмотреть в качестве возможного исхода *Ракету*, лишь бы эксперт был счастлив.

Ниже представлен листинг программы, используемый в этом параграфе. Программа содержит обучающий алгоритм. Для того чтобы прийти к быстрому решению, в ней используются минимальные и максимальные значения. Вопросы задаются на основе полученных оценок для соответствующих правил с учетом абсолютных разностей максимальных и минимальных значений для оценки степени их важности. Возможные исходы, которые не могут получить достаточно высоких оценок, чтобы заместить лучшие на данный момент предсказания, исключаются из дальнейшего рассмотрения. Запустив программу один раз, постарайтесь изменить ее так, чтобы значения в массиве RULEVALUE вычислялись в виде сумм квадратов отклонений относительно средних, а не абсолютных значений разностей. Последнее решение влияет на порядок следования вопросов (рис. 4.4).

```
10 CLS
20 INPUT "ВВЕДИТЕ ЧИСЛО ПЕРЕМЕННЫХ "; VAR
30 DIM VALUE(VAR), VAR$(VAR), MINI(VAR), MAXI(VAR),
   VARFLAG(VAR), POSSIBLE(VAR), RULEVALUE(VAR)
40 PRINT "ВВЕДИТЕ ИМЕНА ЭТИХ ПЕРЕМЕННЫХ:"
```

```
 50 FOR I=1 TO VAR
 60 PRINT "ПЕРЕМЕННАЯ "; I;" – ";: INPUT VAR$(I)
 70 INPUT "ЕЕ МАКСИМАЛЬНОЕ ЗНАЧЕНИЕ = ";MINI(I)
 80 INPUT "ЕЕ МАКСИМАЛЬНОЕ ЗНАЧЕНИЕ = ";MAXI(I)
 90 NEXT
100 INPUT "ВВЕДИТЕ ЧИСЛО ДОПУСТИМЫХ ИСХОДОВ"; OUTCOMES
110 DIM OUTCOMES$(OUTCOMES), RULES(VAR, OUTCOMES),
    DECISION(OUTCOMES), OUTFLAG(OUTCOMES)
120 PRINT "НАЗОВИТЕ ЭТИ ИСХОДЫ "
130 FOR I=1 TO OUTCOMES
140 PRINT "ИСХОД "; I;" – ";: INPUT OUTCOMES$(I)
150 NEXT
160 REM ПОДРОБНОЕ ОПИСАНИЕ ПЕРЕМЕННЫХ:
170 PRINT "ПЕРЕМЕННАЯ", "МИН. ЗНАЧЕНИЕ  МАКС. ЗНАЧЕНИЕ"
180 FOR I=1 TO VAR
190 PRINT VAR$(I),MINI(I), MAXI(I)
200 NEXT
210 PRINT "ДЛЯ ПРОДОЛЖЕНИЯ НАЖМИТЕ ЛЮБУЮ КЛАВИШУ"
220 X$="": WHILE X$="": X$=INKEY$: WEND
230 REM НАЧАЛО ПРОЦЕССА ОБУЧЕНИЯ
240 CLS
250 PRINT "НАЧАЛО ЦИКЛА ОБУЧЕНИЯ:"
260 PRINT "ВЫ ДОЛЖНЫ ВВЕСТИ ЗНАЧЕНИЯ ПЕРЕМЕННЫХ"
270 PRINT "Я БУДУ УГАДЫВАТЬ ВОЗМОЖНЫЕ ИСХОДЫ"
280 PRINT "ВЫ ДОЛЖНЫ СКАЗАТЬ МНЕ,ПРАВ Я ИЛИ НЕТ"
290 REM ОЧИСТКА МАССИВА VALUE И УСТАНОВКА ФЛАЖКОВ
    МАССИВА VARFLAG
300 FOR I=1 TO VAR
310 VALUE(I)=0
320 VARFLAG(I)=1
330 NEXT
340 REM ОЧИСТКА МАССИВОВ DECISION, POSSIBLE И УСТАНОВ-
    КА ФЛАЖКОВ МАССИВА OUTFLAG
350 FOR J=1 TO OUTCOMES
360 DECISION(J)=0
370 POSSIBLE(J)=0
380 OUTFLAG(J)=1
390 NEXT
400 REM ОПРЕДЕЛЕНИЕ НАИБОЛЕЕ СУЩЕСТВЕННОЙ ПЕРЕМЕН-
    НОЙ BESTVAR ДЛЯ ЗАДАННОГО ЗНАЧЕНИЯ ПРАВИЛА RV
410 RV=0
420 BESTVAR=1
430 FOR I=1 TO VAR
```

```
440 RULEVALUE(I)=0
450 FOR J=1 TO OUTCOMES
460 RULEVALUE(I)=RULEVALUE(I)+ABS((MINI(I)–MAXI(I))*RULES
    (I,J))*VARFLAG(I)*OUTFLAG(J)
470 NEXT
480 IF RULEVALUE(I)>RV THEN BESTVAR=I: RV=RULEVALUE(I)
490 NEXT
500 IF RV=0 THEN 820: REM НАИБОЛЕЕ СУЩЕСТВЕННАЯ ПЕРЕ-
    МЕННАЯ ВКЛАДА НЕ ВНОСИТ
510 PRINT "ПЕРЕМЕННАЯ "; BESTVAR;" ("; VAR$(BESTVAR); ") –";:
    INPUT VALUE(BESTVAR)
520 VARFLAG(BESTVAR)=0: REM УСТАНОВКА ФЛАЖКА НА НУЛЬ,
    Т. К. ЗНАЧЕНИЕ ПЕРЕМЕННОЙ УЖЕ ПОЛУЧЕНО
530 REM КОРРЕКЦИЯ МАССИВА DECISION
540 FOR J=1 TO OUTCOMES
550 X=VALUE(BESTVAR)*RULES(BESTVAR,J): DECISION(J)=
    DECISION(J)+X
560 NEXT
570 REM НАХОЖДЕНИЕ МАКСИМАЛЬНОГО ЗНАЧЕНИЯ DECISION,
    Т. Е. DECISION(BEST), КАК ЛУЧШЕГО ТЕКУЩЕГО ЗНАЧЕНИЯ
    РЕШЕНИЯ
580 DECISION=–10000
590 FOR J=1 TO OUTCOMES
600 IF DECISION(J)>=DECISION THEN DECISION=DECISION(J):
    BEST=J
610 NEXT
620 REM КОРРЕКЦИЯ МАССИВА POSSIBLE
630 FOR J=1 TO OUTCOMES
640 POSSIBLE(J)=DECISION(J)
650 FOR I=1 TO VAR
660 IF VARFLAG(I) THEN IF RULES(I,J)>RULES(I,BEST) THEN
    POSSIBLE(J)=POSSIBLE(J)+(RULES(I,J)–RULES(I,BEST))*
    MAXI(I) ELSE POSSIBLE(J)=POSSIBLE(J)–(RULES(I,BEST)–
    RULES(I,J))*MINI(I)
670 NEXT: NEXT
680 REM НАХОЖДЕНИЕ МАКСИМАЛЬНОГО ЗНАЧЕНИЯ
    POSSIBLE, Т. Е. POSSIBLE(BESTPOSS) – НАИЛУЧШЕГО
    ТЕКУЩЕГО ЗНАЧЕНИЯ
690 MAXPOSS=POSSIBLE(BEST)
700 BESTPOSS=BEST
710 FOR J=1 TO OUTCOMES
720 IF POSSIBLE(J)>=MAXPOSS THEN MAXPOSS=POSSIBLE(J):
    BESTPOSS=J
```

```
730 IF POSSIBLE(J)<POSSIBLE(BEST) THEN OUTFLAG(J)=0:
    PRINT "Это не может быть ";OUTCOMES(J):
    REM Этот исход больше нельзя считать ожидаемым
740 NEXT
750 IF BESTPOSS<>BEST THEN 400: REM Пока еще существует
    неопределенность, поэтому введите другую переменную
760 PRINT "Ожидаемый исход ";OUTCOMES$(BEST);" y/n ";: INPUT A$
770 IF A$<>"Y" AND A$<>"y" THEN 820
780 PRINT "Прекрасно!"
790 PRINT "Для продолжения нажми любую клавишу"
800 X$="": WHILE X$="": X$=INKEY$: WEND
810 GOTO 230: REM Введи другой пример
820 REM Спроси правильный ответ
830 FOR I=1 TO OUTCOMES
840 PRINT I;" ";OUTCOMES$(I)
850 NEXT
860 INPUT "Какой исход возможен ";CORRECT
870 REM Спроси про любую неизвестную на данный момент переменную:
880 FOR J=1 TO VAR
890 IF VARFLAG(J)=0 THEN 920
900 PRINT "Чему равно значение переменной ";J;" (";VAR$(J);") ";:
    INPUT VALUE(J)
910 VARFLAG(J)=0
920 NEXT
930 REM Коррекция массива RULES
940 FOR I=1 TO OUTCOMES
950 IF I=CORRECT OR DECISION(I)<DECISION(CORRECT) THEN 990
960 FOR J=1 TO VAR
970 RULES(J,I)=RULES(J,I)-VALUE(J)
980 NEXT
990 NEXT
1000 FOR J=1 TO VAR
1010 RULES(J,CORRECT)=RULES(J,CORRECT)+VALUE(J)
1020 NEXT
1030 REM Отображение текущего массива правил
1040 PRINT "Текущий массив правил - "
1050 FOR J=1 TO OUTCOMES
1060 PRINT TAB(10+J*10);OUTCOMES$(J);
1070 NEXT
1080 PRINT
1090 FOR I=1 TO VAR
1100 PRINT VAR$(I);
1110 FOR J=1 TO OUTCOMES
```

```
1120 PRINT TAB(10+J*10);RULES(I,J);
1130 NEXT
1140 PRINT
1150 NEXT
1160 PRINT "ДЛЯ ПРОДОЛЖЕНИЯ НАЖМИ ЛЮБУЮ КЛАВИШУ"
1170 X$="": WHILE X$="": X$=INKEY$: WEND
1180 GOTO 230: REM ПРОДОЛЖЕНИЕ ОБУЧЕНИЯ
```

Рис 4.4. Модифицированная программа с учетом минимальных и максимальных значений

Глава 5. РЕАЛЬНАЯ ЭКСПЕРТНАЯ СИСТЕМА

5.1. СНОВА ПРЕДСКАЗАНИЕ ПОГОДЫ

Хорошо иметь экспертную систему, способную отвечать на затейливые вопросы о том, что это за объект: *Птица, Самолет* или *Планер*. Она похожа на прекрасную игрушку. Но что произойдет, если мы зададим системе реальный вопрос, касающийся, например, предсказания погоды? Что она будет делать, если мы спросим ее о том, будет ли завтра дождь?

Это чисто практический вопрос. Мы не знаем, как на него ответить, поэтому если экспертная система в состоянии нам помочь, мы, возможно, почувствуем, что имеем дело с чем-то стоящим.

Итак, обратимся за реально существующими данными в Лондонский центр погоды.

Каждый месяц Центр публикует сведения о выпадении осадков, температуре и состоянии облачности за каждый день. Просматривая эти дневные сводки, мы могли бы сообщать полученные данные эксперту и тем самым тренировать его.

На основании все новых и новых сведений в процессе тренировки мы можем обнаружить, что предсказания эксперта улучшаются.

В табл. 5.1 приведены данные о прогнозе погоды за март 1982 г.

По случайному совпадению это был наиболее солнечный март, начиная с 1967 г., и второй по количеству сол-

Таблица 5.1. **Сводка погоды в Лондоне за март 1982 г.**

День месяца	Минимальная температура, °C	Максимальная температура, °C	Уровень осадков, мм	Солнечное время дня, ч
1	9,4	11,0	17,5	3,2
2	4,2	12,5	4,1	6,7
3	7,6	11,2	7,7	1,1
4	5,7	10,5	1,8	4,3
5	3,0	12,0	0	9,5
6	4,4	9,6	0	3,5
7	4,8	9,4	0	10,1
8	1,8	9,2	5,5	7,8
9	2,4	10,2	4,8	4,1
10	5,5	12,7	4,2	3,8
11	3,7	10,9	4,4	9,2
12	5,9	10,0	4,8	7,1
13	3,0	11,9	0,2	8,3
14	5,4	12,1	0	1,8
15	8,8	9,1	8,8	0
16	2,4	8,5	3,0	3,1
17	4,3	10,8	4,2	4,3
18	3,4	11,1	0	6,6
19	4,4	8,4	5,4	0,7
20	5,1	7,9	3,0	0,1
21	4,4	7,3	1,0	0
22	5,6	14,0	0	6,8
23	5,7	14,0	0	8,8
24	2,9	13,9	0	9,5
25	5,8	16,4	0	10,3
26	3,9	17,0	0	9,9
27	3,8	18,3	0	8,3
28	5,8	15,4	0	7,0
29	6,7	8,8	6,4	4,2
30	4,5	9,6	0	8,8
31	4,6	9,6	3,2	4,2

Примечание. Имеются в виду следующие диапазоны изменения параметров: температура — от 0 до 20 °C; уровень осадков — от 0 до 25 мм; солнечное время дня — от 0 до 12 ч.

нечных дней март с 1929 г. — времени начала регулярных записей прогнозов погоды.

Для нашей экспертной системы выберем следующие четыре переменные: *Минимальную температуру, Максимальную температуру, Уровень осадков и Солнечное время дня.*

Мы ввели в систему приемлемые максимальные и минимальные значения, а также установили два исхода: *Дождь* и *Нет дождя (Сухо)*. Первой строке таблицы соответствует 1 марта 1982 г. Мы ввели указанные переменные и дали информацию системе о том, что на следующий день был *Дождь.*

Затем вводим информацию за 2 марта, и если предсказание неверно, то считаем, что на следующий день снова был *Дождь.* Затем на следующий день — удача, мы оказались правы.

Если теперь обратиться к табл. 5.2, то увидим, что было 18 сухих и 13 дождливых дней в марте 1982 г. Можно предположить с вероятностью пятьдесят на пятьдесят правильный ответ, основываясь лишь на законах стати-

Таблица 5.2. **Фактическая и предсказанная погода в марте 1982 г.**

День месяца	Фактическая погода на завтра	Погода, предсказанная экспертами
1	Дождь	Неизвестна
2	»	Дождь
3	»	»
4*	Сухо	»
5*	»	»
6	»	Сухо
7*	Дождь	»
8	»	Дождь
9	»	»
10	»	»
11	»	»
12	»	»
13	Сухо	Сухо
14	Дождь	Дождь
15	»	»
16	»	»
17*	Сухо	»
18	Дождь	Сухо
19	»	Дождь
20	»	»
21*	Сухо	»
22	»	Сухо
23	»	»
24	»	»
25	»	»
26	»	»
27	»	»
28*	Дождь	»
29*	Сухо	Дождь
30	Дождь	»
31	»	»

* В эти дни прогноз оказался ошибочным.

стики. Как тогда эксперт получил этот результат и лучше ли он, чем тот, который основан на законах статистики?

Из табл. 5.2 можно видеть, что эксперт предсказал *Дождь* или его отсутствие в 22 случаях из 30 (мы не должны учитывать первый день, так как эксперт еще не успел сформировать правила вывода, поскольку не знал исхода первого дня). С точки зрения статистики 73 % успеха — это достаточно точный прогноз. Возможно, даже точнее, чем если бы такие предсказания делал человек-эксперт. Однако давайте рассмотрим полученные результаты более детально.

Первое, что нужно сделать, — успокоиться и не очень задирать нос. Существует простое правило, которое применительно к тому же набору данных позволяет сделать только 10 ошибок за месяц. Оно гласит, что погода завтра будет такой же (*Дождь* или *Нет дождя*), как и сегодня. Если мы воспользуемся этим правилом, то будем допускать ошибки в тех случаях, когда сухому дню предшествует дождливый и наоборот. Мы окажемся правы в большинстве случаев просто потому, что знаем о тенденции существования периодов сухой погоды, перемежающихся с периодами дождливой погоды.

Следовательно, если все экспертные системы так откорректируют свои правила, что будут предсказывать завтра такую же погоду, как сегодня, то они совершат только 10 ошибок в месяц. На практике же наша система сработала лучше, чем при использовании этого правила, — допустила только 8 ошибок. Однако подобное «улучшение» произошло за счет тех случаев, когда эксперт смог уловить изменение погоды. Это, конечно, происходило не всегда (см., например, результаты за 13 марта), но в общем случае имело место.

На самом деле после первого дня наблюдений эксперт предсказал дождь. Это предсказание затем повторялось и было правильным до тех пор, пока не стало ошибкой, так как на самом деле один из следующих дней оказался сухим. Ошибка заставила эксперта скорректировать свои правила, что помогло ему предсказать сухой день за 6 марта. Внезапно погода снова изменилась, и эксперт опять допустил ошибку. К этому моменту он уже трижды скорректировал свои правила, и они давали верный результат вплоть до 17 марта, когда впервые не было дождя. Еще одна коррекция правил произошла 22 марта, после чего эксперт точно предсказал сухую погоду. В ночь с 28 на 29 марта произошли еще две ошибки, когда «вернулся» дождливый период.

В одном наша экспертная система функционировала плохо — она периодически ошибалась, т. е. была несовершенна. Каждый раз, когда система делает ошибки, она корректирует свои правила, т. е. больше не использует прежний набор правил в процессе работы. В некотором смысле это похоже на «мошенничество». В конце концов, можете спросить вы, что здесь предосудительного? В большинстве игр нам позволяют изменять правила каждый раз, когда мы проигрываем. Именно это изменение правил и затрудняет анализ нашему эксперту. Единственное, что ему остается, — поставить опыт и посмотреть, что произойдет. Если бы эксперт имел постоянный набор правил, то об этих правилах можно было бы получить больше информации, однако это не так. На практике приходится иметь дело лишь с общим методом выработки решения.

В некотором смысле это делает экспертную систему более похожей на человека. Если бы вас, например, спросили о том, какая, по вашему мнению, будет погода завтра, то вы, наверное, при ответе на этот вопрос приняли бы во внимание, что за погода была в последнее время, имея в виду погоду не только сегодня. Если погода была в основном сухой, то небольшой ливень не заставил бы вас изменить свое мнение о том, что завтра будет сухо. Вы подходите к вопросу о предсказании погоды с некоторой заданной мысленной «установкой» или предрасположенностью, чтобы интерпретировать сегодняшнюю реальность в контексте недавней погоды в целом.

Предположим, например, что вы хотели угадать, будет ли завтра снег. Если бы это было в середине января, когда недавно выпало много снега и в день наблюдений имелись все признаки, что снег будет продолжать идти, то вы были бы просто счастливы предсказать снег. Если же сейчас середина июля и налицо те же самые признаки (что будет снег), то вы предсказали бы снег с большей неохотой. Факт остается фактом: вы сами используете различные правила в зависимости от времени года. И если все-таки в середине июля действительно пойдет снег, то вы быстро скорректируете собственные правила предсказания погоды и будете склонны считать, что снег завтра возможен, если те же самые признаки появятся вновь. Именно это и делает наша экспертная система. Остается заметить, что ее поведение начинает казаться нам более осмысленным.

Столь же осмысленным выглядит поведение системы при запросе необходимой информации. Напомним, что система сама выбирает, о какой переменной спросить в первую очередь, и может делать предсказание относительно возможного исхода без опроса всех переменных, если уверена в выборе исхода. Почти всегда первый вопрос, который она задавала, касался *Уровня осадков*. Получив цифры о сегодняшнем *Уровне осадков*, система затем вырабатывала решение обычно без опроса других переменных и, как мы видели, была права

Чего же на самом деле она добилась? Был ли предсказан дождь на завтра, если сегодняшний уровень осадков оказался высок, либо сухая погода при низком уровне осадков?

Между высоким и низким уровнями существует неопределенная («серая») область, в которой система не может иметь своего мнения. Оказавшись в этой области, она запрашивает иногда другие переменные, чтобы получить дополнительную информацию об *Уровне осадков*. Как правило, запрашивается информация о *Температуре* и *Солнечном времени дня*.

Итак, поведение системы кажется вполне логичным. Таким же логичным кажется и набор правил, который она сформировала в конце месяца. Массив RULES по состоянию на 31 марта 1982 г. имеет вид

	Завтра дождь	*Завтра сухо*
Минимальная температура	—0,6999	0,6999
Максимальная температура	—2,5	2,5
Уровень осадков	4,1	—4,1
Солнечное время дня	4,6	—4,6

Для предсказания дождя на завтра эксперт будет ориентироваться на верхний *Уровень осадков* (4,1) сегодня и низкую *Температуру*. Следует учитывать также и показания *Солнечного времени дня*, хотя это может показаться на первый взгляд странным. Холодная, сырая и солнечная погода — по сути определенно «неустойчивая» погода, поэтому эксперт и будет ориентироваться на нее. Только в конце марта мы можем достаточно уверенно ожидать в начале апреля солнечные апрельские ливни!

С другой стороны, для предсказания сухой погоды на завтра эксперт, вероятно, будет ориентироваться на теплую, сухую и хмурую погоду сегодня. В любом случае это не те признаки, которые мы могли бы автоматически использовать для предсказания погоды. Но не исключено, что нам следует это сделать. Если наша экспертная система довольно точно предсказывает погоду, основываясь на данном методе, то, может быть, она так же хорошо сможет провести какую-то другую экспертизу и выполнит ее более квалифицированно в некотором отношении, чем мы. Вполне возможно, что система, получившая сначала от нас информацию, сможет затем чему-либо обучить нас. Возникает лишь единственный вопрос, а хорошо ли сделана сама система?

Фактически мы в состоянии достаточно точно определить, насколько хорошо сделана система, сообщив, что она не только правильно ответила в 73 % случаев (для нашего примера), но и что вероятность получения с ее помощью случайного ответа оказалась меньше 0,025, т. е. меньше, чем 25 шансов из тысячи. Вы вправе удивиться, сказав, как можно вычислить это?

Теперь немного статистики. Мы видели, что в марте было 17 дождливых и 13 сухих дней, а эксперт предсказал соответственно 19 дождливых и 11 сухих дней. Эти данные можно представить следующим образом:

	Предсказание эксперта		
	Дождь	*Сухо*	Всего
Фактически			
Дождь	14	3	17
Сухо	5	8	13
Всего	19	11	30

В 14 случаях был предсказан и фактически имел место дождь, а сухие дни были предсказаны и фактически имели место в 8 случаях. В 8 (5+3) случаях допущена ошибка. С помощью представленных значений аналогичные показатели можно получить и вероятностным путем. Это так называемые ожидаемые значения. Описание способа их вычисления приведено в конце параграфа.

	Ожидаемые значения		
	Дождь	*Сухо*	Всего
Фактически			
Дождь	10,77	6,23	17
Сухо	8,23	4,77	13
Всего	19	11	30

Воспользуемся этими таблицами для проведения статистического теста, известного как хи-квадрат тест.

Хи-квадрат тест гласит, что таблица, построенная на основе фактических предсказаний эксперта, отличается от таблицы, построенной на основе вероятностного подхода, причем отличие их таково, что вероятность того, что эксперт работает случайно, меньше 0,025.

Использование подобных утверждений затруднено тем, что вы можете не представлять себе, что такое хи-квадрат тест, либо не знать, можно ли верить такому утверждению. Вы вправе принять это утверждение на веру, если желаете. Или же, если вы действительно хотите знать, как это утверждение работает, то...

В первой таблице содержатся наблюдаемые, а во второй — ожидаемые частоты. Показатель

$$xu = xu + (\text{наблюдаемая} - \text{ожидаемая})^2/\text{ожидаемая}$$

для каждой ячейки в данной таблице Поэтому в нашем примере

$$xu = ((14 - 10{,}77)^2)/10{,}77 + ((3 - 6{,}23)^2)/6{,}23 + \\ + ((5 - 8{,}23)^2)/8{,}23 + ((8 - 4{,}77)^2)/4{,}77 = 6{,}111196.$$

Теперь воспользуемся статистическими таблицами, содержащими хи-квадрат распределение. Для распределения, характеризующегося одной степенью свободы, в колонке 0,025 (или 2,5%) находим значение критерия хи-квадрат, равное 5,02. Поскольку вычисленное нами значение больше табличного, вероятность, что наш результат случайный [1], меньше 0,025. Его можно интерпретировать и иначе. Значение 5,02 находится в колонке, в заголовке которой указано число 0,975, т. е. результат не является случайным с вероятностью больше 0,975.

Некоторым читателям это занятие покажется довольно скучным, и они, конечно, могут его проигнорировать. Однако такая оценка позволяет судить о качест-

---

[1] Другими словами, насколько он совпадает с некоторым гипотетическим распределением. — *Прим. пер.*

венных показателях экспертной системы. Кроме того, у вас может быть довольно много различных задач и вы, возможно, захотите оценить, насколько хороша ваша экспертная система.

Вы вправе также поинтересоваться, что такое степени свободы и как мы пришли к ожидаемым значениям.

Степень свободы DF = (ROWS—1)* (COLS—1) и зависит от числа строк (ROWS) и столбцов (COLS) в данной таблице. В нашем случае ROWS=2 и COLS=2, поэтому DF=1. При большем числе исходов степень свободы превышает 1.

Ожидаемые значения вычисляются для каждой ячейки таблицы следующим образом. Сначала определяются суммарные (наблюдаемые) значения по строке и столбцу. Затем они перемножаются и результат делится на их общую сумму (наблюдаемую). Операции повторяются для каждой ячейки и вносятся в таблицу ожидаемых значений. Если все сделано правильно, то таблица ожидаемых значений будет иметь такое же количество строк и столбцов, а также общую сумму, как и таблица наблюдаемых значений. В нашем примере в первой ячейке таблицы ожидаемых значений было число 10,77. Этот результат получен следующим образом: $17 \cdot 19/30$.

Приведенный алгоритм является идеальным объектом для компьютерной программы. Если вам приходилось использовать какую-либо статистическую таблицу, то вы могли бы отметить, что если значение критерия хи-квадрат равно нулю, то результат, конечно, должен рассматриваться как случайный. Чем больше значение критерия хи-квадрат, тем лучше с нашей точки зрения оценка экспертной системы.

5.2. ПРОГРАММА ВЫЧИСЛЕНИЯ КРИТЕРИЯ ХИ-КВАДРАТ

Если вы заинтересовались контролем результатов работы экспертной системы с помощью теста хи-квадрат, но не были знакомы с этим методом раньше, вам, видимо, понадобится программа, которая делает это. Вы можете представить ваши результаты в виде таблицы, как и раньше, а затем довериться логике указанной программы. Она напечатает в конце работы результаты, но для получения окончательного ответа вам все же необходимо будет обратиться к статистическим таблицам, содержащим распределение хи-квадрат.

```
10 CLS: PRINT: PRINT "    Хи-квадрат тест.": PRINT
20 INPUT "Введите число строк ";ROWS
30 INPUT "Введите число столбцов ";COLS
40 DIM OBSERVED(ROWS+1,COLS+1),EXPECTED(ROWS,COLS)
45 PRINT: PRINT "Введите элементы испытуемой таблицы:"
50 FOR I=1 TO ROWS
60 FOR J=1 TO COLS
70 PRINT "Строка "; I;" Столбец ";J;" = ";: INPUT OBSERVED(I,J)
80 NEXT: NEXT
90 FOR I=1 TO ROWS
100 FOR J=1 TO COLS
110 OBSERVED(ROWS+1,J)=OBSERVED(ROWS+1,J)+OBSERVED(I,J)
120 OBSERVED(I,COLS+1)=OBSERVED(I,COLS+1)+OBSERVED(I,J)
130 OBSERVED(ROWS+1,COLS+1)=OBSERVED(ROWS+1,COLS+1)+OBSERVED(I,J)
140 NEXT: NEXT
150 FOR I=1 TO ROWS
160 FOR J=1 TO COLS
170 EXPECTED(I,J)=OBSERVED(I,COLS+1)*OBSERVED(ROWS+1,J)/
    OBSERVED(ROWS+1,COLS+1)
180 NEXT: NEXT
190 FOR I=1 TO ROWS
200 FOR J=1 TO COLS
210 CHI=CHI+((OBSERVED(I,J)-EXPECTED(I,J))^2)/EXPECTED(I,J)
220 NEXT: NEXT
230 DF=(ROW-1)*(COLS-1)
240 PRINT
250 PRINT "Хи-квадрат = ";CHI:
    PRINT "Число степеней свободы = ";DF
260 PRINT
270 END
```

5.3. ПРАКТИЧЕСКИЕ УПРАЖНЕНИЯ С ВАШЕЙ ЭКСПЕРТНОЙ СИСТЕМОЙ

На этом этапе следует сделать одно замечание относительно вашей экспертной системы. Во многом она прекрасна. В ней нет ничего, что могло бы привести к ошибке. Она работает и будет обучаться на примерах с вашей помощью или вообще обойдется без нее. Системе нужно только дать несколько примеров, чтобы было с чего начать. Вопрос заключается в том, сколько же примеров ей нужно? На какой стадии обучения система начнет выдавать суждения «лучшего качества?»

Рассматривая пример *Птица — Самолет — Планер*, вы, возможно, полагаете, что три обучающих варианта примера было бы достаточно. По одному на каждый возможный объект. Действительно, система увидит все, раз уж предоставите ей три варианта. Верно, она увидит все. Неверно лишь то, что она сможет выработать адекватные правила для последующего распознавания всех вариантов. Потребуется больше чем три попытки для «перетасовки» набора правил, пока, наконец, система отыщет действительно подходящий набор правил и метод работы. Чтобы быть точным, ей требуется предъявить достаточное количество необходимых примеров. Понятно, что данная теория зависит от самого эксперта, имеющего доступ к одному из возможных примеров, пусть даже бесконечное число раз. Вот и все, что говорит теория.

В ряде частных случаев не все так плохо. Действительно, если исходы линейно-сепарабельны, то можно разработать некоторый набор правил, который позволит за конечное время идентифицировать все возможные исходы. Так, в примере *Птица — Самолет — Планер* вы, допустим, хотели дать эксперту оценить каждый из трех возможных исходов бесконечное число раз, но если существует хороший набор правил, то система его определит за конечное число шагов.

В этой ситуации среди вас, несомненно, найдутся такие, кто отложит этот трактат в сторону, либо поняв его внутреннюю бесперспективность, либо на том основании, что если для получения решения необходимо собрать почти бесконечное количество данных, то лучше начать это делать прямо сейчас.

Думаю, что нам стоит, однако, прерваться и несколько снизить требования. В теории, возможно, существует бесконечное число различных примеров, которые вы можете ввести в систему. На практике же у вас нет записей о всех гипотетических возможностях (они бы заняли слишком много места в вашей комнате). У вас есть лишь конечный набор таких данных, чтобы можно было начать работать. Причем за короткий период времени характеристики системы должны стать достаточно хорошими, если вы продолжаете вводить в нее все примеры, которые у вас под рукой.

Итак, что за проблема свалилась на вас теперь? Как предоставить экспертной системе достаточное количество примеров, не тратя при этом всей оставшейся жизни на сидение за дисплеем и не разбивая пальцы до крови от работы на клавишном пульте?

К счастью, все не так безнадежно. Повторяющиеся операции компьютер в состоянии делать сам. Все, что вам нужно, — это взять все N примеров и занести их в определенный массив с именем, например EXAMPLES (VAR+1, N), а затем написать короткую программу, чтобы система выполнила эти операции в ваше отсутствие.

Идея состоит в том, что вы устанавливаете массив EXAMPLES, даете системе возможность работать с ним, а сами идете в бар ближайшего отеля, где можете сообщить собравшейся толпе, что ваша собственная экспертная система в данный момент делает за вас всю работу. Такой комментарий всегда привлекает определенный круг людей из вашего окружения, которые, главным образом, хотели бы знать, говорите вы правду или нет, чтобы выплеснуть на вас все нахлынувшие на них эмоции.

Откажемся, однако, от желания вернуться в отель до тех пор, пока не заставим экспертную систему трудиться.

Сформируем вначале массив EXAMPLES (VAR+ +1, N), где N — число примеров, которые вы собираетесь дать экспертной системе. Очевидно, что первое граничное значение массива резервирует возможность применения VAR перемеиных; VAR+1 элемент используется для того, чтобы хранить возможный исход для этого примера.

Здесь вы могли бы, конечно, немного слукавить: вместо того чтобы делать это, строго придерживаясь правил, вы могли бы заполнить этот массив тогда, когда вводите примеры в экспертную систему в процессе обучения.

Теперь следует вставить повторяющийся фрагмент (цикл) в программу, чтобы она не прекращала работать над примерами и изменяла правила по мере необходимости. Введем индекс I в качестве управляющей переменной цикла для контроля за тем, какой пример эксперт осваивает в данный момент:

```
10 FOR I=1 TO N
20 FOR J=1 TO VAR
```

```
30 VALUE (J)=EXAMPLES(J, I)
40 NEXT : NEXT
```

Недостатком такого цикла является то, что примеры задаются в одном и том же порядке. Возможно, что в ряде случаев это не играет особой роли. Предположим, однако, что у вас есть набор правил, который дает правильный исход для примера I. Они же могут дать правильный ответ и для примера I+1. Следовательно, на этом шаге правила не изменяются. Допустим также, что для следующего, (I+2)-го примера эксперт указал на неверный исход, поэтому правила были изменены и, возможно, на следующем, (I+3)-м примере эксперт получит правильный ответ. Если, однако, окажется, что следующим будет пример I, для которого эти новые правила чем-то плохи, т. е. не исключена ошибка, тогда текущие правила снова придется изменить.

Очевидно, чтобы исключить подобное рассогласование, экспертная система должна вырабатывать различные последовательности выбора примеров для обучения и использовать их снова и снова. Итак, если у вас есть N примеров, то вы имеете N различных способов выбора первого примера, (N—1) второго, (N—2) третьего и т. д. В общем случае набирается N! способов выбора примеров, где $N! = N(N-1)(N-2)(N-3)\dots(2)(1)$. Этот алгоритм довольно сложно запрограммировать на БЕЙСИКе, и, кроме того, нужно еще обеспечить повторение цикла несколько раз для получения хороших результатов экспертизы.

Более разумный подход, позволяющий получить простую программу, работающую для любых значений N, заключается в случайном выборе примеров и их многократном повторении. Пусть мы хотим повторить их 100 раз, тогда

```
10 FOR COUNT=1 TO 100
20 I=INT (RND *N+1)
30 NEXT
```

Думаю, вы несомненно довольны, написав этот фрагмент программы. Но поскольку функции RND и INT могут меняться в зависимости от типа компьютера, давайте определим, как они реализованы, например, в IBM PC, использующей расширенный БЕЙСИК.

Функция RND генерирует случайные числа, которые больше или равны 0 и меньше 1. Следовательно, умножая их на N, получаем значения, большие или равные 0 и меньшие N. После прибавления 1 к этим числам результат оказывается большим или равным 1 и меньшим или равным N, но меньшим N+1. Функция INT формирует целое число, меньшее или равное по модулю числу, используемому в качестве аргумента, т. е. целое число, которое больше или равно 1, но меньше или равно N. Следовательно, мы можем применять этот фрагмент программы для выбора случайным образом примеров в диапазоне от 1 до N. С его помощью экспертная система лишь генерирует случайные числа и если вы действительно хотите заставить ее работать, то должны вставить этот фрагмент в цикл обучения и заставить ее повторять обучение до тех пор, пока не будет получен удовлетворительный результат.

По сравнению с тем, что мы углубленно изучали раньше, эта программа значительно проще. В ней используются VAR переменных и OUTCOMES возможных исходов.

```
5 N-10
10 FOR COUNT -1 TO N
20 DECISION-0
30 FLAG-0
40 I-INT(RND*N+1)
50 REM CORRECT является фактическим исходом для этих данных
60 CORRECT-EXAMPLES(VAR+1,I)
70 REM Определение значения DECISION, используя значение CORRECT:
80 FOR J-1 TO VAR
90 DECISION-DECISION+EXAMPLES(J,I)*RULES(J,CORRECT)
100 NEXT
110 FOR K-1 TO OUTCOMES
120 WRONG-0
130 REM Определение значения WRONG для решения с ошибочным
    исходом
140 IF K-CORRCT THEN 250
150 FOR J-1 TO VAR
160 WRONG-WRONG+EXAMPLES(J,I)*RULES(J.K)
170 NEXT
180 IF WRONG<DECISION THEN 250
190 REM Корректировка RULES, если они могут привести к
    ошибочному ответу
200 FOR J-1 TO VAR
210 RULES(J,K)-RULES(J,K)-EXAMPLES(J,I)
220 NEXT
230 REM FLAG - флажок, указывающий, что CORRECT следует
    скорректировать
240 FLAG-1
250 NEXT K
260 IF FLAG-1 THEN 265 ELSE 270
265 FOR J-1 TO VAR: RULES(J,CORRECT)-RULES(J,CORRECT)+
    EXAMPLES(J,I): NEXT
270 NEXT COUNT
```

Теперь, конечно, вы можете идти в свой отель, а ваша экспертная система будет делать всю домашнюю работу.

Возникает, однако, вопрос о том, создаст ли эксперт, когда вы вернетесь, тот же самый набор правил, что и вы, если бы вы терпеливо стучали по клавишам компьютера до конца своей жизни? Возможно, вы так и думаете, но на самом деле все обстоит иначе, потому что данные, предъявляемые системе, вводились по-разному.

Допустим, например, что изменение правил происходит лишь тогда, когда система делает ошибку. Она выдает возможный исход, когда будет уверена, что больше нет таких переменных, которые можно было бы ввести и под влиянием которых изменить сформированное суждение, основываясь на минимальных и максимальных значениях, которые вы ей любезно предоставили. Если это так, то система решает, что ничто больше не изменит ее решения и принятое решение правильное. Следовательно, система не изменяет ни одного правила. Представим теперь, что два правила с одним и тем же примером дали одинаковые значения DECISION, — возникает накладка.

В нашей предыдущей программе система не изменила бы правил, пока исход, который она угадала благодаря такой накладке, не оказался бы неверным. Если допустить, что она могла бы уклониться от изменения правил, то сделала бы это, хотя бы для того, чтобы оградить вас от необходимости трудиться в поте лица, предоставляя системе дополнительную информацию. Но вы не намерены избегать дополнительных затрат, связанных с вводом новых данных, поэтому вышеприведенная программа контролирует все ваши возможности и изменяет правила при наличии каких-то накладок до тех пор, пока не возникнет следующая накладка. Следовательно, изменение связей системы становится более частым.

Зададим системе в качестве объекта *Птицу*. После контроля текущих правил оказалось, что это могла бы быть *Птица*, но в равной степени это мог бы быть и *Планер*. Однако *Птица* была первым исходом в списке, тогда как *Планер* только дал одинаковый с *Птицей* результат, поэтому система предполагает, что это *Птица*. «Хорошо», — скажете вы. Правила при этом не изменятся, продолжая давать одинаковые значения для *Птицы* и *Планера*, если будет введена *Птица*. Система пока работает правильно.

Однако в программе, которую мы только что написали для экспертной системы, эксперт не сможет установить систему правил с одинаковыми значениями переменных для *Птицы* и *Планера*. Система будет изменять правила, пока существует ненулевое абсолютное значение разности для любого и каждого исхода. Это вовсе не означает, что один набор правил лучше, чем другой, — в конце концов, если система дает правильный ответ, кто будет спорить о том, каким способом он получен?

Если вы хотите, чтобы можно было использовать те же самые правила для обоих случаев, следует несколько усложнить программу. Например, можно изменить ее так, чтобы правила изменялись только при выполнении условия

$$WRONG > DECISION,$$

а не тогда, когда WRONG>=DECISION, что происходит, если условие WRONG<DECISION не выполняется в строке 180. Либо можно изменить условия таким образом, чтобы правила менялись только в случае

$$WRONG = DECISION \text{ AND } K < ANSWER.$$

Это означает, что правила изменяются, если найдено какое-то новое правило, которое открывает другую альтернативу, либо получен неверный исход, дающий такое же значение, что и верный исход, причем неверный исход происходит до того, как может наступить верный исход. Если бы первым наступал верный исход, то не было бы проблем, поскольку в случае верного исхода неверный исход не смог бы его заменить при условии, что полученные значения у них одинаковые.

Следует подчеркнуть, что на практике экспертная система может запросить у вас больше информации о переменных, прежде чем она примет окончательное решение. Система продолжает запрашивать информацию до тех пор, пока не будет уверена, что исключены любые неожиданности, влияющие на ее решение. За счет изменений, осуществляемых теперь чаще, чем раньше, фактическое число элементов массива RULES может изменяться. Поэтому возможности для вариаций различных значений тех или иных исходов могут увеличиться.

Экспертная система, контролируя минимальные и максимальные значения переменных, которые еще не введены в ЭВМ, констатирует наличие большей неопределенности, чем раньше. Она начинает задавать вам вопросы, пытаясь снять эту неопределенность.

Итак, если вы хотите «чистого» эксперимента, то пользуйтесь теми же самыми методами для обучения экспертной системы, которые применялись для «нормальной» работы. Фактически следует написать программу дважды либо использовать оба метода ввода информации. один — с клавишного пульта, другой — с использованием массива EXAMPLES (VAR+I, N) в одной программе. Право выбора остается за вами.

5.4. ПРЯМОЕ ОЦЕНИВАНИЕ

До сих пор мы считали, что наша система обучается так же, как и человек, сидящий за клавишным пультом, которому задают последовательно различные примеры. Не стоит изменять наше представление об этом, ведь оно работает. Главная задача, однако, заключается в том, чтобы найти набор решающих правил, позволяющий сделать приемлемые суждения. Мы, возможно, могли бы придумать лучшее решение, чем просто оставить систему без присмотра, полагаясь главным образом на генератор случайных чисел.

Когда мы рассматривали только два возможных исхода, можно было корректировать эти правила после каждого примера, в зависимости от того, привели они к успеху или нет. То же самое мы делали и с бо́льшим числом исходов. В этом случае мы игнорировали случайности и имели дело со всеми примерами, предъявляемыми системе:

```
10 FOR K=1 TO N: REM Допустим, что у нас N
   примеров
20 FOR I=1 TO OUTCOMES: REM Просматриваем
   все возможные исходы
30 FOR J=1 TO VAR: REM Просматриваем все воз-
   можные переменные
40 IF I = EXAMPLES (VAR+1, K) THEN
   RULES (J, I)+EXAMPLES (J, K) ELSE
   RULES (J, I)=RULES (J, I) —EXAMPLES (J, K)
50 NEXT: NEXT: NEXT
```

Этот фрагмент программы добавляет оценки, полученные для конкретного примера, к оценке, соответствующей данному правилу коррекции исхода, и вычитает их же из всех оценок, соответствующих правилам, принадлежащим другим исходам. Вопроса: сработают или нет текущие правила — здесь не возникает. В конце программы, когда все примеры исчерпаны, мы получаем набор правил, значения которых (теоретически) знаем достаточно точно.

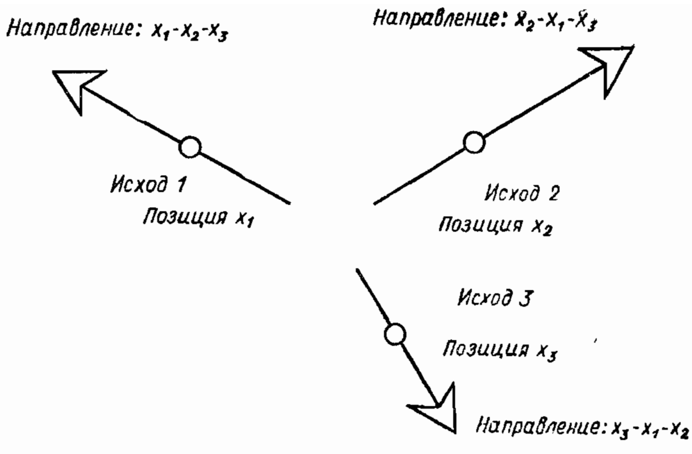


Рис. 5.1. Векторы направлений для трех возможных исходов

Допустим, у нас есть три исхода. Представим их в пространстве в виде углов треугольника (рис. 5.1). Три правила представлены линиями-векторами, имеющими определенные направления, указывающие на один из противолежащих углов треугольника. Углы треугольника на самом деле определяются средними значениями переменных для каждого из исходов. Допустим, что *Исход 1* имеет среднее значение x_1 для какой-то переменной, аналогично *Исход 2* — среднее значение x_2, а *Исход 3* — x_3. Тогда для *Исхода 1* вектор правила можно записать в виде (x_1—x_2—x_3), а для *Исхода 2* — в виде (x_2—x_1—x_3) и т. д. Каждый вектор правила старается повернуться, насколько это возможно, в направлении своего среднего значения, но благодаря другим средним значениям, которые в него входят со знаком минус, он также пытается повернуться и в противоположном направлении.

Вы, возможно, заметили, что фактически мы не использовали средние значения. Допустим, есть n_1 примеров с *Исходом 1*. Мы не пытались делить x_1 на n_1 для того, чтобы получить истинные средние значения. Рассмотрим сначала, что произойдет, если число примеров, имеющих аналогичные исходы, одинаково. В этом случае нет никакой разницы в том, будем ли мы осуществлять такое деление или нет. Имеют смысл только относительные значения каждого из них.

Предположим теперь, что существует различное количество примеров каждого типа. Если вы задали эксперту представительный набор примеров, содержащих, например, *Исход 1*, который случается в 2 раза чаще, чем другие исходы, то это полезная информация. Сведения о том, что один исход наступает в 2 раза чаще, чем другой, и одних примеров в 2 раза больше, чем других, возможно, заставит вас думать, что в 2 раза больше доверия (пользуясь научным термином!) заслуживает исход с бо́льшим числом примеров. Поэтому, получив средние значения для каждого исхода, мы «взвешиваем» их путем умножения на два.

В общем случае, когда имеется n_1 примеров x_1, мы можем перемножить n_1 и x_1. И если x_1 представляет собой среднее значение n_1 примера, то мы вернемся к тому, с чего начали. Простое сложение всех x_i было бы достаточно адекватным. Загвоздка в том, что объекты не всегда такие простые, как это порой кажется.

Если мы пытаемся предсказать, что это за объект — *Птица* или *Самолет*, и задаем эксперту по одному примеру того и другого, он затем вырабатывает определенный набор правил. Предположим теперь, что вы знакомите эксперта с двумя примерами одного типа. Что это означает? Тот же набор правил будет теперь достаточно хорошо выделять различные возможности. Нет причин думать, что один исход в 2 раза вероятнее другого. По сути те примеры, которые вы дали эксперту, не обязательно являются представительной выборкой. И это может вызвать некоторые проблемы. Оставляем на ваше усмотрение, как поступать в таком случае.

Для начала просто предположим, что значения, введенные вами в систему, представительны и выработаны на основе средних значений каждого исхода. Затем отдельно рассмотрим вопрос, важно или нет фактическое число примеров в каждом исходе. Если да, то вы должны, зная количество переменных для каждого исхода, вычислить соответствующие весовые характеристики. Но если вы чувствуете, что примеров, введенных в систему, недостаточно и вы просто заблуждаетесь на этот счет, то следует использовать обычные невзвешенные средние. Очевидно, для получения средних значений вы должны иметь (хранить) данные о количестве примеров каждого типа. Вы можете сделать это путем описания массива N(OUTCOMES), предназначенного для их хранения,

```
10 FOR I=1 TO N
20 N (EXAMPLES(VAR + 1, I)) = N (EXAMPLES
   (VAR+1, I))+1
30 NEXT
```

а затем получить средние значения путем деления RULES(I, J) на N(J).

Единственное неудобство этого метода, честно сознаемся, заключается в том, что он не всегда «работает».

Рассмотрим следующие значения в массиве VALUE(I):

	Птица	*Самолет*	*Планер*
Крылья	1	1	1
Клюв	1	0	0
Двигатель	0	1	0

Птица имеет *Крылья* и *Клюв*, *Самолет* — *Крылья* и *Двигатель*, а *Планер* — только *Крылья* (при указанном наборе переменных). Попытаемся выработать некий набор правил для RULES(I, J), используя средние значения и их разности:

	Птица	*Самолет*	*Планер*
Крылья	—1	—1	—1
Клюв	1	—1	—1
Двигатель	—1	1	—1

Далее, обработаем каждый из наборов переменных в свою очередь соответствующими правилами, чтобы найти значения DECISION(J) с учетом каждого значения VALUE(I), введенного в RULES(I, J).

Для каждого возможного исхода получаем следующие значения DECISION (слева указаны верные исходы):

	Птица	*Самолет*	*Плачер*
Птица	0	—2	—2
Самолет	—2	0	—2
Планер	—1	—1	—1

Совершенно ясно, что мы не получим ключ к разгадке *Планера*, даже если система без ошибок предугадывает другие возможные исходы.

Проблемы, очевидно, не возникли бы, если бы *Планер* характеризовался какой-то другой «уникальной» переменной, например *Гайдроуп*[1]. Однако в примере, который мы рассматриваем, такая переменная отсутствует, поэтому метод не срабатывает. Можно было бы все легко поставить на место, если учитывать *Планер* дважды. Тогда в строке *Крылья* мы имели бы — 2 для *Птицы* и *Самолета* и 0 для *Планера*, но ведь мы не знали этого, пока не проверили содержимое массива правил.

Итак, попытка прямого оценивания элементов массива правил может показаться полезной, но всегда следует помнить о том, что хорошо было бы проверить правила после получения результата, хотя бы просто путем тестирования системы еще на нескольких примерах. Эксперт может построить свои собственные правила в присущей ему манере, что будет скорее всего шагом назад по отношению к процессу обучения.

Глава 6. ОПРОБОВАНИЕ СИСТЕМЫ НА РЕАЛЬНЫХ ПРИМЕРАХ

6.1. ИСПОЛЬЗОВАНИЕ ВАШЕЙ ЭКСПЕРТНОЙ СИСТЕМЫ

Теперь вы получили заготовку полностью оснащенной экспертной системы общего назначения, с помощью которой можете изумлять своих друзей и коллег. Вспомним еще раз основные моменты построения экспертной системы.

1. Вы ввели в систему имена всех возможных переменных и сказали ей ваши соображения относительно минимальных и максимальных значений каждой из них. Эти переменные представлены либо числовыми данными, либо категориями типа Да/Нет. В любом случае они обрабатываются как числа.

2. Вы ввели в систему все имена возможных исходов.

---

[1] Канат (буксирный трос), использующийся для запуска *Планера*. — *Прим. пер.*

Каждый раз, когда эксперт[1] собирается выбрать один из исходов, он спрашивает ваше мнение о наиболее вероятном из них, чтобы, по крайней мере, один из возможных исходов в списке всегда можно было использовать. В идеале исходы должны быть взаимоисключающими, а сумма вероятностей появления каждого из них равна 1. Это означает, что произойти может только один исход, но он выбирается из тех исходов, которые должны произойти.

3. Вы задали системе несколько примеров (сколько смогли) вероятных входов и сообщили ей, какие исходы соответствуют какому входу.

4. Вы организовали обучение, во время которого эксперт анализирует эту груду примеров и модифицирует правила до тех пор, пока не получает набор правил, позволяющий всегда определить правильный ответ.

Теперь ситуация изменилась, вы действительно хотите использовать эксперта для чего-то серьезного. Очевидно, вы можете выполнить те же операции, что и во время обучения, например заняться предсказанием погоды.

День заканчивается, вы идете к своему эксперту, вводите несколько переменных, и он дает вам прогноз погоды. Затем на следующий день вы делаете то же самое. И так далее. Но допустим, что предсказание неверно. Хорошо, если это — период обучения, тогда прежде всего эксперт захочет узнать, было вчерашнее предсказание правильным или нет, чтобы скорректировать правила в случае ошибки. Теперь же, в нашем случае, нет особой нужды в удовлетворении своего любопытства.

Вы могли бы сообщить об исходе за последний день и затем запросить у системы погоду на сегодня. Но в этой спешке, упаковывая зонтик, плащ, веллингтоновские ботинки и другое обычное для английского лета «снаряжение», вы, возможно, забудете сообщить системе сведения о том, что случилось с ее последним предсказанием, либо просто захотите забыть об этом.

Не исключено, что вы пожелаете, чтобы эксперт выполнял для вас медицинскую диагностику. Вы организовали его обучение, сидя в своем медкабинете и принимая пациентов, ожидающих своей очереди за дверью.

[1] Здесь и далее в качестве эксперта имеется в виду экспертная система. — *Прим. пер.*

Входит первый пациент, вы включаете машину, вводите его симптомы в экспертную систему, и на экране высвечивается возможный исход. «Извини, старина, — заметите вы с глубокомысленным видом, — ваш день еще не настал, судя по той информации, которую выдала система, жить вам еще 7 дней. Следующий!» В этот момент вы должны засесть на неделю со следующим пациентом и так до тех пор, пока не представится возможность сообщить эксперту, был ли верен его прогноз (диагноз) относительно исхода для первого пациента. Подобная практика чревата тем, что вы растеряете ваших клиентов, если не предусмотрите заранее каких-то мер для выхода из данной ситуации. Кроме того, вы заставите людей покинуть ваш кабинет как можно скорее, если они узнают, что являются потенциальными вирусоносителями.

Дело в том, что хотя сама идея — дать эксперту возможность иметь обратную связь как можно дольше — хороша, но приходит время, когда вам захочется прекратить обучение и начать «делать дело» с помощью экспертных заключений. С точки зрения программиста, суть всего вышеописанного заключается в возможности запуска обучающей программы в предположении, что любой результат, полученный от эксперта, является верным. Эксперт больше не спрашивает вас, прав он или нет, и не корректирует правила.

В таком простом и ясном примере, как проблема распознавания объектов *Птица — Самолет — Планер*, где эксперт может быстро обучиться формировать только правильные ответы, не играет существенной роли, остановите вы процесс «подачи» обратной связи эксперту или нет. Однако в других случаях, таких как предсказание погоды, когда вы знаете или достаточно уверены, что ошибки будут случаться время от времени, эксперт не в состоянии функционировать без обратной связи. Вы можете ее организовать путем сохранения деталей тех примеров, которые вызвали ошибки, и заполнения результатов (возможных исходов), когда исход становится известным или вы чувствуете похожесть ситуации. На этом пути вы строите существенно больший набор фактических данных, периодически использующихся для обучения эксперта, чтобы быть уверенным, что он не теряет «чувства объекта».

Если теперь мы снова вернемся к примеру с предсказанием погоды, то увидим, что получатся разные результаты в случае реальной экспертной системы по сравнению с результатами, которые имели место в процессе обучения. Во время обучения каждый раз, когда возникали ошибки, происходила корректировка правил.

В табл. 6.1 приведен прогноз погоды экспертной системы за май 1982 г. при использовании правил выбора, сформированных в процессе длительного обучения. Никаких коррекций (дополнительно) не проводилось.

На первый взгляд, результаты не очень впечатляют — действительно, система сделала 9 ошибок за месяц. Но если мы воспользуемся теперь теми же правилами, только без обратной связи, то мы сделаем 6 ошибок в прогнозе за апрель (табл. 6.2). Следовательно, за эти два месяца (61 день) система допустила 15 ошибок, т. е. успех сопутствовал ей в 75 % случаев. Это, конечно, вроде бы и не столь хороший результат, но он существенно лучше, чем просто случайное угадывание. Судя по уровню надежности прогноза, если мы будем использовать такую экспертную систему для реального прогнозирования, то ошибки будут иметь место для каждых 1,75 дня в неделю.

Можно было бы рассматривать как некую дополнительную награду, что эксперт предсказал дождь, а его на самом деле не было. Плохо, если вы носите с собой зонтик, который не нужен! Ужасно, когда эксперт предсказал, что будет сухо, а на самом деле шел дождь! Но последнее случалось только 4 раза за март и апрель 1982 г., т. е. 1 раз в две недели. Поэтому эта новость не совсем уж такая плохая.

Относительно апрельской погоды интересно, однако, отметить, что на этой погоде экспертная система не обучалась. Допустим, проблема предсказания погоды относится к классу линейно-сепарабельных, хотя на самом деле это не так.

Но все-таки предположим линейную сепарабельность задачи предсказания погоды и будем считать, что эксперт обучался на погоде за март. Тогда после положенного срока обучения эксперт в состоянии безошибочно предсказать погоду в марте. Но это вовсе не означает, что он мог бы предсказать погоду в апреле, которая, возможно, имела бы совсем другие показатели. Тот факт, что система все же может создать довольно приличный набор предсказаний, является определенным

Таблица 6.1. Предсказание погоды за март 1982 г.

День месяца	Фактическая погода	Предсказанная погода
1	Дождь	Дождь
2	»	»
3	»	»
4*	Сухо	»
5*	»	»
6*	»	»
7	Дождь	»
8	»	»
9	»	»
10	»	»
11	»	»
12	»	»
13*	Сухо	»
14*	Дождь	Сухо
15	»	Дождь
16	»	»
17*	Сухо	»
18	Дождь	»
19	»	»
20	»	»
21*	Сухо	»
22	»	Сухо
23	»	»
24	»	»
25	Сухо	Сухо
26	»	»
27	»	»
28*	Дождь	»
29*	Сухо	Дождь
30	Дождь	»
31	»	»

* В эти дни прогноз оказался ошибочным.

Таблица 6.2. Предсказание погоды за апрель 1982 г.

День месяца	Фактическая погода	Предсказанная погода
1	Дождь	Дождь
2	Сухо	Сухо
3	»	»
4*	Дождь	»
5	»	Дождь
6	»	»
7	»	»
8	»	»

День месяца	Фактическая погода	Предсказанная погода
9*	Сухо	Дождь
10	»	Сухо
11*	»	Дождь
12*	»	»
13*	»	»
14	»	Сухо
15	»	»
16	»	»
17	»	»
18	»	»
19	»	»
20	»	»
21	»	»
22	»	»
23	»	»
24	»	»
25	Сухо	Сухо
26	»	»
27	»	»
28*	Дождь	»
29	Сухо	»
30	—	—

* В эти дни прогноз оказался ошибочным.

достижением, т.е. существует нечто общее в процессе предсказания погоды, что экспертная система смогла сделать для нас. Она может учесть совершенно новые данные и получить что-то похожее на приемлемое решение, даже если мы не знаем сами как предсказать завтрашнюю погоду, и сообщим экспертной системе самый общий подход к этой проблеме и несколько примеров, чтобы помочь ей иачать работать.

6.2. РЕЗЕРВНЫЕ СУЖДЕНИЯ

Если в процессе выработки решения наша экспертная система не смогла составить собственного мнения о самом лучшем исходе из всех возможных, она в состоянии заявить о своем поражении или выдать какую-то простую догадку, хотя сделать это для нее очень не просто.

Допустим, мы снова вернулись к задаче распознавания объекта *Птица — Самолет — Планер,* и на ранней стадии обучения оказалось, что все три исхода дали одинаковые значения. В том варианте эксперта, каким он был выполнен до сих пор, эксперт сделал бы предположение относительно вероятного исхода, если бы такой произошел, а если бы это оказалось неверным, то скорректировал бы правила. Но вы могли бы сказать, что поскольку все правила дают одинаковый результат, то эксперт реагирует: «Не знаю», — и не отгадывает исход, которого на самом деле не знает.

Ну что же, прекрасно. Конечно, вы могли бы запрограммировать это. Небольшое изменение программы и, если оказалось, что существует больше, чем один вероятный исход, она будет печатать: «Не знаю», — или, еще лучше, выдаст список нескольких наиболее вероятных исходов. Это дало бы вам гораздо больше информации, чем раньше, относительно того суждения, которое имело место. Поэтому вы могли бы включить такой фрагмент в вашу программу. Но будет ли последняя когда-нибудь использовать этот дополнительный фрагмент? Иногда будет. Например, в нашем примере с *Птицей—Самолетом—Планером* система может дать резервный ответ «Я не знаю» в процессе обучения. Но как только определятся правила системы, сразу же будут исключены любые резервные суждения вообще, потому что она всегда будеть выдавать правильные ответы. Это как раз и соответствует тому, что вы от нее хотели.

Вновь привлечем ваше внимание к погоде. Каковы, по-вашему, шансы существования какого-то конкретного набора исходов, относящихся к погоде, при котором есть возможность возникновения двух или более одинаковых суждений на основе заданного набора правил? Трудно дать точный ответ на этот вопрос, приблизительно они равны 0. Значения вводимых в систему переменных постоянно меняются. Будучи вещественными переменными, они могут принимать (в определенных пределах) бесконечное множество значений. Возьмите несколько таких переменных, объедините их с заданным правилом и в результате получите тоже вещественные переменные. Возьмите другие, отличные от этого правила, объедините с имеющимися переменными и оцените, нет ли среди них одинаковых ответов. В общем случае их не будет. Даже если два ответа окажутся очень похожими, они все равно в точности не совпадут.

Далее, если вы думаете о правилах как о методе размещения поверхности между различными исходами, то заметите, что до тех пор, пока поверхность действительно разделяет исходы, неважно, где именно она проходит. Точно так же все равно, какие значения используются в различных правилах, пока последние выполняются. Поскольку эти значения могут варьироваться, не исключено изменение любого из равенств без нарушения основной работы эксперта.

Естественно, скажете вы, совсем необязательно, чтобы эти значения были в точности теми же самыми для резервных суждений. Вполне достаточно, чтобы они были близкими. Но какова степень этой близости? И как это связано с машиной? Если вы разделите все ваши переменные, скажем, на 100, прежде чем применить правила, то и разности между значениями, заданными каждым правилом, также будут уменьшены в 100 раз. Следовательно, эти переменные становятся более близкими. По крайней мере, они дают похожие ответы, и поэтому эксперт мог бы широко использовать резервные суждения. Очевидно, что это не так. Но тогда что рассматривать в качестве корректной альтернативы работы системы?

Если учесть, что до сих пор мы оперировали с понятием «категория», то можно было бы ввести еще один дополнительный исход, а именно *Не знаю*. Но как обучить систему применять это понятие? Если, допустим, эксперт спрашивает вас, будет ли правильным исход *Не знаю*, какой ответ вы ему бы подсказали? Допустим, что вы также ответили ему, что не знаете. Проблема, в сущности, не в том, чтобы решить, знает или не знает эксперт нужный ответ. Важно выяснить, до каких пределов он верит в каждый из допустимых исходов в отдельности. Его окончательным ответом должен быть тот исход, в который он верит больше всего. Но когда мера этой веры близка к тем, которые соответствуют другим исходам, вы предпочитаете резервное суждение или, по крайней мере, сообщаете системе об этом.

Однако с нашей экспертной системой это фактически невозможно. Она спроектирована таким образом, чтобы принять любой вход по любой проблеме и дать наилучший ответ. Если вы хотите знать, какой уровень доверия вы можете вложить в конкретный ответ, то для этого существует только один путь — ввести некоторые данные в систему и установить, насколько часто ответы будут правильными.

Рассмотрим, например, случай *Птица—Самолет—Планер*. После обучения система каждый раз давала правильные ответы. Поэтому ее мнение имеет уровень доверия, равный 1. В случае предсказания погоды она была права в 75 % случаев, следовательно, вероятность правильного ответа составляла 0,75. Заметим, что все эти значения получены в результате обобщения нескольких ответов. Один отдельно взятый конкретный ответ дается с почти абсолютной уверенностью, тогда как другой — с большой долей неуверенности. Проблема заключается в том, что нам неизвестна степень уверенности. В общем случае у нас нет средств, чтобы выяснить это.

6.3. ПРОБЛЕМА РАССТОЯНИЯ

Можно с уверенностью сказать, что большинство читателей, видимо, думают, что мы можем работать по следующему алгоритму:

у нас есть приличный набор правил в экспертной системе, сформированный в результате длительного обучения;

затем мы задаем системе новый вопрос. Она формирует список всех возможных исходов с указанием против каждого соответствующих значений, которые были получены с помощью конкретного набора правил;

наконец, эксперт наверняка выберет из этого списка исход с наибольшим значением. Это значение и соответствующие значения для других невыбранных исходов будут отражать вероятности «правильности» каждого исхода.

Печально, но это далеко не так. Эксперт был натренирован выбирать максимальное значение, но ему ничего не сказано о том, что в действительности означают эти числа с точки зрения искомых вероятностей. Наибольшее значение характеризует наиболее вероятный исход, но для эксперта совершенно не ясно, насколько этот исход более вероятен, чем другие.

Все вышеизложенное связано с проблемой расстояния и измерения расстояний. Именно этим мы теперь и займемся.

В рассмотренной нами обучающей системе эксперт должен поместить разделяющий «забор» между различ-

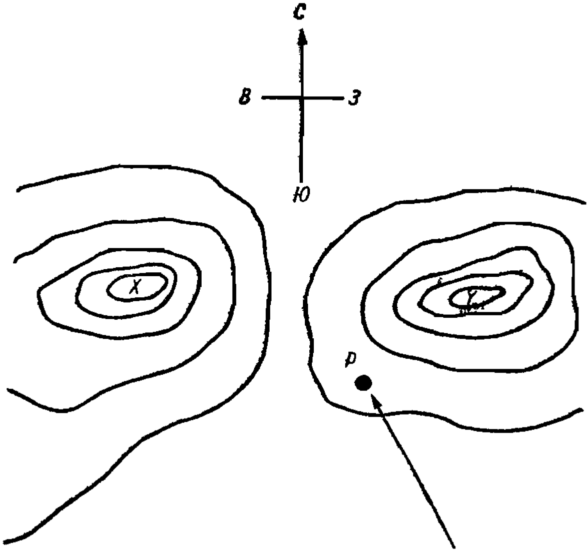


Рис. 6 1. Физическая аналогия для измерения расстояния

ными исходами. До тех пор, пока такой «забор» аккуратно отделяет допустимые исходы один от другого, не играет роли, где он на самом деле расположен. Существенно лишь, по какую сторону «забора» находятся эти переменные, и совершенно не важно, насколько близко они расположены к «забору». В этом смысле концепция расстояния вообще не возникает как таковая.

Чтобы помочь уяснить эту концепцию, представим, что перед нами карта некой загадочной страны (рис. 6.1). На ней нанесены две оси (север—юг, восток—запад) и помечены две позиции (значки X и Y). На карте также проведены контурные линии, соответствующие определенной (одинаковой) высоте места над уровнем моря. Позиции X и Y, как мы видим, соответствуют вершинам гор. Итак, у нас теперь есть две горы X и Y, а точка P, представленная на этом рисунке, расположена на горе Y. Это значит, что любой человек мог бы, взглянув на карту, сказать, какой горе принадлежит точка P.

Теперь разберем некоторые способы, которыми наш эксперт может воспользоваться для решения указанной проблемы.

Программа (или экспертная система) в процессе обучения устанавливает разделяющий «забор» между двумя горами, и так как точка P лежит на горе Y, то граница будет считаться правильно установленной, если точно классифицировать точку P. Но система ничего не говорит нам, до какой степени эту точку можно считать принадлежащей горе Y, будет ли это верно, когда точка P находится на вершине горы или лежит у ее подножья. Самое лучшее, что она может нам сообщить, это то, что данная точка не принадлежит горе X.

Рассмотрим теперь измерение дистанции. Простейший способ, который сразу же приходит на ум, заключается в определении расстояния между точками P и X и соответственно P и Y. Затем делается вывод, что точка P принадлежит той горе, расстояние до которой меньше. Это не такой уж плохой способ, поэтому мы будем следовать ему и в дальнейшем.

К сожалению, возникает вопрос: как обойтись без линейки, поскольку компьютер не имеет карты как таковой, а использует последовательность координат, соответствующих замерам вдоль двух осей.

Точка X характеризуется координатами в направлении Север—Юг (СЮ) и Восток—Запад (ВЗ), т. е. ($СЮ_X$, $ВЗ_X$), а точка Y — соответственно ($СЮ_Y$, $ВЗ_Y$). Для точки P имеем ($СЮ_P$, $ВЗ_P$). Расстояние $d^2 = (СЮ_X - СЮ_P)^2 + (ВЗ_X - ВЗ_P)^2$ от точки P до точки X называется евклидовой мерой расстояния. То же самое можно записать и для расстояния от P до Y. После этого мы вправе реализовать для точки P классификатор на основе минимального евклидова расстояния.

Тот факт, что этот пример с горами рассмотрен в физически реальном пространстве, а наша проблема существует в n-мерном пространстве описаний, не играет особой роли с точки зрения метода решения задачи. В нашем случае необходимо сформировать «позицию» каждой из категорий, обозначающих какой-либо исход. Если рассмотреть случай *Птица—Самолет* и учесть только две переменные (*Клюв* и *Двигатель*), то получим для *Птицы* координаты (1, 0), поскольку у нее есть *Клюв* и нет *Двигателя*, для *Самолета* — (0, 1), так как у него нет *Клюва*, но есть *Двигатель*. Теперь рассмотрим объект P с координатами (1, 0). Его можно классифицировать как *Птицу*, поскольку этот исход соответствует минимальному евклидову расстоянию до точки P. Если в массиве

RULES мы будем хранить позицию каждого исхода, то можно ввести новый объект (VAR переменных) в массив VALUE и вычислить

```
10 FOR I=1 TO VAR
20 DISTANCE = DISTANCE + (VALUE (I) — RULES (I, J))^2
30 NEXT
```

а затем считать, что J-й исход из всего набора исходов OUTCOMES имеет минимальное расстояние DISTANCE. Ошибки не произошло. Этот метод сходится гораздо быстрее, чем метод, в котором используется обучающий алгоритм.

Сложность ситуации, видимо, заключается в том, что мы не представляли себе достаточно четко, что же должна делать экспертная система. Когда мы спрашивали, к какой категории (горе) принадлежит точка P, мы фактически не определили соответствие между исходом и категорией и потому интересовались, какое расстояние короче: (P—X) или (P—Y) либо к какой горе — X или Y — принадлежит точка P? И если в примере *Птица—Самолет* характер вопроса не имел никакого существенного значения, то в примере с предсказанием погоды дело обстоит иначе.

Действительно, в примере с *Птицей—Самолетом* от нас требовалось найти минимальное из двух расстояний: (P—X) или (P—Y). Мы имели возможность решить эту задачу, поскольку переменные являлись функциями точки в пространстве описаний, т. е. существовали только в какой-то отдельно взятой точке, тогда как структура пространства между точками не принималась во внимание. Если же мы работаем с непрерывными переменными, такими, например, как *Уровень осадков*, то они не являются больше функциями точки, а определяются всей занимаемой ими областью. Кроме того, чтобы удостовериться, где фактически находятся вершины гор, достаточно было простых вычислений.

В общем случае положение вершин гор лучше всего определяется путем нахождения среднего значения переменных для каждого исхода. Если мы имеем n (наблюдений) для заданного исхода, то среднее значение

$$m = (x_1 + x_2 + x_3 + \ldots + x_n)/n.$$

Следовательно, независимо от типа переменных (дискретные или непрерывные) можно вычислить средние значения для каждого исхода, поместить их в массив RULES, а затем осуществить классификацию с помощью определения минимального евклидова расстояния. Основная проблема тем не менее остается нерешенной. Мы вычислили расстояния до вершин гор, но не пытались ответить на вопрос, к какой горе принадлежит точка P.

Предположим, что эти горы разного размера, одна из них — настоящая гора, а другая — небольшой бугорок. Допустим, точка P располагается на расстоянии 1 м от центра бугорка и 150 м от центра горы. Вопрос тот же: принадлежит точка P горе или бугорку? Используя классификатор на основе минимального евклидова расстояния, получаем, что точка P расположена на бугорке, а не на горе. Правильно ли это? Ведь ясно, что 1 м для бугорка — довольно большое расстояние, тогда как 150 м для горы может означать лишь полпути к вершине.

Где начинаются и заканчиваются контурные линии? Если мы посмотрим на географическую карту, то интуитивно сможем оценить размеры горы, когда отвечаем на подобные вопросы. Именно это мы и пытаемся сделать сейчас при попытке классификации объектов. В терминах переменных линии уровня являются функциями распределения вероятностей этих переменных. Функция нормального распределения (рис. 6.2) является одним из примеров возможного распределения.

Точка X является средним значением переменной, распределенной по закону нормального распределения. Если мы представим себе двухмерную функцию нормального распределения, то сверху она будет выглядеть как гора с соответствующими линиями уровня.

Каждое нормальное распределение характеризуется стандартным отклонением

$$sd = \sqrt{((x_1 - m)^2 + (x_2 - m)^2 + (x_3 - m)^2 + \ldots + (x_n - m)^2)/n}.$$

Сначала вы вычисляете среднее значение m для n наблюдений каждой переменной, а затем его стандартное отклонение. Последний параметр может служить относительной мерой для сопоставления наших гор, так как она характеризует степень протяженности переменных. Следовательно, мы можем сделать наши горы одного и того же размера, разделив их на стандартное отклонение. Этот процесс носит название нормирования на

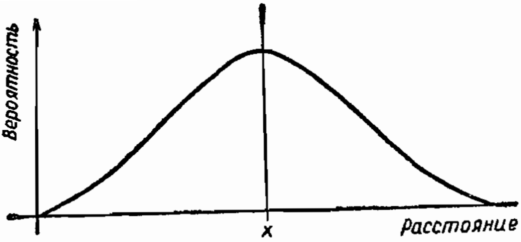


Рис. 6.2. Нормальная функция распределения вероятностей

единичную дисперсию. Используя некоторый набор данных, можно определить среднее значение, дисперсию и стандартное отклонение, а затем новое нормированное расстояние

$$d^2 = ((x - m)/sd)^2.$$

Эта формула позволяет реализовать классификатор на основе минимального нормированного расстояния. Она, конечно, поможет эксперту в выработке решения, но поможет ли она определить, с какой вероятностью верны его утверждения?

Предположим, что переменные были действительно нормально распределены. Стандартное отклонение для такого распределения можно взять из таблиц (или рассчитать на компьютере, что несколько сложнее), чтобы определить соответствующую ему вероятность того, что данное наблюдение принадлежит горе X (или в статическом смысле вычислить, какова вероятность принадлежности заданного наблюдения к данной популяции). Используя такой подход, система будет в состоянии высказывать утверждения относительно возможного исхода и судить о соответствующей вероятности. Однако проблема заключается в том, что многие переменные не подчиняются нормальному закону распределения и для них затруднительно определить адекватный закон распределения.

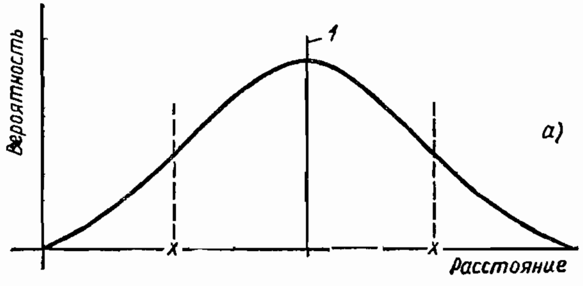
Вернемся к примеру *Птица—Самолет* с двумя переменными — *Клювом* и *Двигателем*. Очевидно, что эти переменные не распределены по нормальному закону. Будучи функциями точки (дискретными функциями), они не напоминают горы — скорее, фонарные столбы: все в них содержится концентрированно, в одном месте. Фактически это в чем-то облегчает жизнь, например можно обойтись без таблиц — вероятности либо 0, либо 1 в зависимости от того, какие переменные и какие исходы мы имеем. Таким образом, система сообщает свое мнение относительно исхода, вероятность которого точно определена. Расстояние, используемое в качестве меры при классификации исходов, всегда больше или равно 0, а вероятность того, что данная классификация правильна, будет равна 1, если расстояние 0, или 0, если расстояние больше 0.

Сложности возникают в том случае, когда мы не знаем, какой вид распределения имеет данная переменная, например *Уровень осадков*. С большой уверенностью можно сказать, что распределение отличается от нормального, так как оно несимметрично (рис. 6.3) относительно среднего значения. Эта переменная не является дискретной и должна иметь непрерывное распределение. Какое же оно? Если нам известен ответ и мы реализовали его в нашей экспертной системе, то последняя очень хорошо предсказывает погоду, но нам от этого не будет легче, если законы распределения различны.

Подход, основанный на измерении расстояния, достаточно хорош, однако переход от расстояний к вероятностным утверждениям относительно исходов является трудной задачей, поскольку расстояние и вероятность — понятия совершенно разные. Проблема расстояний — фактически одна из наиболее общих и сродни проблеме подобия.

Допустим, мы имеем заданный набор переменных и решаем проблему возможного исхода. Мы можем запустить систему с тем набором значений, на котором она была обучена, чтобы определить, сколько элементов из данного набора имеют те же значения, что и наши переменные. Затем заставим систему принять решение путем выбора такого исхода, который содержал бы наибольшее число примеров с одинаковыми значениями переменных. Это не такой уж плохой метод для определения набора данных. Он достаточно прост и основан на данных замера расстояний, поэтому мы можем использовать категории или другие методы описания данных, которые нам больше подходят. Для примера *Птица—Самолет—Планер* этот метод вполне пригоден.

Но вернемся к погоде. Здесь шанс найти даже один набор идентичных показаний очень мал (или требует



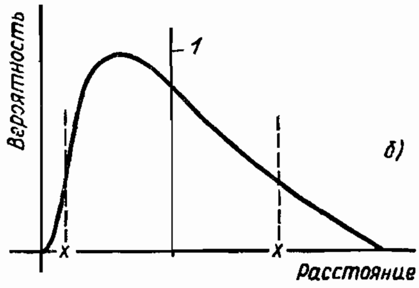


Рис. 6.3. Симметричная (*а*) и асимметричная (*б*) функции распределения (*1* — среднее значение)

длительных наблюдений). Как же тогда мы сможем измерять степень подобия? Снова необходимы измерения расстояний со всеми вытекающими из этого проблемами. Жаль, ведь если бы описанный метод категорий «работал», то мы фактически могли бы назначить вероятности возможным исходам, основываясь на числе различных случаев, соответствующих той или иной категории. Конечно, если у вас какая-то специфическая задача, удобная для реализацией этим методом, то он покажется вам очень простым, легким для понимания и приемлемо точным. Его можно было бы успешно использовать тогда, когда переменные разбиваются на ряд категорий, каждая из которых определяет какой-то конкретный исход. В примере *Птица—Самолет—Планер* такие категории точно характеризуют конкретный исход. Однако, если вы имеете список симптомов заболеваний и хотите указать комплекс конкретных жалоб, а не назвать какую-то одну определяющую жалобу, этот метод даст те же результаты, что и другие (в том числе и наш исходный метод), а также дополнительно определит системе пределы, в которых справедливы полученные суждения.

Очевидно, целесообразно использовать исходный метод для выработки суждения, а затем просмотреть последние записи с целью выявления отдельных случаев, которые могли бы оказаться идентичными, что указало бы на возможность рассмотрения других суждений.

6.4. НЕОБХОДИМОСТЬ ПОНИМАНИЯ СОБСТВЕННЫХ ПРОБЛЕМ

Суть подхода к построению экспертной системы состоит в создании метода, который будет работать с приемлемой надежностью в любых обстоятельствах, т. е. предсказывать достаточно приемлемые исходы независимо от задачи, которую вы поставили перед ним и, более того, независимо от того, понимаете вы сами эту задачу или нет.

Возьмем, например, прогноз погоды. Возможно, что вы — сотрудник службы погоды, но никогда ее не предсказывали и не знаете, как это сделать. Однако вы в состоянии разработать экспертную систему, которая помогла бы вам. Это же можно сказать и о других возможных задачах. Вы можете использовать ту же самую базовую экспертную систему для решения широкого круга задач.

Если же вы действительно столкнулись с задачей, которую сами хорошо понимаете, у вас будет шанс сделать систему много лучше и точнее, написав для нее удачную программу. Весь смысл в том, что если вы действительно поняли задачу, то вам не нужно выслушивать чьи-то советы, как ее реализовать. Не так ли?

Глава 7. ЭКСПЕРТНАЯ СИСТЕМА ДЛЯ РЕШЕНИЯ ЛЮБЫХ ВОЗМОЖНЫХ ЗАДАЧ

7.1. МЕТОД УЗЛОВ

Все экспертные системы, которые мы рассматривали до сих пор, были узко специализированными. Они могли стать экспертами в решении любых возможных задач. В это трудно поверить, но это действительно так. Нет смысла терять время, чтобы убедиться в этом.

Рассмотрим, например, систему с семью переменными (*Крылья, Клюв, Двигатель, Минимальная температура, Максимальная температура, Дождь, Солнечно*) и пятью (*Птица, Самолет, Планер, Завтра дождь, Завтра сухо*) возможными исходами.

Определив ее, начнем сеанс обучения. Сначала научим систему секретам распознавания ситуации *Птица—Самолет—Планер,* предъявляя ей объекты с *Крыльями* и *Двигателями.* Если она будет спрашивать о других переменных, вводите нули или нечто произвольное. Когда вам покажется, что система «схватила» суть объектов, обучите ее распознавать погоду и, если она будет спрашивать вас о *Крыльях, Клюве* или *Двигателе*, снова вводите нули. Затем предоставьте системе возможность попрактиковаться. Теперь, хотите верьте, хотите нет, вы получите эксперта, который одинаково хорошо предсказывает погоду и идентифицирует летающие объекты.

Расширение подобной практики в любой области человеческой деятельности вполне очевидно. От вас требуется только добавлять новые переменные и новые возможные исходы и устраивать различные сеансы обучения. В итоге ваш эксперт станет специалистом во всех мыслимых областях человеческой деятельности с учетом, конечно, ограничений на память компьютера. «Да, действительно, — воскликните вы, — это удивительно!»

Тот, кто проектирует подобные системы для удовлетворения своего любопытства (когда вокруг много других проблем, нуждающихся в решении), не тратит чужого времени. Но вы, возможно, решите, что ваша система работает слишком медленно. И вас начнет раздражать, что она будет продолжать спрашивать вас о *Клюве* и *Двигателе*, тогда как вам хотелось бы узнать прогноз погоды. Одной из трудностей такой схемы организации экспертной системы является то, что эксперт не только «оказывается» в состоянии решить любую задачу, а действительно будет стараться решить любую задачу каждый раз, когда вы включаете эту никуда не годную игрушку. Конечно, с его стороны столь великодушная попытка выглядит благородной. Но восхищение, которое кто-то будет испытывать от этого, быстро надоест.

Очевидно, необходимо иметь две версии такой экспертной системы. Одна из них должна использоваться для идентификации летающих объектов, а вторая — для предсказания погоды. Можно предусмотреть и другие версии, настроенные по вашему желанию на решение каких-то иных вещей. Думается, нет смысла соединять их вместе потому, что у них нет никаких точек соприкосновения.

Приведенные рассуждения подводят нас к понятию узлов.

То, что мы спроектировали, укладывается в концепцию узла. Один узел будет использоваться для идентификации летающих объектов, другой — для предсказания погоды. Оба узла, однако, не связаны, так как в этом нет никакой необходимости. Каждый узел имеет несколько входов (переменных) и выходов (возможных исходов), причем они могут быть соединены друг с другом, если мы захотим это сделать.

Допустим, у нас есть два узла, предсказывающие погоду. Мы можем соединить выход одного узла с входом другого. Тогда первый узел будет предсказывать погоду на вторую половину сегодняшнего дня, а второй — погоду на завтра. В результате эксперт, сформировав краткосрочный прогноз на вторую половину дня, мог бы использовать его как один из входов для долгосрочного (на завтра) предсказания.

Вернемся, однако, к одному большому узлу, который играет роль эксперта в любой области. Этот узел (рис. 7.1) можно рассматривать как вырожденный случай параллельного процесса. Вы сообщаете ему то, что он хотел бы знать, а он в свою очередь что-то вам отвечает. Разбивая один большой узел на два маленьких: предсказание погоды и идентификацию летающих объектов, вы преобразуете последовательный процесс в два параллельных, причем последние не могут выполняться одновременно. Поскольку у этих процессов нет общих точек соприкосновения, они изолированы друг от друга. Но если бы они были связаны, то мы имели бы некое подобие последовательного процесса, организованного из последовательности узлов (в данном случае двух), имеющих структуру дерева, если вы не против такого обобщения. Каждый узел тогда можно было бы установить и обучить по отдельности.

В теории это не дает преимуществ перед большим параллельным процессом, на практике же помогает вам сэкономить время, а также получить большие информации о том, что происходит.

Вернемся вновь к порядком надоевшему вопросу о по-

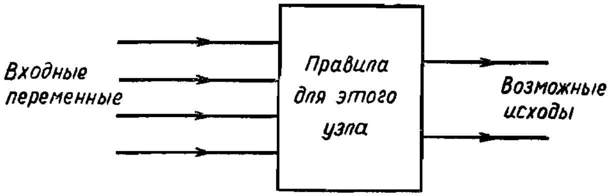


Рис. 7.1. Отдельный узел экспертной системы

годе. Конечно, если у вас есть узел, который сообщает вам о том, собирается завтра идти дождь или нет, это хорошо. Но он мог бы сообщить вам и что-то еще. Например сказать, будет ли сегодня тепло? Вот уж воистину, один (умный) должен отвечать на все, что захочет от него любой дурак! Правда, мы имеем дело с компьютером, а это несколько более сложный случай.

Установим другой узел и начнем обучать его. Научим его сообщать, что сегодня тепло, если температура выше, скажем, 10 °С. Он должен усвоить это без особых усилий, даже если вы лично сомневаетесь, что 10 °С — это тепло.

Вернемся к нашему первому узлу и подадим на его вход четыре переменные: *Максимальную температуру, Минимальную температуру, Уровень осадков, Солнечное время дня.* Пятая переменная представляет собой возможный исход — *Сегодня тепло*, который и будет выходом первого узла. Он, очевидно, равен 1, если *Максимальная температура* больше 10 °С и 0 в противном случае. Затем обучите второй узел предсказывать *Дождь* или *Сухо* на завтра.

Этот процесс показан на рис. 7.2. Вы увидите, что по какой-то причине поведение системы в целом при наличии пяти переменных будет отличаться от ее поведения в случае четырех переменных. Похоже, что система получила некоторую дополнительную информацию, хотя фактически вы ей дали только сведения о точной температуре, которые система уже имела.

Эта дополнительная информация на самом деле предназначена не для вас. Речь идет всего лишь о том, был или нет предыдущий день теплым. Не так уж много, но все-таки. Ее особенность в том, что она появилась в результате нелинейного преобразования *Максимальной температуры.* Поскольку эксперт до сих пор работал

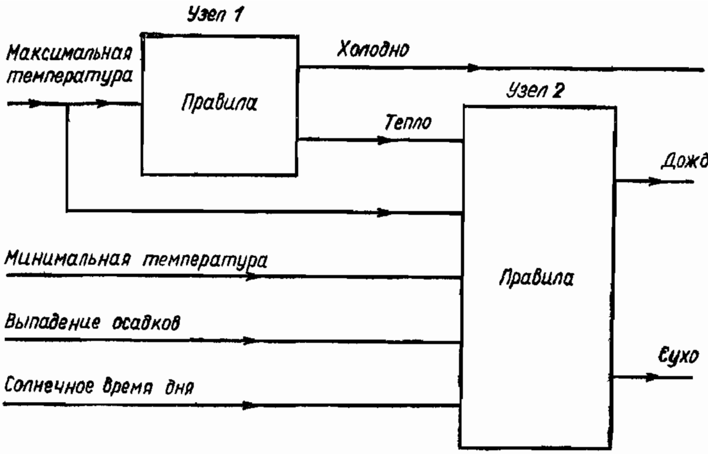


Рис. 7.2. Добавление еще одного узла

только с линейными преобразованиями, вы снабдили его тем, чего он до сих пор не имел и не смог бы сформировать самостоятельно.

Если вы глубоко не убеждены в этом, то попробуйте предположить, что эксперт имеет только четыре переменные (*Минимальную температуру, Максимальную температуру, Уровень осадков* и *Солнечное время дня*) и из них формирует набор правил для предсказания погоды. Затем вы спрашиваете о погоде и система, игнорируя три другие переменные, берет *Максимальную температуру* и умножает ее на число, взятое из сформированного набора правил. В соответствии с полученным результатом она и делает свой прогноз. Добавим теперь информацию о том, что *День теплый* (или холодный). Этот вход для второго узла равен 0, если *Максимальная температура* ниже 10 °С, т. е. алгоритм вычислений в этом случае не меняется. Но если *Максимальная температур*а окажется выше 10 °С, то на этом входе появляется 1. Затем это значение умножается на некоторое число из нового набора правил и добавляется к выходу второго узла. Следовательно, как только температура увеличивается, процесс принятия решений неожиданно переходит на другой (более высокий) уровень. Математически такой вход описывается единичной ступенчатой функцией, которая и задает нелинейное преобразование данных.

Итак, вы получаете дополнительную переменную, действительно новую переменную, и, конечно, вправе сказать, что у вас уже пять переменных. Каждый раз, когда эксперт спрашивал: «Сегодня теплый день?» — вы могли бы просто взглянуть на *Максимальную температуру* и присвоить пятой переменной соответственно 0 или 1. Но вы могли бы также дать возможность эксперту сделать эту работу за вас путем установки специального узла. Ему фактически требуется на входе только одна переменная — *Максимальная температура.* Прежде всего следует обучить этот узел распознавать теплый (или холодный) день, чтобы в первую очередь ответить на этот вопрос. Затем выбирается возможный исход *(День теплый* или наоборот) и последний используется в качестве дополнительного входа в следующем узле, который и должен предсказать, будет ли *Дождь.*

Это похоже на процесс подключения блоков стереосистемы Вы берете выход с одного блока и соединяете его с входом другого блока. *Максимальная температура* здесь ассоциируется с таким проводом, который подается на оба блока. И, как и в случае со стереосистемой, вы, пытаясь наладить ее работу, будете до бесконечности продолжать соединять и разъединять блоки. К счастью надеюсь, она все-таки начнет работать.

Указанные рассуждения заставляют внести изменения в вашу программу, но, надеюсь, они не будут слишком большими. Допустим, вы решили, что программа будет иметь N узлов. Тогда для описания переменных, исходов и правил вы должны написать

DIM VAR$ (VAR, N), VALUE (VAR, N), OUTCOMES$ (OUTCOMES, N), RULES (VAR, OUTCOMES, N)

Другими словами, достаточно добавить еще одну размерность для каждого используемого массива переменных, а затем ссылаться на соответствующий узел, просто указывая его номер N.

Например, вы пришли к узлу, который формирует суждение OUTCOMES (J, N1) — возможный исход J в узле N1, тогда программа должна просмотреть список всех переменных, чтобы определить, не является ли этот возможный исход (выход рассматриваемого узла) входом какого-то другого узла нашей системы. Если да, то VALUE (I, N2) = 1 для переменной I на узле N2, если полагать, что VAR$ (I, N2) = OUTCOMES$ (J, N1). Это напоминает автоматическую проверку условия: установлен ли некий переключатель в нашей системе в положение «включено»?

Если такой переключатель включен, то это эквивалентно установлению другого значения переменной в системе. Следующим шагом алгоритма будет проверка возможности системы формировать остальные выходы. Если она сможет это сделать, то мы имеем другой возможный исход (выход сработавшего на этом шаге узла), который является входом очередного узла, и т. д. до тех пор, пока она не придет к чему-то, по-вашему мнению, полезному. Например, к ответу на вопрос, который вы перед ней поставили.

Конечно, фактическое отличие в поведении системы, имеющей больше одного узла, состоит в том, что на каждом узле происходит нелинейное преобразование данных. Однако это отличие не является следствием присутствия узлов как таковых.

Рассмотренная ситуация неожиданно заключает в себе одно важное преимущество. Узлы как бы приоткрывают нам, что происходит в голове эксперта. На выходе каждого узла формируются возможные исходы, которые вы можете рассматривать как некоторые промежуточные решения. Например, вход: *Максимальная температура* сегодня; выход: *Сегодня тепло*; вход: *Минимальная температура, Уровень осадков, Солнечное время дня*; выход: Держу пари, что завтра будет *Дождь.*

Как в точности будет вести себя система, зависит от ее программы. Как видно, вопрос о том, будет ли сегодня теплая погода, звучит далеко не так тривиально. Не говоря о том, что этот вопрос обсуждается почти всеми, у людей вошло в привычку думать об этом каждый раз, когда они слышат упоминание о погоде.

Нет сомнения в том, что любые ваши экспертные системы будут иметь дело с большим числом узлов, переменных и возможных исходов и решать задачи существенно более сложные, например нейрохирургические. Решая подобные задачи, вы избежите многих непредвиденных осложнений, если будете заставлять эксперта показывать вам некоторые промежуточные выводы, перед тем как система окончательно порекомендует вам «удалить голову пациента».

7.2. ПЕРЕМЕННЫЕ, КОТОРЫМИ ВЫ ДО СИХ ПОР ПОЛЬЗОВАЛИСЬ

Теперь, пока вы не задумаетесь, почему это так, вы, наверное, удивитесь огромному набору не связанных между собой утверждений, сваленных в кучу под вывеской экспертной системы.

Действительно, поразительный факт, сколько всего здесь накопилось. Все время вы пишете фрагмент программы, затем добавляете к нему еще фрагмент, модифицируете первый, составляете новый фрагмент, думаете о чем-то еще, выходите, чтобы приготовить кофе, вас осеняет блестящая идея и вы добавляете...

Прежде всего следует навести порядок, отсортировать нужные вещи. Здесь к вашим услугам собрана коллекция основных переменных, которыми вы, возможно, пользовались до сих пор.

DIM VAR$ (MAXVAR, N) — это список имен всех переменных, заложенных в экспертную систему. Для каждого узла существует свой список. Максимальное число переменных в одном узле равно MAXVAR, число узлов при этом равно N.

Когда вы разрабатываете экспертную систему и собираетесь описать массивы, вы должны предусмотреть, чтобы система спросила вас:

INPUT «Введите сколько у вас узлов»; N

и после этого

INPUT «Введите максимальное число переменных, входящих в один узел»; MAXVAR.

Очевидно, вы собираетесь бесполезно потратить весь объем памяти, выделенный под этот массив, ибо он заполнится только тогда, когда (случайно) все узлы будут иметь одинаковое число переменных. И хотя существуют способы, чтобы этого избежать, мы их рассматривать не будем, так как они могут усложнить нам жизнь. Было бы, конечно, разумнее использовать свой массив для переменных каждого узла, но трудно предположить, как вы описали бы их все в БЕЙСИКе, не зная, сколько узлов должна иметь программа на первом уровне. Поэтому полагаем, что в каждом узле не больше MAXVAR переменных.

Массив значений переменных для каждого из N узлов, соответствующий указанным именам переменных, выглядит следующим образом: DIM VALUE (MAXVAR, N).

Массив «флажков» VARFLAG(MAXVAR, N) отмечает, какие переменные используются в процессе принятия решений, а какие нет. Для элементов этого массива устанавливается 0, если переменная уже была введена, и 1, если система лишь ожидает, что она будет введена. Последовательность обработки при этом такова. Сначала вы именуете переменные VAR$ (x, y). Затем система запрашивает значения некоторых переменных VAR$ (x, y), для которых VARFLAG (x, y) = 1, и помещает их в массив VALUE (x, y). После этого она устанавливает значение VARFLAG (x, y) = 0.

Массив RULEVALUE(MAXVAR, N) содержит «оценки» для каждой переменной любого из N узлов. Система использует их, чтобы выяснить, значение какой переменной она должна запросить на следующем шаге. В операторе VAR = (x, y)система спрашивает, есть ли среди них такая переменная, которая имеет наибольшее значение RULEVALUE (x, y).

В массивах MINI (MAXVAR, N), MAXI (MAXVAR, N) хранятся минимальные значения переменных VAR$ (x, y). Минимальное значение VALUE (x, y) определяется значением элемента массива MINI (x, y), а его максимальное значение — значением элемента массива MAXI (x, y). Вы сами должны задать эти значения после имени переменных VAR$ (x, y). Они используются для того, чтобы помочь эксперту оценить относительную важность каждой переменной в процессе выработки решения и вычислить значения RULEVALUE (x, y). Если вы не знаете минимальных и максимальных значений переменной, то ими следует просто задаться: безусловно, должны существовать некоторые предельные значения.

В массиве VAR (N) содержится информация о количестве переменных, связанных с определенным узлом. Система всегда знает максимальное число переменных, связанных с одним из узлов (MAXVAR). Но она должна знать, сколько переменных фактически связано с каждым узлом. Именно в этом массиве эти числа и хранятся. Когда мы рассматривали ситуацию только с одним узлом, например в предыдущих примерах, вместо массива у нас использовалась простая переменная VAR. Для нескольких узлов мы имеем дело вместо нее с массивом VAR (x).

Если у вас уже написана программа для экспертной системы с одним узлом и вы решили преобразовать ее в систему с несколькими узлами, то придется немного попотеть. Продолжая использовать переменную VAR в основной программе, вам нужно будет осуществлять эквивалентную замену VAR=VAR (x) каждый раз, когда будете применять ее при расширении программы, в той части, где используется узел *x*.

В массиве OUTCOMES (N) хранится количество возможных исходов, которые соответствуют каждому узлу. Комментарий, сопутствующий этому массиву, в большей степени такой же, как и комментарий по поводу VAR (N). Вы можете продолжать использовать старую переменную OUTCOMES для допустимых исходов. Нужно только осуществить эквивалентную замену OUTCOMES= =OUTCOMES (x) при работе с узлом x.

В массиве OUTCOMES$ (MAXOUT, N) содержится список имен всех возможных исходов экспертной системы по аналогии с тем, как в VAR(MAXVAR, N) содержится список всех имен переменных. MAXOUT является максимальным числом возможных исходов в одном из узлов, и следует в процессе установки системы ввести в программу оператор: INPUT «Введите максимальное число возможных исходов в узлах»; MAXOUT. Здесь целесообразно использовать во многом похожие комментарии, как и в случае VAR$ (MAXVAR, N).

Массив RULES(MAXVAR, MAXOUT, N) можно назвать ядром системы или массивом правил, на основе которых экспертная система формирует свои суждения. Существует *N* наборов правил, т. е. по одному набору для каждого узла. Каждый набор правил имеет VAR(x) переменных, входящих в узел, и OUTCOMES (x) исходов, выходящих из него. Массив RULES(MAXVAR, MAXOUT, N) устанавливается путем предъявления системе полного набора примеров и предоставления ей возможности проработать их и обучиться на них.

В массиве DECISION(MAXOUT) содержатся значения, сформированные для каждого правила на каждый возможный исход любого из узлов. При наличии *N* узлов вы можете описать массив DECISION(MAXOUT, N) так, чтобы под руками было все необходимое. В некоторой степени это зависит от того, какой метод последовательного анализа вы выбрали для обработки всех узлов. Если вы построили свою систему для одновременной обработки только одного узла, то нет необходимости описывать двухмерный массив, так как в любом случае можно вычислить DECISION (y) и корректировать его значение каждый раз, когда система переходит к новому узлу.

В массиве POSSIBLE (MAXOUT) хранятся возможные значения каждого исхода. Допустим, для данного узла введены только некоторые переменные. Тогда, используя соответствующие значения VALUE (x, y) и RULES (x, y, узел), можно вычислить для этих переменных DECISION (y), причем DECISION (y) = POSSIBLE (y). Но для переменных, которые в данный момент не известны, значение POSSIBLE (y) может быть угадано системой исходя из минимальных и максимальных значений, размещенных в MINI (x, узел) и MAXI (x, узел).

Идея состоит в том, что наилучшее предсказание будет сделано системой на основе максимального значения, хранящегося на данный момент в массиве DECISION (y). Затем вычисляется POSSIBLE (y) в предположении, что все оставшиеся переменные примут свои минимальные или максимальные значения в попытке опровергнуть наилучшее на данный момент предсказание. Если же система не сможет опровергнуть таким способом текущий выбор, то возможный исход выбирается для текущего узла. При опровержении текущего выбора путем нахождения другого исхода, имеющего максимум в POSSIBLE (y), который превосходит текущий максимум в DECISION (y), очевидно, потребуются значения других переменных.

В массиве EXAMPLES (MAXVAR+1,50, N) содержатся любые примеры, которые можно предъявить эксперту для обучения. Резервируется память для хранения до 50 примеров на каждый из N узлов. Система должна знать, сколько примеров у вас есть для каждого узла, поэтому вы можете хранить это число в массиве EXS (N), обновляя его каждый раз, когда захотите изучить новые примеры. Если процесс обучения проводится с клавишного пульта, то вы можете поместить любой пример, предъявляемый системе, в массив EXAMPLES, чтобы она могла потренироваться на этом примере позже. Это убережет ваши пальцы от лишней работы по вводу примеров с клавиатуры. Очевидно, вы будете использовать не 50, а какое-то другое число примеров. Делайте, как вам удобнее.

Допустим, вы построили экспертную систему, состоящую из одного узла, и хотите преобразовать ее в систему с несколькими узлами. Сначала придумайте какой-либо пример и запустите программу с одним узлом. Полученные результаты можно использовать в качестве базовых для оценки дальнейшего поведения системы. Затем просмотрите список переменных и преобразуйте программу для работы системы с несколькими узлами. Введите в программу дополнительно вопросы о значении параметров N, MAXVAR и MAXOUT. Организуйте еще один цикл, охватывающий основную программу, для перебора узлов:

```
FOR NODE=I TO N
.
.
.
основная программа
.
.
.
NEXT NODE
```

Снова запустите систему с примером для N=1. Результаты должны быть такие же, как и прежде. Заметного прогресса не будет. Однако результаты изменятся, если программа допустит ошибку и выдаст об этом соответствующий сигнал. Скорректируйте ошибку. Вы, вероятно, забыли преобразовать некоторые переменные, например VAR или OUTCOMES, в массивы VAR (x) или OUTCOMES (y). Процесс исправления подобных ошибок называется *отладкой* программы.

Если вы отладили программу для одного узла, а затем запустили ее с двумя узлами, причем воспользовались предыдущим примером для проверки второго узла, то, как и раньше, ничего не должно измениться. Вы будете удивлены и изумлены тем, что вроде бы многое должно изменниться, но на самом деле ничего не изменилось. Этот факт довольно поучителен.

Запустим программу снова, но с двумя узлами. Вы должны заметить, что ничего не меняется. В любом случае продолжайте корректировать программу до тех пор, пока ничего не будет меняться. Теперь наступил момент, когда вы можете начать «связывать» узлы вместе. Определив какой-либо возможный исход для узла NODE, запишем

```
10 FOR I=1 TO N
20 FOR J=1 TO VAR (I)
```

```
30 IF OUTCOMES$ (OUTCOME, NODE) =
   =VAR$ (J,I) THEN VALUE (J,I)=1:VARFLAG
   (J, I)=0
40 FOR K=1 TO OUTCOMES (NODE)
50 IF VAR$ (J, I)=OUTCOMES$ (K, NODE)
   AND K<>OUTCOME THEN VARFLAG (J, I)=
   =0
60 NEXT
70 NEXT:NEXT
```

В этом фрагменте программы берется текущий выход (исход) из узла NODE и проводится просмотр списка имен переменных, чтобы выяснить, есть ли в нем аналогичные переменные, являющиеся входом любого другого узла. Если есть, то значение этой переменной устанавливается равным 1, а «флажок» для этой переменной VARFLAG (J, I) устанавливается равным 0. Значения 1/0 хорошо согласуются с состояниями типа Да/Нет. Можно использовать и другие значения, более удобные с вашей точки зрения, например максимальные значения MAXI (J, I).

Программа просматривает и все остальные возможные исходы для того же узла NODE, которые не осуществлялись. Учитывая, что все другие исходы взаимоисключаются, и выбрав один исход, система не может выбрать другие исходы. Поэтому ее значения, соответствующие элементам массивы VARFLAG (J, I) для этих переменных, должны быть установлены равными 0, чтобы экспертная система не запрашивала их значений.

Итак, когда вы ввели в компьютер значение переменной, например VALUE (BESTVAR, NODE), можно осуществить следующие операции:

```
80 FOR I=1 TO N
90 FOR J=1 TO VAR (I)
100 IF VAR$ (BESTVAR, NODE)=VAR$ (J, I)
    THEN VALUE (J, I)=VALUE (BESTVAR, NO-
    DE):VARFLAG (J, I)=VARFLAG (BESTVAR,
    NODE)
110 NEXT:NEXT
```

Если любая заданная переменная оказалась на входе более чем одного узла, то ее значение нельзя запрашивать больше 1 раза.

Идея в целом состоит в том, чтобы «выжать» из информации, которую вы подаете на вход, максимум возможного. Если вы задали значение переменной, то оно должно «работать» путем подачи ее на столько входов, на сколько это возможно. Если выбран какой-то исход, то его следует подать на входы тех узлов, которые в нем нуждаются, и сделать это как можно быстрее в надежде определить дальнейшие возможные исходы с меньшим вашим вмешательством.

Если вы снова установите экспертную систему с двумя узлами и сделаете оба узла одинаковыми, так что оба они будут содержать ваш предыдущий пример, то вы убедитесь, что, сформировав решение на выходе первого узла, экспертная система немедленно сформирует решение и на выходе второго узла. Ответ будет тем же самым, но без всякой помощи с вашей стороны — не стоит удивляться, система получила всю информацию для второго узла в процессе решения задачи для первого узла, поэтому ей не требуется больше никакая другая информация.

Итак, вы имеете базовую систему с несколькими узлами, которая адекватна для решения ряда проблем. Давайте установим систему с двумя узлами и введем один пример.

Узел 1. Переменные: *Оперение, Металл, Клюв.* Исходы: *Животное, Машина.*

Узел 2. Переменные: *Животное, Машина, Клюв, Двигатель.* Исходы: *Птица, Самолет, Планер.*

Точные детали этого примера не играют особой роли. Заметим, что сначала нужно решить, какой возможный исход избрать: *Животное* или *Машину.* Затем система должна использовать эту информацию, чтобы решить, какой допустимый исход выбрать: *Птицу, Самолет* или *Планер.* Ей нет необходимости спрашивать о *Клюве* дважды (эта информация уже была дана на вход узла 1) и не нужно задавать вопросы о *Животном* или *Машине* (она должна была сформировать это для своих нужд, решая задачу на уровне узла 1).

7.3. ПРОХОЖДЕНИЕ ПО УЗЛАМ

Допустим, вы решили сформировать экспертную систему с более чем одним узлом. Тогда эксперт будет иметь несколько выходов, на которых он в состоянии сформировать определенные суждения. В зависимости от этих суждений он сможет затем построить большее число суждений. Проблема, с которой вы столкнетесь в процессе построения подобной системы, заключается в том, что программа должна «уметь» переходить от одного узла к другому. Вы должны обучить эксперта методу, который определял бы, какой по порядку узел следует обрабатывать на следующем шаге, причем от выбора этого метода будет зависеть, хорошо или плохо работает система в целом.

Положение усложняется тем, что не существует какого-то одного идеального метода для любой ситуации. В зависимости от области применения вашей экспертной системы один метод может быть лучше, другой хуже. Это такая же точно проблема, как поиск неизвестной по дереву или на графе. Для того чтобы углубиться в детали описания методов, потребовалось бы обратиться к специальным книгам. К счастью, процедуры и методы, использованные нами до сих пор, значительно сузили круг рассматриваемых проблем, давая возможность описать их, используя ограниченные средства.

Для начала вы могли бы выбирать узлы для просмотра точно так же, как мы выбирали переменные в случае одного узла. Напомню, что у нас был массив RULEVALUE (VAR) со значениями для каждой из VAR переменных. Эти значения вычислялись как максимальное число вариаций, которые каждая переменная могла внести в данную задачу. Возьмем минимальные и максимальные значения каждой переменной, вычислим разность между ними, просмотрим затем массив правил и установим в нем значение, соответствующее этой переменной. Перемножим их для каждого возможного исхода и получим некую «меру важности» каждой переменной Выделим наиболее «важную» переменную, и пусть эксперт для начала выдаст запрос на значение этой переменной. Если эксперт не в состоянии сделать какое-то суждение, то продолжим процесс, запросим следующую по степени важности переменную и т. д.

Не существует каких-либо причин, почему мы не должны использовать этот же подход при решении задачи о последовательности прохождения узлов. Будут, конечно, определенные различия при программировании, так как некоторые переменные могут встречаться более чем в одном узле, но в общем случае этот подход также позволит эксперту разделять их по степени важности и заставит его выбирать прежде всего наиболее важную переменную. Другими словами, подобный подход кажется вполне приемлемым. Он сводится просто к выбору тех переменных, которые дают больше всего информации и имеют максимальное число узлов.

Фактически, поступая таким образом, мы пытаемся описать методику поиска объекта по дереву. Обычно при описании этого метода ссылаются на структуру дерева. Вам, очевидно, известна суть этого поиска: спуск по одной из ветвей, пока не достигнута тупиковая вершина, затем спуск по следующей ветви и т. д.

В этом случае мы, однако, ищем не так. У нас есть метки на каждом пути, соответствующие именам переменных, которые мы ввели. Мы лишь сообщаем системе: найди более зна́чимую переменную, а затем исследуй все пути, содержащие метку этой переменной.

В заключение следует сказать несколько слов. В общем случае мы не знаем точно, какая проблема встанет перед нами в следующую минуту, поэтому и не можем сразу же определить самый подходящий способ для ее решения. Мы знаем, однако, что система должна использовать переменные, соответствующие данной задаче, не учитываем структуру дерева (или сети узлов) и концентрируем все внимание на этих переменных. Описанный метод назовем «движение по цепочке вперед».

Он позволяет получить ответ, даже если возникнет необходимость ввести каждую переменную до того, как вы получите специфический ответ, который вас устраивает. Некоторые специалисты считают это узким местом метода, поскольку последний функционирует без учета структуры сети узлов, через которую эксперт проходит, или конкретных целей, которые вы, возможно, желаете достичь. Можно сказать, что это мало похоже на метод, скорее похоже на липкую грязь, налипающую при движении через сеть узлов, если руководствоваться правилом: «Этот узел выглядит достаточно интересно, посмотрим на следующий!»

Но если система игнорирует конкретный путь, организованный структурой узлов, она оставляет на ваше усмотрение возможность игнорировать структуру, которую они составляют. Вам остается только определить ваши узлы, а затем имена переменных и возможных исходов для каждого из них. Этот подход также имеет ряд недостатков.

Наиболее существенный из них состоит в том, что связь между узлами осуществляется с помощью назначения имен соответствующим переменным и исходам для каждого узла. Это значит, что если вы случайно забыли, как пишется некоторое имя переменной, то вам не удастся сделать требуемую связь. Похожая ситуация: вы нажимаете не ту клавишу на своем компьютере и получаете неверный ответ. Если число переменных велико и вы пытаетесь подобрать имя, тратите на это целый день, то это уже проблема. Не существует общих рецептов, которые могли бы помочь в принципе в подобных случаях, но на практике хорошим подспорьем является формирование и отображение на экране списка переменных, из которого вы можете выбрать нужное вам имя, чтобы каждый раз не вспоминать его, делая ошибки.

Особенность данного метода в том, что эксперт может выявить самую «важную» для себя переменную и запросить ее значение, а в результате в этом нет необходимости хотя бы потому, что у вас в данный момент нет значений переменной, либо вы поняли бесполезность информации о ней для конкретной задачи, либо, наконец, вы хотели бы, чтобы эксперт спросил вас о чем-то другом.

Это препятствие можно легко преодолеть путем использования нескольких узлов. Допустим, что у вас есть N узлов. Вы можете иметь доступ к ним, поместив последние внутрь цикла, т. е. система будет последовательно рассматривать узлы 1, 2 и т. д. до N. Для каждого узла система с помощью предыдущего метода решает, какую переменную ей нужно запрашивать в первую очередь. Затем, найдя некий возможный исход, она проверяет все другие узлы, чтобы понять, является ли данный исход входной переменной какого-то другого узла. При этом точный порядок прохождения от одного узла к другому оказывается жестко заданным в момент формирования системы. В этом конкретном случае, если вы знаете задачу, в решении которой хотели бы натренировать эксперта, можно при формировании системы конкретно указать, что необходимо сделать в первую очередь. Если же вы не знаете требуемого порядка или не хотите думать об этом, то вам нечего терять.

Грубо говоря, проблема в том, с какого места эксперт должен начать работу над порученной ему задачей, и хуже всего, если вы сами не знаете, с чего начать. Все это не так просто. Обратите внимание: система могла бы заняться поиском наиболее важных переменных и запросить их значение, но то, что для нее кажется важным, вовсе не важно для вас. Например, вы могли бы определить некоторую начальную точку, указав системе траекторию ее регулярного прохождения через различные узлы. Но предположим, что в одной ситуации вы имеете один набор переменных, а в другой — другой. В общем случае, возможно, вам не всегда известны значения всех переменных и в зависимости от того, что вы фактически имеете, эксперт должен начинать вычисления с различных точек и проходить в силу этого по различным узлам.

В идеальном случае вы обязаны иметь возможность сообщать эксперту, какой информацией располагаете, и делать из этого выводы. Вы можете передать эксперту имя переменной и ее значение. Это имя впоследствии определяет, какой узел или узлы должны рассматриваться системой прежде всего. И, учитывая опасность неверного введения имени переменной, либо следует выбирать его из списка имен переменных, либо проверять, что ваш ввод соответствует некой существующей переменной до того, как система начнет с ней работать.

Наряду с проблемой, где эксперту начать работу, существует аналогичная проблема, где эксперту ее закончить.

Было бы очень хорошо, если бы эксперт сам определил все, что он должен сделать. В этом случае вы всегда получаете ответ на поставленную вами задачу. На практике же вам не хочется сидеть и вводить данные для тех узлов, которые для решения основной проблемы могут оказаться достаточно случайными. Метод прохождения узлов в порядке возрастания их номера позволяет управлять этим процессом, учитывая, что узел N мог бы содержать окончательное, общее решение. Цель тогда состоит в том, чтобы достичь узла N как можно быстрее. В общем случае логично идти к узлу, ближайшему к узлу N. Вопрос в том, какой узел считать ближайшим.

Самый простой вариант — рассматривать каждый узел либо как активный, либо как «мертвый». Если на его выходе уже сформирован возможный исход, то он «мертв» — нет необходимости рассматривать его снова. Если же на его выходе никакой возможный исход не сформирован, то он активен и вы должны найти для него этот исход. Можно перебирать узлы по порядку, находя решения для активных узлов и выбирая последовательно ближайшие к нему узлы. В какой-то точке легче всего наблюдать, что узел M является ближайшим к узлу N. Почему бы не обработать этот узел первым? Это позволило бы сэкономить много времени.

Хорошо, коль вы можете это сделать, вам нет необходимости иметь на первом месте другие узлы. В противном случае ищите, с чего вам начать, чтобы закончить в нужном месге. Именно здесь вы должны осознать, что поиск узлов не так прост, чтобы полностью разобраться в нем.

Наиболее общий метод возвращает нас немного назад: необходимо сообщить о том, значения каких переменных у нас есть, ввести их и выяснить, какие возможные исходы из этого следуют.

Если вы не хотите вводить больше информации, чем необходимо, и представляете себе, какие промежуточные результаты хотелось бы увидеть, картина усложняется. Допустим, например, вы просто хотите определить наикратчайший путь от некоторой начальной точки до конечного решения. Значит ли это, что для кратчайшего пути следует «посетить» минимальное число узлов? Или же должно быть задано минимальное число вопросов? Вы можете отвечать на эти вопросы по своему усмотрению, но ясно, что на них нет однозначного ответа.

Задавая минимальное число вопросов, вы проделываете минимальную возможную работу. Посещая минимальное число узлов, вы получаете минимальное количество промежуточных результатов. По-видимому, вы хотели бы получить столько промежуточных результатов, сколько возможно при минимальных усилиях. Итак, вы желали бы посетить максимально возможное число узлов при условии, что система задала бы вам минимально возможное число вопросов? Может оказаться, что две эти цели одновременно недостижимы.

Рискуя увеличить затраты на программирование, можно следить за расстоянием до узла N в терминах узлов и в терминах неиспользованных переменных для каждого узла сети. Однако рост затрат на программирование может быть значительным. Особенно когда существует более одного пути до узла N из некоторых узлов сети. Вы должны будете выяснить не только расстояние по одному пути, но и сколько всего путей существует и какова их длина до узла N. Кроме того, необходимо следить за всеми этими путями, чтобы выбрать самый лучший из них.

Это, конечно, не такая уж неразрешимая проблема. Если вы обратитесь к литературе по графам и сетям, то найдете много способов получить решение. Но это не так тривиально. В общем случае затраты на программирование можно существенно уменьшить, впрочем, как и затраты вашего интеллекта, если вы воспользуетесь одним из следующих методов.

1. Предложите всю доступную информацию системе. Укажите, для каких переменных вы имеете информацию, и дайте эксперту возможность работать на всех узлах, в которых эта информация используется, чтобы увидеть, какие выводы он сможет сделать.

2. Дайте системе возможность запросить информацию о тех переменных, которые кажутся наиболее «важными». Полагая, что вы имеете всю необходимую информацию, она, вероятно, потребует предоставить ей все, что она запрашивает. Позвольте системе сделать все выводы, которые она сможет, пока не получит нужного вам ответа, либо предоставьте ей информацию по всем переменным.

3. Дайте системе возможность последовательно переходить от узла к узлу. На этом пути она ничего не пропустит и со временем дойдет до конца. Когда вы впервые сформируете систему, постарайтесь установить узлы в таком порядке, который имеет определенный смысл для вас. Вы могли бы разрешить системе сделать все заключения, какие она может и когда может, используя данный метод, ибо может так случиться, что она придет к финалу быстрее, чем вы ожидали. И если система придет к какому-то узлу, для которого уже получен возможный исход, она может не рассматривать его либо проделать уже сделанную работу дважды.

Имеет смысл напомнить, что вы могли бы делать то же самое и с системой, состоящей из одного узла, или с системой, которая выглядит как один узел. Если, скажем, вы имеете только RULES (I, J) для представления одного узла, то могли бы иметь набор возможных исходов, причем некоторые из них совпали бы с другими переменными.

Если эксперт решил остановиться на некотором возможном исходе и это случайно совпало с какой-то входной переменной, то этот исход мог бы использоваться для того, чтобы включить (задействовать) эту входную переменную, после чего «флажок» должен быть установлеи на этом исходе, чтобы не учесть его действие больше 1 раза.

Вы могли бы продолжить решение путем описания тех имен переменных, для которых намерены получить информацию, либо позволить системе выбрать самой эти переменные, либо в свою очередь разрешить ей просматривать все переменные.

Сложность такого подхода связана с наличием отдельных узлов, фактически заключенных в один большой узел, что дает определенный простор для ошибочных заключений, предлагаемых системой. Предположим, например, система задумана для предсказания погоды и для начала она предсказала, что сегодня будет теплый день. Что должно остановить ее — комментарий, что день сегодня был холодным? Кроме всего прочего для заданного набора правил эти два наиболее вероятных исхода могут быть взаимоисключающими, но система иногда выдает оба эти исхода перед тем, как предложит любые другие.

Конечно, можно предположить какой-то метод для разрешения этого противоречия. Стремление избежать ошибочных заключений — один из лучших способов получения работоспособных программ. Будучи исходно не таким идеальным методом, он быстро станет лучшим во всех отношениях, если окажется единственным надежно работающим методом.

7.4. ТЩАТЕЛЬНО ПРОДУМАННЫЕ УЗЛЫ

Устанавливая вашу экспертную систему для решения конкретных задач, вы, возможно, столкнетесь с таким узлом, который покажется вам либо слишком сложным, либо слишком простым, чтобы у вас возникло желание обучить его на конкретных примерах.

Допустим, вы решили установить прекрасную систему для медицинской диагностики. Одной из ее входных переменных является *Температура* пациента. Вы должны заставить систему сказать, действительно ли у пациента высокая температура или нет. Это легко сделать. Если она выше, скажем, 37,2 °C, то на выходе некоторого узла следует иметь в качестве исхода — *Исход 1*, полагая, что он соответствует высокой температуре.

Ясно, что если вы дадите эксперту достаточно примеров для обучения, то система в конце концов научится формировать свое собственное суждение. Но для этого требуется слишком много рутинной работы. Причем вы знаете, какими должны быть правила. Какой смысл тогда во всей этой затее с сеансами обучения?

Спокойно. Отбросьте мысль об обучении и установите тщательно продуманный узел для оценки *Температуры* пациента. Задача состоит в том, чтобы найти значения массива RULES (I, J), т. е. получить правильный ответ. Опишем этот узел:

	Высокая температура	*Низкая температура*
Константа	0	37,2
Температура	1	0

Узел характеризуется массивом RULES (2,2). Допустим теперь, что у вас есть вход VALUE (I). Элемент VALUE(0) содержит 1, а элемент VALUE(1) — *Температуру пациента.* Умножив вход на первое правило (*Высокая температура*), мы всегда будем иметь в результате значение, равное VALUE (I). Умножив вход на второе правило (*Низкая температура*), мы получим 37,2. Очевидно, первое правило дает значение более высокое, чем второе, только тогда. когда VALUE (I) — *Температура* пациента — выше 37,2 °С. Второе же правило даст более высокое значение, чем первое, в любой момент, если *Температура пациента* ниже 37,2°С, поэтому система всегда будет выносить правильный ответ при сравнении *Высокой* и *Низкой температур.* Это оказывается значительно проще, чем задавать системе полный набор примеров и ждать, пока она отсортирует свои ответы.

В общем случае решающее правило всегда зависит от значения DECISION$=b_1x_1+b_2x_2+...+b_nx_n$. Поэтому если вы сможете облечь решение в такую форму, то это окажется самым простым вариантом.

Если вы задумаетесь о том, как представить проблему в подобном виде, то всегда сможете задать системе несколько примеров и посмотреть, сможет ли она что-то решить для вас. Если ваша задача достаточно сложна для системы, то могут возникнуть определенные трудности. Комментарии в этом случае излишни, так как не существует в общем случае метода, позволяющего облечь в эту форму все, что вам бы хотелось.

Допустим, входная переменная может изменяться от

0 до 100. Если она принимает значения в диапазоне от 50 до 60, это может иметь особый смысл для вас и вам хотелось бы, чтобы экспертная система обнаруживала эту ситуацию. В подобной ситуации нет ничего необычного, а 50 и 60 могут означать, например, границы оптимального диапазона изменения этой переменной. В данный момент нельзя со всей очевидностью сказать, как должен быть записан набор правил, позволяющих выявлять заданный диапазон значений. Но вы могли бы исследовать эту проблему в двух направлениях. Во-первых, можно было бы использовать для этого два узла. Первый узел мог бы определять, превысила ли входная величина значение 50, — точно так же, как это было выше в случае с регистрацией *Высокой температуры*. Если превышение произошло, то система должна передать выход первого узла на вход второго узла, который определяет, меньше ли его значение, чем 60, или нет. Если система сможет получить правильные результаты от обоих узлов, то она в состоянии констатировать, что эта переменная лежит в диапазоне от 50 до 60. Таким образом, ваша проблема решена. Следовательно, вполне возможно определить принадлежность переменной к заданному диапазону значений, используя этот подход.

С другой стороны, кажется, что было бы значительно проще написать фрагмент программы, который решал бы те же проблемы,

```
IF VALUE (I)>=50 AND VALUE (I)<=60 THEN
OUTCOMES (J)=1:...
```

и т. д., чем втискивать проблему в несвойственную ей форму.

Но раз уж вы начали делать такие вещи, то вам придется иметь дело и с оставшимися проблемами, например, с такой: как связать столь разные фрагменты в одну общую программу? Лучше всего поместить такие фрагменты программ в общую программу на самой ранней стадии, так чтобы программа начинала работать каждый раз с новым узлом. Это значит, что данные фрагменты программы выполнялись бы чаще, чем это необходимо, но была бы гарантия, что эксперт вовсе не позабудет их выполнить.

Возвращаясь к проблеме медицинской диагностики, допустим, что мы хотим знать, была ли у пациента простуда (озноб). Мы опишем озноб как состояние, характеризующееся высокой температурой и румянцем на щеках (либо другим набором симптомов, которые вы пожелали бы выбрать). Экспертная система уже знает температуру пациента, поэтому вам осталось выяснить, румяны ли его щеки. Имея достаточно примеров, эксперт выработал бы для себя набор правил. Хотя с таким же успехом вы могли бы ввести свои собственные правила. Одним из примеров таких логических связок может служить следующий:

IF *(Высокая температура)* AND *(Румяные щеки)*...

Допустим, у нас как у медиков другая точка зрения, мы вправе предположить, что достаточно иметь только один из этих симптомов, тогда

IF *(Высокая температура)* OR *(Румяные щеки)*...

Логические связки (операции) AND и OR используются очень широко, поэтому было бы полезно познакомиться с некоторыми правилами их применения:

	Исход 1	*Исход 2*
Постоянная	0	$n-1/2$
Переменная 1	1	0
Переменная 2	1	0
.	.	.
.	.	.
.	.	.
Переменная n	1	0

Исход 1 дает максимальное значение n, если все n переменных равны 1. *Исход 2* дает одинаковое значение ($n-1/2$) вне зависимости от того, какие значения переменных он получил. Отсюда ясно, что эксперт всегда выберет *Исход 2*, за исключением того случая, когда все n переменных равны 1. Другими словами, *Исход* 1 будет выбран, если мы имеем

(Переменная 1) AND *(Переменная 2)* AND *(Переменная 3)*... AND... *(Переменная n)*.

Используя следующий набор правил, *Исход 2* всегда равен 1/2. *Исход 1* — его значение 0 — будет меньше этого значения до тех пор, пока значения всех переменных будут равны 0. Как только одна из переменных примет значение 1, *Исход 1* примет максимальное значение. Поэтому эксперт выбирает *Исход 1*, когда мы имеем

(Переменная 1) OR *(Переменная 2)* OR *(Переменная 3)*... OR... *(Переменная n)*.

	Исход 1	*Исход 2*
Постоянная	0	1/2
Переменная 1	1	0
Переменная 2	1	0
.	.	.
.	.	.
.	.	.
Переменная n	1	0

Очевидно, что оба эти набора правил очень похожи. Фактически, если для *Исхода 2 Постоянная* имеет значение x—1/2, эксперт будет рассчитывать на *Исход 1*, как только x или бо́льшая часть входных переменных станет равна 1, а не 0. Для связки AND $x=n$, а для связки OR $x=1$, хотя вы можете иметь любое другое число, если это удобно для вашего случая.

Преимущество установки собственных правил заключается в том, что вы будете точно знать о намерениях системы в любой момент в любой ситуации. Преимущества же ситуации, когда вы позволяете системе самой выработать с помощью примеров собственные правила, в том, что это обеспечивает относительную легкость ввода новых правил, а также снимает с вас необходимость выработки самих правил.

7.5. О ВОЗМОЖНОСТИ НЕПОСРЕДСТВЕННОГО ПРОГРАММИРОВАНИЯ

Возвращаясь к системе медицинской диагностики, задумаемся над примером, обсужденным нами выше. Этот пример с простудой. Допустим, в нашем представлении симптомами простуды являются высокая температура и румянец на щеках. Мы, как смогли (явно или с помощью обучения на примерах), внесли эти симптомы в базу знаний в форме правил, которые позволили бы распознать простуду.

Но, видимо, было бы проще написать одну строку программы на БЕЙСИКе:

IF TEMPERATURE>37,2 AND FEVER = 1 THEN PRINT «Простуда». Здесь FEVER=1, если пациент имеет румянец на щеках. Да, конечно, это было бы проще. Думаю, вы едва ли ошибетесь. Фактически можно было бы иметь целый набор подобных утверждений, которые составляли бы некую систему для диагностики каких-то вещей (эти вещи вы учли бы при программировании).

Это было бы легче и проще сделать, используя такой язык программирования, как БЕЙСИК.

Если бы вы имели некоторую практику программирования, то вполне могли бы сесть и написать программу так же быстро, как и сформировать систему правил для экспертной системы общего назначениая. Во всем этом есть доля случайности до тех пор, пока вы не сформировали нужную вам программу. Однако и при использовании систем общего назначения должна присутствовать определенная доля случайности, чтобы быть уверенным, что она «покроет» все возможные варианты применения, которые могут возникнуть.

Единственное, что остается, — это обратиться к теоретическим требованиям, иногда предъявляемым к экспертным системам. Говорят, что в идеальном случае они являются программами общего назначения, т. е. содержат метод, позволяющий делать экспертное суждение в общем виде. Предполагается также, что к экспертной системе вы добавляете базу знаний, получая таким образом экспертов различного рода для каждой новой базы знаний. Здесь следует отметить, что есть нечто такое, что вы не можете делать, если просто сядете и напишите программные фрагменты для решения какой-то конкретной задачи.

Так, если вы напишите программу решения задачи медицинской диагностики, то она будет непригодна для оперативной настройки на решение других задач. Все это не более чем учтивая критика. Ведь большинство экспертных систем фактически работают только на конкретных типах задач, т. е. фактически их нельзя назвать системами общего назначения. Та система, которую вы строите, возможно, является системой общего назначения, но других таких систем не так уж много. И, как мы убедились, существуют также свои ограничения с точки зрения затрат времени.

7.6. ХРАНЕНИЕ ВАШЕЙ ЭКСПЕРТНОЙ СИСТЕМЫ

День заканчивается, вы сидите за компьютером, тренируя свою экспертную систему, добиваясь от нее совершенства, пора идти спать. Вы выключаете компьютер и вот — экспертиза исчезает как туман! Вы знаете, что это должно было случиться, ведь это были только данные. Но мало пользы в таком эксперте, который необходимо строить заново каждый раз, когда вы хотите воспользоваться им. Вам хотелось бы, чтобы он работал с каждым днем лучше, а не хуже.

Очевидно, сама программа может быть сохранена с помощью оператора SAVE на диске, но как именно вы сохраните данные, будет зависеть от того, как захотите их использовать. Вы можете, например, считать, что лучше применять технику обработки файлов, а не массивов, чтобы все можно было взять с диска, а не хранить в памяти компьютера. Однако самый простой и быстрый способ сохранить всю информацию — это написать программу загрузки всех массивов и связанных с ними переменных на диск и их считывания с диска. В результате вы получите относительно простой и быстрый способ обработки массивов и не потеряете, например, результаты длинной обучающей сессии. Если вы поступите так, то, вероятно, почувствуете, что вам на ум продолжают приходить идеи о том, как заставить вашу экспертную систему работать лучше.

Например, вы могли бы иметь данные для различных экспертных систем (фактически же их баз знаний, если вы помните этот жаргон), хранящиеся в различных файлах, предлагая пользователю выбрать, какую из них загрузить, когда он начнет работать с системой. И когда вы создаете новую базу знаний, вы пишите программу выбора варианта (опции), хранящегося под новым именем файла. Действуя подобным образом, вы постепенно закончили бы построение огромной коллекции экспертных систем, доступной для вас в считанные секунды.

Единственная опасность, подстерегающая вас на этом пути, — это возможность закончить процесс построения экспертных систем чем-то полезным, которым действительно можно воспользоваться в перспективе. И если это случится, то вы не будете знать, когда же вы, наконец, закончите этот процесс.

7.7. ПРОГРАММА, ИСПОЛЬЗУЮЩАЯ МНОГО УЗЛОВ

Чтобы сохранить это «неудобство», связанное с необходимостью вечно о чем-то думать, мы предлагаем полную, основанную на использовании «меню» и многих узлов машинно-обучаемую экспертную систему (рис. 7.3), позволяющую кому-то уже после нескольких секунд практики достичь некоторых приемлемых результатов,

```
10 MINI=-10000: FALSE%=0: TRUE%=NOT FALSE%
15 MSG1$=" в качестве возможного исхода": MSG2$=" Могу я вывести возможный
   исход: ": MSG3$=" Входная переменная вышла за границы допуска. "
20 CLS
30 GOSUB 2750
40 PRINT "   1. Инициализация системы"
50 PRINT "   2. Ввод примеров"
60 PRINT "   3. Тренировка системы"
70 PRINT "   4. Обучение системы"
80 PRINT "   5. Нормальное функционирование системы"
90 PRINT "   6. Запоминание текущего состояния системы"
100 PRINT "   7. Загрузка экспертной системы"
110 PRINT "   8. Контроль правил и примеров"
120 PRINT "   9. Выход"
130 PRINT: PRINT: INPUT "       Выберите нужный вариант ";OP: CLS
140 ON OP GOTO 160,1390,1580,490,490,1770,2010,2260,150: GOTO 130
150 END
160 REM Инициализация системы
170 GOSUB 2750
180 PRINT "Для установки экспертной системы"
190 PRINT "ответьте на следующие вопросы:"
200 PRINT: INPUT "Сколько узлов имеет система ";N
210 PRINT: INPUT "Каково максимальное число переменных на входе любого из
    узлов"; MAXVAR
220 PRINT: INPUT "Каково максимальное число исходов на выходе любого из
    узлов": MAXOUT
230 GOSUB 2810
240 FOR NODE=1 TO N
250 GOSUB 2750
260 PRINT "Узел";NODE
270 INPUT "Сколько переменных на входе этого узла ";VAR(NODE)
280 PRINT: PRINT "Назовите эти переменные:"
290 FOR I=1 TO VAR(NODE)
300 PRINT "Переменная ";I;: INPUT VAR$(I,NODE)
310 X$="N"
320 FOR J=1 TO NODE
330 FOR K=1 TO VAR(J)
340 IF VAR$(I,NODE)=VAR$(K,J) AND NODE<>J THEN PRINT "(";VAR$(I,NODE);
    ") уже использовалась в узле";J: PRINT "с минимальным значением";MINI(K,J);
    " и максимальным значением ";MAXI(K,J): INPUT "Это та же переменная [y/n]";X$
350 IF X$="Y" OR X$="y" THEN
360 NEXT K
370 NEXT J
380 IF X$="N" OR X$="n" THEN
390 NEXT I
400 GOSUB 2750
410 PRINT "Узел";NODE
420 INPUT "Сколько возможных исходов на выходе этого узла";OUTCOMES(NODE)
430 PRINT: PRINT "Назовите эти исходы: "
```

```
440 FOR I=1 TO OUTCOMES(NODE)
450 PRINT "Исход ";I;: INPUT OUTCOMES$(I,NODE)
460 NEXT I
470 NEXT NODE
480 GOTO 20
490 REM Нормальное функционирование/Обучение экспертной системы
500 IF OP=4 THEN TRAINING%=TRUE% ELSE TRAINING%=FALSE%
510 FOR NODE=1 TO N
520 FOR I=1 TO VAR(NODE)
530 VARFLAG(I,NODE)=1
540 VALUE(I,NODE)=0
550 NEXT I
560 FOR J=1 TO OUTCOMES(NODE)
570 OUTFLAG(J,NODE)=1
580 NEXT J
590 NEXT NODE
600 FOR NODE=1 TO N
610 REM Начало работы с новым узлом
620 VAR=VAR(NODE): OUTCOMES=OUTCOMES(NODE)
630 GOSUB 2750
640 PRINT "Узел";NODE: PRINT
650 IF TRAINING% THEN PRINT "Обучение экспертной системы" ELSE
    PRINT "Нормальное функционирование"
660 PRINT: DECISION=MINI: MAXPOSS=MINI: BEST=1: BESPOSS=1
670 REM Вычисление DECISION и BEST:
680 FOR J=1 TO OUTCOMES
690 DECISION(J)=0
700 FOR K=1 TO VAR
710 IF VARFLAG(K,NODE)=0 THEN
720 NEXT K
730 IF DECISION(J)>DECISION THEN BEST=J: DECISION=DECISION(J)
740 NEXT J
750 REM Вычисление POSSIBLE и BESTPOSS:
760 FOR J=1 TO OUTCOMES
770 POSSIBLE(J)=DECISION(J)
780 FOR K=1 TO VAR
790 IF VARFLAG(K,NODE) AND OUTFLAG(J,NODE) THEN IF RULES(K,J,NODE)>RULES(K,BEST,
    NODE) THEN POSSIBLE(J)=POSSIBLE(J)+(RULES(K,J,NODE)-RULES(K,BEST,NODE))*MAXI
    (K,NODE) ELSE POSSIBLE(J)=POSSIBLE(J)-(RULES(K,BEST,NODE)-RULES(K,J,NODE))*
   MINI(K,NODE)
800 NEXT K
810 IF POSSIBLE(J)<POSSIBLE(BEST) THEN OUTFLAG(J,NODE)=0:
820 IF POSSIBLE(J)>=MAXPOSS THEN MAXPOSS=POSSIBLE(J): BESTPOSS=J
830 NEXT J
840 IF BESTPOSS=BEST THEN 1080: REM Неопределенности больше нет
850 USEFUL%=FALSE%
860 RV=0: BESTVAR=0
870 FOR I=1 TO VAR
880 RULEVALUE(I,NODE)=0: MEAN=0: NUMBER=0
890 FOR J=1 TO OUTCOMES
900 MEAN=MEAN+RULES(I,J,NODE)*VARFLAG(I,NODE)*OUTFLAG(J,NODE)
910 NUMBER=NUMBER+OUTFLAG(J,NODE)
```

```
920 NEXT J
930 IF NUMBER>0 THEN MEAN=MEAN/NUMBER
940 FOR J=1 TO OUTCOMES
950 RULEVALUE(I,NODE)=RULEVALUE(I,NODE)+ABS(MINI
    (I,NODE)-MAXI(I,NODE))*((RULES(I,J,NODE)-MEAN)^2)*
    VARFLAG(I,NODE)*OUTFLAG(J,NODE)
960 NEXT J
970 IF RULEVALUE(I,NODE)>RV THEN BESTVAR=1:
    RV=RULEVALUE(I,NODE)
980 NEXT I
990 IF RV=0 THEN 1150: REM НЕТ ЗНАЧЕНИЙ RULEVALUE>0,
    ПОЭТОМУ СПРАШИВАТЬ БОЛЬШЕ НЕЧЕГО
1000 VARFLAG(BESTVAR,NODE)=0
1010 FOR J=1 TO OUTCOMES-1
1020 FOR K=J+1 TO OUTCOMES
1030 IF RULES(BESTVAR,J,NODE)<>RULES(BESTVAR,K,NODE)
     AND OUTFLAG(J,NODE) AND OUTFLAG(K,NODE) THEN
     USEFUL%=TRUE%
1040 NEXT K
1050 NEXT J
1060 IF NOT USEFUL% THEN 860
1070 GOSUB 2560
1080 REM СОПОСТАВЛЕНИЕ ЛЮБОГО ОТЛИЧАЮЩЕГОСЯ
     ЗНАЧЕНИЯ VALUE
1090 FOR NN=1 TO N
1100 FOR I=1 TO VAR(NN)
1110 IF VAR$(I,NN)=VAR$(BESTVAR,NODE) THEN VALUE(I,NN)=
     VALUE(BESTVAR,NODE): VARFLAG(I,NN)=0
1120 NEXT I
1130 NEXT NN
1140 IF BESTPOSS<>BEST THEN 660
1150 IF NOT TRAINING% THEN PRINT "ПРЕДЛАГАЕТСЯ";
     OUNCOMES$(BEST,NODE);MSG1$: CORRECT=BEST: GOTO 1270 ELSE
     PRINT MSG2$; OUTCOMES$(BEST,NODE);" [y/n] ";: INPUT A$:
     IF A$="Y" OR "y" THEN CORRECT-BEST ELSE GOSUB 2610
1160 REM ОПРЕДЕЛЕНИЕ НАИЛУЧШЕГО VALUE
1170 FOR BESTVAR=1 TO VAR
1180 IF VARFLAG(BESTVAR,NODE) THEN GOSUB 2560
1190 NEXT BESTVAR
1200 GOSUB 2680
1210 INPUT "ВЫ ХОТИТЕ СОХРАНИТЬ ЭТОТ ПРИМЕР [y/n]";E$:
     IF E$="N" OR E$="n" THEN 1270
```

```
1220 EXS(NODE)=EXS(NODE)+1
1230 FOR I=1 TO VAR
1240 EXAMPLES(I,EXS(NODE)=VALUE(I,NODE)
1250 NEXT I
1260 EXAMPLES(MAXVAR + 1, EXS(NODE), NODE)=CORRECT
1270 REM СОПОСТАВЛЕНИЕ НАИЛУЧШЕГО OUTCOMES
1280 FOR NN=1 TO N
1290 FOR I=1 TO VAR(NN)
1300 IF VAR$(I,NN)=OUTCOMES$(CORRECT,NODE) THEN
     VALUE(I,NN)=MAXI(I,NN): VARFLAG(I,NN)=0
1310 FOR J=1 TO OUTCOMES
1320 IF VAR$(I,NN)=OUTCOMES(J,NODE) AND J <> CORRECT THEN
     VALUE(I,NN)=MINI(I,NN): VARFLAG(I,NN)=0
1330 NEXT J
1340 NEXT I
1350 NEXT NN
1360 NEXT NODE
1370 PRINT: PRINT "ВЫ ХОТИТЕ ПРОДОЛЖИТЬ ";:
     ELSE INPUT "НОРМАЛЬНОЕ ФУНКЦИОНИРОВАНИЕ [y/n]";B$
1380 IF B$="Y" OR B$="y" THEN 510 ELSE 20
1390 REM ВВОД ПРИМЕРОВ
1400 FOR NODE=1 TO N
1410 GOSUB 2750
1420 PRINT "УЗЕЛ"; NODE: PRINT
1430 PRINT "ВВОД ПРИМЕРОВ:"
1440 PRINT: INPUT "СКОЛЬКО ПРИМЕРОВ ПРИГОТОВЛЕНО ДЛЯ
     УЗЛА"; NUMBER
1450 FOR I=1 TO NUMBER
1460 GOSUB 2750
1470 PRINT "ПРИМЕР"; I;" ДЛЯ УЗЛА"; NODE;: PRINT: PRINT
1480 FOR BESTVAR=1 TO VAR(NODE)
1490 GOSUB 2560
1500 EXAMPLES(BESTVAR, I+EXS(NODE), NODE)=VALUE(BESTVAR,
     NODE)
1510 NEXT BESTVAR
1520 GOSUB 2610
1530 EXAMPLES(MAXVAR+1, I+EXS(NODE), NODE)=CORRECT
1540 NEXT I
1550 EXS(NODE)=EXS(NODE)+NUMBER
1560 NEXT NODE
1570 GOTO 20
1580 REM ТРЕНИРОВКА ЭКСПЕРТНОЙ СИСТЕМЫ
```

```
1590 GOSUB 2750
1600 PRINT "ЭКСПЕРТ ТРЕНИРУЕТСЯ НА ПРИМЕРАХ"
1610 FOR NODE=1 TO N
1620 FOR P=1 TO EXS(NODE)*10
1630 EXAMPLE=INT(RND*EXS(NODE)+1)
1640 CORRECT=EXAMPLES(MAXVAR+1,EXAMPLE,NODE)
1650 VAR=VAR(NODE): OUTCOMES=OUTCOMES(NODE)
1660 FOR I=1 TO OUTCOMES
1670 DECISION(I)=0
1680 FOR J=1 TO VAR
1690 VALUE(J,NODE)=EXAMPLES(J,EXAMPLE,NODE)
1700 DECISION(I)=DECISION(I)+VALUE(J,NODE)*RULES(J, I, NODE)
1710 NEXT J
1720 NEXT I
1730 GOSUB 2680
1740 NEXT P
1750 NEXT NODE
1760 GOTO 20
1770 REM ЗАПОМИНАНИЕ ТЕКУЩЕГО СОСТОЯНИЯ
     ЭКСПЕРТНОЙ СИСТЕМЫ
1780 OPEN "EXPERT.DAT" FOR OUTPUT AS#3
1790 PRINT # 3,N;MAXVAR;MAXOUT
1800 FOR NODE=1 TO N
1810 PRINT # 3,VAR(NODE);OUTCOMES(NODE);EXS(NODE)
1820 FOR I=1 TO MAXVAR
1830 PRINT # 3,CHR$(34);VAR$(I,NODE);CHR$(34);",";MIN
     (I,NODE);MAXI(I,NODE)
1840 NEXT
1850 FOR I=1 TO MAXOUT
1860 PRINT # 3,CHR$(34);OUTCOMES$(I,NODE);CHR$(34)
1870 NEXT
1880 FOR I=1 TO MAXVAR+1
1890 FOR J=1 TO EXS(NODE)
1900 PRINT # 3,EXAMPLES(I,J,NODE)
1910 NEXT J
1920 NEXT I
1930 FOR I=1 TO MAXVAR
1940 FOR J=1 TO MAXOUT
1950 PRINT # 3,RULES(I,J,NODE)
1960 NEXT J
1970 NEXT I
```

```
1980 NEXT NODE
1990 CLOSE # 3
2000 GOTO 20
2010 REM ЗАГРУЗКА ЭКСПЕРТНОЙ СИСТЕМЫ
2020 OPEN "EXPERT. DAT" FOR INPUT AS # 3
2030 INPUT # 3, N, MAXVAR, MAXOUT
2040 GOSUB 2B10
2050 FOR NODE=1 TO N
2060 INPUT # 3, VAR(NODE), OUTCOMES(NODE), EXS(NODE)
2070 FOR I=1 TO MAXVAR
2080 INPUT # 3, VAR$(I, NODE), MINI(I, NODE), MAXI(I, NODE)
2090 NEXT I
2100 FOR I=1 TO MAXOUT
2110 INPUT # 3, OUTCOMES$(I, NODE)
2120 NEXT I
2130 FOR I=1 TO MAXVAR+1
2140 FOR J=1 TO EXS(NODE)
2150 INPUT #3, EXAMPLES(I, J, NODE)
2160 NEXT J
2170 NEXT I
2180 FOR I=1 TO MAXVAR
2190 FOR J=1 TO MAXOUT
2200 INPUT # 3, RULES(I, J, NODE)
2210 NEXT J
2220 NEXT I
2230 NEXT NODE
2240 CLOSE # 3
2250 GOTO 20
2260 REM КОНТРОЛЬ ПРАВИЛ И ПРИМЕРОВ
2270 FOR NODE=1 TO N
2280 PRINT "УЗЕЛ"; NODE: PRINT
2290 PRINT "ТЕКУЩИЕ ПРИМЕРЫ:": PRINT
2300 FOR I=1 TO VAR(NODE)
2310 PRINT VAR$(I, NODE);
```

```
2320 FOR J=1 TO EXS(NODE)
2330 PRINT TAB(20+J*3);EXAMPLES(I,J,NODE);
2340 NEXT J
2350 PRINT
2360 NEXT I
2370 PRINT: PRINT " Исход";
2380 FOR J=1 TO EXS(NODE)
2390 PRINT TAB(20+J*3);EXAMPLES(MAXVAR+1,J,NODE);
2400 NEXT J: PRINT: PRINT
2410 PRINT " Текущие правила:": PRINT
2420 FOR J=1 TO OUTCOMES(NODE)
2430 PRINT TAB(21+J*3);OUTCOMES$(I,NODE);
2440 NEXT J
2450 PRINT
2460 FOR I=1 TO VAR(NODE)
2470 PRINT VAR$(I,NODE);
2480 FOR J=1 TO OUTCOMES(NODE)
2490 PRINT TAB(20+J*3);RULES(I,J,NODE);
2500 NEXT J
2510 PRINT
2520 NEXT I: PRINT
2530 PRINT " Для продолжения нажмите любую клавишу";
     X$="": WHILE X$="": X$=INKEY$: WEND
2540 NEXT NODE
2550 GOTO 20
2560 REM Подпрограмма для определения значения переменной BESTVAR в узле NODE
2570 PRINT " Переменная ";BESTVAR;" (";VAR$(BESTVAR,NODE);") есть <1/0> ";:
     INPUT VALUE(BESTVAR,NODE)
2580 IF VALUE(BESTVAR,NODE)<MINI(BESTVAR,NODE) OR VALUE(BESTVAR,NODE)>MAXI
     (BESTVAR,NODE) THEN PRINT: PRINT MSG3$: PRINT" Допустимый минимум";
     MINI(BESTVAR,NODE):PRINT"и максимум";MAXI(BESTVAR,NODE): GOTO 2570
2590 VARFLAG(BESTVAR,NODE)=0
2600 RETURN
2610 REM Подпрограмма для определения исхода CORRECT в узле NODE
2620 PRINT: PRINT " Укажите допустимый исход:"
2630 FOR QQ=1 TO OUTCOMES(NODE)
2640 PRINT QQ;" ";OUTCOMES$(QQ,NODE)
2650 NEXT QQ
2660 INPUT " используя его номер ";CORRECT
2670 RETURN
2680 REM Подпрограмма коррекции значений RULES(J,I,NODE)
2690 FLAG=0
2700 FOR I=1 TO OUTCOMES
2710 IF DECISION(I)>=DECISION(CORRECT) AND I<>CORRECT THEN FLAG=1:
     FOR J=1 TO VAR: RULES(J,I,NODE)=RULES(J,I,NODE)-VALUE(J,NODE): NEXT J
2720 NEXT I
2730 IF FLAG=1 THEN FOR J=1 TO VAR: RULES(J,CORRECT,NODE)=RULES(J,CORRECT,NODE)
     +VALUE(J,NODE): NEXT J
2740 RETURN
2750 REM Подпрограмма распечатки заголовка
2760 CLS
2770 PRINT "       ЭКСПЕРТНАЯ СИСТЕМА "
```

```
2780 PRINT STRING$(30,45)
2790 PRINT: PRINT
2800 RETURN
2810 REM Подпрограмма описания массивов
2820 IF INIT%=1 THEN PRINT" Система уже была инициализирована!":
     PRINT" Может появиться сообщение об ошибке.".
     PRINT" Начни сначала, чтобы избежать его.": GOTO 2850
2830 DIM VAR$(MAXVAR,N),VALUE(MAXVAR,N),VARFLAG(MAXVAR,N),RULEVALUE(MAXVAR,N),
     MINI(MAXVAR,N),MAXI(MAXVAR,N),EXAMPLES(MAXVAR+1,50,N),EXS(N),VAR(N),
     OUTCOMES(N),OUTCOMES$(MAXOUT,N),RULES(MAXVAR,MAXOUT,N)
2835 DIM DECISION(MAXOUT),POSSIBLE(MAXOUT),OUTFLAG(MAXOUT,N)
2840 INIT%=1
2850 RETURN
```

Рис. 7.3. Экспертная система, состоящая из нескольких узлов, с «меню»

например мирового господства (как только вы сможете придумать несколько примеров, чтобы дать их системе для обучения).

Комментарии к программе. Программа, приведенная на рис. 7.3, является единственной, где используются файлы на диске. Данная программа, как и все другие в этой книге, написана применительно к расширенному БЕЙСИКу (BASICA) ПЭВМ IBM-PC. То же самое можно сказать и об операциях ввода/вывода. Без учета операторов ввода/вывода оставшаяся часть программы должна работать и с другими версиями БЕЙСИКа, включая и Locomotive BASIC 2, используемый на ПЭВМ Amstrad PC-1542. Однако если у вас действительно возникнут затруднения при работе с данной программой в момент загрузки или записи экспертной системы на диск (опции 6 и 7), то обратите внимание на строки, содержащие операции ввода/вывода, чтобы определить, какую модификацию нужно сделать. Для Locomotive BASIC только две строки нуждаются в такой модификации: 1780 и 2020. Для того чтобы программа работала нормально, они должны быть переписаны в следующем виде:

```
1780 OPEN #3 OUTPUT «EXPERT. DAT»
2020 OPEN #3 INPUT «EXPERT. DAT»
```

Если кому-то покажется странным, почему я выбрал третий номер канала ввода/вывода, то замечу, что этот номер является максимальным, который можно использовать для этих целей в версии BASICA без изменения параметров, устанавливаемых по умолчанию, и минимальным, который применяется для этих целей в версии Locomotive BASIC 2. Это означает, что # 3 работает одинаково хорошо для обоих типов интерпретаторов.

7.8. ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ СИСТЕМЫ

Если вы действительно ввели рассмотренную программу в компьютер, то первое, что вам, видимо, захочется сделать, — это тихо удалиться на неделю куда-то, чтобы залечить свои опухшие от работы пальцы. Но коль скоро вы сделали это, вам не терпится попробовать исполнить программу. В этом случае под рукой должны быть готовы несколько примеров, чтобы вы могли определить, работает система или нет.

Итак, ниже приведено несколько примеров, позволяющих определить, что она будет делать, дополненных фактической распечаткой сеанса работы программы и полученных результатов. Вы сможете увидеть, что должно быть на экране, если вы корректно введете программу.

Замечу, однако, что точный результат, который вы получите, может несколько отличаться от того, который помещен в книге, просто потому, что числа, генерируемые генератором случайных чисел RND, зависят от компьютера — его состояния в момент исполнения программы.

7.8.1. Пример системы с одним узлом. Во-первых, при запуске программы командой RUN вы должны получить меню, состоящее из девяти альтернатив (опций). Выберете первую опцию, чтобы инициализировать систему. Сообщите при этом, что у вас один узел. Введите также максимальное число переменных и возможных исходов, равное 2. Сообщения на экране будут выглядеть следующим образом:

ЭКСПЕРТНАЯ СИСТЕМА

1. Инициализация системы.
2. Ввод примеров.
3. Тренировка экспертной системы.
4. Обучение экспертной системы.
5. Нормальное функционирование системы.
6. Запоминание текущего состояния экспертной системы.
7. Загрузка экспертной системы.

8. Контроль правил и примеров.
9. Выход.
Выберите нужную опцию? 1

ЭКСПЕРТНАЯ СИСТЕМА

Для установки экспертной системы ответьте, пожалуйста, на следующие вопросы:
Сколько узлов имеет система? 1
Каково максимальное число переменных на входе любого из узлов? 2
Каково максимальное число исходов на выходе любого из узлов? 2

На входе этого узла две переменные. Назовите их *Крылья* (с минимальным значением 0 и максимальным — 1) и *Двигатель* (с минимальным значением 0 и максимальным — 1). Сообщите, что у вас два возможных исхода (*Птица* и *Самолет*).

ЭКСПЕРТНАЯ СИСТЕМА

Узел 1
Сколько переменных на входе этого узла? 2
Назовите эти переменные:
Переменная 1? *Крылья*
Минимальное значение переменной? 0
Максимальное значение переменной? 1
Переменная 2? *Двигатель*
Минимальное значение переменной? 0
Максимальное значение переменной? 1

ЭКСПЕРТНАЯ СИСТЕМА

Узел 1
Сколько возможных исходов на выходе этого узла? 2
Назовите, пожалуйста, эти исходы:
Исход 1? *Птица*
Исход 2? *Самолет*

Теперь программа должна вернуть вас к главному меню и вы должны сделать одно из двух: либо пойти на опцию «Обучение системы», начав ее потихоньку обучать, либо пойти на опцию «Ввод примеров», позволив системе самообучаться. Мы выберем опцию «Ввод примеров» для облегчения процедуры и зададим системе два примера.

Она спросит вас относительно переменных и возможных исходов. Введите в качестве первого примера *Птицу*, а в качестве второго примера — *Самолет*.

ЭКСПЕРТНАЯ СИСТЕМА

1. Инициализация системы.
2. Ввод примеров.
3. Тренировка экспертной системы.
4. Обучение экспертной системы.
5. Нормальное функционирование системы.
6. Запоминание текущего состояния экспертной системы.
7. Загрузка экспертной системы.
8. Контроль правил и примеров.
9. Выход.

Выберите нужную опцию? 2

ЭКСПЕРТНАЯ СИСТЕМА

Узел 1

Ввод примеров:
Сколько примеров приготовлено для узла? 2

ЭКСПЕРТНАЯ СИСТЕМА

Пример 1 для узла 1
Переменная 1 (*Крылья*) есть ⟨ 1/0 ⟩ ? 1
Переменная 2 (*Двигатель*) есть ⟨ 1/0 ⟩ ? 0
Укажите возможный исход:
1. *Птица*
2. *Самолет*

используя его номер? 1

ЭКСПЕРТНАЯ СИСТЕМА

Пример 2 для узла 1
Переменная 1 (*Крылья*) есть ⟨ 1/0 ⟩ ? 1
Переменная 2 (*Двигатель*) есть ⟨ 1/0 ⟩ ? 1
Укажите возможный исход:
1. *Птица*
2. *Самолет*

используя его номер? 2

Затем переходим к опции «Тренировка системы». Здесь вы можете удобно устроиться в кресле и расслабиться на минуту, пока эксперт формирует свои правила выбора.

ЭКСПЕРТНАЯ СИСТЕМА

1. Инициализация системы.
2. Ввод примеров.
3. Тренировка экспертной системы.
4. Обучение экспертной системы.
5. Нормальное функционирование системы.
6. Запоминание текущего состояния экспертной системы.
7. Загрузка экспертной системы.
8. Контроль правил и примеров.
9. Выход.

Выберите нужную опцию? 3

ЭКСПЕРТНАЯ СИСТЕМА

Экспертная система тренируется.

После отработки этой опции нужно перейти на опцию 8, чтобы взглянуть на правила, которые выработала экспертная система. Следует просмотреть массив EXAMPLES, содержащий примеры, введенные вами, в виде

1 1 — эта строка соответствует *Крыльям*, в обоих случаях «Да»,
0 1 — эта строка соответствует *Двигателю*, «Да» только для второго примера,
1 2 — эта строка соответствует возможным исходам: 1 для *Птицы* и 2 для *Самолета*.

Массив RULES должен при этом выглядеть следующим образом:

1 —1 — это строка для *Крыльев*,
—2 2 — это строка для *Двигателя*.

Проконтролируйте полученные правила и убедитесь, что они позволяют разделить два исхода так, как это должно быть. Как только вы в этом убедитесь, вам будет проще прийти к мысли о том, что система в целом работает, хотя не обязательно делать это каждый раз.

ЭКСПЕРТНАЯ СИСТЕМА

1. Инициализация системы.
2. Ввод примеров.
3. Тренировка экспертной системы.
4. Обучение экспертной системы.
5. Нормальное функционирование системы.
6. Запоминание текущего состояния экспертной системы.

7. Загрузка экспертной системы.
8. Контроль правил и примеров.
9. Выход.
Выберите нужную опцию? 8

Узел 1

Текущие примеры:

Крылья	1	1
Двигатель	0	1
Исход	1	2

Текущие правила:

	Птица	*Самолет*
Крылья	1	—1
Двигатель	—2	2

Для продолжения программы нажмите любую клавишу, чтобы вернуться к основному меню. Далее переходите на опцию 5 — «Нормальное функционирование».

Сначала программа спросит о наличии *Двигателя*. Заметим, что в массиве RULES эта переменная оказалась самой важной (что и есть на самом деле). Ответьте 1, и система угадает *Самолет* и не задаст больше никаких других вопросов.

ЭКСПЕРТНАЯ СИСТЕМА

1. Инициализация системы.
2. Ввод примеров.
3. Тренировка экспертной системы.
4. Обучение экспертной системы.
5. Нормальное функционирование системы.
6. Запоминание текущего состояния экспертной системы.
7. Загрузка экспертной системы.
8. Контроль правил и примеров.
9. Выход.
Выберите нужную опцию? 5

ЭКСПЕРТНАЯ СИСТЕМА

Узел 1

Нормальное функционирование системы.
Переменная 2 (*Двигатель*) есть ⟨1/0⟩? 1

Предлагается *Птица* в качестве возможного исхода.
Вы хотите продолжить Нормальное функционирование (*y*/*n*)? *n*

ЭКСПЕРТНАЯ СИСТЕМА

1. Инициализация системы.
2. Ввод примеров.
3. Тренировка экспертной системы.
4. Обучение экспертной системы
5. Нормальное функционирование системы.
6. Запоминание текущего состояния экспертной системы.
7. Загрузка экспертной системы.
8. Контроль правил и примеров.
9. Выход.

Выберите нужную опцию? 9
Ок

Если все так, как здесь представлено, то программа работает очень хорошо и вы можете попробовать нечто более интересное.

7.8.2. Пример системы с двумя узлами. Испытаем систему с двумя узлами. С помощью первого узла попробуем установить, является ли объект, который мы задумали, *Машиной* или *Животным.* Выберем, например, в качестве переменных *Крылья* и *Металл*, а в качестве возможных исходов *Животное* и *Машину.*

С помощью второго узла установим, является ли объект *Птицей* или *Самолетом.* В этом случае переменные: *Крылья, Металл, Животное, Машина*, а исходы: *Птица* и *Самолет.*

ЭКСПЕРТНАЯ СИСТЕМА

1. Инициализация системы.
2. Ввод примеров.
3. Тренировка экспертной системы.
4. Обучение экспертной системы.
5. Нормальное функционирование системы.
6. Запоминание текущего состояния экспертной системы.
7. Загрузка экспертной системы.
8. Контроль правил и примеров.
9. Выход.

Выберите нужную опцию? 1

ЭКСПЕРТНАЯ СИСТЕМА

Для установки экспертной системы ответьте, пожалуйста, на следующие вопросы:
Сколько узлов имеет система? 2
Каково максимальное число переменных на входе любого из узлов? 4
Каково максимальное число исходов на выходе любого из узлов? 2

ЭКСПЕРТНАЯ СИСТЕМА

Узел 1

Сколько переменных на входе этого узла? 2
Назовите эти переменные:
Переменная 1? *Оперение*
Минимальное значение переменной? 0
Максимальное значение переменной? 1
Переменная 2? *Металл*
Минимальное значение переменной? 0
Максимальное значение переменной? 1

ЭКСПЕРТНАЯ СИСТЕМА

Узел 1

Сколько возможных исходов на выходе этого узла? 2
Назовите, пожалуйста, эти исходы:
Исход 1? *Животное*
Исход 2? *Машина*

ЭКСПЕРТНАЯ СИСТЕМА

Узел 2

Сколько переменных на входе этого узла? 4
Назовите эти переменные:
Переменная 1? *Оперение*
(Оперение) уже была в узле 1
с минимальным значением 0 и максимальным значением 1 это та же переменная (*y*, *n*)? *y*
Переменная 2? *Металл*
(Металл) уже была в узле 1
с минимальным значением 0 и максимальным значением 1 это та же переменная (*y*/*n*)? *y*
Переменная 3? *Животное*
Минимальное значение переменной? 0

Максимальное значение переменной? 1
Переменная 4? *Машина*
Минимальное значение переменной? 0
Максимальное значение переменной? 1

ЭКСПЕРТНАЯ СИСТЕМА

Узел 2

Сколько возможных исходов на выходе этого узла? 2
Назовите, пожалуйста, эти исходы:
Исход 1? *Птица*
Исход 2? *Самолет*

Обратите внимание на возможность выявлять переменные, которые уже использовались ранее, такие как *Оперение* и *Металл* в узле 2.

Введем теперь несколько примеров. Когда вы будете вводить примеры для узла 1, задумайте для начала *Животное* и отвечайте на вопросы, имея в виду этот исход. Затем задумайте в качестве исхода *Машину* и соответственно этому отвечайте на вопросы. Двух примеров достаточно.

Затем введите два примера для узла 2. Один — предполагающий в качестве исхода *Птицу*, другой — *Самолет*. Будьте особенно внимательны при вводе имен переменных и исходов. Орфографические ошибки могут привести к ошибкам в программе из-за несоответствия имен.

ЭКСПЕРТНАЯ СИСТЕМА

1. Инициализация системы.
2. Ввод примеров.
3. Тренировка экспертной системы.
4. Обучение экспертной системы.
5. Нормальное функционирование системы.
6. Запоминание текущего состояния экспертной системы.
7. Загрузка экспертиой системы.
8. Контроль правил и примеров.
9 Выход.

Выберите нужную опцию? 2

ЭКСПЕРТНАЯ СИСТЕМА

Узел 1

Ввод примеров.
Сколько примеров приготовлено для узла? 2

ЭКСПЕРТНАЯ СИСТЕМА

Пример № 1 для узла 1
Переменная 1 (*Оперение*)? 1
Переменная 2 (*Металл*)? 0
Укажите возможный исход:
1. *Животное*
2. *Машина*
используя его номер? 1

ЭКСПЕРТНАЯ СИСТЕМА

Пример 2 для узла 1
Переменная 1 (*Оперение*)? 0
Переменная 2 (*Металл*)? 1
Укажите возможный исход:
1. *Животное*
2. *Машина*
используя его номер? 2

ЭКСПЕРТНАЯ СИСТЕМА

Узел 2

Ввод примеров.
Сколько примеров приготовлено для узла? 2

ЭКСПЕРТНАЯ СИСТЕМА

Пример 1 для узла 2
Переменная 1 (*Оперение*)? 1
Переменная 2 (*Металл*)? 0
Переменная 3 (*Животное*)? 1
Переменная 4 (*Машина*)? 0
Укажите возможный исход:
1. *Птица*
2. *Самолет*
используя его номер? 1

ЭКСПЕРТНАЯ СИСТЕМА

Пример 2 для узла 2
Переменная 1 (*Оперение*)? 0
Переменная 2 (*Металл*)? 1
Переменная 3 (*Животное*)? 0
Переменная 4 (*Машина*)? 1
Укажите возможный исход:
1. *Птица*

2. *Самолет*
используя его номер? 2

После этого можно провести «Тренировку системы». Когда она будет закончена, выберем опцию «Нормальное функционирование». Прежде всего система спросит вас о *Крыльях*. Предположим, вы задумали *Птицу* и отвечаете 1. Она спрашивает вас далее о *Металле*, и вы отвечаете 0. В результате система предлагает *Животное* в качестве возможного исхода и сразу же переходит на узел 2, где она в свою очередь предполагает *Птицу* в качестве возможного исхода, не задавая больше ни одного вопроса.

Произошло следующее: раз определяется узел 1 и в результате получено, что предполагаемый объект — *Животное*, то система обладает следующей информацией. Объект имеет *Оперение*, но он не металлический, следовательно, объект является *Животным*, а не *Машиной*. Если мы теперь посмотрим на узел 2, то увидим, что это «покрывает» всю информацию, которую требуется знать на входе этого узла. Поэтому система оказывается в состоянии корректно предсказать результат на выходе узла 2, т. е. *Птицу*.

ЭКСПЕРТНАЯ СИСТЕМА

1. Инициализация системы.
2. Ввод примеров.
3. Тренировка экспертной системы.
4. Обучение экспертной системы.
5. Нормальное функционирование системы.
6. Запоминание текущего состояния экспертной системы.
7. Загрузка экспертной системы.
8. Контроль правил и примеров.
9. Выход.

Выберите нужную опцию? 5

ЭКСПЕРТНАЯ СИСТЕМА

Узел 1

Нормальное функционирование.
Переменная 1 (*Оперение*)? 1
Переменная 2 (*Металл*)? 0
Предлагается *Животное* в качестве возможного исхода.

ЭКСПЕРТНАЯ СИСТЕМА

Узел 2

Нормальное функционирование
Предлагается *Птица* в качестве возможного исхода.
Вы хотите продолжить нормальное функционирование (*y*, *n*)? *y*

Проверьте систему еще раз, загадав *Самолет*.

Оперение? Ответ 0; *Металл*? Ответ 1. Система дает *Машину* в качестве возможного исхода и может затем непосредственно перейти к узлу 2, где и придет к выводу, что загадочный объект — *Самолет*.

ЭКСПЕРТНАЯ СИСТЕМА

Узел 1

Нормальное функционирование
Переменная 1 (*Оперение*)? 0
Переменная 2 (*Металл*)? 1
Предлагается *Машина* в качестве возможного исхода.

ЭКСПЕРТНАЯ СИСТЕМА

Узел 2

Нормальное функционирование
Предлагается *Самолет* в качестве возможного исхода.
Вы хотите продолжить нормальное функционирование (*y*/*n*)? *n*

С помощью опции 8 вы можете просмотреть примеры и правила.

ЭКСПЕРТНАЯ СИСТЕМА

1. Инициализация системы.
2. Ввод примеров.
3. Тренировка экспертной системы.
4. Обучение экспертной системы.
5. Нормальное функционирование системы.
6. Запоминание текущего состояния экспертной системы.
7. Загрузка экспертной системы.
8. Контроль правил и примеров.
9. Выход.

Выберите нужную опцию? 8

Узел 1

Текущие примеры:

Оперение	1	0
Металл	0	1
Исход	1	2

Текущие правила:

	Животное	*Машина*
Оперение	1	—1
Металл	—1	1

Для продолжения нажмите любую клавишу.

Узел 2

Текущие примеры:

Оперение	1	0
Металл	0	1
Животное	1	0
Машина	0	1
Исход	1	2

Текущие правила:

	Птица	*Самолет*
Оперение	1	—1
Металл	—1	1
Животное	1	—1
Машина	—1	1

Для продолжения нажмите любую клавишу.

7.8.3. А теперь нечто практически полезное. Надеюсь, что пальцы ваших рук уже отошли и восстановились после усилий, затраченных на ввод программы, и вы чувствуете себя лучше. Вопрос теперь стоит так: может ли экспертная система делать нечто практически полезное? Что-то непохожее на явно тривиальные игры? Хорошо, мы можем дать ей возможность решить одну практическую задачу — найти неисправность кассетного магнитофона.

Предположим, существуют следующие возможные неисправности, которые можно наблюдать в этом случае:

1) нет питания; 2) лента не движется; 3) запись не работает; 4) прерывистый звук; 5) искаженный звук; 6) нестабильная скорость; 7) повышенный фон.

Они возникают в результате совместного действия ряда причин, например: 1) магнитофон не включен; 2) нажата кнопка временного останова; 3) происходит заедание ленты; 4) кассета вставлена неверно; 5) удален предохранительный выступ; 6) загрязнена головка; 7) потянута лента; 8) плохое качество записи; 9) проблемы с усилителем; 10) загрязнен тонвал; 11) плохой ракорд.

Устранить неисправности можно, если: 1) включить питание; 2) отжать кнопку временного останова; 3) заменить кассету; 4) правильно вставить кассету; 5) очистить головку; 6) перезаписать ленту; 7) проверить усилитель; 8) очистить тонвал; 9) проверить ракорд.

Установим экспертную систему из двух узлов. Первый узел предназначается для того, чтобы найти причину неисправности, поэтому в качестве переменных на входе он использует 7 возможных неисправностей, а в качестве своих возможных исходов — 11 причин возникновения неисправностей. Второй узел служит для выработки рекомендаций (указания действий) по устранению неисправностей, поэтому в качестве входных переменных используется 11 причин возникновения неисправностей, а в качестве возможных исходов — 9 рекомендаций (действий) по устранению неисправностей.

Теперь введем примеры.

Для первого узла введем 11 примеров, т. е. по одному на каждую возможную причину неисправности, ответив на вопросы, задаваемые системой, предположив для этого какую-то конкретную причину.

Для второго узла введем 9 примеров, т. е. по одному на каждую рекомендацию по устранению неисправности, ответив на вопросы о причинах, вызвавших эту неисправность. Вы вводите 1 для каждой причины неисправностей, которую можно устранить данным восстанавливающим работоспособность действием, или 0, если причину нельзя устранить данным восстанавливающим действием.

После этого дайте возможность экспертной системе потренироваться (либо, вместо того чтобы вводить примеры и тренироваться, можно провести обучение экспертной системы). Нужно отметить, что чем система больше, тем больше времени занимает ее обучение и тренировка, поэтому не думайте, что компьютер «ушел и пропал», если он на некоторое время «отлучился», чтобы освоить данную задачу и поупражняться в ее решении. Когда он, наконец, вернется, выработав набор правил, вы сможете выбрать опцию 8 и посмотреть на этот набор правил и примеры.

Ниже приводятся примеры для обучения и соответствующий им набор правил.

ЭКСПЕРТНАЯ СИСТЕМА

1. Инициализация системы.
2. Ввод примеров.
3. Тренировка экспертной системы.
4. Обучение экспертной системы.
5. Нормальное функционирование системы.
6. Запоминание текущего состояния экспертной системы.
7. Загрузка экспертной системы.
8. Контроль правил и примеров.
9. Выход.

Выберите нужную опцию? 8

Узел 1

Текущие примеры:

Нет питания	1	0	0	0	0	0	0	0	0	0	0
Лента не движется	1	1	1	1	0	0	0	0	0	0	0
Запись не работает	1	1	1	1	1	0	0	0	0	0	1
Прерывистый звук	0	0	0	1	0	1	1	1	0	0	0
Искаженный звук	0	0	0	0	0	1	1	1	1	1	0
Нестабильная скорость	0	0	1	0	0	0	1	1	0	1	0
Повышенный фон	0	0	0	0	0	0	0	1	1	0	1
Исход	1	2	3	4	5	6	7	8	9	10	11

Текущие правила:

Магнитофон не включен

Нажата кнопка временного останова

Происходит заедание ленты

Кассета установлена неверно

Удален предохранительный выступ

Загрязнена головка

Потянута лента

Плохое качество записи

Проблемы с усилителем

Загрязнен тонвал

Плохой ракорд

Нет питания	3—3—1—2—1—1—1—1—1—1—1
Лента не движется	—5—3—5—4—7—6—6—5—4—6—6
Запись не работает	—8—6—8—8—5—8—8—8—8—8—7
Прерывистый звук	—5—6—6—2—5—4—4—4—6—8—5
Искаженный звук	—5—5—7—8—6—4—6—7—4—3—7
Нестабильная скорость	—5—7—2—7—5—7—3—4—6—3—5
Повышенный фон	—3—3—3—3—4—4—5—1—1—5—1

Для продолжения нажмите любую клавишу.

Узел 2

Текущие примеры·

Магнитофон не включен	1	0	0	0	0	0	0	0	0
Нажата кнопка временного останова	0	1	0	0	0	0	0	0	0
Происходит заедание ленты	0	0	1	1	0	0	0	0	0
Кассета установлена неверно	0	0	1	1	0	0	0	0	0
Удален предохранительный выступ	0	0	1	0	0	0	0	0	0
Загрязнена головка	0	0	0	0	1	0	0	0	0
Потянута лента	0	0	1	0	0	0	0	0	0
Плохое качество записи	0	0	0	0	0	1	0	0	0
Проблемы с усилителем	0	0	0	0	0	0	1	0	0
Загрязнен тонвал	0	0	0	0	0	0	0	1	0
Плохой ракорд	0	0	0	0	0	0	0	0	1
Исход	1	2	3	4	5	6	7	8	9

Текущие правила:

Включить питание

Отжать кнопку временного останова

Заменить кассету

Правильно вставить кассету

Очистить головку

Перезаписать ленту

Проверить усилитель

Очистить тонвал

Проверить ракорд

Магнитофон не включен	1—1—1—1—1—1—1—1—1
Нажата кнопка временного останова	—1 1—1—1—1—1—1—1—1
Происходит заедание ленты	—2—2—1 1—2—2—2—2—2
Лента установлена неверно	—2—2—1 1—2—2—2—2—2
Удален предохранительный выступ	—1—1 2—2—1—1—1—1—1
Загрязнена головка	—1—1—1—1 1—1—1—1—1

Потянута лента	—1	—1	2	—2	—1	—1	—1	—1	—1
Плохое качество записи	—1	—1	—1	—1	—1	1	—1	—1	—1
Проблемы с усилителем	—1	—1	—1	—1	—1	—1	1	—1	—1
Загрязнен тонвал	—1	—1	—1	—1	—1	—1	—1	1	—1
Плохой ракорд	—1	—1	—1	—1	—1	—1	—1	—1	1

Для продолжения нажмите любую клавишу.

Если вы действительно ввели все это, то было бы лучше воспользоваться опциями 6 и 7 для запоминания и загрузки текущей экспертной системы. Эти опции позволяют лишь записать все данные на диск и считать их с диска так, чтобы вы могли многократно запускать ту же экспертную систему, а не вводить ее в компьютер каждый раз заново.

После этого вы вправе перейти к нормальному функционированию экспертной системы, когда она будет пытаться угадать, что произошло с вашим кассетным магнитофоном.

Как именно будет вести себя система, зависит от тех примеров, на которых происходило ее обучение. Ниже показано, что произошло с рассмотренными правилами и примерами.

ЭКСПЕРТНАЯ СИСТЕМА

1. Инициализация системы.
2. Ввод примеров.
3. Тренировка экспертной системы.
4. Обучение экспертной системы.
5. Нормальное функционирование системы.
6. Запоминание текущего состояния экспертной системы.
7. Загрузка экспертной системы.
8. Контроль правил и примеров.
9. Выход.

Выберите нужную опцию? 5

Система спрашивает: была ли скорость нестабильной — ответ 0, затем она интересуется степенью искажения звука, непрерывностью звучания и повышенным фоном — снова ответ 0. Затем она спрашивает, нет ли питания, и если вы ответили 1 (это означает, что питания действительно нет), система немедленно делает вывод, что кассетный магнитофон не был включен, и переходит к узлу 2, где дается рекомендация, чтобы вы включили питание. Это вполне оправдано.

ЭКСПЕРТНАЯ СИСТЕМА

Узел 1

Нормальное функционирование.
Переменная 6 (*Нестабильная скорость*)? 0
Переменная 5 (*Искаженный звук*)? 0
Переменная 4 (*Прерывистый звук*)? 0
Переменная 7 (*Повышенный фон*)? 0
Переменная 1 (*Нет питания*)? 1
Возможный исход: магнитофон не включен.

ЭКСПЕРТНАЯ СИСТЕМА

Узел 2

Нормальное функционирование.
Возможный исход: включить питание.
Вы хотите продолжить нормальное функционирование (*y/n*)? *y*.

Перейдем к другому примеру, где система, узнав от вас, что звук кассетника искаженный и прерывистый, делает вывод о том, что загрязнена головка записи/воспроизведения, и посоветует очистить головку. Снова констатируем, что решение это вполне обосновано.

ЭКСПЕРТНАЯ СИСТЕМА

Узел 1

Нормальное функционирование.
Переменная 6 (*Нестабильная скорость*)? 0
Переменная 5 (*Искаженный звук*)? 1
Переменная 1 (*Нет питания*)? 0
Переменная 4 (*Прерывистый звук*)? 1
Переменная 2 (*Лента не движется*)? 0
Переменная 3 (*Запись не работает*)? 0
Возможный исход: загрязнена головка.

ЭКСПЕРТНАЯ СИСТЕМА

Узел 2

Нормальное функционирование.

Возможный исход: очистить головку.
Вы хотите продолжить нормальное функционирование (*y*/*n*)? *y*.

И, наконец, в третьем примере, когда вы сообщите, что лента не движется, система сделает вывод о том, что кассетник находится в режиме временного останова, и посоветует отпустить кнопку «Временный останов».

ЭКСПЕРТНАЯ СИСТЕМА

Узел 1

Нормальное функционирование.
Переменная 6 *(Нестабильная скорость)?* 0
Переменная 5 *(Искаженный звук)?* 0
Переменная 4 *(Прерывистый звук)?* 0
Переменная 7 *(Повышенный фон)?* 0
Переменная 1 *(Нет питания)?* 0
Переменная 2 *(Лента не движется)?* 1
Возможный исход: загрязнена головка.

ЭКСПЕРТНАЯ СИСТЕМА

Узел 2

Нормальное функционирование.
Возможный исход: отпустить кнопку «Временный останов».

Вы хотите продолжить нормальное функционирование (*y*/*n*)? *n*.

Раз уж вы ввели этот пример и проверили на нем функционирование экспертной системы, то не исключена возможность получения от системы решения какой-то полезной на практике задачи. В этом случае, безусловно, программа потребовала бы значительно больше узлов, переменных и гораздо более сложный набор взаимных связей.

Возможно, вам показалось, что было бы более естественным использовать во многих приведенных до сих пор примерах ответы типа Да/Нет, а не 1/0. Вероятно, вы правы. Фактически вы могли бы разработать программу, поведение которой было бы более естественным для любого заданного объекта. Напомним еще раз, что не все входные переменные обязательно должны иметь значения 0 или 1. Они могут быть также вещественными числами — например такой показатель, как *Уровень осадков*. Разрабатывая экспертную систему, следует иметь это в виду.

Глава 8. КАК ВЫ МОЖЕТЕ ИСПОЛЬЗОВАТЬ ЭКСПЕРТНУЮ СИСТЕМУ

8.1. ВЫБОР ПРОБЛЕМЫ

Предположим, вы хотите построить свою собственную экспертную систему. Но не такую, которая описана в этой книге. Вы хотите построить нечто исключительно свое. В первую очередь надо решить: «В чем эта система будет экспертом?»

Вы можете, при желании, ответить, что она должна быть экспертом в отношении всего, чего угодно. Хотеть этого не вредно. Но если вы ответите так, то, скорее всего, ваша система будет напоминать систему, описанную в предыдущей главе. В конце концов, чтобы быть экспертом абсолютно во всем, нужен предельно общий замысел и абсолютный минимум заранее заданных установок. Камнем преткновения при таком подходе является то, что система может работать для решения широкого круга проблем и не быть особенно хорошей для каждой из них в отдельности.

В идеале вы вправе выбрать для экспертизы не очень широкую, но и не очень узкую область исследований. Это звучит не столь определенно, но на то есть свои причины.

Предположим, что вы избрали очень узкое поле деятельности. Например, диагностирование неисправностей в автомобиле. Вы можете посчитать эту область вполне подходящей — ведь автомеханики требуют за это высокую плату, и было бы неплохо их заменить компьютером. Но посмотрим на эту проблему более внимательно.

Допустим, ваш автомобиль не заводится утром. Вы хотите, чтобы компьютер определил неисправность. Для начала посмотрим в инструкцию по эксплуатации автомобиля, где, вероятно, будет указано, что машина не заводится по одной или нескольким нижеследующим причинам.

Возможно, израсходован бензин, разрядился аккумулятор, в распределитель попала вода или загрязнились контакты свечей зажигания. Заложим это в экспертную систему. Теперь запустим мысленно, как это делали раньше, систему, и на экране «возникает» вопрос: «Есть ли бензин в баке?» Вы идете к машине, чтобы получить ответ на этот вопрос. Если бензин есть, то вам может быть задан вопрос: «Нет ли воды в распределителе?» И опять вы идете смотреть. И если воды нет, то через 5 мин вы станете откручивать свечу, проверяя, не грязная ли она...

Вообще вам нет смысла включать компьютер только для этого. Вы могли бы взять с собой руководство по эксплуатации автомобиля, сэкономив электроэнергию.

А все потому, что определить причину, по которой автомобиль не заводится, очень просто. Настоящая трудность состоит в обследовании разных узлов машины с целью получения первоначально необходимой информации. Компьютер не будет заглядывать в машину вместо вас, поэтому в действительности вам нужен автомеханик.

Конечно, в авторемонтном деле есть задачи, для решения которых целесообразно использовать экспертные системы. Допустим, мы — владелец автомастерской и работающим в ней недостаточно опытным механикам регулярно приходится проверять различные автомобили и выполнять достаточно сложную работу. Вы можете посчитать, что было бы неплохо установить для их консультации компьютер. С другой стороны, вы могли бы просто дать им руководство по ремонту. Но если дело лишь в том, что машина не заводится, то эта проблема слишком «узка́» для того, чтобы ею заниматься всерьез. Проблема должна быть достаточно «крупной», а наибольшая польза будет получена в том случае, если задачу не удалось решить с помощью других средств, например руководства по эксплуатации и ремонту.

Вместе с тем проблема и не должна быть слишком «крупной». Причины этого опять же являются весьма практическими. В первую очередь это проблема построения экспертной системы, достаточно компетентной в широкой области знаний. С расширением проблемной области с вашей стороны требуется больше усилий для воплощения системы, а при недостаточно полном воплощении польза этой системы весьма сомнительна.

Существующие экспертные системы не являются «знатоками», например, во всех вопросах медицинской диагностики. Они компетентны лишь в какой-то узкой области. Все знающая и все умеющая система, часто ставящая неправильный диагноз, может убить столько же больных, сколько вылечить.

Наконец, система может быть наиболее полезной, если удается заставить ее работать. Вероятно, это прозвучит глупо, но представьте, что у вас есть система, которую вы хотели бы видеть экспертом в игре на футбольном тотализаторе. Теперь скажите, хотя бы приблизительно, как вы создадите такую систему? Результаты в некоторых видах спорта и азартных играх могут в какой-то мере быть предсказаны компьютером. Но в футболе? Стоит лишь нескольким основным игрокам прийти на игру с больной от похмелья головой — и исход матча может очень сильно отличаться от любого результата, предсказанного компьютером.

Эта проблема слишком расплывчата, чтобы решать ее с помощью компьютера. Не совсем ясно, какого рода правила управляют ходом игры в действительности, и не совсем очевидно, что существует способ, с помощью которого вы могли бы раскрыть какие-либо правила.

В общем вы можете получить представление о том, есть ли у вас подходящая область для экспертной системы (или любой программы), спросив себя, много ли в этой проблеме такого, что можно измерить. Если да, то у вас есть шанс. Если нет, то забудьте о ней.

Говоря «измерить», мы, конечно, не подразумеваем обязательно измерение длины, ширины и высоты. Ответ типа Да/Нет является мерой, вполне устраивающей компьютер. Но в общем случае вы должны быть в состоянии описать проблемную область, используя единицы измерения какой-либо системы. Если в вашем случае что-либо не удастся свести до измеримых параметров (например, качество игры игрока X), то этот случай вряд ли может быть переложен на компьютер.

8.2. АНАЛИЗ ПРОБЛЕМЫ

Когда вы нашли подходящую проблемную область, то на следующем этапе вы должны начать анализировать ее.

Во-первых, необходимо иметь общее представление о проблеме. Обычно такое представление уже имеется. В конце концов большинство систем не пишутся случайно. Они, как правило, возникают потому, что специалист в какой-либо области считает, что компьютер сможет ему помочь, или потому, что программист знает специалиста, у которого есть проблема. Если у вас нет такого общего представления, то пора понемногу разобраться в выбранной вами области, в первую очередь, чтобы выяснить, правы ли вы были, считая, что вашу задачу можно решить с помощью компьютера, и убедиться, что вы выбрали подходящую проблему.

Затем надо сделать следующий шаг. Вы берете эксперта и используете накопленные им знания по вашему вопросу. Вы можете выстроить эти данные в довольно стройную цепочку: возможные исходы, измеримые свидетельства (переменные) и связывающая их аргументация. Возможные исходы могут быть очень простыми — «в горах есть золото», «в ваших легких бронхит». Что-то вроде этого. Но важно ли само наличие исхода (или его отсутствие)? Или существует другая мера, связанная с этим исходом? Важно сесть и решить, что в идеале и в какой форме вы хотите получить от системы; выработать перечень возможных диагнозов или заключений, рекомендаций, а также указать, как измерить эти результаты.

Переменные — это свидетельства, имеющиеся в распоряжении эксперта-человека. Вам нужно выяснить, что они собой представляют и как измеряются. Задав эксперту вопрос о том, что он надеется выяснить, вы должны знать, что он учитывал, когда приходил к своему заключению.

Эти связи между переменными и представляют собой правила, используемые экспертом. По какой внутренней программе действовал эксперт, идя к своему заключению? Это, может быть, самый сложный вопрос. Вполне возможно, что эксперт-человек не знает всех используемых им правил. Поэтому вы можете лишь попытаться узнать, что, по его мнению, он делает, а потом идти и внедрять полученное, чтобы узнать, что из этого выйдет.

Собрав вместе всю эту начальную информацию, вы обнаружите, что программа легко «вырисовывается» сама собой.

Например, в нашей системе общего назначения требования к общности целей диктовали необходимость того, чтобы она была самообучающейся, вырабатывающей свои правила путем анализа примеров.

Стало совершенно ясно, что конкретная, использованная нами структура логического вывода недостаточно универсальна хотя бы потому, что подходящие результаты для одного случая не оказались столь же существенными в другом случае (т. е. мы создаем систему, полезную до определенных пределов).

Но когда перед вами определенные списки возможных исходов, переменных и определенный перечень правил, ситуация меняется. Если правила могут сильно варьироваться, то вам, возможно, придется написать отдельную программу для каждого из них. Если же они сводятся к обычному стандартному виду, то вы можете сохранить их в файле как информацию, а затем написать программу работы с ними.

Когда правила логически обусловлены, вам удастся написать достаточно простую программу для получения определенных выводов. Если они носят вероятностный характер, то нужен определенный метод обработки этих вероятностей.

Кажется неубедительным утверждение о том, что никто не может дать вам очень подробных советов, что нужно делать, и вы должны сами это решить. На самом деле, это как раз то, что вы должны делать. Ны можете использовать некоторые мысли из этой книги, которые помогут выработать правильный подход. Так же как экспертная система общего назначения терпит неудачу при необходимости учесть конкретные детали, потерпит ее и общий метод построения экспертной системы. Каждая прикладная область имеет свои особенности, предполагающие индивидуальный подход.

Те, кто строил свои экспертные системы, отмечают, что самым трудным было начальное накопление результатов, переменных и правил. После того как они получили эту информацию от эксперта-человека и изложили ее на бумаге, все остальное начинало становиться на свои места. Но даже если на этой стадии не вырисовывается окончательной структуры, отмечается еще один факт, облегчающий в некоторой степени ситуацию, — процесс на этом не заканчивается.

С помощью начальной информации вы создаете нечто для обработки на компьютере — некую пробную программу, которая, по вашему мнению, заработает. Затем вы отлаживаете ее на нескольких примерах и даете полученные результаты для оценки эксперту-человеку.

Обычно система допускает ошибки и между программистом, экспертом и компьютером возникает процесс обратной связи, при котором правила работы программы постепенно изменяются, пока все не будет получаться приемлемым образом.

Иногда вы можете почувствовать, что процесс идет не совсем верно. В конце концов, если вы напишите программу для начисления зарплаты, не зная, как она начисляется, люди могут решить, что вы занялись не своим делом и лучше бы вам было заняться, например, разведением цыплят.

Но именно так выглядит разработка экспертных систем. Можно не соглашаться с тем, что это приемлемый метод в ситуации, когда никто на самом деле не знает, как все должно происходить. Но если, в конце концов, все получится, то, вероятно, использованные средства оправдывают цель.

И надо признать, что известны люди, писавшие даже такие программы, как программы начисления зарплаты, способом, не очень отличным от вышеизложенного, причем они смогли продать то, что у них получилось.

Действительно трудной частью проблемы является ответ на вопрос: зачем любой эксперт будет тратить свое явно дорогое время, выдавая секреты своей профессии некоему лицу, которое запишет их на диск, сделает миллион копий и будет продавать их по 5 долларов за штуку? Такой эксперт наверняка сумасшедший. Создается впечатление, что эксперты считают, что система не будет до конца воплощать их экспертизу, т. е. будет недостаточно эффективна и поэтому не навредит их делу, или что они и в самом деле не против того, чтобы их экспертиза обесценивалась и они вылетели «в трубу». Не новая ситуация. Ведь спросите себя: «Куда делись все клерки, начисляющие зарплату?» Может быть, они встали в очередь за пособием по безработице, желая еще раз встретиться с тем славным программистом? С тем самым, которому они разъяснили запутанное дело начисления зарплаты.

Но может вам следует быть щедрым и поделиться с вашим экспертом доходами? Может стоит быть щедрым и поделиться доходами и со мной?

Глава 9. КРУПНОМАСШТАБНЫЕ ЭКСПЕРТНЫЕ СИСТЕМЫ

9.1. СИСТЕМА MYCIN — МЕДИЦИНСКАЯ ДИАГНОСТИКА

Все, что мы рассматривали до сих пор,— это какая-то одна персональная экспертная система. Существуют, конечно, и другие. Ну, а в какой степени наша система похожа на существующие системы? Лучший ответ мы получим, описав несколько других систем и посмотрев, в чем заключаются сходства с нашей, если они есть.

Система MYCIN — это экспертная система, разработанная для медицинской диагностики. В частности, она предназначена для работы в области диагностики и лечения заражения крови и менингитных инфекций. Система ставит соответствующий диагноз, исходя из представленных ей симптомов, и рекомендует курс медикаментозного лечения любой из диагностированных инфекций. Она состоит в общей сложности из 450 правил, разработанных с помощью группы по инфекционным заболеваниям Стэнфордского университета.

Ее основополагающим моментом, который одновременно может привести и к наибольшим сложностям, является использование вероятностного подхода. Медицинская диагностика — наука не точная. Если пациент укажет определенный набор симптомов того или иного заболевания, то такая взаимосвязь не всегда абсолютна. Рассмотрим различия между системой медицинской диагностики и экспертной системой, например, в области химии. Для упрощения рассмотрим гипотетические системы, так как кроме прочего это поможет нам избежать вторжения в область компетенции эксперта-человека.

Допустим, у меня есть экспертная система для химического анализа, и в качестве информации я задаю ей результаты теста на лакмусовую бумагу. А именно: при опускании лакмусовой бумаги в изучаемый раствор она, например, краснеет. Теперь, исходя из этого, эксперт может «поставить диагноз», что мой раствор кислый. Просто — и никаких сомнений.

Теперь переключимся на систему медицинской диагностики и сообщим ей, что у обследуемого больного сильный кашель. Это может означать, что у него бронхит, туберкулез или просто... сильный кашель. Абсолютной уверенности в значении этого симптома нет.

Но так или иначе экспертная система должна справляться с такой неопределенностью. Наша собственная система делает это в определенных пределах, но не очень точно. Если вспомнить предыдущие дискуссии об использовании вероятностных оценок, то ясно, насколько трудно иметь дело с подобными проблемами.

Система MYCIN справляется с задачей путем назначения показателя определенности каждому из своих 450 правил. Поэтому можно представлять MYCIN как систему, содержащую набор правил вида «ЕСЛИ... ТО» с определенностью P.

А теперь обратите внимание на то, что мы использовали выражение «показатель определенности», а не «вероятность». Почему? Они чем-то отличаются?

Если вы прочитали предыдущие главы книги, то вам известно все о вероятностях. Но вы не осознавали еще, что вероятности могут быть разного типа. Ранее мы занимались (и будем заниматься впредь) статистическими вероятностями. Вся теория математической статистики основана на предположении, что при наличии достаточного числа примеров она будет точно описывать поведение изучаемой системы.

Но есть люди, которые считают, что это неверный подход при использовании систем логического вывода, т. е. систем, изменяющих степень доверия к возможному исходу в зависимости от получаемых входных данных. Принято, что для таких систем больше подходит теория логических вероятностей, чем классическая теория математической статистики, поскольку в случае с системой, основанной на логическом выводе, фактически отсутствует частотная модель происходящего.

Детальный разбор теории логических вероятностей в этой книге не рассматривается, но в качестве некоторой компенсации автор мог бы предположить, что, придерживаясь статистической модели, вы не уйдете далеко от истины, а теоретики логических вероятностей не очень-то разбираются в деталях вычислений, которые нужно делать, даже если вы примете их теории. Другими словами, даже хорошее знание этой теории не улучшит вашу программу. (Такая критика логической вероятности известна как «теория незрелого винограда».)

Тем не менее, система MYCIN оперирует с понятием

Логической вероятности, а *Показатели определенности* в MYCIN — это примерно то, что большинство сочтет условными вероятностями вида $P(H:E)$, т. е. вероятность данной гипотезы H при условии, что событие E осуществилось.

Поскольку последние не являются вероятностями в том смысле, который в них вкладывается, расчеты, использованные до настоящего времени, здесь не совсем подходят, и поэтому в системе MYCIN применяется специальный метод суммирования своих *Показателей определенности* при работе программы.

Для начала выясним, откуда появились эти *Показатели определенности*. В случае MYCIN их предоставили люди-эксперты, которые первоначально изложили и правила. Когда они предложили правило, то указали и свою степень доверия к этому правилу по шкале от 1 до 10.

Вновь возникает вопрос, поставленный ранее относительно вероятностей, — откуда нам известно, насколько точны эти вероятности? На самом деле это нам неизвестно. Несомненно, вероятности достаточно точны, поскольку использование MYCIN дает хорошие результаты. Но метод исключительно специфичен, что, конечно, не порадует статистиков.

Установив эти правила и связанные с ними *Показатели определенности*, MYCIN идет по цепочке назад от возможного исхода, чтобы убедиться, можно ли верить такому исходу. Установив все необходимые исходные предпосылки, касающиеся этого исхода, MYCIN формирует суждение по данному исходу, рассчитанное на основе *Показателей определенности*, связанных со всеми правилами, которые нужно использовать, чтобы прийти к такому исходу.

Допустим, чтобы получить исход Z, требуется определить предпосылки X и Y, дающие возможность вывести Z. Но правила для определения X и Y могут иметь связанные с ними *Показатели определенности* P и Q. Если значения P и Q были равны 1,0, то исход Z не вызывает сомнения. Если P и Q меньше 1,0 (как это обычно бывает), то исход Z не последует наверняка. Он может получиться лишь с некоторой степенью определенности.

Вспоминая предыдущие дискуссии о вероятностях, мы видим, что подсчитать, какова же определенность появления исхода Z в этих условиях, не является очень сложной задачей. Все зависит от точной формы всех составляющих и от того, как они взаимосвязаны друг с другом.

Поэтому MYCIN вместо попыток получить точное решение этой проблемы просто собирает вместе все относящиеся к делу *Показатели определенности*, чтобы дать представление об относительной значимости ответов.

Видимо, необходимо отметить, что MYCIN не выдает диагноз и не раскрывает его точный *Показатель определенности.* Система выдает целый список диагнозов, называя *Показатель определенности* для каждого из них. Все диагнозы с показателями выше определенного специфического для каждого диагноза уровня принимаются как в той или иной степени вероятные, и пользователю вручается список возможных исходов.

Математически это выглядит несколько сомнительно, но на практике все получается достаточно хорошо. Вот вам и математика...

Медицинская диагностика сама по себе — весьма сомнительная процедура. Врач не выдает со стопроцентной гарантией заключений о здоровье каждого из своих пациентов. Он просто считает, что тот или иной диагноз вполне возможен, полагая также возможными и некоторые другие диагнозы. К тому же у больного может быть не одна, а несколько жалоб. В этом случае, чтобы получить один, самый лучший, точный диагноз, надо исключить обоснованную возможность существования не одного, лучшего и точного, а нескольких диагнозов.

Говорят, что медики, имевшие дело с MYCIN, были вполне удовлетворены системой, оценив ее способности наравне со своими. Возможно, в этом и кроется доказательство достоинств системы (не попробуешь — не узнаешь).

Следует отметить также, что в системе MYCIN используется английский язык. Когда системе нужна информация от пользователя, она запрашивает ее по-английски. Пользователь, вводя информацию, делает это тоже по-английски, причем вполне естественным для него способом. На первый взгляд, это удобно, но кроется ли за этим нечто большее?

В этом деле два аспекта. Первый из них касается оценки пользователем конечного продукта (стоило ли это делать), второй — самого исполнения (легко ли было это делать).

Если говорить о первом аспекте, то считается, что одним из преимуществ экспертных систем является то, что ими может воспользоваться любой специалист, обладающий скромными предварительными знаниями о компьютерах. Очевидно, что пользователям больше понравится система, оперирующая на их профессиональном языке, чем та, которая заставит их общаться, например, на БЕЙСИКе. Врачи, например, при работе с системой MYCIN считают ее язык нетрудным и вряд ли захотят тратить время на оценку системы, которой сложно пользоваться. А если никто не желает тратить время на изучение некой системы, то можно утверждать, что ее полезность в расчете на длительную перспективу близка к нулю.

Ну, а практическое осуществление? В конце концов все знают, что организовать работу компьютера на естественном языке — одна из труднейших проблем. Как обойти эту проблему?

Здесь кроется небольшая хитрость, потому что в действительности MYCIN не использует в процессе своего функционирования естественный язык.

Весь фокус в том, что представители каждой профессии любят иметь дело со своим собственным профессиональным жаргоном. В нем есть специальные слова, стереотипные формулировки, которые типичны для их профессии. Можно отметить, что даже у маленьких детей есть свой язык, которым они пользуются для того, чтобы исключить посторонних из своих бесед, и который укрепляет солидарность среди их друзей. У взрослых тоже есть свои языки (включая специалистов по компьютерам — они не хуже других).

Если же подойти к этому более серьезно, то когда кто-то говорит о точных понятиях, он должен всегда придавать словам одно и то же значение. Это приводит к образованию специфического в данной сфере языка, вроде бы для постороннего обычного английского, но на самом деле совсем другого.

Возьмем, к примеру, слова «хронический» и «острый». На медицинском языке они просто обозначают продолжительность заболевания — давно ли оно у пациента («хроническое») или недавно («острое»). Поэтому ясно, что если врач говорит, что у пациента хронический кашель, он не имеет в виду, что кашель просто ужасный. Не имеет он в виду, говоря, что у больного острый приступ кашля, что тот может в любую минуту умереть. Он просто заключает таким образом, как долго его пациент кашляет.

Может быть все это покажется ненужным отступлением, но оно оказывается полезным при использовании экспертной системы. Говоря о MYCIN, было замечено, что врачи, работающие в этой области диагностики, очень точно применяют слова и весьма стереотипные фразы. Гораздо точнее, чем это делает большинство людей при обычной беседе. Поэтому появилась возможность легко определить весьма ограниченный набор слов английского языка, при помощи которого можно выразить все что угодно по данной теме практически без хлопот.

Стандартные фразы и грамматические формы были без труда приспособлены к программе, и в результате получился существенно вырожденный диалект английского языка, легко поддающийся программированию.

Врачи оказались очень довольны таким результатом, потому что, сами не осознавая этого, они говорили, используя очень небольшой набор слов английского языка, по крайней мере, когда сообщали о своей работе. Возможно, они становились знаменитыми ораторами, возвращаясь вечером домой, но это другое дело.

В некотором роде это имеет нечто общее с системой DENDRAL, в которой, как вы увидите позже, вообще не используется английский язык. В ней применяется графический язык, приспособленный к специфической деятельности химиков. Это очень ограниченный «слой» английского языка, но смысл тот же. Приспособьте систему к языку предметной области экспертов, которые ею пользуются, и вы повысите шансы для пользователей, а также сможете лучше применить имеющиеся в этой области знания, которые в некоторой степени и сформировали этот язык.

Однако (всегда должно быть что-то отрицательное), сделав это, вы получите экспертную систему, почти не применимую к другим областям экспертизы только из-за различий с языком другой предметной области.

Что касается удобств для пользователя, система MYCIN обладает качеством, как и многие другие экспертные системы, о котором много говорят, — способностью объяснять потребителю, почему и что она делает.

В качестве простого примера представьте, что ваша экспертная система спрашивает, есть ли у вас кашель. Вместо того чтобы просто ответить «Да» или «Нет», вы можете поинтересоваться у экспертной системы: «Зачем?». Зачем она задала этот вопрос? Тогда система могла бы выдать сообщение, что кашель иногда указывает на больные легкие. Или что-то вроде этого.

Программисты поймут, что это не более чем простой комментарий программы. Каждый раз при программировании в системе какого-либо правила вам нужно лишь запрограммировать короткий текст, объясняющий назначение этого правила. Если система задает вопрос, связанный с этим правилом, то она может выдать подобный текст в качестве разъяснения по пожеланию пользователя (очень похоже на оператор REM [1] в БЕЙСИКе).

Другими словами, здесь нет ничего хитрого. Тогда почему это представляет интерес?

В конечном счете причиной использования такого приема является его привлекательность для пользователя. Пользователей, особенно тех, кто не очень разбирается в компьютерах, впечатляет система, работающая настолько по-человечески, что она в состоянии объяснить свои действия, когда ее об этом просят. И, как уже отмечалось, нет большого смысла создавать систему, которой никто не будет пользоваться.

Но это не все. Возьмем, например, программиста (допустим, вы пишете программу). Программные комментарии весьма полезны. Они, несомненно, помогают в отладке программы, поскольку напоминают вам, зачем вы написали тот или иной блок программы и что он должен делать. Однако сложность при использовании комментариев состоит в том, что вам придется просматривать всю программу, чтобы прочитать их.

Такая жалоба кажется несерьезной, но в длинной программе, которая может остановиться где угодно, имеет право на существование и лучший способ записи происходящего.

Это в некоторой степени объясняет, для чего используются вопросы типа «Зачем?» в экспертной системе, т. е. «живые» комментарии, которые можно вызвать в любой момент, чтобы вспомнить, что происходит, не прерывая выполнение программы.

Теперь отвлечемся немного от вопросов программирования и вспомним о проблемах, действительно возни-

---

[1] REM — оператор, открывающий строку — комментарии в программе на языке БЕЙСИК. — *Прим. пер.*

кающих при построении экспертной системы, — о приобретении знаний и их применении.

Программа содержит набор правил и условий их использования, но обычно этого недостаточно для ее хорошей работы. Эксперт-человек сидит перед дисплеем и отлаживает систему на каком-либо конкретном примере. Неожиданно машина задает глупый вопрос. В этом случае эксперт (человек) может поинтересоваться: «Зачем?» и, получив информацию о работе системы, попытаться определить, в чем ошибка, имея в виду исправить ее, добавив еще одно правило или изменив некоторые старые правила.

Все это не было бы из ряда вон выходящим, если бы в действительности система выдавала стандартный комментарий. Но иногда системные действия и сообщения являются более сложными, и в этих случаях такой подход является гораздо более оправданным.

Рассуждая о MYCIN, заметим, что подобный вариант реализован в системе для изменения набора правил TEIRESIAS. В сущности она очень напоминает средство «трассировки» и аварийной выдачи состояния памяти программы, но оказывается более удобной для пользователя, чем обычный процесс распечатки в листинг шестнадцатеричного содержания оперативной памяти на момент прерывания.

Когда за дисплеем системы MYCIN сидит эксперт-человек, можно задать вопрос «Зачем?» в ответ на запрос информации и получить описание хода рассуждений системы до данного момента. В частности, система отображает правило, запросившее в данном случае информацию, и приводит значения любого или всех составляющих этого правила.

Если вспомнить систему, описанную в начале книги, то вы могли бы поставить ей задачу идентифицировать объект как *Птицу, Самолет* или *Планер*. В какой-то момент система спрашивает вас, есть ли у этого объекта *Двигатель*, и вы задаете вопрос: «Зачем?».

Очевидно, довольно легко определить на данном этапе, в каком состоянии другие переменные. Система могла бы ответить, что она уже знает, что у объекта есть *Крылья* и нет *Клюва*. Если дополнить программу, можно вычислить, что если система получит ответ «Да» на вопрос о *Двигателе*, то сама сделает вывод о том, что этот объект — *Самолет*. В противном случае система придет к заключению, что данный объект — *Планер*.

Если же правила, исходя из которых работала система, были неправильными, то она заявила бы, что при наличии *Двигателя* объект представляет собой *Планер*, что неверно.

Естественно, что такой системе, как MYCIN, необходимо больше информации для формулировки своих заключений, но принцип остается тем же. Вопрос «Зачем?» позволяет увидеть моментальный снимок ситуации в рассуждениях системы, что полезно как для начального развития системы и ее последующей отладки, так и для того, чтобы еще раз убедить пользователей, что она работает не по наитию.

Кроме того, система TEIRESIAS отвечает на вопрос «Как?» по отношению к любому конкретному утверждению. Возвращаясь к нашему примеру *Птица—Самолет—Планер*, можно ввести специальный код, чтобы дать возможность пользователю задать вопрос: «Как? Крылья». Другими словами, если система считает, что у рассматриваемого объекта есть *Крылья*, то как она пришла к этому выводу? В данном случае ответ очень прост, так как сам пользователь сообщил ей об этом. Поэтому довольно просто выдать соответствующее короткое сообщение.

Однако более сложные системы нуждаются в иных методах идентификации, поскольку обычно пользователь не станет спрашивать о верности какого-либо утверждения, сделанного им самим. Он будет интересоваться теми или иными промежуточными выводами, сделанными самой системой.

В данном случае нужно будет идти назад по цепочке рассуждений к этому промежуточному выводу, сделанному системой, чтобы выяснить, какие правила и какая информация использовалась для его получения. Очевидно, что и такую возможность целесообразно учитывать при совершенствовании программы, когда система допустит ошибку, задав, например, неподходящий в данном случае вопрос, а эксперту-человеку захочется узнать, каким образом она дошла до такого состояния. Получается, что если вопрос «Зачем?» дает возможность получить моментальный снимок состояния системы в данный момент, то вопрос «Как?» позволяет проследить, как система пришла к данному состоянию.

Легко заметить, что дополнительные возможности, вроде предоставляемых системой TEIRESIAS, могут быть полезны и при совершенствовании экспертных систем, и в хорошо работающей системе. Если система работает хорошо и в состоянии объяснить свои действия на понятном английском языке, то вы становитесь обладателем системы, которую можно использовать для обучения в доступной ей предметной области.

В конце концов, если у вас есть знакомый студент-медик, не преуспевающий в медицинской диагностике, и прекрасно делающая это компьютерная система, то вы вполне можете усадить его за дисплей и предоставить возможность поучиться у системы. Такая методика, по мнению некоторых, значительно снижает количество апоплексических ударов у тех, кто раньше вынужден был заниматься трудной задачей обучения молодых специалистов.

Программа под названием GUIDON разработана для работы в комплексе с системой MYCIN, чтобы использовать знания последней в процессе обучения. Набор правил системы PUFF (см. § 9.2), опять же с учетом программы GUIDON, адаптирован для работы с MYCIN, что позволило вести обучение при нарушении дыхания.

Конечно, эти экспертные системы можно было бы применять для обучения в том виде, как они есть. Но некоторые модификации системы позволяют ей более гибко управлять действиями обучаемых, непосредственно взаимодействуя с ними больше, чем это было бы возможно, если бы студенты просто сидели, уставившись на экран, и ждали, когда наступит время идти домой.

Учитывая всю работу, проделанную в области экспертных систем медицинской диагностики, можно подумать, что врачи больше не нужны. Несомненно, доля правды в таком утверждении есть, но пока ни одно авторитетное лицо в медицине не высказало мысль о том, что экспертная система может быть допущена к врачеванию на своем участке. И (наверное) если поблизости будет врач-специалист (человек), то гораздо выгоднее все же обратиться к нему по поводу диагноза, чем тратиться на компьютер.

В самом деле, немного жаль. Идея иметь экспертную систему с лотком для выдачи таблеток прямо под клавиатурой довольно привлекательна. Конечно, если не болеешь сам.

9.2. СИСТЕМА PUFF — АНАЛИЗ НАРУШЕНИЯ ДЫХАНИЯ

Познакомившись с системой MYCIN, рассмотрим следующий случай. Что произойдет, если взять MYCIN и потрясти ее над корзиной для бумаг, пока все специфические для нее области знания не будут вытряхнуты в корзину и останется только общий механизм рассуждений?

У вас получится EMYCIN — пустая (Empty) MYCIN, которая будет экспертной системой более или менее общего назначения, но временно не являющейся экспертной ни в какой области. Это и сделали ученые Стэнфордского университета. И, получив пустую EMYCIN, стали «заполнять» ее кое-чем другим.

Это кое-что другое было набором из 50 (или около этого) правил, касающихся легочных заболеваний, которые будучи заложенными в EMYCIN, послужили основой программы, весьма удачно названной PUFF[1] и предназначенной для диагностики нарушений дыхания.

Идея системы мыслилась так: больной заходит в кабинет врача и выдыхает в прибор[2]. В этом нет ничего нового — прибор просто регистрирует объем выдыхаемого пациентом воздуха и скорость его движения при выдохе. Исходя из этих данных, врач в состоянии продиагностировать состояние пациента.

Например, пациент может быть здоров или болен. Болезнь может быть бронхитом, эмфиземой, а может быть и еще чем-то.

Что бы это ни было, смысл заключается в том, чтобы ввести полученную информацию в PUFF, и пусть она ставит диагноз.

Для начала следует отметить, что PUFF не получает информацию непосредственно от прибора, в который дышит больной. Несомненно, его можно модифицировать, чтобы все было именно так, но этого пока не сделано.

На самом деле прибор выдает врачу некоторую, возможно необходимую, информацию о дыхании пациента. Врач также должен представить некоторые сведения о пациенте, например его пол, возраст, курит ли он.

В этот момент можно забыть о приборе, в который

---

[1] В переводе с английского это слово имеет несколько значений, наиболее близкие из них по теме: одышка, тяжелый вздох. — *Прим. пер.*

[2] Этот прибор называется спирограф. — *Прим. пер.*

дышат, и обратить свое внимание исключительно на компьютер, куда загружена экспертная система. Теперь у нас есть список переменных, а также определенные значения, связанные с этими переменными, и машина должна поставить диагноз.

В пробном сеансе системе PUFF были предложены 150 наборов информации о пациенте, и в результате она поставила в 90 % случаев такой же диагноз, как и врач.

Приняв во внимание все переменные, мы могли бы записать их и перечислить возможные исходы (отличное здоровье, бронхит и т. п.). Затем мы вправе представить системе 150 наборов данных для опробования и выработать свои собственные правила, чтобы сформировать диагноз. После этого мы могли бы обнаружить, что иногда она верно поставит последующие диагнозы.

Но все-таки несправедливо по отношению к ученым из Стэнфорда предполагать, что обе системы полностью идентичны, ибо это не так.

Прежде всего PUFF не сама вырабатывает собственные правила. Стэнфордские ученые объединились с учеными из Тихоокеанского медицинского центра, которые и сказали им, как ставить диагнозы. Такой подход является обычным в большинстве используемых сейчас экспертных систем, поскольку не все склонны слишком доверять программам. Кроме того, удается построить более эффективную систему, зная заранее, как она должна работать. Поэтому медики выдвинули набор правил, согласно которым можно ставить диагноз. Специалисты по компьютерам воплотили эти правила в PUFF. Все выглядит весьма просто, хотя стэнфордцы и не заявляли, что это очень трудно.

Если говорить о форме этих правил, то стоит отметить следующие моменты. Как и в MYCIN, правила PUFF являются правилами типа «ЕСЛИ..., ТО...», поэтому мы можем принять такую формулировку (абсолютно не на медицинском языке): «ЕСЛИ (пациент тяжело дышит) и (выкуривает 200 сигарет в день) И (беспрестанно кашляет), ТО (у него кашель курильщика).

Подобным образом (не говоря о деталях, взятых в скобки) многие описывают свои экспертные системы. Это описание имеет огромное преимущество — любой человек, пользующийся компьютером, знаком с функцией оператора «ЕСЛИ..., ТО»[1]. Вы можете даже написать программу на БЕЙСИКе, в которой примените фрагмент с оператором типа «ЕСЛИ..., ТО». (И в этом нет ничего предосудительного, потому что на самом деле PUFF представляет собой пример экспертной системы, которая действительно была переписана на БЕЙСИК.)

Но не попадите в ловушку, считая, что именно так и нужно поступать. Все операторы типа «ЕСЛИ..., ТО» состоят только из логически связанных предположения и заключения. Однако существует довольно много способов прийти к такому же заключению, не используя эту конструкцию.

Например, наша экспертная система не хранит правила в виде «ЕСЛИ..., ТО», но содержит ту же информацию, как если бы это было так. Экспертная система вырабатывает свои правила, и каждый раз, применяя их, она успешно проявляет ту же логику, как если бы в ней были операторы типа «ЕСЛИ..., ТО», условия которых соответствовали бы выработанным ею правилам.

С этими правилами в экспертных системах связан довольно большой терминологический «слой». Обычно они называются «правилами продукции», потому что могут сформировать исход или вывод. Часто первую часть заключения, следующую за «ЕСЛИ», называют «предшествующей», а вторую, следующую за «ТО», — «последующей». Но вы можете называть их «факт и гипотеза», «утверждение и вывод», «переменные и возможные исходы» или любыми другими подходящими для вас терминами. Единственной заслуживающей внимания причиной стандартизировать терминологию является необходимость сделать так, чтобы другие могли вас понять. Однако в подобной относительно новой области знаний пока не существует стандартных терминов для облегчения понимания существа дела.

Первоначально в PUFF входили всего 55 правил, что было весьма обнадеживающим, поскольку в большинстве компьютеров (как считают) можно разместить именно это количество.

Теперь, прежде чем мы начнем рассуждать о том, что делает PUFF со своими правилами, надо сказать еще кое-что. Во-первых, система PUFF работала сначала не

[1] Этот оператор в БЕЙСИКе имеет оригинальный вид IF..., THEN, т. е. «ЕСЛИ...,ТО», — *Прим. пер.*

очень хорошо. Это обычный недостаток большинства программ в процессе разработки. Проблема заключалась в том, что медики не сформулировали с самого начала совершенные диагностические правила. Те правила, которые они предоставили стэнфордцам, попросту не могли логически диагностировать все, с чем они сталкивались. Здесь-то и появляется «инженер знаний».

Во-вторых, система PUFF выдает неверный диагноз. Медики говорят, что этот диагноз плох. Ну, а каким (спрашивает «инженер знаний») он должен быть? И почему? Какое из существующих правил неверное? Какое новое правило нужно добавить?

Медики думают об этом и разрабатывают новые предложения, которые добавляются в базу правил PUFF. Все на первый взгляд выглядит обыденно, процесс манипулирования правилами происходит до тех пор, пока программа не заработает. И в интеллектуальном плане, может, так оно и есть. Но не тогда, когда речь идет о построении экспертной системы.

На уровне механики этого дела должно быть некое «приспособление для залуживания дыр» в системе, чтобы заставить ее работать лучше, — например способ изменения и добавления правил. Ведь почти каждая система нуждается иногда в изменениях, и опасность состоит в том, что если это трудно сделать, то ничего и не делается так часто, как это нужно, или так тщательно, как это нужно. Подобная практическая задача заслуживает определенного внимания.

Обобщая сказанное, можно предположить, что компания медиков могла бы сразу представить верные правила. Но здесь важно отметить, что они не могли этого сделать. У специалиста-человека, работающего над той или иной проблемой, обычно есть некоторое представление о том, как он или она разрешат эту проблему, но большинство людей согласятся, что с самого начала у них нет абсолютно точного представления об этом. По крайней мере, точного в той степени, чтобы заложить его без изменений в компьютер.

Происходящее — скорее процесс взаимного обучения, в котором и компьютерная программа, и человек-эксперт выясняют, как эксперт работал в течение нескольких лет. Возможно, при посредничестве «инженера знаний» между экспертом и программой последняя станет более сложной и гибкой, так как постепенно раскроются точные методы, которыми пользуется эксперт.

Все это указывает на необходимость решения основной проблемы: как заложить знания специалиста в машину? По большинству аспектов она не является проблемой чисто вычислительной техники. Задача состоит в том, чтобы найти эксперта и достаточно хорошо понять, что он говорит, а затем написать программу, способную делать то, что делает эксперт. Рассмотренная проблема зачастую считается самым большим (или самым маленьким) узким местом в деле построения экспертной системы.

Конечно, вы слукавили. Изображая систему, вырабатывающую собственные правила, мы обошли проблему поиска и понимания эксперта-человека, что можно сделать только за счет конечного продукта.

Важно отметить, не говоря о простейших системах, что добавлять правила и видоизменять старые следует без особого напряжения.

Каким именно образом хранятся правила, не очень важно. Они могут быть закодированы в явном виде (с риском, что значительные изменения будет трудно внести) или храниться в виде списка «предшествующие и последующие», как обычные данные.

Последний способ теоретически наиболее популярен у создателей экспертных систем. Обработка правил как данных дает теоретически возможность легко видоизменять их, оставляя в то же время неизменной программу обработки правил.

После сформирования набора правил позволительно спросить, что делает с ними PUFF? В нашей собственной экспертной системе она постепенно перебирала бы эти правила, приходя к тем выводам, которые могли бы быть и тогда, когда они могли бы быть. Но PUFF (и MYCIN) так не поступают, они более гибкие системы.

В целом существуют два метода обработки наборов правил — это так называемое движение по цепочке вперед и назад (рис. 9.1 и 9.2). Описанная нами ранее система использовала движение по цепочке вперед, так как она вырабатывала то, что могла, из того, что у нее было, и, сделав это, выносила следующую предпосылку. Это движение, как следует из названия, представляет собой движение вперед через правила, обусловленное

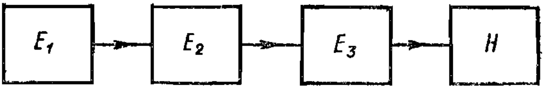


Рис. 9.1. Движение по цепочке вперед. Если правило E_1 верно, то переход к E_2, если и оно верно, то переход E_3 и т. д. к H. Итак, чтобы установить H, все отношения E_1, E_2 и E_3 должны быть истинными. При движении по цепочке вперед (или с управлением от данных) именно так все и происходит. Системе задается E_1, после этого E_2, затем E_3, и, наконец, система выводит суждение о H. При движении по цепочке назад (или с управлением от конечной цели) система сначала рассматривает правило H, желая установить справедливость его существования. Оглядываясь назад, она выясняет, что для установления H ей нужно знать E_3. А чтобы знать E_3, ей требуется знать E_2, а чтобы знать E_2, ей надо знать E_1. Поэтому система запрашивает данные о E_1, E_2, а затем и E_3. После этого она может двигаться вперед — снова к H. Разница между этими методами особенно заметна, если существует большое количество различных выводов (целей, возможных исходов, гипотез, H), к которым система может прийти, и способов достижения поставленных целей

особенностями, содержащимися в заложенной информации.

Движение же по цепочке назад — это действие в обратном направлении (ну и ну!), и оно более мотивировано по содержанию. Действие начинается с выдвижения гипотезы, например «болен ли пациент бронхитом?», и предпринимаются попытки выяснить, так это или нет. Для этого сначала находится правило, одним из возможных исходов которого является бронхит. Затем оно проверяется на предмет того, какие «предшественники» соответствуют этому «последователю», т. е. какие переменные дадут в результате бронхит. Если информации по этим переменным нет, то выбирается одна из них и предпринимается попытка получить информацию по ней. Очевидно, это может быть сделано путем запроса

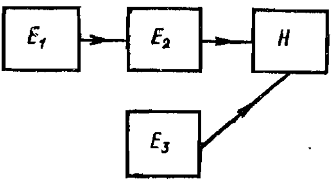


Рис. 9.2. Альтернативное рассуждение. Здесь H легче всего установить, запросив E_3, а не E_1.
В данном случае движение по цепочке вперед может начаться с E_1, которое определит E_2, а затем H. Движение по цепочке назад начнется с H, после чего надо обратиться к E_2 или E_3. Обращение к E_3 даст возможность получить H от E_3, не обращая внимания на E_1

у оператора такой информации или с помощью движения назад через имеющиеся правила с целью найти другое правило, которое, если подойдет, несет нужную информацию как один из своих возможных исходов.

Рассмотренный метод рекурсивен (если вы думали об этом), а рекурсивные методы не очень просто запрограммировать на БЕЙСИКе. Но методологически если подумать о нашей собственной системе, то это похоже на движение к последнему узлу системы (или к тому узлу, возможный исход которого обозначен как некая цель). Затем предпринимается попытка получить соответственно значения на всех входах путем движения назад через все предшествующие узлы, которые задают вход этому конечному узлу.

В пользу обратного движения по цепочке говорит и то, что оно придает системе направленность. Система не пытается ничего выяснить с помощью специальных приемов, она старается конкретно установить, имели или не имели место некоторые важные для нее моменты.

Можно спорить при наличии хорошо сформулированного набора правил, имеет ли большое значение выбор того или иного метода. Но если у вас имеется экспертная система, сформированная в течение длительного времени и содержащая (чтобы щедро раздать их) широкий набор правил, в различной степени полезных) с большим выбором возможных исходов (в различной степени интересных), то она должна позволять максимально быстро получить требуемое заключение и не пытаться установить истину (или ложность) любого события, каждый раз, когда ее используют.

И, наконец, говоря о PUFF, можно задать вопрос, какова польза от этой программы? Почему бы не оставлять проблемы, подобные этой, человеку-эксперту? Прежде всего потому, что специалистов в некоторых областях явно не хватает.

В конце концов специалисты могли бы написать книгу или учебник на эту тему, и хватит. Зачем же беспокоиться о написании компьютерной программы? Частично это предложение обоснованно, и, видимо, есть доля правды в предположении, что люди разрабатывают экспертные системы просто «за интерес».

Однако следует заметить, что первоначальные варианты PUFF, как и других экспертных систем, работали не очень хорошо. Несомненно, люди-эксперты считали, что они знают, что делают, формулируя начальный набор правил. Но оказалось, что они знали это не так уж и хорошо. И если бы они написали книгу по данному вопросу, чего бы она стоила?

Так или иначе, можно сделать вывод: компьютеры — действительно великие думающие машины. Если вы знаете технологию изготовления, например, какой-либо вещи, то можете попытаться запрограммировать ее на компьютере. И если программа не будет хорошо работать, то будьте уверены, что вы не смогли выразить то, что хотели. А это значит, что вы и сами не очень-то в этом разбираетесь. Или сделали это не столь хорошо, как могли бы.

Только тогда, когда программа заработает, вы сможете действительно почувствовать, что полностью изучили проблему. Поэтому упражнение в программировании будет не только упражнением в обучении компьютера, а и самообучением.

Иногда говорят, что человек должен глубоко знать предмет, чтобы его преподавать. По аналогии можно сказать, что нужно очень хорошо владеть предметом, чтобы быть способным обучить кучу микросхем разбираться в его существе.

9.3. СИСТЕМА DENDRAL РАСПОЗНАВАНИЕ ХИМИЧЕСКИХ СТРУКТУР

Рассматривая экспертные системы, мы будем немного удивлены, узнав, что система DENDRAL была создана еще в 1965 г. Конечно, в 1965 г. уже существовали компьютеры, но это был век транзисторов, бумажных перфолент и перфокарт, а не век микросхем и дисплеев.

Все это не относится к самой системе DENDRAL, кроме замечания о том, что это — старейшая, самая разработанная экспертная система в мире. Или, по крайней мере, старейшая система, названная экспертной.

Как и система PUFF, она появилась в Стэнфордском университете и представляет собой плод совместных усилий специалистов по компьютерам и группы экспертов, но на этот раз в области химии. Ее основная идея заключается в следующем.

Химик, приготавливая вещество, часто хочет знать, какова его химическая структура. Для этого существуют разные способы. Во-первых, специалист может сделать определенные умозаключения на основе собственного опыта. Во-вторых, он может исследовать это вещество на спектрометре и, изучая структуру спектральных линий, уточнить свои первоначальные догадки. Во многих случаях это даст ему возможность точно определить структуру вещества, и все будут довольны.

Проблема состоит в том, что все это требует времени и значительной экспертизы со стороны человеческого сообщества. Здесь-то и появляется система DENDRAL, пытающаяся автоматизировать процесс определения правильной химической структуры вещества.

В самых общих чертах процесс принятия решения практически такой же, как в нашей экспертной системе или системе PUFF. Пользователь дает системе DENDRAL некоторую информацию о веществе, а также данные спектрометрии (инфракрасной, ядерного магнитного резонанса и масс-спектрометрии), и та в свою очередь выдает диагноз в виде соответствующей химической структуры. Но если вы чуть глубже посмотрите на работу этой системы, то обнаружите, что различия настолько велики, что за ними почти не видно сходства. Сложности возникают на самом деле в силу существования двух главных проблем. Во-первых, структуру химических веществ нелегко описать при помощи простых слов, а во вторых, количество возможных структур очень велико — существуют буквально миллионы возможных вариантов.

Итак, взглянем на первую проблему — описание структур.

Если вы хоть немного знакомы с химией, то вам приходилось встречать рисунки химических структур атомов и их связей. Вы могли, например, видеть рисунок бензольного кольца, т. е. приблизительно знаете, как оно выглядит. И если у вас есть только бензольное кольцо и еще одна или две другие структуры, то вы можете назвать их и действовать далее во многом так же, как при диагностике, например, бронхита, имея определенную информацию. Проблема заключается в том, что существует множество взаимосвязанных возможностей формирования химических структур. Очень сложно назвать их все. Единственное, что можно сделать, — это нарисовать их. А некоторые структуры настолько замысловаты, что рисунки оказываются очень сложными. Обычно проблему решают путем графического описания каждой структуры, изображая различные узлы и их связи.

Все это хорошо. Но важно заметить, что мы не в состоянии использовать такой подход для описания, например, медицинских вопросов. Язык химиков просто не приспособлен для этого. Можно часто слышать, что смысл экспертной системы заключается в создании логически мыслящей программы общего назначения. Другими словами, экспертная система будет экспертом в любой области по вашему выбору, стоит лишь «вынуть» набор правил, соответствующих одному классу объектов, и вставить новый набор правил, характерных для другого класса объектов.

Система DENDRAL является хорошей иллюстрацией того, насколько этот идеал трудно реализуем, так как она показывает тот предел, за которым один описательный язык хорошо подходит к одной проблеме и не подходит к другой. Очень неудобно (или даже практически невозможно) насильно приспосабливать медицинскую диагностику к языку химиков, как и описание химических структур к языку медицины. Учитывая, что для различных предметных областей больше всего подходят определенные жаргоны, мы обнаруживаем необходимость своего подхода к каждой предметной области.

Вторая проблема касается числа возможных структур. В медицинской диагностике в общем допустимо держать в памяти все возможные исходы, это верно и для некоторых других областей. Но когда число исходов исчисляется миллионами, это становится просто бессмысленным. И тогда возникает вопрос о правильном выборе структуры, если машина не «знает» детально, какие структуры можно получить.

Система DENDRAL решает эту проблему путем выдачи возможных химических структур за один цикл и их последующих проверок, чтобы определить верную структуру. Если бы она делала это без каких-либо ограничений в процессе работы, то могла бы выдать все химические структуры и столкнулась бы с первоначальной проблемой — их оказалось бы слишком много.

Можно для простоты представить систему DENDRAL, состоящей из двух частей, как если бы в одной экспертной системе были две самостоятельные системы. Первая из них содержит набор правил для выработки возможных химических структур. Вводимая информация состоит из ряда заключений, сделанных химиком, и позволяет судить, какие структуры вероятны в том или ином случае. Это в некотором роде очень похоже на системы, рассматриваемые нами до сих пор.

На выходе первой системы имеется не один простой ответ. Обычно это целая серия возможных структур — программа не в состоянии точно сказать, какая из них верна́. Затем DENDRAL «берет» каждую из этих структур по очереди и использует вторую экспертную систему, чтобы определить для каждой из них, каковы были бы результаты спектрального анализа, если бы это вещество существовало и было на самом деле исследовано на спектрометре.

Входными данными для второй экспертной системы является некая химическая структура, выданная программой. Правила, которые она применяет, разработаны профессиональными химиками и соответствуют работе спектрометра. Выходными данными служат показания спектрометра, полученные в результате моделирования. Теперь вспомним, что, кроме того, в систему DENDRAL были введены настоящие показания спектрометра, действительно полученные при исследовании данного вещества. И DENDRAL сравнивает свои гипотетически полученные результаты с настоящими результатами. Если они одинаковы, то это может быть искомая структура, если нет — следует выработать другую структуру.

Процесс, часто именуемый «генерация и проверка», позволяет постоянно сокращать число возможных рассматриваемых вариантов, чтобы в любой момент оно было как можно меньше. В отличие от некоторых экспертных систем DENDRAL задумана не как «игрушка». Она не используется лишь для проверки теоретических основ экспертных систем, а реально применяется для определения химических структур. Написано более 20 научных работ по результатам работы системы DENDRAL с реальными задачами. Говорят, что в своей области она может поспорить с экспертами-людьми. И это, конечно, дает основания для оптимистических надежд, что в скором времени будут созданы другие полезные системы для использования в иных областях знаний. Но до какой степени в них будут применяться принципы, заложенные в DENDRAL, еще предстоит обсудить.

Конечно, общая схема сужения области поиска, основанная на принципе «генерации и проверки», с несколькими первоначальными ограничениями, позволяющая вырабатывать возможные решения и проверять их другими критериями, может быть применена и в других областях. Но саму программу системы DENDRAL, в которой решения описываются графическим языком химических структур и проверяются путем моделирования работы спектрометра, не просто приспособить для этого.

Достаточно общим представляется утверждение о том, что простые экспертные системы удается довольно легко модернизировать для работы в другой области, но их применение в этой области ограниченно. А экспертные системы, приносящие реальную пользу, оказываются очень сложными, и поэтому область их применения ограничена. Будучи приспособленными к решению отдельных специфических задач. они становятся слишком узкоспециальными, нацеленными на эту проблемную область, и их трудно, а то и невозможно приспособить к другим областям.

Как еще одну иллюстрацию сказанного рассмотрим систему META-DENDRAL. Вы помните, что DENDRAL предназначается для моделирования работы спектрометра с целью получения результатов спектрометрии данной химической структуры. Ну, а META-DENDRAL — это экспертная система, которая используется для формирования результатов такого моделирования.

Проблема заключается в следующем: как смоделировать процесс спектрометрирования? Какие правила нужно заложить в экспертную систему, чтобы она могла это сделать? Для ответов на эти вопросы создана система META-DENDRAL. Она работает примерно так же, как и система DENDRAL, т. е. может генерировать полный набор возможных правил, которые она затем применяет к входным данным, и проверять каждое правило на предмет его способности объяснить результаты.

В частности, она получает в качестве информации определенные показатели спектрометра и графическое описание структуры вещества, для которого получены эти результаты. Затем система начинает формировать ряд правил для этих показаний спектрометра, опробует каждое из них и проверяет полученные результаты, выясняя, не дают ли они заданной структуры. Если нет, то она формирует новое правило. Если да, то это правило является допустимым.

Проработав большое количество вариантов, META-DENDRAL смогла сформировать такой набор правил, которые давали показания, похожие на те, что получались исходя из конкретно заданной структуры.

В некотором роде (весьма приблизительно) это напоминает систему, описанную в начале книги, когда мы предъявляли экспертной системе ряд примеров с известными возможными исходами и предоставляли ей возможность выработать для себя набор правил. Квинтэссенция одной из самых главных проблем в данной области состоит в том, что начиная строить экспертную систему, вы порой вообще не знаете, какие правила должны использоваться, и вам нужен способ, желательно несложный, чтобы выяснить это.

Рассматривая систему PUFF, мы заметили, что люди-эксперты сами часто не знают до конца тех правил, которыми они пользуются на практике. И это в области, где они на самом деле являются экспертами! Но почему должны существовать люди-эксперты в области компетенции системы META-DENDRAL? Кто будет тратить свое время, вычисляя показания спектрометра, похожие на те, что дает вещество, молекулярную структуру которого мы и так знаем? В конце концов, если у вас есть такое вещество, то вы можете всегда исследовать его на спектрометре и решить таким образом проблему. А если такого вещества у вас нет, то какая разница, что покажет спектрометр? Необходимость в эксперте в предметной области системы META-DENDRAL возникла только из-за существования самой этой системы и отсутствия специалиста, к которому можно было бы обратиться за помощью. Итак, каждый «танцует от печки».

Проблема, конечно, интересная. Большинство экспертных систем хранят в себе знания людей-экспертов в той или иной форме. Но система, способная подобным образом приобретать знания, существенно меньше полагается на уже имеющуюся сумму знаний и действительно может пополнить наши знания. Причем не только в конкретных случаях, но и в целом в той области, в которой она действует.

Обратимся вновь к примеру с предсказанием погоды. Может быть, вы и не разбираетесь в этом, не столь важно. Вы все равно можете создать экспертную систему и обучить ее на ряде примеров реальной погоды, и со временем она наверняка поймет сущность вопроса и будет достаточно точно предсказывать погоду. Сам факт, что система в состоянии это сделать, означает, что она обладает набором правил лучшим, чем любые правила, имеющиеся у вас для прогнозирования погоды. Возможно, приведенный пример и тривиален, но дело в принципе, а система META-DENDRAL очень хорошо его реализует.

9.4. СИСТЕМА PROSPECTOR — ПОИСК ПОЛЕЗНЫХ ИСКОПАЕМЫХ

Конечно прекрасно, если система умеет лечить больных, и так же здóрово, когда она в состоянии выяснить молекулярную структуру вещества, но вряд ли кто будет возражать, что лучше всего, если система находит золото в горах. Настоящее золото, которое вы можете потратить.

Правда, PROSPECTOR[1] не помогает в поиске именно золота, так как создатели этой системы занимались не столь романтичными, но, тем не менее, достаточно ценными месторождениями. Однако принцип тот же. Система PROSPECTOR — это экспертная система, созданная для содействия поиску коммерчески оправданных месторождений полезных ископаемых. А это уже интересно.

Как считают многие, человек-специалист в этой области нагружает на своего мула баки и тазы, приготавливает достаточно сэндвичей, чтобы продержаться всю зиму, и отправляется в горы, дабы там приложить свои способности. А по весне он, применив уже свои специальные знания, сможет, шатаясь от усталости, прийти в город и заявить свои права на золотоносный участок. После этого его обычно убивают, а его дочь пытается отплатить за него и вернуть золото. Все остальное в этой истории знакомо: она получает золото вместе с рукой того, кто помогал ей, и затем живет счастливо.

Но остается вопрос, на который почти никогда не дают удовлетворительного ответа: «А что же он действительно делал там, в горах?»

Так начинается рассказ о системе PROSPECTOR.

---

[1] В переводе с английского PROSPECTOR — изыскатель, золотодобытчик. — *Прим. пер.*

И как во всякой экспертной системе, прежде всего надо узнать, что же делают настоящие эксперты-изыскатели, когда ищут золото (или более прозаические, но достаточно ценные залежи). Обычный ответ — это ряд выводов и умозаключений, некоторые из них точные, некоторые вероятностные, из которых эксперт формирует суждение по данному вопросу.

Если вам интересно, то метод, используемый в системе, состоит не в том, чтобы, указав компьютеру на обширную территорию, спросить его, где нужно искать золото, — задача будет слишком абстрактной для машины. Важно точно указать определенную территорию, о которой имеются соответствующие данные, и запросить у компьютера оценку вероятности существования здесь определенного месторождения. В некотором роде это напоминает постановку медицинского диагноза, когда вы представляете эксперту (человеку или машине) пациента и спрашиваете, чем он болен. Вы не спрашиваете эксперта о диагнозе всех пациентов вообще.

Изыскатель в своем роде может поступать так же, как и медик. Система PROSPECTOR по аналогии с MYCIN содержит большое количество правил, относящихся к различным объектам, а также возможных исходов, выведенных на их основе. В этой системе используется также «движение по цепочке назад».

Но в отличие от медицинской диагностики полученный результат в PROSPECTOR состоит не только из простого заключения по данному вопросу.

Представьте, что нужно выяснить, чем болен данный пациент. Требуется выбрать один из нескольких возможных исходов. Причем не очень важно установить точную вероятность правильности диагноза. Необходимо просто получить наиболее вероятный диагноз.

Теперь сравните это с поиском полезных ископаемых. В любой конкретной ситуации любой конкретный минерал наверняка будет присутствовать в каком-то количестве, пусть даже микроскопическом. Поэтому вывод о том, что данный минерал присутствует, вряд ли представляет ценность сам по себе. Может быть, не важно знать точное количество, а достаточно знать просто, что этого минерала «много». Именно так и есть. Ответ типа Да/Нет не подойдет, учитывая весьма высокую стоимость изыскательских работ для проверки предсказания системы. Определение степени риска имеет большое значение, так как неправильное решение может стоить дорого.

Допустим, вы поставили больному диагноз. Он может оказаться правильным, и, назначив лечение от этой болезни, вы будете правы. Все хорошо. Но существует вероятность того, что диагноз поставлен неправильно, возможно, вы не нанесете этим больному большого вреда, если все же будете лечить его от определенной вами болезни (правда, не нужно ампутировать конечности по совету компьютера).

Итак, суть медицинской диагностики состоит в распознавании и лечении от возможного заболевания, и если диагноз не подтвердится в действительности, то это может и не иметь существенного значения.

При разведке полезных ископаемых ситуация несколько другая. Очевидно, вам хотелось бы копать землю везде, где может оказаться золото. Но еще важнее — не копать золото там, где его совсем нет. Ведь эта работа потребует времени и стоит денег, и, копая здесь, вы не копаете в другом месте.

Весь фокус состоит в том, чтобы найти наиболее подходящее место и копать там. Поэтому точная оценка вероятности в данной ситуации более важна, чем при медицинском диагностировании. Дело не в том, что золото дороже здоровья (пусть даже это так), а в том, что различны подходы, основываясь на которых, вы действуете.

Заявив так, интересно посмотреть, как система PROSPECTOR использует вероятности, делая свои заключения. Не бесспорно, но кажется, что ее методы являются одними из лучших среди всех разработанных методов для любой из существующих ныне систем.

Самый простой случай — правила, выражающие логические отношения. Это правила типа «ЕСЛИ x, ТО z», где событие z непосредственно вытекает из x. Они остаются такими же простыми, даже если сопоставить x некоторую вероятность, потому что вы вправе оспорить факт: когда вероятность x равна p, то и вероятность z также равна p.

Если у x всего один аргумент, то это правило существенно упрощается. Обычно вместо x мы подставляем более сложное логическое выражение, например $(x \cap y)$ или $(x \cup y)$.

Если элементы отношения связаны с помощью логического И ($\cap$) и отдельным элементам этого отношения сопоставлены определенные вероятности, то система PROSPECTOR выбирает минимальное из этих значений и присваивает эту минимальную вероятность рассматриваемому возможному исходу. Поэтому когда вероятность $x=0{,}1$, вероятность $y=0{,}2$, то вероятность исхода $z=0{,}1$. Легко видеть, почему выбран такой метод: чтобы z было истинным, и x, и y должны быть истинными. Это является «жестким» ограничением, поэтому следует брать минимальное значение.

Правда, этот метод небезгрешен — можете вернуться к разделу о вероятностях. Ошибка иногда заключается вот в чем. Если x и y — независимые переменные, то вероятность $P(x \cap y) = P(x)P(y)$. Если же x и y — зависимые переменные и полностью коррелированы между собой, тогда $P(x) = P(y)$ и $P(x \cap y) = P(x) = P(y)$. В общем случае при частично коррелированных переменных истина находится где-то посередине.

Когда элементы отношения связаны логическим ИЛИ ($\cup$), то в случае независимых переменных имеем $P(x \cup y) = P(x) + P(y) - P(x)P(y)$. А если переменные полностью коррелированы, тогда $P(x \cup y) = P(x) = P(y)$. Фактическое значение при частично коррелированных переменных будет находиться где-то посередине.

Итак, метод системы PROSPECTOR несколько специфичен, но это все же метод, дающий ответ в соответствующих пределах.

В следующем, более интересном методе имеется ряд утверждений (правил), каждое из которых вносит вклад в вероятность некоторых гипотез.

Например, если золото рассыпано вокруг крупными самородками, то мы можем сказать, что «в тех кучах есть золото» с вероятностью 0,9.

Само по себе подобное заявление очень похоже на те, с которыми мы уже встречались. Разница заключается в том, что нас интересует лишь одна гипотеза, связанная с предположением «в тех кучах есть золото». И поэтому существует большое количество утверждений, относящихся к этой гипотезе, подтверждающих ее или противоречащих ей. И тогда возникает проблема: как сосчитать все эти вероятности?

Система PROSPECTOR пользуется методом, основанным на применении формулы Байеса (см. § 2.5) с целью оценки априорной и апостериорной вероятностей какого-либо события. Под «событием» здесь понимается что угодно, например справедливость гипотезы о золоте и кучах. В общих чертах каждая гипотеза имеет априорную вероятность того, что она верна, равную, скажем, $P(H)$. Так, априорная вероятность того, что «в тех кучах есть золото», может быть равна 0,1, поэтому запишем $P(H)=0{,}1$.

Ну, а теперь Вы — золотоискатель, сидите у экрана и даете системе дополнительные свидетельства о наличии золота. При получении этих дополнительных свидетельств, например «только что найден кусок золота размером с кулак», вероятность $P(H)$ изменяется и становится равной $P(H:E)$ — вероятности осуществления гипотезы при наличии нового подтверждающего свидетельства. Таким образом, система корректирует старое значение, используя оператор присваивания $P(H)=$ $=P(H:E)$, а затем проверяет новые свидетельства. Вопрос состоит в том, как вычислить $P(H:E)$.

Что ж, достопочтенный Байес дает ответ. Вот он:

$$P(H:E)=P(E:H)\,P(H)/(P(E:H)\,P(H)+$$
$$+P(E:\text{не }H(P(\text{не }H)).$$

Может быть, лучше всего будет внести это выражение в программу и забыть о нем. Но если вы не в состоянии убедить себя сделать это, то попробуйте другой вариант: $P(H:E)=LS*\ P(H)/(P(\text{не}H)+LS*\ P(H)$ — это, по крайней мере, выглядит проще. Здесь

$$LS=P(E:H)/P(E:\text{не }H).$$

Все приведенные формулы в действительности обозначают одно и то же, так как LS — это *отношение правдоподобия*, известное в основном только среди статистиков. В данном случае оно характеризует отношение вероятности получения данного свидетельства при условии, что гипотеза верна, к вероятности получения этого же свидетельства при условии, что гипотеза неверна.

Итак, рассмотрим гипотезу, что «в тех кучах есть золото». Если она верна, то вероятность того, что вам попадается кусок золота размером с кулак, равна, быть может, 0,3. А если она не верна, то вероятность такого события резко снижается, скажем, до 0,1. В этом случае $LS=3{,}0$.

Вернемся теперь к формуле $P(H:E)$ и получим

$$P(H:E) = 3P(H)/(1-(P(H)+3P(H)) = \\ = 3P(H)/(1+2P(H))$$

Тогда, если первоначально шансы на наличие самородков золота в кучах были «пятьдесят на пятьдесят», т. е. $P(H)=0{,}5$, то $P(H:E)=1{,}5/(1+1)=0{,}75$.

Другими словами, шансы повышаются.

Появляется новое значение $P(H)$ для данной гипотезы, а по мере поступления очередных свидетельств новые значения $P(H:E)$ будут постоянно корректировать $P(H)$.

Но рассказ на этом не заканчивается. Всегда найдется несколько пессимистов, которые отметят, что имеются свидетельства, указывающее на отсутствие золота вокруг. В частности, вы можете обнаружить, что вокруг не рассыпаны куски золота размером с кулак, и вам захочется учесть это. Вычисления будут такими же, как и раньше, за исключением того, что вместо E вы поставите в формулу «неE», чтобы засвидетельствовать отсутствие золота. После этого вы подсчитаете отношение правдоподобия и установите $P(H)$. Важно заметить, что вам понадобится новый набор вероятностей — новое отношение правдоподобия, так как старое основывалось на свидетельстве о наличии золота.

Итак, если мы возьмем выражение $LN=P(\text{не}E:H)/P(\text{не}E:\text{не}H)$ в качестве отношения правдоподобия, связанного со свидетельством об отсутствии золота, то сможем рассчитать новое значение $P(H)=P(H:\text{не}E)$, как и прежде, только вместо LS используем LN.

Допустим, например, вы не нашли кусок золота. Тогда вероятность того, что вы не найдете лежащего поблизости самородка, при условии «золото в кучах есть» может быть равна, скажем, 0,9, а вероятность того, что вы не найдете рассыпанных вокруг самородков при условии «золота в тех кучах нет», равна 1,0, т. е. $LN=0{,}9/1{,}0=0{,}9$. Поэтому отсутствие самородков, рассыпанных вокруг, уменьшит вероятность того, что «золото есть в тех кучах», но незначительно.

В приведенном выше примере снизится значение $P(H)$ с 0,5 примерно до 0,47. Конечно, если система спросила о рассыпанных вокруг самородках и вы ответили, что они есть, то ей не нужно уже будет интересоваться у вас, правда ли, что золотых самородков вокруг **не было**. Соответственно величины LS и LN необходимы для того, чтобы сопоставить результаты двух вопросов (положительного и отрицательного) и исключить необходимость задавать каждый вопрос в отдельности.

В целом правила в системе PROSPECTOR записываются в виде ЕСЛИ...,ТО (LS, LN). Причем каждое правило устанавливается с отношением правдоподобия как для положительного, так и для отрицательного ответа. Значения соответствующих коэффициентов после вычисления оседают в умах экспертов, других людей, изыскателей. Создатели системы при этом задавали экспертам вопросы типа: «Если бы в тех кучах было золото, то каковы, по вашему мнению, были бы шансы, что вокруг разбросаны самородки?» или «Если бы в тех кучах не было золота, то как бы вы оценили шансы того, что самородков вокруг не было?»

Всего требуется найти ответы на четыре вопроса, чтобы перекрыть весь спектр возможностей и задать значения вероятностей $P(E:H)$, $P(E:\text{не}H)$, $P(\text{не}E:H)$, $P(\text{не}E:\text{не}H)$.

На практике же можно свести эти четыре вопроса к двум, отметив, что

$$P(\text{не } E:H) = 1 - P(E:H) \text{ и } P(\text{не } E:\text{не } H) = 1 - P(E:\text{не } H).$$

Очевидно, чтобы не усложнять дела, мы рассмотрели весьма специфический случай, в котором пользователь знает ответы на вопросы, задаваемые системой. В конце концов даже круглый дурак должен знать, есть у него в руке золотой самородок или нет. Но в жизни ответы, как правило, не такие однозначные. Геологи в процессе поиска ископаемых склонны задавать весьма специфические вопросы, такие как: «Повсеместно ли роговая обманка изменилась на биотит?» По правде говоря, можно посочувствовать тому, кто не совсем уверен, как ответить на такой вопрос.

Решение системы PROSPECTOR состоит в том, что она предлагает пользователю шкалу ответов в диапазоне от —5 до +5. Нижний предел — это определенно «Да», верхний — определенно «Нет».

Обычно ответ пользователя находится где-то между крайними значениями, и PROSPECTOR корректирует $P(H)$, учитывая LS и LN, с помощью линейной интерполяции. Вы можете легко представить себе это в виде линейной шкалы, где *LN* — значения слева, а *LS* — справа от шкалы.

Действительно, эта система теоретически весьма элегантна и, как сообщается, хорошо зарекомендовала себя на практике. Но, как и большинство экспертных систем, ее конструкция не настолько модульна, как это кажется сначала.

Основные неприятности появляются, когда пользователь начинает отвечать не просто Да/Нет. Например, системе может потребоваться узнать возраст скальной породы, и ответ будет выражаться определенным числом лет. Возникает ситуация, при которой значения *LN* и *LS* будут не просто значениями. В общем, отношения правдоподобия — это функции, значения которых должны где-то храниться и вычисляться при вводе данных. Проблема уж не такая и сложная, но налицо отход от принципа модульности. Кроме того, строго логические вопросы требуют несколько другого стиля программирования более общих вероятностных закономерностей.

В результате, несмотря на то что принципы, использованные в системе PROSPECTOR, применимы в других областях значений, конкретиую программу порой бывает трудно модифицировать. Это же относится и ко многим экспертным системам.

Тем не менее общее впечатление (опять же во многих системах) таково, что после опроса и выработки ряда правил экспертами самое трудное позади, так как после этого становится до некоторой степени очевидной форма, в которой должна быть реализована (запрограммирована) система. В очень редких случаях некоторые возможные способы ее реализации выглядят непривлекательно или выделяются из общего русла.

Это напоминает разницу между системным анализом и процессом программирования. Если под программированием вы подразумеваете проведение всей работы, то это довольно сложно. Но если программированием вы называете только кодирование структуры, полученной системными аналитиками, то это достаточно просто. В области экспертных систем серьезная проблема заключается в раскрытии этой общей структуры прежде, чем приступят к программированию каждой отдельной области экспертизы.

9.5. ДРУГИЕ ПРИМЕРЫ

Трудность в выборе примеров экспертных систем состоит в том, что само определение экспертной системы настолько широко, что любой может добавить к нему практически что угодно. Это не значит, что определение бессмысленно, наоборот, оно достаточно гибкое: вы можете слегка расширить его (тогда список существующих экспертных систем станет неизмеримо бо́льшим) или жестко ограничить (тогда в него вряд ли попадет хоть одна из существующих экспертных систем).

И это не только наша с вами проблема. Когда готовилось первое издание книги, Центральное агентство по компьютерам и телекоммуникации Великобритании (ССТА) было настолько обеспокоено перспективой насыщения Великобритании дешевыми японскими экспертными системами, что организовало изучение вопроса «кто и что делает в данной области».

Исследование (в нем не было и тени неуважения к любой из заинтересованных сторон) проводилось путем распространения анкеты, которая начиналась с определения экспертной системы (данного в самом начале этой книги), а потом в той или иной форме анкетируемым задавался вопрос, есть ли у них это, и если да, то что оно делает. Подобный подход во многом разумный, но было бы целесообразнее применить его к предмету, в существовании которого почти нет сомнений.

Например, ближе к концу книги вы найдете программу со схемой логического вывода на основе байесовского подхода, которую легко применить для постановки медицинского диагноза. Стоит загрузить ее в свой компьютер, и у вас будет экспертная система, об этом вы можете сообщить ССТА. Если бы все окружающие поступили так же, то количество экспертных систем в Великобритании резко увеличилось бы. Или сделайте то же самое с системой, описанной в первой части книги. Или, опустившись еще ниже, напишите программу, способную что-то делать, неважно что, и скажите, что это экспертная система. Кто будет с вами спорить? Все это говорит о том, что спросить кого-то о наличии у него экспертной системы — далеко не то же самое, что поинтересоваться, есть ли у него, скажем, цветной телевизор.

Тем не менее ниже приводится список некоторых систем (табл. 9.1), отличительной особенностью которых

Таблица 9.1. **Список систем**

Наименование системы	Назначение системы
MYCIN	Медицинская диагностика
PUFF	То же
PIP	» »
CASNET	» »
INTERNIST	» »
SACON	Техническая диагностика
PROSPECTOR	Геологическая диагностика
DENDRAL	Определение химической структуры вещества
SECHS	То же
SYNCHEM	» »
EL	Анализ цепей
MOLGEN	Генетика
MECHO	Механика
PECOS	Программирование
R1	Конфигурирование компьютеров
SU/X	Машинная акустика
VM	Медицинские измерения
SOPHIE	Обучение электронике
GUIDON	Медицинское обучение
TEIRESIAS	Построение базы знаний
EMYCIN	То же
EXPERT	» »
KAS	» »
ROSIE	Построение экспертных систем
AGE	То же
HEARSAY III	» »
AL/X	» »
SAGE	» »
Micro-Expert	» »

является наличие большой базы знаний. Этот перечень, конечно, весьма неполный, потому что происшедшее в последнее время расширение сферы применения экспертных систем означает, что полный перечень будет огромным и устареет почти сразу же после его опубликования.

В этом перечне нужно отметить некоторые интересные моменты. В первую очередь, несомненно, количество экспертных систем, занятых построением других экспертных систем, — возможно, это поможет продемонстрировать сложность определения экспертной системы. Возьмем, к примеру, компиляторы. Это компьютерные программы, предназначенные помогать людям в написании других компьютерных программ. Допуская же существование определения экспертных систем, в том числе и систем, помогающих людям создавать экспертные системы, не столкнемся ли мы так или иначе с определением, столь же широким, как и понятие «компьютерная программа?» Может, все это и не имеет особого значения. В конце концов мы вправе сказать: «Они полезны, и кого волнует, как они называются?» Такая позиция в конце концов может оказаться самой лучшей, иначе существует опасность стать настолько педантичными, что мы будем критиковать работу других лишь потому, что она не укладывается в заготовленные рамки.

Другая интересная группа систем — это системы, используемые для построения баз знаний. Мы уже отмечали, что заложить знания эксперта в экспертную систему не так просто. Для решения этой задачи помощь компьютера будет очень кстати.

Это не означает, конечно, что мы сами себе не можем помочь. Наша система, обучающаяся на примерах, и есть та самая система построения баз знаний. Если у вас имеется собственная действующая экспертная система (которую мы уже описали), то вам интересно убедиться, может ли она работать в какой-либо из перечисленных проблемных областей.

Поставить довольно простой медицинский диагноз, должно быть, весьма несложно. Используйте симптомы в качестве вводимых переменных и те или иные заболевания в качестве возможных исходов. Тогда вы можете на нескольких примерах посмотреть, как система с этим справляется (хотя убедитесь, что система, основанная на байесовском подходе, более эффективна для этих целей).

Возьмем техническую диагностику. Вы можете попробовать построить систему, объясняющую, почему не заводится ваш автомобиль. Что касается геологии, то это зависит больше от того, насколько хорошо вы знаете предмет. Преимущество медицины заключается в том, что за небольшую плату вы в состоянии купить медицинскую энциклопедию, в которой найдете информацию по многим болезням. К сожалению, настольных книг по поиску полезных ископаемых не так много.

Любой интересующийся палеонтологией может попробовать классифицировать ископаемых животных. Например, у пластинчатожаберных была асимметричная раковина, а у брахиоподов — симметричная. Законным тогда является вопрос: что это — пластинчатожаберный или брахиопод? Правда, в данном случае для решения вам вряд ли понадобится компьютер. Достаточно взглянуть на раковину! Но в голове энтузиаста могут родиться и более сложные примеры.

Химия тоже удобный объект. Химический анализ (на школьном уровне) включает серию опытов со специфическими результатами. Поэтому экспертная система — ваш помощник при некоторых видах химического анализа.

Анализ электрических цепей также принципиально возможен. Однако если учесть, что система использует лишь альтернативные ответы типа Да/Нет, она не в состоянии работать с аналоговыми цепями (в которых ток непрерывно изменяется). Но система может очень просто анализировать дискретные процессы. Возьмите систему, составленную из ряда узлов, и пусть каждый узел характеризует один компонент, входное значение которого равно 0 или 1. Свяжите между собой различные узлы. Тогда вам могла бы понадобиться экспертная система, чтобы представить целую плату, составленную из микросхем. При определенных нормальных условиях (т.е. построенное совпадает с замыслом проектировщика) управление микросхемами осуществлял бы другой узел (элемент), предназначенный для того, чтобы отличить нормальные условия от ненормальных. В этой ситуации экспертная система имитировала бы работу микросхем, одновременно управляя операцией для возникновения ненормальных условий.

Однако для многих сфер применения, перечисленных выше, наша экспертная система скорее всего не подойдет, в основном из-за того, что ее назначение слишком уж общее, чтобы она могла удачно адаптироваться к некоторым специфичным сферам применения.

Возьмем, например, МЕСНО — экспертную систему, которая может давать умные ответы на вопросы, касающиеся задач по механике.

Представьте, что у вас есть система, состоящая из блоков, веревок и грузов. На месте одного из грузов — пустая чаша, подвешенная на веревке. При пустой чаше система не уравновешена. Вопрос такой: «Какой груз надо положить в эту чашу, чтобы привести систему в состояние равновесия?» Проблема достаточно хорошо знакома из курса физики. Чтобы ответить на этот вопрос, система должна знать расположение блоков, соответствующие законы физики и уметь применять эти знания для получения ответа. МЕСНО умеет это делать.

Наша многоузловая система тоже в состоянии сделать это, но очень грубо. Считая каждый узел блоком, мы примем, что входы могли бы обозначать груз, подвешенный к этому блоку, а выходы — это веревки, на которых держится блок. Проблема заключается в том, что наша система с ее возможными исходами типа Да/Нет может лишь (пройдя первый блок) сказать, что на данной веревке есть (или нет) сила, действующая на блок, но не определить значение этой силы. Поэтому решение получается недостаточно точным.

Вы можете попытаться решить проблему, «используя» стандартные грузы — например сам факт наличия или отсутствия груза. Затем пусть «управляющий» узел проверит равновесие в какой-либо точке (точках), используя факт равновесия как его вход, а затем устанавливая добавочные грузы здесь и там в соответствии с их входами, пока система не будет сбалансирована. Но здесь (может!) не быть достаточной точности. Конечно, как-то система работает.

Нет другого способа, кроме создания системы, которая, по вашему мнению, может что-то делать, и дальнейшего заключения о том, что же она делает на самом деле. В этом и состоит исследовательская работа.

Глава 10. ЭКСПЕРТНАЯ СИСТЕМА, ОСНОВАННАЯ НА ПРАВИЛАХ ЛОГИЧЕСКОГО ВЫВОДА

10.1. СИСТЕМА, ДЕЙСТВУЮЩАЯ В ОБРАТНОМ ПОРЯДКЕ

Хотя система, описанная ранее, возможно, вас вполне устраивает, в гл. 9 было показано, что существуют и альтернативные системы, которые в определенных ситуациях могут принести больше пользы.

В некотором смысле одной из главных проблем при построении экспертной системы является то, что даже если все необходимые знания налицо, они иногда существуют в неподходящем для применения виде.

Допустим, вы хотите построить экспертную систему в области медицинской диагностики. В этом случае вам вряд ли нужно строить систему, использующую обучение на примерах, как это было в случае с предсказанием погоды, потому что имеется большое количество доступной информации, позволяющей непосредственно решать такие проблемы. Дело в том, что эта информация приведена в неподходящем для обработки на компьютере виде.

Возьмите медицинскую энциклопедию и найдите в ней статью, например, о гриппе. Вы обнаружите, что в ней приведены все симптомы, причем они бесспорны. Другими словами, при наличии указанных симптомов всегда можно поставить точный диагноз.

Но чтобы использовать информацию, представленную в таком виде, вы должны обследовать пациента, решить, что у него грипп, а потом заглянуть в энциклопедию, чтобы убедиться, что у него соответствующие симптомы. Что-то здесь не так. Ведь необходимо, чтобы вы могли обследовать пациента, решить, какие у него симптомы, а потом по этим симптомам определить, чем он болен. Энциклопедия же, похоже, не позволяет сделать это так, как надо. Нам нужна не болезнь со множеством симптомов, а система, представляющая группу симптомов с последующим названием болезни. Именно это мы сейчас и попробуем сделать.

Идеальной будет такая ситуация, при которой мы сможем в той или иной области предоставить машине в приемлемом для нее виде множество определений, которые она сможет использовать примерно так же, как человек-эксперт.

Именно это и пытаются делать такие программы, как PUFF, DENDRAL, PROSPECTOR, так что попробуем и мы.

С учетом байесовской системы логического вывода примем, что бóльшая часть информации не является абсолютно точной, а носит вероятностный характер. Это неплохо звучит, когда, рассказывая другим о вашем подходе, вы подчеркиваете, что используете байесовскую систему логического вывода. Мы же сосредоточим основное внимание на том, в каком формате требуется представить вводимую в программу информацию об области, в которой система должна стать экспертом, потому что сбор этой информации наверняка будет самой трудной частью работы.

Итак, начнем программирование:

```
3000 DATA Симптомы
3010 DATA 1, Симптом_1
3020 DATA 2, Симптом_2
3030 DATA END, END
```

Полученный формат данных мы будем использовать для хранения симптомов. При слове «симптомы» создается впечатление, что мы связаны исключительно с медициной, хотя речь может идти о чем угодно. Суть в том, что компьютер задает множество вопросов, содержащихся в виде символьных строк Симптом_1, Симптом_2 и т. д.

Например, Симптом_1 может означать символьную строку «Много ли вы кашляете?», или, если «вы пытаетесь отремонтировать неисправный автомобиль, — строку «Ослаб ли свет фар?»

Таким образом, вы можете быстро подготовить множество вопросов.

Теперь оформим болезни:

```
2000 DATA Болезни
2010 DATA Болезнь_1,p,[j, py, pn]999
2020 DATA Болезнь_2,p,[j, py, pn]999
```

В таком виде мы будем хранить информацию о болезнях. Это не обязательно должны быть болезни — могут быть любые результаты, и каждый оператор содержит один возможный исход и всю информацию, относящуюся к нему.

Разберем фрагменты программ, представленные в виде отдельных записей. Первое поле записи характеризует название возможного исхода, например *Грипп*. Следующее поле — p — это априорная вероятность такого исхода $P(H)$, т.е. вероятность исхода в случае отсутствия дополнительной информации. После этого идет ряд повторяющихся полей из трех элементов. Первый элемент — j — это номер соответствующего симптома (свидетельства, переменной, вопроса, если вы хотите назвать его по-другому). Следующие два элемента — $P(E:H)$ и $P(E:\text{не}H)$ — соответственно вероятности получения ответа «Да» на этот вопрос, если возможный исход верен и неверен. Последнее поле — 999 — характеризует код остановки, который дан здесь, чтобы программа могла сказать, что она закончила цикл опроса по той или иной болезни.

Например:

2010 DATA Грипп,0.01,1,0.9,0.01,2,1,0.01,3,0,0.01,999

Здесь сказано: существует априорная вероятность $P(H)=0{,}01$ того, что любой наугад взятый человек болеет гриппом.

Допустим, программа задает вопрос 1 (симптом 1). Тогда мы имеем $P(E:H)=0.9$ и $P(E:\text{не}H)=0.01$, а это означает, что если у пациента грипп, то он в девяти случаях из десяти ответит «Да» на этот вопрос, а если у него нет гриппа, он ответит «Да» лишь в одном случае из ста. Очевидно, ответ «Да» подтверждает гипотезы о том, что у него грипп. Ответ «Нет» позволяет предположить, что человек гриппом не болеет.

Так же и во второй группе симптомов (вероятностный) (2,1,0.01). В этом случае $P(E:H)=1$, т.е. если у человека грипп, то этот симптом должен присутствовать. Соответствующий симптом может иметь место и при отсутствии гриппа ($P(E:\text{не}H)=0.01$), но это маловероятно.

Вопрос 3 исключает грипп при ответе «Да», потому что $P(E:H)=0$. Это может быть вопрос вроде такого: «Наблюдаете ли вы такой симптом на протяжении большей части жизни?» — или что-нибудь вроде этого.

Нужно подумать, а если вы хотите получить хорошие результаты, то и провести исследование, чтобы установить обоснованные значения для этих вероятностей. И если быть честным, то получение такой информации — вероятно, труднейшая задача, при решении которой компьютер не сможет существенно помочь вам. Но, если вы сможете написать программу общего назначения, ее основой будет теорема Байеса, утверждающая: $P(H:E) = P(E:H)P(H)/(P(E:H)P(H)+P(E:\text{не}H) * P(\text{не}H))$.

Вероятность осуществления некой гипотезы H при наличии определенных подтверждающих свидетельств E вычисляется на основе априорной вероятности этой гипотезы без подтверждающих свидетельств и вероятности осуществления свидетельств при условиях, что гипотеза верна или неверна.

Поэтому, возвращаясь к нашим болезням, оказывается $P(H:E)=py.p/(py.p+pn.(1-p))$.

В данном случае мы начинаем с того, что $P(H)=p$ для всех болезней. Программа задает соответствующий вопрос и в зависимости от ответа вычисляет $P(H:E)$. Ответ «Да» подтверждает вышеуказанные расчеты, ответ «Нет» тоже, но с $(1-py)$ вместо py и $(1-pn)$ вместо pn. Сделав так, мы забываем об этом, за исключением того, что априорная вероятность $P(H)$ заменяется на $P(H:E)$. Затем продолжается выполнение программы, но с учетом постоянной коррекции значения $P(H)$ по мере поступления новой информации.

Описывая алгоритм, мы можем разделить программу на несколько частей.

Часть 1.

Программа ищет среди операторов DATA информацию о том, сколько записано болезней и симптомов. Вы можете сообщить ей это заранее, но вам не придется пересчитывать их количество, если программа в состоянии сделать это самостоятельно.

Теперь можно описать любые требуемые массивы.

Часть 2.

Программа просматривает операторы DATA, чтобы найти все априорные вероятности $P(H)$. Она также вырабатывает некоторые значения массива правил и размещает их в массиве RULEVALUE. Это делается для того, чтобы определить, какие вопросы (симптомы) являются самыми важными, и выяснить, о чем спрашивать в первую очередь. Если вы вычислите для каждого вопроса RULEVALUE (I) = RULEVALUE (I)+ABS (P(H : E) — P(H : неE)), то получите значения возможных изменений вероятностей всех болезней, к которым они относятся.

Часть 3.

Программа находит самый важный вопрос и задает его. Существует ряд вариантов, что делать с ответом: вы можете просто сказать: «Да» или «Нет». Можете попробовать сказать: «Не знаю», — изменений при этом не произойдет. Гораздо сложнее использовать шкалу от —5 до +5, чтобы выразить степень уверенности в ответе.

Часть 4.

Априорные вероятности заменяются новыми значениями при получении новых подтверждающих свидетельств.

Часть 5.

Подсчитываются новые значения правил. Определяются также минимальное и максимальное значения для каждой болезни, основанные на существующих в данный момент априорных вероятностях и предположениях, что оставшиеся свидетельства будут говорить в пользу гипотезы или противоречить ей. Важно выяснить: стоит ли данную гипотезу продолжать рассматривать или нет? Гипотезы, которые не имеют смысла, просто отбрасываются. Те же из них, чьи минимальные значения выше определенного уровня, могут считаться возможными исходами.

После этого программа возвращается к части 3 и продолжает свою работу.

10.2 ПРОГРАММА НА БЕЙСИКе

Ниже приведено эквивалентное описание этих программ:

```
10 REM Программа определяет ряд болезней и симптомов
20 CLS: PRINT " ЭКСПЕРТ": PRINT "________"
30 READ A$,A$
40 HYPOTHESES%=0: EVIDENCE%=-1
50 WHILE A$<>"Симптомы"
60 READ P,J%
70 WHILE J%<>999
80 READ PY,PN,J%
90 WEND
100 HYPOTHESES%=HYPOTHESES%+1
110 READ A$
120 WEND
130 WHILE A$<>"END"
140 READ A$,B$
150 EVIDENCE%=EVIDENCE%+1
160 WEND
170 DIM P(HYPOTHESES%),RULEVALUE(EVIDENCE%),
    QUESTIONS%(HYPOTHESES%),MINI(HYPOTHESES%),MAXI(HYPOTHESES%),
    VAR%(HYPOTHESES%),VARFLAG%(EVIDENCE%)
180 FOR J%=1 TO EVIDENCE%
190 VARFLAG%(J%)=1
200 NEXT
```

Программа перебирает список болезней и, считывая код останова (число 999), выдает значение HYPOTHESES%, т. е. количество найденных болезней или гипотез. Затем она считывает список симптомов или возможных свидетельств, подсчитывая их, пока не встретит слова END в операторе DATA, и выдает в виде EVIDENCE% число перечисленных симптомов.

Описание массивов:

P(HYPOTHESES%)	Используется для хранения текущих значений вероятностей.
RULEVALUE(EVIDENCE%) . .	Используется для хранения «значения» каждого из свидетельств, выраженного количеством изменений, которые могут быть внесены в вероятности гипотез.
QUESTIONS%(HYPOTHESES%)	Используется для хранения перечня соответствующих вопросов, наиболее важных для каждой гипотезы, уменьшается индекс элемента массива каждый раз, когда задается вопрос.
MINI(HYPOTHESES%)	Используется для хранения минимальных возможных вероятностей, которых может достичь каждая гипотеза.
MAXI(HYPOTHESES%)	Используется для хранения максимальных возможных вероятностей, которых может достичь каждая гипотеза.
VAR%(HYPOTHESES%)	Используется для хранения общего числа вопросов из списка по каждой болезни.

VARFLAG%(HYPOTHESES%) . . Массив флажков, первоначально устанавливаемых на 1. После задания каждого нового вопроса соответствующий флажок устанавливается в 0, чтобы избежать повторного задания того же вопроса.

```
210 REM Определение априорных вероятностей и значений правил
220 RESTORE: READ A$
230 FOR I%=1 TO HYPOTHESES%
240 READ A$,P(I%),J%
250 P=P(I%)
260 WHILE J%<>999
270 READ PY,PN
280 QUESTIONS%(I%)=QUESTIONS%(I%)+1
290 RULEVALUE(J%)=RULEVALUE(J%)+
300 READ J%
310 WEND
320 VAR%(I%)=QUESTIONS%(I%)
330 NEXT
340 REM Определение наиболее подходящих симптома и вопроса
350 RV=0: BESTVAR%=0
360 FOR J%=1 TO EVIDENCE%
370 IF RULEVALUE(J%)>RV THEN BESTVAR%=J%: RV=RULEVALUE(J%)
380 RULEVALUE(J%)=0
390 NEXT
400 IF BESTVAR%=0 THEN PRINT "Симптомов больше нет": END
410 READ A$
420 FOR J%=1 TO BESTVAR%
430 READ P%,A$
440 NEXT
450 PRINT: PRINT: PRINT "Вопрос:": PRINI A$
460 VARFLAG%(BESTVAR%)=0
```

В этот момент программа обнаружила вопрос и выдала его на экран. Вам нужно ответить на него. В программе ваш ответ хранится в переменной RESPONSE%, он задан в диапазоне от —5 до +5.

```
470 REM Анализ ответа пользователя и корректировка вероятностей
480 INPUT "Ответ по шкале от -5 (Нет) до +5 (Да) ";RESPONSE%
490 REM Корректировка вероятностей на основе RESPONSE%
500 RESTORE: READ A$
510 FOR I%=1 TO HYPOTHESES%
520 READ A$,P
530 FOR K%=1 TO VAR%(I%)
540 READ J%,PY,PN
550 IF J%<>BESTVAR% OR QUESTIONS%(I%)=0 THEN 610
560 QUESTIONS%(I%)=QUESTIONS%(I%)-1
570 P=P(I%)
580 PE=P*PY+(1-P)*PN
590 IF RESPONSE%>0 THEN P(I%)=P*(1+(PY/PE-1)*RESPONSE%/5)
    ELSE P(I%)=P*(1+(PY-(1-PY)*PE/(1-PE))*RESPONSE%/5)
600 IF P(I%)=INT(P(I%)) THEN QUESTIONS%(I%)=0
610 NEXT
620 READ S%
630 NEXT
```

Вы, возможно, слегка озадачены расчетами P(I%), изменяющимися в зависимости от того, положительным или отрицательным является значение RESPONSE%. Здесь используется следующая идея Если бы RESPONSE%=+5, то ответ был бы «Да» и тогда

$$P(H:E) = P(E:H)\,P(H)/(P(E:H)\,P(H) + $$
$$+ P(E:\text{не } H(P(\text{не } H)),$$

если бы RESPONSE % = —5, то

$$P(H:\text{не } E) = P(\text{не } E\text{ı}H)\,P(H)/(P(\text{не } E:H)\,P(H) + $$
$$+ P(\text{не } E:\text{не } H)\,P(\text{не } H)),$$

и если бы RESPONSE % = 0, то $P(H:E) = P(H)$.

Если же значение находится между крайними точками, то у нас будет понемногу и того, и другого, т. е. данная программа вычисляет, сколько же этого «одного» и «другого».

Если P(I%)=INT(P(I%)), то P(I%) равно 0 или 1, и мы можем свести QUESTIONS%(I%) к 0, поскольку по поводу этой гипотезы не остается сомнений и, следовательно, по данному элементу списка вопросов больше задавать не надо.

```
640 REM ОПРЕДЕЛЕНИЕ НОВЫХ ЗНАЧЕНИЙ ПРАВИЛ
    И МИНИМАЛЬНЫХ И МАКСИМАЛЬНЫХ ЗНАЧЕНИЙ
    ВЕРОЯТНОСТЕЙ
650 RESTORE
660 READ A$
670 MAXOFMIN=0: BEST%=0
680 FOR I%=1 TO HYPOTHESES%
690 P=P(I%)
700 A1=1: A2=1: A4=1
710 READ A$,PRIOR
720 FOR K%=1 TO VAR%(I%)
730 READ J%,PY,PN
740 IF VARFLAG%(J%)*QUESTIONS%(I%)=0 THEN 810
750 IF PN>PY THEN PY=1-PY: PN=1-PN
760 RULEVALUE(J%)=RULEVALUE(J%)+
770 A1=A1*PY
780 A2=A2*PN
790 A3=A3*(1-PY)
800 A4=A4*(1-PN)
810 NEXT
820 MASI(I%)=P*A1/(P*A1+(1-P)*A2)
830 MINI(I%)=P*A3/(P*A3+(1-P)*A4)
840 IF MAXI(I%)<PRIOR THEN QUESTIONS%(I%)=0:
    PRINT "МОЖНО ИСКЛЮЧИТЬ"; A$
850 IF MINI(I%)>MAXOFMIN THEN BEST%=I%:
    MAXOFMIN=MINI(I%)
860 READ S%
870 NEXT
```

Судя по этому фрагменту программы, вычислить новые значения правил будет довольно просто, хотя не совсем ясно, как это сделать для MINI(I%) и MAXI(I%).

Во-первых, мы сравнивали значение PY с P, чтобы определить, какое из них больше. Если PY больше — отлично, это значит, что ответ «Да» увеличивает значение $P(H)$, а ответ «Нет» уменьшает его. Но нас интересуют события, противоречащие этим ответам, поэтому при работе с вероятностями нужно использовать дополнение, т. е. $(1-PУ)$ и $(1-P)$. Таким образом, мы можем произвольно формулировать свои вопросы, и пока справедливы эти вероятности, программа будет знать, есть ли у нее свидетельства в пользу избранной гипотезы или нет.

Рассмотрим теперь MAXI(I%) — максимально возможную вероятность, и вычислим ее значение, предполагая, что на все неотвеченные вопросы в конце концов будет дан такой ответ, как если бы они поддерживали нашу гипотезу. Снова определяем $P(H:E)$ так, как мы делали бы в процессе корректировки вероятностей. Но для максимально возможного значения считаем E не просто ответом на один вопрос, а ответом на все наиболее существенные вопросы.

Поэтому $P(E:H)$, например, — это вероятность того, что произойдут все свидетельства в пользу выбранной гипотезы при условии, что последняя верна. Поскольку вопросы независимы друг от друга, эта вероятность равна произведению всех наиболее значимых значений $P(E:H)$. Итак,

P (произошли все свидетельства в пользу $H:H)=$
$=P(E_1:H)P(E_2:H)P(E_3\cdot H)...(E_n:H)$,

что является значением, которое мы заложили в $A1$.

Теперь можно записать:

$A1=P$ (все свидетельства E в пользу $H:H$),
$A2=P$ (все свидетельства E в пользу H: неH),
$A3=P$ (все свидетельства E не в пользу $H:H$),
$A4=P$ (все свидетельства E не в пользу H: неH).

Откровенно говоря, те, кто немного запутался, не отчаивайтесь. Я обычно поступаю так: записываю, что, по моему мнению, это значит, и иду подстригать газон. Потом возвращаюсь и записываю то, что нужно записать, если сразу хорошо подумать. Потом снова иду стричь газон. Затем я осознаю, что я не совсем верно все понял в прошлый раз, и вношу изменения в уравнения. И так далее.

К концу дня, когда газон выглядит замечательно и я сижу с бокалом в руке, я вдруг понимаю, что все сделано неверно, и перед тем как заснуть, уже знаю, что мне следовало сделать, но забываю записать это.

На следующий день весь процесс повторяется.

К концу недели я добираюсь до каменной основы моего бывшего газона, и меня уже трясет, наверное, от усталости.

Тогда я вдруг осознаю, что первый ряд уравнений, записанный мной, был правильным, но я потерял листок, на котором их записал.

Вы тоже можете проводить время подобным образом. Польза от этого заключается в том, что лет в 35, а то и раньше, вам можно будет назначать пенсию по инвалидности. А может быть, вы сможете все это скрупулезно спланировать с самого начала.

Тем не менее, имея *A1*, *A2*, *A3*, *A4*, вы в состоянии рассчитать минимальное и максимальное возможные значения исходов, как это показано. Нужно лишь скопировать программу и отметить, что научный труд — это плод нечеловеческих страданий.

Наконец, мы получаем наиболее вероятные заключения о возможных исходах согласно программе:

```
880 REM Поиск наиболее вероятного исхода
890 FOR I%=1 TO HYPOTHESES%
900 IF MINI(BEST%)<=MAXI(I%) AND I%<>BEST% THEN MAXOFMIN=0
910 NEXT
920 IF MAXOFMIN=0 THEN 340
930 RESTORE: READ A$
940 FOR I%=1 TO BEST%
950 READ BEST$,A$
960 FOR K%=1 TO VAR%(I%)
970 READ J%,PY,PN
980 NEXT
990 READ A$
1000 NEXT
1010 PRINT: PRINT: PRINT "Наиболее вероятный исход - ";BEST$
1020 PRINT "с вероятностью";P(BEST%)
1030 END
2000 DATA Болезни
2010 DATA Болезнь 1,0.01,1,0.8,0.1,2,0.2,0.9,999
2020 DATA Болезнь 2,0.5,1,1,0.5,2,0,0.1,999
3000 DATA Симптомы
3010 DATA 1,Симптом 1
3020 DATA 2,Симптом 2,END,END
```

Чтобы понять, что это дает, надо взять максимальное и минимальное возможные значения и сравнить их с максимумом любых других значений.

Если оно больше, то остановитесь, все закончено.

Если нет, то какой-то другой возможный исход мог оказаться более вероятным, чем этот конкретно исход. Вы не в состоянии пока сделать окончательный вывод — придется вернуться и начать изучение следующего элемента списка.

Если все это выглядит слишком сложно, то можете просто взять результат с самым большим значением $P(I)$, основываясь на уже заданных вопросах, и считать его верным, если вероятность достаточно высока для вас, или, по крайней мере, достаточно верным.

Чтобы все окончательно встало на свои места, я приведу пример конкретного результата, который вы можете получить на экране дисплея.

```
EXPERT
Вопрос:
Симптом_1
Отвечайте по шкале —5 (Нет) до +5 (Да) 5
Вопрос:
Симптом_2
Отвечайте по шкале —5 (Нет) до +5 (да) 5
Болезнь 2 можно исключить
Наиболее вероятный исход — болезнь 1
С вероятностью 1.764057E—02
Ок
RUN
EXPERT
Вопрос:
Симптом_1
Отвечайте по шкале —5 (Нет) до +5 (Да) —5
Болезнь 2 можно исключить
Наиболее вероятный исход — болезнь 1
С вероятностью 2 239642E—03
Ок
```

Несмотря на то что эта программа работает неплохо (чудо современной техники), многое в ее работе зависит от списка вопросов, которым вы ее снабдили. Теоретически расчеты производятся в предположении, что каждый вопрос не зависит от остальных. В противном случае степень достоверности ответов ухудшается.

Например, если вы спросите: «Есть ли у вас температура?», а затем — «Вас лихорадит?», то очевидно, что заданные вопросы взаимосвязаны. Просто нет смысла задавать оба этих вопроса, но если вы все же зададите их, то это изменит значения вероятностей, связанных с соответствующими исходами, потому что фактически одно и то же свидетельство будет учтено дважды.

Кроме того, вы должны задать системе достаточное количество вопросов, чтобы она имела возможность реально поставить тот или иной диагноз. Может, этот комментарий излишен, но, допустим, вы привели описание обычной простуды и гриппа, считая, что при обоих заболеваниях у вас будет плохое самочувствие, насморк и кашель. Не исключено, что вероятности в каждом случае будут несколько отличаться друг от друга, но ведь они должны существенно отличаться, чтобы программа «смогла увидеть» разницу. Необходимы четкие, недвусмысленные вопросы, способные «развести» возможные исходы настолько четко, насколько это возможно.

Скорее всего вы будете работать, как и любой другой человек, разработавший экспертную систему, т. е. напишите программу и снабдите ее набором правдоподобных правил, соответствующих тем определениям, которые вы поместили в операторах DATA.

Потом вы притворитесь больным пневмонией, если у вас медицинская экспертная система, или представите, что сел ваш аккумулятор, если ваша экспертная система занимается технической диагностикой, и, помня об этом, ответите на вопросы, которые задаст вам система.

Затем система выдаст неверный ответ или задаст глупый вопрос — и вам придется возиться с этими вопросами и вероятностями, пока она не станет работать хорошо. На самом деле описанный процесс не имеет никакого отношения к компьютерам, но очень важен для понимания предмета, в котором вы хотели бы, чтобы ваш компьютер стал экспертом. Это уже само по себе очень интересно.

В противном случае вы всегда можете вернуться к системе, описанной в начале книги, — подбросьте ей лишь несколько примеров, и пусть она работает самостоятельно.

10.3. МЕДИЦИНСКАЯ БАЗА ЗНАНИЙ

Если вы реализуете только что описанную байесовскую схему логического вывода, то прежде чем от нее будет толк, потребуется еще одна вещь — *база знаний.* Строго говоря, это ваше дело обеспечить ее некими специфичными для данной области знаниями. Но такая работа может отнять много времени. Поэтому предлагаем в качестве некоторой помощи базу знаний в области медицинской диагностики.

Если вы загрузите полученные данные о болезнях в компьютер с помощью операторов DATA со строки 2000, а их симптомы — со строки 3000 и далее, то экспертная система будет иметь информацию примерно о 100 различных заболеваниях и их диагностике. Приведенные цифры приблизительно верны, и хотя вы не сможете практиковать как врач, у вас появится возможность погрузиться в экзотические фантазии ипохондрика. Кроме того, вы получите представление о построении базы знаний в других областях.

Загрузив базу знаний, попробуйте изменить некоторые позиции, и вы увидите, как это влияет на экспертные способности системы.

Например, вы хотите, чтобы экспертная система учитывала тот факт, что хронический бронхит (болезнь номер 10) чаще встречается у мужчин, чем у женщин (симптом 53). Поэтому к описанию болезни 10 следует добавить цифры 53, 0.8, 0.5, т. е. если у пациента хронический бронхит, то он мужского пола с вероятностью 0,8, а если у пациента нет хронического бронхита, то вероятность, что пациент мужского пола, равна 0,5. Другими словами, здоровое население — это «смесь» из одинакового количества мужчин и женщин.

Если вы добавите еще симптом 53 к достаточному числу болезней, то вы, скорее всего, обнаружите, что значение RULEVALUE (53) растет, и один из вопросов, который, консультируясь у вас, задаст система уже на ранней стадии, будет просьба указать пол пациента. Что, видимо, небезосновательно — мало кто из врачей захочет ставить диагноз без такой информации.

Теперь попробуйте рассмотреть и другие болезни, например бубонную чуму, и посмотрите, сможет ли экспертная система продиагностировать ее. Если возникнет необходимость в дополнительных симптомах (вопросах), то ими можно легко дополнить список переменных с конца и пронумеровать их, как уже имеющиеся симптомы.

Затем попытайтесь написать базу знаний со своими вопросами, симптомами и «болезнями», относящимися, например, к проблеме: «почему ваша машина не заводится по утрам?» (может, у нее бубонная чума?)

```
2000 DATA Болезни
2010 DATA Простуда,0.02,1,0.9,0.05,2,0.8,0.02,3,0.8,0.02,5,0.6,
     0.01,6,1,0.01,7,0.2,0.01,8,0.5,0.01,15,0.8,0.01,34,0,0.01,
     999
2020 DATA Аллергический ринит, 0.01,1,1,0.01,2,1,0.01,6,0.9,
     0.01,10,0.7,0.1,11,0.7,0.01,12,0.6,0.01,20,0.9,0.01,999
2030 DATA Синусит,0.01,14,0.08,0.01,13,0.9,0.01,15,0.8,0.01,7,
     0.6,0.01,22,0.5,0.01,2,0.5,0.001,6,0.5,0.01,63,0.9,0.01,999
2040 DATA Фарингит,0.02,3,1,0.01,16,0.9,0.01,8,0.5,0.01,11,0.9,
     0.01,37,0.8,0.3,64,0.4,0.01,999
2050 DATA Тонзиллит,0.001,3,1,0.01,7,0.9,0.01,15,1,0.01,16,0.7,
     0.01,19,0,0.5,8,0.8,0.01,34,0,0.01,64,0.8,0.01,999
2060 DATA Грипп,0.01,3,0.9,0.01,1,0.9,0.01,6,0.5,0.01,7,0.7,
     0.01,8,1,0.01,15,1,0.01,17,0.8,0.01,18,0.6,0.01,34,0,0.01,
     999
2070 DATA Ларингит,0.01,4,1,0.01,8,0.6,0.01,15,0.05,0.01,16,
     0.17,0.01,37,0.8,0.3,999
2080 DATA Опухоль гортани,0.00004,4,1,0.01,34,0.99,0.01,37,0.8,
     0.3,999
2090 DATA Острый бронхит,0.005,5,0.01,0.01,8,1,0.01,12,1,0.01,
     15,1,0.01,18,0.5,0.01,21,1,0.01,31,0.9,0.01,34,0,0.01,22,
     0.9,0.01,999
2100 DATA Хронический бронхит,0.005,5,1,0.01,12,0.9,0.01,14,0.5,
     0.01,21,1,0.01,22,0.8,0.01,34,1,0.01,36,0.9,0.01,37,0.8,
     0.3,999
2110 DATA Астма,0.02,12,0.8,0.01,22,1,0.01,23,0.5,0.01,24,0.5,
     0.01,25,0.5,0.01,26,0.5,0.01,31,0.8,0.01,999
2120 DATA Эмфизема,0.01,22,1,0.01,5,0.001,0.01,26,0.8,0.01,12,
     0.001,0.01,21,0.01,0.01,37,0.8,0.3,999
2130 DATA Пневмония,0.003,8,1,0.01,15,1,0.01,18,0.8,0.01,22,1,
     0.01,23,0.5,0.01,26,0.5,0.01,28,1,0.01,29,0.02,0.01,27,0.2,
     0.01,31,0.9,0.01,36,1,0.9,7,0.9,0.01,17,0.9,0.01,32,0.5,
     0.001,999
2140 DATA Плеврит,0.001,31,0.8,0.01,32,0.8,0.01,22,0.5,0.01,5,
     0.8,0.01,8,0.9,0.01,15,1,0.01,0,0.01,999
2150 DATA Пневмоторакс,0.0002,18,0.8,0.01,22,0.8,0.01,32,0.8,
     0.01,999
2160 DATA Бронхоэктаз,0.00001,21,1,0.0,27,0.5,0.01,5,1,0.01,14,
     0.5,0.01,999
2170 DATA Абсцесс легкого,0.00001,33,0.9,0.01,18,0.5,0.01,21,
     0.5,0.01,27,0.5,0.01,999
2180 DATA Пневмокониоз,0.001,22,1,0.01,36,1,0.01,21,0.8,0.01,9,
     1,0.001,999
```

```
2190 DATA Рак легкого,0.001,5,1,0.01,21,0.8,0.01,27,0.5,0.01,
     22,0.5,0.01,18,0.8,0.01,12,0.5,0.01,37,0.99,0.3,999
2200 DATA Интерстициальный фиброз,0.00001,22,0.8,0.01,35,0.8,
     0.01,21,0.6,0.01,999
2210 DATA Отек легкого,0.001,22,0.9,0.01,25,0.9,0.01,30,0.5,
     0.01,27,0.5,0.01,26,0.5,0.01,12,0.8,0.01,999
2220 DATA Гастрит,0.01,41,0.8,0.01,43,0.8,0.01,42,0.5,0.01,8,
     0.4,0.01,37,0.9,0.5,999
2230 DATA Хиатальная грыжа,0.001,18,0.9,0.01,32,0.5,0.001,42,
     0.8,0.001,57,0.9,0.01,16,0.9,0.01,41,0.8,0.01,999
2240 DATA Язва двенадцатиперстной кишки,0.01,37,0.8,0.2,42,
     0.99,0.001,41,0.8,0.01,999
2250 DATA Язва желудка,0.01,42,0.9,0.001,18,0.5,0.01,20,0.8,
     0.01,41,0.7,0.01,56,0.9,0.01,62,0.0001,0.01,999
2260 DATA Дивертикулярная болеэнь,0.001,42,0.6,0.001,43,0.5,
     0.01,41,0.5,0.01,8,0.5,0.01,49,0.5,0.01,999
2270 DATA Болезнь Крона,0.0001,42,0.9,0.001,43,0.9,0.01,15,0.9,
     0.01,8,0.7,0.01,62,0.00001,0.01,999
2280 DATA Расстройство кишечника,0.00001,42,0.9,0.00,43,0.8,
     0.01,41,0.5,0.01,999
2290 DATA Аппендицит,0.001,34,0.1,0.9,42,0.9,0.001,41,0.8,0.01,
     8,0.8,0.01,44,0,0.5,999
2300 DATA Пищевое отравление,0.001,42,0.5,0.001,41,0.9,0.01,43,
     0.9,0.01,7,0.8,0.01,999
2310 DATA Гастроэнтерит,0.01,41,0.8,0.01,42,0.7,0.001,43,0.9,
     0.01,8,0.5,0.01,999
2320 DATA Каменнопочечная болевнь,0.001,42,0.7,0.001,999
2330 DATA Острый пиелонефрит,0.001,42,0.9,0.001,8,0.8,0.01,41,
     0.7,0.01,67,0.9,0.01,999
2340 DATA Желчный конкремент,0.01,42,0.5,0.001,41,0.5,0.01,57,
     0.9,0.01,999
2350 DATA Холецистит,0.001,42,0.8,0.001,8,0.9,0.01,41,0.8,0.01,
     45,0.8,0.001,999
2360 DATA Герпес,0.001,42,0.5,0.001,18,0.5,0.001,60,0.9,0.01,
     59,0.9,0.01,2,0.6,0.01,46,0.5,0.01,999
2370 DATA Глубокий тромбоз вен,0.0005,40,0.8,0.01,999
2380 DATA Ревматический артрит,0.001,15,0.8,0.01,17,0.8,0.01,
     40,0.5,0.001,999
2390 DATA Сердечная недостаточность,0.001,22,0.9,0.01,36,0.5,
     0.01,25,0.5,0.001,12,0.6,0.01,18,0.5,0.001,32,0.3,0.001,
     40,0.5,0.01,42,0.5,0.01,28,0.3,0.001,47,0.9,0.01,999
```

```
2400 DATA Состояние тревоги,0.01,46,0.9,0.01,28,0.3,0.01,47,
     0.6,0.01,39,0.8,0.01,23,0.6,0.01,48,0.6,0.01,16,0.3,0.01,
     43,0.2,0.01,22,0.5,0.01,50,0.5,0.01,57,0.5,0.01,58,0.5,
     0.01,15,0.5,0.01,7,0.6,0.01,4,0.5,0.01,999
2410 DATA Депрессия,0.01,47,0.5,0.01,7,0.5,0.01,49,0.5,0.01,50,
     0.5,0.01,15,0.6,0.01,62,0.8,0.01,999
2420 DATA Коронарный тромбоз,0.01,18,0.5,0.01,32,0.9,0.001,20,
     0.5,0.01,36,0,0.2,38,0.6,0.01,22,0.5,0.01,23,0.6,0.01,41,
     0.5,0.01,15,0.9,0.01,999
2430 DATA Ангина,0.01,37,0.8,0.37,18,0.9,0.01,36,0.9,0.01,22,
     0.5,0.01,27,0.5,0.01,38,0.5,0.01,41,0.3,0.01,999
2440 DATA Легочная эмболия,0.0001,22,1,0.01,18,0.7,0.01,21,0.6,
     0.01,27,0.5,0.001,25,0.5,0.001,,26,0.4,0.0001,999
2450 DATA Припадок,0.001,28,0.8,0.01,38,0.7,0.01,51,0.8,0.001,
     58,0.9,0.01,61,0.9,0.01,999
2460 DATA Ишемическая болезнь сердца,0.001,28,0.8,0.01,38,0.7,
     0.01,51,0.8,0.001,34,01,0.01,20,0.5,0.01,58,0.9,0.01,61,
     0.9,0.01,999
2470 DATA Туберкулез,0.0001,7,0.5,0.01,8,0.5,0.01,12,0 5,
     0.01,15,0.5,0.01,18,0.5,0.01,5,0.5,0.01,30,0.5,0.01,27,
     0.5,0.001,22,0.5,0.01,62,0.0001,0.01,23,0.5,0.01,999
2480 DATA Геморрой,0.01,52,0.9,0.001,49,0.8,0.01,56,0.9,0 01,
     59,0.5,0.01,999
2490 DATA Гипотиреоз,0.001,49,0.8,0.01,17,0.5,0.01,24,0,0.01,
     23,0.001,0.01,39,0.001,0.01,4,0.5,0.01,43,0,0 01,46,0.001,
     0.01,48,0 001,0.01,62,0.9,0.05,999
2500 DATA Слизистый колит,0.0007,43,0.5,0.01,49,0.5,0.01,42,
     0.8,0.001,41,0.3,0.01,57,0 9,0.01,999
2510 DATA Рак толстой кишки,0.001,43,0.9,0.01,49,0.9,0.01,52,
     0.5,0.001,42,0.5,0.001,56,0.9,0.01,62,0.001,0.01,999
2520 DATA Язвенный колит,0.0004,42,0.8,0.001,43,0.8,0.01,52,
     0.6,0.001,23,0.5,0.01,41,0.5,0.001,8,0.5,0.01,34,0.4,
     0.01,56,0.9,0.01,999
2530 DATA Болезнь Меньера,0.0005,38,0.9,0.001,41,0.8,0 01,34,
     0.5,0.01,20,0.9,0.01,999
2540 DATA Шейный спондилез,0.006,54,0.9,0.01,7,0 5,0.01,39,0.5,
     0.01,58,0.9,0.01,61,0.5,0.01,999
2550 DATA Субдуральное кровотечение,0.000001,55,0.99,0.0001,28,
     0.9,0.001,7,0.9,0.01,41,0.9,0.01,38,0.9,0.01,20,0.5,0.01,
     34,0.5,0.01,999
2560 DATA Опухоль мозга,0.000001,7,0.9,0.01,41,0.9,0.01,38,0.8,
     0.01,50,0.8,0.01,34,0.5,0.01,999
```

```
2570 DATA Менингит,0.000001,8,0.9,0.01,7,0.9,0.01,41,0.9,0.01,
     28,0.7,0.01,54,0.9,0.01,2,0.9,0.01,60,0.5,0.01,999
2580 DATA Субарахноидальное кровотечение,0.000001,7,0.99,0.01,
     54,0.9,0.01,38,0.7,0.01,28,0.7,0.01,41,0.8,0.01,2,0.8,
     0.01,999
2590 DATA Острая глаукома,0.01,2,0.9,0.01,7,0.9,0.01,41,0.7,
     0.01,20,0.8,0.01,0.34,0.8,0.01,63,0.8,0.01,68,0.9,0.01,999
2600 DATA Височный артериит,0.001,7,0.9,0.01,17,0.7,0.01,2,0.8,
     0.01,63,0.99,0.01,999
2610 DATA Диспепсия,0.1,18,0.7,0.01,57,0.7,0.01,42,0.7,0.01,41,
     0.7,0.01,46,0.5,0.01,20,0.9,0.01,999
2620 DATA Блокада сердца,0.0003,22,0.5,0.01,58,0.9,0.01,39,0.6,
     0.01,18,0.6,0.01,999
2630 DATA Пернициозная анемия,0.0004,22,0.9,0.01,58,0.9,0.01,
     39,0.9,0.01,36,0.9,0.01,45,0.5,0.01,42,0.5,0.01,50,0.5,
     0.01,28,0.4,0.01,999
2640 DATA Мигрень,0.1,7,1,0.01,15,0.9,0.01,41,0.9,0.01,43,0.5,
     0.01,20,0.9,0.01,34,0.9,0.01,63,0.99,0.01,999
2650 DATA Гипертония,0.15,7,0.5,0.01,39,0.5,0.01,15,0.5,0.01,
     34,0.9,0.01,999
2660 DATA Экзема,0.3,59,0.9,0.01,60,1,0.01,999
2670 DATA Крапивница,0.03,59,0.9,0.01,60,1,0.01,46,0.5,0.01,999
2680 DATA Чесотка,0.001,59,1,0.01,60,1,0.01,999
2690 DATA Корь,0.02,15,1,0.01,8,1,0.01,6,0.9,0.01,2,0.9,0.01,
     11,0.9,0.01,5,9,0.01,43,0.5,0.01,6,0.8,0.01,7,0.5,0.01,
     34,01,0.01,999
2700 DATA Краснуха коревая,0.01,8,0.5,0.01,60,0.9,0.01,54,0.2,
     0.01,34,0,0.01,64,0.5,0.01,999
2710 DATA Ветряная оспа,0.001,60,1,0.01,59,1,0.01,8,0.8,0.01,7,
     0.5,0.01,15,0.5,0.01,34,0,0.01,999
2720 DATA Псориаз,0.02,46,0.6,0.01,3,0.5,0.01,60,0.99,0.01,59,
     0.5,0.01,999
2730 DATA Питириазная рожа,0.01,60,1,0.01,59,0.9,0.01,34,0.5,
     0.01,999
2740 DATA Розовые угри,0.01,60,0.9,0.01,2,0.5,0.01,34,0.8,
     0.01,999
2750 DATA Тиреотоксикоз,0.0001,46,0.9,0.01,47,0.8,0.01,48,0.9,
     0.01,23,0.9,0.01,39,0.9,0.01,22,0.8,0.01,43,0.8,0.01,62,
     0.00001,0.01,2,0.5,0.01,24,0.9,0.01,64,0.3,0.01,68,0.3,
     0.01,999
2760 DATA Сахарный диабет,0.01,62,0.0001,0.01,61,0.5,0.01,2,
     0.5,0.01,66,0.99,0.01,68,0.1,0.01,999
```

```
2770 DATA Рак желудка,0.0003,41,0.5,0.01,42,0.7,0.001,62,
     0.0001,0.01,52,0.6,0.001,56,0.5,0.01,999
2780 DATA Фибрилляция предсердия,0.001,39,0.8,0.01,38,0.5,
     0.01,42,0.4,0.01,58,0.5,0.01,999
2790 DATA Лимфогранулематоз,0.0001,23,0.5,0.01,63,0.6,0.01,54,
     0.8,0.01,59,0.7,0.01,64,0.99,0.01,999
2800 DATA Инфекционный мононуклеоз,0.001,8,0.9,0.01,7,0.9,0.01,
     3,0.9,0.01,15,0.9,0.01,64,0.8,0.001,54,0.8,0.01,45,0.5,
     0.001,60,0.5,0.01,999
2810 DATA Лимфома,0.0001,64,0.9,0.01,54,0.8,0.01,15,0.8,0.01,
     62,0.001,0.01,8,0.8,0.01,23,0.5,0.01,59,0.8,0.01,999
2820 DATA Свинка,0.01,64,0.99,0.01,8,0.8,0.01,15,0.9,0.01,16,
     0.7,0.01,54,0.6,0.01,3,0.8,0.01,999
2830 DATA Паралич Белла,0.0003,51,0.9,0.01,63,0.5,0.01,999
2840 DATA Болезнь Паркинсона,0.001,48,0.9,0.01,51,0.8,0.01,42,
     0.3,0.01,50,0.2,0.01,28,0.1,0.01,999
2850 DATA Ревматическая лихорадка,0.01,3,0.8,0.01,15,0.8,0.01,
     0.8,0.8,0.01,64,0.8,0.01,60,0.5,0.01,59,0.001,0.01,48,0.1,
     0.01,999
2860 DATA Цистит,0.01,66,0.9,0.01,65,0.9,0.01,67,0.9,0.01,8,
     0.5,0.01,999
2870 DATA Опухоль почки,0.001,8,0.6,0.01,62,0.0001,0.01,0.41,
     0.5,0.01,42,0.5,0.01,65,0.7,0.01,999
2880 DATA Опухоль мочевого пузыря,0.0004,65,0.9,0.01,42,0.5,
     0.01,66,0.5,0.01,57,0.5,0.01,8,0.3,0.01,999
2890 DATA Ирит,0.0005,2,0.9,0.01,68,0.9,0.01,999
2900 DATA Острый гепатит,0.001,8,0.8,0.01,15,0.8,0.01,17,0.5,
     0.01,42,0.5,0.01,45,0.5,0.01,41,0.5,0.01,999
3000 DATA Симптомы
3010 DATA 1,Много ли вы чихаете?
3020 DATA 2,Болят ли или слезятся ваши глаза?
3030 DATA 3,Болит ли у вас горло?
3040 DATA 4,Охрип ли ваш голос?
3050 DATA 5,Много ли вы кашляете?
3060 DATA 6,Есть ли у вас насморк?
3070 DATA 7,Болит ли у вас голова или вы вообще страдаете от
     головных болей?
3080 DATA 8,Есть ли у вас повышенная температура (более 37
     градусов С)?
3090 DATA 9,Приходится ли вам подолгу находиться в запыленной
     атмосфере?
3100 DATA 10,Испытываете ли вы кожный зуд?
3110 DATA 11,Пересохло ли у вас в горле?
```

```
3120 DATA 12,Слышны ли у вас хрипы при дыхании?
3130 DATA 13,"Забит" ли у вас нос?
3140 DATA 14,Была ли у вас в последнее время простуда или
     другая подобная инфекция?
3150 DATA 15,Ощущаете ли вы общее болезненное состояние?
3160 DATA 16,Ощущаете ли вы трудности при глотании?
3170 DATA 17,Болят ли у вас мышцы?
3180 DATA 18,Ощущаете ли вы боли в груди?
3190 DATA 19,Удалены ли у вас гланды?
3200 DATA 20,Есть ли у вас симптомы, проявляющиеся в виде
     "приступов"?
3210 DATA 21,Выделяется ли у вас мокрота при кашле?
3220 DATA 22,Есть ли у вас одышка?
3230 DATA 23,Много ли вы потеете не только при физической
     нагрузке, но и в состоянии покоя?
3240 DATA 24,Учащен ли ваш пульс? Нормальный пульс - 60-80
     ударов в минуту или немного чаще для лиц старше 70 и
     моложе 20 лет.
3250 DATA 25,Случаются ли у вас сильные приступы одышки,
     которые вызывают серьёзную озабоченность?
3260 DATA 26,Наблюдается ли у вас посинение кожи?
3270 DATA 27,Есть ли кровь в мокроте, когда вы кашляете?
3280 DATA 28,Не ощущаете ли вы себя сконфуженным происходящим
     вокруг?
3290 DATA 29,Не наблюдаете ли вы у себя (или у пациента)
     проявлений бессвязной речи и плохой координации движений?
3300 DATA 30,Бывает ли у вас сухой (без выделения мокроты)
     кашель?
3310 DATA 31,Ощущаете ли вы боль при дыхании или кашле?
3320 DATA 32,Бывают ли у вас очень сильные боли в груди?
3330 DATA 33,Ощущаете ли вы периодически озноб или лихорадку?
3340 DATA 34,Наблюдаются ли у вас длительно (шесть и более
     недель) симптомы какой-то болезни?
3350 DATA 35,Опухли ли конечные фаланги ваших пальцев (с
     ногтей при этом сходит защитная пленка и они загибаются)?
3360 DATA 36,Наблюдали ли вы симптомы, возникающие обычно при
     большой физической нагрузке?
3370 DATA 37,Курите ли вы? Для ответа используйте деленное на
     пять число выкуриваемых вами сигарет в день (максимальный
     ответ 5, -5 означает, что вы не курите).
3380 DATA 38,Бывают ли у вас ощущения головокружения?
3390 DATA 39,Бывают ли у вас ощущения "сердцебиения" (сердце
     бьется быстрее или не так ровно, как следует)?
```

```
3400 DATA 40,Не распухли ли ваши лодыжки?
3410 DATA 41,Была ли у вас рвота или сильная тошнота?
3420 DATA 42,Есть ли у вас боль в животе?
3430 DATA 43,Был ли у вас понос?
3440 DATA 44,Удален ли у вас аппендикс?
3450 DATA 45,Есть ли у вас признаки желтухи (например,
         пожелтение белков глаз)?
3460 DATA 46,Чувствуете ли вы какую-то напряженность и
         тревогу?
3470 DATA 47,Трудно ли вы засыпаете? Часто ли вы просыпаетесь
         по ночам?
3480 DATA 48,Наблюдаете ли вы у себя непроизвольные
         подергивания или дрожь?
3490 DATA 49,Страдаете ли вы от запоров, когда стул бывает
         редко или затруднен?
3500 DATA 50,Жалуетесь ли вы на память (существует трудность
         в запоминании отдельных фактов)?
3510 DATA 51,Потеряли ли вы (полностью или частично)
         способность говорить?
3520 DATA 52,Бывает ли у вас кровотечение из области заднего
         прохода?
3530 DATA 53,Укажите ваш пол. Введите 5 для мужчины и -5 для
         женщины. Для анализа безотносительно к полу введите 0.
3540 DATA 54,Есть ли у вас ощущение онемения или боли в
         области шеи?
3550 DATA 55,Не повреждали ли вы голову за последние несколько
         недель (важным может оказаться даже небольшое повреждение)?
3560 DATA 56,Был ли у вас за последнее время ненормальный стул?
3570 DATA 57,Бывает ли у вас сильная отрыжка или выход газов?
3580 DATA 58,Часто ли у вас наблюдается слабость или даже
         обмороки?
3590 DATA 59,Есть ли у вас зуд в какой-то части тела
         (независимо от наличия или отсутствия сыпи)?
3600 DATA 60,Есть ли у вас на коже какая-то сыпь?
3610 DATA 61,Чувствуете ли вы онемелость какой-то части тела
         или ощущение покалывания?
3620 DATA 62,Есть ли у вас излишний вес или он ниже нормы
         (введите 5 при определенном избытке веса или -5 при его
         дефиците и 0, если вес в норме)?
3630 DATA 63,Есть ли у вас боли в области лица или лба?
3640 DATA 64,Есть ли у вас какие-то припухлости под кожей?
3650 DATA 65,Изменился ли цвет вашей мочи?
```

```
3660 DATA 66,Нет ли у вас слишком частого мочеиспускания?
3670 DATA 67,Наблюдаете ли вы боли при мочеиспускании?
3680 DATA 68,Есть ли у вас нарушение зрения - все расплывается,
     двоится, вы видите вспышки (дефекты зрения, которые могут
     быть исправлены очками, не играют роли)?
3690 DATA END,END
```

Чтобы вы могли убедиться, что все сделано верно, а также удостовериться в моем прекрасном здоровье, привожу пример реального диалога с компьютером при использовании данной базы знаний.

RUN
EXPERT

Вопрос:
Есть ли у вас боль в животе — боль в области от низа грудной клетки до паха?
Отвечайте по шкале от —5 (Нет) до +5 (Да): —5
Каменнопочечную болезнь можно исключить.
Вопрос:
Ощущаете ли вы общее болезненное состояние?
Отвечайте по шкале от —5 (Нет) до +5 (Да): —5
Тонзиллит можно исключить,
Грипп можно исключить,
Острый бронхит можно исключить,
Пневмонию можно исключить,
Плеврит можно исключить,
Каменнопочечную болезнь можно исключить,
Корь можно исключить.
Вопрос:
Наблюдается ли у вас какая-либо сыпь на коже?
Отвечайте по шкале от —5 (Нет) до +5 (Да): —5
Тонзиллит можно исключить,
Грипп можно исключить,
Острый бронхит можно исключить,
Пневмонию можно исключить,
Плеврит можно исключить,
Каменнопочечную болезнь можно исключить,
Экзему можно исключить,
Крапивницу можно исключить,
Чесотку можно исключить,
Корь можно исключить,
Ветрянку можно исключить,
Псориаз можно исключить.

Вопрос:
Была ли у вас рвота или сильная тошнота?
Отвечайте по шкале от —5 (Нет) до +5 (Да): —5
Тонзиллит можно исключить
Грипп можно исключить,
Острый бронхит можно исключить,
Пневмонию можно исключить,
Плеврит можно исключить,
Язву двенадцатиперстной кишки можно исключить,
Каменнопочечную болезнь можно исключить,
Экзему можно исключить,
Крапивницу можно исключить,
Чесотку можно исключить,
Корь можно исключить,
Ветрянку можно исключить,
Псориаз можно исключить.
Вопрос:
«Задыхаетесь» ли вы?
Отвечайте по шкале от —5 (Нет) до +5 (Да): —5
Тонзиллит можно исключить,
Грипп можно исключить,
Бронхит можно исключить,
Астму можно исключить,
Эмфизему можно исключить,
Пневмонию можно исключить,
Плеврит можно исключить,
Пневмокониоз можно исключить,
Язву двенадцатиперстной кишки можно исключить,
Каменнопочечную болезнь можно исключить,
Легочную эмболию можно исключить,
Экзему можно исключить,
Крапивницу можно исключить,
Чесотку можно исключить,
Корь можно исключить,
Ветрянку можно исключить,
Псориаз можно исключить.
Вопрос.
Болит ли у вас голова и страдаете ли вы вообще от головных болей?
Отвечайте по шкале от —5 (Нет) до +5 (Да): —5
Тонзиллит можно исключить.
Грипп можно исключить,
...можно исключить, ...

И т. д. и т. п., пока у вас в самом деле не появится головная боль или сильная тошнота.

Тем не менее это выглядит так, и, вероятно, вы заметили, что здесь происходит накопление вариантов выдачи сообщений, причем к имеющемуся списку постоянно добавляются новые элементы, в которых не было обнаружено ничего неподходящего Это весьма похоже на стишок о «Старом дядюшке Томе Кобли и всех остальных». Вы можете избежать неприятностей, поправив программу так, что она будет выдавать только те гипотезы, которые исключались в результате последнего вопроса. Вы в состоянии повлиять и на скорость выполнения программы, если ее измените.

Главное, вы должны запомнить, что вся информация в списках оператора DATA может быть получена гораздо быстрее, если ее разместить блоками. Тогда потребуется полностью пересчитывать значения RULEVALUE после каждого вопроса. Нужно только скорректировать этот массив для тех гипотез, которые были изменены последним вопросом.

Но, по крайней мере, программа достаточна понятна, что делает ее удобной отправной точкой для всех, желающих почувствовать вкус к приключениям.

Глава 11. ВАВИЛОНСКАЯ БАШНЯ

Когда-то давным-давно за столом сидела группа ученых-компьютерщиков, погруженных в глубокое раздумье. В перерывах между размышлениями они подсчитывали, сколько у них всего транзисторов. «А не кажется ли вам, — сказал один из них, — что если мы соединим все эти транзисторы вместе, то у нас получится компьютер» И все согласились, прихлебывая пиво, с таким утверждением. Тем же вечером, чуть позже, они достали свои паяльники и принялись за работу.

Время шло, и машина, которую они строили, становилась все больше и больше, и, наконец, стало ясно, что она и в самом деле сможет выполнить любую поставленную перед ней задачу. Не было в мире более мощного компьютера, и они мечтали о дне, когда они включат его и за считанные секунды получат исчерпывающие ответы на все проблемы Вселенной

По мере того как машина «росла», огромные армии программистов были набраны для написания программ, чтобы управлять этим новым монстром, и каждая из программ записывалась секретными знаками на языке машинных кодов, потому что в то время существовал единственный язык программирования.

Близился День великого включения, и весть о новом огромном компьютере разнеслась по далям и весям, вызывая везде серьезные опасения людей, наслышанных об ужасной познавательной силе, которая будет скоро выпущена на свободу.

В ночь перед самым завершением работ по созданию машины та же группа компьютерщиков снова собралась за столом, и, потягивая пиво, стала спорить: какая из программ должна быть удостоена чести называться первой программой, написанной для новой машины.

«Вы знаете, — сказал один из них глубокомысленно, — не имеет значения, какая программа первая». На что все остальные с важным видом кивнули головами. «Важен язык, на котором написана программа». Все продолжали кивать головами, но начали соображать, какие же другие языки кроме машинного кода могут существовать.

Но тот же человек продолжал, что машинный код не только трудно писать, но и сложно понимать. Он ориентирован на машину, а не на решаемую проблему, и обладает величайшим из всех недостатков — неудовлетворителен с академической точки зрения.

Тогда же ночью каждый из этих ученых-компьютерщиков пошел домой и разработал собственный язык высокого уровня вместе с транслятором к нему для перевода в коды машины. А когда наступил новый день, они собрались у огромной машины и решили, что перед тем как включить ее, нужно решить раз и навсегда, какой из языков самый лучший для использования. Будучи к тому же джентльменами, они решили уладить эту проблему путем дискуссии.

Насколько известно, они до сих пор дискутируют по этой проблеме, а машина так и не включена, потому что истинная проблема состояла в том, что ни один из них не понимал языков, разработанных другими.

Если даже эта история не является на сто процентов историческим фактом, она очень правдива. Поскольку когда единственным языком был машинный код, или АССЕМБЛЕР (его проще понять, но в сущности это то же самое), каждый программист понимал в той или иной степени, что делал его коллега. А сейчас существует так много языков программирования, что практически невозможно знать их все. Так же трудно понять идею, выраженную на незнакомом языке.

Вот почему все примеры в этой книге даны на БЕЙСИКе. Это язык, доступный многим. Но главный вопрос тем не менее остался без ответа: лучший ли это язык для применения? И не стоит ли попробовать какой-нибудь другой? Интуитивно вы можете считать, что существуют серьезные аргументы в пользу применения, скажем, языка LISP, так как на нем написаны многие экспертные системы. А может лучше ПРОЛОГ — более поздняя версия LISP?

Чтобы получить ответ на этот вопрос, следует рассмотреть возможности языка программирования. В первую очередь надо отметить, что он фактически не сможет сделать ничего такого, чего нельзя было бы сделать в машинном коде. Хоть это и очевидно, но все равно стоит отметить, что программа на любом языке перед использованием на машине переводится в машинный код и именно машинный код характеризует ЭВМ, работающую с программой.

Если данные не удается свести к машинному коду, они не рассматриваются, т. е., используя машинный код, можно сделать все, на что способна машина, и ни на каком другом языке нельзя сделать большего.

Возникает также вопрос: не ограничивает ли тот или иной язык возможную сферу деятельности? Ответ будет утвердительным. Возьмите, например, БЕЙСИК, сделав вид, что не существует таких вещей, как PEEK и POKE[1]. Теперь представьте, что по какой-то причине вы хотите заглянуть в байт 3096 RAM; посмотреть, что там есть, и изменить только один бит в нем. Это невозможно — разве каким-нибудь сверхъестественным способом, поскольку БЕЙСИК (без PEEK или POKE) не позволяет получить абсолютно точный адрес и «настраивать отдельные биты.

[1] Операторы языка БЕЙСИК, позволяющие записывать в ячейку памяти и считывать из нее двоичный код. — *Прим. пер.*

На самом деле в этом примере нет ничего необычного. Многие языки высокого уровня не имеют средств для формирования точного адреса, хотя все же существуют случаи, когда знание точного адреса необходимо.

Итак, никакой язык не в состоянии сделать больше, чем машинный код, меньше — может. Для чего же тогды нужны другие языки? Во-первых, они проще машинного кода, а во-вторых, они иногда содействуют, хотя и не всегда, процессу машинного мышления.

Первый момент очевиден. БЕЙСИК легче, чем машинный код, изучить и использовать. Это же справедливо и по отношению к большинству языков высокого уровня. Но в плане содействия процессу мышления все не так очевидно. И, даже допуская это, нужно помнить и об обратной стороне медали.

Рассмотрим, например, цикл FOR...NEXT на БЕЙСИКе. Его организация, конечно, в чем-то способствует логическому мышлению. Вы можете мысленно себе его представить, записать на бумаге, отобразить на экране дисплея. После этого лучше не думать о нем и сосредоточиться на том, что находится в теле цикла. Здесь-то и появляется обратная сторона медали — мы заранее предположили, что в БЕЙСИКе непременно будем пользоваться циклами FOR. На самом деле цикл FOR... ...NEXT так сильно воздействует на процесс мышления, что, когда мы сталкиваемся с проблемой программирования, соблазн постараться решить наши проблемы с помощью оптимального расположения этого цикла столь велик, что мы забываем задаться вопросом, а есть ли в самом деле смысл использовать циклы FOR...NEXT в данном случае? И даже если от них мало толку, мы все равно скорее всего будем иметь дело именно с ними.

Чтобы наши рассуждения стали ясиее, подумайте, чем отличается компьютер от той проблемы, которую ему надо решить?

Машинный код ориентирован исключительно на компьютер, и если вам просто нужно управлять ЭВМ, то он позволяет наилучшим образом создать короткую, быстро действующую программу.

Языки высокого уровня в той или иной степени ориентированы на проблему. Они также управляют машиной, но позволяют вам сосредоточиться и на решении проблемы. Рассмотрим, например, БЕЙСИК. Если вы хотите работать с числовымн матрицами, то циклы

FOR...NEXT дадут вам идеальную возможность доступа к различным элементам этих матриц, так необходимую для работы. Но не нужно думать, что раз язык проблемно ориентирован, он прекрасно ориентирован на любые конкретные проблемы. Существуют разные проблемы, которые и решаются различными способами, о чем говорят названия некоторых языков программирования.

Например, COBOL (Common Business Oriented Language) — акроним выражения "Язык, ориентированный на решение общих проблем бизнеса"; ALGOL (Algorithmic Language) — акроним выражения "Алгоритмический язык", нацелен на решение проблем, допускающих четкое алгоритмическое описание (алгоритм — это рецепт решения проблемы в основном в области логики и математики); FORTRAN (Formula Translator) — акроним выражения "Транслятор формул" — язык решения математических задач. А БЕЙСИК (BASIC)? Это Beginners All-Purpose Symbolic Instruction Code, что расшифровывается, как „язык символьных команд общего назначения для начинающих, которые не очень хорошо умеют программировать".

Но вернемся к экспертным системам и зададим вопрос: существует ли язык для решения экспертных задач?

Очевидно, чтобы ответить на этот вопрос, надо знать, что за проблемы возникают, когда вы строите экспертную систему. А сделать это трудно, потому что термин «экспертные системы» объединяет множество проблем.

Тем не менее отметим несколько моментов. Для начала подумайте о рассмотренной нами экспертной системе с несколькими узлами. В БЕЙСИКе узлы связаны друг с другом специальными метками, записанными в виде символьных (строковых) переменных, которые потом можно отыскать среди других переменных; совпадающие строки и определяют наличие связи. Но это не очень хорошо потому, что эти переменные представляют собой данные, а те, вероятно, будут потеряны, как только вы выключите машину, если у вас нет специальной программы для их сохранения на диске.

С целью устранения рассмотренных недостатков был разработан язык ЛИСП (LISP) — List Processing Language, что означает "Язык обработки списков", так как мы имеем дело с набором списков.

Допустим, у нас теперь есть два узла: *узел 1* и *узел 2* со своими входами и выходами (рис. 11.1). Определим

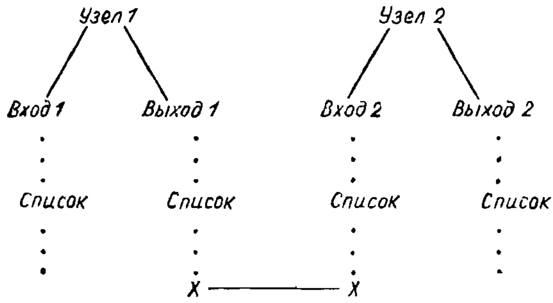


Рис. 11.1. Структуры

список для *узла 2.* Он содержит все входы и выходы этого узла. Вычислим два подсписка в *узле 2:* вход и выход, чтобы показать, что каждый из них представляет. Сделаем то же и для *узла 1.* Если он имеет *выход 1*, содержащий X, который совпадает с тем же элементом подсписка *вход 2* для *узла 2*, то это можно выразить в программе, написанной на языке ЛИСП. Определим, например, структуры, как показано на рис. 11.1.

После этого нетрудно написать программу для формирования запроса типа: является ли любой из входов *узла 2* выходом *узла 1.* Безусловно, рассмотренный пример не сложен. Истинная же ценность ЛИСПа в том, что он позволяет определить эти структуры в их взаимосвязи, причем степень сложности при этом не ограничивается. Вы можете заниматься также дальнейшим усложнением структур, если потребуется. При реализации таких возможностей на БЕЙСИКе разболелась голова не у одного программиста средней руки.

Следующий момент. Все указанные системы использовали движение по цепочке вперед. Они «собирали» фрагменты данных и потом «шли» вперед по программе, чтобы понять, что с этими данными можно сделать. Допустим, мы хотели бы иметь программу, обеспечивающую движение по цепочке назад. В указанном примере различие будет следующим.

При движении по цепочке вперед мы задаем программе, например, *вход 1 узла 1*, который создает выход, передаваемый на *узел 2* как его вход, поэтому *узел 2* может сформировать выход

При движении по цепочке назад программа, просматривая *узел 2*, пытается предугадать, может ли он создать какой-то определенный выход, и смотрит в противоположном направлении, чтобы понять, какие входы ему нужны. При этом программа «видит», что для создания некоторых из этих входов требуется *узел 1*. Тогда она возвращается к *узлу 1* и запрашивает у него нужную информацию. Это нелегко сделать на БЕЙСИКе главным образом потому, что в большинстве его диалектов не предусматривается возможность использовать рекурсивную способность процедуры, т. е. способность вызывать самою себя.

Самым обычным примером рекурсии (если вы не знаете, что это такое) является вычисление факториала, например $3! = 3 \cdot 2 \cdot 1 = 6$. Если бы вы имели возможность применить рекурсию, то могли бы написать

$$\text{FAC}(\text{N}) = \text{N} * \text{FAC}(\text{N} - 1), \text{ если } \text{N} > 1,$$

$$\text{FAC}(\text{N}) = 1, \text{ если } \text{N} \leqslant 1.$$

Например,

$$\text{FAC}(3) = 3 * \text{FAC}(2) = 3 * 2 * \text{FAC}(1) = 3 * 2 * 1 = 6.$$

Не всегда легко приводить примеры простейших случаев, но представим, что у нас есть рекурсивная процедура под названием NODE(N). Она пытается, скажем, оценить узел (N) и поэтому проверяет входы этого узла, а параллельно с этим выясняет, не являются ли какие-то из них выходами узла (N—1). Если да, то вызывает NODE(N—1). Затем процедура проверяет узел (N—2) и т. д. до тех пор, пока не вернемся к началу цепочки.

Написание процедуры NODE (N) требует некоторых рассуждений, но затем ее можно использовать в произвольных по сложности структурах, о которых мы говорили раньше. В большинстве диалектов языка БЕЙСИК не предусматривается рекурсия, но это не исключает возможности применения рекурсивных программ. На БЕЙСИКе можно написать программу, действующую аналогичным образом, но это будет непросто.

Использование ЛИСПа или ПРОЛОГа облегчает программирование подобных задач. Преимуществом этих языков является то, что они были созданы, чтобы оперировать с очень сложными символьными структурами, которые имеют мало общего с обычной математикой. Применительно к экспертной системе особенность состоит в том, что когда мы заставляем нашу систему действовать так, как это делал бы человек-эксперт, она вынуждена работать с большим количеством довольно сложных, нечисловых структур.

Указание на нечисловые или нематематические (в обычном смысле) структуры дает возможность посмотреть на другую сторону медали. Дело в том, что если вы используете обычную математику, то программы, написанные на этих языках, получаются очень громоздкими.

Возьмем, например простой оператор присваивания на БЕЙСИКе $X = 2 + 2$ и рассмотрим его вариант на языке ЛИСП. Он будет иметь вид

(SETQ X ADD (2,2)).

Это выглядит не очень сложно. Но представьте себе, как будет выглядеть сложное арифметическое выражение.

Все дело в том, что ЛИСП создан для содействия решению определенного типа проблем, не связанных с арифметическими выражениями. Некоторые пользователи ЛИСПа, возможно, захотят сделать ряд математических расчетов, поэтому такая возможность и была предусмотрена, но реализована она теми же средствами, что и другие возможности. В результате обычные арифметические вычисления выглядят в записи на ЛИСПе столь странным образом.

В БЕЙСИКе же все наоборот. Начинающим обычно чаще приходится выполнять математические расчеты, поэтому этот язык больше нацелен на решение расчетных задач. Конечно, вы легко обнаружите, что язык, разработанный для подобных целей, обладает недостатками в других областях.

Рассмотрим теперь ПРОЛОГ (Prolog). Название этого языка — Programming in Logic — означает «логическое программирование». С его помощью очень легко писать программы, используя то, что специалисты по логике называют «исчислением предикатов первого порядка», хотя совсем не обязательно знать, как это называют специалисты. На мой взгляд, ПРОЛОГом намного легче пользоваться, чем, скажем, языком ЛИСП, и у него много сторонников в сфере экспертных систем. Чтобы вам стало все понятно, приведем следующий пример:

смертный (X): — человек (X),
человек (Сократ).

Первая строка означает, что X является смертным, если верно, что X — человек. Вторая — то, что Сократ — человек. Первая строка выражает правило, вторая — факт. При работе этой небольшой программы можно задать системе вопрос: является ли Сократ смертным, и система ответит: «Да». Для этого вы должны задать вопрос на ПРОЛОГе (? —). На дисплее это может выглядеть следующим образом:

? — смертный (Сократ).
Да.

Другими словами, вы спросили систему: существует ли такой Сократ, который смертен? Система замечает, что в первой строке она может сопоставить, что X эквивалентен Сократу и то, что это будет верно, если это тот X, который обладал признаком человека. Во второй строке она действительно определяет, что есть такой X, являющийся Сократом, про которого верно то, что X — человек, и тогда она вправе дать ответ: «Да», — т. е. при условии, что X эквивалентен Сократу, верно, что X смертен.

Это простой пример, но его достаточно, чтобы показать целесообразность использования языка ПРОЛОГ для экспертных систем, потому что вы можете задать большое количество правил и фактов. Система, построенная с помощью ПРОЛОГа, самостоятельно проделает тяжелую работу, пробираясь через правила и факты в их логической последовательности. Однако в системе на ПРОЛОГе автоматически требуется применение механизма движения по цепочке назад, при котором устанавливается конечная цель. Система автоматически движется по цепочке назад от этой цели, чтобы попытаться установить ее истинность или, наоборот, ложность. Это было бы хорошо, если бы именно к этому вы стремились. Но когда в вашей системе используется этот механизм или, того хуже, вы вычисляете «значения правил» по методу, описанному в этой книге, у вас могут возникнуть трудности, так как ПРОЛОГ в подобной ситуации оказывается неидеальным языком, если вы попробуете отойти от свойственного ему механизма движения по цепочке назад.

Это служит хорошим примером того, как язык оказывает воздействие на ход рассуждений. Вы наверняка сможете определить, что та или иная система написана на ПРОЛОГе, увидев, что в ней используется движение по цепочке назад. Одной из главных причин, почему я разработал метод, основанный на вычислении «значений правил», было то, что он легко реализуется на БЕЙСИКе.

Сегодня также отмечается тенденция к созданию экспертных систем с помощью другого языка — Си.

Главное, что нужно отметить, говоря о языке Си, — он очень похож на немного «приглаженную» разновидность языка АССЕМБЛЕР, другими словами, напоминает машинный код. Его привлекательность в том и состоит, что он накладывает очень мало ограничений на реализуемые структуры, а то и совсем не накладывает их.

На Си достаточно трудно составлять программы, но он позволяет вам программировать все, что угодно, и не накладывает ограничений на выбор объекта программирования. Конечно, в идеале нам нужен один большой язык, который способен на «все» в любых ситуациях и на котором не очень трудно программировать. Но мы вряд ли будем иметь такой язык просто потому, что никто не знает, как он должен выглядеть.

Даже мой родной английский язык не универсален — достаточно обратить внимание на наличие большого числа специальных «диалектов», развившихся в течение многих лет, которые расширили рамки английского языка, что он смог проникнуть в новые области, например в язык математиков и врачей. И если мы захотим создать язык, идеально подходящий, например, для медицинской диагностики, то нам понадобятся английский (для общения с пациентами), «медицинский» (для общения с медиками, возможно, подойдет латынь) и «математический» (для осуществления расчетов). Вполне можно утверждать, что этих языков больше, чем языков, которыми владеют многие практикующие врачи.

Но языковая проблема не такая трудная, как это может показаться, по той простой причине, что большинство людей, использующих компьютеры, имеют ограниченный доступ к языкам. Это разрешает большинство споров о том, какой язык следует применять. Зачастую лучшим языком оказывается тот, которым вы свободно владеете. Значительное количество исследований в области искусственного интеллекта проводится с привлечением языков ЛИСП или ПРОЛОГ. Во всяком случае все профессионалы-специалисты по экспертным системам обычно используют именно их. Это становится привычкой, а привычка — вторая натура. У других такая привычка: применять язык БЕЙСИК.

Несомненно, что язык так или иначе оказывает влияние на ход ваших рассуждений, и программист, использующий БЕЙСИК, наверняка будет обращаться с экспертной системой по-другому, чем тот, кто работает на языках ЛИСП или ПРОЛОГ.

В конечном итоге ценность каждого подхода определяется качеством конечного результата, и в крайнем случае вы всегда можете вернуться к машинному коду. Единственная опасность тогда заключается лишь в том, что вы можете состариться, так и не закончив свою программу.

Глава 12. ТЕХНИЧЕСКОЕ ПРИЛОЖЕНИЕ К МАТЕРИАЛУ ОТДЕЛЬНЫХ ГЛАВ

Полагая, что эта книга читается как рассказ, цель которого — включить вас в процесс созидания, можно представить вас сидящими и рассуждающими в таком духе: «Прекрасно, я вижу как эта система работает. Я мог бы даже написать теперь собственную экспертную систему, но я не могу вспомнить, как и с помощью каких символов этот Нейлор определял правила или...». Чтобы вспомнить это, достаточно посмотреть в предметный указатель или заглянуть в конспект, который вы сделали. С единственной целью облегчить вашу работу ниже приводится список наиболее важных терминов, которые вам нужно знать.

12.1. СОБЫТИЯ

Под событием можно понимать любую вещь.

Пусть H — событие, заключающееся в том, что данная гипотеза верна. Пусть E — событие, заключающееся в том, что наступило определенное доказательство (свидетельство), которое может или не может подтвердить правильность указанной гипотезы.

$P(H)$ — вероятность того, что событие H истинно.

$P(E)$ — вероятность того, что событие E произошло.

Вероятности являются числами в диапазоне от 0 до 1. Если вероятность равна 0, то данное событие не произойдет никогда. Если вероятность равна 1, то данное событие происходит всегда.

$P(\text{не } H) = 1 - P(H)$ — вероятность того, что событие H ложно.

Два события независимы, если вероятность их совместного наступления равна произведению вероятностей наступления каждого события в отдельности.

$P(E_1 \& E_2)$ — совместная вероятность наступления как события E_1, так и события E_2.

E_1 и E_2 независимы, если и только если $P(E_1 \& E_2) = P(E_1)P(E_2)$.

$P(H{:}\ E)$ — условная вероятность наступления события H, определяемая с учетом того, что событие E уже наступило.

Если события H и E независимы, то $P(H{:}\ E) = P(H)$. В общем случае $P(H{:}\ E) = P(H \& E)/P(E)$. Аналогично $P(E{:}H) = P(E \& H)/P(H)$, поэтому $P(H{:}E) = P(E{:}H)P(H)/P(E)$.

12.2.1. Теорема Байеса — $P(H{:}E) = P(E{:}H)P(H)/P(E{:}H)P(H) + P(E{:}\text{не}H)P(\text{не}H))$. Относительно использования вероятностей $P(H{:}E)$ или $P(E{:}H)$: вначале может показаться, что всегда имеет смысл применять вероятность $P(H{:}E)$, ведь мы хотим знать, какова вероятность того, что гипотеза H верна́ при наступлении данного свидетельства, а не наоборот.

Задача заключается в том, что $P(H{:}E)$ порой вовсе не очевидна. Если вы знаете $P(H{:}E)$ в самом начале, то было бы мало толку писать программу для компьютера для ее подсчета. Вероятность же $P(E{:}H)$, напротив, является величиной более очевидной, если учитываются данные по рассматриваемой проблеме.

Вы можете задать вопрос: «Если H истинно, то какова вероятность, что подтверждающее его свидетельство может наступить?» А для $P(E{:}\text{не}H)$ поинтересоваться: «Если H ложно, то какова вероятность, что подтверждающее его свидетельство может наступить?» Эти два вопроса дают возможность вычислить значение $P(H{:}E)$ в программе, последовательно улучшая имеющуюся оценку после получения каждого нового подтверждающего свидетельства.

Для двух или более событий E_1 и E_2:

если E_1 и E_2 независимы, то $P(E_1 \& E_2:H) = P(E_1:H)P(E_2:H)$.

Заметим: ошибочным является утверждение о том, что $P(H:E_1 \& E_2) = P(H:E_1)P(H:E_2)$.

Если E является событием, заключающимся в том, что «все E_i наступили» и, кроме того, они все не зависят друг от друга, мы можем вычислить

$$P(E:H) = P(E_1:H)P(E_2:H) \dots P(E_n:H),$$

$$P(\text{не } E:H) = P(\text{не } E_1:H)P(\text{не } E_2:H) \dots P(\text{не } E_n:H).$$

12.2.2. Априорные и апостериорные вероятности. Допустим, мы имеем гипотезу H и некое подтверждающее или не подтверждающее ее свидетельство, которое назовем E, тогда:

$P(H)$ — априорная вероятность истинности гипотезы H. Это вероятность наступления H без учета факта существования E;

$P(H:E)$ или $P(H:\text{не}E)$ — апостериорная вероятность гипотезы H, т.е. вероятность H при условии, что нам известен факт существования E.

Согласно теореме Байеса

$$P(H:E) = P(E:H)P(H)/(P(E:H)P(H) + P(E:\text{не } H)P(\text{не } H))$$

или, если мы нашли, что событие E не наступило,

$$P(H:\text{не } E) = P(\text{не } E:H)P(H)/(P(\text{не } E:H)P(H) + P(\text{не } E:\text{не } H)P(\text{не } H)).$$

Примем, что для некоторого события H существует большое число отдельных свидетельств, подтверждающих или не подтверждающих его, которые последовательно выявляются в процессе функционирования. Назовем их соответственно $E_1, \dots, E_n$. Если бы все они были выявлены одновременно и не зависели друг от друга, то мы могли бы подсчитать $P(E:H)$ как произведение отдельных вероятностей $P(E_i:H)$, а затем вычислить $P(H:E)$, где E — событие, состоящее в том, что «произошло осуществление всех E_i». Аналогично мы определили бы $P(\text{не}E:H)$ как произведение всех $P(\text{не}E_i:H)$.

Вместо этого мы, возможно, посчитаем более удобным работать поэтапно, суммируя отдельные свидетельства и их влияние на условную вероятность по мере наступления отдельных E_i. Это можно сделать, используя априорные и апостериорные вероятности, следующим образом.

1. $P(H)$ — априорная вероятность события H.

2. Для данного свидетельства E_i запишем $P(E_i{:}H)$ и $P(E_i{:}\text{не}H)$.

3. С учетом теоремы Байеса подсчитаем $P(H{:}E_i)$ или $P(H{:}\text{не}E_i)$ в зависимости от исхода E_i, т.е. вычислим апостериорную вероятность события H.

4. Теперь можно не обращать внимания на все наступившие E_i и переобозначить текущую апостериорную вероятность события H как новую априорную вероятность H. Итак, пусть $P(H)$ равна $P(H{:}E_i)$ или $P(H{:}\text{не}E_i)$ в зависимости от значения E_i.

5. Затем выберем новое свидетельство E_i для рассмотрения и перейдем к п. 1.

12.2.3. Шансы в пользу наступления какого-то события можно вычислить, зная вероятность этого события, т. е.

$$O(E) = P(E)/(1 - P(E))$$
$$\text{и } P(E) = O(E)/(1 + O(E)).$$

12.2.4. Аппроксимации. Рассмотрим выражения $P(A \text{ и } B) = \min(P(A), P(B))$ и $P(A \text{ или } B) = \max(P(A), P(B))$.

Эти результаты, строго говоря, не совсем верны. Степень их ошибочности зависит от степени независимости событий A и B. Если даже у вас нет информации об их независимости, эти соотношения могут быть полезными. Заметим также, что

$$\min(x,y) = x, \text{ если } x < y,$$
$$\min(x,y) = y, \text{ если } y < x;$$
$$\max(x,y) = x, \text{ если } x > y;$$
$$\max(x,y) = y, \text{ если } y > x.$$

12.2.5. Комбинаторика. Пусть дано n событий, из которых мы выбираем x событий, тогда существует $\binom{n}{x}$ различных вариантов такого выбора:

$$\binom{n}{x} = n!/(n-x)!\,x!, \text{ где } n! = n(n-1)(n-2)\ldots(n-(n-1)),$$

например $4! = 4\cdot 3\cdot 2\cdot 1 = 24$.

Если имеются n возможных свидетельств и любое из них может произойти, то существуют $\sum_{x=0}^{x=n} \binom{n}{x}$ возможных комбинаций.

12.2.6. Описательная статистика. Такая статистика, как следует уже из названия, необходима для описания или суммирования (обобщения) основных свойств (особенностей) данных, имеющихся в нашем распоряжении.

Математическое ожидание (часто используется и другой термин — среднее значение)

$$m = \sum_{i=1}^{i=n} \frac{x_i}{n},$$

дисперсия

$$v = \sum_{i=1}^{i=n} \frac{(x_i - m)^2}{n},$$

стандартное отклонение

$$sd = \mathrm{SQR}(v).$$

12.2.7. Нормальное распределение. Большинство приведенных в этой книге методов основаны на непараметрической статистике. При этом не учитываются какие-либо предположения относительно поведения рассматриваемых переменных.

Параметрическая статистика, напротив, предполагает наличие некой математической модели поведения рассматриваемых переменных. Одно такое «семейство» параметрической статистики составляет группа статистики, предполагающая, что переменные происходят главным образом из популяции, описываемой функцией нормального распределения. Нормальное распределение удобно тем, что имеет разработанную теорию и легко обрабатывается при проведении исследований.

Однако прежде, чем воспользоваться этой теорией, необходимо убедиться, что ваши данные действительно подчиняются нормальному закону распределения. На практике многие переменные (в противоположность теории) такому закону не подчиняются вопреки желанию статистиков. Вы должны упорно сопротивляться искушению применить нормальное распределение к тем переменным, которые на деле не распределены по нормальному закону.

12.2.8. Дискретные и непрерывные переменные. Дискретные переменные — это такие переменные, которые принимают лишь определенные фиксированные значения, например ответ типа Да/Нет является примером такой переменной. Непрерывные же переменные могут принимать любые числовые значения, например уровень осадков относится к такому типу переменных.

Непрерывные переменные удается преобразовать в дискретные путем квантования их по уровню на группы — выше 25 и ниже 25 мм.

Вы, вероятно, потерпите неудачу, если будете использовать одни и те же методы вычислений как для дискретных, так и для непрерывных переменных. Нужно сначала определить тип переменных, а затем выбрать соответствующий метод. В общем случае иметь дело с дискретными переменными гораздо проще, чем с непрерывными. Это происходит потому, что методы работы с непрерывными переменными обычно основаны на параметрической статистике, но вопрос выбора подходящих параметрических семейств относится к разряду довольно сложных.

12.3. ПОВЕРХНОСТИ

Общее уравнение поверхности имеет вид

$$y = \sum_{i=0}^{i=n} b_i x_i,$$

где b_i — константы; x_i — переменные.

Поверхность стола, например, является двухмерной ($n=2$), а x_1 и x_2 — прямоугольными (ортогональными) координатами этой поверхности. Допустим, x_1 и x_2 не зависят друг от друга или некоррелированы. В общем случае переменных может быть столько, сколько нужно. Они могут быть независимы друг от друга. Их можно использовать для описания чего-нибудь вообще не существующего в реальном трехмерном пространстве. С их помощью удается описать, например, конкретные модели поведения, болезни или чего-то еще. В этом случае понятие поверхности является удобным с точки зрения математического описания.

12.4. ПРОБЛЕМА РАЗДЕЛЕНИЯ

Допустим, мы имеем n категорий объектов, описанных замерами на определенном наборе переменных. Затем нам предъявляется еще один объект и набор замеров для этого объекта на том же наборе переменных. Мы должны определить, к какой из n категорий принадлежит объект. В этом и состоит проблема разделения (классификации).

Вообще говоря, поставленная проблема решается только в том случае, если указанные категории линейно сепарабельны. Это значит, что между каждой из категорий можно поместить разделяющую поверхность или ряд поверхностей. Поверхность определяется в терминах, которые легко измерить для данных объектов.

Мы можем решить, что категории являются взаимоисключающими, т. е. объект в состоянии попасть только в одну из них. В этом случае мы поместим его в наиболее вероятную категорию. Другими словами, поместим его в такую категорию, для которой значение $P(H{:}E)$ максимально. Здесь гипотеза H определяет одну категорию, а событие E — все свидетельства, относящие объект к «членам» этой категории.

Заметим, что математически $P(H{:}E)$ является уравнением поверхности. Это та поверхность, которая «указывает» в направлении конкретной гипотезы H. Зная $P(H{:}E)$, определяют константы b_i для уравнения поверхности, а на основании свидетельства E по каждой переменной вычисляют значения x_i. Мы не должны однако, использовать значения вероятностей в явном виде.

Если же категории не являются взаимоисключающими, то мы вправе поместить наш объект в несколько категорий. В этом случае мы не просто выбираем наиболее вероятную категорию, а имеем дело с пороговым критерием. Наш объект помещается (относится) в те категории, для которых он превысил значение порога данного критерия.

Если разделяющая поверхность имеет вид

$$y = \sum_{i=0}^{i=n} b_i x_i,$$

то мы можем отнести объект к какой-либо категории, если $y > y_c$.

12.5. ОБУЧАЮЩИЙ АЛГОРИТМ

1. Проведите n наблюдений x на объекте, который должен быть классифицирован.

2. Вычислите значение

$$y = \sum_{i=0}^{i=n} b_i x_i$$

для каждой возможной категории. Вначале $b_{ik}=0$ для всех i и k.

3. Найдите такую категорию k, для которой y_k имеет наибольшее значение.

4. Если объект принадлежит к категории k, то такая классификация корректна. Ничего не меняя, перейдите к п. 6.

5. Если указанная попытка классифицировать некорректна, то проведите следующую модификацию:

$b_{ik}=b_{ik}+x_i$ для той категории k, куда согласно классификации должен был попасть объект. Делайте то же самое для всех b_{ik} в k;

$b_{ik}=b_{ik}-x_i$ для всех категорий k, в которые данный объект не мог бы попасть и для которых y_k больше, чем y_k, соответствующее корректной классификации. Проделайте то же для всех b_{ik} в этих некорректных категориях.

6. Сделайте еще одно наблюдение и переходите к п. 1.

Чтобы этот алгоритм заработал, требуется набор объектов для обучения с заранее известными классификационными категориями. Как только он станет нормально работать, его можно использовать на других объектах, для которых не известна корректная классификация.

12.6. ПАРАЛЛЕЛЬНЫЕ И ПОСЛЕДОВАТЕЛЬНЫЕ ПРОЦЕДУРЫ

При параллельной процедуре используется сразу вся возможная информация и проводится одно окончательное вычисление на ее основе, т. е.

$$y_k = \sum_{i=0}^{i=n} b_{ik} x_i \text{ (для всех категорий } k\text{)}.$$

В случае последовательной процедуры осуществляется переход от одной переменной к другой с учетом информации, полученной на каждом шаге, последовательно по всем i для любых категорий k, т. е. $y_{ik} = y_{i-1,k} + b_{ik} x_i$.

Последовательная процедура в конечном счете должна, как только собрана вся информация, дать тот же самый результат, что и параллельная процедура. Если же для получения какого-то вывода постоянно требуется вся информация, то разница между двумя этими методами чисто символическая.

Если какой-либо вывод может быть получен на основе использования лишь части всей возможной информации, то целесообразнее применить последовательную процедуру, поскольку результат удастся получить гораздо быстрее.

12.7. МИНИМАЛЬНЫЕ И МАКСИМАЛЬНЫЕ ЗНАЧЕНИЯ

Если переменные x_i, которые должны вводиться в систему, имеют соответствующие минимальные и максимальные значения, то эта информация позволит процедуре принятия решения быстрее сформировать необходимое заключение.

Допустим, все $\max(x_i)$ всегда приводят к определенной категории при классификации, тогда как все $\min(x_i)$ оспаривают подобную классификацию. Тогда для неизвестных еще нам переменных x_i вычисляются два возможных исхода, основанных на $\max(x_i)$ и $\min(x_i)$. Если ни один из них не опровергнут наиболее вероятного текущего исхода, формируемого системой, то этот наиболее вероятный возможный исход и будет взят в качестве корректного возможного исхода. Поэтому такие значения x_i фактически оказываются ненужными.

Если для классификации требуется использовать все значения x_i, то любые различия в стратегиях поиска решений оказываются чисто умозрительными. Если же необходимы не все значения x_i, то встает вопрос о применении наиболее эффективной возможной стратегии поиска решения.

12.8.1. Стратегия поиска, основанная на наличии цели. Эта стратегия опирается на выбор некоторой категории, ее фиксации и проверки соответствующих x_i до тех пор, пока можно решить, является ли данная категория корректной или нет. Поступая таким образом, система продвигается вперед, проверяя следующую категорию, и т. д.

12.8.2. Стратегия поиска, основанная на наличии данных. Суть стратегии заключается в выборе некоторого x_i, которое, исходя из определенных предпосылок, выглядит как некая полезная переменная. Имея значение этой переменной, система оценивает, какую пользу можно извлечь из полученной информации. Затем она выбирает другое значение x_i. В процессе такого перебора появляются выводы относительно различных категорий.

12.8.3. Выбор последующих переменных. Независимо от того, какая используется стратегия: основанная на наличии цели или на наличии данных — существует некоторая свобода, с учетом которой должно быть проверено следующее значение x_i. Это проблема из категории тех, как выбрать «хороший» вопрос из серии возможных вопросов, причем трудно точно определить, что такое «хорошо». Однако из широкого круга вопросов мы можем рассмотреть: 1) на сколько категорий это оказывает влияние? 2) в каких пределах это влияет на данные категории?

Во-первых, для каждой неизвестной на текущий момент переменной x_i мы вправе вычислить разность [y_{ik} (используя $\max(x_i)$) — y_{ik} используя $\min(x_i)$], чтобы определить ее максимально возможное изменение. Во-вторых, с помощью теоремы Байеса и соответствующих вероятностей легко подсчитать значение [$P(H{:}E)$ — —$P(H{:}\text{не}E)$], где E — наблюдение x_i, чтобы выяснить максимальное изменение переменной, вызванное каждым новым свидетельством.

Чтобы сделать все это для каждой переменной, приходится, прежде чем задавать какой-то вопрос, проводить довольно много вычислений (правда, это делает компьютер) по сравнению с тем, когда просто задают следующий вопрос по списку. Преимущество такого подхода заключается в том, что для пользователя он достаточно прост.

Чтобы ускорить время обработки, требуется вычислить:

а) $\max(\max(x_i) - \min(x_i))$ для того, чтобы судить о степени важности вопроса. Препятствием к использованию этого метода является то, что значения x_i могут изменяться в широком диапазоне, но не быть важными в каком-то отношении;

б) $\max(\mathrm{var}(b_{ik}))$, где var — дисперсия, для того чтобы судить, до какой степени та или иная переменная используется при установлении категорий. Препятствием к применению этого метода является то, что переменная, которая возможно будет часто использоваться, имеет небольшую разницу $(\max(x_i) - \min(x_i))$ и большой диапазон изменения b_{ik}, что и приводит к такому результату.

12.9. ПРОМЕЖУТОЧНЫЕ ВЫВОДЫ

Эти выво́ды не всегда, строго говоря, необходимы, но они могут быть полезны для «очеловечивания» системы. Промежуточный вывод иногда используется в качестве входной переменной для другой ступени экспертной системы. Следует заметить, что наличие таких выводов существенно усложняет процесс написания системы.

Однако есть один момент, который важно иметь в виду, рассматривая промежуточные выводы. Все зависит от такого каверзного вопроса, как независимость.

В большинстве статистических методов предполагается независимость различных свидетельств E_i, и часто такое предположение ничем не подтверждается.

Если существует n факторов (свидетельств), подтверждающих справедливость гипотезы H, и они коррелированы друг с другом, а вычисления проделаны в предположении, что такой корреляции не существует, то гипотеза H получит бо́льшую поддержку, чем этот факт сам по себе того заслуживает. Введение промежуточных выводов в процессе рассуждений может помочь исключить подобный эффект.

Рассмотрим события E_1 и E_2: они оба представлены в подтверждение гипотезы H_1. Пусть H_1 — промежуточный вывод, действующий как свидетельство для дальнейшего вывода H_2. Допустим также, что E_1 и E_2 коррелированы в какой-то степени между собой. Тогда H_1 получит бо́льшую поддержку, чем нужно. Однако фактически ее все еще достаточно для того, чтобы считать гипотезу H_1 истинной, следовательно, реального ущерба это пока не принесло.

Но без H_1 ошибка в вычислении E_1 и E_2 перешла бы в H_2, постепенно становясь все более и более серьезной по мере того, как добавляются возможно коррелированные E_i в процессе вычислений.

Присутствие H_1 и других промежуточных выводов позволяет избавиться от накопленных ошибок. Для H_2 следует начать новую последовательность вычислений, основанную на меньшем числе свидетельств, уменьшая тем самым риск формирования новой неопределенности.

12.9.1. Система, комментирующая действия. Промежуточные выводы могут быть использованы для объяснений о текущем состоянии системы. В общем случае такая система комментариев в состоянии сообщить, на какой стадии процесса рассуждений находится система, а часто и как она пришла к этому. Проблема заключается в том, что подобный подход заставляет компьютер работать в постоянной готовности дать ответ, а это условие трудно обеспечить. Не исключено даже снижение эффективности системы.

Например, какого рода объяснения можно считать достаточными, если компьютер спрашивает о переменной x_i, потому что он вычислил, что данное x_i повлияло бы иа значение $P(H:E)$ больше, чем любое другое x_i?

Большинство стратегий выбора «хорошего» x_i, предложенных в этой книге, основаны на нахождении такого x_i, которое выглядит в каком-то отношении важным с точки зрения математики. Но наивный пользователь, возможно, будет чувствовать себя счастливее, если появятся объяснения, содержащие, например, листинг с категориями, подвергнутыми наибольшим воздействиям со стороны x_i, вне зависимости от выбора конкретной переменной.

12.10. ЛИНЕЙНАЯ ИНТЕРПОЛЯЦИЯ ОТКЛИКОВ

Допустим, пользователь не очень уверен в ответе на какой-то вопрос. Он, возможно, хотел ответить, ориентируясь на определенную шкалу, например, от —5 до +5, где —5 означает «Нет»; 0 — «Не знаю»; +5 — «Да».

Представим теперь, что априорная вероятность любой реакции равна $P(E)$.

Если есть неопределенность в ответе, то система должна использовать эту вероятность, взяв ее значение $P(E)$, а также значение $P(\text{не}E)$, так как если пользователь не уверен относительно существования конкретного свидетельства, то этого свидетельства, возможно, не существует.

Пусть P — это отклик пользователя. Если $P \geqslant 0$, то

$$P(E) = P(E) + (1 - P(E))\,P/5,$$

если же $P \leqslant 0$, то

$$P(E) = P(E) + P(E)\,P/5.$$

Очевидно, что $P(\text{не}E) = 1 - P(E)$ и при $P = 0$ параметр $P(E)$ остается неизменным. Тогда новое значение

$$P(H:E) = P(H:E)\,P(E) + P(H:\text{не } E)\,P(\text{не } E).$$

Другими словами, вычисляются оба исхода и взвешивается окончательный исход с учетом определенности, которую пользователь выразил как в пользу данного свидетельства, так и против него.

12.11. ФОРМАТ ДАННЫХ

1. Массивы.

DIM R(E, H)

		Гипотеза 1	*Гипотеза* 2... и т. д.
Свидетельство 1	. . .	x	x
Свидетельство 2	. . .	x	x
.. и т. д			

Здесь в массиве R содержится последовательность уравнений для поверхности, т.е. осуществляется их классификация.

2. Операторы данных

DATA Гипотеза 1, Априорная Вероятность Гипотезы 1 [Свидетельство $j_y P(E_j : H)$, $P(E_j : \text{не}H)$]

DATA Гипотеза 2, ... и т. д.

и

DATA j, Свидетельство j.

В операторах данных содержится информация, аналогичная той, что в массиве R каждый оператор DATA очень похож на столбец этого массива.

Преимущества использования массивов заключаются в том, что процесс идет быстрее — данные могут быть потеряны, если вы не позаботитесь об их сохранении на диске.

Преимущества применения операторов DATA состоят в том, что данные труднее потерять и легко модифицировать, хотя это может и замедлить процесс.

В общем случае предпочтительнее сделать еще одно усилие и обеспечить хранение информации в файле на диске.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ОГЛАВЛЕНИЕ

Издание для досуга

Нейлор Крис

КАК ПОСТРОИТЬ СВОЮ ЭКСПЕРТНУЮ СИСТЕМУ

Заведующий редакцией *А. Б. Желдыбин*
Редактор издательства *А. А. Устинов*
Художник обложки *Т. Н. Хромова*
Художественные редакторы: *Т. А. Дворецкова, Т. Н. Хромова*
Технический редактор *Т. Ю. Андреева*
Корректор *Н. А. Смирнова*
ИБ № 3261

Сдано в набор 25.07.90. Подписано в печать 19.02.91. Формат 84×108 1/32. Бумага типографская № 2. Гарнитура литературная. Печать высокая. Усл. печ. л. 15,12. Усл. кр.-отт. 15,33. Уч.-изд. л. 14,44. Тираж 130 000 экз. Заказ № 603. Цена 3 р.

Энергоатомиздат. 113114, Москва, М-114, Шлюзовая наб., 10

Владимирская типография Госкомпечати СССР
600000, г. Владимир, Октябрьский проспект, д. 7